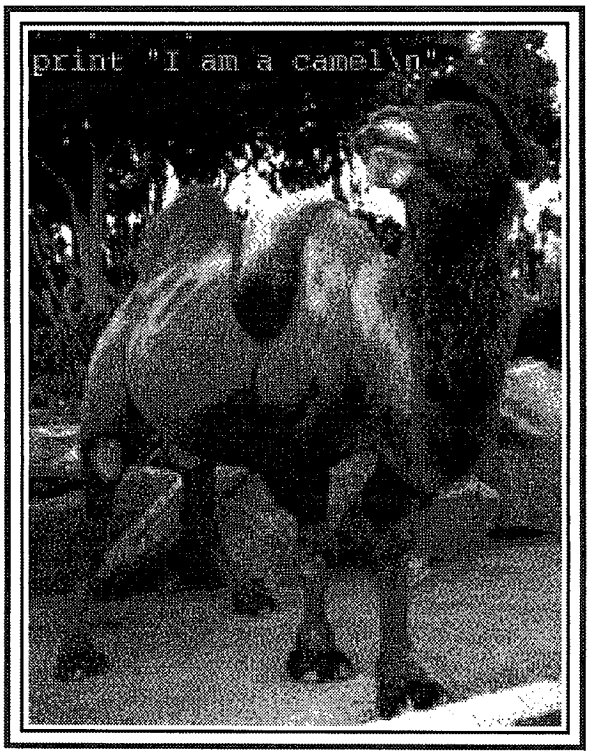
INTRODUCTION TO PERL WITH APPLICATIONS TO CGI PROGRAMMING



Samuel Gasster
UCLA Extension Course
No. 418.104
1997

INSTRUCTORS

Dr. Samuel D. Gasster
Email Address: [redacted]

J. C. Kern
Email Address: kern@aero.org

COPYRIGHT 1995 – 2008 ©
SAMUEL GASSTER
J. C. KERN

NO. OF MEETINGS:

12 classes, 3 hours per class. Computer work outside of class will be required.

PREREQUISITES:

- Introduction to UNIX, or equivalent (students must be comfortable with using the UNIX environment).
- Familiarity with at least one high level computer language (e.g., C, Pascal, Fortran) required.
- Familiarity with HTML and UNIX shell programming helpful.

PURPOSE:

This course in intended to provide the student with a basic knowledge of the Perl programming language. The language will be illustrated with a variety of applications that will be beneficial to systems administrators and Internet CGI programmers. Students will be able to write application scripts for a range of problems as well as CGI applications for the World Wide Web. Many examples will be provided and discussed in detail.

READING LIST:

The required reading materials for this course are:

- Lecture Notes
- *Teach Yourself Perl 5 in 21 Days*, David Till, Sams Publishing (1996).

The recommended reading materials for this course are:

- *CGI Programming on the World Wide Web*, Shishir Gundavaram, O'Reilly & Associates, Inc. (1996).
- *Perl 5 Desktop Reference*, Johan Vromans, O'Reilly & Associates (1996).

Both of the text books are available at the UCLA Student store, local book stores and can be ordered on-line from various sources such as The BookPool (http://www.bookpool.com/).

In addition to these books there are many other books on Perl and CGI Programming, some of which I have listed in the Reading List.

GRADING SYSTEM:

The students grade will be based on homework assignments and attendance. There are 10 homework assignments. Try complete them all on-time as no extensions will be given.

ASSIGNMENTS:

There will be reading assignments each week from ~~my~~ the notes and the course textbook(s). There will be one homework assignment each week, for the first 10 weeks. The homework will be primarily programming assignments so the students need to have access to a computer running Perl.

COMPUTER TIME:

Students will be required to have access to a computer which is running Perl. This can be a home computer, a computer at work, or a computer accessible through the Internet provided by an Internet Access Provider.

A list of access providers can be found on many of the on-line search services and at the following locations:

- http://www.currents.net/resources/netprov/netprov.html
- The List at http://thelist.iworld.com/

COURSE CONTENT:

This is a complete outline of the lectures along with reading assignments and homework. have up.

COURSE WEB SITE

There is a website for this class. The URL is http://www.muon.com/ucla. Material relevant to the course will be posted to this site. This will include a reading list, Perl and CGI links on the WWW and homework solutions.

ACKNOWLEDGEMENTS

Some of the material in these notes was derviced from various sources available on the internet. Whenever possible I have contacted the author(s) for permission and would like to acknowledge their work and contribution.

© 1996-1997 Samuel D. Gasster, ALL RIGHTS RESERVED.

Perl and CGI Programming Reading List

This is a list of books and reference materials on Perl and CGI Programming and related topics. Most of the publishers and many book stores have web pages. You should be able to find out a great deal about most of these books on the WWW. Many of these books can be purchased at local bookstores and of course on the WWW.

Perl and CGI Programming

1. *Teach Yourself Perl 5 in 21 Days*, David Till, Sams Publishing (1996). **[TEXTBOOK for this course]**

2. *Perl 5 Desktop Reference* , Johan Vromans, O'Reilly & Associates, Inc. (1996). **[REFERENCE BOOK for this course]**

3. *Perl by Example* , Ellie Quigley, Prentice Hall PTR, New Jersey (1995).

4. *Learning Perl*, Randal L. Schwartz, O'Reilly & Associates, Inc. (1994).

5. *Programming Perl*, Larry Wall and Randal L. Schwartz, O'Reilly & Associates, Inc. (1994). **[Watch for latest edition due out September 1996]**

6. *CGI Programming on the World Wide Web* , Shishir Gundavaram, O'Reilly & Associates, Inc. (1996). **[TEXTBOOK for this course]**

7. *60 Minute guide to CGI Programming with Perl 5* , Robert Farrell, IDG Books (1996).

8. *CGI Programming in C & Perl*, Thomas Boutell, Addison-Wesley (1996).

9. *Teach Yourself CGI Programming with Perl in a Week* , Eric Herrmann, Sams.net (1996).

10. *Developing CGI Applications with Perl*, John Deep, Wiley (1995).

UNIX

1. *UNIX System V Release 4, An Introduction* , Osborne (199X).

2. *Learning the Unix Operating System* , Grace Todino and John Strang, O'Reilly & Associates, Inc., Sebastopol (1987). *The Owl Book.*

3. *Exploring the UNIX System* , Stephen Kochan, Patrick Wood and Jonathan Leer, Sams, 1995, 4th Edition, 500 pages, ISBN 0-672-30551-8

4. *Introducing the UNIX System* , H. McGilton, McGraw Hill, 1983, ISBN 0-07-045001-3

5. *Introducing the UNIX System V* , R. Morgan, McGraw Hill, 1987, ISBN 0-07-043152-3

6. *J. Ranade UNIX Primer* , Jay Ranade and Saba Zamir, McGraw Hill, 1993, ISBN 0-07-051249-3

7. *Learning the Unix Operating System* , Grace Todino, John Strang, and Jerry Peek, O'Reilly & Associates, 1993, 3rd Edition, 108 pages, ISBN 1-56592-060-0

8. *Learning the vi Editor* , Linda Lamb, O'Reilly & Associates, 1992, 5th Edition, 192 pages, ISBN 0-937175-67-6

9. *UNIX For Dummies* , 2nd Edition, John Levine and Margaret Levine Young, IDG Books, 1995, 384 pages, ISBN 1-56884-905-2

HTML

1. *Creating Cool Web Pages with HTML (w/3.5 disk)* , Dave Taylor, IDG Books, 1996, 2nd Edition, 250 pages, ISBN 1-56884-822-6

2. *Foundations of World Wide Web Programming with HTML and CGI (w/CD)* , Ed Tittel, Mark Gaither, et al., IDG Books, 1995, 650 pages, ISBN 1-56884-703-3

3. *HTML 3.0 Manual of Style* , Larry Aronson, Ziff-Davis, 1995, 197 pages, ISBN 1-56276-352-0

4. *Teach Yourself More Web Publishing with HTML in a Week* , Laura Lemay, Sams, 1995, 451 pages, ISBN 1-57521-005-3

5. *Teach Yourself Web Publishing with HTML 3.0 in a Week* , Laura Lemay, Sams, 1996, 2nd Edition, 400 pages, ISBN 1-57521-064-9

6. *Teach Yourself Web Publishing with HTML in 14 Days (w/CD)* , Laura Lemay, Sams, 1995, 800 pages, ISBN 1-57521-014-2

Internet and World Wide Web

1. *The Whole Internet, User's Guide and Catalog* , Ed Krol, O'Reilly & Associates, Inc., Sebastopol (1994).

2. *Guide to the Internet*, Steven Nelson, Microsoft Press, ISBN 1-55615-822-X.

3. *Internet 101* , A. Glossbrenner, McGraw Hill, 2nd Edition, ISBN 0-07-024207-0.

4. *A Student's Guide to the Internet*, Carol Lea Clark, Prentice Hall (1996). ISBN 0-13-442310-0

[TITLE PAGE | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Acknowledgements

Some of the material in these notes was obtained from various sources available on the Internet and World Wide Web. Whenever possible I have contacted the author(s) for permission and would like to acknowledge their work and contribution. These comments are in no particular order.

Larry Wall
For creating Perl.

Greg Johnson (ccgreg@showme.missouri.edu)
I found his on-line Perl Course http://www.phlab.missouri.edu/perl/perlcourse.html very helpful and he gave me permission to use his material. Thanks!

Jeff Collins
Friend, colleague and fellow Perl Hacker.

Matthias Neeracher (MacPerl)
For porting Perl to my favorite platform!

Don Coscarelli (http://www.phantasm.com/)
A really phantastic guy who provided me the momentum to learn HTML, CGI and Perl. Its been fun from the start!

All Perl Hackers
Who have shared their ideas, bugs and solutions via the Internet and WWW.

Tim Berners-Lee
For inventing the World Wide Web.

Diane, ~~and~~ David and Emily
For putting up with my seemingly endless hours in front of my Mac.

[TITLE PAGE | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

GLOSSARY OF TERMS

This Glossary is intended to help the students understand some of the jargon used in this course. It is not intended to be a complete dictionary of computer science terminology. I welcome suggestions, additions and corrections.

ASCII
American Standard Code for Information Interchange
a standard code used for representing information on computer systems and networks. It uses the printing and non-printing characters that can be generated by the keyboard. Since 7 bits are available to represent each character it is possible to represent a total of 128 different characters.

binary file
a file that contains codes which are not part of the ASCII character set. A binary file can contain any type of information that can be represented by an 8 bit byte - a possible 256 values.

bit
a binary digit of information that can have either of two values, 1 or 0 (zero).

blind carbon copy
a copy of a message which all persons on the `Bcc:` list receive. Recipients cannot see who else has received the message: hence the term "blind".

boot
this term derives from `bootstrap loader', a short program that was read in from cards or paper tape, or toggled in from the front panel switches. This program was always very short (great efforts were expended on making it short in order to minimise the labour and chance of error involved in toggling it in), but was just smart enough to read in a slightly more complex program (usually from a card or paper tape reader), to which it handed control; this program in turn was smart enough to read the application or operating system from a magnetic tape drive or disk drive. Thus, in successive steps, the computer `pulled itself up by its bootstraps' to a useful operating state.

buffer
a temporary area of memory which data can be read from and written to.

byte
a unit of memory or data consisting of 8 bits which is used to represent one character.

daemon
a background process (pronounced "demon") that carries out tasks on behalf of every user. Daemons spend most of their time sleeping until something comes along which requires their help. A UNIX system has a lot of daemons.

environment variable
a variable that defines an aspect of your working environment such as your home directory, editor or printer. These variables are set during the login procedure.

file-creation mode mask
a number consisting of three octal digits which is subtracted from a similar number used by the system to create files and directories.

file system
the physical or logical device that holds a collection of files and directories. This might be a hard disk drive or a partition on a disk drive.

Hash
Something like SPAM!

home directory
the directory you are placed in immediately after you login to the system.

host
a computer system that provides a working environment to many users. So a "UNIX host" is such a system that runs the UNIX operating system.

inode number
a unique number associated with each filename. This number is used to look up an entry in the inode table which gives information on the type, size, and location of the file and the userid of the owner of the file.

link
a connection between an inode number and the filename(s) associated with it.

local host
the host on the network which you are currently using.

Metacharacter
a single character that is used in place of another character or set of characters.

multitasking
the execution of two or more processes at the same time.

parent directory
the directory that is one level above the current directory.

password
a security device consisting of a string of characters known only to the user and the system. The user must supply this string when prompted to gain access to the system.

pathname
a file name given as the sequence of directories that lead to the file. A pathname can be either a full pathname or a relative pathname

Perl
Praticle Extraction and Reporting Language

regular expression
a text pattern consisting of a combination of alphanumreic characters and special characters known as metacharacters.

relative pathname
the name of a directory or file given in relation to your current working directory. See UNIX File System for a more detailed explanation.

remote host
any host on the network which you are not presently using.

root directory

the directory located at the top of the Unix file system. It is represented by the "/" (forward slash) character.

shell

a utility program that enables the user to interact with the UNIX operating system. Commands entered by the user are passed by the shell to the operating system which carries them out. The results are then passed back by the shell and displayed on the user's display.
There are several shells available. The user may select which one they wish to use.

shell prompt

a character at the start of the command line which indicates that the shell is ready to receive your commands. The character is usually a '%' (percent sign) or a $ (dollar sign). It may be different on your system.

standard input

the source of information for a command. This is assumed to be the keyboard unless input is redirected or piped from a file or another command.

standard output

the destination for information from a command. This is assumed to be the terminal display unless ouput is redirected or piped to a file or another command.

syntax

rules governing the way in which characters and words must be put together to form a command that can be recognised and acted upon by the Unix operating system.

system administrator

a person or persons with responsibility for managing the system(s) you are using.

[TITLE PAGE | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

UNIX REVIEW

411 Borel Ave., Suite 100
San Mateo, CA 94402
Tel: 415-358-9500
Fax: 415-358-9739
abinstock@mfi.com

EDITOR-IN-CHIEF
Andrew Binstock
abinstock@mfi.com

MANAGING EDITOR
Robyn P. Freidberg

TECHNICAL EDITOR
Ralph Barker
rbarker@mfi.com

PRODUCTION EDITORS
Sarah A. Piroh
Kevin Lipski

EDITORIAL ASSISTANT
Diane T. Tottori

ART DIRECTOR
David Bergeron

CONTRIBUTING EDITORS
Steven Baker, *Author*
Jon Bentley, *Member of Technical Staff, AT&T Bell Laboratories*
Bruce Eckel, *Lecturer and Trainer*
Kevin Greenfield, *President, Implements Inc.*
William Groenewold, *Author and Lecturer*
Eric F. Johnson, *Author*
Sean Kelly-Bootle, *Author*
Dinah McNutt, *President, Sysadmason Enterprises Inc.*
Richard Morin, *Consultant, Santa Forda Computer Laboratory*
Kevin Reichard, *Author*
Randal L. Schwartz, *General Manager, Stonehenge Consulting Services*
Jim Waldo, *Software Engineer, Sun Microsystems Laboratories Inc.*
Tom Yager, *President, Applied Courseware Concepts*

GROUP PUBLISHER
Katie A. Brennan
kbrennan@mfi.com

EDITORIAL REVIEW BOARD
Dr. James R. Bell, *Director of Corporate Alliances, Hewlett-Packard*
Dr. Stephen R. Bourne, *Director, Network Management Engineering, Cisco Systems*
Dr. Ronald J. Griffith, *Vice President, Product Development, ADP Claims Solution Group*
Gail Hamilton, *General Manager, Telecommunications Platform Division, Hewlett-Packard*
Reed Hastings, *CEO, Pure Software*
Dr. John Mashey, *Director, Systems Technology, Silicon Graphics*
Dr. Marshall Kirk McKusick, *Author and Consultant*
Carol Realini, *President, I. Frank and Associates*
Deborah Scherrer, *Consultant*

DEPARTMENT ILLUSTRATIONS
Jane Hambleton

PHOTOGRAPHY
Carter Dow

HOMEPAGE
http://www.unixreview.com

Printed in the USA

VIEWPOINT

Internet Service Providers?

If you have spent much time on the Internet, you have quickly found that service, as in the expression Internet Service Provider (ISP), is more a goal than a reality. In essence, it is a marketing term that is supposed to connote something more than just access. Right now, however, I would settle for just access.

UNIX Review's parent company, Miller Freeman Inc., uses Netcom as its primary ISP. As you may have read, Netcom was down for 13 hours in late June due to a programming error. Normally, such an occurrence might not raise more than an eyebrow, but because Netcom is one of the largest ISPs (it is one of the few that is publicly traded), the amount of chaos this slip caused was enormous. In fact, so many people called to complain, the phone system went down from the sheer overload of calls.

Netcom immediately claimed this downtime was unprecedented. In actuality, Miller Freeman has experienced several service disconnects traceable to Netcom.

UUNET, another publicly traded ISP and the grand old man of commercial Internet providers, is hardly better. It provides Internet service for *UNIX Review*'s old domain, *ureview.com*. For a month, we have been unable to do our biweekly dial-in to pick up UUCP mail from folks with the old address. Finally, we called UUNET for support. The next day, we were told the UUNET access number had been changed a month before, and in addition, our password had been changed. Had anybody told us or sent us mail? No. Predictably, the invoices for our access managed to arrive unabated.

You undoubtedly have your own irritations: inability to dial in, slow response times, sudden disconnects, lack of replies to support queries, and the like. All this suggests the question: why, in a market so large, so primed for growth, and by all accounts so lucrative, is good service or good access so difficult to obtain?

The problem, oddly enough, is not the ISPs: it's you and me. Having grown with the Internet and adopted its culture, we still expect to have everything we want for free. Or, in the most extreme case, to pay only a token amount (but not without protest: witness the grousing about the $50 per year registration fee for domain names). The fact is, the government no longer subsidizes the Internet. And if you want to be part of the new frontier, you must pay your share.

Right now, the ISP market is so competitive nobody is making money. Netcom is losing money, UUNET just floats by—and these are the big guys. Local ISPs are already falling apart. One market analyst expects that 90% of all ISPs will be out of business or acquired by 1999. This consolidation is happening because profits are small—too small to provide service, much less good service.

If the Internet is to deliver all we think it can, we must learn the new economics quickly. Browsers, good sites, reliable service—all these things will no longer be free. Like phone service or cable TV, it is going to be pay-as-you-go. Get used to it.



INTRODUCTION TO PERL WITH APPLICATIONS TO CGI PROGRAMMING

Course Content

- **LECTURE 1:INTRODUCTION**
  - About this course. Lectures, homework, grading, project.
  - What is CGI Programming?
  - How does it work? Implementation.
  - How to write CGI applications?
  - What is Perl? Version(s). Where can I find Perl? CPAN
  - Perl resources: books, internet, news groups, journals
  - Perl manual, man pages
  - Links to Perl resources.
  - Running Perl: Command line, Perl scripts, Debugging
  - Writing your first Perl script.

- **LECTURE 2: GETTING STARTED WITH PERL**
  - Perl Syntax
  - Printing
  - Perl Variables and Data Types
  - Perl Special Variables
  - Perl Operators
  - Perl Style: writing readable Perl scripts

- **LECTURE 3: CONTROL STRUCTURES AND PERL FUNCTIONS**
  - Control structures in Perl.
  - Built-in Perl Functions: Part I.

- **LECTURE 4: INPUT/OUTPUT, FILE MANIPULATION AND REFERENCES**
  - Filehandles, File Manipulation, File and Directory Information.
  - Report writing.
  - References - pointer the Perl way.

- **LECTURE 5: Perl SUBROUTINES, FUNCTIONS AND LIBRARIES**
  - Subroutine and function syntax
  - Built-in Perl Functions: Part II.
  - Perl libraries

- **LECTURE 6: REGULAR EXPRESSIONS IN Perl.**
  - What is a regular expression?
  - How do we use them?
  - Perl Metacharacters for use in regular expressions.

- **LECTURE 7: PERL PACKAGES AND MODULES**
  - What is a Package and how are they used?
  - How to create your own packages.

- What is a Module and how are they used?
- How to create your own modules.
- What Modules/Packages exist?

- **LECTURE 8: Perl GENERAL TOPICS**
  - Debugging Perl Scripts.
  - UNIX System and Process Calls.
  - Access to processes and System Calls.
  - Sockets and Network access with Perl.
  - Where do I go next? Miscellaneous Topics.

- **LECTURE 9: CGI PROGRAMMING.**
  - CGI Overview (see Lecture 1) (revisited)
  - Server Configuration (overview)
  - Input to CGI
  - Output from the CGI
  - Forms and CGI
  - Debugging and Testing CGI Applications

- **LECTURE 10:HYPERMEDIA, FORMS AND SECURITY**
  - Hypermedia Documents
  - Advanced Form Applications
  - CGI Security Issues
  - Server Side Includes

- **LECTURE 11: CGI APPLICATIONS USING FORMS AND GATEWAYS**
  - Multiple Form Interaction
  - Gateways, Databases, and Search/Index Utilities
  - Gateways to Internet Information Servers

- **LECTURE 12: ADVANCED CGI APPLICATIONS**
  - CGI Design
  - Advanced and Creative Applications

ADDITIONAL COURSE MATERIAL

- READING LIST
- HOMEWORK ASSIGNMENTS
- PERL MANUAL
- PERL AND CGI PROGRAMMING WWW LINKS
- GLOSSARY OF TERMS
- UNIX AND INTERNET REVIEW

[TITLE PAGE]

© 1996-1997 Samuel D. Gasster, ALL RIGHTS RESERVED.

Perl
and
CGI Programming Links

A SET OF LINKS
COMPILIED BY
Samuel Gasster

up-dated: Feb. 1997

Perl

The Perl Language Home Page
http://www.perl.com/perl/

The Perl Journal
http://work.media.mit.edu/tpj/

The Perl Institute
http://www.perl.org/

Randal L. Schwartz's Unix Review article on Perl
http://www.teleport.com/~merlyn/UnixReview/

Searchable Perl Man Pages
http://www.aisd.com/technology/perl/man

Perl News Groups

- comp.lang.perl.misc
- comp.lang.perl.announce
- comp.lang.perl.modules
- comp.lang.perl.tk

Comprehensive Perl Archive Network (CPAN)

Canada
enterprise.ic.gc.ca

California
ftp.digital.com
ftp.cdrom.com

Colorado

ftp.cs.colorado.edu

Florida
ftp.cis.ufl.edu

Illinois
uiarchive.cso.uiuc.edu

Massachusetts
ftp.iguide.com

New York
ftp.rge.com

Oklahoma
ftp.uoknor.edu

Texas
ftp.sedl.org
ftp.metronet.com
ftp.sterling.com

HTML

NCSA--A Beginner's Guide to HTML Home Page
http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html

W3 Consortium HTML Page
http://www.w3.org/pub/WWW/MarkUp/

HTML Reference Manual
by Michael J. Hannah, Sandia National Laboratories <mjhanna@sandia.gov>
http://www.sandia.gov/sci_compute/html_ref.html

The Bare Bones Guide to HTML
by Kevin Werbach
http://werbach.com/barebones/barebone.html

CGI Programming

CGI Security
http://www.cerf.net/~paulp/cgi-security/

How To Set Up and Maintain a WWW Site
http://www-genome.wi.mit.edu/WWW/

Welcome to WebLib!
http://selsvr.stx.com/~weblib/

Teach Yourself Web Publishing: The Examples
http://slack.lne.com/Web/Examples/

Selena Sol's Public Domain CGI Script Archive and Resource Library

http://www.eff.org/~erict/Scripts/

Matt's Script Archive
http://www.worldwidemart.com/scripts/

Download (ftp) the software from *CGI Programming on the World Wide Web* , by Shishir Gundavaram

- CGI_Lint-1.0.pm - The CGI checking program
- common.pl - CGI utilities

BOOKS/JOURNALS/REFERENCES

- The Perl Journal
  http://orwant.www.media.mit.edu/the_perl_journal/
- The Bookpool - Technical Books 20% to 35% Off!
  http://www.bookpool.com/
- O'Reilly Home Page
  http://www.ora.com/

MISCELLANEOUS

- Status of the W3C Reference Library
- The World Wide Web Initiative: The Project
- HTML Style Sheets
- Internet Starter Kit for MacIntosh
- Web Page Basics

SEARCH ENGINES

- AltaVista: Main Page
- Yahoo
- Lycos, Inc. Home Page

[TITLE PAGE | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 1:INTRODUCTION

READING ASSIGNMENT:
TYPERL5 Ch. 1 and 16
CGI Programming on the WWW Ch. 1

HOMEWORK ASSIGNMENT:
Assignment No. 1 (Due Next Week).

Perl MANPAGES:

- Perl Manual
- perlrun
- perldebug

TOPICS

- About This Course.
  - Lectures, homework, grading, course project.
  - Outside computer work.
  - UNIX Review
  - Internet access, telnet and ftp
  - World Wide Web

- What is CGI Programming?
  - How does it work? Implementation.
  - Overview of CGI applications

- What is Perl?
  - Version(s).
  - Where can I find it? CPAN
  - Perl resources: books, internet, news groups, journals
  - Perl manual, man pages
  - Links to Perl resources.

- Running Perl
  - Command line
  - Perl scripts
  - Debugging
  - Writing your first Perl script.
    simple syntax
    comments

About This Course

- Lectures, homework, grading, course project.
- Outside computer work.
- UNIX Review
- Internet access, telnet and ftp
- World Wide Web

What will we learn in this course?

My goals for this class are to:

A. Provide a broad overview to the Perl programming language. This should enable the student to write Perl programs for a variety of applications, including CGI programs.

B. Introduce the student to the concept of CGI programming and how it relates to the WWW.

C. Provide the students with the skills and understanding to develop, test and implement their own CGI applications.

I want this to be a hands on course with lots of practical experience, not a course in computer science theory!

What we will not cover:

A. I do not plan, nor do I have the time to, cover all aspects of Perl. Perl has grown significantly since its first release and there just isn't time in a 12 week course to cover everything.

B. Software Engineering. I assume that every student has done some programming and understands how to develop an algorithm and code it up. I want to focus on teaching the basics of Perl and CGI. Therefore the students should attempt to write simple, clear and easy to read code. Once you've mastered the basics you can write more efficient and obfuscated code.

C. I won't debate Perl versus *Language-X* . If you're into that sort of debate

read the news groups. Enough said!!

D. Perl on your favorite platform (unless maybe its a Mac!). Perl has been ported to so many different computer systems that there is not sufficient time in a 12 week class to cover all the implementation specific issues. Therefore, I have chosen to focus on Unix. See the release notes if you are on a different system.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 2]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

UNIX and INTERNET REVIEW

This document provides a basic introduction to UNIX and descriptions of some essential UNIX commands. The student should already have some familarity with Unix (this stuff should not look completely foreign).

- File System Structure
- Basic Commands
  - cd
  - cp
  - lpr
  - ls
  - mkdir
  - more
  - mv
  - pwd
  - rm
  - rmdir
- More Commands
  - echo
  - finger
  - grep
  - kill
  - lpq
  - lprm
  - man
  - passwd
  - ps
  - whois
- Input and Output Redirection
- Control Sequences
- Editing
  You should be familar with at least one of the editors found on Unix systems.
  If not consult the reading list for a reference book and learn one now. These include:
  - vi
  - pico
  - emacs
- Internet Topics
  - telnet
  - ftp
  - Internet addresses
- More Help

[TITLE PAGE | TABLE OF CONTENTS]

This material was adapted from a review I found at the Baylor College of Medicine Human Genome Center

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

The UNIX File System Structure

The UNIX file system is composed of two types of entities: **files** and **directories**. Every file and every directory has a **pathname** which indicates where that file or directory lives.

Files

Files store information. There are many types of files. Two of the more common types are text files, which you read, and executable files, which you run. If you choose the option "Save As..." at the bottom of this window, a dialog box will appear and prompt you for a "Name for saved document:". If you type in a name and choose "OK," a text file will be created under the name you provided.

Files with names starting with a period (i.e. *.xstartup*) are invisible in normal UNIX use. You can see them by using a variation of the ls command. Just don't delete them. They are there (and invisible) for a reason. Filenames in UNIX generally do not contain embedded spaces. If you end up with a filename with embedded spaces, you can refer to it by enclosing the name in double quotes.

Directories

Directories organize files and other directories, creating the tree structure of UNIX. When you login to your account, you arrive in your personal home directory, the root of your personal section of the UNIX file system tree. Your home directory, in turn, is a subdirectory within other directories in the file system.

Pathnames

Files and directories are specified by pathnames. The pathname of a file indicates its location in the filesystem. Two files can have the same name as long as their pathnames are different. If you choose the option "Save As..." at the bottom of this window, the dialog box asking you for a filename will automatically fill in the pathname of your home directory. An example of a pathname for a user's home directory is /usr/jpower. The pathname for that user's mail directory might then be /usr/jpower/Mail. When you tell UNIX to look for a text file, it checks for that file in the current directory. If the file is in another directory, you must supply the pathname to the file.

[Navigating the File System | UNIX Review]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Basic UNIX Commands

This document provides descriptions of selected UNIX commands.

cd

The command **cd *my_dir*** changes your position to the directory specified, in this case *my_dir*. The command **cd** without an argument moves you to your home directory.

cp

The command **cp *first_file copy_file*** copies the contents of *first_file* into the file *copy_file*. To indicate that the new file is to have the same name as *first_file*, use a period (.) instead of of providing a name for the second file. (In this case, the files must be in separate directories, as two files cannot have the same name if they are in the same directory.) For example: **cp** *some_directory/my_file* . copies *my_file*, located in *some_directory*, and creates a file named *my_file* in the current working directory.

lpr

The command **lpr** *print_file* sends *print_file* to the default printer (see **echo**). The form **lpr -P*other_printer print_file*** sends *print_file* to *other_printer*.

ls

The command **ls** lists the files in the current directory. The form **ls -F** shows the difference between directories and ordinary files. The form **ls -a** lists all files, even those that are normally invisible in UNIX (files whose names start with a period, i.e. *.xstartup*).

mkdir

The command **mkdir *new_dir*** creates a new subdirectory named *new_dir* in the current directory.

more

The command **more *my_file*** displays the text of *my_file* one page at a time. To see the next page, hit the space bar; to see the previous page, type *b* ; to quit paging the file, type *q* .

mv

The command **mv *file_name dir_name*** moves the file *file_name* from the current directory into the directory *dir_name*, where *dir_name* is a subdirectory of the current directory. The form **mv *old_file new_file*** renames *old_file* and calls it *new_file*.

pwd

The command **pwd** prints the pathname of the current, or working, directory.

rm

The command **rm *my_file*** deletes *my_file*. The form **rm -i *my_file*** asks if you really want to remove the file *my_file* before it proceeds.

rmdir

The command **rmdir *my_dir*** removes the directory *my_dir*. The directory must be empty before it can be deleted.

note: If you get an error message that a directory is not empty when it appears to be, check for invisible files (see **ls**).

[more UNIX commands | UNIX Review]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

More UNIX Commands

This document provides descriptions of selected UNIX commands.

echo

The command **echo *MY_VARIABLE*** displays the current value of environment variables. Two variables that might interest you at some point are *$PRINTER* and *$DISPLAY*.

finger

The command **finger *user_name*** gives you information on the user specified by *user_name*. The argument *user_name* can be the name of someone locally (i.e. *jpower*) or someone at a remote location (i.e. *jpower@bcm.tmc.edu*).

grep

The command **grep *string filename*** searches *filename* for *string*. It outputs every line which contains *string*. The form **grep -v *string filename*** outputs every line which does not contain *string*. The argument *string* is read by **grep** as a regular expression.

kill

The command **kill *my_process*** sends a terminate signal to the process specified by the process id (PID) *my_process*. In cases where the terminate signal does not work, the command **kill -9 *my_process*** sends a kill signal to the process. For info on getting the PID for a process, see **ps**.

lpq

The command **lpq** outputs the current queue for the default printer (see **echo**). The form **lpq -P*other_printer*** outputs the current queue for *other_printer*.

lprm

The command **lprm *job_number*** removes *job_number* from the queue for the default printer (see **echo**). To remove a job, you must be the owner of that job. To find *job_number* to send to **lprm**, use the command **lpq**. The form **lprm -P*other_printer job_number*** removes *job_number* from the print queue of *other_printer*.

man

The command **man *command*** displays the UNIX manual page for *command*. The manual pages describe usage and options for every UNIX command.

passwd

The command **passwd** allows you to change the password you use to login to the computer. The process is self-explanatory once you type the command.

ps

The command **ps** lists the processes running on your machine. The form **ps gux** lists only your processes. The form **ps aux** lists all processes running on your machine. The second column of the listing, the PID column, provides the information required by the **kill** command.

whois

The command **whois *lookup_string*** performs a directory lookup on persons at your home institution, where *lookup_string* is all or part of someone's first name, last name, or phone number.

[basic UNIX commands | UNIX Review]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Regular Expressions

This document provides a brief description of the syntax of regular expressions. Regular expressions are used in Unix shell programming as well as in Perl. Perl regular expressions and the related operations are much more powerful.

- A list of characters enclosed by [and] matches any character in the list.
- A list of characters enclosed by [and] with ^ as the first character matches any character not in the list.
- A range of characters can be expressed by separating the first and last characters in the range by a hyphen and enclosing them by [and].
- A period (.) matches any single character.
- A regular expression matching a single character (i.e. not a range) can be followed by a repetition operator:
    - **?** indicates that the preceding item is optional, matched once at most.
    - ***** indicates that the preceding item will be matched zero or more times.
    - **+** indicates that the preceding item will be matched one or more times.
    - **{n}** indicates that the preceding item is matched exactly **n** times.
    - **{n,}** indicates that the preceding item is matched **n** or more times.
    - **{,m}** indicates that the preceding item is optional and is matched **m** times at most.
    - **{n,m}** indicates that the preceding item is matched at least **n** times, but no more than **m** times.

[TITLE PAGE | TABLE OF CONTENTS | BACK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Input and Output Redirection

This document provides basic descriptions of redirection and pipe commands.

input

By convention, a UNIX command reads input from *standard input* (the keyboard). To get a command to read from a file instead, you need the command, the filename, and the character '<' : **my_command < *my_input***. Think of the '<' as an arrow pointing in the direction the data is flowing, from the file to the command.

output

The output of a UNIX command is sent to *standard output* (the screen) by convention. To get a command to send the output into a file instead, you need the command, the filename, and the character '>' : **my_command > *my_output***. The arrow analogy holds true in this direction as well, with the data flowing from the command to the file.

To append the output of a command to a file without erasing its previous contents, use the notation : **my_command >> *my_output***.

pipes

If you have a series of commands in which the output of one command is the input of the next, you can pipe the output through without saving it in temporary files: **first_command | next_command**. For example, if you wanted to print out a sorted version of a file that contained a list of names and phone numbers, you could use a pipe (as well as input redirection): **sort < *my_phone_list* | lpr**.

UNIX Review

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

UNIX Control Sequences

This document provides descriptions of selected UNIX control sequences.

The carat symbol (^) represents the control key.

^C: Abort the current program or process.

^D: Exit xterm, ftp session, telnet session, gcg session, etc.

^Z: Suspend execution of the current program or process. To restart execution in the foreground, type **fg**. To restart execution in the background, type **bg**.

[UNIX Review | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

telnet

Telnet is an indispensible tool used for logging in to other computers on the Internet. Telnet can be used to get from one type of computer (i.e. Macintosh) to another (i.e. UNIX machine). Many public services provide access via telnet. To use telnet, you must have the address for your destination computer. The syntax is then: **telnet *address*** . (Click here for more on Internet addresses.) Once you are connected to a computer via telnet, you have access to everything that you would if you were directly logged into that machine.

For example, to access the UCLA sun lab computers:

```
host> telnet sun.sunlab.ph.ucla.edu

telnet sun.sunlab.ph.ucla.edu
Trying 128.97.116.4...
Connected to sun.sunlab.ph.ucla.edu.
Escape character is '^]'.


UNIX(r) System V Release 4.0 (sun)

login: perl05
Password:

sun>
```

You type in your user name and password (not echoed).

command mode

The telnet escape character (**ctrl-]**) gets you into telnet command, indicated by the presence of the telnet prompt (telnet>). Here are some useful commands:

- **close** - close current connection
- **open *address*** - open new connection
- **quit** - exit telnet
- **?** - gives list of commands

[telnet | ftp | Unix Review | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

ftp

Ftp stands for file transfer protocol. Ftp allows you to obtain information, usually documents or programs, from remote computers.

Ftp is especially useful because it doesn't care what type of computers are involved. The *client* (where the ftp process is started) and the *server* (the destination of the ftp process) can be completely different kinds of computers.

The command to start ftp is **ftp *address*** . This connects you to the destination machine. (Click here for more on Internet computer addresses.) You will then be prompted for your login and password on that machine.

anonymous ftp

Many computers are set up as anonymous ftp servers. This means that anyone can login and obtain whatever has been put there without having an account and password on that machine; the login is **anonymous** and the password is the guest's email address.

For example, to connect to the Genome Center's ftp server:

```
imgen> ftp gc.bcm.tmc.edu

Connected to KIWI.IMGEN.BCM.TMC.EDU.
220 kiwi FTP server ...
Name (gc.bcm.tmc.edu:jpower): anonymous
331 Guest login ok, send ident as password.
Password: jpower@bcm.tmc.edu
230 Guest login ok, access restrictions apply.
```

commands

The following commands will get you started with ftp:

- **binary** - starts binary mode for file transfer
- **cd** - changes directories on remote machine (syntax is that of cd)
- **lcd** - changes directories on local machine (syntax is that of cd)
- **ls** - lists files on remote machine (syntax is that of ls)
- **prompt** - toggles prompt mode (questions vs. no questions)
- **pwd** - gives current location on remote machine (works like local pwd)

- **quit** - closes connection and quits out of ftp
- **get**
- **put**
- **mget**
- **mput**

be careful...

- If you transfer a binary file in ascii mode it will be completely useless to you.
- Some files are very big and take a long time to transfer, especially when the network is loaded during business hours. This is especially true if you are downloading to your computer via a modem.
- If you **get** a file onto your machine as *cool_file* , it will overwrite any file in the destination directory named *cool_file* without warning you.
- Many large files are stored in compressed format. You will need software to uncompress the downloaded files to obtain the files you want. Also some files are encoded into an ASCII format (from binary) making them easier for email and other programs to handle these files. This often results in a larger file and will also require special software to de-code these files. Software to handle compressed and encoded files is available on the internet.

[telnet | ftp | Unix Review | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Internet Addresses and Domain Names

Every computer on the Internet has an **Internet Protocol (IP)** address, which consists of a set of 4 numbers, each less than 256 and separated by a period. For example:

```
192.112.36.5
205.163.122.34
```

This number system is unique on the internet and works very well, but its not easy for humans to remember all these numbers. To get around this problem the **DOMAIN NAME SYSTEM (DNS)** was created. The organization that assigns Internet addresses, the InterNIC, also assigns a domain name corresponding to each IP address. The domain name is a series of tags arranged in order of decreasing specificity.

For example, the UCLA computer with IP address **128.97.116.4** is assigned the domain name **sun.sunlab.ph.ucla.edu**. Let's disect this domain name, starting at the left.

sun	the name of the computer attached to the sunlab network
sunlab	a subnet or local network on the public health network at UCLA
ph	the school of public health network (subnet) at UCLA
UCLA	the name of the UCLA network
edu	the type of institution educational

The domain name system addresses will end in one of the following codes:

- *edu* - educational institution
- *com* - commercial institution
- *gov* - government institution
- *mil* - military institution
- *net* - network service
- *org* - other organization

Some Internet addresses end in an additional tag indicating the country (i.e. **ca** for Canada and **au** for Australia).

For more information try the InterNIC website or read The Whole Internet User's Guide & Catalog.

[telnet | ftp | Unix Review | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 1: WHAT IS CGI PROGRAMMING?

OVERVIEW OF CGI PROGRAMMING

WHAT IS CGI AND CGI PROGRAMMING?

CGI

COMMON GATEWAY INTERFACE

- CGI is a standard programming interface created by the NCSA (National Center for Supercomputing Applications). Current version is CGI/1.1.
- CGI allows you to have interactive web pages
- CGI can be easy ---for some applications
- CGI is part of the web server that allows communication with other programs that run on the server

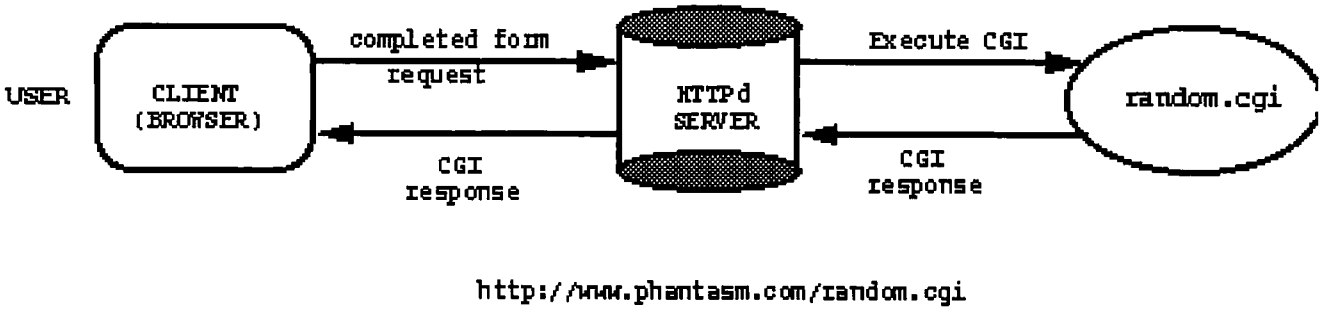


Figure 1. Diagram of how Client-Server-CGI works.

1. Client (browser) program makes a connection to the server over the internet
2. Client sends an HTTP request message to the server
3. Server processes the request message, optionally using CGI programs
4. Server sends back HTTP response message
5. Server closes the connection

- CGI requires special types of input and has specific rules about program output -- this is primarily to allow the server and browser to handle the information correctly
- Client-Server communicate using the HyperText Transfer Protocol (HTTP)

- The CGI specification provides a uniform method of running applications and parsing data between the client and server

---

CGI APPLICATIONS OVERVIEW AND EXAMPLES

There are several different classes of applications that lend themselves to CGI. Some of these are:

- **FORMS**
  Surveys, order forms for merchandise

- **IMAGEMAPS**
  Images which can provide hyperlinks to other documents depending on which part of the image you click

- **GATEWAYS**
  Access to databases and other information servers.

- **VIRTUAL DOCUMENTS**
  Documents which are dependent on a variety of factors and are not necessarily the same for every user or a given user who visits a site at different times.

These are not necessarily disjoint applications. For example, consider the on-line book store. One would like to browse or search a list of books available for ordering. This requires a gateway service to the book store inventory data base. Once the user decides on the books she wants to order, she can then fill out the on-line order form (or in some cases add the book(s) to their on-line shopping cart). The format of the order form may vary depending on information supplied by the user regarding such issues as shipping/handling and sales tax (location dependent).

EXAMPLES

GENERIC FORM EXAMPLE

This is a generic form for use with cgi-lib.

What's your name? [ ]

How old are you? [ ]

What kind of computer do you have?

I like to drink coke. ☒

submit

GENERIC FORM RESPONSE

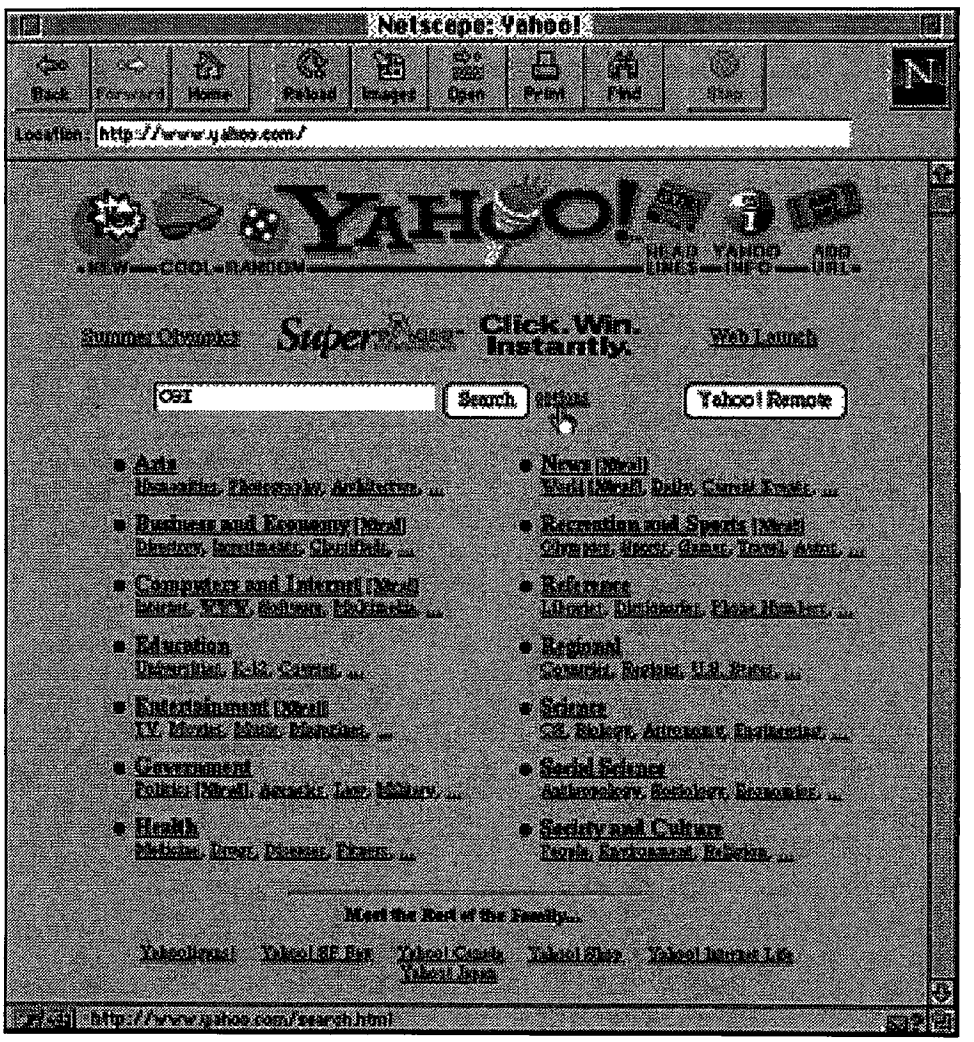
FIELD	INPUT
age	13
coke	on
computer	Mac
name	Bob

When the server receives a request to execute a CGI program, it parses the input data and sets several environment variable that are then available to the CGI program. We'll cover this in much more detail in later lectures. For now here is a sample list of environment variables.

```
CONTENT_LENGTH=36
CONTENT_TYPE=application/x-www-form-urlencoded
DOCUMENT_ROOT=/home/httpd/html
GATEWAY_INTERFACE=CGI/1.1
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
HTTP_ACCEPT_LANGUAGE=en-US, en, en-GB
HTTP_CONNECTION=Keep-Alive
HTTP_HOST=icg.stwing.upenn.edu
HTTP_REFERER=http://www.seas.upenn.edu/~mengwong/forms/form.html
HTTP_USER_AGENT=Mozilla/2.0 (Macintosh; I; 68K)
PATH=/sbin:/usr/sbin:/bin:/usr/bin
QUERY_STRING=
REMOTE_ADDR=130.221.194.141
REMOTE_HOST=at3-2.aero.org
REMOTE_IDENT=unknown
REQUEST_METHOD=POST
SCRIPT_FILENAME=/home/httpd/cgi-bin/generic.pl
SCRIPT_NAME=/cgi-bin/generic.pl
SERVER_ADMIN=webmaster@icg.resnet.upenn.edu
SERVER_NAME=icg.resnet.upenn.edu
SERVER_PORT=80
```

```
SERVER_PROTOCOL=HTTP/1.0
SERVER_SOFTWARE=Apache/1.0.5
```

CGI EXAMPLE
YAHOO INTERFACE



Note the search form that allows users
to interact with the Yahoo database.

INTERNAL WORKINGS OF CGI

- Client requests a special document that the server has been configured to recognize.
- Server has been configured to look in special directories (`/cgi-bin`) for these files (`.cgi`)

- Server does not return file contents, as it would for an HTML file, but instead executes the program

- Sample server request from client (browser):

```
1       GET /cgi-bin/welcome.pl HTTP/1.0
2       Accept: www/source
3       Accept: text/html
4       Accept: image/gif
5       User-Agent: Mozilla/2.0 (Macintosh; I; 68K)
6       From:
```

Line 1: The GET request (the request type) identifies the file to access (/cgi-bin/welcome.pl) and the communications protocol. The server must be configured to recognize files in the /cgi-bin directory as CGI scripts (in this case with extension .pl - for Perl, what else!).

Line 2-4: The Accept key identifies the data formats that the client will accept.

Line 5: The User-Agent key identifies the client (browser) that sent the request, which was a Netscape 2.0 browser running on a Macintosh computer.

Line 6: The From key identifies the user (that's you!). I have left this blank because I do not believe that most browsers currently send this information. Many users were not happy about their email addresses being sent to every site they visited! Kind of like *Caller ID*. (See RFC1413: http://sunsite.auc.dk/RFC/rfc/rfc1413.html)

There is more information available, which we shall discuss in later lectures.

- How do CGI programs get their input?
  - This is platform dependent
  - On UNIX systems data is passed using **STDIN** and shell **ENVIRONMENT** variables

- The CGI program is then executed and returns its output to the server using **STDOUT** (on UNIX systems, again this is system dependent)

- CGI output data stream consists of two parts:

1. **HTTP header.**
   - describes the format of the returned data (MIME type)
   - header ends with a blank line

2. **Body of data stream.**
   - The data must conform to the type described in the header. The server just forwards the data to the client (browser) after wrapping it in the HTTP communications protocol.

SAMPLE DOCUMENT WITH HTTP HEADER

```
HTTP/1.0 200 OK
Date: Thursday, 22-February-96 08:28:00 GMT
Server: Apache-SSL-US/1.0.5+1.0+1.1.1
MIME-version: 1.0
Content-type: text/html
Content-length: 3000

        <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
        <HTML>
        <HEAD>

        <TITLE>Welcome to Sam's Perl/CGI Course</TITLE>

        </HEAD>
        <BODY>
        <H1 ALIGN=CENTER>Course Outline</H1>

        ....body goes here...

        </BODY>
        </HTML>
```

- Servers will accept both full and partial HTTP headers. The minimum requirement is to send the `Content-type` followed by a blank line.

- **What is MIME?**

MIME

Multipurpose Internet Mail Extensions

MIME is the standard for how to send multipart, multimedia, and binary data using the world-wide Internet email system. Typical uses of MIME include sending images, audio, word processing documents, programs, or even plain text files when it is important that the mail system does not modify any part of the file. MIME also allows for labelling message parts so that a recipient (or mail program) may determine what to do with them.

- `Content-type` is specified using MIME

MEDIA TYPES

The Internet Working Group's Request for Comments #1521 paper, specifies that content types, content subtypes, character sets, access types, and conversion values for MIME mail will be assigned and listed by the IANA (Internet Assigned Numbers Authority (iana@isi.edu)).

RUNNING CGI SCRIPTS

- The server must be configured to allow users to execute CGI programs
- Each server vendor (NCSA, CERN, Netscape, Microsoft, Apache, etc.) defines a **server configuration file**: *srm.conf* for NCSA server
- The server needs to know, amoung other things:
  - What directories are allowed for CGI scripts

```
ScriptAlias     /cgi-bin/       /usr/local/etc/httpd/cgi-bin/
ScriptAlias     /my-cgi-bin/    /usr/local/etc/httpd/my-cgi-bin/
```

  - What file types are allowed to be executed:

```
AddType         application/x-httpd-cgi         .pl .sh .cgi
```

- Typically this information is preconfigured by your IAP.

CGI PROGRAMMING - FIRST LOOK

- Now that you have some basic idea about how CGI works what programming language should you use to create your CGI programs?
- You can in principal use ANY programming language!
- Things to consider when choosing a programming language for your CGI applications:
  - Ease of text manipulation
  - Interface to other software libraries and utilities
  - Access to Environment variables

- Security
- What system you are on; not all languages are available under all operating systems
- Learning curve
- Availability of support: technical, existing scripts, ..., etc.

- Some of the languages used for CGI applications:
  - AppleScript (Macintosh only)
  - C/C++ (Unix, Windows, Macintosh)
  - Shell Programming (Unix only)
  - Perl (Unix, Windows, Macintosh, ..., etc.)
  - Tcl (Unix)
  - Visual Basic (Windows only)
  - JAVA (Unix, Windows, Macintosh, Braun)
  - Others?

There are different pros and cons with using each of these languages. It is not my intent to debate which language is *best*, especially since this is a course on Perl. Let's discuss some of Perl's advantages regarding the points made above.

- Perl is highly portable, available on many platforms and is FREE!
- Perl, as we shall see, has one of the best string manipulation capabilities, as we'll as the ability to handle binary data and interface with external databases.
- Perl, as we shall see (I hope), is comparatively easy to learn, yet powerful!
- Perl allows you to access the system and operating system environment (from a security standpoint this has its pluses and minus as we'll discuss later).
- Perl is highly extensible, which makes it easy to build tools for performing various CGI and related functions (we'll also learn about this when we

discuss Perl Modules).

- May be best of all, there are a large number of developers who have written, and are writing, CGI scripts in Perl. So there are a lot of resources available.

CGI Script Example in Perl

```
#!/usr/local/bin/perl
#
#------------------------------------------------
#       file: random.cgi
#
# Random Image Displayer
# Version 1.0 Created by: Matt Wright
# Created On: 7/1/95   Last Modified: 7/1/95
# contact:
#       mattw@alpha.pr1.k12.co.us
#   http://alpha.pr1.k12.co.us/~mattw/home.html
#
#   modified by Sam Gasster 11/95 for
#   the Phantasm Web site
#
#------------------------------------------------
#       Define Necessary Variables

$max = 4;
$server = "http://www.phantasm.com";
#
$dir = "/graphics/phantasm";
$file = "MainImg_";
$ext = "jpg";

############## End Options ##################

# Choose Random Number
srand;
$num1 = rand($max);
$num = int $num1;

# Print URL to Random Image
print "Location: $server/$dir/$file$num.$ext\n\n";

exit;
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 1]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 1: INTRODUCTION TO PERL

What is Perl? Why Use Perl?

Practical Extraction and Reporting Language.

- Perl was created by **Larry Wall** <lwall@netlabs.com>, with the help of oodles of other folks.
- Interpreted/compiled language, intended to be practical (easy to use, efficient, complete)
- Optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information
- Useful for Unix System Administration tasks and CGI programming
- Perl combines some of the features of C, **sed**, **awk**, and **sh**, so people familiar with those languages should have little difficulty with it. (Language historians will also note some vestiges of **csh**, Pascal, and even BASIC-PLUS.)
- Expression syntax corresponds quite closely to C expression syntax
- No arbitrary memory limits
    - Unlike most Unix utilities, Perl does not arbitrarily limit the size of your data--if you've got the memory, Perl can slurp in your whole file as a single string.
    - Recursion is of unlimited depth. And the hash tables used by associative arrays grow as necessary to prevent degraded performance.
- Perl uses sophisticated pattern matching techniques to scan large amounts of data very quickly.
- Perl can also deal with binary data, and can make dbm files look like associative arrays (where dbm is available). There are also available many modules for handling other types of database files.
- Why are we learning Perl in this course? Perl is presently one of the most

frequently used languages on the W3 for writing CGI applications.

VERSIONS

The latest version of Perl is 5.003. Perl was orignially developed in the Unix environment, but has since be ported to a large number of platforms.

- Unix (various flavors, Solaris, BSD, Liniux, etc.)
- Macintosh OS (MacPerl)
- Windows (3.X, W95 and NT)
- AMIGA
- AOS
- ATARI
- LYNXOS
- MPE
- MSDOS
- MVS
- NETWARE
- NT
- OS2
- QNX
- RISCOS
- VMS
- XENIX

In this course we will only discuss the use of Perl in the Unix environment such as one might find on a workstation. Those of you who would like to run Perl on a different platform using a different OS should consult the CPAN and the release notes for the particular port.

HOW DO I GET PERL?

• Perl is free!

- Perl can be downloaded from one of the Comprehensive Perl Archive Network (CPAN) sites. A list of CPAN sites can be found on The Perl Language Home Page. For convience I have listed those in North America here as well.

- The master CPAN site is at FUNET, the Finnish University NETwork (Finland, Europe). Select the site nearest to you from the list below, **Registered CPAN sites**, to get the best response time and bandwidth.

- CPAN REAMDE FILE

Registered CPAN sites
North America

Canada
enterprise.ic.gc.ca

California
ftp.digital.com
ftp.cdrom.com

Colorado
ftp.cs.colorado.edu

Florida
ftp.cis.ufl.edu

Illinois
uiarchive.cso.uiuc.edu

Massachusetts
ftp.iguide.com

New York
ftp.rge.com

Oklahoma
ftp.uoknor.edu

Texas
ftp.sedl.org
ftp.metronet.com
ftp.sterling.com

PERL RESOURCES

There is a tremendous amount of information available about Perl in the form of books, notes, tutorials and sample scripts. Much of this information is available on the Internet.

- **Books and Journals**
  - Consult the Reading List
  - Search on-line and always try The Perl Language Home Page
  - Many computer journals have articles on Perl and there is now The Perl Journal
- **Internet and WWW**
  - There is a wealth of Perl information accessible via the Internet and WWW.
  - Consult my list of Perl Links (see my notes and http://www.muon.com/)
  - Perl Home Page: http://www.perl.com/perl/
- **Perl News Groups**
  - comp.lang.perl.misc
  - comp.lang.perl.announce
  - comp.lang.perl.modules
  - comp.lang.perl.tk
- **Other News Groups that might be of interest**
  - comp.infosystems.www.authoring.cgi
  - comp.infosystems.www.authoring.html
  - comp.infosystems.www.authoring.images
  - comp.infosystems.www.authoring.misc
- **Mailing Lists**
  There are several mailing lists devoted to Perl and related matters. For a list and short description check: http://www.perl.com/perl/info/mailing-lists.html
  You can also get information about the mailing list archives at this site.
- **Perl Man Pages**
  This is the online documentation that comes with the Perl distribution. It can be accessed on Unix sysems using the command

```
man perl
```

There are also postscript and html versions available, check The Perl Language Home Page.

- **Perl FAQs**
  There are several lists Perl Frequently Asked Questions available on the Internet. Check the links mentioned above.

- **Perl Consultants**
  More and more Perl experts are showing up everyday. To find one, try advertising in the `comp.lang.perl.misc` news group or checking the online search engines. Also check The Perl Journal and The Perl Language Home Page

---

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 1]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 1: RUNNING PERL

- **Is Perl on my system?**

  If you are on a Unix system try the following commands:

```
$whereis perl
perl: /usr/bin/perl /usr/local/bin/perl4.036 /usr/local/bin/pe:
/usr/local/bin/perl5.001m

$which perl
/bin/perl
```

- **What version do I have?**

  - The latest version is 5.003
  - If you installed it yourself check the release notes
  - If it is already on your system (this is the case with almost all IAPs) then try:

```
$perl -v

This is perl, version 5.002

Copyright 1987-1996, Larry Wall

Perl may be copied only under the terms of either the
Artistic License or the GNU General Public License,
which may be found in the Perl 5.0 source kit.
```

- **Configuring your Environment:**
  Unix Environment Variables for Perl

HOME
Used if chdir has no argument.

LOGDIR
Used if chdir has no argument and HOME is not set.

PATH
Used in executing subprocesses, and in finding the script if `-S` is used.

PERL5LIB

A colon-separated list of directories in which to look for Perl library files before looking in the standard library and the current directory. If PERL5LIB is not defined, PERLLIB is used.

For example you could put the following line into your `.cshrc` file on a Unix system:

```
setenv PERL5LIB /exp/home/gasster/perllib
```

When running taint checks (because the script was running setuid or setgid, or the `-T` switch was used), neither variable is used. The script should instead say

```
use lib "/my/directory";
```

PERL5DB

The command used to get the debugger code. If unset, uses

```
BEGIN { require 'perl5db.pl' }
```

PERLLIB

A colon-separated list of directories in which to look for Perl library files before looking in the standard library and the current directory. If PERL5LIB is defined, PERLLIB is not used.

Apart from these, Perl uses no other environment variables, except to make them available to the script being executed, and to child processes. However, scripts running setuid would do well to execute the following lines before doing anything else, just to keep people honest:

```
$ENV{'PATH'} = '/bin:/usr/bin';    # or whatever you need
$ENV{'SHELL'} = '/bin/sh' if defined $ENV{'SHELL'};
$ENV{'IFS'} = ''          if defined $ENV{'IFS'};
```

A *sample* listing of environment variables:

```
$printenv
HOME=/exp/home/gasster
PATH=/bin:/usr/bin:/usr/ucb:/usr/sbin:/usr/local/bin:/etc:.
```

```
LOGNAME=gasster
TERM=vt220
TZ=US/Pacific
SHELL=/usr/local/bin/tcsh
MAIL=/var/mail/gasster
USER=gasster
HOST=polaris
HOSTTYPE=sun4
MANPATH=/usr/man:/usr/local/man
PERL5LIB=/exp/home/gasster/perllib
```

- **How do I run Perl?**
  - From the Command line
  - From a script or program file

Perl on the Command Line

One of the very handy aspects of Perl is that it can be run from either the **command line** or as an **executable script**. The command line execution is extremely useful when performing various functions from the Unix shell environment, and I'll show some examples below.

Syntax

```
perl [switches] filename args
```

Upon startup, Perl looks for your script in one of the following places:

1. Specified line by line via `-e` switches on the command line.
2. Contained in the file specified by the first filename on the command line. (Note that systems supporting the `#!` notation invoke interpreters this way.)
3. Passed in implicitly via standard input. This only works if there are no filename arguments--to pass arguments to a STDIN script you must explicitly specify a - for the script name.

Perl COMMAND LINE SWITCHES

There are several command line switches available in Perl. At this stage of the course I'm only mentioning a few useful ones. Later in the course I'll mention the others. In the meantime you should consult the Perl manpage, perlrun, for more information.

-v

prints the version and patchlevel of your Perl executable.

-c

causes Perl to check the syntax of the script and then exit without executing it. Actually, it *will* execute `BEGIN`, `END`, and `use` blocks, since these are considered as occurring outside the execution of your program.

-d

runs the script under the Perl debugger

-e *commandline*

may be used to enter one line of script. If -e is given, Perl will not look for a script filename in the argument list. Multiple -e commands may be given to build up a multi-line script. Make sure to use semicolons where you would in a normal program.

-F *regexp*

specifies a regular expression to split on if -a is also in effect. If regexp has // around it, the slashes will be ignored.

-i *extension*

specifies that files processed by the <> construct are to be edited in-place. It does this by renaming the input file, opening the output file by the original name, and selecting that output file as the default for print() statements. The extension, if supplied, is added to the name of the old file to make a backup copy. If no extension is supplied, no backup is made.

-n

causes Perl to assume the following loop around your script, which makes it iterate over filename arguments somewhat like **sed -n** or **awk**:

```
while (<>) {
    ...             # your script goes here
}
```

Note that the lines are not printed by default.

-p

causes Perl to assume the following loop around your script, which makes it iterate over filename arguments somewhat like **sed**:

```
while (<>) {
    ...                 # your script goes here
} continue {
    print;
}
```

Note that the lines are printed automatically. To suppress printing use the -n switch. A -p overrides a -n switch.

BEGIN and END blocks may be used to capture control before or after the implicit loop, just as in awk.

-w

prints warnings about identifiers that are mentioned only once, and scalar variables that are used before being set. Also warns about redefined subroutines, and references to undefined filehandles or filehandles opened readonly that you are attempting to write on. Also warns you if you use values as a number that doesn't look like numbers, using an array as though it were a scalar, if your subroutines recurse more than 100 deep, and innumerable other things.

EXAMPLES

```
#
#       a Perlish way to cat a file
#

perl -ne 'print' filename
...
....listing of the contents of filename
...

#
# print out todays date
#

$perl -e '$date=`date`;print "It is now $date\n";'
It is now Sat Aug 24 15:28:07 PDT 1996


#
#       listing of user names from the /etc/passwd file
#

$perl -ne '@data=split(/:/);print "$data[0]\n"' /etc/passwd
root
daemon
bin
```

```
sys
adm
lp
smtp
uucp
nuucp
listen
nobody
noaccess
sandy
news
relliot
dpennells
ineuman
hwong
vibrant
jwill


#
#       pipe who into perl with a match operator
#       a grepish kind of thing
#
$who |perl -ne 'print if /gasster/;'
gasster    pts/22       Aug 24 14:43    (pm2-24.pacificnet.net)
```

Perl Scripts

Your First Perl Script

I will use the terms script and program interchangablly.

Did you really think you could take this class with out having to see the Perl version of the *Hello World* program! Well I won't disappoint you. Here it is:

Hello World Script

In your favorite editor (vi, emacs, pico, ...) enter these lines (ignore the line numbers!!):

```
1       #!/usr/local/bin/perl -w
2       #
3       #       My first Perl Script
4       #       file: hw.pl
5       #       s. gasster      7/96
6       #
7       print ("Hello World\n");
```

Save the file as `hw.pl`.

After you have entered the above script using a text editor, and saved the result to the file `hw.pl`, you must tell the shell that it is an executable file. To do this you must set the permissions:

```
$chmod +x hw.pl
```

is one way to do this. Once you have done this you can check that the file is execuable using:

```
$ls -l hw.pl
-rwxrwxr-x  1 gasster         10 Aug 14 17:00 hw.pl*
```

Output

```
$hw.pl
Hello World
```

Discussion

Things to Note:

- Perl comments start with #
- Perl statements end with ;

Line 1 is a comment line that the shell will use to determine what type of script this file contains. In this case it is told that the Perl interpreter can be found in `/usr/local/bin`. I have also told the shell to run Perl with the `-w` switch on, which is helpful when debugging. You can also include other switches on this line as well, but be aware that some operating systems silently chop off kernel interpretation of the `#!` line after 32 characters, some switches may be passed in on the command line, and some may not; you could even get a - without its letter, if you're not careful. If you leave out line 1 then to run this script you would have to explicitly call Perl:

```
$perl hw.pl
Hello World
```

Lines 2-6 are just comment lines since they use the Perl comment character `#`. Perl will ignore everything from the `#` character to the end of the line.

Line 7 is really the heart of the script, in spite of being simple. It uses the Perl **print()** function to print the string *Hello World* to your terminal screen (`STDOUT`), followed by a newline (`\n`). The string was enclosed by double quotes(`" "`), which allows the interpretation of the escape sequence `\n` by the Perl interpreter.

You can also execute this example directly from the command line. The `Hello World` example can also be run using:

```
perl -e 'print ("Hello World\n");'
Hello World
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 1]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 1

---

1. **Perl and Internet Access**
   You will need access to a computer running Perl (and eventually one which runs an HTTP server on which you can test your cgi scripts). This can be a stand alone system (at work, at home) or one provided by an Internet access provider. If you do not have access to the internet, then I stongly suggest that you sign up right away with an internet access provider (IAP). You will need dial up access (PPP or SLIP, I also suggest nothing less than a 14.4 Kbps modem) and a shell account with telnet and ftp access. Talk to me if you need help with this. Typcial rates for this type of service are about $20/month (some charge an initial setup fee); some included unlimited access others limit number of hours per month. Ask questions or at least read their FAQ page and contract!

   For a list of local providers check out: http://www.currents.com/

2. **Where is Perl and what version is it?**
   Become familar with Perl on your system. Where is it? Try running the following commands and record the results.

```
$whereis perl

$which perl

$perl -v
```

   If you get an error check your path environment variable. Some places to look for Perl are:

```
/usr/local/bin
/usr/bin
```

3. **Explore the Internet and CGI**
   Got out on the Internet and find five (5) examples of CGI programs. Provide the following information:
   A. The URL for the web site
   B. A brief description of the application
   C. Any additional information you might be able to ascertain about the application, such as what language is used to program the CGI.

4. **FTP Exercise**

Go to one of the CPAN sites and ftp one of the files you find to your local machine (I suggest trying a relatively small one). Describe where you went, what file you selected and how you downloaded it using FTP.

5. **Perl on the Internet**
   You will net WWW access and a web browser for this exercise.
   Using The Perl Language Home Page (http://www.perl.com/perl/) as a starting point, answer the following questions:
   a. What is one of Larry Walls favorite books (Hint: it is NOT Programming Perl, at least that's not the answer I'm looking for.).

   b. What is the total number of *Really Nasty* Perl bugs reported?

   c. How many Alien Ports of Perl are listed at CPAN? What are they?

   d. Find where the Perl manual is and down load it (this can take a little time so try this on off hours or if you have a high speed line at work). It should be available in various formats: html, text or postscript. Pick your favorite! Everyone should have a copy so that when the systems administrator says "RTFM", you can reply, "Hey no problem!"

These exercises are to start getting you familar with writing Perl scripts

---

6. **Running Perl from the command line**
   Try running the following command on your system.

```
$date | perl -ne 'print "Today is $_"'
```

   What are the results? What type of system are you running on, and what flavor/version of perl are you using? Explain how this command line works; what do the switches `n` and `e` do?

7. **Perl script example**
   Write a Perl script that will print a line of text (could be anything you want) to your terminal 5 times. The first line will contain the text with a newline (\n), each sucessive line will have the text spaced over by tabs (\t). The number of tabs corresponds to the line number, e.g., line 2 gets 2 tabs, and so on. Run this script two ways, assuming it is named `fivelines.pl`:

```
$perl fivelines.pl
```

`$fivelines.pl`

Both should work. Explain what you did to get the second method to work.

8. **Perl script example**
Using your favorite editor, enter the following program exactly as shown here.

```
#!/usr/local/bin/perl -w
#
#
print "What is your name?";
$name = <STDIN>;
chomp($name);
$name = reverse( $name );
print "Is your name $name?\n";
exit ;
```

Make it executable and run it. What are the results? Now comment out the line which the `chomp()` function and run the script. What are the results?

9. **Perl script example**
Using what you know so far about Perl, what is wrong with the following script? You can try running it and see what happens.

```
#!/usr/local/bin/perl -w
#
#
     set the value of count
$count = 1;

print "What is your favorite number?"
$fav_num = <STDIN>;
$num  = $fav_num + $count;
print "Is your name $num?\n";
```

10. **Perl script example**
Using your favorite editor, enter the following program, make it executable and run it. What are the results? What type of system are you running on, and what flavor/version of perl are you using?

Don't worry about what these commands mean, we'll cover all this stuff!

```
#!/usr/local/bin/perl -w
#
#
foreach $var (sort keys(%ENV))
{
        print "$var  = \t $ENV{$var}\n";
```

```
}
exit ;
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 1 | HOMEWORK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 2: GETTING STARTED WITH PERL

READING ASSIGNMENT:
TYPERL 5 Ch 2-5, 10, 17

HOMEWORK ASSIGNMENT:
Assignment No. 2 (Due Next Week).

Perl MANPAGES:

- perlsyn
- perlfunc
- perldata
- perlop
- pervar
- perlstyle

TOPICS

- Perl Syntax
- Printing
- Perl Variables and Data Types
- Perl Special Variables
- Perl Operators
- Perl style -- writing readable Perl scripts
- Discussion of class project.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 3]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 2: Perl Syntax

INTRODUCTION

- A Perl script consists of a sequence of declarations and statements.
  - Need only declare report formats and subroutines
- All uninitialized user-created objects are assumed to start with a null or 0 value until they are defined by some explicit operation such as assignment.
- There are some very subtle and obscure things one can do with Perl, however in this course we will try to be clear and explicite

DECLARATIONS

- Perl is a free form language
  (except for format declarations which are discussed later)
- The # is used to designate a comment. Everything from the # to the end of the line is ignored by the Perl Interpreter
- A declaration can be put anywhere a statement can, but has no effect on the execution of the primary sequence of statements--declarations all take effect at compile time.
- Typically all the declarations are put at the beginning or the end of the script.
- Declaring a subroutine allows a subroutine name to be used as if it were a list operator from that point forward in the program. You can declare a subroutine without defining it by using the following syntax:

```
sub getUserInput;
$UserName = getUserInput "Enter your name";
```

SIMPLE STATEMENT

- Single task for the Perl interpreter to perform

- Statements consist of tokens separated by white space
  - A **token** is the smallest unit making up a Perl statement
  - **White space** consists of spaces, tabs (`\t`) and newlines (`\n`).
- New statments begin on a new line
- **Every simple statement must be terminated with a semicolon**, unless it is the final statement in a block, in which case the semicolon is optional. (A semicolon is still encouraged there if the block takes up more than one line, since you may eventually add another line.)
- Since Perl is free form all of the following are acceptable:

```
$myname = "Sam";
$myname
        =
        "Sam"
        ;
```

which consists of the tokens: `$myname`, `=`, `"Sam"`, and `;`

- **EXAMPLES**

```
$x = 1;

@UserList = qw(Sam Sue Sally Rob Jack);

$UserName = $UserList[$index];

print "$the_max\n";
```

COMPOUND STATEMENTS

- A sequence of statements that defines a scope is called a block.
- Sometimes a block is delimited by the file containing it (in the case of a required file, or the program as a whole)
- Sometimes a block is delimited by the extent of a string (in the case of an `eval`, more on this later!).
- But generally, a block is delimited by curly brackets, also known as braces. We will

call this syntactic construct a **BLOCK**.

- **BLOCK EXAMPLE**

```
if ( $x == 5 ) {
        print ("$the_max\n");
        $x = $x * 3;
}
```

WORDS

- Any unquoted word must start with an alphanumeric character. It can contain other alphanumeric characters and an underscore (_)
- Words are CaSe SeNsiTivE.
- Unquoted words could conflict with existing reserved words (use of the `-w` switch will catch this).
- If the word has no special meaning to Perl (i.e., not a FILEHANDLE or other reserved word) it will be treated as if surrounded by single quotes
- **EXAMPLES**

```
$the_word = dog;
$animal = crazy_cat;
# Unquoted string "dog" may clash with future reserved word.
[Nice warning from Perl]

$zoo = 5;
$Zoo = 6;
$ZOO = 7;
print "$zoo\n $Zoo\n $ZOO\n";
 5
 6
 7
```

QUOTES IN PERL

- Use of quotes can be confusing
- Perl uses three types of quotes:

```
single quote     '
double quote     "
back quote       `
```

- Depending on which set of quotes is used, the quotes will either allow interpolation of special characters or protect them from *interpolation*

Interpolation
In this context interpolation refers to whether or not the Perl interpreter will treat special characters and variable names literally or interpret them as the appropriate data type and replace that in the string. This is also refered to as *scalar variable substitution* . This the same as escaped and un-escaped characters in shell scripting.

- Perl also allows alternative characters to represent the standard quote characters:

Alternative Quote Symbol	Standard Quote
q	' (single)
qq	" (double)
qx	` (back)
qw	quote words (Perl 5 only)

- Perl Quoting Rules
These are important to understand since it affects if variables are interpolated (escaped).

  - Any string can be delimited by either single or double quotes
  - Double quotes must match unless embedded within single quotes
  - When a string is enclosed by double quotes scalar and array variables are interpoled, associative arrays are not
  - A single quote maybe enclosed in double quotes
  - A string enclosed by single quotes is printed literally; no interpolation
  - If double quotes are to be interpreted literally, then enclose them in single quotes.
  - Back quotes are used for command substitution; if command substitution is to occur, then the back quotes **must not** be enclosed in either single or double

quotes.

(are we having fun yet?)

- **EXAMPLES**

```
$somestring = 'This is any string';
$somestring = "This is any string";

$MyString = 'He said, "This is it!"';
print "$MyString\n";
     He said, "This is it!"

$yetanotherstring = "I can't do it now.";
print "$yetanotherstring\n";
     I can't do it now.

$wow = 'I make $100,000 for teaching, ha!';
print "$wow\n";
     I make $100,000 for teaching, ha!

$wow = "I make \$100,000 for teaching, ha!";
print "$wow\n";
     I make $100,000 for teaching, ha!

print "\$wow\n";      #what will this print?

$mydate = `date`;
print "The date is $mydate\n";
     The date is Sat Jan 19 15:33:21 PST 1995
        #The Perl interpreter will execute the shell command date and
        #store the results in the scalar $mydate.


q(A single quoted string);       'A single quoted string'

qq(A double quoted string);      "A single quoted string"

qw(dog cat mice fish birds);     "dog", "cat", "mice", "fish", "birds"
```

Note: The use of back quotes to execute shell commands is primarily a Unix operation. It may or may not work on any given port of Perl. Read your release notes and documentation for your port of Perl.

LITERALS

A literal is the same thing as a constant (literally!).

string literals

alphanumeric characters preceeded by a backslash (\).

```
\t              tab
\n              newline
\033     escape character
```

- Double quoted strings may contain string literals, which will be interpolated.

```
"This is an introduction to CGI Programming.\n"
```

STRING LITERALS

ESCAPE SEQUENCE	ASCII EQUIVALENT
\t	Tab
\n	Newline
\r	Carriage Return
\f	Form feed
\b	Backspace
\a	Alarm/bell
\e	Escape
\033	Octal Character
\xff	Hexidecimal Character
\c[	Control Character
\l	Next character is converted to **lower case**
\u	Next character is converted to **upper case**
\L	Next character is converted to **lower case** until \E is found
\U	Next character is converted to **upper case** until \E is found
\E	Ends upper or lower case conversion started with \U or \L
\Q	Do not look for special pattern characters
\v	Vertical tab
\\	Backslash

NUMERIC LITERALS

Literal numbers in Perl can be represented by positive or negative integers in decimal, octal or hex.

- Floats: floating point numbers in fixed or scientific notation
- Octal: leading `0`
- Hexidecimal: leading `0x`
- **EXAMPLES**

```
$count = 12;
$pi = 3.14159;
$finestructure = 7.2992701E-3;

$num = 033;                 #octal
$othernum = 0xfc;           #hexidecimal

print "$num \n";                    #what will these print out?
print "$othernum \n";
```

SPECIAL LITERALS

The Perl special literals shown in the table below are used as separate words and are NOT interpolated if quoted. Note that each one starts and ends with a double underscore (_).

LITERAL	DESCRIPTION
__LINE__	Current Line Number
__FILE__	Current File name (name of script file)
__END__	Logical end of script; everything after this is ignored
__DATA__	Tells Perl compiler that the Perl code is finished and everything after the __DATA__ token is availble for reading (see SelfLoader man page).

EXAMPLES

```
#!/usr/local/bin/perl -w
#
#
$theline = __LINE__;
$thefilename = __FILE__;
print ("I am at line number $theline\n");
```

```
print ("In a file called:  $thefilename\n");

__END__
I can put any information I want here!
and Perl won't give a hoot!

$hoot.pl
I am at line number 4
In a file called:  hoot.pl
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 2]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 2:
Perl Printing

FILEHANDLES

The I/O data streams in Unix are:

```
stdin  - Input
stdout - Output
stderr - Output for error messages
```

Each of these is by default connected to the terminal for an interactive session.

Perl will inherit these from the shell (Unix), however, it will access them indirectly using something called a **FILEHANDLE**. The corresponding filehandles are:

```
STDIN   - stdin
STDOUT  - stdout
STDERR  - stderr
```

By default the `print()` and `printf()` functions send their output to the `STDOUT` filehandle. You don't have to explicitly open (or close) these file handles Perl does that for you automatically.

OPEN AND CLOSE

`open`

Opens `filename` and attaches it to FILEHANDLE. If `filename` is omitted, a scaler with the same name as FILEHANDLE must exist, e.g., `STDIN`. Returns TRUE if successful, FALSE if failure.

```
open (FILEHANDLE)
open (FILEHANDLE, filename)
```

FILE MODES

To open for:	filename should look like:
read	"filename" or "<filename"
write	">filename"
read/write	"+>filename" or "+<filename"
append	">>filename"
pipe in	"Command |"
pipe out	"| Command"

NOTE: When you open a file for write, if the file already exists any contents are destroyed. If the file does not exist a new file is created.

---

close

Closes the file, pipe, or socket associated with **FILEHANDLE**. Returns TRUE if successful, FALSE if failure.

```
close (FILEHANDLE)
```

EXAMPLES

```
open(GUESTBOOK, ">guestbook.html");
...
....print to the file
...
close(MYFILE);

open(DATABASE, "<employees.db");
...
...read the file...
close(DATABASE);
```

For readability it is standard practice in Perl to use **ALL CAPS** for **FILEHANDLE** names.

The **print** function

SYNTAX

```
print (FILEHANDLE LIST)
print (LIST)
print
```

DESCRIPTION

Prints a string or a comma-separated list of strings. Returns TRUE (1) if successful, otherwise a 0 is returned. FILEHANDLE may be a scalar variable name, in which case the variable contains the name of or a reference to the filehandle, thus introducing one level of indirection.

EXAMPLES

```
#!/usr/local/bin/perl -w
#
print "This is my example \n";
print STDOUT "This is another example!\n";
#
$age = 25;
print "\t My age is $age \n\n";
```

OUTPUT

```
This is my example <BR>
This is another example!<BR>
         My age is 25 <BR>
```

Printing to a file

```
#!/usr/local/bin/perl -w
#
#       Let's have some fun with
#       Dr. Suess
#
#  from: "Oh Say Can You Say," by Dr. Suess
#
$line1 = q(We have two ducks. One blue. One black.);
$line2 = q(And when our blue duck goes "Quack-quack");
$line3 = q(our black duck quickly quack-quacks back.);
$line4 = q(The quacks Blue quacks make her quite a quacker);
$line5 = q(but Black is a quicker quacker-backer.);

#       print to STDOUT

print "$line1\n";
print "$line2\n";
print "$line3\n";
print "$line4\n";
print "$line5\n";
```

```
# print to a file, open for write only
#
open(DRSUESS, ">quack.txt");

print DRSUESS "$line1\n";
print DRSUESS "$line2\n";
print DRSUESS "$line3\n";
print DRSUESS "$line4\n";
print DRSUESS "$line5\n";

close (DRSUESS);
print "Done\n";
```

This script created the file `quack.txt` and it now contains:

```
We have two ducks. One blue. One black.
And when our blue duck goes "Quack-quack"
our black duck quickly quack-quacks back.
The quacks Blue quacks make her quite a quacker
but Black is a quicker quacker-backer.
```

The `printf` function

SYNTAX

```
printf FILEHANDLE LIST
printf LIST
```

DESCRIPTION

The `printf` function is inherited directly from the C-programming language. It therefore follows the same syntax regarding format conversions. The first item in LIST will be interpreted as the print format, and the remaining list items are the values to be printed.

The format conversions are specified in the following table.

FORMAT CONVERSIONS

SPECIFIER	DESCRIPTION
%c	single character
%d	Integer decimal format (base-10)
%e	scientific notation (floating point) [-]m.ddddde±xx
%f	floating point (fixed point notation) [-]m.dddd
%g	floating point number (compact format)
%o	Integer in octal format (base-8)
%s	character string
%u	unsigned integer
%x	Integer in hexadecimal format (base-16)
%%	Print a % (percent sign)

EXAMPLE 1

```
#!/usr/local/bin/perl -w
#
$fmt = "My age is %d and my sign is %s\n";
$age = 25;
$mysign = 'Aries.';

printf STDOUT $fmt,$age,$mysign;

$fmt = "My age is %o (octal) and my sign is %s\n";
printf STDOUT $fmt,$age,$mysign;

$fmt = "My age is %x (hex) and my sign is %s\n";
printf STDOUT $fmt,$age,$mysign;
```

OUTPUT

```
My age is 25 and my sign is Aries.
My age is 31 (octal) and my sign is Aries.
My age is 19 (hex) and my sign is Aries.
```

EXAMPLE 2

```
#!/usr/local/bin/perl -w
#
#-----------------------------------------------------
#       file: price2.pl
#
#       A simple cost-computing program.
#
```

```
#       s. gasster      8/29/96
#
#       C-program from "Joy of C", by Miller and Quilici
#       converted to Perl
#
#-----------------------------------------------------
#       define the constants
#
#
$ITEMS              = 23;
$LIST_PRICE         = 13.95;
$DISCOUNT_RATE      = .18;
$SALES_TAX_RATE     = .075;
#
#       compute costs
#
                    # total price of all items
  $total_price = $ITEMS * $LIST_PRICE;

                    # discounted price of item
  $discount_price = $total_price - $total_price * $DISCOUNT_RATE;

                    # amount of sales tax
  $sales_tax = $discount_price * $SALES_TAX_RATE;

                    # cost including sales tax
  $final_cost = $discount_price + $sales_tax;

  $DR = $DISCOUNT_RATE * 100;
  $STR = $SALES_TAX_RATE * 100;
#
#       print results
#       NOTE how the "%" sign is escaped.
#
  printf("List price per item:\t\t%7.2f\n", $LIST_PRICE);

  printf("List price of %3d items:\t\t%7.2f\n", $ITEMS, $total_price);

  printf("Price after %2.1f%% discount:\t%7.2f\n", $DR, $discount_price);

  printf("Sales tax at %2.1f%%:\t\t%7.2f\n", $STR,  $sales_tax);

  printf("Total cost:\t\t\t%7.2f\n", $final_cost);

  exit 1;

__END__
```

OUTPUT

```
List price per item:              13.95
List price of  23 items:                  320.85
Price after 18.0% discount:      263.10
Sales tax at 7.5%:                19.73
Total cost:                      282.83
```

Here Document Syntax

Perl provides a very useful and simple syntax for printing large amounts of text, which, as we shall see in later lectures, is very helpful when printing HTML from a CGI script. It is called the *Here Document* syntax, which gets its name from Unix shell programming. This is a line oriented form of quoting.

SYNTAX

```
print <<terminatorstring;
...
...
.....some lines of text
...
...
terminatorstring
```

The value of `item` are all the text from the current line down to the `terminatorstring`. The `terminatorstring` may be either an identifier (a word) or a quoted string. The types of quotes determine the treatment of the text; if unquoted the text is treated as if double quoted.

NOTE: the `terminatorstring` must be on a line by itself with no trailing space, only a carriage-return. Failing to do this results in a common error.

The syntax is best illustrated by several examples, and their discussion.

Simple Examples

```
        $amount = 15.0;
        print<<EOF;
The flowers cost \$$amount.
EOF

        print<<"EOF";
The flowers cost \$$amount.
EOF
```

Output:

```
The flowers cost $15.
```

```
        print<<'EOF';
```

```
The flowers cost \$$amount.
EOF
```

Output:

```
The flowers cost \$$amount.\n
```

Execute commands

```
print<<`EOT`;
echo 'Today is:'
date
EOT
```

Output:

```
Today is:
Fri Aug 30 14:46:39 PDT 1996
```

Stack'em

```
print<<'EOT_1', <<'EOT_2';
This is a silly example.
EOT_1
But it does work.
EOT_2
```

Output:

```
This is a silly example.
But it does work.
```

Repeat'em

```
print<< x 5;
This is a silly example.
```

Output:

```
This is a silly example.
This is a silly example.
This is a silly example.
This is a silly example.
```

```
This is a silly example.
```

You can also load variables using this general syntax. Here is an example:

```
$finestructure = 1./137.;
$pi = 3.14159;
$gravity = 9.8;

$fmt1 = <<"End_of_Text";
The fine structure constant is equal to %8.4g. \n
The ratio of the circumference of a circle to its diameter
is given by %7.5f.  The local gravitational acceleration is
%3.1f m/sec/sec.
End_of_Text

printf $fmt1, $finestructure, $pi, $gravity
```

OUTPUT

```
The fine structure constant is equal to 0.007299.

The ratio of the circumference of a circle to its diameter
is given by 3.14159.  The local gravitational acceleration is
9.8 m/sec/sec.
```

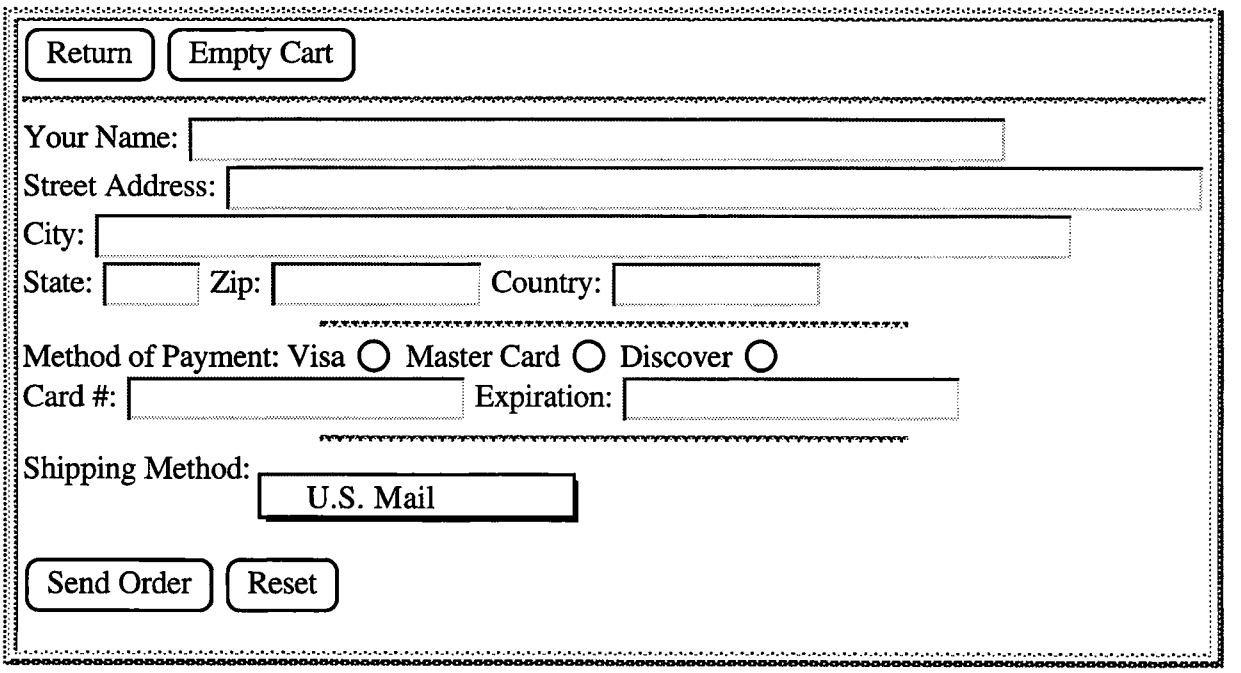
HTML Example

```
        print << EndofForm;
<FORM ACTION="$self" METHOD="POST">
<INPUT NAME="action" TYPE="SUBMIT" VALUE="Return">
<INPUT NAME="action" TYPE="SUBMIT" VALUE="Empty Cart">
<HR>
Your Name: <INPUT NAME="name" TYPE="TEXT" COLS=30 SIZE="50"><BR>
Street Address: <INPUT NAME="addr" TYPE="TEXT" COLS=50 SIZE="60"><BR>
City: <INPUT NAME="city" TYPE="TEXT" COLS=15 SIZE="60"><BR>
State: <INPUT NAME="state" TYPE="TEXT" COLS=2 SIZE="5">
Zip:  <INPUT NAME="zip" TYPE="TEXT" COLS=12 SIZE="12">
Country:  <INPUT NAME="country" TYPE="TEXT" COLS=12 SIZE="12">
<CENTER><HR WIDTH="50%"></CENTER>
Method of Payment:  Visa <INPUT NAME="pmt" TYPE="RADIO" VALUE="visa">
Master Card <INPUT NAME="pmt" TYPE="RADIO" VALUE="nc">
Discover <INPUT NAME="pmt" TYPE="RADIO" VALUE="dis"><BR>
Card #: <INPUT NAME="card" TYPE="TEXT" COLS=15 SIZE="20">
Expiration: <INPUT NAME="exp" TYPE="TEXT" COLS=9 SIZE="20"><BR>
<CENTER><HR WIDTH="50%"></CENTER>
Shipping Method: <SELECT NAME="ship" VALUE="FedEx" SIZE= 1>
<OPTION>U.S. Mail
<OPTION>UPS Ground
```

```
<OPTION>UPS Air
<OPTION>FedEx
</SELECT>
<P><INPUT  NAME="action" TYPE="SUBMIT" VALUE="Send Order">
<INPUT  TYPE="RESET">
</FORM>
EndofForm
```

Output

The output from this print will generate the enclosed HTML text. If correctly sent to a browser it would produce:



[TITLE PAGE | TABLE OF CONTENTS | LECTURE 2]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 2:
Perl Variables and Data Types

Perl uses three primary data types:

- Scalars
- Lists or Arrays
- Associative Arrays

Each type has its own namespace. Variable names are also case sensitive.

SCALAR VARIABLES

Stores exactly one item (scalar value), which could be a line of input, a string or a numeric value.

Scalar variable names:

- Always start with $
- second character must be a letter
- remaining characters can be any number of letters, digits or underscores (_)

Which are valid scalar variable names?

```
$First_name

Last_name

$42

$A15

$averylongvariablename

$_myvalue

$data
$Data
$DATA
```

Values are assigned to a scalar variable using the **assignment operator (=).**

```
$data = 5;

$UserName = "Frank N. Stein";
```

Integer Scalar Values - Integer Literals

Perl can store integer values using scalar variables. An integer value consists of

- optional ± sign
- sequence of digits.

```
123

-5

146986
```

Largest integer in Perl?

Perl stores integers internally as floating point numbers. The overall precision depends on the machine type (see Homework).

Floating Point Scalar Values or Literals

Perl also allows the use of floats. These consist of

- optional ± sign
- sequence of digits
- optional exponent (e or E)

```
75.1

-0.554

1.912e-06
```

Largest Float in Perl?

This depends on machine type. (See homework).

For MacPerl, Version 5.0.7r1m (Perl 5.001), the range of values is reported to be:

- Ints: -2^31 .. 2^31-1
- Floats: roughly -10^308 .. 10^308, with about 15 decimal digits of accuracy

Using MacPerl, Version 5.0.7r1m (Perl 5.001), on a Quadra 650, I ran the following script:

```
$num = 1.e154;
print "num = $num\n";
$num2 = $num * $num;
print "num2 = $num2\n";
$num3 = $num2 * 10.;
print "num3 = $num3\n";
```

OUTPUT

```
num = 1e+154
num2 = 1e+308
num3 = INF
```

OCTAL AND HEXIDECIMAL SCALARS

- **OCTAL**
  - Base 8 (digits: 0, 1, 2, ..., 7)
  - Octal scalars start with a 0 (zero)

```
$myoct = 047;
printf "My number is %d, decimal\n", $myoct ;
printf "My number is %o, octal\n", $myoct ;

        My number is 39, decimal
        My number is 47, octal
```

- **HEXIDECIMAL (or Hex)**
  - Base 16 (digits: 0, 1, 2, ..., 9, A, B, C, ..., F) [case insensitive]
  - Hex scalars start with 0x (zero x)

```
$myhex = 0x1f;
printf "My number is %d, decimal\n", $myhex;
printf "My number is %x, Hex\n", $myhex;

        My number is 31, decimal
        My number is 1f, Hex
```

STRING SCALARS

- String scalars or character strings in Perl are a sequence of one or more letters,

digits, spaces or special characters.

[NOTE: Unlike C, Perl character strings do not contain a hidden null termination character (\0). Remember Perl is supposed to be *easy* to use!]

- Single and Double quoted strings

INTERCHANGEABILITY OF STRINGS AND NUMERIC VALUES

- Perl can convert a string representation of a number into the corresponding numeric value, where it makes sense.
- When numeric values are read into Perl they are read as strings and converted into numeric values when appropriate.
- If the string contains characters that are NOT digits then Perl converts them to zero.

EXAMPLES

```
$ten = "10";
$result = 5 + $ten;
$answer = "hello" + 37;
#
print "\n\$ten = $ten\n";
print "\$result = $result\n";
print "\$answer = $answer\n";

$ten = 10
$result = 15
$answer = 37
```

What will the following Perl script print?

```
$ans1 = "hello" + 37;
$ans2 = "hello" + "37";
$ans3 = "hello" . "37";

print "\$ans1 = $ans1\n";
print "\$ans2 = $ans2\n";
print "\$ans3 = $ans3\n";
```

LISTS AND ARRAY VARIABLES

LIST

- Sequence of scalar values enclosed in (), with each element separated by a comma

- Perl allows list elements to be of any scalar type: string or numeric
- Scalar variables can be included as elements or part of a list element
- Expressions are also valid

```
(1, 2, "dog", "cat")

()                [empty list]

$three = 3;
(1, 2, $three, 4, 5)

($x + $y, $a +1, $c * 2)
```

ARRAY VARIABLES

- Names always start with @
- Second character is always a letter
- the remaining characters can be any number of letters, digits or underscores (_)

```
@array = (1, 2, "dog", "cat");

@names = (   "Sam",
             "Joe",
             "Jane",
             "Clare",
             "Egbert" );
```

```
@months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);

print "The months are: @months\n";

The months are: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

ACCESSING ARRAY ELEMENTS

- Each element of an array is a scalar, thus each element is referred to as if it were a scalar.
- Perl starts indexing at zero (default)

```
@names = qw(Sam Joe Jane Clare Egbert);

$student = $names[0];

print "$student \n";

        Sam
```

- Scalar variables can be used to index an array

```
$index = 2;
$student = $names[$index];

print "$student \n";

        Jane
```

LIST RANGES

List range operator (..)

Perl lets you save time when generating a sequential list of numbers or letters by using the **list range operator**.

```
(1..10) gives (1,2,3,4,5,6,7,8,9,10)

(15,16..20,30) gives (15,16,17,18,19,20,30)

(75.1..80.4) gives (75.1,76.1,77.1,78.1,79.1,80.1)
```

For numerical lists the increment is always 1. The left hand value is the starting value and the ending value is less than or equal to the right hand value.

STRING LISTS

You can also use the list range operator to generate lists with strings. Here are some examples:

```
#put ("a", "b", "c", "d", "e", "f") into @mylist
@mylist = (q/a/..q/f/);

@day_of_month = ("01".."31");
```

You can also use scalar values to generate the list:

```
$var = 34;
```

```
($var..$var+6) gives (34,35,36,37,38,39,40)
```

ARRAY VARIABLE EXAMPLES

```
#Extracting scalar values from an array

@names = qw(Sam Joe Jane Clare Egbert);
($s1, $s2) = @names;
print "$s1\n$s2\n";

#Length of an array: scalar()
$num_names = scalar( @names);
print "There are $num_names students in the class\n";

#the following also works:
#perl counts the number of elements in the array names
$num_names = @names;
print "There are $num_names students in the class\n";
```

```
Sam
Joe
There are 5 students in the class
There are 5 students in the class
```

Array Slices

Perl provides a method to extract subset from arrays. The subset is called and ***array slice***. You can use explicit index values, variable, and list range operators to select the elements for the subarray. The values do not need to be in any special order and you can repeat values. You can also assign values to array slices.

```
@names = qw(Sam Joe Jane
            Clare Egbert David
            Greg Nick Evan
            Helen Phyllis);

@Astudents = @names[1,4];
print "@Astudents\n";

@Bstudents = @names[3,2];
print "@Bstudents\n";

$current = 3;
@thestudent = @names[$current];
print "@thestudent\n";

@Cstudent = @names[6..9];
```

```
print "@Cstudent\n";

@names[9,10] = qw(Jack Jill);
print "@names[7..10]\n";
```

Results?

READING FROM STDIN

The following statement reads **a single line** from STDIN (including the newline character \n) and assigns it to the scalar variable:

```
$line = <STDIN>;
```

Reading multiple lines is possible with this statement, but a little tedious. A simpler way is to use an array variable. Recall that many Perl functions and operations are sensitive to whether the result is being assigned to a scalar or an array variable; this is one example:

```
@TheInput = <STDIN>;
```

Each line of input will be assigned to a different element of the list or array (including the newline character \n). The input is terminated by entering **^D** (Cntl-D) to signal end-of-file (EOF) on input (STDIN in this case).

```
print "Enter some lines of text, end with ^D\n";
@TheInput = <STDIN>;
print "@TheInput\n";
```

Input (by Dr. Suess)

```
One year we had a Christmas brunch
with Merry Christmas Mush to munch.
But I don't think you'd care for such.
We didn't like to munch mush much.
```

Output

```
One year we had a Christmas brunch
with Merry Christmas Mush to munch.
But I don't think you'd care for such.
We didn't like to munch mush much.
```

ARRAY LIBRARY FUNCTIONS

There are several functions that are useful for manipulating arrays. These functions are part of the standard Perl library. These are:

sort - sort an array into alphabetical order.

```
@sortedlist = sort(@mylist);
```

reverse - reverse the elements of an array.

```
@newlist = reverse(@mylist);
```

chop - remove the last character from the elements of an array.

```
chop(@mylist);
```

chomp - remove the newline character (`\n`) from the elements of an array.

```
chop(@mylist);
```

join - merge the elements of an array into a single string.

```
$string = join(EXPR, @array);
```

Joins the separate strings of `@array` into a single string, `$string`, with fields separated by the value of `EXPR`, and returns the string.

split - split a string into the elements of an array.

```
@array = split(/EXPR/, $string);
```

splits the string, `$string`, based on the pattern specified by `EXPR`.

EXAMPLES

```
@names = qw(Sam Joe Jane
            Clare Egbert David
            Greg Nick Evan
            Helen Phyllis);

#       sort the name list
@sorted_names = sort(@names);
#
#       print the sorted list, one name at a time
#
foreach $name (@sorted_names) {
        print "$name\n";
}
        Clare
        David
        Egbert
        Evan
        Greg
        Helen
        Jane
```

```
        Joe
        Nick
        Phyllis
        Sam

#       now reverse the sorted array and print
#
@rev_names = reverse(@sorted_names);
foreach $name (@rev_names) {
        print "$name\n";
}
        Sam
        Phyllis
        Nick
        Joe
        Jane
        Helen
        Greg
        Evan
        Egbert
        David
        Clare

#
#       and now for something weird
#       chop off last character from each name
#       in the reversed array
chop(@rev_names);
foreach $name (@rev_names) {
        print "$name\n";
}
        Sa
        Phylli
        Nic
        Jo
        Jan
        Hele
        Gre
        Eva
        Egber
        Davi
        Clar

#
#       join the sorted array into a single
#       string with elements separated by ::
#
$all_names = join("::", @sorted_names);
print "$all_names\n";
Clare::David::Egbert::Evan::Greg::Helen::Jane::Joe::Nick::Phyllis::Sam

#
#       now split the joined array apart
#       split on ::
#
@yet_another_array = split(/::/, $all_names);
```

```
foreach $name (@yet_another_array) {
        print "$name\n";
}
        Clare
        David
        Egbert
        Evan
        Greg
        Helen
        Jane
        Joe
        Nick
        Phyllis
        Sam
```

Had enough? Let's move on!

ASSOCIATIVE ARRAYS

In many applications we would like to be able to reference the elements of an array or list, not with a numerical subscript, but with any scalar value which is most convienent for the application. For example, imaging you wanted to keep track of how often a given domain accessed your web site. You could imagine keeping a list of domain names in one column and a total number of visits in another column (don't worry just yet how you find out the domain name of who visited your web site, we'll talk about that later).

Domain Name	No. of Hits
wow.pow.org	15
rocket.nasa.gov	10
bio.uc.uk	4
sky.ncsa.org	33
tank.army.mil	45

You could create an array to hold the totals and *just remember* that array element 0 is `xxx.yyy.org`, an so on. That gets very cumbersome, so Perl provides a beautiful and powerful technique to handle this type of situation. This is the **Associative Array**.

SYNTAX

```
#       (general form Perl4 and Perl5)

%arrayname = {KEY, VALUE, KEY, VALUE, ..., KEY, VALUE}

#               (Perl5 only)

%arrayname = {KEY => VALUE, KEY => VALUE, ..., KEY => VALUE}
```

- The arrayname convention is the same as previously described for scalar and array names.
- Perl uses the % sign to distinguish associative arrays from scalars and lists.
- Perl does **NOT** store the elements of an associative array in any specific order, so don't expect the element ordering to correspond to the order you entered.

From our previous example we might have:

```
%webhits = {"wow.pow.org" => 15,
            "rocket.nasa.gov" => 10,
            "bio.uc.uk" => 4,
            "skf.ncsa.org" => 33,
            "tank.army.mil" => 45
            };
```

Of course, in a real application such as this you would not load the associative array by hand.

- Referencing specific elements of an associative array

```
$value = $name{KEY};

$nasahits = $webhits{"rocket.nasa.gov"};
```

- Adding a value to an associative array

```
$name{KEY} = $newvalue;

$webhits{"rocket.nasa.gov"} = 12;
```

If KEY exists, then the new value is inserted into the array, if it does not exist then it is created.

- You can copy an array into an associative array, assuming the array has an **even** number of elements

```
@mylist = ("red", 10, "blue", 15, "green", 20);

%colors = @mylist;

Result is the same as:
```

```
%colors = ("red"=>10, "blue"=>15, "green"=>20);
```

- Deleting an element pair from an associative array.
  Use the built-in Perl function **`delete`**

```
delete( $colors{"red"} );
```

- List of the associative array KEYS or indices can be obtained using the built-in Perl function **`keys`**

```
@colorkeys = keys ( %colors );
print "@colorkeys\n";

blue green red
```

  Use the `sort` function to alphabetize.

- List of the associative array VALUES can be obtained using the built-in Perl function **`values`**

```
@colorvalues = values ( %colors );
print "@colorvalues\n";

15 20 10
```

- Perl lets you access both the KEYS and VALUES using the **`each`** function.

```
($key, $value) = each( %colors );

while( ($key, $value) = each( %colors ) ) {
        print "$key = $value\n";
}

blue = 15
green = 20
red = 10
```

- Test if a given key is defined for a specified associative array:

```
exists( $array{$key} );
```

  returns TRUE if the key exists.

ASSOCIATIVE ARRAY EXAMPLES

Consider the following Perl script.

```
#!/usr/local/bin/perl -w
#-----------------------------------------------------
#
#       file: read_grades.pl
#       s. gasster 6/1/95
#
#       read in the following pairs
#       (student name, letter grade)
#
#-----------------------------------------------------
#
print "Enter student name and grade (name:grade)\n";
while ($line = <STDIN>) {
    chomp( $line );
    ($name, $grade) = split(/:/,$line);
    $student_list{$name} = $grade;
     print "Enter student name and grade (name:grade)\n";
}

print "Student Names and Grades\n";
print "------------------------\n";
foreach $name ( keys(%student_list) ) {
    print "$name\t\t$student_list{$name}\n";
}
```

INPUT

```
Albert Einstein:C
Larry Wall:A+
Perl Bailey:B-
Joe Hacker:C
Don Duck:B
```

OUTPUT

```
Student Names and Grades
------------------------
Don Duck                B
Albert Einstein         C
Perl Bailey             B-
Larry Wall              A+
Joe Hacker              C
```

We will see many more examples of using associative arrays throughout this course. The associative array data type is one of the Perl language's most powerful features.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 2]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 2:
Perl Special Variables

SPECIAL VARIABLES AND CHARACTERS

The following table lists some of Perls special variables, what they are for and the default value (if it exists). Some of these will be discussed later in the course, but you will want to use this table for reference.

Most of the punctuational names have reasonable mnemonics, or analogues in one of the shells. Nevertheless, if you wish to use the long variable names, you just need to say

```
use English;
```

at the top of your program. This will alias all the short names to the long names in the current package. See the perlvar man page for additional information.

VARIABLE	DEFAULT	PURPOSE
$_	N/A	The default input and pattern search variable
$0	N/A	Contains the name of the Perl script being executed
$n (n=1,2,...)	N/A	Contains the subpattern from a successful parentheses pattern match
$&	N/A	The string from the last successful pattern match
$`	N/A	The preceding string to the last successful pattern match
$'	N/A	The string following to the last successful pattern match
$+	N/A	The last bracket matched from the last search pattern.
$*	0	Controls internal string multiline pattern matching.
$.	N/A	The current input line number of the last filehandle read.
$/	\n	Input record separator, newline bydefault
$|	0	If set to nonzero, forces the flush of the currently selected stream after every write.
$,	N/A	The output field separator for the print command.
$"	Space	The separator that joins elements of arrays interpolated in strings.

$\	N/A	The output record separator for the print command.
$;	\034	The subscript separator for multidimensional array emulation (Perl 5).
$#	N/A	The output format for printed numbers
$%	N/A	The page number of the currently selected output stream.
$=	60	The page length of the currently selected output stream.
$-	N/A	The number of lines left on the current page.
$~	filehandle	The name of the current report format for th ecurrently selected output stream.
$^	filehandle	The name of the current top of page format for the currently selected output stream.
$:	\n-	The characters used to fill a continuation field.
$^L	\f	The default form feed character.
$?	N/A	The status value returned from the last system, pipe close, or backtick command.
$!	N/A	Contains the current value of errno.
$@	N/A	The Perl syntax error from the last eval statement.
$$	N/A	The process ID (PID) of the current running Perl script.
$<	N/A	The real user ID (UID) of the currently running process.
$>	N/A	The effective UID of the currently running process.
$(	N/A	The real group ID (GID) of the currently running process.
$)	N/A	The effective GID of the currently running process.
$[	0	The index of the first element of an array.
$]	N/A	The string printed out when Perl is run with the -v command line option.
$^A	N/A	The accumulator for form line and write operations.
$^D	N/A	The current value of the debugging flags
$^F	2	The maximum number of system file descriptors.
$^I	N/A	The current value of the in-place editing flag (e.g.,`-i`)
$^P	N/A	Internal debugging flag.
$^T	N/A	The time, in seconds since Jan. 1, 1970, that the script has been running.
$^W	N/A	The current value of the warning switch.
$^X	N/A	The name of the Perl binary that was executed.
$ARGV	N/A	The name of the current file when reading from <>

STDIN	stdin	Default input stream
STDOUT	stdout	Default output stream
STDERR	stderr	Default error stream

SPECIAL ARRAYS IN PERL

The following table lists some of Perls special arrays with a brief description of what they are for. Some of these will be discussed later in the course.

ARRAY	DESCRIPTION
@ARGV	Contains command line arguments. $ARGV[0] is the first argument
@EXPORT	The list of methods the package will export by default.
@EXPORT_OK	The list of methods the package will export by request.
@INC	Contains the include path of directories to search for scripts to be evaluated by the **do** or **require** commands
@ISA	The list of base classes of the package.
@_	The parameter array for subroutines (used to pass data into a subroutine or function).
%INC	Contains entries for each file included by the **do** or **require** command
%ENV	Contains your environment variable settings. Changes made affect child processes
%OVERLOAD	Used to overload operators in a package.
%SIG	Contains signal handlers for various signals.

EXAMPLES

We will discuss most of these variables throughout the course. It is important for you to be aware of their existence and functionality. Here are a few examples.

```
@names = qw(Sam Joe Jane Clare Egbert David Greg Nick Evan Helen Phyllis);
print "@names\n";
#
# reset the list separater variable
#

$" = ":";
print "@names\n";
```

```
Output:
Sam Joe Jane Clare Egbert David Greg Nick Evan Jack Jill
Sam:Joe:Jane:Clare:Egbert:David:Greg:Nick:Evan:Jack:Jill
```

or

```
use English;
@names = qw(Sam Joe Jane Clare Egbert David Greg Nick Evan Helen Phyllis);
print "@names\n";
#
# reset the list separater variable
#

$LIST_SEPARATOR = ":";
print "@names\n";

Output:
Sam Joe Jane Clare Egbert David Greg Nick Evan Jack Jill
Sam:Joe:Jane:Clare:Egbert:David:Greg:Nick:Evan:Jack:Jill
```

Another example

```
#!/usr/local/bin/perl -w
#
#

print "Welcome to Widget Works Thingamajig\n";
print "Thingamajig Ver. 1.0 running under Perl Version $]\n"

#...the rest of the code...

__END__
```

Output

```
Welcome to Widget Works Thingamajig
Thingamajig Ver. 1.0 running under Perl Version 5.001
```

EXAMPLE Using $_

```
print "Enter Your Name> ";
        $_ = <STDIN>;
        chomp;
        print;
```

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 2: Perl Operators

Assignment Operator

- Assignment operator in Perl =
- Value on the right hand side is assigned to the variable on the left hand side.
- Multiple assignments are possible

```
$myage = 25;
$myname = "Jack Perl";
$a = $b = 3.14159;
print "$myage\n";
print "$myname\n";
print "a = $a and b = $b\n";

25
Jack Perl
a = 3.14159 and b = 3.14159
```

Arithmatic Operators

Operator	Example	Meaning
+	$a + $b	Sum of $a and $b
-	$a - $b	Difference of $a and $b
*	$a * $b	Product of $a times $b
/	$a / $b	Quotient of $a divided by $b
%	$a % $b	Remainder of $a divided by $b (modulus)
**	$a ** $b	$a to the power of $b

Note

Perl has no complex data type so exponents of a negative base must always be an integer

Examples

```
$x = 15.;
$y = 33.;
$n = 4;

$ans = $x + $y;
print "The answer is: $ans\n";
        The answer is: 48
```

```
$ans = $x - $y;
print "The answer is: $ans\n";
        The answer is: -18

$ans = $x * $y;
print "The answer is: $ans\n";
        The answer is: 495

$ans = $x / $y;
print "The answer is: $ans\n";
        The answer is: 0.454545454545455

$ans =  $y % $x;
print "The answer is: $ans\n";
        The answer is: 3

$ans = $x ** $n;
print "The answer is: $ans\n";
        The answer is: 50625

$ans = $y - 5. * $y % $n / $x ** $n;
print "The answer is: $ans\n";
        The answer is: 32.9999802469136
```

How'd we get the last answer?

Assignment Operators

Operator	Example	Meaning
=	$var = 5	assign 5 to $var
++	$var++ or ++$var	increment $var by 1 and assign to $var
--	$var-- or --$var	decrement $var by 1 and assign to $var
+=	$var += 3	increase $var by 3 and assign to $var
-=	$var -= 2	decrease $var by 2 and assign to $var
.=	$str .= "ing"	concatenate"ing" to $str and assign to $str
*=	$var *= 4	multiply $var by 4 and assign to $var
/=	$var /= 2	divide $var by 2 and assign to $var
**=	$var **= 2	raise $var to the second power and assign to $var
%=	$var %= 2	divide $var by 2 and assign remainder to $var
x=	$str x= 20	repeat $str 20 times and assign to $str

Note

Be careful when using ++ and -- with in assignments and loop structures, the results depend on whether these are used as pre or post fix operators.

Examples

What does each print statement output?

```
$var = 5;
$y = $var++;
print "$var\n";
print "Y = $y\n\n";

$var = 5;
$y =  ++$var;
print "$var\n";
print "Y = $y\n\n";

$var = 5;
$y =  $var-- ;
print "$var\n\n";

$var = 5;
$y =  --$var;
print "$var\n";
print "Y = $y\n\n";

$var = 5;
$var += 3;
print "$var\n";
print "Y = $y\n\n";

$var = 5;
$var -= 2;
print "$var\n\n";

$var = 5;
$var *= 4;
print "$var\n\n";

$var = 5;
$var /= 2;
print "$var\n\n";

$var = 5;
$var **= 2;
print "$var\n\n";

$var = 5;
$var %= 2;
print "$var\n\n";
```

Results?

String Operators

Operator	Example	Meaning
.	$a . $b	concatenate $b to the end of $a
x	$a x $b	value of $a strung together $b times

Examples

```
$str = "work";
print "The string is: $str\n";
$str .= "ing ";
print "The string is: $str\n" . "\n";
$str x= 5;
print "The string is: $str\n";
```

```
The string is: work
The string is: working

The string is: working working working working working
```

Relational Operators

Numeric Operator	String Operator	Example	Meaning
==	eq	$str eq "Word"	Equal to
!=	ne	$str ne "Word"	Not equal to
>	gt	$var > 10	Greater than
>=	ge	$var >= 10	Greater than or equal to
<	lt	$var < 10	Less than
<=	le	$var <= 10	Less than or equal to

Examples

```
$a = 'dog';
$b = 'dog';
$test = $a eq $b;
printf "Result 1 = %d\n", $test;

$a = 14;
$b = 33;
$test = $a <= $b;
printf "Result 2 = %d\n", $test;

$a = 'a';
$b = 'M';
$test = $a ge $b;
printf "Result 3 = %d\n", $test;

$a = 4;
$b = 4;
$test = $a != $b;
printf "Result 4 = %d\n", $test;
```

OUTPUT

```
Result 1 = 1
Result 2 = 1
Result 3 = 1
```

```
Result 4 = 0
```

Logical Operators

Operator	Example	Meaning
&&, and	$a && $b	True if $a is true and $b is true
||, or	$a || $b	True if $a is true or if $b is true
!, not	! $a	True if $a is not true

Examples

```
$true = 1;
$false = 0;

$true and print "It is true 1\n";
$false or print "It is true 2\n";

$false and print "It is false\n";
$true and print "It is not false\n";

not ($false) and print "It is not false 2\n";
```

OUTPUT

```
It is true 1
It is true 2
It is not false
It is not false 2
```

OPERATOR PRECEDENCE

All of the operators used in Perl have been assigned an order in which they will be performed when an expression is evaluated; first to last, highest to lowest precedence. Understanding this is very important. **Table 4.6** of *TYPERL5* lists the precedence of the Perl operators. Review it!

I strongly recommend using parentheses when you want there to be no confusion as to the order for various operations. Strive for clarity!

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 2]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 2: Perl Style

Each programmer will have his or her own style of coding. For the purposes of this class I would like the students to read the Perl sytle manual pages and try to follow the suggestions given there.

I also want each script that you write to have the following information at the top of the file. It does not have to be formatted in exactly this way, but should contain the same information.

```
#!/usr/local/bin/perl -w
#
#-------------------------------------------------------------#
#       Filename:       myfunc.pl
#
#       Algorithm Description:
#               A brief description of what the program is supposed
#               to do and how it does it.
#
#       Input/Output/Restrictions:
#               None/None/Don't run this program on Tuesdays.
#
#       Programmer:
#               A. Hacker
#
#       Revison History:
#               code started                Jan. 1, 1996
#               fixed minor syntax errors   Jan. 2, 1996
#
#-------------------------------------------------------------#
#
#       libraries or modules
#
use English;    #load the English.pm module
use CGI;                #load the CGI.pm module

#
#       constants
#
$NUsers = 100;  #maximum number of users allowed

#
#       subroutine and function declarations
#
sub getUserID;  #function to get the User ID number

...
.....
The rest of your code here
......
#
```

```
#       subroutine definitions
#

sub getUserID {

#       the actual subroutine code can go here

}

__END__
```

The __END__ isn't required by Perl, but it will be helpful for me when reading your programs. Also, you can put additional comments after the __END__ if the writers bug really bites you!

Commenting your code is essential. Not only so I can understand what you are doing, but when you revist your code in several months you want to be able to understand what you did. I have to read your homework assigments, so if I can't figure out at least what you are trying to do, you're in big trouble. But, please, don't get carried away!

Since Perl lets you use long variable names, don't hesitate to use descriptive names for your variables. This is another way to help you and others understand your code.

For example:

```
print "Enter your User ID:";
$x1 = <STDIN>;
```

This works, but the following code is easier to understand, and the variable name can be tracked within the code.

```
print "Enter your User ID:";
$UserID = <STDIN>;
...
...
$NewPay{$UserID} = $pay{$UserID} + $raise{$UserID};
```

Thus is is clear that we are tracking someone by their User ID, and giving them a pay raise based on their current salary and a raise value that is keyed to their User ID.

Basic points to follow:

- **Think before you program!**
  Take the time to think about the algorithm you are trying to develop to solve a given problem. Then think about how you can implement it. Drawing pictures or pseudocode or other techniques can help.

- **Write readable code!**
  By this I mean keep things as simple as possible while trying to develop the program (implement the algorithm). You can go back later, after it is working and make it more efficient (obscure?).

- **Don't assume** things will always work they way you intend it to work. If you are trying to open a file, you might want to check if the file exists and provide a graceful exit from the program if it does not. Almost every operation in Perl has a return value. Think of useful ways to use this to make sure your program runs without bombing or at least can trap an error and exit gracefully.

- **Initialize!**
  Perl is very friendly and will assume that an uninitialized variable has the null value. But don't be lulled into a false sense of security and laziness (yea I know what Larry says!) by this. It is good practice to always initialize your variables before you use them, then YOU know what their value is when you use them.

- **Modularity and Reusability!**
  You may find that there are some simple (and not so simple) tasks that you are required to do over and over again. Consider creating a function to do this. One example is a simple Perl function to return the last few lines of an HTML page. This is typically always the same stuff. A short function to perform this function may help make your code more readable.

```
sub HTMLend() {

return <<END_of_Text;
<HR>
<ADDRESS>
Hey you gotta problem, email the WebMaster!
<A HREF="mailto:killer@dontbugme.com">killer@dontbugme.com</A>
</ADDRESS>
</BODY></HTML>
END_of_Text
}
```

This will put the follow at the bottom of the page:

Hey you gotta problem, email the WebMaster! *killer@dontbugme.com*

- **Be Consistent!**
  If you start a program or set of programs, subroutines and functions using a particular style, don't keep changing it. If you decide you don't like it one place try a global substitution (hey, you can do this using Perl...cool!).

- **ADD COMMENTS!**

- **ADD MORE COMMENTS!**
- **ADD EVEN MORE COMMENTS!**
- **Read the Perl Style Man Pages**
  See me if you need a printed copy, or download the complete documentation from CPAN or check http://www.muon.com/

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 2]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 2

---

1. Write a program to prompt for a series of numbers until ^D then print out the number of values entered, the sum and the average value of these numbers.

2. Write a program that
   i. reads a number and then a list of strings all on separate lines
   ii. prints the string corresponding to the number entered

3. What is the result if you create a list using the list range operator and the right hand value is less than the left hand value?

4. Given the follwing values, what do each of the expressions below evaluate to?

```
$x1 = 5.;
$myfavorite_number = 23.14;
$a = 2;
$b = -19.;
```

a. `$ans = $x1 * $b % $a`

b. `$ans = $x1 / $b / $a`

c. `$ans = $myfavorite_number / $b + $a`

d. `$ans = $x1 += $b`

e. `$ans = $x1 -= $b * $a **$x1`

5. What does the following Perl program produce for output?

```
#!/usr/local/bin/perl

@mylist = ('k'..'t');
$str = 'ij';
```

```
print "@mylist\n";
$new = $str . @mylist;
print "$new\n";
$Newnew = $str . "@mylist";
print "$Newnew\n";

__END__
```

Explain the results.

6. **Associative Array Problem: Processing a Server Access Log**
The HTTP server that your browser connect to, creates a log file to keep track of server access. Typical entries look like:

```
129.219.10.154 - - [10/Sep/1996:09:22:59 -0700] "GET /catalog/super/images/scover.jpg HTTP/1.0" 200 8139
206.170.65.68 - - [10/Sep/1996:09:23:01 -0700] "GET /images/onlineca.gif HTTP/1.0" 200 21137
206.170.65.68 - - [10/Sep/1996:09:23:04 -0700] "GET /catalog/super/index.html HTTP/1.0" 200 3727
206.170.65.68 - - [10/Sep/1996:09:23:05 -0700] "GET /catalog/super/images/superbar.gif HTTP/1.0" 200 7447
206.170.65.68 - - [10/Sep/1996:09:23:07 -0700] "GET /catalog/super/images/factotea.jpg HTTP/1.0" 200 8620
206.170.65.68 - - [10/Sep/1996:09:23:17 -0700] "GET /catalog/super/images/order.gif HTTP/1.0" 200 4696
206.170.65.68 - - [10/Sep/1996:09:23:36 -0700] "GET /catalog/super/images/scover.jpg HTTP/1.0" 200 8139
206.170.65.68 - - [10/Sep/1996:09:23:46 -0700] "GET /catalog/super/images/bmlogo1.gif HTTP/1.0" 200 29218
129.219.10.154 - - [10/Sep/1996:09:24:25 -0700] "GET /catalog/super/144powertruck.html HTTP/1.0" 200 3427
129.219.10.154 - - [10/Sep/1996:09:24:30 -0700] "GET /catalog/super/images/superbar.gif HTTP/1.0" 200 7447
129.219.10.154 - - [10/Sep/1996:09:24:36 -0700] "GET /catalog/super/images/superbar.gif HTTP/1.0" 200 7447
```

Write a Perl script to do the following:

- read a file called `access.log`, containing this data
- extract the domain name and time fields (hint: use split)
- convert time from `HR:MIN:SEC` to `HR.XX` where `XX` is the fraction of an hour
- store this information in an associative array, with the time (`HR.XX`) as the key
- print out a sorted listing by time with two columns: `TIME` and `DOMAIN NAME`

IP address.

Since we have not talked about loops you don't have to make this program very general unless you want to. You should be able to do it with what we've covered in class so far, even if its a little tedious. If you want to use loops that's great. If you don't want to type in the test data, try down loading it from: http://www.muon.com/

7. Write a Perl program which prompts the user to enter the temperature in degrees C and the prints out the corresponding temperature in degrees farhenheit. Note the following conversion formula:

```
F = (9/5)*C + 32
```

8. Write a Perl script that performs the following calculations:
   - Read in an integer (`$x1`) and a floating point number (`$f1`).
   - Print these values as integer and floating point number to 4 places, respectively.
   - Compute `$x1` to the power `$f1`, print the result to 5 decimal places.
   - Compute `$f1` to the power `$x1`, print the result to 2 decimal places.
   - Compute the remainder of `$f1` and `$x1`, print the result to 6 decimal places.

9. **Integer and Floating Scalar values**
   - Enter and run the script below on your machine
   - Report the type of computer you are running, the Operating System/Version and the version of Perl you are using.
   - Discuss the results

```
#!/usr/local/bin/perl

$value = 1234567890;
printf ("first value is %d\n", $value);

$value = 1234567890123456;
printf ("second value is %17d\n", $value);

$value = 12345678901234567890;
printf ("third value is %20d\n", $value);

__END__
```

What happens if you change the `d` to `f` format types?

   - Enter and run the script below on your machine
   - Report the type of computer you are running, the Operating system and version and the version of Perl you are using.
   - Discuss the results

```
#!/usr/local/bin/perl

$value = 34.0;
print ("first value is ", $value, "\n");

$value = 114.6e-01;
print ("second value is ", $value, "\n");

$value = 178.263e19;
print ("third value is ", $value, "\n");
```

```
$value = 12345678900000000000000000000000;
print ("fourth value is ", $value, "\n");

$value = 1.23e+999;
print ("fifth value is ", $value, "\n");

$value = 1.23e-999;
print ("sixth value is ", $value, "\n");

__END__
```

10. Using the "Here Doc" syntax, write a Perl script the sends an HTML document to STDOUT which contains the following elements:

    - Centered header (H1) - call it anything you like.
    - Form with the following elemets
        - Textbox for User Name
        - Two radio buttons labeled 14.4 and 28.8, with title `Modem Speed`
        - Submit and Reset Buttons

Always indicate the version and port of Perl you are using on all your homework.

[TITLE PAGE | TABLE OF CONTENTS | HOMEWORK | LECTURE 2]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 3: CONTROL STRUCTURES AND FUNCTIONS

READING ASSIGNMENT:
TYPERL 5 Ch. 2, 8, 13

HOMEWORK ASSIGNMENT:
Assignment No. 3 (Due Next Week).

Perl MANPAGES:

- Perl Syntax (perlsyn)
- Perl Functions(perlfunc)

TOPICS

- Control structures in Perl.
- Built-in Perl functions: Part I

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 4]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 3: CONTROL STRUCTURES

This lecture discusses two general types of program flow control:

- Expression Modifiers
- Conditional Statements

Perl has no explicite Boolean data type like some other languages. However, it still has values it treats as **TRUE** and **FALSE**.

TRUE	any non-zero or non-null value
FALSE	zero or null

EXAMPLES

```
 0          FALSE
 1          TRUE
15          TRUE
"dog"       TRUE
$x=5        TRUE
""          FALSE
```

Expression Modifiers

- Recall simple statement syntax in Perl

```
expression;
```

- Perl lets you modify this expression to allow conditional evaluation
- Syntax for expression modifiers or single line conditional expressions is

```
statement keyword condexpr;
```

  Where *statement* is any Perl statement, *keyword* is the modifier and *condexpr* is the conditional expression.

- Valid Modifiers are:

```
if
unless
while
until
```

SYNTAX AND EXAMPLES

if modifier

```
EXPR2 if EXPR1;
```

if EXPR1 is TRUE then EXPR2 is evaluated.

```
perl -e '$x=5;print "\$x=$x\n" if $x==5;'

print ("The answer is $answer\n") if ($test == 0);
```

unless modifier

```
EXPR2 unless EXPR1;
```

if EXPR1 is FALSE, then EXPR2 is evaluated or EXPR1 is executed unless EXPR1 is TRUE.

```
$x = 10.;
$y = 2.;
$result = $x/$y unless ($y == 0);
print "Result = $result\n";

or

print ("The answer is $answer\n") unless ($test != 0);
```

while modifier

```
EXPR2 while EXPR1;
```

Repeat EXPR2 as long as EXPR1 is TRUE.

```
perl -e '$x=10;print $x--,"\n" while $x != -1;'
        10
        9
        8
        7
        6
        5
        4
        3
        2
        1
        0

$count = 10;
print ("Not zero yet $count\n") while ($count-- > 0);
        Not zero yet: 9
```

```
        Not zero yet: 8
        Not zero yet: 7
        Not zero yet: 6
        Not zero yet: 5
        Not zero yet: 4
        Not zero yet: 3
        Not zero yet: 2
        Not zero yet: 1
        Not zero yet: 0


$count = 10;
print ("Not zero yet $count\n") while (--$count > 0);
        Not zero yet 9
        Not zero yet 8
        Not zero yet 7
        Not zero yet 6
        Not zero yet 5
        Not zero yet 4
        Not zero yet 3
        Not zero yet 2
        Not zero yet 1
```

Note that in the last two examples the conditional expression is evaluated first, then print() statement is executed as appropriate. Also note the difference between the post-fix and pre-fix `--`.

until modifier

`EXPR2 until EXPR1;`
repeat execution of `EXPR2` as long as `EXPR1` is `FALSE`, or `EXPR2` is executed until `EXPR1` is `TRUE`.

```
perl -e '$x=10;print $x--,"\n" until $x == -1;'
        10
        9
        8
        7
        6
        5
        4
        3
        2
        1
        0

$count = 10;
print ("Not zero yet $count\n") until ($count-- <= 0);
        Not zero yet 9
        Not zero yet 8
        Not zero yet 7
        Not zero yet 6
        Not zero yet 5
```

```
Not zero yet 4
Not zero yet 3
Not zero yet 2
Not zero yet 1
Not zero yet 0
```

CONDITIONAL STATEMENTS

Perl allows the program to loop and branch over various statements and program blocks based on conditional expressions.

DECISION STRUCTURES

- **if .. elsif .. else construct**

SYNTAX

```
if (EXPR) { Block }
if (EXPR) { Block } else { Block }
if (EXPR) { Block } elsif (EXPR) { Block }
          else { Block }
```

`Block` will be executed if `EXPR` is `TRUE`.

CAUTION: Common error is to type `elseif` instead of `elsif`!!

- **unless construct**

SYNTAX

```
unless (EXPR) { Block }
unless (EXPR) { Block } else { Block }
unless (EXPR) { Block } elsif (EXPR) { Block }
              else { Block }
```

`Block` will be executed unless `EXPR` is `TRUE`.

EXAMPLES

```
if ($UserName eq "") {
        print ("Error: you are required to enter your name\n");
} else {
        print ("Thanks, $UserName, for your input\n");
}
```

Menu/Options Example

```
#!/usr/local/bin/perl -w
#
#       example of a user menu
#

print <<End_of_Menu;
(1)Read File
(2)Add Data
(3)Edit Data
(4)Save Data
(5)Exit
End_of_Menu

print "Select an Option> "
$option = <STDIN>;
chomp($option);

if ($option == 1) {
        print ("Option $option selected\n");

        #do something here

} elsif ($option == 2) {
        print ("Option $option selected\n");

        #do something here

} elsif ($option == 3) {
        print ("Option $option selected\n");

        #do something here

} elsif ($option == 4) {
        print ("Option $option selected\n");

        #do something here

} elsif ($option == 5) {
        print ("Option $option selected\n");
        exit (1);
} else {
        print ("ERROR: $option is not a valid option\n");
        exit(0);
}

#       rest of program here

__END__
```

Another Example

```
$x = 10.;
print "Enter Demoninator> ";
$y = <STDIN>;
chomp($y);
```

```
unless ($y == 0) {
        $result = $x/$y;
        print "Result = $result\n";
} else {
        print "Error: divide by zero!\n";
}
```

```
OUTPUT
Enter Demoninator> 4
Result = 2.5
Enter Demoninator> 33
Result = 0.303030303030303
Enter Demoninator> 0
Error: divide by zero!
```

LOOPING STRUCTURES

- **while loop**

SYNTAX

```
    while (EXPR) { Block }
```

`Block` will be executed while `EXPR` is `TRUE`.

EXAMPLES

```
#       INFINITE LOOP - WATCH OUT!

$secret_word = "duck";
print ("Enter a guess at the secret word (Hint: its an animal)\n");
while(1) {
        $line = <STDIN>;
        chomp($line);
        if ($line eq $secret_word) {
                print ("Hoorah! You guessed the secret word!\n");
                exit(1);
        } elsif ( $line eq "chicken") {
                print ("close! Guess again\n");
        } else {
                print ("Wrong, guess again\n");
        }
}
```

```
Enter a guess at the secret word (Hint: its an animal)
cat
Wrong, guess again
dog
Wrong, guess again
rooster
Wrong, guess again
chicken
```

```
close! Guess again
moose
Wrong, guess again
duck
Hoorah! You guessed the secret word!
```

Reading a Line

```
$secret_word = "duck";
print ("Enter a guess at the secret word (Hint: its an animal)\n");
#
while($line = <STDIN>) {
        chomp($line);
        if ($line eq $secret_word) {
                print ("Hoorah! You guessed the secret word!\n");
                exit(1);
        } elsif ( $line eq "chicken") {
                print ("close! Guess again\n");
        } else {
                print ("Wrong, guess again\n");
        }
}
```

• until loop

SYNTAX

```
until (EXPR) { Block }
```

`Block` will be executed unitl `EXPR` is `TRUE`.

EXAMPLES

```
#!/usr/local/bin/perl
#
#       listing 2.7 from TYPERL 5
#
$correct_answer = 43;      # the correct answer

print ("What is 17 plus 26?\n");

$input_answer = <STDIN>;
chomp ($input_answer);

until ($input_answer == $correct_answer) {
        print ("Wrong! Keep trying!\n");
        $input_answer = <STDIN>;
        chomp ($input_answer);
}
print ("You've got it!\n");
```

```
What is 17 plus 26?
55
```

```
Wrong! Keep trying!
33
Wrong! Keep trying!
43
You've got it!
```

- **do/until loop**

SYNTAX

```
do { Block } until (EXPR)
```

The **BLOCK** will be executed until **EXPR** is TRUE.
Note that the do constructs will always execute the loop at least once.

EXAMPLES

```
#!/usr/local/bin/perl
#
#       file: find_word.pl
#       search a file line by line for
#       a given word.  Finds only the
#       first occurance
#
$word = "quickly";

$datafile = "quack.txt";
unless (open(DATA,"$datafile") ) {
        die ("Could not open $datafile, $!\n");
}

do {
        print "\nReading file: $datafile\n";
        $line = <DATA>; #read a line from datafile
        chomp($line);
        print "$line\n";
} until ($line =~ m/$word/);

print "I found it!\n";
close(DATA);
```

```
Reading file: quack.txt
We have two ducks. One blue. One black.

Reading file: quack.txt
And when our blue duck goes "Quack-quack"

Reading file: quack.txt
our black duck quickly quack-quacks back.
I found it!
```

- **do/while loop**

SYNTAX

```
do { Block } while (EXPR)
```

The **BLOCK** will be executed while **EXPR** is TRUE.
Note that the `do` constructs will always execute the loop at least once.

EXAMPLES

In this example we ask the user to enter something and then loop until they enter something other than a null (i.e., a carriage return, since we strip off the newline character, leaving nothing).

```
$prompt = "Enter something";     #prompt to get user input
$UserInput = "";                 #what the user enters

#get input from user, don't accept null input
do {
        print ("$prompt> ");
        $UserInput = <STDIN>;
        chomp( $UserInput );     #remove newline character
} while ($UserInput eq "");
print "You entered \"$UserInput\"\n";
```

Typical Input/Output:

```
Enter something>
Enter something>
Enter something> What the heck
You entered "What the heck"
```

- **for loop**

SYNTAX

```
for (EXPR1; EXPR2; EXPR3) { Block }
```

  - o EXPR1 is used to set the initial value of the loop variables
  - o EXPR2 is used to test whether the loop should continue or stop
  - o EXPR3 is used to update the loop variables

EXAMPLES

The `for` loop is useful when you want to step through a numerical sequence or similar construct. It is of course equivalent to a corresponding `while` loop syntax.

```
$pi = 4. * atan2(1.,1.);
```

```
$i_max = 10;

for($i = 0; $i<= $i_max; $i++) {
        $theta[$i] = $pi * $i/4.;
        $cosinetable[$i] = cos( $theta[$i] );
        printf "cos(%2d *pi/4) = %6.3f\n", $i, $cosinetable[$i]
}
```

OUTPUT

```
cos( 0 *pi/4) =  1.000
cos( 1 *pi/4) =  0.707
cos( 2 *pi/4) =  0.000
cos( 3 *pi/4) = -0.707
cos( 4 *pi/4) = -1.000
cos( 5 *pi/4) = -0.707
cos( 6 *pi/4) = -0.000
cos( 7 *pi/4) =  0.707
cos( 8 *pi/4) =  1.000
cos( 9 *pi/4) =  0.707
cos(10 *pi/4) =  0.000
```

• foreach loop

SYNTAX

```
foreach VARIABLE (ARRAY) { Block }
```

- VARIABLEis local to the foreach loop and regains it's former value when the loop terminates (see example below).
- If VARIABLE is missing, the special scalar $_ is used.
- The first time the loop is executed VARIABLE is assigned the first element from the list ARRAY, each subsequent trip through the loop, VARIABLE is assigned the next element in ARRAY.

The foreach loop is very useful for traversing through arrays and associative arrays.

EXAMPLES

```
        $tmp = "Watch me pull a rabbit out of my hat!";
        print "$tmp\n";

        @fish = qw(carp cod cat halibut salmon trout);

foreach $tmp (@fish) {
        print ("I like to eat $tmp\n");
}
        print "$tmp\n";
```

OUTPUT

```
Watch me pull a rabbit out of my hat!
I like to eat carp
I like to eat cod
I like to eat cat
I like to eat halibut
I like to eat salmon
I like to eat trout
Watch me pull a rabbit out of my hat!  #$tmp unchanged!
```

Is this cool or what!! But watch out! Don't think that the last value assigned to `$tmp` will be available outside of the `foreach` loop. If `$tmp` is undefined outside of the loop it stays that ways (try it yourself). Its probably good practive to not use the same name for a global variable and a variable local to a loop. If you need to retain a local variable from a loop block you'll have to assign it to another variable.

Another example:

```
#!/usr/local/bin/perl -w

foreach $hour (1..24) {
        if ($hour > 0 && $hour <12) {
                print ("Good Morning\n");
        } elsif ($hour == 12) {
                print ("Enjoy your lunch!\n");
        } elsif ( $hour > 12 && $hour <17) {
                print ("Good Afternoon\n");
        } else {
                print ("Good Evening\n");
        }
}
__END__

Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
Enjoy your lunch!
Good Afternoon
Good Afternoon
Good Afternoon
Good Afternoon
Good Evening
Good Evening
Good Evening
Good Evening
Good Evening
Good Evening
```

```
Good Evening
Good Evening
```

Hash Example

```
%alphabet = (a=>apple,
             b=>bird,
             c=>cat,
             d=>dog,
             e=>engine);

foreach (sort keys %alphabet) {
        print "$_ is for $alphabet{$_}\n";
}
```

OUTPUT

```
        a is for apple
        b is for bird
        c is for cat
        d is for dog
        e is for engine
```

LOOP CONTROL

Perl provides mechanisms for interupting the normal flow within a loop. Perl provides labels and control statements to allow control to be transfered out of the loop.

Note: The loop control statements described here will **NOT** work with the

```
do {BLOCK} while/until {EXPR}
```

constructs, since modifiers don't take loop labels. Sorry! You can always wrap another block around it to do that sort of thing (see the example below).

- **last**

SYNTAX

```
        last;
        last LABEL;
```

The `last` statement allows you to break out of the current block within a loop, terminating the loop.

- **next**

SYNTAX

```
next;
next LABEL;
```

The `next` statement allows you to terminate the current iteration of a loop, jumping to the loop control statement and starting the next iteration. Caution: when using `next` in a `while` or `until` loop don't skip any lines of code which iterate the loop termination condition, putting you into an infinite loop.

- **redo**

SYNTAX

```
redo;
redo LABEL;
```

The `redo` statement tells Perl to restart the current iteration of the loop or block.

- **labels**

SYNTAX

```
LABEL: ...loop EXPR...
```

Labels by themselves do nothing. They are used to control the follow of a loop. Using capitals for labels helps to make them visible in the code and avoids confusion with reserved words. Labels can be used with blocks that are not part of a loop.

EXAMPLES

This example uses `last` and `redo` with one block label:

```
ATTEMPT:{
        print ("Enter course number> ");
        chomp( $number = <STDIN> );

        print ("Enter course name> ");
        chomp( $course_name = <STDIN> );

        $department{$number} = $course_name;

        print ("Are you done (y/Y)?> ");
        chomp( $ans = <STDIN> );

        last if ( $ans eq q/y/ or $ans eq q/Y/ );
        redo ATTEMPT;
}
```

```
foreach $Num (sort keys %department){
        print "Course $Num is $department{$Num}\n";
}

OUTPUT
        Enter course number> 100
        Enter course name> Quantum Mechanics
        Are you done (y/Y)?> n
        Enter course number> 101
        Enter course name> Advanced Basket Weaving
        Are you done (y/Y)?> n
        Enter course number> 102
        Enter course name> Orbital Mechanics
        Are you done (y/Y)?> n
        Enter course number> 201
        Enter course name> Introduction to Perl
        Are you done (y/Y)?> y
        Course 100 is Quantum Mechanics
        Course 101 is Advanced Basket Weaving
        Course 102 is Orbital Mechanics
        Course 201 is Introduction to Perl
```

This script computes the average salary of employees earning over $50,000 per year. There are $MAX_EMP employees, and anyone with an annual salary below $50,000 is not included in the tally.

```
#!/usr/local/bin/perl -w
#
#       From "Perl by Example," by ellie Quigley
#       Ch. 7, pg. 122, Ex. 7.17
#       with slight modifications
#       s. gasster      Sat, Sep 7, 1996
#
$MAX_EMP = 5;
print "\nEnter q/Q to quit\n";

EMPLOYEE: for($emp=1; $emp <= $MAX_EMP; $emp++) {
        do {
                print "What is the monthly rate for employee #$emp?> ";
                chomp( $monthly = <STDIN> );

                last EMPLOYEE if ($monthly eq 'q' or $monthly eq 'Q');
                next EMPLOYEE if (($year = $monthly*12.0) <= 50000.);

                $number++;
                $total_sal += $year;
                next EMPLOYEE;
        } while ($monthly ne 'q');
}
unless ($number eq "") {
        $average = $total_sal/$number;
        print "$number employees earn over \$50,000 annually\n";
```

```
        printf "Their average salary is \$%.2f\n", $average;
}
__END__
```

INPUT/OUTPUT

```
        Enter q/Q to quit
        What is the monthly rate for employee #1?> 10000
        What is the monthly rate for employee #2?> 13000
        What is the monthly rate for employee #3?> 4500
        What is the monthly rate for employee #4?> 1200
        What is the monthly rate for employee #5?> 7500
        4 employees earn over $50,000 annually
        Their average salary is $105000.00
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 3]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 3: Built-in Perl Subroutines and Functions Part I

Functions discussed in this lecture:

```
sin, cos, atan2
exp, log
int, abs, sqrt
srand, rand
index, rindex, length, substr
join
chop, chomp
```

MATHEMATICAL FUNCTIONS

TRIGONOMETRIC FUNCTIONS

Perl allows the computation of the sine and cosine of an argument whose value is in RADIANS. Note: 180 degrees = π radians.

The syntax is:

```
$value = sin(EXPR);

$value = cos(EXPR);
```

where EXPR can be any expression or scalar expressed in radians.

The arc tangent function of two arguments is calculated using:

```
$value = atan2(EXPR1, EXPR2);
```

the return value which is the angle (in RADIANS) whose tangent is EXPR1/EXPR2.

As part of the homework we will show how to compute some of the other trigonometric functions, or one can use the Trig.pm Perl module (discussed in a later lecture).

LOGARITHMIC AND EXPONENTIAL FUNCTIONS

Perl lets you compute exponentials and logarithms using the base of the natural logarithms, e (= 2.71828....).

```
$value1 = exp(EXPR1);

$value2 = log(EXPR2);
```

The return value `$value1` is the value `e` raised to the power determined by the expression `EXPR1`. The return value `$value2` is the natural logarithm (base e) of `EXPR2`. When computing logarithms, remember that `EXPR2 > 0`.

MISCELLANEOUS MATHEMATICAL FUNCTIONS

`int EXPR`

The `int` function returns the integer portion of `EXPR`. If `EXPR` is omitted, uses `$_`.

EXAMPLE

```
#!/usr/local/bin/perl -w

print "Enter an integer> ";
chomp($num = <STDIN>);
if ( int($num/2) != ($num/2) ){
        print "$num is odd.\n";
} else {
        print "$num is even.\n"
}

OUTPUT
        Enter an integer> 55
        55 is odd.
        Enter an integer> 404
        404 is even.
        Enter an integer> 2048
        2048 is even.
```

`abs(EXPR)`

This function computes and returns the absolute value of EXPR.

```
if EXPR >= 0 then abs(EXPR) returns EXPR
if EXPR >  0 then abs(EXPR) returns -EXPR


$val = abs( -15 );
print ("$val\n");

15
```

`sqrt(EXPR)`

This function computes the square root of EXPR.

```
$num = sqrt( 4.);
print "$num\n";

2

$num2 = sqrt( 37. + 3*$num);
```

```
print "$num2\n";

6.557438524302
```

RANDOM NUMBER GENERATION

Random numbers are very useful for a variety of tasks. One example which we will discuss when we get to CGI programming involves returning the user some information based on the selection of a random number. This could be an image randomly selected from a given set of images.

We will not cover the theory behind random number generation, see the references for more information. Perl provides two functions which allow the user to generate a sequence of random numbers with a uniform distribution function (i.e., equal probability) over the specified range.

srand(SEED)
: This function is used to initialize the random number generator. If no SEED value is given then Perl will use the time function to get a value. If you do not call srand with a different SEED then each call to `rand()` will not generate a different random number.

rand(MAX)
: This function returns a floating point random number between 0 and MAX.

EXAMPLE

```
#!/usr/local/bin/perl -w

@image_list = qw( bluesky.jpeg
                  ocean.jpeg
                  moon.jpeg
                  jupiter.jpeg
                  tree.jpeg);

#initialize random number generator
        srand;
        $index = int( rand(5) );

$selection = $image_list[$index];

print "The image selected is: $selection\n";

OUTPUT
The image selected is: jupiter.jpeg
The image selected is: bluesky.jpeg
The image selected is: bluesky.jpeg
The image selected is: jupiter.jpeg
The image selected is: jupiter.jpeg
The image selected is: tree.jpeg
The image selected is: tree.jpeg
The image selected is: moon.jpeg
The image selected is: moon.jpeg
The image selected is: bluesky.jpeg
The image selected is: ocean.jpeg
The image selected is: ocean.jpeg
The image selected is: ocean.jpeg
The image selected is: tree.jpeg
```

STRING MANIPULATION FUNCTIONS

index STR,SUBSTR,POSITION
index STR,SUBSTR

The index function is used to determine the position of a substring within a string. Returns the position of the first occurrence of SUBSTR in STR at or after POSITION. If POSITION is omitted, starts searching from the beginning of the string. The return value is based at 0 (or whatever you've set the $[variable to--but don't do that). If the substring is not found, returns one less than the base, ordinarily -1.

EXAMPLE

```
#!/usr/local/bin/perl

print "Enter a line of text> ";
chomp( $input = <STDIN> );

$position = index($input, "the");

if ($position >= 0) {
        print ("pattern found at position $position\n");
} else {
        print ("pattern not found\n");
}

OUTPUT
Enter a line of text> Look at the cat on the tree.
pattern found at position 8
```

rindex STR,SUBSTR,POSITION
rindex STR,SUBSTR

The rindex function performs a right-to-left substring search. Works just like index except that it returns the position of the LAST occurrence of SUBSTR in STR. If POSITION is specified, returns the last occurrence at or before that position.

EXAMPLE

```
#!/usr/local/bin/perl

print "Enter a line of text> ";
chomp( $input = <STDIN> );

$position = rindex($input, "the");

if ($position >= 0) {
        print ("pattern found at position $position\n");
} else {
        print ("pattern not found\n");
}

OUTPUT
Enter a line of text> Look at the cat on the tree.
pattern found at position 19
```

length EXPR

The `length` function returns the number of bytes in a string, i.e., the length in characters of the value of EXPR. If EXPR is omitted, returns length of `$_`.

EXAMPLE

```
#!/usr/local/bin/perl
#
#       compute the average word length

$wordcount = $charcount = 0;
while ($line = <STDIN>) {
        @words = split(/\s+/, $line);
        foreach $word (@words) {
                next if ($word =~ /^\d+\.?\d+$/);
                $word =~ s/[,.;:]$//;
                $wordcount += 1;
                $charcount += length($word);
        }
}
printf ("Average word length: %4.3f ",
        $charcount / $wordcount,"\n");

INPUT
The rain in spain
falls mainly on the plain.

OUTPUT
Average word length: 3.889
```

substr EXPR,OFFSET,LEN
substr EXPR,OFFSET

The `substr` function will get or alter a portion of a string. Extracts a substring out of EXPR and returns it. First character is at offset 0, or whatever you've set `$[` to. If OFFSET is negative, starts that far from the end of the string. If LEN is omitted, returns everything to the end of the string. If LEN is negative, leaves that many characters off the end of the string.

EXAMPLE

Extract columns 20-40 from a flat file database of the form:

```
01234567890123456789012345678901234567890123456789012345678901234567890123456789
Marvin Martian     314 Canal Rd.   Mars     1-1-456-8832-45683


chomp($line = <FLATFILE>);

$address_field = substr($line,19,20);

print "Address = $address_field\n";

OUTPUT

Address = 314 Canal Rd.   Mars
```

join EXPR,LIST

The `join` function will join a list into a string using a separator. Joins the separate strings of `LIST` into a single string with fields separated by the value of `EXPR`, and returns the string.

EXAMPLE

```
@list = ( q/Marvin Martian/,
          q/314 Canal Rd./,
          q/Mars/,
          q/1-1-456-8832-45683/);

$User_Address = join(':', @list);
print "$User_Address\n";

OUTPUT
Marvin Martian:314 Canal Rd.:Mars:1-1-456-8832-45683
```

SCALAR CONVERSION FUNCTIONS

chop(EXPR) (Perl 4 and Perl 5)

The `chop` function removes the last value from a scalar variable or the last value from every element in an array.

```
$line = q(This is the endD);
$result = chop($line);
print "$result\n";
print "$line\n";

OUTPUT
D
This is the end
```

chomp(EXPR) (Perl 5 Only)

The `chomp` function checks to see if the last characters of a string or list of strings match the value defined by the system variable `$/`. If they do then chomp removes them. The system variable `$/` is also known as `$INPUT_RECORD_SEPARATOR` in the `English` module.

EXAMPLE

```
$line = <STDIN>;
$result = chomp($line);
print "$result\n";
print "$line\n";

OUTPUT
Enter a line> This is the endD
1
This is the endD
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 3]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 3

1. Create a text file, called `address.dat`, with the following data:

```
Albert Einstein:137 Photon Lane:Newtron:CA:91159
Betty Boop:362434 Wilshire Blvd:Hollywood:CA:90024
Mickey Mouse:12 Toon Lane:Disneyland:CA:91456
Joe Hacker:59 Prison Lane:San Quentin:CA:94115
Herman Munster:1313 Mockingbird Lane:Hollywood:CA:90024
Ed Bighead:234 Rocko Road:Otown:NY:02144
Sam Iam:456 Rocko Road:Otown:NY:02144
```

   Write a perl script that will read this file print the contents to your screen in the following format:

```
Name
Street Address
City, State  Zipcode
```

2. Write a perl function that will read in cartesian values `(X,Y)` and convert them to the corresponding polar coordinate values `(r, theta)`. Test your code on the following sequence of values `(X,Y)`:

```
(X, Y)
(0.5, 0.5)
(1.4, -3.5)
(10., 0.5)
(-0.5, 15.35)
(0.0, -2.3)
```

3. Write a perl program that will randomly print out to the terminal one of 5 possible quotations each time the program is executed. An example of a quotation would be (and yes you get to select these yourself):

   A penny saved is
   A penny earned

-Ben Franklin (I think)

4. Write a Perl program that will prompt the user for a *text string* then read a user specified file and count the total number of times that string appears, what line number(s) it appears on, and what was the offset position on that line of the string. Use the following text as a test and search for "Science" and "and".

   The Department of Computer Science, the youngest department in the Physical Sciences Division, is located in the recently renovated Ryerson Laboratory, one of the oldest buildings on campus.

   Despite the department's youth, it has already become a leading research center in two areas of Computer Science: Theory and Artificial Intelligence. The department also houses research on programming language implementation, and numerical solutions of partial differential equations.

   There is considerable interaction with other departments and universities in topics ranging from mathematics and language to psychology and medicine. Some cooperative efforts include research with the Graduate School of Business, the Department of Psychology, and the University of Michigan.

   The input/output should look something like:

```
Enter a file name> UofC.txt
Enter a string to search for> xy
Will search for "xy " in file UofC.txt
The string "xy" was not found in UofC.txt


Enter a file name> UofC.txt
Enter a string to search for> at
Will search for "at " in file UofC.txt
Found "at" on line no. 2, At offset 34
Found "at" on line no. 2, At offset 60
Found "at" on line no. 3, At offset 5
Found "at" on line no. 8, At offset 18
Found "at" on line no. 9, At offset 3
Found "at" on line no. 12, At offset 37
Found "at" on line no. 12, At offset 42
Found "at" on line no. 13, At offset 36
Found "at" on line no. 14, At offset 9
Total Number of occurances of at = 9
```

5. Write a Perl script that will compute the `log_y(X)`, i.e., the logarithm of `X` in base `y`. Do some error checking. Test your program on the following:

a. log_2 (2048)
b. log_10(1000)
c. log_4(-5)
d. log_10(0.001)
e. log_10(0.5)
f. log_3(0.001)
g. log_16(64)

[TITLE PAGE| TABLE OF CONTENTS | HOMEWORK | LECTURE 3]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 4

NOTE

Unless I specifically provide you with data to test your programs, you should test each of your programs with test data that you create. Try to use enough values so that the input data will exercise all of the options/capabilities of your code. This isn't supposed to be exhaustive, I just want to see that your code works. Don't forget to include any test data with your solution so I can verify that your code processed the data correctly. Thanks.

---

1. **Report Writing Problem**

   Using the file `address.dat` that we created in Assignment 3, write a Perl script to prompt the user (menu) and perform the following functions using that file as input:

   A. Display on your terminal only the names in the file, in alphabetical order (use last name).

   B. Print the complete file to a new file (specified by the user) formatted as a table, with the following headings:

   ```
   Name        Street                  City            State          Zip
   ```

   This file is suitable for printing, so print it (I'd like to see it). But this is not a menu option.

   C. Given a user specified zip code, print to the screen all records with that zip code in the format as shown above (no colons!). If the users enter a zip code not found in the file, handle this error in a graceful manner.

   D. Exit the script

2. **Report Writing Problem**

   Write a Perl script that will read the file `congress.db` (see below) and print out the following information in a nice tabular format:

   ```
   NAME    STATE  PARTY    ADDRESS   PHONE
   ```

Your program should check the file to see if it exists and that you have read access. The NAME, ADDRESS and PHONE fields should be left justified in their columns. The STATE and PARTY fields should be centered in their columns. Assume the page size is 5 lines long and print the following header at the top (center) of each page:

```
********** CONGRESSIONAL DATA **********   PAGE X
```

where X is of course the page number.

For extra credit try sorting the listing by state and party.

congress.db file:

```
Christopher S. Bond:MO:Republican:293 Russell Senate Office Building:202/224-5721
John Ashcroft:MO:Republican:705 Hart Senate Office Building:202/224-6154
William Clay:MO:Democrat:2306 Rayburn House Office Building:202/225-2406
Vic Fazio:CA:Democrat:2113 Rayburn House Office Building:202/225-5716
Richard Gephardt:MO:Democrat:1226 Longworth House Office Building:202/225-2671
Ike Skelton:MO:Democrat:2227 Rayburn House Office Building:202/225-2876
Karen McCarthy:MO:Democrat:1232 Longworth House Office Building:202/225-4535
Pat Danner:MO:Democrat:1323 Longworth House Office Building:202/225-7041
Mel Hancock:MO:Republican:438 Cannon House Office Building:202/225-6536
Bill Emerson:MO:Republican:2268 Rayburn House Office Building:202/225-3121
Harold Volkmer:MO:Democrat:2409 Rayburn House Office Building:202/225-2956
Dianne Feinstein:CA:Democrat:331 Hart Senate Office Building:202/224-3841
Barbara Boxer:CA:Democrat:112 Hart Office Building:202/224-3553
Frank Riggs:CA:Republican:1714 Longworth House Office Building:202/225-3311
Wally Herger:CA:Republican:2433 Rayburn House Office Building:202/225-3076
John Doolittle:CA:Republican:1526 Longworth House Office Building:202/225-2511
Robert Matsui:CA:Democrat:2311 Rayburn House Office Building:202/225-7163
Lynn Woolsey:CA:Democrat:439 Cannon House Office Building:202/225-5161
George Miller:CA:Democrat:2205 Rayburn House Office Building:202/225-2095
Nancy Pelosi:CA:Democrat:2457 Rayburn House Office Building:202/225-4965
James Talent:MO:Republican:1022 Longworth House Office Building:202/225-2561
```

3. **File Access Problem**

Write and test a Perl script that will perform the following functions:

- Accept a list of file names from the user. These can be either fully qualified file names (path + filename) or just directory names. If the user gives you a directory name then get all the text files from each directory to operate on. If the user gives you full file names then operate only on those specific files. Your script should be portable so don't rely on any DOS/Windows file name length limitations.

- Strip all lines that start (column 1) with # from the files
- Makes a backup copy of each original file, before striping, with `.bck` appended to the filename
- Overwrites the original file after striping the # lines
- Test to make sure that each file exists, it is a text file, and that you have the correct access to perform these operations
- Perform error handling and have a graceful exit if failure.

4. PRIMORDIAL GUESTBOOK OR USER FEEDBACK

This is a precursor to one of the more popular WWW CGI applications: The Guestbook. We'll work on this program several times during the course and eventually have a CGI application that should work via the web.

Write and test a Perl script that does the following:

i. Load the following information that the user enters via prompts when the program is run (not via HTML forms...that comes later) and store it in an associative array:
   - Users full name (don't accept blanks here, they must enter something)
   - Users email address (don't accept blanks here, they must enter something)
   - City, State and Country
   - User comments or feedback - limit this to 2000 bytes maximum - tell them this and if they goof, have them try again.
   - Date and Time of user entry

ii. Save this data, wrapped in HTML, to a text file. It should be readable with a browser. Your program should:
   - check if file exists, if yes add the data, if not create a new file.
   - Additions to existing file should be added with the most recent first. Therefore when someone views this page, the most recent additions are at the top.

iii. A sample page might look like:

Welcome to the Primordial Guestbook

Bill Clinton <president@whitehouse.gov>
Washington DC, USA
Fri Sep 13 10:55:50 PDT 1996

Sam: I am very impressed with your website. If re-elected I will invite you to join me in Washington as the new Minister of Perl and CGI programming. So vote for me. -Bill

---

Joe Quackenbush <jquack@xyz.anydomain.com>
Anytown, CA, USA
Wed Sep 11 21:55:50 PDT 1996

I really like your website. The graphics are great, the information is great and your grandfather is great. The guestbook was the most fun. I hope I am the winner of the $1,000,000. prize! ;-) Joe.

Primordial Guestbook
© Yournamehere 1996

---

5. **IRS Audit** (oh bummer!)
You are going to write a Perl program that will decide whether to audit a given tax return (if you want to put in a hack so you never get audited I'll over look that). The tax payer database is a colon (:) separated database with the following information, one record per line:

```
last name:first name:gross income:taxable income:deduction_1: ...:deduction_N
```

where each deduction is specified by a letter (a..t) and N has a maximum value of 20. Example entry:

```
Ashcroft:John:150000:35000:a:b:c:d:e:f:g:h:i:j
```

To determine if the tax payer should be audited a score is assigned to each tax payer. The score is computed based on the following rules:

- Initial score = 0
- If taxable income < half gross income, add 20 points to score
- If number of deductions > 5, add 5 points for every deduction over 5
- If the first letter of their taxpayer's last name has an ordinal value in the alphabet which is an even number, add 40 points (e.g., A=1 (odd), B=2 (even), ...,etc.)
- If they try to cheat by leaving all fields but their name blank, they get an automatic audit

To determine if the tax payer gets audited, total their score. Then compute a random number between 0 and 100. If their score is greater than

this random number, an audit occurs. Print out a list of each tax payer's name followed by either `AUDIT` or `NO AUDIT`.

If you want to have some fun, try using names from the above database: congress.db! ;-)

[TITLE PAGE | TABLE OF CONTENTS | HOMEWORK | LECTURE 4]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 4: MORE ON I/O AND FILE MANIPULATION

READING ASSIGNMENT:
TYPERL 5 Ch. 6, 11, 12, 18

HOMEWORK ASSIGNMENT:
Assignment No. 4 (Due Next Week).

Perl MANPAGES:

- perlfunc
- perlform
- perlref
- perldsc
- perllol

TOPICS

- Filehandles,File Manipulation, File And Directory Information
- Report Writing
- References --> pointers the Perl way!.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 5]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 4: FILE ACCESS AND MANIPULATION

FILEHANDLES

- Names you associate with a file, device, pipe or socket (we'll discuss these later)
- Create a **FILEHANDLE** and attach it to a file by using the **open** command
- Perl gives you several different **FILEHANDLES** for *free*
- It is standard practice to use all capital letters for **FILEHANDLES**
- Don't use the same FILEHANDLE for different files within the same program. Use unique FILEHANDLE names.

OPEN AND CLOSE (REVIEW)
File Access Modes

open

Opens filename and attaches it to FILEHANDLE. If filename is omitted, a scaler with the same name as FILEHANDLE must exist, e.g., STDIN.

```
open (FILEHANDLE)
open (FILEHANDLE, filename)
```

FILE ACCESS MODES

To open for:	filename should look like:
read	"filename" (Default mode) or "<filename"
write	">filename"
read/write	"+>filename" or "+<filename"
append	">>filename"
pipe in	"Command |"
pipe out	"| Command"

NOTE: When you open a file for write, if the file already exists any contents are destroyed. If the file does not exist a new file is created.

close

Closes the file, pipe, or socket associated with **FILEHANDLE**.

```
close (FILEHANDLE)
```

NOTE:

- When you open a file for write, if the file already exists any contents are **destroyed**. If the file does not exist a new file is created.
- You cannot read and write to the same file at the same time.
- In append mode you cannot read from the file you are appending to.

EXAMPLES

```
open(DATAFILE, "/home/smith/results.dat");      #open for read only

        ...rest of code...

close (DATAFILE);       #close the data file

$myfile = "classlist.txt";
open(CLASSLIST, ">$myfile");    #open for write only

        ...rest of code...

close (CLASSLIST);      #close the file


open(FILE, ">>webhits.log");    #open for append only

        ...rest of code...

close (FILE);   #close the file
```

VERIFYING FILE OPEN
die and warn functions

The open function returns:

- nonzero value if successful
- zero if failed to open file

You can test for a successful open and then do your processing:

```
        if ( open( MYFILE, "mydata.dat") ) {

                        ...read the file and do some processing
        }
        next_statement;
```

If the file open failed then `next_statement` will be executed. You could also handle failure using `if {} else {}`. In many cases if the file open failed we just want to exit the program with some error message printed out. To do this use the `die()` or `warn()` functions.

`die(message)`
When the `die` function is executed the Perl interpreter prints the `message` and terminates. If you leave off a terminating newline character (`\n`) from the message then Perl will print the name of the file and the line number where the die occurred. Note that die writes to STDERR not to STDOUT.

`warn(message)`
The warn function behaves similarly to the die function, except that it does not terminate the current program, execution will continue. Note that warn also writes to STDERR not to STDOUT.

`$! ($OS_ERRNO or $ERRNO)`
The special variable `$!` contains the current value of the system error message. It can return a numeric or string values (we'll use this one). A handy mnemonic for `$!`: *What just went bang?*

Using the `die` function we can test for successful file open using:

```
#!/usr/local/bin/perl -w
#
#
$infile = "matter_antimatter.dat";

unless ( open( MYFILE, "$infile") ) {
    die ("Red Alert, Red Alert, could not open: $infile. ABORT!: $!");
    }

#       do some processing and printing to MYFILE
#       here


    close(MYFILE)
```

```
The resulting output, assuming the file does NOT exist, is:

Red Alert, Red Alert, could not open: matter_antimatter.dat.
ABORT!: No such file or directory at opentest.pl line 7.
```

We can use the `warn` function in cases where we don't need to terminate the program, but do want to know that the file was not opened. As we shall see shortly, there are better ways to handle and test file status.

```
$infile = "matter_antimatter.dat";
unless ( $ISOPEN = open( MYFILE, $infile) ) {
        warn ("Red Alert, Red Alert, could not open: $infile. Sorry!");
}

if ($ISOPEN) {

...file is open...do something with it...

} else {

...file is not open...do something else...
}
```

```
OUTPUT
# Red Alert, Red Alert, could not open: matter_antimatter.dat. Sorry!.
File 'PerlCourse:LectureNotes_html:Examples:scotty.pl'; Line 13
file is not open...do something else
```

Another Example

There is much simpler way to combine the `open` and `die/warn` functions, one that is very common. Here it is.

```
#!/usr/local/bin/perl -w
#
#
$myfile = 'congress.db';
open(MYFILE, "$myfile") or die("Hey, can't open $myfile: $!");
...
...
...
__END__


Hey, can't open congress.db: No such file or directory at - line 2.
```

READING FROM A FILE

Once the file is open for `READ ACCESS`, we can read the file contents into our program. There are several options to do this.

- **One line at a time** - scalar read
  The following command will open a file and read one line at a time, storing each line into the scalar variable `$line`. Note that when Perl reads a line from a file, it includes the newline character at the end of each line.

```
#!/usr/local/bin/perl -w
#
#       file: perlcat.pl
#
print "Enter a file name> ";
$infile = <STDIN>;
chomp($infile);

open(MYFILE, "$infile") or
    die ("Could not open $infile: $!");

while ($line = <MYFILE>) {
        print ($line);
}

close(MYFILE);
```

If you run this with a file that does not exist you get:

```
gasster> perlcat.pl
Enter a file name> dogbyte
Could not open dogbyte: No such file or directory
```

```
at perlcat.pl line 9, <STDIN> chunk 1.
```

- **Whole file** - array read
  The following command will open a file and read the complete file into the array `@input`, storing each line into a separate element of the array.

```
#!/usr/local/bin/perl
print "Enter a file name> ";
$infile = <STDIN>;
chomp($infile);

open(MYFILE, "$infile") or
        die ("Unable to open $infile: $!");

@array = <MYFILE>;
print (@array);
close(MYFILE);
```

WRITING TO A FILE

Writing to a file is fairly straight forward. First the file must be opened in *write mode*, then we can use print or printf with the corresponding `FILEHANDLE` to write to the file. And remember that if you open an existing file for `write` you will **destroy** any contents that may have existed.

Here are some examples.

```
#!/usr/local/bin/perl -w

#    create some text
@textarray = <<End_Of_Text;
The rain in spain
falls mainly on the plain.
End_Of_Text

#       open file for writting
$file = "junk.dat";
open(MYFILE, ">$file") or
    die ("Can't open $file for write: $!");

#    write data to file
foreach $line (@textarray){
        print MYFILE $line;
}
#       close file
close(MYFILE);
print "Done...\n";
```

STDIN, STDOUT and STDERR

Recall that Perl provides you with the standard Unix I/O streams by default. These are called `stdin`, `stdout` and `stderr`. Perl associates the corresponding `FILEHANDLES` `STDIN`, `STDOUT` and `STDERR` with each of these streams, respectively.

By default STDIN, STDOUT and STDERR are connected to your terminal. Also, since Perl opens/closes these for you, open and close statements are not required.

REDIRECTION

Unix allows you to redirect input and output using < and >.

```
myprog <infile >outfile
```

myprog is getting its input from stdin, which is being redirected to read from infile. Similarly the output from myprog is being sent to stdout, which is being redirected to outfile. Some Unix shells also allow you redirect stderr to a file.

You can do the same thing with a Perl program that is read and writing to STDIN and STDOUT respectively (at least on a Unix system, check your release notes on a non-Unix port of Perl).

```
#!/usr/local/bin/perl

# read all of file 1 from STDIN into an array
#
@File_1 = <STDIN>;

#
#       print all of the array to STDOUT
#       but make all letters capital
#
print "\U@File_1\E";
```

Example of running this file on itself and letting the output go to STDOUT.

```
gasster> capfile.pl <capfile.pl
#!/USR/LOCAL/BIN/PERL

 # READ ALL OF FILE 1 FROM STDIN INTO AN ARRAY
 #
 @FILE_1 = <STDIN>;

 #
 #      PRINT ALL OF THE ARRAY TO STDOUT
 #      BUT MAKE ALL LETTERS CAPITAL
 #
 PRINT "\U@FILE_1\E";
```

If we wanted to save the output to a file call allcaps.out, then we would use:

```
gasster> capfile.pl <capfile.pl >allcaps.out
```

FILE STATUS
FILE-TEST OPERATORS

Perl provides a method to determine the status of a file and some of its characteristics. The possible tests are

listed in the table below on FILE-TEST OPERATORS. The syntax for a file test is:

```
-op EXPR
```

where op is one of the characters listed in the table below and EXPR is an expression that evaluates to a string which contains a fully qualified file name (path and filename using whatever syntax is appropriate for your system).

The most commonly used ones in this course will test for the existence of a file, is it readable, writable, empty, and is it a text file.

FILE-TEST OPERATORS

OPERATOR	DESCRIPTION
b	Is *EXPR* a block device?
c	Is *EXPR* a character device?
d	Is *EXPR* a directory?
e	Does *EXPR* exist?
f	Is *EXPR* an ordinary file?
g	Does *EXPR* have its `setgid` bit set?
k	Does *EXPR* have its *sticky* bit set?
l	Is *EXPR* a symbolic link?
o	Is *EXPR* owned by the user?

FILE-TEST OPERATORS

OPERATOR	DESCRIPTION
p	Is *EXPR* a named pipe?
r	Is *EXPR* a readable file?
s	Is *EXPR* a non-empty file?
t	Does *EXPR* represent a terminal?
u	Does *EXPR* have its `setuid` bit set?
w	Is *EXPR* a writable file?
x	Is *EXPR* a executable file?
z	Is *EXPR* an empty file?

FILE-TEST OPERATORS

OPERATOR	DESCRIPTION
A	How long since last access to *EXPR*?
B	Is *EXPR* a binary file?
C	How long since last access to *EXPR's* inode?
M	How long since *EXPR* last modified?
O	Is *EXPR* owned by the *real user* only?
R	Is *EXPR* readable by the *real user* only?
S	Is *EXPR* a socket?
T	Is *EXPR* a text file?
W	Is *EXPR* writable by the *real user* only?
X	Is *EXPR* executable by the *real user* only?

NOTE: *real user* refers to the `userid` specified at login, not the effective user ID, which is the `userid` under which you are currently working.

EXAMPLES

```
#!/usr/local/bin/perl -w
#
#       file: makeallcaps.pl
#
#       get input file name
#
        print "Enter input file name> ";
        $infile = <STDIN>;
        chomp($infile);
#
#       is INFILE a text file?
#
if (-T $infile) {
    warn ("$infile is a text file, did ya know that?\n");
} else {
    warn ("$infile is NOT a text file, did ya know that?\n");
}
#
#
#       get output file name
#
print "Enter output file name> ";
while( $outfile = <STDIN>) {
        chomp($outfile);

#       check if output file exists
#       if yes, get another file name since we don't want
#       to overwrite the file
```

```
        if (-e "$outfile") {
                warn ("Output file $outfile already exists.\n");
        } else {
                last;
        }
print "Enter output file name> ";

}
#
#       try to open input file
#       if there's a problem die
#
open(INFILE, "$infile") or
        die ("Input file $infile cannot be opened.\n");
#
#       OK to open output file, die if there's a problem
#
open(OUTFILE, ">$outfile") or
        die ("Output file $outfile cannot be opened: $!");

#       read INFILE and copy all CAPS version
#       to OUTFILE
#
while ( $line = <INFILE>) {
        chomp ($line);
        print OUTFILE ("\U$line\E\n");
}

exit (1);
```

Example of running `makeallcaps.pl`:

```
gasster> makeallcaps.pl
Enter input file name> Einstein.txt
Einstein.txt is a text file, did ya know that?
Enter output file name> junk.dat
Output file junk.dat already exists.
Enter output file name> Einstein.caps
gasster> ls -tl E*
-rw-r--r--   1 gasster  users         93 Sep 21  1996 Einstein.caps
-rw-r--r--   1 gasster  users         93 Sep  8 15:43 Einstein.txt
gasster> diff Einstein.txt Einstein.caps
1,3c1,3
< Try not to become a man of sucess
< but rather try to become a man of value.
< - Albert Einstein
---
> TRY NOT TO BECOME A MAN OF SUCCESS
> BUT RATHER TRY TO BECOME A MAN OF VALUE.
> - ALBERT EINSTEIN
```

You can use file test operators with `FILEHANDLES` provided the file is opened. It is the `open` function that assigns or associates the `FILEHANDLE` with the particular file. Thus if the file is not opened, then the `FILEHANDLE` is not associated with any file.

```
open(MYFILE, "/home/jupiter/smith/weblog.dat") or
```

```
        die ("Can't open weblog.dat");

if (-z MYFILE) {        # is the file empty?
        die ("Like wow, this file is empty, I'm outta here!");
}
```

USING COMMAND LINE ARGUMENTS: @ARGV

Perl uses the array @ARGV to pass command line arguments into your Perl program. It is just like any other array in Perl, except it is initialized when your program is started by whatever values you types on the command line. It is also possible for you to set this yourself within your program if you don't want to use command line arguments, or for testing purposes.

Word search program example

(from TYPERL 5)

```
#!/usr/local/bin/perl
#-----------------------------------------------------
#
#       file:wordsearch2.pl
#
#       adapted from Listing 6.13 of
#       TYPERL 5, by David Till
#
#       s. gasster      Sun, Sep 8, 1996
#
#-----------------------------------------------------
#
$filecount = 1;
$totalwordcount = 0;
$search_word = $ARGV[0];        #word to search for in files

$numARGS = @ARGV;       #get the number of arguments

print ("Word to search for: $search_word\n");

while ($filecount <= $numARGS - 1) {
        unless (open (INFILE, $ARGV[$filecount])) {
        die ("Can't open input file $ARGV[$filecount]\n");
        }

        $wordcount = 0;
        while ($line = <INFILE>) {
        chomp ($line);
        @words = split(/ /, $line);
        $w = 1;
        while ($w <= @words) {
                if ($words[$w-1] eq $search_word) {
                $wordcount += 1;
                }
         $w++;
        }
     }
print ("occurrences in file $ARGV[$filecount]: ");
print ("$wordcount\n");
```

```
$filecount++;
$totalwordcount += $wordcount;
}
print ("total number of occurrences: $totalwordcount\n");

__END__
```

INPUT/OUTPUT

```
gasster> wordsearch2.pl man Einstein.txt kettering.txt kipling.txt
Word to search for: man
occurrences in file Einstein.txt: 2
occurrences in file kettering.txt: 0
occurrences in file kipling.txt: 0
total number of occurrences: 2
gasster>
```

READING FROM MULTIPLE FILES

The Perl operator **<>** (sometimes referred to as the *diamond* operator) allows you to process multiple files listed on the command line. When the Perl interpreter sees a line like either of the following:

```
$line = <>;              #scalar context

@in_array = <>;          #array context
```

it takes the file name to read and associate with <> from the command line. If there are multiple files listed then each will be processed after the others have been completely processed. Note that if there are no files listed on the command line the <> will assume you want input from STDIN. The diamond operator actually using our old friend @ARGV, and cycling through it. Thus if you wanted to cycle through a list of files and set @ARGV yourself you could do something like:

```
@ARGV = qw(file1 file2 file3);
while (<>) {
   print ;
}
```

What will this little program do?

---

eof FUNCTION

You almost never really need to use this function since most of the input operators will return `undef` when the input is exhausted and you can test on that (look for a null value as we have been doing). Read the textbook and the perlfunc man page description of `eof`.

FILE ACCESS

READ and WRITE Access

To open a file for both read and write access (up-date mode) use the following syntax:

```
open(MYFILE, "+>myfile")

or

open(MYFILE, "+<myfile")
```

However, **be very careful** when using these modes.

- +< is usually preferred for read/write updates; the +> mode would **clobber** the file first
- On some systems you have to do a `seek` whenever you switch between reading and writing. The effect of this is to flush out the buffer. You can get some unexpected results if you don't (see example below and the `seek` function in perl man page perlfunc).

WEIRD EXAMPLE

The following Perl script gave some strange results when run under Perl 5.003 on a Sun SPARC IPC, SunOS R4.1.2. The script is supposed to read in the first 3 lines of the file, then write each line to the same file, but in all capital letters. Here's the script, followed by the result:

```
#!/usr/local/bin/perl -w

#
#       file: rwtest.pl
#
#       test program for read/write
#       file access
#---------------------------------------#
#
#    open file for read/write

open(MYFILE, "+<testfile.txt") or
        die("Can't open testfile.txt");
#
#       read first 3 lines
#       then write them back
#       to the file in all caps
#
for ($i=1; $i<=3; $i++){
        $line = <MYFILE>;
        $data[$i-1] = $line;
 }
#   print the lines to STDOUT
print "First 3 lines are:\n";
print "\U@data\E\n";
print "-" x 65;
print "\n" x 2;

# print the lines to output file
# note I have not used seek to set
# the position within the file
#
for ($i=1; $i<=3; $i++){
    print MYFILE "$i: \U$data[$i-1]\E";
}
```

```
# close and go home!
close(MYFILE);
exit;

__END__
```

The input file:

```
I keep six honest serving-men
  (They taught me all I knew);
Their names are What and Why and When
 And How and Where and Who.
- Rudyard Kipling

Try not to become a man of success
but rather try to become a man of value.
- Albert Einstein

There is a great difference between knowing and understanding:
you can know a lot about something and not really understand it.
-Charles F. Kettering
---------------------------------
```

The results of running this script:

```
I keep six honest serving-men
  (They taught me all I knew);
Their names are What and Why and When
 And How and Where and Who.
- Rudyard Kipling

Try not to become a man of success
but rather try to become a man of value.
- Albert Einstein

There is a great difference between knowing and understanding:
you can know a lot about something and not really understand it.
-Charles F. Kettering
---------------------------------
I keep six honest serving-men
  (They taught me all I knew);
Their names are What and Why and When
1: I KEEP SIX HONEST SERVING-MEN
2:   (THEY TAUGHT ME ALL I KNEW);
3: THEIR NAMES ARE WHAT AND WHY AND WHEN
```

Not what I would have expected. It appears that in addition to our intended output, the original 3 lines read from the file have also been printed to the file. It is as if there was a buffer that did not get flushed. To fix this refer to the section on `seek` and `tell`.

seek and tell Functions

The `seek` and `tell` functions are used for positioning read/write within a file.

SYNTAX

`tell(FILEHANDLE)`

`tell` returns the distance, in bytes, from the beginning of the file to the current position in the file.

`seek(FILEHANDLE, POSITION, RELTO)`

`seek` returns TRUE if repositioning was successful, FALSE (0) if failed.

- `FILEHANDLE` - filehandle of currently opened file
- `POSITION` - number of bytes to skip in the file, may be positive (forward), zero, or negative (backward)
- `RELTO` - determines from what point in the file the positioning is done. Possible values are:
  - `0` - relative to the beginning of the file
  - `1` - relative to the current position in the file
  - `2` - relative to the end of the file

EXAMPLES

```
#!/usr/local/bin/perl -w
#
#       seek and tell
#       examples
#
#       file:seektellex.pl
#
#       move around the file
#       and print out line of data
#-----------------------------------------------#
#
#  open file for read/write

open(FILE, "+<testfile.txt") or
        die("Can't open testfile.txt: $!");


$whereami = tell(FILE);     # get current position
print "Current position = $whereami bytes\n";
$line = <FILE>;     # read line & print it
print "Line from file: $line\n";

seek(FILE,45, 0) or die("seek failed: $!");     #move forward 45 bytes
$whereami = tell(FILE);
print "Current position = $whereami bytes\n";
$line = <FILE>;      #read line, this also moves us forward
print "Line from file: $line\n";

seek(FILE,75, 0) or die("seek failed: $!");     #move forward 75 bytes
$whereami = tell(FILE);
print "Current position = $whereami bytes\n";
$line = <FILE>;
print "Line from file: $line\n";

seek(FILE,15, 1) or die("seek failed: $!");
$whereami = tell(FILE);
print "Current position = $whereami bytes\n";
$line = <FILE>;
print "Line from file: $line\n";

seek(FILE,0, 2) or die("seek failed: $!");
$whereami = tell(FILE);
print "Current position = $whereami bytes\n";
#
```

```
#       can't print anything from file here since
#       we are at the end of the file
#
#
seek(FILE,-100, 2) or die("seek failed: $!");
$whereami = tell(FILE);
print "Current position = $whereami bytes\n";
$line = <FILE>;
print "Line from file: $line\n";

close(FILE);

print "Done...\n";

exit (1);
```

The result of running this script on testfile.txt:

```
Current position = 0 bytes
Line from file: I keep six honest serving-men

Current position = 45 bytes
Line from file: me all I knew);

Current position = 75 bytes
Line from file: e What and Why and When

Current position = 114 bytes
Line from file: ere and Who.

Current position = 740 bytes
Current position = 640 bytes
Line from file: P SIX HONEST SERVING-MEN

Done...
```

WEIRD EXAMPLE - FIXED

To fix our problem discussed above, we insert a call to the `seek` function before we write to the file.

```
#!/usr/local/bin/perl -w

#
#       file: rwtest.pl
#
#       test program for read/write
#       file access
#
open(MYFILE, "+<testfile.txt") or
        die("Can't open testfile.txt");

#       read first 3 lines
#       then write them back
#       to the file in all caps
#
for ($i=1; $i<=3; $i++){
        $line = <MYFILE>;
        $data[$i-1] = $line;
```

```
 }

print "First 3 lines are:\n";
 print "\U@data\E\n";
 print "-" x 65;
 print "\n" x 2;

#
#  get current position

  $offset = tell( MYFILE );
  print "Current offset = $offset bytes\n";
#
#       added call to seek
#       goto end of file
#
        seek(MYFILE, 0, 2) or
                die("Error with file seek: $!");

for ($i=1; $i<=3; $i++){
    print MYFILE "$i: \U$data[$i-1]\E";
}
close(MYFILE);
exit;
```

The results of running this script:

```
I keep six honest serving-men
  (They taught me all I knew);
Their names are What and Why and When
 And How and Where and Who.
- Rudyard Kipling

Try not to become a man of success
but rather try to become a man of value.
- Albert Einstein

There is a great difference between knowing and understanding:
you can know a lot about something and not really understand it.
-Charles F. Kettering
---------------------------------
1: I KEEP SIX HONEST SERVING-MEN
2:   (THEY TAUGHT ME ALL I KNEW);
3: THEIR NAMES ARE WHAT AND WHY AND WHEN
```

Now that's more like it!

The next few sections deal directly with functions designed to manipulate files and directories in a Unix environment. The are closely related to Unix system functions. If you are unfamiliar or rusty on your Unix commands for file creation, deletion, permission, and directories you should review these topics in a Unix book.

Some of these functions may not work on non-Unix ports of Perl. Again **CYRN** (check your release notes)! For example, a Mac does not have the Unix style file permissions.

DIRECTORY MANIPULATION FUNCTIONS

mkdir (dirname, perms)
creates the directory specified by `dirname` with access permission specified by `perms`. The permissions are given in the standard *cryptic* Unix octal format (e.g, `0644`, see pg. 411 of *TYPERL 5* or *Unix Power Tools*). Returns `TRUE` if directory successfully created otherwise returns `FALSE`.

rmdir (dirname)
delete an empty directory; returns `TRUE` if directory successfully deleted, `FALSE` if the directory is not empty or if not deleted.

chdir (dirname)
set the value of the current default working directory to `dirname`. Returns `TRUE` if current directory correctly set, `FALSE` if there is an error.

opendir (DIRHANDLE, dirname)
opens the directory specified by `dirname`, so the program can read its contents (lists of files) and associates the directory handle `DIRHANDLE` with that directory. Returns `TRUE` if open operation was successful, `FALSE` otherwise.

closedir (DIRHANDLE)
closes the directory opened by the `opendir` function.

readdir (DIRHANDLE)
allows access to the name of each file and subdirectory in the directory associated with `DIRHANDLE` by the `opendir` command. If used in a scalar context it returns the name of the first file name in the directory, with each successive call to `readdir` returning the next file name in the list. If used in an array or list context the whole file list is sorted in the array.

$location = telldir (DIRHANDLE);
returns your position (the current directory location) in the list of files for the open directory specified by `DIRHANDLE`.

seekdir (DIRHANDLE, location)
sets the current directory location to that specified by the `location` parameter for the directory specified by `DIRHANDLE`.

rewind (DIRHANDLE)
sets the current directory location to the beginning of the list for the directory specified by `DIRHANDLE`.

EXAMPLE

```
#!/usr/local/bin/perl
#
#    file: dirtest.pl
#
#     demo dir manipulation functions
#
#     s. gasster  10/96
#-------------------------------------#
#

# reset LIST_SEPARATOR so stuff is on separate lines
$" = "\n";
```

```
# open current default directory
opendir(DIR, ".") or
   die ("can't open dir: $!");

# get list of files (and directories)
@files = readdir(DIR);
print "Files in current directory:\n @files\n";

# close directory handle
closedir(DIR);

# make a new directory

mkdir("TESTING", 0777) or
  die("Can't create new directory: $!");

# cd to new directory
chdir("TESTING") or
  die("Can't cd to new directory: $!");

# create a new file

open(NEWFILE, ">newfile.txt") or
  die("Can't create new file: $!");

print "Creating file: newfile.txt\n";

# put something in our new file
print NEWFILE <<EOI;
This is some text in the new file.
This is some text in the new file.
This is some text in the new file.
This is some text in the new file.
EOI
# close newfile
close(NEWFILE);

# get directory listing of new directory

opendir(DIR, ".") or
   die ("can't open dir: $!");
@files = readdir(DIR);
print "Files in the new directory:\n @files\n";
closedir(DIR);

exit (1);
```

OUTPUT

Note that on a Unix system the directories `./` and `../` refer to the current and parent directories.

```
gasster> dirtest.pl
Files in current directory:
 .
..
howto04
gethostname.pl
gethostadd.pl
getnets.pl
```

```
xdump.pl
price2.pl
price.pl
test.pl
hoot.pl
heretest.pl
word_search.pl
capfile.pl
allcaps.out
makeallcaps.pl
viewfiles.pl
kettering.txt
kipling.txt
Einstein.txt
wordsearch2.pl
rwtest.pl
testfile.txt
prog_12_10.pl
opentest.pl
getfilestats.pl
junk.dat
pipetestin.pl
pipetestout.pl
perlcat.pl
perlcat_errors
Einstein.copy
checkfile.pl
bug1.pl
dumpenv.pl
fmt_ex1.pl
dirtest.pl
Creating file: newfile.txt
Files in the new directory:
 .
..
newfile.txt

gasster> ls -tF
TESTING/          perlcat.pl*       wordsearch2.pl*   hoot.pl*
dirtest.pl*       opentest.pl*      Einstein.txt      test.pl*
fmt_ex1.pl*       pipetestin.pl*    kipling.txt       price.pl*
bug1.pl           junk.dat          kettering.txt     price2.pl*
dumpenv.pl*       pipetestout.pl*   viewfiles.pl*     xdump.pl*
checkfile.pl*     getfilestats.pl*  allcaps.out       getnets.pl*
Einstein.copy     testfile.txt      capfile.pl*       gethostname.pl*
makeallcaps.pl*   rwtest.pl*        word_search.pl*   gethostadd.pl*
perlcat_errors    prog_12_10.pl*    heretest.pl*      howto04/

gasster> cd TESTING/
/exp/home/gasster/perlcourse/TESTING
gasster> ls -lt
total 2
-rw-r--r--   1 gasster  users        140 Oct  6  1996 newfile.txt
gasster> cat newfile.txt
This is some text in the new file.
This is some text in the new file.
This is some text in the new file.
This is some text in the new file.
gasster>
```

telldir/seekdir Example

This example opens the current default directory and loops over the files in that directory. The listing prints the value returned by the `telldir` function, saving the value when the counter is at 5. The `seekdir` function is then used to go to that location in the directory and read the *next* file name from the list.

```
#!/usr/local/bin/perl
#
#    file: dirseek.pl
#
#     demo dir manipulation functions
#     modified version to rename file
#         and delete it after renaming.
#
#     s. gasster  10/96
#--------------------------------------#
#

# reset LIST_SEPARATOR so stuff is on separate lines
#$" = "\n";

# open current default directory
opendir(DIR, ".") or
   die ("can't open dir: $!");

# loop over list of files (and directories)
$count = 0;
while ($file = readdir(DIR) ) {
    print "File in current directory: $file\n";
    $location = telldir(DIR);
    if ($count == 5) {
      $saved_loc = $location;
    }
    print "Location [$count] = $location\n";
    $count++;
}

#  used saved directory location
seekdir(DIR, $saved_loc) or
   dir ("Seek failed: $!");

#  read the next file location
#  from the  position set by seekdir
$thefile = readdir(DIR);
print "The file location is now: $thefile\n";

# close directory handle
closedir(DIR);

print "Done...\n";
exit (1);
```

OUTPUT

```
gasster> dirseek.pl
File in current directory: .
Location [0] = 12
```

```
File in current directory: ..
Location [1] = 24
File in current directory: howto04
Location [2] = 40
File in current directory: gethostname.pl
Location [3] = 64
File in current directory: gethostadd.pl
Location [4] = 88
File in current directory: getnets.pl
Location [5] = 108
File in current directory: xdump.pl
Location [6] = 128
File in current directory: price2.pl
Location [7] = 148
File in current directory: price.pl
Location [8] = 168
....
.....(listing truncated to save space)....
....
File in current directory: dirseek.pl
Location [39] = 1024
The file location is now: xdump.pl
Done...
gasster>
```

FILE ATTRIBUTE FUNCTIONS

rename (oldname, newname)
rename a file from `oldname` to `newname`; returns `TRUE` if successful, `FALSE` if failed
The arguments can be any valid expression that returns a string that is a valid file name.

$numdeleted = unlink(filelist);
delete all the files named in `filelist`. Returns the actual number of files deleted.

link(newline, file)
create a link (*hard link*) to a file.

symlink(newlink, file)
create a symbolic link to a file.

filename = readlink(linkname);
returns the filename corresponding to a symbolic link. If `linkname` is not a symbolic link then `readlink` fails.

EXAMPLES

Let's modify our previous example to rename newfile.txt and then delete it. Only the additional code is shown

```
...
... previous Perl code goes here (dirtest.pl)
...
# rename newfile.txt
#    get new name from user
print "Enter new file name> ";
chomp($newname = <STDIN>);
```

```
# verify that new name does not exist

if (-e $newname) {
    die("File $newname already exists: $!");
}

# rename file

rename( "newfile.txt", $newname) or
    die("Can't rename newfile.txt: $!");

# get directory listing of new directory
# to verify rename

opendir(DIR, ".") or
   die ("can't open dir: $!");
@files = readdir(DIR);
print "Files in the new directory:\n @files\n";

#now delete the file

unless ( $num_deleted = unlink( $newname ) ) {
   die ("Can't delete $newname: $!");
}

# get directory listing to verify
# that file is gone
@files = readdir(DIR);
print "Files in the new directory:\n @files\n";

closedir(DIR);

print "Done...\n";
```

OUTPUT

```
...
... same listing as previously shown above
...
Enter new file name> Myfile.doc          #additional I/O
Files in the new directory:
 .
..
Myfile.doc
Files in the new directory:

Done...
gasster> ls -laF TESTING/
total 4
drwxr-xr-x   2 gasster  users        512 Oct  6 14:25 ./
drwx------   4 gasster  users       1024 Oct  6 14:25 ../
gasster>
```

The last few lines show that the renamed file Myfile.doc is now gone.

FILE PERMISSION FUNCTIONS

`$num = chmod(permissions, filelist);`

change the Unix access permissions for list of files.
`permissions` is the standard Unix file permissions mask that you want to give each file in the list. This function will not work on files that you do not have permission to modify. Returns the number of files (`$num`) whose permissions were successfully changed.

`$num = chown(userid, groupid, filelist);`

change the owner characteristics of a file. The three arguments are:

- userid - numerical user ID of the new owner of the file
- groupid - numerical group ID of the new owner of the file (-1 preserved the current groupid for the file)
- filelist - list of file whose ownership you wish to change.

Returns the number (`$num`) of files changed. You may need to have superuser or root privileges to use this function.

`$oldmaskvalue = umask(maskval);`

set the default permissions for files and directories when they are created.
`maskval` is the new mask value, and `umask` returns the previous mask value in `$oldmaskvalue`.

MISCELLANEOUS ATTRIBUTE/MANIPULATION FUNCTIONS

`truncate (filename/FILEHANDLE, length);`

truncates a file to the value specified by length (bytes). Can use either a filename or `FILEHANDLE`. If a `FILEHANDLE` is used then the file must be open.

`stat (filename/FILEHANDLE);`

returns a `LIST` containing information about the file. The items returned are given in the following table (in order that they appear in the list).

ELEMENTS OF THE LIST RETURNED BY `stat()` FUNCTION

LIST ELEMENT NUMBER	DESCRIPTION
0	Device on which the file resides
1	Internal reference number for the file (inode number)
2	Permissions for the file (file mode, returned in binary format)
3	Number of hard links to the file
4	User ID of the file owner (numerical value)
5	Group ID of the file owner (numerical value)
6	Device type, if file is actually a device
7	Size of file (Bytes)
8	Time file was last accessed
9	Time file was last modified
10	Time file status was last changed
11	Optimal (preferred) block size for I/O operations on the file system containing the file
12	The number of blocks allocated for this file

NOTE:

The time values returned by the stat function are the number of seconds since `00:00:00, January 1, 1970.` However, **VERIFY** that this is the same for your port of Perl. For example, the Mac (MacPerl) uses 1904 not 1970.

EXAMPLES

```
#  change file permissions
print ("Enter new file permissions (XXXX)> ");
chomp( $new_perms = <STDIN> );

# translate perms to octal
$new_perms = oct($new_perms);

chmod($new_perms, $file) or
  die("Can't change permissions on $file: $!");

# verify new perms
#  get file stats
($dev,$ino,$mode,$nlink, $uid, $gid, $rdev,
$size, $atime, $mtime,$ctime,$blksize, $blocks) = stat($file);

# convert binary file mode to correct octal mode for use
# with chmod type permission format (weird!)

$perms = $mode & 07777;
printf "File mode for %s = %o\n", $file, $perms;

exit (1);
```

INPUT/OUTPUT

```
gasster> ls -l TESTING/                #initial directory listing
total 0
-rw-r--r--   1 gasster  users          0 Oct  6 14:33 guestbook.cgi
-rw-r--r--   1 gasster  users          0 Oct  6 14:34 shoppingcart.pl

gasster> chmod_ex.pl                    #change perms on file
Enter file name> ./TESTING/guestbook.cgi
File mode for ./TESTING/guestbook.cgi = 644
Enter new file permissions (XXXX)> 755
File mode for ./TESTING/guestbook.cgi = 755

gasster> ls -l TESTING/                 #check result
total 0
-rwxr-xr-x   1 gasster  users          0 Oct  6 14:33 guestbook.cgi
-rw-r--r--   1 gasster  users          0 Oct  6 14:34 shoppingcart.pl
gasster>

checking
rwx = 4+2+1 = 7
r-x = 4+0+1 = 5
r-x = 4+0+1 = 5
```

Example of `stat()` output (for guestbook.cgi):

```
59834218        device ID
982273          file inode number
33188           file permissions (binary)
1               number of hard links to file
6075            UID of file owner  (that's me!)
200             GID of file owner
```

```
0              device type
0              size of file in bytes (its empty!)
844637635      time since last access
844637635      time since last modification
844663879      time since last file status change
8192           preferred block size for I/O on the file system
0              number of allocated blocks

Converted modification time from
seconds since epoch to localtime:
File modification time:
55  seconds
33  minutes
14  hours            (compare with results of ls -l above)
6   day of month
9   month (Oct, since Jan = 0)
96  Year
0   day of week (Sun = 0)
279 day of year
1   flag for daylight savings
```

PIPES

- **What is a pipe?**

  pipe

  A connection between the standard output of one process and the standard input of a second process

  The Unix OS allows two commands to have their input and output linked using a pipe. Thus one can pipe the output of one command (from `stdout`) into the input (via `stdin`) of the other command.

  Count the number of times the die function is called in all my Perl scripts:

```
$grep die *.pl | wc -l
$ 12
```

  Sorted directory listing

```
$ls |sort
Einstein.txt
allcaps.out
capfile.pl
getfilestats.pl
heretest.pl
hoot.pl
kipling.txt
makeallcaps.pl
price.pl
rwtest.pl
testfile.txt
viewfiles.pl
word_search.pl
xdump.pl
```

- **You can pipe in Perl!**

Perl lets you open pipes for both input to, and output from, your Perl program. We use our old friend the open command:

EXAMPLE 1

```
#!/usr/local/bin/perl -w
#-----------------------------------------------#
#       demo of perl pipes
#
#       file: pipetestout.pl
#
#       s. gasster
#       perl course
#
#       9/14/96
#
#-----------------------------------------------#
#
#       open for output
#
open(MYPIPE, "|cat>junk.dat") or
        die("Can't pipe to junk.dat: $!");

print MYPIPE<<EOT;
This is a test of the pipe
command.  It should use
cat to output to a file.
EOT

close(MYPIPE);
exit (1);
__END__
```

This example opens a pipe to the cat program and dumps some text to the file junk.dat using this pipe.

EXAMPLE 2

```
#!/usr/local/bin/perl -w
#-----------------------------------------------#
#       file: pipetestin.pl
#
#       demo of perl pipes
#
#       s. gasster
#       perl course
#
#       9/14/96
#
#-----------------------------------------------#
#       open for input
#
open(MYPIPE, "cat *.txt|") or
        die("Can't pipe: $!");

#       read the input files
        @input = <MYPIPE>;
```

```
close(MYPIPE);
#
#       dump to STDOUT
#
        print "Input text files:\n";
        print "@input";

exit (1);
__END__
```

This example uses cat to concatenate all the *.txt files into one temporary file which is then used as input to the Perl script. After the open, which assigns a `FILEHANDLE` to the pipe we only need to refer to the `FILEHANDLE` to read the file. In this example I have chosen to read all the files into one array, `@input`. You could also read this information one line at a time using

```
$input = <MYPIPE>;
```

Some of the topics not discussed that you should read about in the textbook and references:

- read
- sysread
- syswrite
- getc
- binmode

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 4]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 4: REPORT WRITING

INTRODUCTION

Recall that Perl = Practical Extraction and Reporting Language.

After you've done some Practical Extraction you will need to Report your results.

Perl's report generation syntax may seem a bit *weird* at first, but its not too difficult to master. In fact if you're familiar with Fortran, Perl has borrowed the `format` and `write` commands from this language (hard to believe huh?).

For the purposes of this course the report generation capability may provide a useful adjunct to our CGI capabilities where most of our output will be formatted using HTML. There are instances where it is simplest to return output formatted by Perl and parse it to the user using the HTML tag `<PRE>`.

Perl allows you to specify the filehandle for your output so you can send it to `STDOUT` (by default) or to any disk file.

REPORT WRITING

Perl has a mechanism to help you generate simple reports. To facilitate this, Perl helps you layout your output page(s) using the `format` statement. Perl lets you keep track of things such as how many lines on a page, what page you're on, when to print page headers, etc.

- **Formats**
  - A format is a template for a report
  - The format defines the column labels and fixed part of the report as well as the variable part of the report
  - Formats can also define the headers for each page of the report
  - Unfortunately, Perl does not support footers in formats. You have to do your own processing to add footers to each page of the report.
  - Formats have their own namespace
  - Format definitions can be placed anywhere in the Perl program, but can NOT be nested
- **Why Use Formats?**
  - More efficient than a bunch of printf's
  - Built-in page header processing
  - Built-in filled text blocks

- **Defining a Print Format**

SYNTAX

Output record formats are declared as follows:

```
format NAME =
FORMLIST
.
```

If NAME is omitted, then the default format STDOUT is defined. FORMLIST consists of a sequence of lines, each of which may be of one of three types:

1. A comment, indicated by putting a # in the first column.
2. A *picture line* giving the format for one output line.
3. An argument line supplying values to substitute into the previous picture line.

- Note that the format definition ends with a line containing a period by itself. The period MUST be the first and only character on the line.
- The FORMLIST consists of a sequence of picture lines (also called field lines)
- A picture line can have picture fields (also called field holders) which define places where variable text will be substituted
- If a picture line has picture fields, it is followed by a value line that defines the scalar values to be used for the fields
- The values on the value line are separated by commas

The final line terminates the format statement and consists of a period (.) by itself (first column) on the line.

EXAMPLES

This simple example uses STDOUT since the format has no name:

```
format =
#this is my first simple format
     ************
     Hello World!
     ************
.
```

This example uses a named format:

```
format GREETING =
************
Hello World!
************
.
```

• Displaying a Print Format

To invoke the print format we need to do two things:

- Set the system variable `$~ ($FORMAT_NAME)` to the format you want to use
- Invoke the `write` function

EXAMPLES

```
$~ = "GREETING";
write;

format GREETING =
************
Hello World!
************
.
```

This script produces the output:

```
************
Hello World!
************
```

Formats, like packages and subroutines, are declared rather than executed, so they may occur at any point in your program. Usually it's best to keep them all together, either at the beginning or all at the end of your program.

They have their own namespace apart from all the other *types* in Perl. This means that if you have a function named MYFUNC, it is not the same thing as having a format named MYFUNC.

The default name for the format associated with a given filehandle is the same as the name of the filehandle. Thus, the default format for STDOUT is named STDOUT, and the default format for filehandle TEMP is named TEMP. For readability I recommend using all capital letters for format names, and in general using unique names.

• Displaying Values in a Print Format: Value-Field Formats

In order to print any of the Perl data types or expressions we need to define value fields which appear in the format picture lines. Each value field specifies the value to be displayed. The desired format for the value is also specified.

Value field formats are printed exactly as they look unless they contain special value field characters which define the print format.

• Basic Picture Fields

- The basic types of picture fields are:

Perl Value Fields

FIELD	VALUE-FIELD FORMAT
@<<<<<<	left adjusted 7-character field
@>>>>>	right adjusted 5-character field
@|||||	centered 6-character field
@###.####	numeric format with implied decimal point (7-digits)
@*	a multi-line field.

- The field width is specified by the number of <'s, >'s, |'s or #'s (including the @)
- For a numeric field, the number of #'s to the right of the decimal point in the definition specifies the number of places to the right of the decimal place in the output
- The pad character is space
- If the value is too large for the field width, the value is truncated
- If the value contains a newline, output is stopped at the first newline encountered. (But see Multiline Fields below.)

EXAMPLES

Example 1

```
#!/usr/local/bin/perl -w
#
#  simplified example from "TYPERL5," by Till
#  Listing 11.4
#-----------------------------------------------#
#
#   write out the header
$company = 'Camel Widgets, Inc.';
$totalstring = 'Grand Total:';

$~ = "COMPANY"; #set current format
write;

#   load customer information and print it
$custline = 'Larry Wall#Sat, Oct 5, 1996';
($customer, $date) = split(/#/, $custline);

$~ = "CUSTOMER";
write;

#   load the customer order data
%orders = ( 'Flying Widget'=>195.33,
            'Green Night Widget'=>350.44,
            'UnderWater Widget'=>1000.15
          );
```

```
#   loop over order data and print as we go
$total = 0;
while ( ($item, $cost) = each(%orders)  ){
    $~ = "ORDERLINE";
    write;
    $total += $cost;
}

#    finally, print the total purchase
$~="TOTAL";
write;

exit (1);


# define formats

format COMPANY =
************* @|||||||||||||||||||||||||||| *************
$company
.

format CUSTOMER =
@<<<<<<<<<<<<<<<<<<<<<<<<<<<<          @>>>>>>>>>>>>>>>>>
$customer, $date
.

format ORDERLINE =
          @<<<<<<<<<<<<<<<<<<<<<<<<<<<<          $@####.##
$item, $cost
.

format TOTAL =
                                                  ----------
@<<<<<<<<<<<<<<                                   $@####.##
$totalstring, $total
.
```

OUTPUT

```
*************       Camel Widgets, Inc.       *************
Larry Wall                                  Sat, Oct 5, 1996
          UnderWater Widget                         $ 1000.15
          Flying Widget                             $  195.33
          Green Night Widget                        $  350.44
                                                    ----------
Grand Total:                                        $ 1545.92
```

NOTE: The $ is interpreted literally on the picture line.

Example 2

This example reads a database of hardware information and prints out a nicely formatted report.

```
#!/usr/local/bin/perl -w
```

```
#
#-----------------------------------------------------------------#
#       Filename:       hardware_rpt.pl
#
#   VERSION: 1.0
#
#       Algorithm Description:
#               read the hardware.dat data file and printout
#               a formated report.
#
#       Input:
#               reads from file: hardware.dat
#       this file is a database with fields
#               delimited by |.
#
#       Output:
#               prints report to STDOUT
#
#       Restrictions:
#               None
#
#       Programmer:
#               s. gasster
#
#       Revision History:
#               original code from "Perl 5 How-To," by Glover, et al.
#               modified to open/close data file        Oct 5, 1996
#
#-----------------------------------------------------------------#
#
#
$InFile = "hardware.dat";  # input file name

#
#       open data file
#

open(DATA, "$InFile") or
    die ("Problem opening $InFile: $!");

#    read a line from the file until
#    there are no more records.

while ($line = <DATA>){
    chomp($line);

    # Split the input line on the | symbol.
    ($number,$type,$name,$price,$desc) = split (/\|/, $line);

    # Print out the item information.
    write;
}

close (DATA);
#

format STDOUT=
@>>>> @|||||| @<<<<<<<<< @<<<<<<<<<<<<< $@####.## ^<<<<<<<<<<<<<<<<<<<<<<<<<<
$.,$number,$type,$name,$price,$desc
~                                                  ^<<<<<<<<<<<<<<<<<<<<<<<<<<
$desc
```

```
~                                                ^<<<<<<<<<<<<<<<<<<<<<<<<<<
$desc
.

exit (1);
```

OUTPUT

```
   1   101   Hardware   Hammer          $   25.00 A thing to hit nails with.
   2   121   Hardware   Nail            $    0.15 A thing to be hit by a
                                                  hammer. (see hammer)
   3   142   Hardware   Sander          $   10.15 Used to sand all sorts of
                                                  things, except sand.
   4   206   Household  Kitchen Sink    $  100.00 Lost the last time I went
                                                  on vacation. (next time
                                                  this I won't check it in)
   5   210   Household  Windows         $   45.00 User friendly and does not
                                                  need to be plugged in.
   6   242   Household  Vacuum          $  300.00 I'm not sure what this is
                                                  used for; but people seem
                                                  to have them anyway.
   7   266   Household  Microwave       $  100.00 Kitchen item. (Also see
                                                  hot water heater)
   8   312   Garden     Rake            $   25.00 Used to rake leaves, cut
                                                  grass and break noses if
                                                  left lying on the ground.
   9   344   Garden     Gravel          $   30.00 Why bother the dog is
                                                  going to dig through it
                                                  again anyway.
  10   362   Garden     Top Soil        $   10.00 Is there such a thing as
                                                  "Bottom Soil"?
  11   384   Garden     Mosquitos       $    0.00 Some are large as a dog
                                                  and will carry you away.
  12   500   Sports     Hockey Stick    $   10.00 Used to push the puck
                                                  around.
  13   501   Sports     Bike            $  400.00 Sits in the garage and
                                                  collects dust.
  14   556   Sports     First Aid Kit   $   25.00 Used more times than not
                                                  unfortunately.
  15   601   Misc       Dog Food        $    1.50 The stuff the cat eats.
  16   623   Misc       Cat Food        $    0.75 The stuff the dog eats.
  17   644   Misc       Socks           $    1.00 What both the dog and cat
                                                  destroy.
```

- **Multiline Fields**
  - The special @* picture field is used to output multiple lines of data from one scalar value
  - If used, it should appear on a picture line by itself

EXAMPLE

```
#!/usr/local/bin/perl -w
#
#
$multiline_text = <<EOS;  # Get some multiline data
```

```
This is the First Line
This is the Second Line
This is the Third line
This last line is really long so it hangs out here.
EOS

#    Output the data using the
#    default format (STDOUT)

write;

#
#
format STDOUT =
**************************************************
@*
$multiline_text
**************************************************
.
```

OUTPUT

```
**************************************************
This is the First Line
This is the Second Line
This is the Third line
This last line is really long so it hangs out here.
**************************************************
```

- **Filled Fields**
  - There are also picture fields that allow the creation of filled text blocks:

^<<<<	Left-justified text blocks
^>>>>	Right-justified text blocks
^||||	Center-justified text blocks

  - The value used with a filled field must be a scalar variable or an array element containing a string scalar. This is because Perl will modify the value of this variable while filling the field.
  - To create the filled text block, Perl breaks the value into words, puts as much text as possible into the first line of the text block, outputs this line, and then removes this data from the variable so that it can be used for the next line of the filled text block
  - So to create the filled text block, a sequence of picture fields on successive picture lines is used
  - The word-break characters default to space, newline and hyphen. These can be changed by setting the special $: variable.

EXAMPLES

```
format STDOUT =
************************************************
Right-justified text block:  ^>>>>>>>>>>>>>>>>>>>
$multiline_text
                             ^>>>>>>>>>>>>>>>>>>>
$multiline_text
                             ^>>>>>>>>>>>>>>>>>>>
$multiline_text
************************************************
.

# Get some multiline data

$multiline_text = <<EOS;
This is the First Line
This is the Second Line
This is the Third line
EOS

write;   # Output the report
```

OUTPUT

```
************************************************
Right-justified text block:      This is the First
                                  Line This is the
                                Second Line This is
************************************************
```

Note that this example creates at most three lines of filled text in a rectangle 20 characters wide on the right side of the output.

```
format STDOUT =
************************************************
Center-justified text block:
             ^||||||||||||||||||||
$multiline_text
             ^||||||||||||||||||||
$multiline_text
             ^||||||||||||||||||||
$multiline_text
************************************************
.

$multiline_text = <<EOS;  # Get some multiline data
This is the First Line
This is the Second Line
This is the Third line
EOS


write;                              # Output the report
```

OUTPUT

```
************************************************
```

```
Center-justified text block:
              This is the First Line
              This is the Second Line
              This is the Third line
************************************************
```

- **Blank Line Suppression**
  - To suppress the output of a blank line, put a tilde (~) anywhere in the picture line where a space would have been output
  - The tilde is translated to a blank on output
  - Blank line suppression is useful in filled fields. In the filled field example above, if the filled text is less than three lines, blank lines are output. These can be suppressed with the tilde indicator.

EXAMPLE

```
format STDOUT =

************************************************
Right-justified text block:
~                          ^>>>>>>>>>>>>>>>>>>>>>>>
$multiline_text
~                          ^>>>>>>>>>>>>>>>>>>>>>>>
$multiline_text
~                          ^>>>>>>>>>>>>>>>>>>>>>>>
$multiline_text
************************************************
.

$multiline_text = <<EOS;  # Get some multiline data
This is the First Line
This is the Second Line
EOS


write;                                  # Output the report
```

OUTPUT

Results shown without and with ~ to suppress blank lines.

```
************************************************
Center-justified text block:
                            This is the First Line
                           This is the Second Line

************************************************

************************************************
Center-justified text block:
                            This is the First Line
                           This is the Second Line
```

```
**************************************************
```

- **Repeated Picture Lines**
  - To repeat the output of picture line until a blank line would be output, put two consecutive tildes anywhere in the picture line where two spaces would have been output
  - The tildes are translated to blanks on output
  - Repeated picture lines are also very useful with filled fields. In the filled field example above, if the filled text is more than three lines, the text is truncated and lost. To fix this we use the double ~~.

EXAMPLE

```
$multiline_text = <<EOS;  # Get some multiline data
This is the First Line.
This is the Second Line.
This is the third Line.
This is the fourth Line.
EOS

write;

format STDOUT =
**************************************************
Right-justified text block:   ^>>>>>>>>>>>>>>>>>>>
$multiline_text
~~                            ^>>>>>>>>>>>>>>>>>>>
$multiline_text
**************************************************
.
```

OUTPUT

```
**************************************************
Right-justified text block:      This is the First
                                 Line. This is the
                               Second Line. This is
                               the third Line. This
                               is the fourth Line.
**************************************************
```

This is the usual way that filled text blocks are done in a format definition.

- **Top-Of-Page Formats**
  - The Top-Of-Page (TOP) format is output whenever a new page is started
  - A new page is started before the first output record and whenever there is insufficient space on the current page to output the next record

- The page output delimiter is a formfeed (NOTE: the ASCII code for this may be OS dependent.)
- The default number of lines per pages is 60 (typical portrait mode).
- Each filehandle has an associated TOP format. By default, the name of this TOP format is same as the name of the filehandle, but with "_TOP" appended. Thus, the TOP format for STDOUT is STDOUT_TOP. The TOP format of the currently selected filehandle (the current default filehandle) can be set by assigning a format name to the special variable $^.
- Each filehandle also keeps tracks of the current page number in the special variable $%.

• Changing Format Defaults

- EACH filehandle maintains the following values:

Format Name	(Default: Name of filehandle)
TOP Format Name	(Default: Name of filehandle with "_TOP" appended)
Page Length	(Default: 60 lines)
Current Page Number	(Initially 0)
Number Of Lines Left On The Page	

- To change any of these for a given filehandle you must first make that filehandle by the current default filehandle (CDF) using the select function and then set the appropriate special variable as follows:

$~	Format Name
$^	TOP Format Name
$=	Page Length
$%	Current Page Number
$-	Number Of Lines Left On The Page

- If the page length is changed, its new value will not be used until a new page is started
- When the $- variable becomes 0, a new page is started

EXAMPLE

Let's improve our hardware database report by adding a Top-Of-Page format, and page number. Since the default page size would normally hold all the information, we'll also reset this to a smaller page size for purposes of illustration.

```
#!/usr/local/bin/perl -w
#
#----------------------------------------------------------------#
#       Filename:       hardware_rpt.pl
#
#   VERSION: 2.0
#
#       Algorithm Description:
#               read the hardware.dat data file and printout
#               a formated report.
#
#       Input:
#               reads from file: hardware.dat
#       this file is a database with fields
#               delimited by |.
#
#       Output:
#               prints report to STDOUT
#
#       Restrictions:
#               None
#
#       Programmer:
#               s. gasster
#
#       Revision History:
#               original code from "Perl 5 How-To," by Glover, et al.
#               modified to open/close data file        Oct 5, 1996
#
#----------------------------------------------------------------#
#
#
    use English;

# Set the page to be 20 lines long.
# (This is only for demonstration purposes)
$FORMAT_LINES_PER_PAGE = 20;


$InFile = "hardware.dat";   # input file name

#
#       open data file
#

open(DATA, "$InFile") or
    die ("Problem opening $InFile: $!");

#    read a line from the file until
#    there are no more records.

while ($line = <DATA>){
    chomp($line);

    # Split the input line on the | symbol.
    ($number,$type,$name,$price,$desc) = split (/\|/, $line);

    # Print out the item information.
    write;
}
```

```
close (DATA);
#
#    format declarations are here
#
format STDOUT_TOP=
Count Item #  Item Type  Item Name      Price     Description   Page @>>>>>>
$FORMAT_PAGE_NUMBER
================================================================================
.

format STDOUT=
@>>>> @|||||| @<<<<<<<<< @<<<<<<<<<<<< $@####.## ^<<<<<<<<<<<<<<<<<<<<<<<<<
$.,$number,$type,$name,$price,$desc
~                                                 ^<<<<<<<<<<<<<<<<<<<<<<<<<
$desc
~                                                 ^<<<<<<<<<<<<<<<<<<<<<<<<<
$desc
.

exit (1);
```

OUTPUT

```
Count Item #  Item Type  Item Name      Price     Description   Page       1
================================================================================
    1   101   Hardware   Hammer         $   25.00 A thing to hit nails with.
    2   121   Hardware   Nail           $    0.15 A thing to be hit by a
                                                  hammer. (see hammer)
    3   142   Hardware   Sander         $   10.15 Used to sand all sorts of
                                                  things, except sand.
    4   206   Household  Kitchen Sink   $  100.00 Lost the last time I went
                                                  on vacation. (next time
                                                  this I won't check it in)
    5   210   Household  Windows        $   45.00 User friendly and does not
                                                  need to be plugged in.
    6   242   Household  Vacuum         $  300.00 I'm not sure what this is
                                                  used for; but people seem
                                                  to have them anyway.
    7   266   Household  Microwave      $  100.00 Kitchen item. (Also see
                                                  hot water heater)
    8   312   Garden     Rake           $   25.00 Used to rake leaves, cut
                                                  grass and break noses if
                                                  left lying on the ground.


Count Item #  Item Type  Item Name      Price     Description   Page       2
================================================================================
    9   344   Garden     Gravel         $   30.00 Why bother the dog is
                                                  going to dig through it
                                                  again anyway.
   10   362   Garden     Top Soil       $   10.00 Is there such a thing as
                                                  "Bottom Soil"?
   11   384   Garden     Mosquitos      $    0.00 Some are large as a dog
                                                  and will carry you away.
   12   500   Sports     Hockey Stick   $   10.00 Used to push the puck
                                                  around.
   13   501   Sports     Bike           $  400.00 Sits in the garage and
                                                  collects dust.
   14   556   Sports     First Aid Kit  $   25.00 Used more times than not
```

```
                                                    unfortunately.
15    601    Misc        Dog Food        $    1.50 The stuff the cat eats.
16    623    Misc        Cat Food        $    0.75 The stuff the dog eats.
17    644    Misc        Socks           $    1.00 What both the dog and cat
                                                    destroy.
```

- **Mixing Print (Or Printf) With Write**
  - Whenever you output data to a filehandle using the write function (and thus using the filehandle's format), Perl keeps track of the current page number and line number within that page
  - If you also output data to the filehandle using the print or printf functions, Perl does **NOT** update the internal line or page counts
  - So if you want to mix these two types of output, you must update `$-` and `$%` yourself whenever you output via `print()` or `printf()`

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 4]

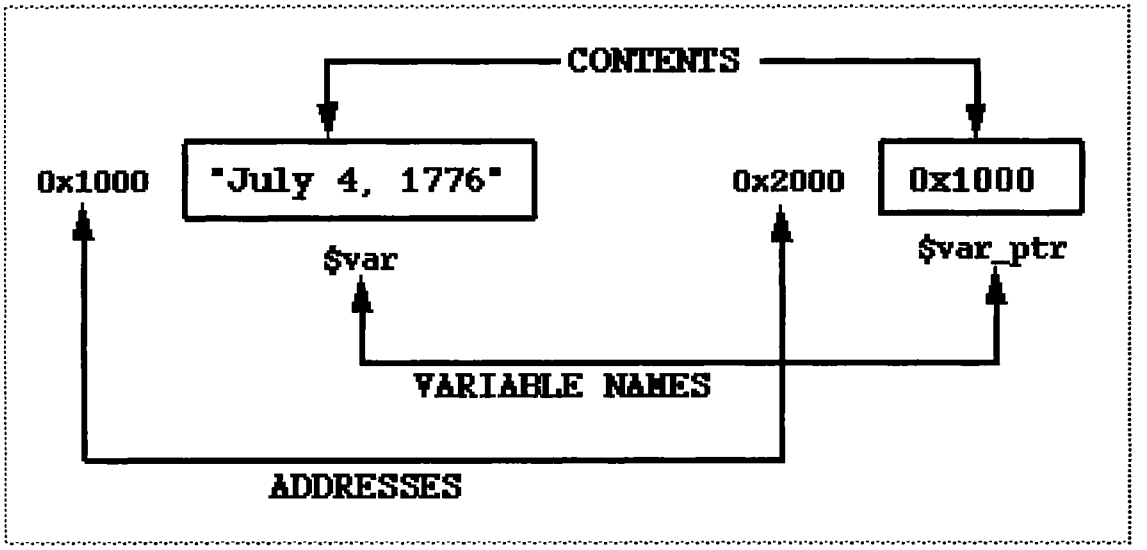
© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 4: PERL REFERENCES

INTRODUCTION

- **What is a reference?**
  - A **reference** is an entity that can point to any given data type in Perl. Those familiar with C programming can view a reference as a **pointer**.
  - A pointer contains the address of the entity (variable) pointed to.

Figure 1. Pointers contain the address of the object they point to.



- The use of references can get fairly complicated. To keep things simple, and hopefully understandable, we will only discuss limited applications of references. In any case we don't need to necessarily use some of the more subtle, complicated and confusing aspects to perform our ultimate goal...CGI! (remember that?).
  For more advanced usage, read the man pages and Perl books.

- **Why are references useful?**
  - Allows passing arrays into subroutines (we'll discuss this and give examples in Lecture 5)
  - Allows creation of complex data structures in Perl (recall Perl's inherent arrays are really only 1D (vectors).

- **In this lecture we will discuss the following references (pointers):**
  - references to scalar, array and hash variables
  - references to anonymous arrays, hashes and subroutines
  - multidimensional data arrays and structures

- **The following are left to later lectures:**
  - references to subroutines (code references)
  - references to filehandles

CREATING A REFERENCE AND DEREFERENCING

- One basic method of creating a reference to a variable, subroutine or value is to use the backslash (\) operator.

- **Pointer to a scalar**

```
     #   initialize variable
$var = "July 4, 1776";

     #   create pointer (reference) to $var
$var_ptr = \$var;

     #  dereference the pointer to $var
print "Our independence day is $$var_ptr\n";

     #  address contained in $var_ptr
print "\$var lives at: $var_ptr\n";
```

OUTPUT

```
Our independence day is July 4, 1776
$var lives at: SCALAR(0x4d84aec)
```

Note that the second print statement prints out a weird looking result. It is just the address where Perl (MacPerl in this case!) has chosen to start storing the string contained in `$var`. Also note that Perl has enclosed this address with a label `SCALAR()` to signify that it is a reference to a scalar variable.

To **dereference** the reference, we used a `$`. Thus

- `$var` refers to the scalar variable containing `"July 4, 1776"`
- `$var_ptr` is a reference (a scalar variable) which holds the address of `$var`, i.e., it points to `$var`
- `$$var_ptr` is a scalar value which can be interpreted as the contents of the location referenced by `$var_ptr`.

- **Pointer to an array**

```
    # create array of pets
@pets = qw(dog cats fish hamster bird);

    # create pointer to @pets
$pets_ptr = \@pets;

    # look inside!
print "\n@pets\n";
print "Pointer to Pets: $pets_ptr\n";
```

```
    # deference array pointer
foreach (@$pets_ptr){
        print "A $_ makes a good pet if you clean up after it!\n";
}
```

OUTPUT

```
dog cats fish hamster bird
Pointer to Pets: ARRAY(0x4d86aac)
A dog makes a good pet if you clean up after it!
A cats makes a good pet if you clean up after it!
A fish makes a good pet if you clean up after it!
A hamster makes a good pet if you clean up after it!
A bird makes a good pet if you clean up after it!
```

In this case we use `@` to dereference the whole array. If you want to deference a specific element of the array use `$$pets_ptr[$index]`. Note that Perl now indicates that the address contained in the pointer belongs to an `ARRAY()`.

- **Pointer to an associative array (hash)**

```
    # declare hash
%webhits = ( qq(www.muon.com)=>10,
             qq(www.nasa.gov)=>15,
             qq(tank.army.mil)=>25 );

    # create pointer to hash
$webhits_ptr = \%webhits;


foreach $key (keys %webhits){
        print "$key = $webhits{$key}\n";
        print "$key = $$webhits_ptr{$key}\n";
}

    # or

foreach $key (keys %$webhits_ptr){
        print "$key = $webhits{$key}\n";
        print "$key = $$webhits_ptr{$key}\n";
}

print "Pointer to webhits: $webhits_ptr\n";
```

OUTPUT

```
www.nasa.gov = 15
www.nasa.gov = 15
tank.army.mil = 25
tank.army.mil = 25
www.muon.com = 10
www.muon.com = 10
www.nasa.gov = 15
www.nasa.gov = 15
tank.army.mil = 25
tank.army.mil = 25
```

```
www.muon.com = 10
www.muon.com = 10
Pointer to webhits: HASH(0x4d87abc)
```

Again you can deference the complete hash pointer using `%$webhits_ptr` or deference individual elements using `$$webhits_ptr{$key}`.

- **Other methods to deference a pointer to an array or hash**

Note that the following print statements all give the same result (they print fish):

```
@pets = qw(dog cat fish hamster bird);
$pets_ptr = \@pets;

print "$$pets_ptr[2]\n";
print "${$pets_ptr}[2]\n";
print "$pets_ptr->[2]\n";
```

Similarly, the following print statements all give the same result (they print 15):

```
%webhits = ( qq(www.muon.com)=>10,
             qq(www.nasa.gov)=>15,
             qq(tank.army.mil)=>25 );

$webhits_ptr = \%webhits;

$key = qq(www.nasa.gov);

print "$$webhits_ptr{$key}\n";
print "${$webhits_ptr}{$key}\n";
print "$webhits_ptr->{$key}\n";
```

- **Pointer to a subroutine or function**
  This can be a little tricky, and we'll wait until Lecture 5, where we discuss subroutines to discuss this. You can read ahead if you like.

THE ref() FUNCTION

`ref(EXPR)`

Returns a `TRUE` value (i.e., non-zero) if `EXPR` is a reference, `FALSE` otherwise. The value returned depends on the type of thing the reference is a reference to. Builtin types include:

```
REF
SCALAR
ARRAY
HASH
CODE
GLOB
```

```
@space = qw(planet solarsys galaxy universe);
$space_ptr = \@space;
$ref_type = ref($space_ptr);
```

```
print "\$space_ptr is a reference of type: $ref_type\n";
```

OUTPUT

```
$space_ptr is a reference of type: ARRAY
```

ANONYMOUS ARRAY, HASH AND SUBROUTINE REFERENCES

So far the references we discussed point to the address associated with a named variable or subroutine that has been defined within the Perl namespace. If we `undef($var)`, the variable `$var` and its *value* go away (are deleted from Perl's namespace), however, the pointer `$var_ptr` still points to the same location! Be careful not to dereference this pointer from then on. If you are lucky nothing bad will happen, you'll just get a null value back, but don't depend on that! **Don't use a pointer that points to nothing!**

```
$var = 10;
$var_ptr = \$var;
print "\$var = $var\n";
print "Pointer = $var_ptr\n";

# bye bye
        undef($var);
print "\$var = $var\n";
print "Pointer = $var_ptr\n";
print "Here goes nothing...$$var_ptr\n";
```

OUTPUT

```
$var = 10
Pointer = SCALAR(0x4cfb07c)
$var =
Pointer = SCALAR(0x4cfb07c)
Here goes nothing...
```

It is possible in Perl 5 (kids don't try this with Perl 4) to create an **anonymous reference**, that is a pointer that points to a variable or subroutine that does not have a predefined name. This is handy in that we can create **multidimensional data structures**, the so called Lists of Lists, Hash of Lists, List of Hashes and Hash of Hashes (whewww!!). Let's look at some examples.

- **Anonymous Array References**
  This is an array without a name. Perl stores the array into the reference, and creates the namespace for the array. To create an anonymous array use square brackets around a list.

```
@pets = qw(dog cat fish hamster);
$pets_ref = [@pets];
print "Anonymous array reference: \$pets_ref = $pets_ref\n";
printf "List values: \@pets = @$pets_ref\n";
print "Deference array element 2: $pets_ref->[2]\n";
```

OUTPUT

```
Anonymous array reference: $test = ARRAY(0x4cfc65c)
List values: @test = dog cat fish hamster
Deference array element 2: fish
```

- **Anonymous Hash or Associative Array References**
  This is an associative array without a name. Perl stores the hash into the reference, and creates the namespace for the hash. To create an anonymous hash use curly brackets around a set of `key => value` assignments.

```
    # create the anonymous hash
$form_input = {  'UserName'   => 'Joe Hacker',
                 'UserEmail'  => 'joe@comp.world.com',
                 'ModemSpeed' => 'ISDN'};

print "Anonymous hash reference: \$form_input = $form_input\n";

foreach $key ( keys %{$form_input} ) {
        print "$key = $form_input->{$key}\n";
}
```

OUTPUT

```
Anonymous hash reference: $form_input = HASH(0x4d835b8)
UserName = Joe Hacker
ModemSpeed = ISDN
UserEmail = joe@comp.world.com
```

- **Anonymous Subroutine References**
  This can be a little tricky, and we'll wait until Lecture 5, where we discuss subroutines to discuss this. You can read ahead if you like.

MULTIDIMENSIONAL ARRAYS AND HASHES

Recall that the builtin arrays in Perl are only one dimensional (1D, vectors). This can be a real pain, especially when the data we are dealing with is more naturally handled with a higher dimensional data structure such as a matrix (2D, 3D or higher).

The most important thing to understand about all data structures in Perl -- including multidimensional arrays -- is that even though they might appear otherwise, Perl `@ARRAYs` and `%HASHes` are all internally one-dimensional. They can only hold scalar values (meaning a string, number, or a reference). They cannot directly contain other arrays or hashes, but instead contain references to other arrays or hashes.

You can't use a reference to an array or hash in quite the same way that you would a real array or hash. Since references are rather like pointers that know what they point to, that means that when you have something that looks to you like an access to two-or-more-dimensional array and/or hash, that what's really going on is that in all these cases, **the base type is merely a one-dimensional entity** that contains references to the next level. It's just that you can use it as though it were a two-dimensional one.

Now, because the top level only contains references, if you try to print out your array in with a simple `print()` function, you'll get something that doesn't look very nice, like this:

```
@LoL = ( [2, 3], [4, 5, 7], [0] );
print $LoL[1][2];
  7
print @LoL;
  ARRAY(0x83c38)ARRAY(0x8b194)ARRAY(0x8b1d0)
```

That's because Perl doesn't (ever) implicitly dereference your variables. If you want to get at the thing a reference is referring to, then you have to **explicitly deference the pointer** yourself using either prefix typing indicators, like

```
${$blah}
@{$blah}
@{$blah[$i]}
```

or else postfix pointer arrows, like

```
$a->[3],
$h->{fred}
```

Here are some examples:

- **List of Lists: 2D array (matrix)**

  Example 1: An anonymous 2D array (List of Lists)

```
#!/usr/local/bin/perl -w

my $pi = 4.*atan2(1.,1.);
my $d2r = $pi/180.;

print "Enter angle (deg.)> ";
my $angle = <STDIN>;

chomp($angle);
my $theta = $angle * $d2r;

my $m2d_ref = [ [cos($theta), sin($theta)],
                [-sin($theta), cos($theta)] ];

printf "M00 = %6.4f\n", $m2d_ref->[0][0];
printf "M01 = %6.4f\n", $m2d_ref->[0][1];
printf "M10 = %6.4f\n", $m2d_ref->[1][0];
printf "M11 = %6.4f\n", $m2d_ref->[1][1];
exit (1);
```

  OUTPUT

```
Enter angle (deg.)> -30.
M00 = 0.8660
M01 = -0.5000
M10 = 0.5000
M11 = 0.8660
```

Example 2: User Names Array

```
@UserNames = (
        [ "fred", "barney" ],
        [ "george", "jane", "elroy" ],
        [ "homer", "marge", "bart" ],
      );

    # access a single element
print "$UserNames[1][2]\n";

# print the whole array, one element at a time

foreach $i (0..$#UserNames){
  foreach $j (0..$#{$UserNames[$i]} ) {
    print "Element $i $j is $UserNames[$i][$j]\n";
  }
}
```

OUTPUT

```
elroy
Element 0 0 is fred
Element 0 1 is barney
Element 1 0 is george
Element 1 1 is jane
Element 1 2 is elroy
Element 2 0 is homer
Element 2 1 is marge
Element 2 2 is bart
```

Note on the inner loop (`$j`) how the range for the loop is determined by the size of the sub-array.

- **Hash of Lists**

Example

In this example a hash is created with the keys being the days of the week. The values are lists of files which are to be served on a given web site based on the day of the week. Only the file name is required since they will all have the extension `.html`, which can be appended later.

Note the use of the time and localtime functions to get the day of the week to use as a key for the hash. This script only prints out the file names, a more complicated script would use them as part of a CGI application!

```
#!/usr/local/bin/perl
#
#       file: files_by_day.pl
#

    # create named hash of lists
%files_by_day = (
   "Mon" => ["Mon01", "Mon02", "Mon03"],
   "Tue" => ["Tue01", "Tue02"],
```

```
    "Wed" => ["Wed01", "Wed02", "Wed03"],
    "Thu" => ["Thu01", "Thu02", "Thu03"],
    "Fri" => ["Fri01", "Fri02", "Fri03"],
    "Sat" => ["Sat01", "Sat02", "Sat03", "Sat04"],
    "Sun" => ["Sun01", "Sun02"]
);

@days = qw(Mon Tue Wed Thu Fri Sat Sun);

    # get week day
($sec,$min,$hour,$mday,$mon,$year,
      $wday,$yday,$isdst) = localtime(time);

    #use $wday, an integer corresponding to the DOW
$day_of_week = $days[$wday];
print "Today is $day_of_week\n";

    # extract list from hash
@file_list = @{$files_by_day{$day_of_week}};

    # print list of files
foreach $file (@file_list){
   print "File: $file".".html\n";
}
```

OUTPUT

```
Today is Thu
File: Thu01.html
File: Thu02.html
File: Thu03.html
```

If I wanted to change the content of my web site each week, I could write a Perl script using this type of file selector, and up-load my files once each week with the appropriate names. Rather than selecting the individual files, you could imagine selecting a different directory based on the day of the week and loading all of the files in that directory for your web site, on that specific day.

And so on, you get the idea...see the perl man pages (perldsc) for more punishment!

STRUCTURES

structure

> *A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling.*
>
> -- Kernighan and Ritchie, The C Programming Language.

Hey! We can do that in Perl! Given what we know about hashes and references this should be easy! Let's look at an example.

Example: Book Catalog

```
#create book DB structure
```

```
%Books = ( ISBN1 => {author       => 'David Till',
                     title        => 'Teach Yourself Perl 5',
                     publisher    => 'Sams.net',
                     year         =>  1996,
                     edition      =>  2,
                     price        =>  39.99,
                     keywords     => [ 'Perl 5','programming','computers'] },

           ISBN2 => {author       => 'Shishir Gundavaram',
                     title        => 'CGI Programming on the World Wide Web',
                     publisher    => 'OReilly & Associates, Inc.',
                     year         =>  1996,
                     edition      =>  1,
                     price        =>  29.95,
                     keywords     => [ 'CGI','programming','WWW'] }
         );


# print out book information

foreach $key1 (keys %Books) {
    print "\n". ( "-" x 20 ) . "\n";
    @the_keys = keys %{$Books{$key1}};
    print "The keys are: @the_keys\n";

    foreach $thekey (keys %{$Books{$key1}}) {
        if ($thekey eq 'keywords') {
            print "The key words are: \n";
            foreach $kw (@{$Books{$key1}->{$thekey}}) {
                print "\t$kw\n";
            }
        } else {
                print "The $thekey is $Books{$key1}->{$thekey}\n";
        }
    }
}
```

OUTPUT

```
--------------------
The keys are: keywords year title price author edition publisher
The key words are:
        Perl 5
        programming
        computers
The year is 1996
The title is Teach Yourself Perl 5
The price is 39.99
The author is David Till
The edition is 2
The publisher is Sams.net

--------------------
The keys are: keywords year title price author edition publisher
The key words are:
        CGI
        programming
        WWW
The year is 1996
```

```
The title is CGI Programming on the World Wide Web
The price is 29.95
The author is Shishir Gundavaram
The edition is 1
The publisher is OReilly & Associates, Inc.
```

REFERENCES TO FILEHANDLES

Yes, you can create a reference to a filehandle. One of the key uses for such a construct is to pass a filehandle to subroutine. Therefore, the discussion of this topic is deferred to Lecture 5.

NOTE

I do not like *Day 18* in *TYPERL5*! The discussion is hard to follow, there are several errors and in general its unclear! The `perlref` man page is not much better! But read them anyway. ;-)

Also, don't panic, since it is not essential that one use references to create CGI applications.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 4]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 4

NOTE

Unless I specifically provide you with data to test your programs, you should test each of your programs with test data that you create. Try to use enough values so that the input data will exercise all of the options/capabilities of your code. This isn't supposed to be exhaustive, I just want to see that your code works. Don't forget to include any test data with your solution so I can verify that your code processed the data correctly. Thanks.

---

1. **Report Writing Problem**

   Using the file `address.dat` that we created in Assignment 3, write a Perl script to prompt the user (menu) and perform the following functions using that file as input:

   A. Display on your terminal only the names in the file, in alphabetical order (use last name).

   B. Print the complete file to a new file (specified by the user) formatted as a table, with the following headings:

   ```
   Name        Street                  City            State          Zip
   ```

   This file is suitable for printing, so print it (I'd like to see it). But this is not a menu option.

   C. Given a user specified zip code, print to the screen all records with that zip code in the format as shown above (no colons!). If the users enter a zip code not found in the file, handle this error in a graceful manner.

   D. Exit the script

2. **Report Writing Problem**

   Write a Perl script that will read the file `congress.db` (see below) and print out the following information in a nice tabular format:

   ```
   NAME    STATE  PARTY    ADDRESS   PHONE
   ```

Your program should check the file to see if it exists and that you have read access. The NAME, ADDRESS and PHONE fields should be left justified in their columns. The STATE and PARTY fields should be centered in their columns. Assume the page size is 5 lines long and print the following header at the top (center) of each page:

```
********** CONGRESSIONAL DATA **********   PAGE X
```

where X is of course the page number.

For extra credit try sorting the listing by state and party.

congress.db file:

```
Christopher S. Bond:MO:Republican:293 Russell Senate Office Building:202/224-5721
John Ashcroft:MO:Republican:705 Hart Senate Office Building:202/224-6154
William Clay:MO:Democrat:2306 Rayburn House Office Building:202/225-2406
Vic Fazio:CA:Democrat:2113 Rayburn House Office Building:202/225-5716
Richard Gephardt:MO:Democrat:1226 Longworth House Office Building:202/225-2671
Ike Skelton:MO:Democrat:2227 Rayburn House Office Building:202/225-2876
Karen McCarthy:MO:Democrat:1232 Longworth House Office Building:202/225-4535
Pat Danner:MO:Democrat:1323 Longworth House Office Building:202/225-7041
Mel Hancock:MO:Republican:438 Cannon House Office Building:202/225-6536
Bill Emerson:MO:Republican:2268 Rayburn House Office Building:202/225-3121
Harold Volkmer:MO:Democrat:2409 Rayburn House Office Building:202/225-2956
Dianne Feinstein:CA:Democrat:331 Hart Senate Office Building:202/224-3841
Barbara Boxer:CA:Democrat:112 Hart Office Building:202/224-3553
Frank Riggs:CA:Republican:1714 Longworth House Office Building:202/225-3311
Wally Herger:CA:Republican:2433 Rayburn House Office Building:202/225-3076
John Doolittle:CA:Republican:1526 Longworth House Office Building:202/225-2511
Robert Matsui:CA:Democrat:2311 Rayburn House Office Building:202/225-7163
Lynn Woolsey:CA:Democrat:439 Cannon House Office Building:202/225-5161
George Miller:CA:Democrat:2205 Rayburn House Office Building:202/225-2095
Nancy Pelosi:CA:Democrat:2457 Rayburn House Office Building:202/225-4965
James Talent:MO:Republican:1022 Longworth House Office Building:202/225-2561
```

3. **File Access Problem**

Write and test a Perl script that will perform the following functions:

- Accept a list of file names from the user. These can be either fully qualified file names (path + filename) or just directory names. If the user gives you a directory name then get all the text files from each directory to operate on. If the user gives you full file names then operate only on those specific files. Your script should be portable so don't rely on any DOS/Windows file name length limitations.

- Strip all lines that start (column 1) with # from the files
- Makes a backup copy of each original file, before striping, with `.bck` appended to the filename
- Overwrites the original file after striping the # lines
- Test to make sure that each file exists, it is a text file, and that you have the correct access to perform these operations
- Perform error handling and have a graceful exit if failure.

4. PRIMORDIAL GUESTBOOK OR USER FEEDBACK

This is a precursor to one of the more popular WWW CGI applications: The Guestbook. We'll work on this program several times during the course and eventually have a CGI application that should work via the web.

Write and test a Perl script that does the following:

i. Load the following information that the user enters via prompts when the program is run (not via HTML forms...that comes later) and store it in an associative array:
   - Users full name (don't accept blanks here, they must enter something)
   - Users email address (don't accept blanks here, they must enter something)
   - City, State and Country
   - User comments or feedback - limit this to 2000 bytes maximum - tell them this and if they goof, have them try again.
   - Date and Time of user entry

ii. Save this data, wrapped in HTML, to a text file. It should be readable with a browser. Your program should:
   - check if file exists, if yes add the data, if not create a new file.
   - Additions to existing file should be added with the most recent first. Therefore when someone views this page, the most recent additions are at the top.

iii. A sample page might look like:

Welcome to the Primordial Guestbook

Bill Clinton <president@whitehouse.gov>
Washington DC, USA
Fri Sep 13 10:55:50 PDT 1996

Sam: I am very impressed with your website. If re-elected I will invite you to join me in Washington as the new Minister of Perl and CGI programming. So vote for me. -Bill

---

Joe Quackenbush <jquack@xyz.anydomain.com>
Anytown, CA, USA
Wed Sep 11 21:55:50 PDT 1996

I really like your website. The graphics are great, the information is great and your grandfather is great. The guestbook was the most fun. I hope I am the winner of the $1,000,000. prize! ;-) Joe.

Primordial Guestbook
© Yournamehere 1996

---

5. **IRS Audit** (oh bummer!)
You are going to write a Perl program that will decide whether to audit a given tax return (if you want to put in a hack so you never get audited I'll over look that). The tax payer database is a colon (:) separated database with the following information, one record per line:

```
last name:first name:gross income:taxable income:deduction_1: ...:deduction_N
```

where each deduction is specified by a letter (a..t) and N has a maximum value of 20. Example entry:

```
Ashcroft:John:150000:35000:a:b:c:d:e:f:g:h:i:j
```

To determine if the tax payer should be audited a score is assigned to each tax payer. The score is computed based on the following rules:

- Initial score = 0
- If taxable income < half gross income, add 20 points to score
- If number of deductions > 5, add 5 points for every deduction over 5
- If the first letter of their taxpayer's last name has an ordinal value in the alphabet which is an even number, add 40 points (e.g., A=1 (odd), B=2 (even), ...,etc.)
- If they try to cheat by leaving all fields but their name blank, they get an automatic audit

To determine if the tax payer gets audited, total their score. Then compute a random number between 0 and 100. If their score is greater than

this random number, an audit occurs. Print out a list of each tax payer's name followed by either `AUDIT` or `NO AUDIT`.

If you want to have some fun, try using names from the above database: congress.db! ;-)

[TITLE PAGE | TABLE OF CONTENTS | HOMEWORK | LECTURE 4]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 5: Perl SUBROUTINES, FUNCTIONS AND LIBRARIES

READING ASSIGNMENT:
TYPERL 5: Ch. 9, 14, Ch 20 (pp. 716-719)

HOMEWORK ASSIGNMENT:
Assignment No. 5 (Due 10/28).

Perl MANPAGES:

- perlsyn
- perlfunc
- perldata
- perlop
- perlstyle

TOPICS

- Subroutine and function syntax
- Built-in Perl functions: Part II
- Perl libraries

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 6]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 5:
Perl SUBROUTINES AND FUNCTIONS

INTRODUCTION

- **What is a Subroutine or Function?**

  A subroutine is a separate piece of code that performs a specific task or function. The subroutine is called or invoked by the *main* program or package, or calling program or package (in the case of a subroutine called from within a subroutine).

- **Why do we use them?**

  - **Modularity**
    Breaking up the problem you are trying to solve into smaller pieces makes debugging and managing code easier, since each task is smaller and clearly defined (hopefully).

  - **Readability**
    The overall problem can be understood in terms of several well defined tasks and functions. Combining these together to achieve the overall goal is then fairly straight forward. The main part of the program may then consist of some parameter/variable initialization and calls to functions. It should then be easy to understand the program from a top-down perspective.

  - **Reusability**
    Often we find that it is necessary to perform a given task many times. Rather than repeat that block of code each time it makes sense (for the reasons just cited) to make that block a subroutine or function.

SUBROUTINE SYNTAX

The following syntax provides Perl with the definition of the subroutine `subname` (and the declaration if it has not been declared yet):

```
sub subname {

   ...statement block...
}
```

`subname`
: name of the subroutine or function, consists of an alphabetic character followed by one or more letters, digits or underscores.

`statement block`
: body of the subroutine which includes one or more statements

Warning: Don't use subroutine names that Perl already uses! Both you and Perl can get confused!

Subroutine definitions can appear anywhere in the main program. However, it is much easier to read your code

if they are grouped together, generally at the end of the program (within the same file). It is also possible to have your subroutine definitions in one or more additional files which are then loaded into the current file. We'll talk about that later in this lecture when we discuss libraries. In this course I'd either like to see all your subroutines defined either at the end of the main program (in the same file) or loaded in from separate files (your library).

EXAMPLE

```
#!/usr/local/bin/perl -w
#
#    example of program layout
#
#    main body of program....
#

&printHello;  #subroutine with no arguments


#
# subroutine/function definitions here
#

sub printHello {

    print "Hello\n";

}

__END__
```

OUTPUT

```
Hello
```

CALLING A SUBROUTINE OR FUNCTION

- To reference or call a subroutine use:

```
&print_values(...);

$result = &getUserInput;
```

- In Perl 5, if subroutines have already been defined you can leave off the &

```
sub print_values {   # define subroutine here

.....stuff....
}


print_values(...);   # call subroutine here
```

The Perl interpreter already knows that `print_values` is a subroutine since it has been defined. If you want to leave off the & and put all subroutine definitions at the end of your program then supply the forward reference declaration:

```
sub print_values;     # declare subroutine here

print_values(...);    # call subroutine here


sub print_values {    # define subroutine here

.....stuff....
}
```

This can sometimes confuse the Perl interpreter, so use with caution. If problems arise try putting back the & and seeing if things start working.

RETURN VALUES

- By default, Perl automatically considers the value of the last statement evaluated within a subroutine as the return value. As an example lets look at our `Hello` program.

```
#!/usr/local/bin/perl -w
#
#    example of program layout
#
#    main body of program....
#

$result = &printHello;
print "The return value is: $result\n";

#
# subroutine/function definitions here
#

sub printHello {

    print "Hello\n";

}
```

OUTPUT

```
Hello
The return value is: 1
```

Why did it return a value of 1?

- Subroutine return values can be used anywhere a Perl expression is valid.

EXAMPLE

In this example the return value from the function `intrand` is used to index an array.

```
#!/usr/local/bin/perl
#
#    modified version of Program 9.4
```

```
#    TYPERL 5 by David Till
#
#
$LOOP_MAX = 100;

srand();   #init random number seed

print ("Random number tester.\n");

#    create an array which keeps a count
#    of the frequency of occurence of the
#    random values [0, $max_val)
#
for ($count = 1; $count <= $LOOP_MAX; $count++) {
        $randnum[&intrand] += 1;
}

print ("Totals for the digits 0 through 9:\n");
print ("@randnum\n");


#
#  function to return a random integer
#  in the range [0,$max_val)
#
sub intrand {
    my($max_val) = 10;
    $num = int(rand($max_val));
}
__END__


OUTPUT

Random number tester.
Totals for the digits 0 through 9:
8 11 3 13 13 10 16 10 9 7
```

• Perl's RETURN STATEMENT

Another way to explicitly declare the return value from a subroutine is to use the `return` statement:

```
    return (retval);
```

`retval` = desired return value; either a scalar or list value; can be the result of an expression. Use of the return statement can also make a complex function easier to understand, since you can explicitly see what value(s) are being returned. You can also have more than one return point from a subroutine or function.

EXAMPLE

From the example above of the `intrand` function:

```
#
#  function to return a random integer
#  in the range [0,$max_val)
```

```
#
sub intrand {
    my($max_val) = 10;
    $num = int(rand($max_val));

    return ($num);
}
```

- **CAUTION:** If the last expression executed in a subroutine is a conditional expression you might not get the return value you expect. I recommend always explicitly returning a value using the `return` statement.

LOCALIZATION

- Variable localization provides
    - Control over where and how long variables survive
    - Allows for information hiding -- data or variables not required to be visible to other program elements are kept private
    - Provides protection -- if you or someone else use the same variable name in another program element, they will not conflict
- Variables created by Perl are put into the global namespace by default (actually they are put into the namespace of the default package, which is `main`).
- Perl uses two keywords for localization:

```
my      (Perl 5 only)
local   (Perl 4 and Perl 5)
```

my
: creates private variables; declares the list of variables to be confined to the enclosing block, subroutine or `eval()`.
`my` variables are *lexically scoped* ; they are totally hidden and not visible to functions called from within the enclosing block.

local
: creates semi-private values of global variables; declares values of the list of variables to be confined to the enclosing block, subroutine or `eval()`.
`local` variables are *dynamically scoped* ; dynamic variables are also visible to functions called from within the enclosing block.

In either case, the variable (`my`) or `local` value disappears when the enclosing block, etc, is exited.

my is preferred over `local` (see the *Camel* book), so we'll use it in this course.

EXAMPLES

```
#   all variables are global

$max_val = 10;
```

```
print "The counter max is: $max_val\n";

&counter;

  for ($i=1; $i<=$max_val ; $i++) {
    print "The outside count is: $i\n";
  }

sub counter {
  for ($i=1; $i<=$max_val ; $i++) {
    print "The inside count is: $i\n";
  }
}
```

OUTPUT

```
The counter max is: 10
The inside count is: 1
The inside count is: 2
The inside count is: 3
The inside count is: 4
The inside count is: 5
The inside count is: 6
The inside count is: 7
The inside count is: 8
The inside count is: 9
The inside count is: 10
The outside count is: 1
The outside count is: 2
The outside count is: 3
The outside count is: 4
The outside count is: 5
The outside count is: 6
The outside count is: 7
The outside count is: 8
The outside count is: 9
The outside count is: 10
```

We can localize the variable `$max_val` in both the `main` program (package) and in the subroutine `counter`. If we localize it in main without declaring a value in the subroutine `counter`, then it won't have a value in `counter`. We can also localize it in the subroutine.

```
my $max_val = 10;     #localize to main

print "The counter max is: $max_val\n";

&counter;

  for ($i=1; $i<=$max_val ; $i++) {
    print "The outside count is: $i\n";
  }

sub counter {
  for ($i=1; $i<=$max_val ; $i++) {     #$max_val not local to sub
    print "The inside count is: $i\n";
  }
}
```

```
OUTPUT

The counter max is: 10
The outside count is: 1
The outside count is: 2
The outside count is: 3
The outside count is: 4
The outside count is: 5
The outside count is: 6
The outside count is: 7
The outside count is: 8
The outside count is: 9
The outside count is: 10


$max_val = 10;    #global to main

print "The counter max is: $max_val\n";

&counter;

  for ($i=1; $i<=$max_val ; $i++) {
    print "The outside count is: $i\n";
  }

sub counter {
  my $max_val = 5           #$max_val local to sub
  for ($i=1; $i<=$max_val ; $i++) {
    print "The inside count is: $i\n";
  }
}

OUTPUT

The counter max is: 10
The inside count is: 1
The inside count is: 2
The inside count is: 3
The inside count is: 4
The inside count is: 5
The outside count is: 1
The outside count is: 2
The outside count is: 3
The outside count is: 4
The outside count is: 5
The outside count is: 6
The outside count is: 7
The outside count is: 8
The outside count is: 9
The outside count is: 10
```

CAUTION!

Consider the following Perl statements (line numbers shown only for reference):

```
1    my $line = <STDIN>;
```

```
2     my($line) = <STDIN>;

3     my ($line) = <STDIN>;

4     my($line) = scalar(<STDIN>);
```

It is intended that each of these *read a single line of input* from `<STDIN>` into the scalar variable `$line`. However, this is not what happens in all cases. Lines 1 and 4 will do this, since in Line 1 `my` is localizing the scalar `$line` which is forcing `<STDIN>` into a scalar context; Line 4 we've explicitly forced the scalar context. In Lines 2 and 3 we've enclosed `$line` in `()`, which creates a **LIST** context, which means that the read from `<STDIN>` will now be in a list context, reading multiple lines; **NOT** what we wanted.

PASSING VALUES TO A SUBROUTINE

- Subroutines can have values passed into them from the main or calling program. These values are called **ARGUMENTS**.
- Perl uses the system array variable `@_` to pass arguments to a subroutine. It contains the list of arguments in the order they were passed.
- All function parameters are passed as one single, flat list of scalars, and all return values likewise returned to the caller as one single flat list of scalars. **THIS IS A VERY IMPORTANT POINT!**
- You can access the passed values by direct access to `@_` (with elements `$_[0], $_[1], ...,etc.`), or by assigning values from `@_`.
- Using `@_` directly allows you to modify the global values in the calling program (pass by reference), otherwise, if you assign values from `@_`, then the original global values are not affected.

EXAMPLES

As our first example let's rewrite the `intrand` function to accept a `$max_val` as an input parameter supplied by the user.

```
#!/usr/local/bin/perl
#
#     modified version of Program 9.4
#     TYPERL 5 by David Till
#
#
$LOOP_MAX = 100;

srand();
print ("Random number tester.\n");
print ("Enter maximum value for random number> ");
chomp($max_val = <STDIN>);

#     create an array which keeps a count
#     of the frequency of occurence of the
#     random values [0, $max_val)
#
for ($count = 1; $count <= $LOOP_MAX; $count++) {
        $randnum[&intrand($max_val)] += 1;
}
```

```
printf ("Totals for the digits 0 through %d:\n",$max_val-1);
print ("@randnum\n");


#
#  function to return a random integer
#  in the range [0,$max_val)
#
sub intrand {
    my($max_val) = @_;     # local value passed into sub
    $num = int(rand($max_val));
}
__END__
```

```
OUTPUT

Random number tester.
Enter maximum value for random number> 5
Totals for the digits 0 through 4:
18 19 19 22 22

Random number tester.
Enter maximum value for random number> 16
Totals for the digits 0 through 15:
5 9 3 12 9 7 3 8 7 8 4 6 4 8 2 5
```

This example shows how Perl converts a passed array into a flattened list of scalars. It is possible to pass two arrays, but we'll use references for that.

```
  # load our array and print it
@nums =  (1..10);
print "The original array: @nums\n";

   # pass the array to 3 subroutines
&printnums_1(@nums);
&printnums_2(@nums);
&printnums_3(@nums);

#
#  define sub here
#

sub printnums_1 {
    my( @nums, $lastnum) = @_;              #array, scalar
    print "The numbers are: @nums\n";
    print "The last number is: $lastnum\n";
}

sub printnums_2 {
    my( $firstnum, @nums ) = @_;              #scalar, array
    print "The numbers are: @nums\n";
    print "The last number is: $firstnum\n";
}

sub printnums_3 {
    my( @nums, @array ) = @_;              #array, array
```

```
    print "The numbers are: @nums\n";
    print "The array is: @array\n";
}

OUTPUT

The original array: 1 2 3 4 5 6 7 8 9 10

The numbers are: 1 2 3 4 5 6 7 8 9 10
The last number is:

The numbers are: 2 3 4 5 6 7 8 9 10
The last number is: 1

The numbers are: 1 2 3 4 5 6 7 8 9 10
The array is:
```

- **Passing variables by reference to a subroutine**

As previously mentioned Perl passes all arguments to a subroutine as an implicit reference using the array `@_`. The problem with the process of passing arguments of any type, is that the are interpolated into a flat list of scalar values. This is ok in simple cases, but in more complex cases it gets tricky to separate out the input values into their original data elements (arrays or hashes for example).

To address this problem we can pass references instead of the values. Here is a simple example. Note that this subroutine modifies the orignal values in the main program.

```
#!/usr/local/bin/perl -w

$val_1 = 75;
$val_2 = 33;

print "Before switch: \$val_1 = $val_1 and  \$val_2 = $val_2\n";

&switch( \$val_1, \$val_2);  #pass in references to variables

print "After switch:  \$val_1 = $val_1 and  \$val_2 = $val_2\n";

sub switch {
   my ($a, $b) = @_;   # get inputs as references

 # deference the inputs
 # and switch values
   ($$a, $$b) = ($$b, $$a);
}

OUTPUT
Before switch: $val_1 = 75 and  $val_2 = 33
After switch:  $val_1 = 33 and  $val_2 = 75
```

QUESTION: What is the return value from `&switch`?

- **Passing multiple arrays to a function**

Again we can also use references to pass arrays into a function.

```
#!/usr/local/bin/perl -w
#
#  compute the product of two
#  vectors
#     A dot B = (a1*b1 + a2*b2 +...+ aN*bN)
#

@v1 = (1..5);  # init vectors
@v2 = (2..6);
@v3 = (1..10);

$ans1 = vectordot( \@v1, \@v2 );  # compute dot products
$ans2 = vectordot( \@v1, \@v3);

print "The results are:\nAns_1 =  $ans1 \nAns_2 = $ans2\n";

#
#    define function
#
sub vectordot {

        # get references to input vectors
        # make them local
  my($vec1, $vec2) = @_;

  my(@v1) = @$vec1;     # deference inputs
  my(@v2) = @$vec2;     # use local variables

                        #  are the inputs of the same size?
  if ( scalar(@v1) - scalar(@v2) != 0 ) {
      warn "Input vectors must be of same size: vectordot\n";
      return ("");   # return null
  }

  my($max) = scalar(@v1)-1;
  my $result = 0;

  for ($i=0; $i<= $max; $i++) {
      $result += $v1[$i] * $v2[$i];
  }

return ($result);
}

OUTPUT

# Input vectors must be of same size: vectordot
The results are:
Ans_1 =  70
Ans_2 =
```

- **Multiple return values**

As mentioned, Perl allows the return command to have 0 or more arguments. Thus you can return multiple values from a function as shown by this example.

```
#!/usr/local/bin/perl -w
#
#  multiple return values example
```

```
#  compute minimum and maximum
#  values of an array
#
#-----------------------------------#

@b = (1);
($min, $max) = &minmax(@b);
print "Min = $min \nMax = $max\n";


@a = (5..10);
($min, $max) = &minmax(@a);
print "Min = $min \nMax = $max\n";

@a = (5, 10, 44, -1, 33, 3, 504);
($min, $max) = &minmax(@a);
print "Min = $min \nMax = $max\n";

#
#  define subroutine
#
sub minmax {
#
#  compute both the minimum
#  and maximum values
#  in an array
#
    my(@input) = @_;

     # test if @input has enough elements
if (scalar(@input) < 2) {
    warn ("minmax function requires array with >= 2 elements\n");
    return ( () );  #return null list
}

# find min and max

my $min = $input[0]; # set initial values
my $max = $input[0];

foreach $next (@input){
    $min = ($next < $min) ? $next : $min;
    $max = ($next > $max) ? $next : $max;
}

  return ($min, $max);
}

OUTPUT

# minmax function requires array with >= 2 elements
# Use of uninitialized value.
# Use of uninitialized value.
Min =
Max =

Min = 5
Max = 10

Min = -1
Max = 504
```

Note: This example uses a new syntax borrowed from the C programming language:

```
$x = (EXPR1) ? EXPR2 : EXPR3;
```

if EXPR1 is TRUE then EXPR2 is returned, otherwise EXPR3 is returned. This is equivalent to:

```
if (EXPR1) {
  $x = EXPR2;
} else {
  $x = EXPR3;
}
```

- **Passing Arrays by Name using Aliases (type globs)**
  - Normally Perl copies an array passed into a subroutine into the variable `@_`. You can tell Perl not to copy the original array, but rather to operate directly on it. This in effect passes the name of the array into the subroutine.

```
@array = (5..30);

&doit(*array);

sub doit {
   local (*input) = @_;
   $size = scalar( @input );
   ....
   .....

}
```

  - The process of creating another name to refer to the same variable is known as *aliasing* . In the above example, `array` and `input` refer to the same list `@array`.

```
#!/usr/local/bin/perl

$foo = 26;
@foo = ("here's", "a", "list");

&testsub (*foo);

print ("The value of \$foo is now $foo\n");

sub testsub {
        my (*printarray) = @_;
        foreach $element (@printarray) {
                print ("$element\n");
        }
        $printarray = 61;
}
```

  - **Advantages** of aliases is that you don't waste time copying arrays; however, this may only

be important when working with very large arrays.

- **Disadvantages**
  - Program readability -- tracking which variables are being modified can be tricky.
  - Aliasing affects ALL variables with the same name, regardless of type. Thus if you have two variable `$input` and `@input`, both get passed when you use `*input` as a subroutine argument.

PERL 5 PREDEFINED SUBROUTINES

BEGIN

Executed at the startup of your program

```
#!/usr/local/bin/perl -w
#
BEGIN {
   print ("Welcome to the Widget Tool Program\n");
}
...
....rest of program...
...
__END__
```

OUTPUT

```
Welcome to the Widget Tool Program
```

END

Executed at the termination of your program

```
#!/usr/local/bin/perl -w
#
BEGIN {
   print ("Welcome to the Widget Tool Program\n");
}
...
....rest of program...
...

END {
   print ("Thank you for using the Widget Tool Program\n");
}
__END__
```

OUTPUT

```
Welcome to the Widget Tool Program

...
.....program I/O
...
```

```
Thank you for using the Widget Tool Program
```

AUTOLOAD

Executed when your program is trying to find a subroutine not included in the current file.

```
#!/usr/local/bin/perl -w
#
BEGIN {
   print ("Welcome to the Widget Tool Program\n");
}

AUTOLOAD {
   print "subroutine $AUTOLOAD not found\n";
   print "Arguments: @_\n";
}



$apple = &getgravity("Newton");

...
....rest of program...
...

END {
   print ("Thank you for using the Widget Tool Program\n");
}
__END__
```

OUTPUT

```
Welcome to the Widget Tool Program
subroutine main::getgravity not found
Arguments: Newton

...
.....program I/O
...

Thank you for using the Widget Tool Program
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 5]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Built-in Perl Functions Part II

Functions discussed in this lecture:

uc and lc	splice
ucfirst and lcfirst	shift and unshift
sprintf	push and pop
hex and oct	split
ord and chr	sort
scalar	reverse
pack and unpack	map
defined and undef	wantarray

STRING MANIPULATION FUNCTIONS: Part II

uc and lc functions

The uc (upper case) and lc (lower case) convert the case of a specified string ($string). The return value ($retval) is the converted string.

```
$retval = uc($string);
$retval = lc($string);
```

EXAMPLE

```
$str1 = "this is my first example.";
$str2 = "THIS IS MY SECOND EXAMPLE.";

$uc_str = uc($str1);
$lc_str = lc($str2);

print "Upper case:\n $uc_str\n";
print "Lower case:\n $lc_str\n";

OUTPUT

Upper case:
 THIS IS MY FIRST EXAMPLE.
Lower case:
 this is my second example.
```

ucfirst and lcfirst functions

The ucfirst (upper case) and lcfirst (lower case) convert the case of the first character of a specified string ($string). The return value ($retval) is the converted string.

NOTE: These functions operate on strings, which are Perl scalars. You can't feed them an array directly, you have to operate on the individual array elements (bummer!).

```
$retval = uc($string);
$retval = lc($string);
```

EXAMPLE

```
$str1 = "this is my first example.";
$str2 = "THIS IS MY SECOND EXAMPLE.";

$uc_str = ucfirst($str1);
$lc_str = lcfirst($str2);

print "Upper case:\n $uc_str\n";
print "Lower case:\n $lc_str\n";

OUTPUT

Upper case:
 This is my first example.
Lower case:
 tHIS IS MY SECOND EXAMPLE.
```

---

sprintf

The `sprintf` function (a descendant from C) is similar to the `printf` function we've already discussed. However, rather than printing the formatted string to a `filehandle` it is returned by the function.

EXAMPLE

```
#!/usr/local/bin/perl -w
#
$age = 25;
$mysign = 'Aries.';

$fmt = "My age is %d and my sign is %s\n";
$outstr1 = sprintf ($fmt,$age,$mysign);

$fmt = "My age is %o (octal) and my sign is %s\n";
$outstr2 = sprintf($fmt,$age,$mysign);

$fmt = "My age is %x (hex) and my sign is %s\n";
$outstr3 = sprintf($fmt,$age,$mysign);

print "$outstr1\n$outstr2\n$outstr3\n";


OUTPUT

My age is 25 and my sign is Aries.

My age is 31 (octal) and my sign is Aries.
```

```
My age is 19 (hex) and my sign is Aries.
```

SCALAR CONVERSION FUNCTIONS

hex EXPR

The hex function converts EXPR from a hexadecimal number to a decimal number. If EXPR is omitted, uses $_. hex does not handle hexidecimal numbers that start with 0x or 0X (we'll see how to strip these off using regexps in Lecture 6, or use oct() function).

EXAMPLE

```
$decval = 255;

$hexval = sprintf("%x",$decval);

$num = hex($hexval);

print "Decimal value = $decval\n";
print "Hex value = $hexval\n";
print "Number = $num\n";

OUTPUT

Decimal value = 255
Hex value = ff
Number = 255
```

---

oct EXPR

The oct function interprets EXPR as an octal string and returns the corresponding decimal value. If EXPR happens to start off with 0x or 0X, interprets it as a hex string instead.

EXAMPLE

```
$decval = 255;

$octval = sprintf("%o",$decval);

$num = oct($octval);

print "Decimal value = $decval\n";
print "Octal value = $octval\n";
print "Number = $num\n";

OUTPUT

Decimal value = 255
Octal value = 377
Number = 255
```

The next two functions allow you to manipulate the ASCII character set. The ASCII character set

defines 256 characters which are specified using 8 bits. If you specify a value greater than 8 bits these functions use only the lowest 8 bits.

ord EXPR

The ord function returns the numeric ASCII value of the first character of EXPR. If EXPR is omitted, uses $_.

EXAMPLE

```
$str = "\nThis is my fist example";
$asciival = ord($str);
print "The ASCII value is: $asciival\n";

OUTPUT

The ASCII value is: 13    (newline character ascii value).
```

NOTE: Some ASCII characters are non-printing in the sense that you don't see them in *black ink* on the page when you print them, but you can see their effect, such as with the newline character.

---

chr EXPR

The chr function returns the character represented by EXPR in the ASCII character set.
NOTE: This is a Perl 5 only function.

EXAMPLE

```
$string = chr(65);
print ("The character is: $string\n");

OUTPUT
The character is: A
```

---

scalar EXPR

The scalar function forces EXPR to be interpreted in a scalar context and returns the value of EXPR. For example, the array @mylist can be interpreted in Perl in an *array or list context*, in which case it refers to all the elements of @mylist. But @mylist can also be interpreted in a *scalar context* in which case $size_mylist = @mylist, returns the number of elements in the array @mylist. The scalar function just forces the scalar context. We've seen its use before primarily in determining the size of an array or list.

EXAMPLE

```
@mylist = qw(fish cat dog mouse hamster bird);
print "The list of pets is ", scalar(@mylist)," elements long.\n";

OUTPUT

The list of pets is 6 elements long.
```

---

pack TEMPLATE,LIST

The pack function takes an array or list of values, specified by LIST, and packs it into a binary structure, returning the string containing the structure. The TEMPLATE is a sequence of characters that give the order and type of values, as shown in the table below.

FORMAT CHARACTERS FOR THE pack FUNCTION

CHARACTER	DESCRIPTION
A	An ASCII string, will be space padded.
a	An ASCII string, will be null padded.
b	A bit string (ascending bit order, like vec()).
B	A bit string (descending bit order).
h	A hex string (low nybble first).
H	A hex string (high nybble first).
c	A signed char value.
C	An unsigned char value.
s	A signed short value.
S	An unsigned short value.
i	A signed integer value.
I	An unsigned integer value.
l	A signed long value.
L	An unsigned long value.
n	A short in "network" order.
N	A long in "network" order.
v	A short in "VAX" (little-endian) order.
V	A long in "VAX" (little-endian) order.
f	A single-precision float in the native format.
d	A double-precision float in the native format.
p	A pointer to a null-terminated string.
P	A pointer to a structure (fixed-length string).

u	A uuencoded string.
x	A null byte.
X	Back up a byte.
X	Back up a byte.

- Each letter may optionally be followed by a number which gives a repeat count.
- For all types **except** `a, A, b, B, h` and `H`, and `P` the `pack` function will gobble up that many values from the `LIST`.
- A `*` for the repeat count means to use however many items are left.
- The `a` and `A` types gobble just one value, but pack it as a string of length count, padding with nulls or spaces as necessary. (When unpacking, `A` strips trailing spaces and nulls, but `a` does not.) Likewise, the `b` and `B` fields pack a string that many bits long.
- The `h` and `H` fields pack a string that many nybbles long.
- The `P` packs a pointer to a structure of the size indicated by the length.
- Real numbers (floats and doubles) are in the native machine format only; due to the multiplicity of floating formats around, and the lack of a standard *network* representation, no facility for interchange has been made. This means that packed floating point data written on one machine may not be readable on another - even if both use `IEEE` floating point arithmetic (as the endian-ness of the memory representation is not part of the `IEEE` spec). Note that Perl uses doubles internally for all numeric calculation, and converting from double into float and thence back to double again will lose precision.

EXAMPLES

```
$foo = pack("cccc",65,66,67,68);
    # foo eq "ABCD"

$foo = pack("c4",65,66,67,68);
    # same thing

$foo = pack("ccxxcc",65,66,67,68);
    # foo eq "AB\0\0CD"

$foo = pack("a4","abcd","x","y","z");
    # "abcd"

@mylist = ("test1", "test2");
$strings = pack("a6" x 2, @mylist);
print "$strings\n";

OUTPUT
test1\0test2\0
```

NOTE: The same template may generally also be used in the `unpack` function.

unpack TEMPLATE,EXPR and uuencode

The unpack function converts a binary structure into normal perl variables. unpack does the reverse of pack: it takes a string representing a structure and expands it out into a list value, returning the array value. (In a scalar context, it merely returns the first value produced.) The TEMPLATE has the same format as in the pack function. See the above table.

In addition, you may prefix a field with a %<number> to indicate that you want a <number>-bit checksum of the items instead of the items themselves. Default is a 16-bit checksum.

EXAMPLE

```
@mylist = ("test1", "test2");
$strings = pack("a6" x 2, @mylist);
print "$strings\n";
@newlist = unpack("A6" x 2, $strings);
print "@newlist\n";

OUTPUT
test1\0test2\0
test1 test2
```

Note that when I decoded the strings packed with a, which had the null termination character \0 inserted (a very *C-thing* to do). I used A, which stripped out the \0 characters.

EXAMPLE: uuencode

This is a Unix system function for converting a file (binary or ASCII) to a file containing only ASCII characters. uuencode will convert all characters even unprintable characters. It is a handy way to send binary files via email and across networks.

EXAMPLE

In this example I uuencoded the file quack.txt:

```
gasster> cat quack.txt
We have two ducks. One blue. One black.
And when our blue duck goes "Quack-quack"
our black duck quickly quack-quacks back.
The quacks Blue quacks make her quite a quacker
but Black is a quicker quacker-backer.
gasster>

gasster> uuencode quack.txt quack.txt >quack.uue
gasster> cat quack.uue
begin 644 quack.txt
M5V4@:&%V92!T=V\@9'5C:W,N($]N92!B;'5E+B!/;F4@8FQA8VLN"D%N9"!W
M:&5N(&]U<B!B;'5E(&1U8VL@9V]E<R B475A8VLM<75A8VLB"F]U<B!B;&%C
M:R!D=6-K('%U:6-K;'D@<75A8VLM<75A8VMS(&)A8VLN"E1H92!Q=6%C:W,@
M0FQU92!Q=6%C:W,@;6%K92!H97(@<75I=&4@82!Q=6%C:V5R"F)U="!";&%C
?:R!I<R!A('%U:6-K97(@<75A8VME<BUB86-K97(N"F%C

end
gasster>
```

I then used Perl to decode this file:

```
#!/usr/local/bin/perl

#
#
BEGIN {
    unshift( @INC, "/home/gasster/perllib");

    print "\n\nWelcome to the uudecoder program!\n\n";
}

require "getUserInput.pl";

$path = "/home/gasster/perlcourse/";
$uufile = getUserInput("Enter uuencoded file name");
$infile = $path . $uufile;

open (CODEDFILE, "$infile") or
        die ("Can't open input file: $infile");

$outfile = getUserInput("Enter name for decoded file");
open (OUTFILE, ">$outfile") or
        die ("Can't open output file");

while ($line = <CODEDFILE>) {
    $decoded = unpack("u", $line);
    print OUTFILE ($decoded);
}

close (OUTFILE);
close (CODEDFILE);
exit (1);
```

Decoding:

```
gasster> uudecode.pl


Welcome to the uudecoder program!

Enter uuencoded file name> quack.uue
Enter name for decoded file> quack.out

gasster> cat quack.out
We have two ducks. One blue. One black.
And when our blue duck goes "Quack-quack"
our black duck quickly quack-quacks back.
The quacks Blue quacks make her quite a quacker
but Black is a quicker quacker-backer.
```



Becareful unpacking a uuencoded file on a non-Unix system. Some of the escape characters might have different ASCII values and the text might not get parsed as it would on a Unix system.

defined EXPR

The defined function tests whether a value, variable, or function is defined. Returns a boolean value (TRUE or FALSE) saying whether EXPR has a real value or not. In Perl, any scalar or array element that has not been assigned a value is assumed to contain the NULL string. Many operations return the undefined value under exceptional conditions, such as end of file, uninitialized variable, system error and such.

This function allows you to distinguish between an undefined null scalar and a defined null scalar with operations that might return a real null string, such as referencing elements of an array. You may also check to see if arrays or subroutines exist. Use of defined on predefined variables is not guaranteed to produce intuitive results. When used on a hash array element, it tells you whether the value is defined, not whether the key exists in the hash.

EXAMPLE

```
#
#  use the fact that some operations return undefined
#  when they run out of data to print the contents
#  of the Users list
#
@Users = qw(joe jack nancy fred al peter sally);
print "$val\n" while defined($val = pop(@Users));

OUTPUT
sally
peter
al
fred
nancy
jack
joe

#
#  make sure you've defined a subroutine
#

if (defined &getUserInput) {
   $input = &getUserInput("Hey, give me some input");
} else {
   die ("Function getUserInput is not defined: main prog\n")
}

OUTPUT
# Function getUserInput is not defined: main prog
```

undef EXPR

The undef function removes a variable or function definition. Undefines the value of EXPR. Use only on a scalar value, an entire array, or a subroutine name. (Using undef() will probably not do what you expect on most predefined variables or DBM list values, so don't do that.)

Always returns the undefined value. You can omit the EXPR, in which case nothing is undefined, but you still get an undefined value that you could, for instance, return from a subroutine.

EXAMPLE

```
$val = 5;
$result = undef $val;
print "The result is $result, which should be a NULL value\n";

OUTPUT
The result is , which should be a NULL value
```

LIST-MANIPULATION FUNCTIONS

splice ARRAY, OFFSET, LENGTH, LIST
splice ARRAY, OFFSET, LENGTH
splice ARRAY, OFFSET

splice removes the elements designated by OFFSET and LENGTH from an array, and replaces them with the elements of LIST, if any. Returns the elements removed from the array. The array grows or shrinks as necessary. If LENGTH is omitted, removes everything from OFFSET onward. Thus, splice can be used to REPLACE, APPEND and DELETE list elements.

EXAMPLES

Replacement:

```
@Users = qw(joe jack nancy fred al peter sally);
print "Original User List:\n@Users\n";

@replaced = splice(@Users, 3, 2, qw(sam sue));
print "New User List:\n@Users\n";
print "Replaced: @replaced\n";
OUTPUT

Original User List:
joe jack nancy fred al peter sally
New User List:
joe jack nancy sam sue peter sally
Replaced: fred al
```

Appending:

```
@Users = qw(joe jack nancy fred al peter sally);
print "Original User List:\n@Users\n";

@removed = splice(@Users, scalar(@Users), 2, qw(sam sue));
print "New User List:\n@Users\n";
print "Removed: @removed\n";
OUTPUT

Original User List:
joe jack nancy fred al peter sally
New User List:
joe jack nancy fred al peter sally sam sue
Removed:          #blank since nothing was removed/replaced
```

Deletion:

```
@Users = qw(joe jack nancy fred al peter sally);
print "Original User List:\n@Users\n";

@removed = splice(@Users, 4, 3);
print "New User List:\n@Users\n";
print "Deleted: @removed\n";

OUTPUT
Original User List:
joe jack nancy fred al peter sally
New User List:
joe jack nancy fred
Deleted: al peter sally
```

---

shift (ARRAY) and unshift (ARRAY, LIST)

The shift function shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down. If there are no elements in the array, returns the undefined value. If ARRAY is omitted, shifts the @ARGV array in the main program, and the @_ array in subroutines.

unshift does the opposite of shift. Prepends LIST to the front of ARRAY, and returns the new number of elements in the array. Note the LIST is prepended whole, not one element at a time, so the prepended elements stay in the same order.

EXAMPLES

```
@Users = qw(joe jack nancy fred al peter sally);
$size = scalar(@Users);
for ($i=1; $i<=$size ; $i++) {
   $user = shift( @Users );
   print "User No. $i is $user\n";
}

OUTPUT
User No. 1 is joe
User No. 2 is jack
User No. 3 is nancy
User No. 4 is fred
User No. 5 is al
User No. 6 is peter
User No. 7 is sally

----
@Users = qw(joe jack nancy fred al peter sally);

while ( $user = <STDIN> ){
chomp($user);
$newsize = unshift(@Users, $user);
}
print "New size = $newsize\n";
print "@Users\n";

INPUT/OUTPUT

bob
jake
frank
```

```
New size = 10
frank jake bob joe jack nancy fred al peter sally
```

---

push (ARRAY, LIST) and pop (ARRAY)

push treats ARRAY as a stack, and pushes the values of LIST onto the end of ARRAY. The length of ARRAY increases by the length of LIST. Returns the new number of elements in the array.

The pop function undoes the effect of the push function, it returns the last value of ARRAY, shortening the array by 1. If there are no elements in the array, returns the undefined value. If ARRAY is omitted, pops the @ARGV array in the main program, and the @_ array in subroutines, just like shift() .

EXAMPLE

```
@Users = qw(joe jack nancy fred al peter sally);

while ( $user = <STDIN> ){
chomp($user);
$newsize = push(@Users, $user);
}
print "New size = $newsize\n";
print "@Users\n";

INPUT/OUTPUT
egbert
june
diane
New size = 10
joe jack nancy fred al peter sally egbert june diane

----
@Users = qw(joe jack nancy fred al peter sally);
$size = scalar(@Users);
for ($i=$size; $i>=1 ; --$i) {
   $user = pop( @Users );
   print "User No. $i is $user\n";
}

OUTPUT
User No. 7 is sally
User No. 6 is peter
User No. 5 is al
User No. 4 is fred
User No. 3 is nancy
User No. 2 is jack
User No. 1 is joe
```

---

split /PATTERN/,EXPR,LIMIT
split /PATTERN/,EXPR
split /PATTERN/
split

The split function breaks a string into an array of strings, and returns it.

- If not in a list context, returns the number of fields found and splits into the @_ array. For example:

```
$str = "dog:cat:mouse:fish";
$pets = split(/:/, $str);
print "@_\n";
print "pets = $pets\n";

OUTPUT
dog cat mouse fish
pets = 4
```

- If EXPR is omitted, splits the $_ string.
- If PATTERN is also omitted, splits on whitespace (after skipping any leading whitespace).For example:

```
$_ = "dog cat mouse fish";
@pets = split;
print "pets = @pets\n";

OUTPUT
pets = dog cat mouse fish
```

Anything matching PATTERN is taken to be a delimiter separating the fields. (Note that the delimiter may be longer than one character.)

- If LIMIT is specified and is not negative, splits into no more than that many fields (though it may split into fewer). If LIMIT is unspecified, trailing null fields are stripped (which potential users of pop() would do well to remember). If LIMIT is negative, it is treated as if an arbitrarily large LIMIT had been specified.

The LIMIT parameter can be used to partially split a line, for example:

```
$str = "dog:cat:mouse:fish";
@pets = split(/:/, $str, 2);  #only spilt into two elements
$size = @pets;
print "Size of pets = $size\n";
print "pets = @pets\n";


OUTPUT
Size of pets = 2
pets = dog cat:mouse:fish
```

When assigning to a list, if LIMIT is omitted, Perl supplies a LIMIT one larger than the number of variables in the list, to avoid unnecessary work. In time critical applications it behooves you not to split into more fields than you really need.

- A pattern matching the null string (not to be confused with a null pattern //, which is just one member of the set of patterns matching a null string) will split the value of EXPR into separate characters at each point it matches that way. For example:

```
print join(':', split(/ */, 'hi there'));
```

where the pattern / */ splits on zero or more spaces, and produces the output

`h:i:t:h:e:r:e`. (We'll learn more about specifying patterns in Lecture 6.)

- If the PATTERN contains parentheses, additional array elements are created from each matching substring in the delimiter.

```
split(/([,-])/, "1-10,20");
```

where the pattern `/([,-])/` means to split on either a `,` or `-`, and the `()` tells Perl to include these characters in the resulting list. There it produces the list value:

```
(1, '-', 10, ',', 20)
```

- As a special case, specifying a PATTERN of space (' ') will split on white space just as split with no arguments does.
- A `split` with no arguments really does a `split(' ', $_ )` internally.

---

sort (LIST)

The `sort` function sorts `LIST` in alphabetical order, returning the sorted list.

EXAMPLE

```
@Users = qw(joe jack nancy fred al peter sally);
foreach $user (sort @Users) {
    print "$user\n";
}

OUTPUT
al
fred
jack
joe
nancy
peter
sally
```

reverse (LIST)

The `reverse` function returns `LIST` in reverse order.

EXAMPLE

```
@Users = qw(joe jack nancy fred al peter sally);
foreach $user (reverse sort @Users) {
    print "$user\n";
}

OUTPUT
sally
peter
nancy
joe
jack
fred
```

```
al
```

---

map BLOCK LIST
map EXPR,LIST

The map function evaluates the BLOCK or EXPR for each element of LIST (locally setting $_ to each element) and returns the list value composed of the results of each such evaluation. Evaluates BLOCK or EXPR in a list context, so each element of LIST may produce zero, one, or more elements in the returned value.

EXAMPLE

```
@Users = qw(joe jack nancy fred al peter sally);

@newUsers = map(ucfirst($_), @Users);
foreach $user (sort @newUsers) {
    print "$user\n";
}

OUTPUT
Al
Fred
Jack
Joe
Nancy
Peter
Sally
```

---

wantarray

Returns TRUE if the context of the currently executing subroutine is looking for a list value. Returns FALSE if the context is looking for a scalar.

EXAMPLE

```
#!/usr/local/bin/perl

@array = &mysub();
$scalar = &mysub();

print "Array context: @array\n";
print "Scalar context: $scalar\n";

sub mysub {
    my (@list) = (1..5);
    my $value = 3.14159;

    if (wantarray()) {
        print ("Returning a List\n");
        return (@list);
    } else {
        print ("Returning a scalar\n");
        return ($value);
    }
}
```

```
OUTPUT
Returning a List
Returning a scalar
Array context: 1 2 3 4 5
Scalar context: 3.14159
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 5]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 5: Perl LIBRARIES

THE STANDARD PERL LIBRARY

Perl comes with a standard set of library functions, packages and modules. As a user you can find out what's included in your systems standard library.

Perl uses the `@INC` array to store a list of places to look for Perl scripts to be evaluated by the `do EXPR`, `require` or `use` constructs. The `@INC` consists of any arguments to the `-I` command-line switch, or any paths you specify within your Perl program (see below), followed by a standard list of directory paths. To examine the contents of the `@INC` array on your system try the following (if you can't do command-line then try the equivalent Perl script):

```
gasster> perl -e '$"="\n";print "@INC\n"';
/opt/gnu/lib/perl5/sun4-solaris/5.002
/opt/gnu/lib/perl5
/opt/gnu/lib/perl5/site_perl/sun4-solaris
/opt/gnu/lib/perl5/site_perl
.
gasster>
```

Note that the directory . is included. This is the current directory.

You can also list all the Perl scripts and modules found in this set of directories using the following command (This is for a Unix system. On other systems read your release notes for information about where the libraries are stored).

```
gasster> find `perl -e 'print "@INC"'` -name '*.p*' -print
/opt/gnu/lib/perl5/sun4-solaris/5.002/Config.pm
/opt/gnu/lib/perl5/AnyDBM_File.pm
/opt/gnu/lib/perl5/AutoLoader.pm
/opt/gnu/lib/perl5/AutoSplit.pm
/opt/gnu/lib/perl5/Benchmark.pm
/opt/gnu/lib/perl5/Carp.pm
/opt/gnu/lib/perl5/Cwd.pm
/opt/gnu/lib/perl5/Devel/SelfStubber.pm
/opt/gnu/lib/perl5/DirHandle.pm
/opt/gnu/lib/perl5/DynaLoader.pm
/opt/gnu/lib/perl5/English.pm
/opt/gnu/lib/perl5/Env.pm
/opt/gnu/lib/perl5/Exporter.pm
/opt/gnu/lib/perl5/ExtUtils/Install.pm
/opt/gnu/lib/perl5/ExtUtils/Liblist.pm
/opt/gnu/lib/perl5/ExtUtils/MM_OS2.pm
/opt/gnu/lib/perl5/ExtUtils/MM_Unix.pm
/opt/gnu/lib/perl5/ExtUtils/MM_VMS.pm
/opt/gnu/lib/perl5/ExtUtils/MakeMaker.pm
/opt/gnu/lib/perl5/ExtUtils/Manifest.pm
...
.....shortened to save space....
```

```
...
/opt/gnu/lib/perl5/Tie/Scalar.pm
/opt/gnu/lib/perl5/Tie/SubstrHash.pm
/opt/gnu/lib/perl5/Time/Local.pm
/opt/gnu/lib/perl5/abbrev.pl
/opt/gnu/lib/perl5/assert.pl
/opt/gnu/lib/perl5/bigfloat.pl
/opt/gnu/lib/perl5/bigint.pl
/opt/gnu/lib/perl5/bigrat.pl
/opt/gnu/lib/perl5/cacheout.pl
/opt/gnu/lib/perl5/chat2.pl
/opt/gnu/lib/perl5/complete.pl
/opt/gnu/lib/perl5/ctime.pl
/opt/gnu/lib/perl5/diagnostics.pm
/opt/gnu/lib/perl5/dotsh.pl
/opt/gnu/lib/perl5/dumpvar.pl
/opt/gnu/lib/perl5/exceptions.pl
/opt/gnu/lib/perl5/fastcwd.pl
/opt/gnu/lib/perl5/find.pl
/opt/gnu/lib/perl5/finddepth.pl
...
.....shortened to save space....
...
/opt/gnu/lib/perl5/pwd.pl
/opt/gnu/lib/perl5/shellwords.pl
/opt/gnu/lib/perl5/sigtrap.pm
/opt/gnu/lib/perl5/stat.pl
/opt/gnu/lib/perl5/strict.pm
/opt/gnu/lib/perl5/subs.pm
/opt/gnu/lib/perl5/sun4-solaris/5.002/Config.pm
/opt/gnu/lib/perl5/syslog.pl
/opt/gnu/lib/perl5/tainted.pl
/opt/gnu/lib/perl5/termcap.pl
/opt/gnu/lib/perl5/timelocal.pl
/opt/gnu/lib/perl5/validate.pl
/opt/gnu/lib/perl5/vars.pm
./howto04/concord.pl           <-----NOTE: these are in my current directory
./howto04/concord.pl4            |
./gethostname.pl                 |
./gethostadd.pl                  V
./getnets.pl
./xdump.pl
./price2.pl
./price.pl
./test.pl
./hoot.pl
./heretest.pl
./word_search.pl
./capfile.pl
./makeallcaps.pl
./viewfiles.pl
./wordsearch2.pl
./rwtest.pl
./prog_12_10.pl
./opentest.pl
./getfilestats.pl
./pipetestin.pl
./pipetestout.pl
./perlcat.pl
./checkfile.pl
./bug1.pl
```

```
./dumpenv.pl
./fmt_ex1.pl
./dirtest.pl
./TESTING/shoppingcart.pl
./chmod_ex.pl
./dirseek.pl
./testlib.pl
```

As you can see there are quite a few different perl scripts (files ending with `.pl`) and Perl modules (files ending with `.pm`). We will talk about Perl modules in a later lecture. For now a module is just a Perl library that conforms to specific conventions.

CREATING YOUR OWN LIBRARY

• CREATE A LIBRARY DIRECTORY

Somewhere on your system create a directory where you want all your Perl library functions to be stored. For example, on a Unix platform you could define a subdirectory from you home directory:

```
/home/gasster/perllib
```

• ADD YOUR PERL FILES

Create a set of files with your functions, subroutines and other Perl scripts that you want to be in your library. Put these in your library directory. For example, let's create to functions called `getUserInput` and `printUserInput`. We'll store them in two files in our perllib directory: `getUserInput.pl` and `printUserInput.pl`.

For example:

```
gasster> pwd
/home/gasster/perllib
gasster> ls -l
total 68
-rw-r--r--   1 gasster  users      14667 Aug 17 11:29 cgi-lib.pl
-rw-r--r--   1 gasster  users        662 Oct 12 10:50 getUserInput.pl
drwxr-xr-x   3 gasster  users        512 Sep 28 00:06 modules
-rw-r--r--   1 gasster  users       1051 Aug 19 21:44 num2word.pl.gz
-rw-r--r--   1 gasster  users        526 Oct 12 10:50 printUserInput.pl
-rwxr-xr-x   1 gasster  users       3709 Aug 19 21:17 striphtml.gz
gasster>
```

The functions are:

---

getUserInput.pl

```
#
#-------------------------------------------------------------#
#       file: getUserInput.pl
#
#    function getUserInput
#
```

```
#     reads a line of input from STDIN
#     requires the user to enter something
#     other than a <CR>
#
#     s. gasster 9/96
#-------------------------------------------------------------#
#
sub getUserInput {
        my($prompt) = @_;       #prompt to get user input
        my($UserInput);         #what the user enters

        #get input from user, don't accept null input
        do {
                print ("$prompt> ");
                $UserInput = <STDIN>;
                chomp( $UserInput );    #remove newline character
        } until ($UserInput ne "");

        return $UserInput;
}

1;    # value needed by require function
```

printUserInput.pl

```
#
#-------------------------------------------------------------#
#       file: printUserInput.pl
#
#     function printUserInput
#
#     prints the output from getUserInput
#
#     s. gasster 9/96
#-------------------------------------------------------------#
#

sub printUserInput {

my($input) = @_;

if ($input eq "") {
    warn "Null input value: printUserInput";
    return (0);    #return false
} else {

    print <<"EOI";
The user entered the following line:
$input
EOI
}
return (1);
}

1;    # value needed by require function
```

I've written a short Perl script that uses these two functions for illustrating the concepts in this section: `testlib.pl`. The two functions are loaded using the `require` function which we will discuss in this section (see below).

testlib.pl

```
#!/usr/local/bin/perl -w
#
#   file: testlib.pl
#
#   test the library functions in Perl
#
#----------------------------------------#
#

require "getUserInput.pl";
require "printUserInput.pl";

&printUserInput( &getUserInput("Enter a line") );

exit (1);
```

- **TELL PERL WHERE TO LOOK**

Perl needs to know where to find your library files. There are several different ways to do this.

- Set the Unix Environment variables

  PERL5LIB
  A colon-separated list of directories in which to look for Perl library files before looking in the standard library and the current directory. If PERL5LIB is not defined, PERLLIB is used.

  PERLLIB
  A colon-separated list of directories in which to look for Perl library files before looking in the standard library and the current directory. If PERL5LIB is defined, PERLLIB is not used.

  So for example, if my home directory is /home/users/gasster, I can setup my own Perl library directory where I put my Perl functions, etc.

```
$cd ~gasster     (make sure I'm in my home directory)
$mkdir perllib   (make my own perl library subdirectory)
$vi .cshrc               (edit my .cshrc file to add PERL5LIB, since
...
setenv PERL5LIB /home/gasster/perllib
...
$source .cshrc
$printenv  PERL5LIB
/home/gasster/perllib
```

- From the command-line using the `-I` switch

```
gasster> perl -I/home/gasster/perllib testlib.pl
```

- From within the Perl script itself using the `BEGIN` subroutine to modify the `@INC` array. Here

is our testlib.pl script modified to load the @INC array.

```
#!/usr/local/bin/perl -w
#
#   file: testlib.pl
#
#   test the library functions in Perl
#
#----------------------------------------#
#

BEGIN {
    unshift( @INC, "/home/gasster/perllib");
}

require "getUserInput.pl";
require "printUserInput.pl";


&printUserInput( &getUserInput("Enter a line") );

exit (1);
```

• LOAD THE FUNCTION(S) INTO YOUR PERL PROGRAM

To load our library functions into our Perl program so we can access them, use the `require` function.

require EXPR
require

The `require` function will load the specified Perl code. It will search the paths defined by the `@INC` array. If `EXPR` is not specified then the value of `$_` is used to get the filename.

Note that the file will not be included twice under the same specified name. The file must return `TRUE` as the last statement to indicate successful execution of any initialization code, so it's customary to end such a file with

```
1;
```

unless you're sure it will return `TRUE` otherwise. But it's better just to put the `1;`, in case you add more statements. This was done when I wrote the `getUserInput.pl` and `printUserInput.pl` files; each ends with `1;`.

EXAMPLES

```
require ("myfunc.pl");  #load the file myfunc.pl

require;                #load the filename given in $_
```

Using the various methods described above we can run the testlib.pl program. If we forget to define the path for Perl to locate our functions then we get an error:

```
gasster> testlib.pl
```

```
Can't locate getUserInput.pl in @INC at testlib.pl line 7.
```

First let's use the command-line switch:

```
gasster> perl -I/home/gasster/perllib testlib.pl
Enter a line> This is my only line.  Do you like it?
The user entered the following line:
This is my only line.  Do you like it?
```

Edit testlib.pl and add the `BEGIN {....}` code to add library path to the `@INC` array.

```
gasster> testlib.pl
Enter a line> This is another test; hope it works.
The user entered the following line:
This is another test; hope it works.
```

Now define the `PERL5LIB` environment variable, and for this case we don't need the `BEGIN {....}` code.

```
gasster> setenv PERL5LIB /home/gasster/perllib
gasster> testlib.pl
Enter a line> This is the last test. Let's do something else!
The user entered the following line:
This is the last test. Let's do something else!
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 5]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 5

Note: Problems 1, 2 and 4 should be relatively easy, while problem 3 is long, it should not be too hard.

1. What is wrong with the following program?

```
#!/usr/local/bin/perl

for ($count =1; $count<= 10; $count++) {
    &print_ten( $count );
}

sub print_ten{
    my ($mulitplier) = @_;
    for ($count =1; $count<= 10; $count++) {
        $printval = $multiplier * 10 + $count;
        print ("$printval\n");
    }
}
```

2. Write a subroutine or set of subroutines that will return the following HTML code:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML>
<HEAD>
        <TITLE>PAGE TITLE</TITLE>
</HEAD>
<BODY>
<H1>MY FAVORITE HEADING</H1>

...BODY GOES HERE...(user specified)

</BODY>
</HTML>
```

The text PAGE TITLE and MY FAVORITE HEADING are defined by arguments passed to the subroutine(s), as is the body of the HTML page. Thus in one example, you should be able to call the subroutine in the following manner:

```
$body = &read_the_body;
    $pagetitle = "PAGE TITLE";
    $heading1 = "MY FAVORITE HEADING";
    $mypage = &make_html($pagetitle, $heading1, $body);
    print $mypage;
```

Show that your subroutine(s) work using an example.

---

3. **`files_by_day.pl` Program Revisited**
Recall the code fragment given in Lecture 4: References, called `files_by_day.pl`. In this problem you will *up-grade* this to a real program. I want to see lots of modularity! Use subroutines and functions! Think about how to organize this problem into separate tasks before you start writing code.

```
#!/usr/local/bin/perl
#
#       file: files_by_day.pl
#

# create named hash of lists
%files_by_day = (
    "Mon" => ["Mon01", "Mon02", "Mon03"],
    "Tue" => ["Tue01", "Tue02"],
    "Wed" => ["Wed01", "Wed02", "Wed03"],
    "Thu" => ["Thu01", "Thu02", "Thu03"],
     "Fri" => ["Fri01", "Fri02", "Fri03"],
     "Sat" => ["Sat01", "Sat02", "Sat03", "Sat04"],
     "Sun" => ["Sun01", "Sun02"]
);

@days = qw(Mon Tue Wed Thu Fri Sat Sun);

  # get week day
($sec,$min,$hour,$mday,$mon,$year,
      $wday,$yday,$isdst) = localtime(time);

  # use $wday, an integer corresponding to the DOW
$day_of_week = $days[$wday];
print "Today is $day_of_week\n";

# extract list from hash
@file_list = @{$files_by_day{$day_of_week}};

# print list of files
foreach $file (@file_list){
    print "File: $file".".html\n";
}
```

Each week a database of filenames will be upload to your website. This database will be found in:

```
/web/uploads/weekly.db
```

The database format is a set of colon separated fields:

```
mon:filem1:filem2:...filem_Nm
tue:filet1:filet2:...filet_Nt
wed:filew1:filew2:...filew_Nw
thu:fileth1:fileth2:...fileth_Nth
fri:filef1:filef2:...filef_Nf
sat:filesa1:filesa2:...filesa_Nsa
sun:filesu1:filesu2:...filesu_Nsu
```

Where `Nm, Nt, ..., Nsu` represent the number of files corresponding to each day. Thus, there do not have to be the same number of file names for each day of the week.

In addition to the database, the files specified in the database above are also uploaded to the following directories:

```
/web/uploads/mon/
/web/uploads/tue/
/web/uploads/wed/
/web/uploads/thu/
/web/uploads/fri/
/web/uploads/sat/
/web/uploads/sun/
```

Each night at 11:55PM the Perl program will be run automatically (on a Unix system it could be a cron job) to perform the following functions:

i. Based on the day of the week (DOW) for **TOMORROW** read the database and extract the correct list of files.

ii. Using this list and the DOW copy the correct set of files from the corresponding uploads subdirectory to the following directory:

```
/web/hpages/
```

Each file name should get either `.html` or `.htm` appended to the name. But, before you do the copy, make sure the target directory is empty, i.e., delete any old or existing files.

iii. Finally, your program should create and up-date a log file. The log file will be:

`/web/uploads/logs/daily_log.html`

(if you're on a PC with short names use `Dlog.htm`). This is an html file with a title: Daily Log File.
Each entry will have the form

```
<HR>
data/time
...
.....listing of files in /web/hpages/
.....
<HR>
```

NOTE:
You can test your script using the date and time as input parameters if that is easier for you.

---

4. **Passing a Hash**

What happens when you pass a hash (associative array) to a subroutine? Show an example to illustrate what happens.

[TITLE PAGE| TABLE OF CONTENTS | HOMEWORK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 6:Regular Expressions in Perl

READING ASSIGNMENT:
TYPERL 5 Ch. 7, 13,14
Ch. 17,pp 646-650

HOMEWORK ASSIGNMENT:
Assignment No. 6 (Due 11/11/96).

Perl MANPAGES:

- perlre - manual page

TOPICS

- What is a Regular Expression?
- Regular Expression Syntax
- Pattern Binding Operators
- Regular Expression Special Characters and Metacharacters
- `m//`, `s///`, and `tr///` Functions
- String Manipulation Functions Related to Regular Expressions

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 7]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 6
Working With Patterns and Regular Expressions

HEY!!

Remember, when you write a script show me that it works by giving me some sample Input(s) and Output(s). That is part of the assignment!

---

1. **File Search Script**

   Write Perl scripts to search through designated HTML files for the following

   A. Any line with

      ```
      <!--anytexthere-->
      ```

      and print out the file name and the line containing this pattern. This pattern represents either an HTML comment field or a Server Side Include (SSI), which we will discuss later.

   B. All IMG HTML tags, and extract and report all image file names (full path)

   C. All HREF links containing http, and print out a report listing the file name and the links

   If none of these patterns are found then the script should let the user know.

   For extra credit, write this as one single script with a user menu, prompting for a list of file name(s) or types (*.html) and giving a menu of which pattern to search on.

   TEST YOUR SCRIPT WITH THE FOLLOWING FILE:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML>
<HEAD>
<!--THIS IS A TEST FILE-->

<TITLE>HOMEWORK EXAMPLE</TITLE>

</HEAD>
<BODY>

<H2 ALIGN=LEFT>A LIST OF IMAGES TO USE</H2>

<UL TYPE=DISC>
<LI><IMG SRC="./graphics/hotmoving.gif" ALIGN=MIDDLE WIDTH="79" HEIGHT="43" BORDER="1">
<LI><IMG SRC="./graphics/lighthousea.gif" ALIGN=MIDDLE WIDTH="41" HEIGHT="104" BORDER="1">
<LI><IMG SRC="./graphics/skull_xbones.gif" ALIGN=MIDDLE WIDTH="31" HEIGHT="30" BORDER="1">
</UL>
<P>

<H2 ALIGN=LEFT>A LIST OF LINKS TO USE</H2>
<MENU>
```

```
<LI><A HREF="./xfiles.html">X-Files</A>
<LI><A HREF="http://www.inet.net/web/users/hpages/yfiles.html">Y-Files</A>
<LI><A HREF="HTTP://www.south.com/home/einstein/index.html">Home Sweet Home Page</A>
<LI><A HREF="http://www.muon.com/ucla">UCLA Perl Course</A>
</MENU>

<!--THIS IS A COMMENT LINE-->

</BODY>
</HTML>
```

---

2. Mystery Script?

What does the following perl script do? Show examples of input and output.

```
#!/usr/local/bin/perl

@input = <STDIN>;
$count = 0;
while ($input[$count] ne "") {
        $input[$count] =~ s/^[ \t]+//;
        $input[$count] =~ s/[ \t]+\n$/\n/;
        $input[$count] =~ s/[ \t]+/ /g;
        $count++;
}
print ("Formatted text:\n");
print (@input);
#
#       end of script
#

__END__
```

---

3. Pattern Matching

A. Explain what patterns the following regular expression and match operators match. Give examples.

i. `/(..):(..):(..)/`

ii. `($x) = m|.*/([^/]+)$|;`
Hint: What value will `$x` have if `$_ = "/web/home/joe/hpages/index.html";`

iii. `/\b(\w+)(\s+\1)+\b/i`

iv. $string =~ s/(\w+)/\u$1/g;

B. Explain the difference between the following match operations:

i. /^feelfielfoe$/

ii. /^(feelfielfoe)$/

C. Explain what this little bit of Perl code does (hard, hint: it has something to do with HTML):

```
#!/usr/bin/perl -n -00

while ( /<\s*A\s+HREF\s*=\s*(["'])(.*?)\1.*?>/gi ) {

        print "$2\n";

}
```

---

4. REGEXP Patterns

(Don't forget to show me that these actually work!)

A. Write patterns that match the following:

   i. Three or more lower case letters (a-z).

   ii. Either the string four or the number 4.

   iii. Any letter followed by a vowel, followed by the same letter (e.g., sas).

B. Write the appropriate substitution command, including the necessary regular expression to perform the following substitutions:

   i. Replace all tabs with 4 spaces

   ii. Replace all tabs at the beginning of a line with 4 spaces

---

5. Renaming some files

You have a large number (too many to do this by hand) of files with names in the following form (note case!):

```
SMITHD.HTM
PARKERP.HTM
SAPPJ.HTM
```

Write a Perl script to rename each file so that the first and last character before the `dot (.)` is capitalized, all other letters are lowercase including the extension (`HTM`).

The above filenames would get converted to:

```
SmithD.htm
ParkerP.htm
SappJ.htm
```

Do some error checking and make sure you don't clobber any files! You should also consider away to load the file names by reading the directory.

Lecture 6: Regular Expression Special and Metacharacters

Single Characters and Digits

METACHARACTER	WHAT IS MATCHED
.	any character except newline
[a-z0-9]	any single character in set
[^a-z0-9]	any single character NOT in set
\d	any (one) digit [0-9]
\D	a non-digit, same as [^0-9]
\w	an alphanumeric (word) character
\W	a non-alphanumeric (word) character

Backslashed Characters

METACHARACTER	WHAT IS MATCHED
\s	whitespace character [\r\t\n\f]
\S	non-whitespace character [^\r\t\n\f]
\n	newline
\r	return
\t	tab
\f	formfeed
\b	backspace (in character class [])
\0	null character
\033	octal value
\xfa	hex value
\cD	control character D (^D or CNTL-D)
\[	given metacharacter

Anchored Characters

METACHARACTER	WHAT IS MATCHED
`\b`	word boundary (when not inside [])
`\B`	non-word boundary (inside word)
`^ or \A`	beginning of line or string only
`$ or \Z`	end of line or string only

Repeated Characters (Quantifiers)

METACHARACTER	WHAT IS MATCHED
`x?`	0 or 1 x
`x*`	0 or more x's
`x+`	1 or more x's
`x{m,n}`	at least m, x's and no more than n x's
`abc`	all of a, b and c, respectively
`dog|cat|mouse`	matches one of dog, cat or mouse

Remembered Characters

METACHARACTER	WHAT IS MATCHED
`(regexp)`	used for backreferencing
`\1 or $1`	first set of parentheses
`\2 or $2`	second set of parentheses
`\3 or $3`	third set of parentheses

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 6]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 6:
Perl REGULAR EXPRESSIONS

REGULAR EXPRESSIONS

Introduction

- What are Regular Expressions (regexp)?
- Why are they useful?

Regular Expressions

A sequence or pattern of characters that is matched against a string of text when performing searches and replacements. What qualifies as text is quite general, at least in Perl, giving the programmer a great deal of power.

Utility of Regular Expressions?

Regular expressions provide an extremely powerful tool for manipulating text as we shall see in this lecture. Perl provides matching, substitution and pattern binding operators that use regexps. This allows one to perform potentially difficult and time consuming operations quickly and easily.

MOTIVATION

Consider a web site with 50 different web page files, each of which contains a link to the same URL (e.g., your email address or your home page). What happens if this URL changes and you need to up-date all 50 files? What should you do?

i. Give up?
ii. Brute force it with your favorite editor (after making a large pot of coffee)?
iii. Buy a copy of *Unix Power Tools* and learn `sed` or `awk` (after drinking the large pot of coffee)?
iv. Run a one line Perl command from the command line, then go to a movie?

Yes, there are many ways to handle this, and clever computer wizards can come up with other possible solutions. However, once you have added Perl to your programming toolbox, you will not be quite so intimidated by these tasks.

EXAMPLE

Problem: You want to change the email address for your webmaster in all your HTML files. Here is a one line Perl script to handle the change, assuming all your files are in the current default directory. You know that the text you are looking for is of the form:

```
<ADDRESS>
```

```
Questions or Comments? <BR>
Email the <A HREF="mailto:webmaster@somewhere.com">Webmaster</A>
</ADDRESS>
```

The following script shows one example of how to change the email address:

```
# string we want to change, could have been read from STDIN
$str ='Email the <A HREF="mailto:webmaster@somewhere.com">Webmaster</A>';

# replacement string
$new ='<A HREF="mailto:jblow@webhead.com">WebMadMan</A>';

#substitution operation using regexp
$str =~ s#<A.+\/A>#$new#gi;
print "$str\n";

OUTPUT
Email the <A HREF="mailto:jblow@webhead.com">WebMadMan</A>
```

LEARNING REGULAR EXPRESSIONS

The utility of Perl's regular expressions lies with their use in pattern matching. Typically we have a *target pattern* that we would like to match (find) and possibly manipulate). The trick is to define a regexp to match our *target pattern* and hopefully only that pattern. The typical sequence of operations is:

- Define a regular expression based on the target pattern
- Use it with one of Perl's functions (e.g., split, etc.), or
- Use it with on of the pattern matching functions discussed in this lecture

Pattern

Sequence of characters to be searched for in a character string. Here are some simple examples of string literal patterns.

```
cat
3.14159
http://www.muon.com/
```

The next example uses the `split` function, and a pattern consisting of a single space, to search the scalar `$line` for all single spaces and break the string up based on this pattern.

```
$line = "Barbie ate the 3.14159";

@words  = split(/ /, $line);

foreach $word (@words) {
    print "$word\n";
}

OUTPUT

        Barbie
```

```
ate
the
3.14159
```

Operators Using Regular Expressions

- **Pattern Matching Operators**

There are several pattern matching operators in Perl. The table below provides some examples of how they function. Note that in their *bare* form they use the Perl special variable $_.

Operator	Example	Meaning
m/ /	m/pat/	True if $_ contains pattern "pat"
s/ / /	s/p/r/	Replace occurrences of 'p' with 'r' in $_
tr/ / /	tr/a-z/A-Z/	Translate all lower case to upper case characters in $_

- **Pattern Binding Operators**

By default Perl uses $_ for pattern searches. Perl allows you to specify the search string and return the results of the search operation. The **pattern binding operators** allow you to explicitly specify a different variable to search.

```
$var =~ m/regexp/;
$var !~ m/regexp/;
$var =~ s/old/new/;
```

`$var =~ /regexp/`
returns a value of `TRUE` if pattern `regexp` is found (matched), `FALSE` otherwise. Note that `$var` is not modified.

`$var !~ /regexp/`
returns `TRUE` if pattern `regexp` is **`NOT`** found, `FALSE` otherwise. Note that `$var` is not modified.

`$var =~ s/old/new/`
If a match is found for the pattern `old`, then the pattern `new` is substituted for `old` and the number of substitutions is returned, otherwise `FALSE` is returned. Only the first match will result in a substitution unless you use the `g option` switch (see below). Note that `$var` is modified since we are performing a substitution.

EXAMPLE

```
$line = "Barbie ate the 3.14159, and Barbie ate the cheese";

#perform a match
```

```
if ($line =~ m/cheese/) {
    print "Some one ate the cheese!\n";
} else {
    print "No one is eating cheese here!\n";
}

#perform a match

if ($line !~ m/chicken/) {
    print "No one is eating chicken here!\n";
} else {
    print "Some one is eating chicken!\n";
}

# perform a substitution

$line =~ s/Barbie/Ken/;
print "New line = $line \n";

OUTPUT
        Some one ate the cheese!
        No one is eating chicken here!
        New line = Ken ate the 3.14159, and Barbie ate the cheese
```

- **Pattern Binding Operator Precedence**

The pattern binding operators have precedence relative to all other Perl operators. By definition, `=~` and `!~` have higher precedence than `*` and `/`, and lower precedence than `**`.

- **Match Operator**

```
m/PATTERN/gimosx
 /PATTERN/gimosx      (the m is optional if you use the / / delimiters)
```

The match operator searches a string for `PATTERN`, and in a scalar context returns `TRUE` or `FALSE`, depending on whether the match succeeded or failed. If no string is specified via the `=~` or `!~` operators, then by default `$_` is searched. The `options`, appended after the final `/`, are shown in the table below.

`PATTERN` may contain variables, which will be interpolated (and the pattern recompiled) every time the pattern search is evaluated.

EXAMPLES

```
while (<STDIN>) {
    if (m/Sam/) {
        print;
    }
}
```

```
Sample I/O

The cat in the hat           #(user entered)
Sam I am                     #(user entered)
Sam I am                     #(program response)
My name is Samuel            #(user entered)
My name is Samuel            #(program response)
My name is sam               #(user entered)
Is it the Same thing         #(user entered)
Is it the Same thing         #(program response)
sdmbodneoinSam389jhkjf       #(user entered)
sdmbodneoinSam389jhkjf       #(program response)
```

Note: In this example the matches are case sensitive, and the matching string can appear anywhere! Also, the following is a one line (command line) version of the above example, taking input from the Unix password file.

```
perl -ne 'print if m/Sam/;' /etc/passwd
```

If / (slash) is the delimiter then the initial m is **optional**. With the m you can use any pair of non-alphanumeric, non-whitespace characters as delimiters. This is particularly useful for matching Unix path names that contain /, to avoid LTS (leaning toothpick syndrome). Note that the following are all equivalent:

```
m/http/
/http/
m#http#
```

However, unless its really necessary, I recommend sticking with the / delimiters.

MATCH OPERATOR OPTIONS

SWITCH	DESCRIPTION
g	Match globally, i.e. find all occurrences.
i	Do case-insensitive pattern matching.
m	Treat string as multiple lines.
o	Only compile pattern once.
s	Treat string as single line.
x	Use extended regular expressions.

EXAMPLE

```
perl -ne 'print "$. $_" if m/http/i' index.html
```

Search the index.html file for all lines with a URL containing http or HTTP or any case variation thereof, and print that line and corresponding line number.

Usually Perl will match under the assumption that the string contains only one line (no embedded newline characters). Embedded newlines will not be matched by ^ or $. You can use the m-option to tell Perl to treat the string as a multiline buffer, in which case ^ will match after any newline within the string and $ will match before any newline.

Example using the m-option.

```
$_ = <<EOT;
This is a story
about an animal.
The animal is a fox.
EOT

if ($result = m/^about/) {
   print "Match 1 = $&\n";
   print "Result 1 = $result\n";
} elsif ($result = m/^about/m){
   print "Match 2 = $&\n";
   print "Result 2 = $result\n";
} else {
   print "Sorry, no match\n";
}

OUTPUT

Match 2 = about
Result 2 = 1
```

- **Substitution Operator**

```
s/PATTERN/REPLACEMENT/egimosx
```

The substitution operator searches a string for PATTERN, and if found, replaces that pattern with REPLACEMENT and returns the number of substitutions made. Otherwise it returns FALSE.

If no string is specified via the =~ or !~ operator, the $_ variable is searched and modified. The string specified with =~ must be a scalar variable, an array element, a hash element, or an assignment to one of those.

If the delimiter chosen is single quote, no variable interpolation is done on either the PATTERN or the REPLACEMENT. Otherwise, if the PATTERN contains a **$** that looks like a variable rather than an end-of-string test, the variable will be interpolated into the pattern at run-time. If you only want the pattern compiled once the first time the variable is interpolated, use the /o option. If the pattern evaluates to a null string, the last successfully executed regular expression is used instead.

SUBSTITUTION OPERATOR OPTIONS

SWITCH	DESCRIPTION
e	Evaluate the right side as an expression.
g	Replace globally, i.e. all occurrences.
i	Do case-insensitive pattern matching.
m	Treat string as multiple lines
o	Only compile pattern once.
s	Treat string as single line.
x	Use extended regular expressions.

Any non-alphanumeric, non-whitespace delimiter may replace the slashes as delimiters. If single quotes are used, no interpretation is done on the replacement string (the `/e` modifier overrides this, however). If backquotes are used, the replacement string is a command to execute whose output will be used as the actual replacement text.

An `/e` will cause the replacement portion to be interpreted as a full-fledged Perl expression and `eval()`'d right then and there. It is, however, syntax checked at compile-time.

EXAMPLES

```
# globally replace all occurances of http with ftp in the file index.html
# regardless of case

perl -ne 's/http/ftp/gi; print;' index.html
```

- **Case Sensitive**

  Recall that Perl is case sensitive, so don't forget to use the `i` switch options when you want to invoke case insensitive matching and substitutions.

Perl Regular Expression Special Characters and Metacharacters

In order to enhance the functionality of the pattern matching and binding operators Perl uses a set of special characters and metacharacters. They allow you the ability to define very complex regexps for matching and substitution.

Metacharacters
alphanumeric characters, that when backslashed, have a special meaning, e. g., `\d, \w, \S, ...,`

```
etc..
```

Special characters

alphanumeric characters that have a special meaning unless they are backslashed, e. g., `\, |, (, ), [, {, ^, $, *, +, ?.`

For reference I have provided a set of tables that summarize the regular expression Meta- and Special Characters. Let's look at some of these and work through examples of how they are used. At first regular expressions, pattern matching and binding may seem confusing, but by working through lots of examples it will (hopefully) become clear.

Note also that if you peruse the news group comp.lang.perl.misc, you will notice that a large number of the questions and discussions are in someway related to Perl regular expressions. It is helpful to watch the experts create beautifully simple, yet powerful, regular expressions.

Perl Metacharacters, Special Characters and Regular Expression Examples

---

- **`+` character**

  match one or more of the preceeding characters; always attempts to match as many characters as possible.

```
$_ = "abbc";
/ab+/ matches abb   #"greedy" match

/ +/    #matches one or more spaces
```

Word count program (from *TYPERL5* , slightly modified), that uses a simple regexp to find words in a line of input.

```
#!/usr/local/bin/perl
#       simple word count program
#
$wordcount = 0;

print "Enter a line of text\n";
print "^D to terminate\n";
$line = <STDIN>;

while ($line ne "") {
    chomp ($line);
    @words = split(/ +/, $line);  #REGEXP used in split()
    $wordcount += @words;

    print "Enter a line of text\n";
    $line = <STDIN>;
}
```

```
print ("Total number of words: $wordcount\n");
```

Sample I/O

```
Enter a line of text
^D to terminate
This is a line of text.
Enter a line of text
This is another line of text.
Enter a line of text
More text.
Enter a line of text
Total number of words: 14
```

Are there problems with this simple example?

- **[ ] characters**

defines a pattern that matches one of a group of alternatives.

```
/[\t ]+/ matches one or more, and any combination of tabs (\t) or spaces


# want to match http:, or ftp:, or mailto:
# define a character class with one or more letters followed by a color (:)

if ( /[a-zA-Z]+:/) {
   print "Found it!\n";
}


$str="The rain in spain\t falls mainly on the plain.";
@result = $str =~ m/[ain\t]/g;
  $"= "-\n";
  print "Results\n@result\n";

OUTPUT
Results
a-
i-
n-
i-
n-
a-
i-
n-
        -
a-
a-
i-
n-
n-
a-
i-
```

n

- *** and ? characters**

  - x* matches zero or more occurances of x
  - x? matches zero or one occurances of x

    Watchout for the match on the zero occurrance, this can fool you. This is a match on the NULL string which occurs, for example, at the beginning of every string.

    The special characters +, *, ? and {} are examples of quantifiers. They are placed after the item being quantified and describe how many times to match that item.

    Perl quantifiers are *greedy* -- they match as much as they can as long as the entire regexp still matches. Regexps try to match as early in the string as possible, which takes precedence over being *greedy*, since scanning the string occurs from left-to-right. The pattern would therefore match as far left as possible even if there was somewhere else it could match longer.

    The ? can also be used to turn off greedy matching so that the regexp will try to perform a minimum rather than a maximum match.

EXAMPLES

```
$_ = "fox";
s/x?/ol/;
print;

OUTPUT
olfox     #huh?  The match was on the zero occurrance (NULL) of the x at the
          #beginning of the line!  Use x+ to match the first x.
```

- **Escape Sequences for Special Characters**

To include a character that Perl normally treats as a special or metacharacter you must use the backslash to escape the character.

```
To check for one or more occurances of * in a string use:  /\*+/
```

Perl 5 lets you escape a sequence using \Q. When the interpreter see the \Q, every character after the \Q is treated as a normal character until a \E is seen. An example of quoting metacharacters right in the match operator:

```
$str ="There are five stars in this line *****";
@result = $str =~ m/\Q*\E+/g;
print "@result\n";
```

```
OUTPUT
*****
```

The list `@result`, will have one element (`"******"`). What will `@result` contain if we used `@result = $str =~ m/\Q*\E/g;` instead?

* * * * *

• Matching any Letter or Number

Matching over a range of values can be accomplished using `[ ]` as follows:

```
/[0-9]/ match any digit
/[a-z]/ match any lowercase letter
/[A-Z]/ match any uppercase letter
```

Using the patterns discussed so far we can examine a program which checks for valid Perl variable names. This example if from *TYPERL5* , with slight modification.

```
#!/usr/local/bin/perl

print ("Enter a variable name:\n");
$varname = <STDIN>;
chomp ($varname);

if ($varname =~ /\$[A-Za-z][_0-9a-zA-Z]*/) {
        print ("$varname is a legal scalar variable\n");

} elsif ($varname =~ /[%|@][A-Za-z][_0-9a-zA-Z]*/) {
        print ("$varname is a legal array or hash variable\n");

} elsif ($varname =~ /[A-Za-z][_0-9a-zA-Z]*/) {
        print ("$varname is a legal file variable\n");

} else {
        print ("I don't understand what $varname is.\n");
}
```

```
INPUT/OUTPUT

Enter a variable name:
$var
$var is a legal scalar variable
Enter a variable name:
@var
@var is a legal array or hash variable
Enter a variable name:
%var
%var is a legal array or hash variable
Enter a variable name:
Myfile
Myfile is a legal file variable
```

```
Enter a variable name:
name$var
name$var is a legal scalar variable
```

DISCUSSION

Let's look at each of these regular expressions to see what they match, and don't match. Recall the definitions of value variable names in Perl: The must start with either a `$ (scalar)`, `@ (array or list)` or `% (hash)`, followed by a letter, followed by any number of underscores `(_)`, letters or digits.

`\$[A-Za-z][_0-9a-zA-Z]*`
: This pattern tries to match a scalar variable name since it tries to match a string that starts with the `$`, followed by any upper or lower case letter, followed by zero or more underscores, digits or letters. One problem, as we've seen above, with this regexp is that it does not force the `$` to be the start of a word so we get matches on substrings as shown above.

`[%|@][A-Za-z][_0-9a-zA-Z]*`
: This pattern tries to match an array or hash variable name since it tries to match a string that starts with `[%|@]`, followed by any upper or lower case letter, followed by zero or more underscores, digits or letters. One problem, as we've seen above, with this regexp is that it does not force the `@` to be the start of a word. Also notice that there is not match for a hash which starts with `%`.

`[A-Za-z][_0-9a-zA-Z]*`
: This pattern tries to match any name that contains letters followed by zero or more underscores, digits or letters. Again, since there is nothing restricting the search pattern to start on a word boundary, we will match any qualifying substring that fails the first two `if ()` tests, which is why this program thought `%hash` was a file name, since `hash` meets the test.

The last example used the string name$var which fooled the script. We can fix this using anchors.

- **Anchoring Patterns**

The pattern anchor metacharacters allow you to specify where you want to anchor (starting and ending position) your pattern matching. The anchor metacharacters are shown in the metacharacter summary tables, and repeated here as well.

An anchor lets you restrict where the pattern can match. Think of an anchor as a match to *nothing* (a technical term I borrowed from the Camel Book), that depends on where in the string it occurs. (Its called a *nothing* since its not really a character, but rather a context with zero width).

ANCHOR METACHARACTERS

METACHARACTER	WHAT IS MATCHED
\b	word boundary (when not inside [])
\B	non-word boundary (inside word)
^ or \A	beginning of line or string only
$ or \Z	end of line or string only

- **Word Boundary Pattern Anchors**

These specify whether a match is on a word boundary or inside a word boundary.

A word boundary is the beginning or end of word. A word boundary (\b) is defined as a spot between two characters that has a \w on one side of it and a \W on the other side of it (in either order), counting the imaginary characters off the beginning and end of the string as matching a \W.

Preface the pattern with \b if you want it to occur only at the beginning of a word, and postfix with \b to indicate the end of a word. \B matches only if the pattern is contained in the word. Preface the pattern with \B to match a pattern not at the beginning of a word and postfix the pattern with \B to match a pattern not at the end of a word.

EXAMPLES

```
/^Sam/ matches "Sam I am", but NOT "My name is Sam"

/Sam$/ matches "My name is Sam", but NOT "Sam I am"

/\bSam\b/ matches "Sam I am" but NOT "My name is Samuel"
```

- **Excluding Alternatives**

When the ^ character (caret) is the first character within [], it allows you to exclude certain patterns, i.e., to match any pattern except the one specified.

EXAMPLE

Consider trying to match a colon (:) separated data fields in a string. Assume you want to match the first field in each case.

```
$_ = 'sam:sdkoweinv:123:6543:Sam Gasster:/home/gasster:/bin/tcsh';

/.+:/;      #match one or more characters and the colon
print "Matched: $&\n";

/[^:]+:/;  #match one or more NON-colon characters up to the first colon
print "Matched: $&\n";
```

```
/[^:]+/;  #match one or more NON-colon characters up
print "Matched: $&\n";


OUTPUT
Matched: sam:sdkoweinv:123:6543:Sam Gasster:/home/gasster:
Matched: sam:
Matched: sam
```

Lets do a non-greedy match and compare with the above results:

```
$_ = 'sam:sdkoweinv:123:6543:Sam Gasster:/home/gasster:/bin/tcsh';
/.+?:/;     #match one or more characters and the colon
print "Matched: $&\n";

OUTPUT
Matched: sam:
```

- **Character Range Escape Sequences**

These are short hand expressions for the character range sequences defined earlier using `[]`. Note that `\w` matches a word character **NOT** an entire word; you can match an entire word using `\w+`.

ESCAPE SEQUENCE	WHAT IS MATCHED	Equivalent Range
\d	any digit	[0-9]
\D	any non-digit	[^0-9]
\w	any alphanumeric (word) character	[_0-9a-zA-Z]
\W	a non-alphanumeric (word) character	[^_0-9a-zA-Z]
\s	White Space	[\r\t\n\f]
\S	Anything other than white space	[^ \r\t\n\f]

EXAMPLES

```
/\d+/  matches one or more digits

/^\d?/  matches ZERO or one digits at beginning of string
        so both of these strings will match:
        "1313 Mocking Bird Lane"
        "Mocking Bird Lane"
```

- **Matching Any Character `(.)`**

The period (.) is used to match any character except the newline character. Use in conjunction with the * to match one or more characters. Normally, .* tries to match as much as possible.

```
$line = "Barbie ate the 3.14159, and Bobbie ate the cheese";

$line =~ /(c.*e)/;  #does the line contain a "c" followed by zero or more
                            #characters followed by an "e"?

$result = $1;
print "Match = $result\n";

OUTPUT
Match = cheese
```

Note that we also used (...) to invoke pattern match memory. Perl stores the text matched by each successive set of (...) into $1, $2,..., etc. which are only accessible within the current block and before the next match operation.

If you use the g option and using the pattern binding operator in an array context then you get:

```
$line = "Barbie ate the 3.14159, and Bobbie ate the cheese";

@match = $line =~ /c.*e/g;  #does the line contain a "c" followed by zero or more
                             #characters followed by an "e"?
print "Match = @result\n";

OUTPUT

Match = cheese
```

- **Matching a Specified Number of Occurances {}**

Perl allows you to specify how many occurances of a character constitute a match, using a minimum, maximum or exact range.

```
x{min,max} matches at least min x's and no more than max x's.

x{min,} matches only if min x's occur in the pattern, there is no maximum.

x{N} matches exactly N x's.

/.{0,7}/ will match at most 7 characters
```

NOTE: min, max and N must be integers less than 65,536.

EXAMPLES

```
/do{1,3}g/ will match dog, doog, dooog, but not dg or doooog

here's a little script to test this:
```

```
#!/usr/local/bin/perl
print "Enter a line of text\n";
$line = <STDIN>;
chomp($line);

while ($line ne "") {
    @result = $line =~ /(do{1,3}g)/g;
    print "\nResult = @result\n";
    print "Enter a line of text\n";
    $line = <STDIN>;
    chomp($line);
}
```

```
INPUT/OUTPUT

Enter a line of text
Do you have a dg, dog, doog, dooog or doooog?

Result = dog doog dooog
Enter a line of text
```

Since quantifiers can apply only to the single preceeding character we have:

```
/Ho{2}/    will match "Hoo", but not "HoHo"
/(Ho){2}/  will match "HoHo"
```

- **Specifying Choices (|)**

The character | let you specify two or more patterns to match:

```
/ REGEXP1 | REGEXP2 /

/Sam|Joe/       matches either "Sam" or "Joe"

/[0-9]+|[a-z]+/ matches one or more digits OR one or more lowercase letters
```

EXAMPLE from *TYPERL5*

This script is supposed to determine if the input is a valid Perl integer which can be of the form:

- Standard base 10 number: 1, 15, 1024
- Octal number representation: 04, 032, 0113
- Hexadecimal number representation: 0x5, 0x1ab, 0x1ff

```
#!/usr/local/bin/perl -w
#
print ("Enter a number:\n");
while ( $number = <STDIN> ) {
   chomp ($number);
if ($number =~ /^-?\d+$|^-?0[xX][\da-fA-F]+$/) {
        print ("$number is a legal integer.\n");
} else {
```

```
        print ("$number is not a legal integer.\n");
}
print ("Enter a number:\n");
}       #end of while
```

This script reads from STDIN (until you type ^D) and tests if the input matches the pattern specified. Let's examine the regular expression. The alternatives are:

^-?\d+$
: This pattern matches a string which can have 0 or 1 - signs at the beginning of the string followed by one or more digits, with only digits at the end. This pattern will match all base 10 and octal integers.

^-?0[xX][\da-fA-F]+$
: This pattern matches a string that can have 0 or 1 - signs at the beginning of the string, followed by a 0, followed by either x or X, followed by any one or more digits and any of the letters a through f (either upper or lower case). The $ ensures that only this string matches. This pattern takes care of hexidecimal integer representations.



Note the following pattern:

```
/x|y+/
```

matches either x **OR** one or more y, **NOT** one or more of x OR y!

- **Repeating or Reusing Patterns (Backreferences)**

Perl provides the capability to specify patterns that contain multiple repetitions of the same pattern. Given:

```
/ pat_a pat_b pat_a pat_b pat_a /
```

Use () to store the patterns into memory, then recall them using \n, where n is an integer representing the nth stored pattern. Thus this same sequence can be expressed as:

```
/(pat_a)(pat_b)\1\2\1/
```

The () cause whatever was matched to be remembered (stored) for latter use. Perl uses its pattern sequence memory to store the patterns, for later recall. However, note that it is best used when you want to match the **SAME** pattern in more than one place, but don't really care what characters you match.

The () are also used for grouping within a regexp. If you want to use a set of () only for grouping and don't want the side effect of backreferencing then use the following syntax:

```
(?:regexp)
```

EXAMPLE

Suppose you want to match on a date of the form: `dd-mm-yy`. You also don't care if the - are : and you will allow for single digit days and months.

The corresponding regexp and pattern matching operator is:

```
$pattern = '\d{1,2}(\W)\d{1,2}\1\d{2}';

while ($line = <STDIN>) {
     chomp($line);
     $result = $line =~ /$pattern/;
     print "match = $& \n";
     print "Result = $result\n";
}
```

```
INPUT/OUTPUT

Friday is 13:04:96
match = 13:04:96
Result = 1
Thursday is 15-12-959
match = 15-12-95
Result = 1
123-15-9876
match = 23-15-98
Result = 1
```

- `\d{1,2}` -- this will match 1 or 2 digits
- `(\W)` -- matches any non-alphanumeric character and stores result into `\1`
- `\d{1,2}\1\d{2}` -- matches 1 or 2 digits followed by the same pattern that matched `(\W)` above, followed by exactly 2 digits.

Note that one problem with this regexp is that it will pull out what appears to be a date from a social security number! How can we fix this?



Consider the following pattern:

```
[\da-z]+[:;][\da-z]+[:;][\da-z]+
```

which you might be tempted to write as:

```
([\da-z]+)([:;])\1\2\1
```

However, the pattern matched by the first `[\da-z]+`, is stored in the pattern memory `$1` and therefore only that same pattern will be match by the other two `\1`. So `15:15:15` would match but `15:16:17` would not.

- **Special-Character Precedence**

When you start using `regexps` with a variety of patterns and metacharacters the Perl interpreter (and you) will need to know in what order to assign precedence as it tries to evaluate the expression. The table below summarizes these rules.

Metacharacter or Representation	Description
`()`	Parentheses
`+ * ? {}`	Multipliers or Number or Occurances
`xyz ^ $ \b \B`	Sequences and Pattern Anchors
`|`	Alternatives

Remember that you can use `()` to force precedence, but they also store the pattern into memory.

EXAMPLE: Date Validation Program from *TYPERL5* .

```
#!/usr/local/bin/perl

print ("Enter a date in the format YYYY-MM-DD:\n");
$date = <STDIN>;
chomp ($date);

# Because this pattern is complicated, we split it
# into parts, assign the parts to scalar variables,
# then substitute them in later.

# handle 31-day months
$md1 = "(0[13578]|1[02])\\2(0[1-9]|[1-2]\\d|3[01])";

# handle 30-day months
$md2 = "(0[469]|11)\\2(0[1-9]|[1-2]\\d|30)";

# handle February, without worrying about whether it's
# supposed to be a leap year or not
$md3 = "02\\2(0[1-9]|[1-2]\\d)";

# check for a twentieth-century date
$match = $date =~ /^(19)?\d\d(.)($md1|$md2|$md3)$/;

# check for a valid but non-20th century date
$olddate = $date =~ /^(\d{1,4})(.)($md1|$md2|$md3)$/;

if ($match) {
        print ("$date is a valid date\n");
} elsif ($olddate) {
```

```
        print ("$date is not in the 20th century\n");
} else {
        print ("$date is not a valid date\n");
}
__END__
```

Pattern System Variables

Pattern system variables are set by the pattern matching operator and other operators that use pattern matching (e.g., `s///`).

- **Defined only in the current BLOCK**
- **Retrieving Matched Subpatterns `$n` and `$+` (`$LAST_PAREN_MATCH`)**
  When you use `()` to enclose a pattern or subpattern, Perl stores the results of that match into the pattern memory. Within the regular expression these values are accessed using `\n`, corresponding to the `nth` set of `()`, `n=0,1,...`. These values are also accessible outside of the pattern matching operator using the system variables `$n`.

  In patterns, parentheses are counted from left to right. This rule tells Perl how to handle nested `()`.

  The `$+` variable stores the last subpattern enclosed in parentheses that results in a match. This variable is useful if you are only interested in part of a pattern.
- **Retrieving the Entire Pattern: `$&` (`$MATCH`)**
  When the pattern is successfully matched, the matched text string is stored in the system variable `$&`. Remember, `$&` is only defined within the block containing the match operator. If you need the results of the match outside of this block save it to another variable.
- **Retrieving the unmatched text: `$`` (`$PREMATCH`) and `$'` (`$POSTMATCH`)**
  When the pattern is successfully matched, the matched text string is stored in the system variable `$&`, and the rest of the text is store in two other system variables.
  - `$`` stores the unmatched text preceeding the matched text
  - `$'` stores the unmatched text following the matched text

Example

```
$pattern = '\d{1,2}(\W)\d{1,2}\1\d{2}';

while ($line = <STDIN>) {
chomp($line);

$result = $line =~ /$pattern/;

print "match = $& \n";
print "Not matched = $` \n $' \n";
print "\nResult = $result\n";
}
```

Input

```
        My birthday is 4-15-55, and that was a good year!

Output

match = 4-15-55

Not matched = My birthday is
 , and that was a good year!

Result = 1
```

The Translation Operator, `tr/ / /`

`tr/SEARCHLIST/REPLACEMENTLIST/cds`

Translates all occurrences of the characters found in the search list with the corresponding character in the replacement list. It returns the number of characters replaced or deleted.

If no string is specified via the =~ or !~ operator, the $_ string is translated. (The string specified with =~ must be a scalar variable, an array element, or an assignment to one of those.)

OPTIONS FOR THE `tr/ / /` FUNCTION

OPTION	DESCRIPTION
c	Complement the SEARCHLIST.
d	Delete found but unreplaced characters.
s	Squash duplicate replaced characters.

If the /c modifier is specified, the SEARCHLIST character set is complemented. If the /d modifier is specified, any characters specified by SEARCHLIST not found in REPLACEMENTLIST are deleted. (Note that this is slightly more flexible than the behavior of some tr programs, which delete anything they find in the SEARCHLIST, period.) If the /s modifier is specified, sequences of characters that were translated to the same character are squashed down to a single instance of the character.

If the /d modifier is used, the REPLACEMENTLIST is always interpreted exactly as specified. Otherwise, if the REPLACEMENTLIST is shorter than the SEARCHLIST, the final character is replicated till it is long enough. If the REPLACEMENTLIST is null, the SEARCHLIST is replicated. This latter is useful for counting characters in a class or for squashing character sequences in a class.

EXAMPLES

```
$ARGV[1] =~ tr/A-Z/a-z/;     # convert to lower case

$cnt = tr/*/*/;              # count the stars in $_

$cnt = $sky =~ tr/*/*/;      # count the stars in $sky
```

```
$cnt = tr/0-9//;            # count the digits in $_

tr/a-zA-Z//s;               # bookkeeper -> bokeper

tr/a-zA-Z/ /cs;             # change non-alphas to single space
```

The last example would yield:

```
$_ = <<EOI;
The\train\tin\t\tspain
falls \non \nyour head.
EOI
    print;
    tr/a-zA-Z/ /cs;
    print;

OUTPUT
The     rain    in              spain
falls
on
your head.
The rain in spain falls on your head
```

If multiple translations are given for a single character, only the first one is used:

```
    tr/AAA/XYZ/
```

will translate any A to X.

STRING MANIPULATION FUNCTIONS RELATED TO REGEXPs

grep BLOCK LIST
grep EXPR,LIST

Evaluates the BLOCK or EXPR for each element of LIST (locally setting $_ to each element) and returns the list value consisting of those elements for which the expression evaluated to TRUE. In a scalar context, returns the number of times the expression was TRUE. Note that the Perl `grep` function is named after, and is similar to the same Unix function.

```
    @result = grep(!/^#/, @input);     # weed out comments
```

or equivalently,

```
    @result = grep {!/^#/} @input;     # weed out comments
```

In the first example above the search pattern to match is specified by the regexp `!/^#/`, which means do not match any line beginning with a # character, which signifies comments in Perl. Thus the array `@input` is searched for all lines NOT starting with a comment, and the results stored in the array `@results`.

EXAMPLE: Using `grep` with File-Test Operators

This example is from Day 14 of *TYPERL 5* . In this example we search all readable files in the current directory for a particular word.

```
#!/usr/local/bin/perl
#
#  search all readable files in the current
#  directory for the word specified on the
#  command line
#
#    file: fword.pl
#
$searchword = $ARGV[0];  # search word from ARGV
print "The search word is: $searchword\n";

opendir(CURRDIR, ".") or
        die("Can't open current directory");

@filelist = grep (!/^\./, grep(-r, readdir(CURRDIR)));

closedir(CURRDIR);

foreach $file (@filelist) {
    open (CURRFILE, $file) or
        die ("Can't open input file $file");
        while ($line = <CURRFILE>) {
            if ($line =~ /$searchword/) {
                print ("$file:$.: $line");
            }
        }
    close (CURRFILE);
}

INPUT/OUTPUT

gasster> fword.pl '\$text'
The search word is: \$text
Lec2ex.pl:14: $text = "This is my first line.";
Lec2ex.pl:15: print $text,"\n";
Lec2ex.pl:16: print "$text \n";
Lec2ex.pl:17: print STDOUT $text,"\n";
Lec2ex.pl:19: $_ = $text;
```

---

pos

The `pos` function will find or set the offset for the last/next `m//g` search. Returns the offset of where the last `m//g` search left off for the variable in question. May be modified to change that offset. Only defined in Perl 5.

```
$offset = pos($string);
```

`$string` is the string whose pattern is currently being matched, `$offset` is the number of characters

already matched or skipped.

EXAMPLES

```
#!/usr/local/bin/perl

$string = "Mississippi";

while ($string =~ /i/g) {
     $position = pos($string);
     print("matched at position $position\n");
}

OUTPUT

matched at position 2
matched at position 5
matched at position 8
matched at position 11
```

The first i is in the second character position, so the returned offset is 2. Thus the number displayed is the number of characters offset into the string where the pattern matching operation will resume (in the `while` loop).

You can also set the position where the pattern match will start (resume) by assigning to the `pos` function as follows:

---

study SCALAR
study

The study function tells Perl to optimize input data for repeated searches. Takes extra time to study SCALAR (`$_` if unspecified) in anticipation of doing many pattern matches on the string before it is next modified.

This may or may not save time, depending on the nature and number of patterns you are searching on, and on the distribution of character frequencies in the string to be searched--you probably want to compare run times with and without it to see which runs faster. Those loops which scan for many short constant strings (including the constant parts of more complex patterns) will benefit most. You may have only one study active at a time--if you study a different scalar the first is *unstudied* . (See the Perl manual for how study does its thing!)

EXAMPLE

You should play around with the study function and various pattern matches to see if it makes much difference. Modify this script to read in lots of text and try different patterns. Your mileage may vary (as they say!).

```
$pattern = '\d+';

@howlong = times;  # use times function to see if
                   # study makes a difference
```

```
print "Enter a string to search\n";
$str =<STDIN>;

study ($str);       # comment this out and run again for
                    # comparison

while ($str =~ /$pattern/g) {
     $position = pos($str);
     print("matched at position $position\n");
}
printf "Time: %2.4f\n",$howlong[0];
print "$str\n";

INPUT/OUTPUT
#study on
Enter a string to search
The clock struck 1 and down he run 5 times.
matched at position 18
matched at position 36
Time: 0.4500
The clock struck 1 and down he run 5 times.

#study off
Enter a string to search
The clock struck 1 and down he run 5 times.
matched at position 18
matched at position 36
Time: 0.6833
The clock struck 1 and down he run 5 times.
```

---

quotemeta

The `quotemeta` will quote (escape) regular expression metacharacters. Returns the value of `$string` with all regular expression metacharacters backslashed. This is the internal function implementing the `\Q` escape in double-quoted strings.

```
$newstring = quotemeta($string);
```

EXAMPLE

```
The following two statements are equivalent:

$string = quotemeta($string);

$string =~ s/(\W)/\\$1/g;

INPUT/OUTPUT

Enter a string to search
$five + 3 = 8.
\$five\ \+\ 3\ \=\ 8\.\

\$five\ \+\ 3\ \=\ 8\.\
```

```
Enter a string to search
I like this pattern: [^\d\r$]
I\ like\ this\ pattern\:\ \[\^\\d\\r\$\]\

I\ like\ this\ pattern\:\ \[\^\\d\\r\$\]\
```

In the substitution operator, the `(\W)` matches any non-word characters and loads the matches into the pattern memory. The pattern memory match is recalled using `$1` and then each match is replaced by that match preceded with a backslash (`\\`).

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 6]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

PERL5 Regular Expression Description

From: Tom Christiansen
Newsgroups: comp.lang.perl.misc,comp.lang.misc,comp.lang.tcl,comp.lang.pyth= on
Subject: Irregular Expressions
Date: 13 Feb 1996 19:15:38 GMT
Organization: Perl Consulting and Training

All right, Steve, I'll try to give this one some thought. Since you seem to be a local, if I've answered your question sufficiently, feel free to treat me to a pint of Colorado Kind Ale at the Mountain Sun brewery on Pearl Street in Boulder. :-)

> **WARNING:** This article contains more punctuation that many non-programmers like. That's because regexps and finite automata are concise ways of expressing powerful concepts. They are not for the faint of heart. Don't blame Perl if you don't like regular expressions. And don't blame Thompson or anyone else. If you don't like them, don't use them. But don't despise those of us who do.

Here goes...

Why is Perl so useful for sysadmin and WWW and text hacking? It has a lot of nice little features that make it easy to do nearly anything you want to text. A lot of perl programs look like a weird synergy of C and shell and sed and awk. For example:

```
#!/usr/bin/perl
# manpath -- tchrist@perl.com
foreach $bindir (split(/:/, $ENV{PATH})) {
    ($mandir =3D $bindir) =3D~ s/[^\/]+$/man/;
    next if $mandir =3D~ /^\./ || $mandir eq '';
    if (-d $mandir && ! $seen{$mandir}++ ) {
        ($dev,$ino) =3D stat($mandir);
        if (! $seen{$dev,$ino}++) {
            push(@manpath,$mandir);
        }
    }
}
print join(":", @manpath), "\n";
```

Can anyone see what that does? I'd like to think it's not too hard, even devoid of commentary. It does have some naughty bits, like using side effect operators of assignment operators as expressions and double-plus postfix autoincrement. C programmers don't have a problem with it, but a lot of others do. That's why Guido banned such things in Python (a rather nice language in many ways), and why I don't advocate using them to non-C programmers, whom it generally confuses whether it be done in C or in Perl or C++ or any such language.

By far the most bizarre thing is that dread punctuation lying within funny slashes. Often folks call Perl unreadable because they don't grok regexps, which all true perl wizards -- and acolytes -- adore. The slashes and their patterns govern matching and splitting and substituting, and here is where a lot of the Perl magic resides: its unmatched :-) regular expressions. Certainly the above code could be rewritten in tcl or python or nearly anything else. It could even be rewritten in more legible perl. :-) =

So what's so special about perl's regexps? Quite a bit, actually, although the real magic isn't demonstrated very well in the manpath program. Once you've read the perlre(1) and the perlop(1) man pages, there's still a lot to talk about. So permit me, if you would, to now explain Far More Than Everything You Ever Wanted to

Know about Perl Regular Expressions... :-)

Perl starts with POSIX regexps of the *"modern"* variety, that is, *egrep* style not *grep* style. Here's the simple case of matching a number

```
/Th?om(as)? (Ch|K)rist(ia|e)ns{1,2}s[eo]n/
```

This avoids a lot of backslashes. I believe many languages also support such regular rexpressions.

Now, Perl's regexps "aren't" -- that is, they aren't "regular" because backreferences per sed and grep are also supported, which renders the language no longer strictly regular and so forbids "pure" DFA implementations.

But this is exceedingly useful. Backreferences let you refer back to match part of what you just had. Consider lines like these:

```
1. This is a fine kettle of fish.
2. The best card is Island Fish Jasconius.
3. Is isolation unpleasant?
4. That's his isn't it?
5. Is is outlawed?
```

If you'd like to pick up duplicate "*is*" strings there, you = could use the pattern

```
/(is) \1/           # matches 1,4
```

As written, that will match sentences 1 and 4. The others fail due to mixed case. You can't fix it just by saying

```
/([Ii]s) \1/        # still matches 1,4
```

because the \1 refers back to the real match, not the potential match. So what do we do? Well, POSIX specifies a `REG_ICASE` flag you can pass into your matcher to help support "`grep -i`" etc. To get perl to do this, affix an `i` flag after the match:=

```
/(is) \1/i          # matches 1,2,3,4,5
```

And now all 5 of those sentences match. If you only wanted them to match legit words, you might use the `\b` notation for word boundaries, making it

```
/\b(is) \1/i        # matches 2,3,5
/(is) \1\b/i        # matches 1,5
/\b(is) \1\b/i      # matches 5
```

This means you will see Perl code like

```
if ( $variable =3D~ /\b(is) \1\b/i ) {
    print "gotta match";
} =
```

One might argue that is "should" be written more like

```
if ( rematch(variable, '\b(is) \1\b', 'i') ) {
    print "gotta match";
} =
```

but that's not how Perl works. I suspect that other languages could make it work that way.

If you'd like to know where you matched, you might want to use these:

```
$MATCH                  full match
$PREMATCH               before the match
$POSTMATCH              after the match
$LAST_PAREN_MATCH       useful for alternatives
```

Although the most normal case is just to use `$1`, `$2`, etc, which match the first, second, etc parenthesized subexpressions.

Another nice thing that Perl supports are the notions from C's `ctype.h` include file:

```
C function      Perl regexp

isalnum         \w
isspace         \s
isdigit         \d
```

That means that you don't have to hard-code `[A-Z]` and have it break when someone has some interesting locale settings. For example, under charset=3DISO-8859-1, something like "fa=C1ade" properly matches `/^\w+$/`, because the c-cedille is considered an alphanum. In theory, `LC_NUMERIC` settings should also take, but I've never tried.

This quickly leads to a pattern that detects duplicate words in = sentences:

```
/\b(\w+)(\s+\1)+\b/i
```

In fact, that one matches multiple duplicates as well. If = if if you read in your input data a paragraph at a time, it will = catch dups crossing line boundaries as as well. For example, using some convenient command line flags, here's a =

```
perl -00 -ne 'if ( /\b(\w+)(\s+\1)+\b/i ) { print "dup $1 at $.\n" }'
```

which when used on this article says:

```
    =

dup Is at 10
dup If at 33
```

the `$.` variable (`$NR` in English mode) is the record number. I set it to read paragraph records, so paragraphs 10 and 33 of this posting contain duplicate words.

Actually, we can do something a bit nicer: we can find multiple duplicates in the same paragraph. The `/g` flag causes a match to store a= bit of state and start up where it last left off. This gives us:

```
#!/usr/bin/perl -00 -n
while ( /\b(\w+)(\s+\1)+\b/gi ) { =

    print "dup $1 at paragraph $.\n";
}
```

This now yields:

```
dup Is at paragraph 10
dup if at paragraph 33
dup as at paragraph 33
```

Of course, we're getting a bit hard to read here. So let's use the `/x` flag to permit embedded white space and comments in our= pattern -- you'll want 5.002 for this (the white space worked in 5.000, but the comments were added later :-). For legibility, instead of slashes for the match, I'll embrace the real `m()` function, Since `/foo/` and `m(foo)` and `m{foo}` are all equivalent.

```
    #!/usr/bin/perl -n
    require 5.002;
    use English;
    $RS =3D '';

    while ( =

       m{                       # m{foo} is like /foo/, but helps vi's % ke=
y

                \b              # first find a word boundary

                (\w+)           # followed by the biggest word we can find
                                # which we'll save in the \1 buffer
                (               =

                  \s+           # now have some white space following it
                  \1            # and the word itself
                )+              # repeat the space+word combo ad libitum

                \b              # make sure there's a boundary at the end t=
oo

        }xgi                    # /x for space/comment-expanded patterns
                                # /g for global matching
                                # /i for case-insensitive matching
    )
    { =

        print "dup $1 at paragraph $NR\n";
    }
```

While it's true that someone who doesn't know regular expressions won't be able to read this at first glance, this is not a problem. So even though we can build up rather complex patterns, we can = format and comment them nicely, preserving understandability. I wonder why no one else has done this in their regexp libraries?

I actually wrote a sublegible version of this many years ago. It runs even on ancient versions of Perl. I'd probably to that a bit differently these days -- my coding style has certainly matured. It violates several of my own current style guidelines.

```
    #!/usr/bin/perl
    undef $/; $* =3D 1;
    while ( $ARGV =3D shift ) { =

        if (!open ARGV) { warn "$ARGV: $!\n"; next; } =

        $_ =3D <ARGV>;
        s/\b(\s?)(([A-Za-z]\w*)(\s+\3)+\b)/$1\200$2\200/gi || next;
        split(/\n/);
        $NR =3D 0;
        @hits =3D ();
        for (@_) { =

            $NR++;
```

```
            push(@hits, sprintf("%5d %s", $NR, $_)) if /\200/;
        } =

        $_ =3D join("\n",@hits);
        s/\200([^\200]+)\200/[* $1 *]/g;
        print "$ARGV:\n$_\n";
    }
```

here's that will output when run on this article up to this current point:

```
  51     5. [* Is is *] outlawed?
 124 In fact, that one matches multiple duplicates as well.  [* If =

 125 if if *] you read in your input data a paragraph at a time, it will =

 126 catch dups crossing line boundaries [* as as *] well.  For example, u=
sing
```

Which is pretty neat. =

Speaking of ctype.h macros, Perl borrows the *vi* notation of case translation via `\u`, `\l`, `\U`, and `\L`.= So you could say

```
    $variable =3D "fa=C1ade ni=D2o co=96perate moli=CBre ren=C8e na=D4ve h=CA=
mo tsch=B8=FE";
and then do a =


    $variable =3D~ s/(\w+)/\U$1/g;
```

and it would come out

```
    FA=ABADE NI=8BO CO=F7PERATE MOLI=BBRE REN=8AE NA=A6VE H=90MO TSCH=D0=FE=
```

Oh well. My clib doesn't know to turn `=FE` -> `SS`.= That's a harder issue.

This is much better than writing things like

```
    $variable =3D~ tr[a-z][A-Z];
```

because that would give you:

```
    FA=C1ADE NI=D2O CO=96PERATE MOLI=CBRE REN=C8E NA=D4VE H=CAMO TSCH=B8=FE=
```

which isn't right at all.

Actually, perl can beat *vi* and do this:

```
    $variable =3D~ s/(\w+)/\u\L$1/g;
```

Yielding:

```
    Fa=C1ade Ni=D2o Co=96perate Moli=CBre Ren=C8e Na=D4ve H=CAmo Tsch=B8=FE=
```

which is somewhat interesting.

Speaking of substitutes, we can use a `/e` flag on the substitut= e to get the RHS to evaluate to code instead of just a string. = Consider:

```
s/(\d+)/8 * $1/ge;              # multiple all numbers by 8
s/(\d+)/sprintf("%x", $1)/ge;   # convert them to hex
```

This is nice when renumbering paragraphs. I often write

```
s/^(\d+)/1 + $1/
```

or from within *vi*, just

```
%!perl -pe 's/^(\d+)/1 + $1/'
```

Here's a more elaborate example of this. = If you wanted to expand `%d` or `%s` or whatnot, you might just do

```
s/%(.)/$percent{$1}/g;
```

given a `%percent` definition like this:

```
%percent =3D (
            'd'     =3D> 'digit',
            's'     =3D> 'string',
);
```

But in fact, that's got quite enough. You might well want to call a function, like

```
s/%(.)/unpercent($1)/ge;
```

(assuming you have an `unpercent()` function defined.)

You can even use `/ee` for a double-eval, but that seems going overboard in most cases. It is, however, nice for converting embedded variables like `$foo` or whatever in text into their values. This way a sentence with `$HOME` and `$TERM` in it, assuming there were valid variables, might become a sentence with */home/tchrist* and *xterm* in it. Just do this:

```
s/(\$\w+)/$1/eeg;
```

Ok, what more can we do with perl patterns? `split` takes a pattern. Imagine that you have a record stored in plain text as blank line separated paragraphs with FIELD: VALUE pairs on each line.

```
field: value here
somefield: some value here
morefield: other value here

field: second record's value here
somefield: some value here
morefield: other value here
newfield: other funny stuff
```

You could process that this way. We'll put it into key value pairs in a hash, just as though it had been initialized as

```
%hash =3D (
    'field'         =3D> 'value here',
    'somefield'     =3D> 'some value here',
    'morefield'     =3D> 'other value here',
);
```

I'll use a few command line switches for short cuts:

```
#!/usr/bin/perl -00n
%hash =3D split( /^([^:]+):\s*/m );
if ( $hash{"somefield"} =3D~ /here/) {
    print "record $. has here in somefield\n";
} =
```

The /m flag governs whether ^ can match internally. I believe this is the POSIX value REG_NEWLINE. Normally perl does not have ^ match anywhere but the beginning of the string. (

Or you could eschew shortcuts and write:

```
#!/usr/bin/perl =

use English;
$RS =3D '';
while ( $line =3D <ARGV> ) {
    %hash =3D split(/^([^:]+):\s*/m, $line);
    if ( $hash{"somefield"} =3D~ /here/) {
        print "record $NR has here in somefield\n";
    } =

} =
```

Actually, in the current version of perl, you can use the getline() object method on the predefined ARGV file handle object:

```
#!/usr/bin/perl =

use English;
$RS =3D '';
while ( $line =3D ARGV->getline() ) {
    %hash =3D split(/^([^:]+):\s*/m, $line);
    if ( $hash{"somefield"} =3D~ /here/) {
        print "record $NR has here in somefield\n";
    } =

} =
```

This can be especially convenient for handling mail messages. =

Here, for example, is a bair-bones mail-sorting program:

```
#!/usr/bin/perl -00
while (<>) {
    if ( /^From / ) {
        ($id) =3D /^Message-ID:\s*(.*)/mi;
        $sub{$id} =3D /^Subject:\s*(Re:\s*)*(.*)/mi =

                        ? uc($2)
                        : $id;
    } =

    $msg{$id} .=3D $_;
}
```

```
print @msg{ sort { $sub{$a} cmp $sub{$b} } keys %msg};
```

Now, I still haven't mentioned a couple of features which = are to my mind critical in any analysis of the strengths of Perl's pattern matching. These are stingy matching and lookaheads. =

Stingy matching solves the problem greedy matching. A greedy match picks up everything, as in:

```
$line =3D "The food is under the bar in the barn.";
if ( $line =3D~ /foo(.*)bar/ ) {
    print "got <$1>\n";
}
```

That prints out

```
<d is under the bar in the >
```

Which is often not what you want. Instead, we can add an extra `? =` after a repetition operator to render it stingy instead of greedy.

```
if ( $line =3D~ /foo(.*?)bar/ ) {
    print "got <$1>\n";
}
```

That prints out

```
got <d is under the >
```

which is often more what folks want. It turns out that having both stringy and greedy repetition operators in no way compromises a = regexp engines regularity, nor is it particularly hard to implement. This comes up in matching quoted things. You can do tricks like using [^:] or [^"] or [^"'] for the simple cases, but negating multicharacter strings is hard. You can just use stingy matching instead.

Or you could just use lookaheads.

This is other important aspect of perl matching I wanted to mention. These are 0-width assertions that state that what follows must match or must not match a particular thing. These are phrased as either (?=3Dpattern) for the assertion or (?!pattern) for the negation.

```
/\bfoo(?!bar)\w+/
```

That will match "*foostuff*" but not "*foo*" or "*foobar*", because I said there must be some alphanums after the word *foo*, but these may = not begin with *bar*.

Why would you need this? Oh, there are lots of times. Imagine splitting on newlines that are not followed by a space or a tab:

```
@list_of_results =3D split ( /\n(?![\t ])/, $data );
```

Let's put this all together and look at a couple of examples. Both have to do with HTML munging, the current rage. First, let's solve the problem of detecting URLs in plaintext and highlighting them properly. A problem is if the URL has trailling punctuation, like *ftp://host/path.file.* = Is that last dot supposed to be in the URL? We can probably just assume that a trailing dot doesn't count, but even so, most scanners seem to get this wrong. Here's a different approach:

```
#!/usr/bin/perl =

# urlify -- tchrist@perl.com
```

```
  require 5.002;  # well, or 5.000 if you see below
  =

  $urls =3D '(' . join ('|', qw{
                  http
                  telnet
                  gopher
                  file
                  wais
                  ftp
              } ) =

        . ')';
  =

  $ltrs =3D '\w';
  $gunk =3D '/#~:.?+=3D&%@!\-';
  $punc =3D '.:?\-';
  $any  =3D "${ltrs}${gunk}${punc}";
  =

  while (<>) {
      ## use this if early-ish perl5 (pre 5.002)
      ##  s{\b(${urls}:[$any]+?)(?=3D[$punc]*[^$any]|\Z)}{<A HREF=3D"$1">$1=
</A>}goi;
      ## otherwise use this -- it just has 5.002ish comments
      s{
        \b                          # start at word boundary
        (                           # begin $1  {
          $urls     :               # need resource and a colon
          [$any] +?                 # followed by on or more
                                    #  of any valid character, but
                                    #  be conservative and take only
                                    #  what you need to....
        )                           # end   $1  }
        (?=3D                         # look-ahead non-consumptive assertio=
n
                [$punc]*            # either 0 or more puntuation
                [^$any]             #   followed by a non-url char
            |                       # or else
                $                   #   then end of the string
        )
      }{<A HREF=3D"$1">$1</A>}igox;
      print;
  }
```

Pretty nifty, eh? :-)

Here's another HTML thing: we have an html document, and we want to remove all of its embedded markup text. This requires three steps:

1) Strip <!-- html comments -->
2) Strip <TAGS>
3) Convert &entities; into what they should be.

This is complicated by the horrible specs on how html comments work: = they can have embedded tags in them. So you have to be way more careful. But it still only takes three substitutions. :-) I'll use the /s flag to make sure that my "." can stretch to match = a = newline as well (normally it doesn't).

```
    #!/usr/bin/perl -p0777
```

```
#
############################################################
# striphtml ("striff tummel")
# tchrist@perl.com =

# version 1.0: Thu 01 Feb 1996 1:53:31pm MST =

# version 1.1: Sat Feb  3 06:23:50 MST 1996
#           (fix up comments in annoying places)
############################################################
#
# how to strip out html comments and tags and transform
# entities in just three -- count 'em three -- substitutions;
# sed and awk eat your heart out.  :-)
#
# as always, translations from this nacr=C8 rendition into =

# more characteristically marine, herpetoid, titillative, =

# or indonesian idioms are welcome for the furthering of =

# comparitive cyberlinguistic studies.
#
############################################################

require 5.001;      # for nifty embedded regexp comments

############################################################
# first we'll shoot all the <!-- comments -->
############################################################

s{ <!                   # comments begin with a `<!'
                        # followed by 0 or more comments;

    (.*?)           # this is actually to eat up comments in non =

                        # random places

     (                  # not suppose to have any white space here

                        # just a quick start; =

      --                # each comment starts with a `--'
        .*?             # and includes all text up to and including
      --                # the *next* occurrence of `--'
        \s*             # and may have trailing while space
                        #   (albeit not leading white space XXX)
     )+                 # repetire ad libitum  XXX should be * not +
    (.*?)           # trailing non comment text
   >                    # up to a `>'
}{
    if ($1 || $3) { # this silliness for embedded comments in tags
        "<!$1 $3>";
    } =

}gesx;                  # mutate into nada, nothing, and niente
=0C
############################################################
# next we'll remove all the <tags>
############################################################
```

```
    s{ <                    # opening angle bracket

        (?:                 # Non-backreffing grouping paren
             [^>'"] *       # 0 or more things that are neither > nor ' nor=
 "
                |           #    or else
             ".*?"          # a section between double quotes (stingy match=
)
                |           #    or else
             '.*?'          # a section between single quotes (stingy match=
)
        ) +                 # repetire ad libitum
                            #  hm.... are null tags <> legal? XXX
       >                    # closing angle bracket
    }{}gsx;                 # mutate into nada, nothing, and niente

    ###########################################################
    # finally we'll translate all &valid; HTML 2.0 entities
    ###########################################################

    s{ (
            &               # an entity starts with a semicolon
            ( =

                \x23\d+     # and is either a pound (# =3D=3D hex 23)) and n=
umbers
                 |          #   or else
                \w+         # has alphanumunders up to a semi
            )         =

            ;?              # a semi terminates AS DOES ANYTHING ELSE (XXX)
        )
    } {

        $entity{$2}         # if it's a known entity use that
            ||              #   but otherwise
            $1              # leave what we'd found; NO WARNINGS (XXX)

    }gex;                   # execute replacement -- that's code not a strin=
g
    =0C
    ###########################################################
    # but wait! load up the %entity mappings enwrapped in =

    # a BEGIN that the last might be first, and only execute
    # once, since we're in a -p "loop"; awk is kinda nice after all.
    ###########################################################

    BEGIN {

        %entity =3D (

            lt      =3D> '<',      #a less-than
            gt      =3D> '>',      #a greater-than
            amp     =3D> '&',      #a nampersand
            quot    =3D> '"',      #a (verticle) double-quote

            nbsp    =3D> chr 160, #no-break space
            iexcl   =3D> chr 161, #inverted exclamation mark
            cent    =3D> chr 162, #cent sign
            pound   =3D> chr 163, #pound sterling sign CURRENCY NOT WEIGHT
```

```
curren =3D> chr 164, #general currency sign
yen    =3D> chr 165, #yen sign
brvbar =3D> chr 166, #broken (vertical) bar
sect   =3D> chr 167, #section sign
uml    =3D> chr 168, #umlaut (dieresis)
copy   =3D> chr 169, #copyright sign
ordf   =3D> chr 170, #ordinal indicator, feminine
laquo  =3D> chr 171, #angle quotation mark, left
not    =3D> chr 172, #not sign
shy    =3D> chr 173, #soft hyphen
reg    =3D> chr 174, #registered sign
macr   =3D> chr 175, #macron
deg    =3D> chr 176, #degree sign
plusmn =3D> chr 177, #plus-or-minus sign
sup2   =3D> chr 178, #superscript two
sup3   =3D> chr 179, #superscript three
acute  =3D> chr 180, #acute accent
micro  =3D> chr 181, #micro sign
para   =3D> chr 182, #pilcrow (paragraph sign)
middot =3D> chr 183, #middle dot
cedil  =3D> chr 184, #cedilla
sup1   =3D> chr 185, #superscript one
ordm   =3D> chr 186, #ordinal indicator, masculine
raquo  =3D> chr 187, #angle quotation mark, right
frac14 =3D> chr 188, #fraction one-quarter
frac12 =3D> chr 189, #fraction one-half
frac34 =3D> chr 190, #fraction three-quarters
iquest =3D> chr 191, #inverted question mark
Agrave =3D> chr 192, #capital A, grave accent
Aacute =3D> chr 193, #capital A, acute accent
Acirc  =3D> chr 194, #capital A, circumflex accent
Atilde =3D> chr 195, #capital A, tilde
Auml   =3D> chr 196, #capital A, dieresis or umlaut mark
Aring  =3D> chr 197, #capital A, ring
AElig  =3D> chr 198, #capital AE diphthong (ligature)
Ccedil =3D> chr 199, #capital C, cedilla
Egrave =3D> chr 200, #capital E, grave accent
Eacute =3D> chr 201, #capital E, acute accent
Ecirc  =3D> chr 202, #capital E, circumflex accent
Euml   =3D> chr 203, #capital E, dieresis or umlaut mark
Igrave =3D> chr 204, #capital I, grave accent
Iacute =3D> chr 205, #capital I, acute accent
Icirc  =3D> chr 206, #capital I, circumflex accent
Iuml   =3D> chr 207, #capital I, dieresis or umlaut mark
ETH    =3D> chr 208, #capital Eth, Icelandic
Ntilde =3D> chr 209, #capital N, tilde
Ograve =3D> chr 210, #capital O, grave accent
Oacute =3D> chr 211, #capital O, acute accent
Ocirc  =3D> chr 212, #capital O, circumflex accent
Otilde =3D> chr 213, #capital O, tilde
Ouml   =3D> chr 214, #capital O, dieresis or umlaut mark
times  =3D> chr 215, #multiply sign
Oslash =3D> chr 216, #capital O, slash
Ugrave =3D> chr 217, #capital U, grave accent
Uacute =3D> chr 218, #capital U, acute accent
Ucirc  =3D> chr 219, #capital U, circumflex accent
Uuml   =3D> chr 220, #capital U, dieresis or umlaut mark
Yacute =3D> chr 221, #capital Y, acute accent
THORN  =3D> chr 222, #capital THORN, Icelandic
szlig  =3D> chr 223, #small sharp s, German (sz ligature)
agrave =3D> chr 224, #small a, grave accent
```

```
        aacute =3D> chr 225, #small a, acute accent
        acirc  =3D> chr 226, #small a, circumflex accent
        atilde =3D> chr 227, #small a, tilde
        auml   =3D> chr 228, #small a, dieresis or umlaut mark
        aring  =3D> chr 229, #small a, ring
        aelig  =3D> chr 230, #small ae diphthong (ligature)
        ccedil =3D> chr 231, #small c, cedilla
        egrave =3D> chr 232, #small e, grave accent
        eacute =3D> chr 233, #small e, acute accent
        ecirc  =3D> chr 234, #small e, circumflex accent
        euml   =3D> chr 235, #small e, dieresis or umlaut mark
        igrave =3D> chr 236, #small i, grave accent
        iacute =3D> chr 237, #small i, acute accent
        icirc  =3D> chr 238, #small i, circumflex accent
        iuml   =3D> chr 239, #small i, dieresis or umlaut mark
        eth    =3D> chr 240, #small eth, Icelandic
        ntilde =3D> chr 241, #small n, tilde
        ograve =3D> chr 242, #small o, grave accent
        oacute =3D> chr 243, #small o, acute accent
        ocirc  =3D> chr 244, #small o, circumflex accent
        otilde =3D> chr 245, #small o, tilde
        ouml   =3D> chr 246, #small o, dieresis or umlaut mark
        divide =3D> chr 247, #divide sign
        oslash =3D> chr 248, #small o, slash
        ugrave =3D> chr 249, #small u, grave accent
        uacute =3D> chr 250, #small u, acute accent
        ucirc  =3D> chr 251, #small u, circumflex accent
        uuml   =3D> chr 252, #small u, dieresis or umlaut mark
        yacute =3D> chr 253, #small y, acute accent
        thorn  =3D> chr 254, #small thorn, Icelandic
        yuml   =3D> chr 255, #small y, dieresis or umlaut mark
    );

    ########################################################
    # now fill in all the numbers to match themselves
    ########################################################
    for $chr ( 0 .. 255 ) { =

        $entity{ '#' . $chr } =3D chr $chr;
    }
}
=0C
############################################################
# premature finish lest someone clip my signature
############################################################

# NOW FOR SOME SAMPLE DATA -- Switch ARGV to DATA above
# to test

__END__

<title>Tom Christiansen's Mox.Perl.COM Home Page</title>
<BODY BGCOLOR=3D#ffffff TEXT=3D#000000>
<!--

<BODY BGCOLOR=3D"#000000" TEXT=3D"#FFFFFF"
      LINK=3D"#FFFF00" VLINK=3D"#22AA22" ALINK=3D"#0077FF">

!-->

<A NAME=3DTOP>
```

```
<CENTER>
<h3>
<A HREF=3D"#PERL">perl</a> /
<A HREF=3D"#MAGIC">magic</a> /
<A HREF=3D"#USENIX">usenix</a> /
<A HREF=3D"#BOULDER">boulder</a>
</h3>
<BR>
The word of the day is <i>nidificate</i>.
</CENTER>

Testing: &#69; &#202; &Auml;
</a>

<HR NOSHADE SIZE=3D3>

<A NAME=3DPERL>
<CENTER>
<h1>
<IMG SRC=3D"/deckmaster/gifs/camel.gif" ALT=3D"">
<font size=3D7>
Perl
</font>
<IMG SRC=3D"/deckmaster/gifs/camel.gif" ALT=3D"">
</h1>
</a>

DOCTYPE START1
<!DOCTYPE  HTML PUBLIC "-//IETF//DTD HTML 2.0//EN"
  -- This is an annoying comment > --
>
END1

DOCTYPE START2
<!DOCTYPE  HTML PUBLIC "-//IETF//DTD HTML 2.0//EN"
  -- This is an annoying comment  --
>
END2

<I>
<BLOCKQUOTE>
<DL><DT>A ship then new they built for him
<DD>of mithril and of elven glass...
</DL>
</I>
</BLOCKQUOTE>
</CENTER>

<HR size=3D3 noshade>

<BLOCKQUOTE>
    Wow!  I really can't believe that anyone has read this far =

    in this very long news posting about irregular expressions. :-)
    Is anyone really still with me?  If so, make my day and
    drop me a piece of email.
</BLOCKQUOTE>

<UL>
<LI>
```

```
    <A HREF=3D"/perl/CPAN/README.html">CPAN
    (Comprehensive Perl Archive Network)</a> sites are replicated around th=
e world; please ch
oose
    from <A HREF=3D"/CPAN/CPAN.html">one near you</a>.
    The <A HREF=3D"/CPAN/modules/01modules.index.html">CPAN index</a
>
    to the <A HREF=3D"/CPAN/modules/00modlist.long.html">full module
s file</a>
    are also good places to look.

    <LI><IMG SRC=3D"/deckmaster/gifs/new.gif" WIDTH=3D26 HEIGHT=3D13 ALT=3D=
"NEW">
    Here's a table of perl and CGI-related books and publications, in eithe=
r =

    <A HREF=3D"/perl/info/books.html"><SMALL>HTML</SMALL> 3.0 table format<=
/a>
    or else in =

    <A HREF=3D"/perl/info/books.txt">pre-formatted</a> for old browsers.
```

What's missing from Perl's regular expressions? Anything? Well, yes. The first is that they should be first-class objects. There are some really embarassing optimization hacks to get around not having compiled regepxs directly-usable accessible. The /o flag I used above is just one of them. (I'm ***not*** talking about the study() function= , which is a neat thing to turbo-ize your matching.) A much more egregious hack involving closures is demonstrated here using the match_any = funtion, which itself returns a function to do the work:

```
    $f =3D match_any('^begin', 'end$', 'middle');
    while (<>) {
        print if &$f();
    } =


    sub match_any {
        die "usage: match_any pats" unless @_;
        my $code =3D <<EOCODE;
    sub {
    EOCODE
        $code .=3D <<EOCODE if @_ > 5;
        study;
    EOCODE
        for $pat (@_) {
            $code .=3D <<EOCODE;
        return 1 if /$pat/;
    EOCODE
        } =

        $code .=3D "}\n";
        print "CODE: $code\n";
        my $func =3D eval $code;
        die "bad pattern: $@" if $@;
        return $func;
    } =
```

That's the kind of thing I just despise writing: the only thing worse would be not being able to do it at all. :-(1st-class compiled regexps would surely help a great deal here.

Sometimes people expect backreferences to be forward references, as in the pattern /\1\s(\w+)/, which just isn't the way it works. A related issue is that while lookaheads work, these are not lookbehinds, which can confuse people. This means /\n(?=3D\s)/ is ok, but you cannot use this for lookbehind: /(?!foo)bar/ will not find an occurrence of "*bar*" that is preceded by something which is not "*foo*". That's because the (?!foo) is just saying that the next thing cannot be "*foo*"--and it's not, it's = a "*bar*", so "*foobar*" will match.

There isn't really much support for user-defined character classes. You see a bit of that in the urlify program above. On the other = hand, this might be the most clear way of writing it.

Another thing that would be nice to have is the ability to someone specify a recursive match with nesting. That ways you could pull out matching parens or braces or begin/end blocks etc. I don't know what a good syntax for this might be. Maybe (?{...) for the opening one and (?}...) for the closing one, as in:

```
/\b(?{begin)\b.*\b(?}end)\b/i
```

Finally, while it's cool that perl's patterns are 8-bit clean, = will match strings even with null bytes in them, and have = support for alternate 8-bit character sets, it would certainly make the world happy if there were full Unicode support.

So Steve, did I answer your question? :-)

--tom

PS: There's a bug outstanding on using /m in conjunction with split()

--

Tom Christiansen Perl Consultant, Gamer, Hiker tchrist@mox.perl.com

I don't believe it's written in Perl, though it probably
ought to have been. :-)
--Larry Wall in <1995Feb21.180249.25507@netlabs.com>

LECTURE 7: PERL PACKAGES AND MODULES

READING ASSIGNMENT:
TYPERL 5 Ch. 20
Read the perlmod man page
Try reading Ch. 19 (at least skim it)

HOMEWORK ASSIGNMENT:
Assignment No. 7 (Due 11/18/96).

Perl MANPAGES:

- Perl Modules - perlmod
- Perl POD - perlpod

TOPICS

- What is a Package and how are they used?
- How to create your own packages.
- What is a Module and how are they used?
- How to create your own modules.
- What Modules/Packages exist?
- Overview of some standard Perl modules
- POD: Plan Old Documentation

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 8]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 7: PACKAGES AND MODULES

PACKAGES

- **What is a Package?**

  Perl tracks all variable names and subroutines using a `symbol table` or `namespace`. A **Package** is a collection of names in a symbol table, i.e., in Perl a `namespace` is called a `package`!

- **Why are Packages useful?**

  Packages are useful in that they allow the programmer to organize related variables and functions into a single unit that can be loaded into the main program. It also keeps you from stomping on other variables when you import these variables and functions into your program (assuming its done correctly). Note that packages will hide, but not protect private data.

Declaring a Package

To declare a package use the following syntax:

```
package mypack;
```

This statement will declare the current package to have the name `mypack`.

- The scope of a package declaration is from the declaration until either the end of the enclosing block, or the next package declaration.
- All variables and subroutines (except those declared with `my()`, or those qualified with another package) are placed in the namespace of the current package.
- The **initial or default package** in Perl is **main**. Therefore, global variables in Perl exist in the `main` package (the namespace or symbol table for `main`).
- To specifically reference both a package and a variable or subroutine defined within the namespace of that package use:

```
$mypack::var      #scalar variable $var
                  #defined in the package mypack

%mypack::users    #hash variable %users
                  #defined in the package mypack
```

  NOTE: In Perl 4 you would have used the syntax

  ```
  $mypack'var
  ```

  I hope we're all using Perl 5!

- Declaring `package` without any name forces Perl to require that all variables and subroutines must explicitly specify their package.

```
package;
```

`$mypack::var`

This is enforced until the next `package;` statement.

- A package is independent of a file, since you can have more than one packages in a file. It is, however, typical to have only one package per file, with the file name reflecting the name of the package (see the discussion of modules below). Thus, the package `mypack` would be stored in the file: `mypack.pl`.
- You can nest packages inside other packages. Each package declaration must declare a complete package name, no prefixes are allowed. Caution! This can get confusing.

EXAMPLE

This example declares two packages, `pack1` and `pack2`. The namespace for `pack1` contains a variable `$var`, that is completely separate from the `$var` variable in the `pack2` namespace.

```
#!/usr/local/bin/perl
#
#  Listing 20.4 from TYPERL 5
#  with slight modifications
#
package pack1;
        $var = 26;

package pack2;
        $var = 34;

# return to the main package and
# print the two values

package main;
        print ("\$var = $pack1::var\n");
    print ("\$var = $pack2::var\n");


OUTPUT

        $var = 26
        $var = 34
```

Packages and Subroutines

- The package declaration affects ALL statements in a program (within the scope of the declaration) including subroutines

```
#!/usr/local/bin/perl
#
package pack1;  #declare our package

&greeting("Welcome to my Package!");    # run a subroutine defined in
                                        # the current package
#  subroutine definition
sub greeting {
        my ($x) = @_;
        print "$x\n";
}

# switch back to the main package
# and call the greeting function

package main;

&pack1::greeting("Hello World");
```

OUTPUT

```
Welcome to my Package!
Hello World
```

Packages and System Variables

- System variables are defined in the `main` package, even when referenced from inside another package.
  - `STDIN, STDOUT, STDERR, ARGV`
  - `%ENV, %INC, @INC, $ARGV, @ARGV`
  - System variables with special characters: `$_, $%, ...`, etc.

Symbol Tables

- The symbol table for a package is stored in a hash with the name of the package followed by `::`, e.g,

```
Symbol Table for package main
        %main::

Symbol Table for package pack1
        %pack1::

Symbol Table for nested package pack2 (declared within pack1)
        %pack1::pack2::
```

- You can access the hash that stores the symbol table for any package using the dumpvar.pl program (usually distributed with Perl). Note that all the comments after the # symbol were added by me, the program does not produce these.

```
gasster> perl -e 'require "dumpvar.pl"; &dumpvar;'

$@ = ''               #error message
FileHandle(stdin) => fileno(0)
FileHandle(STDIN) => fileno(0)
$" = ' '              #list interpolated to string record separator
FileHandle(stdout) => fileno(1)
FileHandle(STDOUT) => fileno(1)
$$ = 6944             #PID
%ENV = (
   'EDITOR' => 'vi'
   'HOME' => '/exp/home/gasster'
   'HOST' => 'polaris'
   'HOSTTYPE' => 'sun4'
   'HZ' => 100
   'LOGNAME' => 'gasster'
   'MAIL' => '/var/mail/gasster'
   'MANPATH' => '/usr/man:/usr/local/man'
   'PATH' => '/bin:/usr/bin:/usr/ucb:/usr/sbin:/usr/local/bin:/etc:.'
   'PWD' => '/exp/home/gasster/perllib'
   'SHELL' => '/usr/local/bin/tcsh'
   'SHLVL' => 1
   'TERM' => 'vt220'
   'TZ' => 'US/Pacific'
   'USER' => 'gasster'
)
$/ = '                #input record separator, default \n
'
$0 = '-e'             #program name
FileHandle(STDERR) => fileno(2)
FileHandle(stderr) => fileno(2)
$^T = 846489124       #time program started
```

```
$^X = '/usr/local/bin/perl'      #name by which currently
                                 #executed program was invoked
@INC = (
   0  '/opt/gnu/lib/perl5/sun4-solaris/5.002'
   1  '/opt/gnu/lib/perl5'
   2  '/opt/gnu/lib/perl5/site_perl/sun4-solaris'
   3  '/opt/gnu/lib/perl5/site_perl'
   4  '.'
)
%INC = (
   'dumpvar.pl' => '/opt/gnu/lib/perl5/dumpvar.pl'
)
@_ = (
   0  0
   1  '_'
   2  '*main::_'
)
gasster>
```

EXAMPLE: Creating a Package - Menu.pl

As an example of creating and using a package let's define a simple user interface. It consists of the following subroutines:

`PrintMenu`
: print out a user specified menu

`getUserInput`
: parse the user's choice

`ValidateOption`
: check if user's choice was a valid option

To start off we create our subroutines, `PrintMenu`, `getUserInput` and `ValidateOption`. We then put them into a file, called `Menu.pl`, and wrap them up using the package declaration. Then we put the file `Menu.pl` into our Perl library directory (`/home/gasster/perllib` for my Unix system, `APPS_2:Perl:MacPerl5_1:PERL5LIB:` on my Mac).

Menu.pl

```
#
#-------------------------------------------------------------#
#       file: Menu.pl
#
#    functions:
#               getUserInput
#               PrintMenu
#               ValidateOption
#
#       This package contains the functions for generating
#       a simple user menu and then getting the users choice
#
#    s. gasster 10/96
#-------------------------------------------------------------#
#

package Menu;


#       Print a user defined Menu including
#       a title to STDOUT

sub PrintMenu {

local ($title, %menu_items) = @_;
```

```
local $item;

print "\n\t$title\n";

foreach $item (sort keys %menu_items) {
   print "$item.  $menu_items{$item}\n";
}
        return 1;
}

#    reads a line of input from STDIN
#    requires the user to enter something
#    other than a <CR>

sub getUserInput {
        my($prompt) = @_;       #prompt to get user input
        my($UserInput);         #what the user enters

        #get input from user, don't accept null input
        do {
                print ("$prompt> ");
                $UserInput = <STDIN>;
                chomp( $UserInput );    #remove newline character
        } until ($UserInput ne "");

        return $UserInput;
}

#
#       test if options selected is
#       a valid option
#
sub ValidateOption {
        my ($option, %menu_items) = @_;
        if (exists $menu_items{$option}){
                return 1;
        } else {
                print "Invalid option, please re-enter\n";
                return 0;
        }
}

1;   #don't forget this!
```

Here is a simple program which makes use of the Menu package.

```
require 'Menu.pl';  #load our menu package

#  define our menu options and prompts

$prompt = "Select an Option";

$title = "Sam's Menu";

%options = (A=>"Option 1",
            B=>"Option 2",
            C=>"Option 3");
#
#  print menu and get user selection
#  since I am in the package main
#  I need to call these subroutines
#  using Menu::
#  also note that I didn't need the &
#  in front of the subroutine call
#  since the definitions were already loaded
```

```
#
Menu::PrintMenu($title, %options);

do {
    $option = Menu::getUserInput($prompt);
}while (!Menu::ValidateOption($option, %options));

print "OPTION SELECTED = $option\n";


INPUT/OUTPUT

        Sam's Menu
A.  Option 1
B.  Option 2
C.  Option 3
Select an Option> E
Invalid option, please re-enter
Select an Option> c
Invalid option, please re-enter
Select an Option> a
Invalid option, please re-enter
Select an Option> b
Invalid option, please re-enter
Select an Option> d
Invalid option, please re-enter
Select an Option> 3
Invalid option, please re-enter
Select an Option> C
OPTION SELECTED = C
```

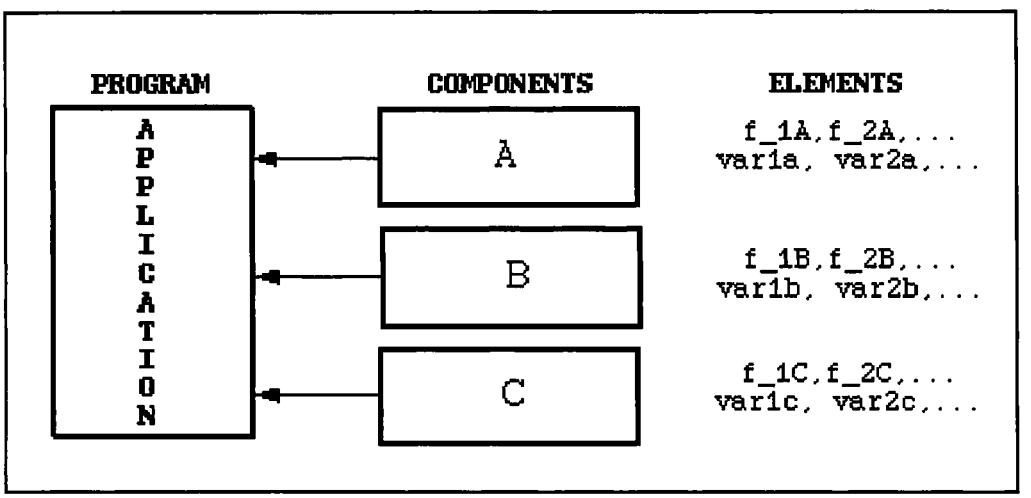
That's all there is to it! Pretty simple!

MODULES

WHAT IS A MODULE?

Consider a program which consists of many components. Each component in turn consists of many different variables and functions. This is illustrated in the figure below.

ILLUSTRATION OF PROGRAM COMPONENTS AND MODULES



- A **MODULE** is the collection of functions and variables in a given program component. Modules are used to collect or bring together a set of functions and variables which perform related or similar tasks.

- The encapsulation necessary to build and define a module is performed with packages. As we just saw with the simple example `Menu.pl`, we can create a package which will hold all the functions and variables necessary to create and parse a menu and user input.

CREATING A MODULE

In Perl you use packages to create modules. There are, however, a few other things that need to be added to the package which we will discuss.

First the basic steps in *module creation* are:

- Decide on what functions and variables you want to encapsulate as a module.
- Create a package, with the name `Mymod`, which contains these functions and variables. Store these in a file with the same name as the package and extension `.pm`:

```
Mymod.pm
```

- Add some additional declarations and statements about what your module will **EXPORT** to the outside world at the beginning of the module package.
- Put your module in your library so Perl can find it! Now use it!

EXPORTING

The file `Mymod.pm` might look something like:

```
#
#...documentation can go here...
# ...usually its in the form of Perl pod...
#
#    file: Mymod.pm
#

package    Mymod;
require    Exporter;
@ISA     = qw(Exporter);
@EXPORT  = qw(func1, func2, func3);
@EXPORT_OK  = qw($var1, %UserNames, @array, $filename, func3, func4);
...
....the acutal code for the module....
...

1;     #our old friend!
```

Let's examine each of these statements:

1. The first statement, `package Mymod;`, defines the package which encapsulates the module (stored in the file `Mymod.pm`).
2. The next statement, `require Exporter;`, tells Perl to load the Exporter module. This module provides import/export controls for Perl modules. The `Exporter` module provides a default method (a subroutine called `import()`) which allows Perl to import into program the names you want to access from a given module.
3. `@ISA = qw(Exporter);`
   The `@ISA` array contains names of *modules* so Perl can search for *subroutines* in other *modules*.

   `@ISA = qw(Exporter)` -- All modules which export names do this. A statement such as `use xxx(a b c);` causes

a subroutine named `import()` to be executed, and this subroutine is taken from the *Exporter* module.

4. The statement `@EXPORT = qw(func1, func2, func3);` tells Perl what symbols (from the module symbol table) to export (into the calling packages symbol table) by default. Don't get too carried away with this and clutter the importing package's namespace with a lot of stuff the user may not really need (unless he/she explicitly wants it).

5. The statement `@EXPORT_OK = qw($var1, %UserNames, @array, $filename, func3, func4);` tells Perl what symbols (from the symbol table) to export on request; thus you must ask for these with a statement of the form:

```
use Mymod qw($var1, %UserNames, func4);
```

This will only import `$var1`, `%UserNames` and `func4` into the namespace of the calling package.

IMPORTING A MODULE

- To load a module into your current program, package or module we *use* the `use` compiler directive.
- `use` is a compiler directive that controls the importation of functions and variables from a module. As an example, we load the `English.pm` module with the syntax:

```
use English;
```

- In files where you want to use `Mymod.pm`, you have the following options:

```
use Mymod;                      # Import default symbols into current package
use Mymod qw($var1, func4);     # Import only $var1 and func4 into current package
use Mymod ();                   # No symbols imported into current package
```

EXAMPLE: The Menu.pm Module

Let's convert our `Menu.pl` package into a Perl module. To do this we need to decide on what variables and functions we would like to export so that the user will have access to them. For the `Menu` module the user needs access to the following functions:

- `getUserInput`
- `PrintMenu`
- `ValidateOption`

The module `Menu.pm` is given below. Note I have made a few small changes to the functions compared to the package discussed above. I have also added `pod` to document this package. `pod` is discussed later in this lecture.

```
#
#-------------------------------------------------------------#
#       file: Menu.pm
#
#    functions:
#               getUserInput
#               PrintMenu
#               ValidateOption
#
#       This package contains the functions for generating
#       a simple user menu and then getting the users choice
#
#    s. gasster 10/96
#-------------------------------------------------------------#
#
```

```
#  start the POD

=head1 NAME

    Menu - User Menu Functions

=head1 SYNOPSIS

    use Menu;

        $prompt = "Select an Option";
        $title = "Sam's Menu";
        %options = (A=>"Option 1",
                    B=>"Option 2",
                    C=>"Option 3");

        PrintMenu($title, %options);
        do {
            $option = getUserInput($prompt);
        }while (!ValidateOption($option, %options));

        print "OPTION SELECTED = $option\n";

=head1 DESCRIPTION

This module exports the simple functions for creating and parsing
a user menu.

=item B<PrintMenu> prints the user specified menu
=item B<getUserInput> parse user input and return selected option
=item B<ValidateOption> returns TRUE if user entered one of the valid options,
                        otherwise FALSE

=head1 HISTORY

Version 1.0  1996/10/01 sdg
Initial version

=head1 BUGS

Let me know.

I welcome any additions to be included in this module.

=head1 AUTHOR

Samuel D. Gasster <gasster@pacificnet.net>

=cut

# end of POD

# start of Module Menu

package      Menu;
require      Exporter;
@ISA       = qw(Exporter);
@EXPORT    = qw(getUserInput PrintMenu ValidateOption Version);

use Carp;

$Version = "1.0";

#       Print a user defined Menu including
#       a title to STDOUT

sub Menu::PrintMenu {
```

```
local ($title, %menu_items) = @_;

local $item;

print "\n\t$title\n";

foreach $item (sort keys %menu_items) {
   print "$item.  $menu_items{$item}\n";
}
        return 1;
}

#    reads a line of input from STDIN
#    requires the user to enter something
#    other than a <CR>

sub Menu::getUserInput {
        my($prompt) = @_;       #prompt to get user input
        my($UserInput);         #what the user enters

        #get input from user, don't accept null input
        do {
                print ("$prompt> ");
                $UserInput = <STDIN>;
                chomp( $UserInput );    #remove newline character
        } until ($UserInput ne "");

        return $UserInput;
}

#
#       test if options selected is
#       a valid option
#
sub Menu::ValidateOption {
        my ($option, %menu_items) = @_;
        if (exists $menu_items{$option}){
                return 1;
        } else {
                carp "Invalid option \"$option\", please re-enter";
                return 0;
        }
}

1;   #don't forget this!
```

Here is a simple test program for the `Menu.pm` module:

```
#!/usr/local/bin/perl

#       file: test_menu.pl
#       test the Menu.pm module
#

use Menu;  #load Menu.pm

print "Menu Program Version No.:  $Menu::Version\n\n";

#  define our menu options and prompts

$prompt = "Select an Option";

$title = "Sam's Menu";
```

```
%options = (A=>"Option 1",
            B=>"Option 2",
            C=>"Option 3");
#
#  print menu and get user selection
#  note that I didn't need the &
#  in front of the subroutine calls
#  since the definitions were already
#  loaded
#
PrintMenu($title, %options);


do {
    $option = getUserInput($prompt);
}while (!ValidateOption($option, %options));

print "OPTION SELECTED = $option\n";

exit (1);
------

INPUT/OUTPUT

Menu Program Version No.:  1.0


        Sam's Menu
A.  Option 1
B.  Option 2
C.  Option 3
Select an Option> a
# Invalid option "a", please re-enter
File 'APPS_2:Perl:MacPerl5_1:PERL5LIB:test_menu.pl'; Line 32
Select an Option> 3
# Invalid option "3", please re-enter
File 'APPS_2:Perl:MacPerl5_1:PERL5LIB:test_menu.pl'; Line 32
Select an Option> F
# Invalid option "F", please re-enter
File 'APPS_2:Perl:MacPerl5_1:PERL5LIB:test_menu.pl'; Line 32
Select an Option> A
OPTION SELECTED = A
```

USING PREDEFINED MODULES

How Do I Find Out What Modules I Have?

To find out all the modules installed on your system, including.those without documentation or outside the standard release, do this:

```
find `perl -e 'print "@INC"'` -name '*.pm' -print
```

When you execute this command you should get something that looks like:

```
$find `perl -e 'print "@INC"'` -name '*.pm' -print
/usr/local/freeware/perl5/lib/sun4-sunos/Config.pm
/usr/local/freeware/perl5/lib/AnyDBM_File.pm
/usr/local/freeware/perl5/lib/AutoLoader.pm
/usr/local/freeware/perl5/lib/AutoSplit.pm
/usr/local/freeware/perl5/lib/Benchmark.pm
/usr/local/freeware/perl5/lib/Carp.pm
```

```
/usr/local/freeware/perl5/lib/Cwd.pm
/usr/local/freeware/perl5/lib/English.pm
/usr/local/freeware/perl5/lib/Env.pm

...shortened to save space...

/usr/local/freeware/perl5/lib/POSIX.pm
/usr/local/freeware/perl5/lib/SDBM_File.pm
/usr/local/freeware/perl5/lib/Socket.pm
```

Note that the path, `/usr/local/freeware/perl5/lib`, will differ depending on how your system is setup. If you are not on a Unix system you should consult your system specific release notes.

The Perl Module List

- This is a list of the Perl Modules as of *Tue Oct 29 08:15:37 1996 GMT* .

Where Can I Get Modules I Don't Have Or Up-Dates?

- Always check the Perl web site first at http://www.perl.com/perl/. This site has an extensive list of modules and related documentation!
- The Perl Modules should also be available from any of the **CPAN** sites.
- Check the news groups from time-to-time for up-date information (comp.lang.perl.modules and comp.lang.perl.announce).

OVERVIEW OF SOME PERL MODULES

Here is a brief survey of the following Perl modules:

English

```
use English;
```

This module provides aliases for the built-in variables whose names no one seems to like to read. Variables with side-effects which get triggered just by accessing them (like **$0**) will still be affected.

For those variables that have an **awk** version, both long and short English alternatives are provided. For example, the `$/` variable can be referred to either **$RS** or **$INPUT_RECORD_SEPARATOR** if you are using the English module.

See the *perlvar* manpage for a complete list of these.

Config - access Perl configuration options

```
use Config;
```

The Config module contains everything that was available to the Configure program at Perl build time. Shell variables from config.sh are stored in the readonly-variable %Config, indexed by their names. This module contains a good example of how to make a variable readonly to those outside of it.

EXAMPLES

```
# determine what compiler was used to build Perl

    use Config;
    if ($Config{'cc'} =~ /gcc/) {
        print "built by gcc\n";
    }


# test what system you are on

use Config;

if ($Config{'osname'} =~ /sunos/) {
        print "We are on a Unix System\n";
} elsif ($Config{'osname'} =~ /MacOS/) {
        print "We are on a Macintosh System\n";
}
```

Env

```
use Env;
use Env qw(PATH HOME TERM);
```

Perl maintains environment variables in a pseudo-associative-array named **%ENV**. For when this access method is inconvenient, the Perl module `Env` allows environment variables to be treated as simple variables.

The Env::*import()* function ties environment variables with suitable names to global Perl variables with the same names. By default it does so with all existing environment variables (`keys %ENV`). If the import function receives arguments, it takes them to be a list of environment variables to tie; it's okay if they don't yet exist.

After an environment variable is tied, merely use it like a normal variable. You may access its value

```
@path = split(/:/, $PATH);
```

or modify it

```
$PATH .= ":.";
```

however you'd like. To remove a tied environment variable from the environment, assign it the undefined value

```
undef $PATH;
```

---

diagnostics - Perl compiler pragma to force verbose warning diagnostics

```
use diagnostics;
use diagnostics -verbose;
enable  diagnostics;
disable diagnostics;
```

This module extends the terse diagnostics normally emitted by both the perl compiler and the perl interpeter, augmenting them wtih the more explicative and endearing descriptions found in the *perldiag* manpage. Like the other pragmata, it affects to compilation phase of your program rather than merely the execution phase.

To use in your program as a pragma, merely invoke

```
    use diagnostics;
```

at the start (or near the start) of your program. (Note that this *does* enable perl's **-w** flag.) Your whole compilation will then be subject(ed :-) to the enhanced diagnostics. These still go out **STDERR**.

Due to the interaction between runtime and compiletime issues, and because it's probably not a very good idea anyway, you may not use `no diagnostics` to turn them off at compiletime. However, you may control there behaviour at runtime using the *disable()* and *enable()* methods to turn them off and on respectively.

The **-verbose** flag first prints out the the *perldiag* manpage introduction before any other diagnostics. The **$diagnostics::PRETTY** can generate nicer escape sequences for pagers.

Test program

```
#!/usr/local/bin/perl
#
    use diagnostics;
    print NOWHERE "nothing\n";
    print STDERR "\n\tThis message should be unadorned.\n";
    warn "\tThis is a user warning";
    print "\nDIAGNOSTIC TESTER: Please enter a <CR> here: ";
    my $a, $b = scalar <STDIN>;
    print "\n";
    print $x/$y;
```

OUTPUT on MacPerl

```
# Parens missing around "my" list.
File
        'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 8 # (#1)
    (W) You said something like

        my $foo, $bar = @_;

    when you meant

        my ($foo, $bar) = @_;

    Remember that "my" and "local" bind closer than comma.

# Identifier "main::x" used only once: possible typo.
File
        'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 10 # (#2)
    (W) Typographical errors often show up as unique identifiers.  If you
    had a good reason for having a unique identifier, then just mention it
    again somehow to suppress the message.

# Identifier "main::NOWHERE" used only once: possible typo.
File
        'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 4 # (#2)
# Identifier "main::y" used only once: possible typo.
File
        'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 10 # (#2)

# Filehandle main::NOWHERE never opened.
File
'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 4 # (#3) (W)
An I/O operation was attempted on a filehandle that was never
initialized. You need to do an open() or a socket() call, or call a
constructor from the FileHandle package.


        This message should be unadorned.
#       This is a user warning.
File 'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 6
```

```
DIAGNOSTIC TESTER: Please enter a <CR> here:

# Use of uninitialized value.
File
'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 10 # <STDIN>
chunk 1 # (#4) (W) An undefined value was used as if it were already
defined.  It was interpreted as a "" or a 0, but maybe it was a mistake.
 To suppress this warning assign an initial value to your variables.

# Illegal division by zero.
File
'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 10 # <STDIN>
chunk 1 # (#5) (F) You tried to divide a number by 0.  Either something
was wrong in your logic, or you need to put a conditional in to guard
against meaningless input.

# Uncaught exception from user code:
        # Illegal division by zero.
File 'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 10 #
<STDIN> chunk 1
File 'APPS_2:Perl:MacPerl5_1:PERL5LIB:diagnostics_test.pl'; Line 10
```

Carp

```
carp    - warn of errors (from perspective of caller)
croak   - die of errors (from perspective of caller)
confess - die of errors with stack backtrace
```

The Carp routines are useful in your own modules because they act like *die()* or *warn()*, but report where the error was in the code they were called from. Thus if you have a routine *Foo()* that has a *carp()* in it, then the *carp()* will report the error as occurring where *Foo()* was called, not where *carp()* was called.

```
use Carp;
croak "We're outta here!";

OUTPUT

        # We're outta here!
        File 'APPS_2:Perl:MacPerl5_1:PERL5LIB:test_Carp.pl'; Line 2
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 7]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 7:
PACKAGES AND MODULES

dumpvar.pl

```
package dumpvar;

# translate control chars to ^X - Randal Schwartz
sub unctrl {
        local($_) = @_;
        return \$_ if ref \$_ eq "GLOB";
        s/([\001-\037\177])/'^'.pack('c',ord($1)^64)/eg;
        $_;
}
sub main'dumpvar {
    ($package,@vars) = @_;
    $package .= "::" unless $package =~ /::$/;
    *stab = *{"main::"};
    while ($package =~ /(\w+?::)/g){
        *stab = ${stab}{$1};
    }
    while (($key,$val) = each(%stab)) {
        {
            next if @vars && !grep($key eq $_,@vars);
            local(*entry) = $val;
            if (defined $entry) {
                print "\$",&unctrl($key)," = '",&unctrl($entry),"'\n";
            }
            if (defined @entry) {
                print "\@$key = (\n";
                foreach $num ($[ .. $#entry) {
                    print "  $num\t'",&unctrl($entry[$num]),"'\n";
                }
                print ")\n";
            }
            if ($key ne "main::" && $key ne "DB::" && defined %entry
                && !($package eq "dumpvar" and $key eq "stab")) {
                print "\%$key = (\n";
                foreach $key (sort keys(%entry)) {
                    print "  $key\t'",&unctrl($entry{$key}),"'\n";
                }
                print ")\n";
            }
        }
    }
}

1;
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 7 | BACK]

CPAN's 211 modules distributions

in alphabetical order by modules contained in the distributions

Tue Oct 29 08:15:37 1996 GMT

The list contains in column 1 the base package name, in column 2 the userid of the author which is a hyperlink to her (fullname)-homedirectory. The filename in column 3 is a hyperlink to the real file. The last column "young" contains a plus sign if the file is younger than two weeks.

See also:

- The HTML version of Tim Bunce's Module List
- The detailed listing of all packages found in these distribution files
- CPAN's who is who

---

module	author	current distribution file	size	uploaded	young
Agent	JDUNCAN	Agent-2.91.tar.gz	7k	28 May 1996	
Alias	GSAR	Alias-2.1.tar.gz	6k	06 Apr 1996	
Apache	DOUGM	mod_perl-0.84.tar.gz	66k	28 Oct 1996	+
Archie	GBOSS	perl_archie.1.5.tar.gz	10k	20 Aug 1995	
Array	AKSTE	Array-PrintCols-1.2.tar.gz	15k	18 Dec 1995	
ARS	JMURPHY	ARSperl-1.2.2.tgz	91k	08 May 1996	
AtExit	BRADAPP	AtExit-1.02.tar.gz	6k	05 Sep 1996	
B	MICB	Compiler-a3.tar.gz	87k	02 Sep 1996	
BSD	JHI	BSD-Resource-1.03.tar.gz	10k	17 Feb 1996	
C	ILYAZ	C-Scan-0.3.tar.gz	8k	06 Sep 1996	
Callback	MUIR	Callback.pm.gz	1k	02 Nov 1995	
Cdk	GLOVER	Curses-DevKit-4.8.tar.gz	71k	12 Jul 1996	
CGI	MUIR	CGI-Out-96.081401.tar.gz	3k	14 Aug 1996	
CGI	DOUGM	mod_perl-0.84.tar.gz	66k	28 Oct 1996	+
CGI	LDS	CGI-modules-2.75.tar.gz	79k	05 Apr 1996	
CGI	LDS	CGI.pm-2.27.tar.gz	124k	24 Oct 1996	+
CGI	MGH	CGI-Response-0.03.tar.gz	11k	02 Dec 1995	
CGI	MIKEH	CGI_Imagemap-1.00.tar.gz	9k	02 Mar 1996	
CGI_Lite	SHGUN	CGI_Lite-1.62.pm.gz	6k	20 Jan 1996	
Class	DMR	Class-Eroot-19960603.tar.gz	7k	03 Jun 1996	
Class	PSEIBEL	MethodMaker-0.9.tar.gz	8k	28 Sep 1996	
Compress	PMQS	Compress-Zlib-0.4.tar.gz	15k	25 Jun 1996	
ConfigReader	AMW	ConfigReader-0.5.tar.gz	17k	19 Feb 1996	
Convert	ANDK	Convert-UU-0.01.tar.gz	4k	16 Jun 1996	
CPAN	ANDK	CPAN-0.41a.tar.gz	31k	01 Oct 1996	
Crypt	GARY	Crypt-DES-1.0.tar.gz	12k	22 May 1996	
Crypt	GARY	Crypt-IDEA-1.0.tar.gz	5k	22 May 1996	
Curses	WPS	Curses-1.0.tar.gz	29k	15 Jun 1996	
Data	GSAR	Data-Dumper-2.02b.tar.gz	19k	13 Apr 1996	
Data	ILYAZ	Data-Flow-0.04.tar.gz	4k	04 Aug 1996	
Date	GBARR	TimeDate-1.05.tar.gz	14k	31 Jul 1996	
Date	STBEY	DateCalc-2.2.tar.gz	24k	26 Jun 1996	
Date	SBECK	DateManip-5.06.tar.gz	29k	25 Oct 1996	+
Date	TOMC	Date-GetDate-2.00.tar.gz	29k	09 Jun 1996	
DBD	ADESC	DBD-Informix-0.24.tar.gz	26k	26 Sep 1996	
DBD	ADESC	DBD-mSQL-0.60p19.tar.gz	26k	11 Jan 1996	
DBD	BENLI	DBD-QBase-0.03.tgz	24k	04 Nov 1995	
DBD	MEWP	DBD-Sybase-0.01.tar.gz	5k	02 Feb 1996	
DBD	MHM	DBD-DB2-0.61.tar.gz	34k	07 Oct 1996	
DBD	TIMB	DBD-Oracle-0.42.tar.gz	57k	28 Oct 1996	+
DBD	TIMB	DBI-0.73.tar.gz	35k	15 Oct 1996	
DBI	TIMB	DBI-0.73.tar.gz	35k	15 Oct 1996	
DCE	DOUGM	DCE-Perl-0.11.tar.gz	16k	20 Aug 1996	
DES	GARY	Crypt-DES-1.0.tar.gz	12k	22 May 1996	
Des	MICB	Des-perl-a1.tar.gz	11k	20 Aug 1995	

Devel	ADESC	Devel-CoreStack-1.3.tar.gz	7k	07	Jul 1996	
Devel	ANDK	Devel-Symdump-1.21.tar.gz	7k	22	Jan 1996	
Devel	ILYAZ	Devel-Peek-0.7.tar.gz	7k	02	Jul 1996	
Devel	ILYAZ	Devel-RegExp-0.1.tar.gz	4k	20	Dec 1995	
Devel	JACKS	CallerItem-1.0.tar.gz	3k	20	Aug 1995	
Devel	JACKS	DumpStack-1.1.tar.gz	2k	20	Aug 1995	
Devel	JOEHIL	traceFunc-0.1.tar.gz	4k	03	Jun 1996	
EventServer	JACKS	EventServer-2.3.tar.gz	27k	07	Feb 1996	
ExtUtils	ANDK	MakeMaker-5.38.tar.gz	105k	03	Sep 1996	
FCGI	SKIMO	FCGI-0.26.tar.gz	22k	15	Oct 1996	+
File	ASHER	File-Tools-2.0.tar.gz	5k	04	Apr 1996	
File	MUIR	File-BasicFLock-96.072401.tar.gz	2k	24	Jul 1996	
File	MUIR	File-Flock-96.042302.tar.gz	4k	24	Apr 1996	
File	MUIR	File-Slurp-96.042202.tar.gz	2k	22	Apr 1996	
File	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
File	JNH	File-Cmp-0.03.tar.gz	3k	21	Oct 1996	+
File	KJALB	File-Lock-0.9.tar.gz	9k	31	Jan 1996	
File	SHIGIO	File-PathConvert-0.1.tar.gz	5k	27	Oct 1996	+
File	TYEMQ	FileKGlob.tar.gz	7k	20	Aug 1995	
Filter	PMQS	Filter-1.10.tar.gz	20k	25	Jun 1996	
FindBin	GBARR	FindBin-1.04.tar.gz	4k	17	Jan 1996	
Font	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
FreezeThaw	ILYAZ	etext0.5.4.tar.gz	300k	21	May 1996	
GD	LDS	GD-1.14.tar.gz	95k	12	Sep 1996	
Getopt	CJM	Getopt-Mixed-1.008.tar.gz	10k	09	Feb 1996	
Getopt	JV	GetoptLong-2.4.tar.gz	13k	02	Oct 1996	
Getopt	JARW	Getopt-Regex-0.01.tar.gz	3k	20	Dec 1995	
HTML	ERYQ	HTML-Stream-1.24.tar.gz	16k	24	Jul 1996	
HTML	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
HTML	GAND	HTML-0.6.tar.gz	25k	29	Aug 1995	
HTML	KJALB	PodParse-0.02.tar.gz	58k	09	Jul 1996	
HTML	YLU	HTML-QuickCheck-1.0b1.tar.gz	4k	01	Oct 1995	
HTML	TOMC	Simple.pm.gz	1k	03	Oct 1995	
HTTP	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
HTTPD	DOUGM	HTTPD-Tools-0.52.tar.gz	20k	21	Mar 1996	
Image	RJRAY	Image-Size-1.2.tar.gz	11k	06	Sep 1996	
Include	GBARR	Include-1.02a.tar.gz	6k	04	Jan 1996	
IniConf	SHUTTON	IniConf.pm-0.91.tar.gz	10k	09	Aug 1996	
IO	GBARR	IO-1.12.tar.gz	17k	19	Sep 1996	
Ioctl	KJALB	Ioctl-0_7.tar.gz	5k	03	May 1996	
IPC	JACKS	IPC_SysV.tar.gz	10k	20	Aug 1995	
Language	JACKS	Prolog-alpha.tar.gz	7k	20	Aug 1995	
Log	JARW	Log-Topics-0.02.tar.gz	3k	05	Jan 1996	
Logfile	ULPFR	Logfile-0.114.tar.gz	31k	16	Sep 1996	
LWP	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
Mail	GBARR	MailTools-1.06.tar.gz	30k	13	Aug 1996	
Mail	KJOHNSON	MailFolder-0.03.tar.gz	20k	03	Aug 1996	
Mail	SDOWD	POP3Client-1_14.tar.gz	4k	06	Jun 1996	
Make	NI-S	Make-0.01.tar.gz	11k	08	Jan 1996	
Math	GARY	Math-BigInteger-1.0.tar.gz	19k	22	May 1996	
Math	GARY	Math-PRSG-1.0.tar.gz	4k	22	May 1996	
Math	GARY	Math-TrulyRandom-1.0.tar.gz	4k	22	May 1996	
Math	ILYAZ	MathPari_a3.tar.gz	13k	02	Oct 1995	
Math	JARW	Math-Amoeba-0.01.tar.gz	3k	29	Dec 1995	
Math	JARW	Math-Brent-0.01.tar.gz	4k	29	Dec 1995	
Math	JARW	Math-Derivative-0.01.tar.gz	2k	29	Dec 1995	
Math	JARW	Math-Fortran-0.01.tar.gz	2k	29	Dec 1995	
Math	JARW	Math-Spline-0.01.tar.gz	3k	29	Dec 1995	
Math	JARW	Math-Trig-0.01.tar.gz	2k	29	Dec 1995	
Math	JARW	Math-VecStat-0.03.tar.gz	2k	20	Feb 1996	
Math	LUKKA	Neural-0.03.tar.gz	32k	03	May 1996	
Math	ULPFR	Math-Approx-0.1.tar.gz	2k	27	Oct 1995	
Math	ULPFR	Math-ematica-1.006.tar.gz	12k	25	Mar 1996	
Math	ULPFR	Math-Matrix-0.2.tar.gz	3k	10	Jul 1996	
MD5	NWINT	MD5-1.7.tar.gz	12k	14	Aug 1996	
MIME	ERYQ	MIME-tools-2.03.tar.gz	200k	28	Oct 1996	+
MIME	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
MLDBM	GSAR	MLDBM-1.21.tar.gz	4k	10	Apr 1996	
Msql	ANDK	MsqlPerl-1.11.tar.gz	31k	08	Sep 1996	

Msql	BJEPS	Msql-RDBMS-0.10.tar.gz	8k	07	Apr 1996	
Net	GBARR	libnet-1.01.tar.gz	37k	05	Sep 1996	
Net	GBARR	Net-FTP-1.18_2.tar.gz	13k	05	Jun 1996	
Net	RIK	NIS-a2.tar.gz	6k	10	Nov 1995	
Net	RIK	NISPlus-0.04-alpha.tar.gz	12k	12	May 1996	
Net	RMOSE	Net-Ping-2.02.tar.gz	6k	27	Sep 1996	
Net	SAMPO	Net_SSLeay.pm-0.04.tar.gz	18k	16	Oct 1996	+
Net	SPIDB	Net-ext-a6.tar.gz	36k	20	Aug 1995	
Net	ULPFR	Logfile-0.114.tar.gz	31k	16	Sep 1996	
Neural	LUKKA	Neural-0.03.tar.gz	32k	03	May 1996	
News	RVA	NNTPClient-0.26.tar.gz	12k	26	Apr 1996	
News	SWMCD	News-Newsrc-1.02.tar.gz	6k	26	Aug 1996	
O	MICB	Compiler-a3.tar.gz	87k	02	Sep 1996	
Object	JACKS	Object-Info-1.1.pm.gz	2k	19	Oct 1995	
Opcode	TIMB	Opcode-1.01.tar.gz	20k	07	May 1996	
OpenGL	STANM	OpenGL-0.4.tar.gz	291k	16	Feb 1996	
Oraperl	TIMB	DBD-Oracle-0.42.tar.gz	57k	28	Oct 1996	+
OS2	CJM	OS2-Attrib-0.002.tar.gz	5k	17	Oct 1996	+
OS2	ILYAZ	perl_mlb.zip	540k	08	Oct 1996	
OS2	ILYAZ	OS2-ExtAttr-0_01.zip	5k	22	Apr 1996	
OS2	ILYAZ	OS2-FTP-0_10.zip	4k	23	Apr 1996	
OS2	ILYAZ	OS2-PrfDB-0_01.zip	6k	09	Apr 1996	
OS2	ILYAZ	OS2-SoftInstaller-0.01.tar.gz	6k	14	Sep 1996	
OS2	ILYAZ	OS2-UPM-0_10.zip	5k	23	Apr 1996	
Parallel	EWALKER	Parallel-Pvm-1.1.tar.gz	142k	16	Oct 1996	+
PDL	KGB	PDL1.00_alpha_release.tar.gz	479k	20	Oct 1996	+
Penguin	FSG	One_Penguin-pre2.tar.gz	18k	18	Apr 1996	
Penguin	JDUNCAN	Penguin-Easy-1.1.tar.gz	4k	27	Aug 1996	
Pg	MERGL	pg95perl5-1.3.1.tar.gz	10k	26	Oct 1996	+
PGP	GEHIC	PGP-0.2alpha.tar.gz	13k	29	Jan 1996	
PGP	GEHIC	PGP-0.3a.tar.gz	11k	14	Aug 1996	
PGPLOT	KGB	PGPLOT1.0b_src.tar.gz	55k	23	Feb 1996	
Pod	BRADAPP	PodParser-1.00.tar.gz	23k	19	Aug 1996	
Pod	KJALB	PodParse-0.03.tar.gz	59k	11	Jul 1996	
Pod	KJALB	PodSimplify-0.04.tar.gz	60k	16	Aug 1996	
Proc	MUIR	Proc-Forkfunc-96.042201.tar.gz	1k	22	Apr 1996	
Proc	MSCHILLI	Proc-Simple-1.12.tar.gz	3k	23	May 1996	
Ptty	NI-S	Ptty-0.01.tar.gz	15k	28	Sep 1995	
PV	AGUL	perlvision.tgz	27k	27	Aug 1995	
Quota	TOMZO	Quota-0_2a.tar.gz	8k	12	Mar 1996	
Ref	MUIR	Ref.pm.gz	1k	02	Nov 1995	
Religion	KJALB	Religion-1.04.tar.gz	4k	21	Oct 1995	
Remedy	RIK	Ar-0.2alpha.tar.gz	8k	19	Nov 1995	
Resources	FRANCOC	Resources-1.03.tar.gz	18k	10	Jun 1996	
Safe	TIMB	Opcode-1.01.tar.gz	20k	07	May 1996	
Search	MIKEH	Search-0.2.tar.gz	14k	01	May 1996	
Server	JACKS	EventDrivenServer-2.2.tar.gz	55k	28	Sep 1995	
Server	JACKS	EventServer-2.3.tar.gz	27k	07	Feb 1996	
Server	JACKS	InitializeServer-1.0.tar.gz	5k	20	Aug 1995	
Set	JHI	Set-Scalar-0_003.tar.gz	13k	17	May 1996	
Set	STBEY	Set-IntegerFast-1.1.tar.gz	26k	08	Jan 1996	
Set	SWMCD	Set-IntSpan-1.03.tar.gz	9k	09	Jun 1996	
SGI	AMOSS	SGI-FM-0.1.tar.gz	2k	20	Aug 1995	
SGI	AMOSS	SGI-GL-0.2.tar.gz	7k	20	Aug 1995	
SGI	AMOSS	SGI-SysCalls-0.1.tar.gz	2k	20	Aug 1995	
SGMLS	DMEGG	SGMLSpm-1.03ii.tar.gz	91k	07	Dec 1995	
SHA	UWEH	SHA-1.0.tar.gz	5k	11	Dec 1995	
SNMP	GSM	SNMP-1.6.tar.gz	515k	27	Apr 1996	
Sort	DMACKS	Sort-PolySort-0.12.tar.gz	5k	24	Aug 1996	
Sort	KJALB	SortVersions-1.1.tar.gz	3k	11	Jul 1996	
Statistics	JKAST	StatisticsDescriptive-1.1.tar.gz	4k	20	Aug 1995	
Statistics	JONO	Statistics-ChiSquare-0.1.tar.gz	4k	05	Nov 1995	
Statistics	TOMFA	Statistics-LTU-2.7.tar.gz	16k	20	Feb 1996	
Storable	RAM	Storable-0.1.tar.gz	11k	02	Oct 1995	
String	JHI	String-Approx-1.5.tar.gz	4k	02	Nov 1995	
String	JHI	String-Scanf-1.1.tar.gz	4k	27	Dec 1995	
String	WINKO	BitCount-1.1.tar.gz	1k	02	Nov 1995	
String	WINKO	Parext-1.1.tar.gz	1k	02	Nov 1995	
String	WINKO	Parity-1.2.tar.gz	1k	02	Nov 1995	

Sx	FMC	Sx-2.2.tar.gz	88k	31	Oct 1995	
Sybase	BPOWERS	Sybase-Login-1.2.tar.gz	7k	09	Jun 1996	
Sybase	MEWP	sybperl-2.05.tar.gz	120k	28	Mar 1996	
Sys	AKSTE	Sys-OutPut-1.5.tar.gz	11k	01	Mar 1996	
Sys	JACKS	AlarmCall-1.1.tar.gz	4k	20	Aug 1995	
SyslogScan	RHNELSON	SyslogScan-0.21.tar.gz	39k	27	Sep 1996	
Tcl	MICB	Tcl-b1.tar.gz	15k	20	Aug 1995	
Tcl	MICB	TclTk-b1.tar.gz	11k	20	Aug 1995	
Term	AKSTE	Term-Query-1.15.tar.gz	25k	02	Oct 1995	
Term	ILYAZ	Term-Gnuplot-0.2.tar.gz	8k	20	Aug 1995	
Term	ILYAZ	Term-ReadLine-0.92.tar.gz	28k	11	Sep 1996	
Term	KJALB	Term-Info-1.0.tar.gz	1k	20	Aug 1995	
Term	KJALB	TermReadKey-2.05.tar.gz	26k	14	Mar 1996	
Term	MRKAE	Screen-1.00.tar.gz	7k	13	Jan 1996	
Test	ANDK	Test-Harness-1.12.tar.gz	6k	08	Jul 1996	
Text	MUIR	Text-Tabs+Wrap-96.090101.tar.gz	3k	02	Sep 1996	
Text	ERYQ	Bib-1.18.tgz	37k	21	Dec 1995	
Text	MJD	Text-Template-0.1b.tar.gz	9k	25	Jan 1996	
Text	ULPFR	perlindex-1.107.tar.gz	11k	10	Mar 1996	
Text	ULPFR	Text-German-0.03.tar.gz	13k	02	Jul 1996	
Text	WINKO	Parity-1.1.tar.gz	1k	03	Oct 1995	
Tie	GBARR	Tie-Dir-1.02.tar.gz	2k	26	Apr 1996	
Tie	GSAR	Tie-IxHash-1.1.pm.gz	4k	25	Sep 1995	
Tie	LUSOL	Tie-Watch-0.95.tar.gz	6k	27	Jul 1996	
Time	MUIR	Time-modules-96.100201.tar.gz	16k	03	Oct 1996	
Time	DEWEG	Time-HiRes-01.01.tar.gz	4k	18	Oct 1996	+
Time	GBARR	TimeDate-1.05.tar.gz	14k	31	Jul 1996	
Tk	ALSCH	Tk-SelFile-0.02.tar.gz	7k	20	Nov 1995	
Tk	BPOWERS	Tk-FileDialog-1.3.tar.gz	9k	09	Jun 1996	
Tk	BPOWERS	Tk-WaitBox-1.2.tar.gz	4k	09	Jun 1996	
Tk	GUYDX	Table-1.02.tar.gz	52k	10	Mar 1996	
Tk	ILYAZ	etext0.5.4.tar.gz	300k	21	May 1996	
Tk	NI-S	Tk-JPEG-b11.tar.gz	504k	02	Apr 1996	
Tk	NI-S	Tk400.200.tar.gz	2.1M	07	Sep 1996	
Tk	PVHP	ptkFAQ-0_03.tar.gz	0.8M	23	May 1996	
URI	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
Usage	JACKS	Usage-0.10.tar.gz	11k	17	Feb 1996	
VRML	HPALM	VRML-0.85.tar.gz	53k	22	Oct 1996	+
Wais	ULPFR	Wais-2.117.tar.gz	45k	01	Oct 1996	
Win32	DAVEROTH	Win32_AdminMisc_v961016.zip	33k	20	Oct 1996	+
Win32	DAVEROTH	Win32odbc_v961025.zip	113k	26	Oct 1996	+
Win32	DAVEROTH	Win32Pipe_v960610.zip	38k	10	Jun 1996	
Win32	LUSOL	Tie-Watch-0.92.tar.gz	5k	11	Jun 1996	
WWW	GAAS	libwww-perl-5.04.tar.gz	183k	22	Oct 1996	+
WWW	JOHNH	WWW-Search-1.002.tar.gz	12k	23	Oct 1996	+
Xbase	PRATP	Xbase-1.07.tar.gz	8k	24	Jan 1996	

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 7 | BACK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 7: POD

POD: Plan Old Documentation

POD or `Plan Old Documentation` is a **VERY** simple method of documenting Perl programs. It is nothing more than text with keywords placed so that an interpreter program can translate these into specific commands (HTML script, man pages, nroff, etc.)

Paragraph Commands

=head1 heading	labeled as major text heading
=head2 heading	labeled as subheading
=item text	part of a list
=over N	intend text over by N characters
=back	restores the indent to the default value
=cut	signifies the end of the pod text
=pod	telling the compiler to lay off of through the next `=cut`

Format Commands

I<text>	italicize text, used for emphasis or variables
B<text>	embolden text, used for switches and programs
S<text>	text contains non-breaking spaces
C <code>	literal code
L <name>	A link (cross reference) to name
L <name>	manpage
L <name/ident>	item in manpage
L <name/"sec">	section in other manpage
L <"sec">	section in this manpage (the quotes are optional)
L </"sec">	ditto
F <file>	Used for filenames
X <index>	An index entry
Z <>	A zero-width character

You can embed pod documentation in your Perl scripts. Start your documentation with a `=head1` command at the beg, and *end it with* an `=cut` command. Perl will ignore the pod text. See any of the supplied library modules for examples. If you're going to put your pods at the end of the file, and you're using an `__END__` or `__DATA__` cut mark, make sure to put a blank

line there before the first pod directive.

The following translators exist:

- pod2man - convert pod to Unix man page format (nroff(1) and troff(1))
- pod2html - convert pod to HTML
- pod2latex - convert pod to LaTex
- pod2fm - convert pod to framemaker file

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 7 | BACK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 7
Packages and Modules

---

1. PRIMORDIAL GUESTBOOK MODULE

A promised we've returned to our guestbook. This time we're going to implement it using packages and modules! When you are creating the Guestbook package and module remember to think about the concepts of modularity and extensibility. As you might guess this won't be the last you see of the guestbook problem. Hopefully by the end of the class you will all have a **WORKING** Guestbook CGI application written in Perl.

For reference you might want to review Problem 4 in Assignment 4.

A. Guestbook Package

First let's convert our guestbook to a package. The overall functionality of the Guestbook package should be that described in Problem 4 in Assignment 4.

- The package name should be `Guestbook` and store it in a file called `Guestbook.pl`.
- The Guestbook package should contain, as a minimum, the following functions:
  - Function to parse all of the required user inputs
  - Function to create new guestbook if one does not exist
  - Function to add current user inputs to existing guestbook

  You may want to include more functions which these functions call. Also, feel free to use ANY EXISTING Perl module that you find helpful, but you are not required to do this.
- Demonstrate that your Guestbook package is working by writing a program called TestGB.pl.

B. Guestbook Module

Now that you have a guestbook package, convert it to a module.

- The module name should be `Guestbook` and store it in a file called `Guestbook.pm`.
- Add the necessary information and calls to your guestbook package to turn it into a module.
- Demonstrate that your Guestbook module is working by modifying TestGB.pl to use the Guestbook module.

Remember:
I want to see examples of the input and output to demonstrate that you've tested these programs.

[TITLE PAGE | TABLE OF CONTENTS | HOMEWORK | LECTURE 7]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Subject: Re: [MacPerl] Simple Explanation for @ISA?
Date: Wed, 30 Oct 1996 17:43:30 +0100
From: Matthias Ulrich Neeracher <neeri@iis.ee.ethz.ch>
To: mac-perl@iis.ee.ethz.ch

Robert Ke <rke@qualcomm.com> writes:
>Sorry to tell you, the @ISA has something to do with OOP.

It's actually even more confusing than that: While the @ISA mechanism often is used for OOP, it is not strictly related to it. You can do OOP without @ISA and use @ISA without OOP.

>It works like the @INC array which stores directories for perl to search
>for files in other directories.
>
>@ISA array contains names of classes so Perl could search for methods in
>other classes.

So far, the explanation is correct (and quite elegant, too). However, your terminology is more OOP biased than Perl itself in fact is. A more accurate statement would be:

The @ISA array contains names of *modules* so Perl can search for *subroutines* in other *modules*.

Stated this way, it becomes clear that @ISA works for OOP, but it also works without OOP. Some important examples are:

@ISA = qw(Exporter) : All modules which export names do this. A statement such as "use xxx(a b c)" causes a subroutine named "import" to be executed, and this subroutine is taken from the "Exporter" module.

@ISA = qw(DynaLoader) : All Mac toolbox modules do this to be able to dynamically load code fragments (This is done in the "bootstrap" routine).

@ISA = qw(AutoLoader) : This can be used to split up modules and load pieces on demand.

Matthias

Matthias Neeracher <neeri@iis.ee.ethz.ch> http://www.iis.ee.ethz.ch/~neeri
"If people are boasting that they use the best tools, you can figure that they can't find any competitive advantage to using those tools, or else they wouldn't be presenting them as a competitive advantage." -- Dave Winer

Subject: Re: [MacPerl] Simple Explanation for @ISA?
Date: Tue, 29 Oct 96 16:17:02 -0500
From: schinder@pjstoaster.pg.md.us (Paul Schinder)
Organization: NASA Goddard Space Flight Center
To: mac-perl@iis.ee.ethz.ch

gasster@pacificnet.net writes:
} I have read several Perl books and the man pages about
} the use of the @ISA array. I am looking for a simple explanation
} of why you need it, and what it does. Most of the discussions
} jump right into OOP related material and it makes understanding
} this array complicated

Having recently read the new Camel on this topic (it's a long flight from Tucson to Baltimore, especially when you have to change planes at Dallas/Fort Worth), I believe I understand it. (Although I'm not object orientedly correct, doing most of my scientific programming in C and FORTRAN, so my use of the terminology may not be exactly right.) The correct pronunciation of @ISA is "is a", and it basically functions like the @INC array. When you "require" or "use" a file that Perl doesn't already know about, it hunts through the directories in the @INC array until it finds it. Similarly, when you try to use a method on an object that Perl doesn't already know about, it hunts through the object's class, and then the classes named in @ISA (and also the base classes of those, I believe) for the method until it finds it. That seems to be all there is to it. You put into @ISA those classes from which you want to inherit methods. Kind of like @INC, but on the scale of subroutines rather than entire files. (Someone who knows more about this stuff correct me if I'm wrong.)

}
} thanks for any help...
} -sam
} --
} **
} Samuel D. Gasster
} gasster@pacificnet.net
} **
}

Paul J. Schinder
NASA Goddard Space Flight Center,
Code 693, Greenbelt, MD 20771 USA
schinder@pjstoaster.pg.md.us

Subject: Re: [MacPerl] Simple Explanation for @ISA?
Date: Tue, 29 Oct 1996 18:02:13 -0800
From: Steve R Webster <srw@zanshin.com>
To: gasster@pacificnet.net
References: 1

```
>Sorry to tell you, the @ISA has something to do with OOP.
>
>It works like the @INC array which stores directories for perl  to search
>for files in other directories.
>
>@ISA array contains names of classes so Perl could search for methods in
>other classes.
>
>If you are don't use OOP, you don't need @ISA.
>
>On Tue, 29 Oct 1996, Samuel D. Gasster wrote:
>
>> I have read several Perl books and the man pages about
>> the use of the @ISA array.  I am looking for a simple explanation
>> of why you need it, and what it does.  Most of the discussions
>> jump right into OOP related material and it makes understanding
>> this array complicated
>>
>> thanks for any help...
>> -sam

One hint that's nonobvious:  "ISA" means two words, "is a".  When you see
the sequence:

package Eagle;

@ISA = (Bird);

this means that the Eagle package "is a" Bird, which is the core of
object-oriented programming.  That is, Eagle objects know how to do all the
things that Bird objects know how to do, plus all the things defined in the
Eagle package, like, say, Eagle::rend_flesh() as well as Bird::squawk().

While Perl is not the place to start learning OOP, it does a fair job of
letting you do OOpish things.  Your best bet is to get a book/friend on OOP
& start reading/badgering.

-steve
```

LECTURE 8:
Perl GENERAL TOPICS

READING ASSIGNMENT:
TYPERL 5 Ch. 13, pp 433-442
Ch. 15 and 21

HOMEWORK ASSIGNMENT:
Assignment No. 8 (Due 11/25/96).

Perl MANPAGES:

- Perl Debugging: perldebug
- Perl Library Functions: perlfunc
- Perl IPC: perlipc

TOPICS

- Debugging Perl Scripts
- Process and Program Manipulation Functions
- UNIX System Functions (SUMMARY)
  - System Library Emulation Functions
  - Socket Functions
  - System V IPC Functions
- Where to go next?

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 9]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 8:
Debugging Perl Scripts

INTRODUCTION

Debugging Your Script

Debugging is the process of finding errors in your program or script. There are basically two kinds of errors. Those that are syntax related that cause your program NOT to compile and run, and those that are related to the program flow (and structure) or algorithm. The syntax errors will generally keep your program from running, and while tricky to track down in some cases, can usually be found by using the syntax error messages provided by Perl.

The algorithmic or program flow errors are sometimes much harder to find. This class of error also includes such things as using a global variable in one part of the program, thinking it is local in another part and modifying it. The Perl debugger can help you step through your code and track the flow and values of variables.

There are several ways to check your Perl script or program. You should always try one or more of these steps before running the debugger.

Check Syntax
perl -c scriptname

```
gasster>perl -c debug_me.pl
# /home/gasster/perlcourse/debug_me.pl syntax OK
```

Generate Warnings
perl -w scriptname

Consider a script which has the line `$test = 1;` and `$test` is used nowhere else.

```
gasster>perl -w debug_me.pl
# Identifier "main::test" used only once: possible typo.
File '/home/gasster/perlcourse/debug_me.pl'; Line 21
# /home/gasster/perlcourse/debug_me.pl syntax OK
```

Checking Syntax and Generate Warnings
perl -cw scriptname

You can also embedded the Perl switches into the Perl script. For example:

```
#!/usr/local/bin/perl -w
#

...the rest of your script...
```

This will turn on Perl's warnings when you try to execute the script.

The Perl Debugger

- **How to start the debugger**

  You can start the debugger from either the command line:

```
perl -d scriptname
```

or embed the -d switch into your program using

```
#!/usr/local/bin/perl -wd
#

...the rest of your script...
```

When you startup the debugger you get the following information:

```
gasster> perl debug_me.pl
Stack dump during die enabled outside of evals.

Loading DB routines from perl5db.pl patch level 0.94
Emacs support available.

Enter h or `h h' for help.

main::(debug_me.pl:21): $pi = 4.0 * atan2(1.,1.);  #useful constant
  DB<1>
```

The debugger is pre-loading some information, tells you that you can run it from `emacs` (Unix editor) and how to get help for the debug commands. The last two lines tell you the first executable line found (and the debugger is waiting to execute it) and `DB<1>` is the debugger prompt.

- **Exiting the debugger**

  To exit the debugger (before your program finishes) you can use `q` or the `EOF` command (either `^D` [Unix] or `^Z`[PC]).

- **Debugger resource file (Unix)**

  There are several things you can do to customize your debugging environment. The commands to do this go into the `.perldb` file in your home directory (this is for Unix, see your release notes to determine if your port of Perl supports a similar file). You can add aliases or command short cuts using:

```
$DB::alias{'len'} = 's/^len(.*)/p length($1)/';
```

This command line in the `.perldb` file is equivalent to the debugger command:

```
DB<4>p length($data)
```

Debugger Command Summary

This is a brief summary of some of the Debugger commands, similar to the on line debugger help.

h h
Summary of debugger commands.

h [db_command]
Get help [on a specific debugger command], enter |h to page.

T
Stack trace.

s [expr]
Single step [in expr].

n [expr]
Next, steps over subroutine calls [in expr].

<CR>
Repeat last n or s command.

r
Return from current subroutine.

c [line]
Continue; optionally inserts a one-time-only breakpoint at the specified line.

l min+incr
List incr+1 lines starting at min.

l min-max
List lines.

l line
List line;

l
List next window, allows you to page through the program.

-
List previous window.

w line
List window around line.

.
Return to the executed line.

l subname
List subroutine.

f filename
Switch to filename.

/pattern/
Search forwards for pattern; final / is optional.

?pattern?
Search backwards for pattern.

L
List breakpoints and actions.

S [[!]pattern]
List subroutine names [not] matching pattern.

t
Toggle trace mode.

t expr
Trace through execution of expr.

b [line] [condition]
Set breakpoint; line defaults to the current execution line; condition breaks if it evaluates to true, defaults to '1'.

b subname [condition]
Set breakpoint at first line of subroutine.

d [line]
Delete breakpoint for line.

D
Delete all breakpoints.

a [line] command
Set an action to be done before the line is executed. Sequence is: check for breakpoint, print line if necessary, do action, prompt user if breakpoint or step, evaluate line.

A
Delete all actions.

V [pkg [vars]]
List some (default all) variables in package (default current).

X [vars]
Same as "V currentpackage [vars]".

< command
Define command before prompt.

> command
Define command after prompt.

! number
Redo command (default previous command).

! -number
Redo number'th to last command.

H -number
Display last number commands (default all).

q or [EOF]
Quit the Perl debugger. You can also use [EOF] for your system, `^D` (Unix,Mac) or `^Z` (PC)

p expr
print the value of `expr` from inside the debugger. Same as `print DB::OUT expr` in current package.

= [alias value]
Define a command alias, or list current aliases.

command
Execute as a perl statement in current package.

R
Pure-man-restart of debugger, debugger state and command-line options are lost.

For a complete listing and discussion read either *TYPERL 5* or *Programming Perl*.

Debugger Sample Session

This section provides an example of debugging a Perl script. I suggest that you have available a copy of your program that include line numbers. You can either have another window open on your system with a text editor that includes line numbers, or a printed copy of your script with line numbers.

NOTE:
You can of course use Perl to get a copy of your script that includes line numbers by using a simple script, such as addlinenums.pl, or the one line perl command:

```
perl -ne 'print "$.\t$_";' filename > newfilename
```

The Script

We will be debugging the following script:

```
1    #!/usr/local/bin/perl -wd
2    #
3    #---------------------------------------------------------------#
4    #   Filename:   debug_me.pl
5    #
6    #   Algorithm Description:
7    #       Cartesian to polar coordinate conversion
8    #
9    #   Input/Output/Restrictions:
10   #       None/None/Don't run this program on Tuesdays.
11   #
12   #   Programmer:
13   #       S. Gasster
14   #
15   #   Revision History:
16   #       code started                Jan. 1, 1996
17   #       fixed minor syntax errors   Jan. 2, 1996
18   #
19   #---------------------------------------------------------------#
20   #
21   $pi = 4.0 * atan2(1.,1.);  #useful constant
22   $rad2deg = 180./$pi;       #convert radians to degrees
23
24   print "Cartesian to Polar Coordinate Conversion\n";
25
26
27   #print ("\nEnter a pair of values x,y (^D to quit)> ");
28
29   $input = &getUserInput("Enter a pair of values x,y");
30
31   # extract input values
32       ($x, $y) = split(/,/, $input );
33
34   # compute radius
35       $r = sqrt( $x*$x + $y*$y );
36
37   # compute angle using atan2
38   # Note that Perl's atan2 handles case where
39   # $x = 0.
40
41       $theta = $rad2deg * atan2( $y, $x);
42
43       printf ("Input (x,y) values (%.2g,%.2g)\n", $x, $y);
44       printf ("Radius = %.2g\nTheta = %.2g (deg.)\n", $r, $theta);
45
46   exit 1;  # successful exit
47
48   # subroutine to get user inputs
49   #
50   sub getUserInput {
51      my($prompt) = @_;       #prompt to get user input
52      my($UserInput);         #what the user enters
53
54      #get input from user, don't accept null input
55      do {
56              print ("$prompt> ");
57              $UserInput = <STDIN>;
58              chomp( $UserInput );    #remove newline character
59      } until ($UserInput ne "");
60
61      return $UserInput;
```

```
62    }
```

Sample Debug Session

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 8]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

addlinenums.pl

```
#!/usr/local/bin/perl -w
#
#-------------------------------------------------------------#
#
#
#   make a copy of any file
#   but add line numbers to the
#   beginning of each line
#
#   input file name: infile.ext
#   output file name: infile_n.ext
#
#  file: addlinenums.pl
#
#   s. gasster  Sat, Apr 9, 1995
#
#
#-------------------------------------------------------------#
#

$new_ext = "_n";

print "\nHere is a list of the files:\n";

foreach $fname (@ARGV) {

# first get the file name and path
     @list = split(/:/,$fname);
     $file = pop(@list);
     $path = join(":",@list);
     print "file path = $path\n";

     print "Old file name = $file\n";

    ($filename,$ext) = split(/\./,$file);

    $newfile = $filename . $new_ext . "." . $ext;

    $pathnewfile = join(":", ($path,$newfile));
    print "New file name = $pathnewfile\n";

# open files
    open(OLDFILE, "<$fname") or
      die ("Error opening file: $fname,  $!");

    open(NEWFILE, ">$pathnewfile") or
      die ("Error opening file: $pathnewfile,  $!");

    while ($line = <OLDFILE>) {

    print NEWFILE "$.   $line";
    }

    close (OLDFILE);
    close (NEWFILE);

}
exit (1);
```

Sample Perl Debug Session

```
gasster> perl debug_me.pl

Stack dump during die enabled outside of evals.

Loading DB routines from perl5db.pl patch level 0.94
Emacs support available.

Enter h or `h h' for help.

main::(debug_me.pl:21):  $pi = 4.0 * atan2(1.,1.);  #useful constant
  DB<1> w
18:     #
19:     #-----------------------------------------------------------------#
20:     #
21==>   $pi = 4.0 * atan2(1.,1.);  #useful constant
22:     $rad2deg = 180./$pi;       #convert radians to degrees
23:
24:     print "Cartesian to Polar Coorinate Conversion\n";
25:
26:
27:     #print ("\nEnter a pair of values x,y (^D to quit)> ");
  DB<1> b 41
  DB<2> L
41:         $theta = $rad2deg * atan2( $y, $x);
  break if (1)
  DB<2> b getUserInput
  DB<3> L
41:         $theta = $rad2deg * atan2( $y, $x);
  break if (1)
51:             my($prompt) = @_;       #prompt to get user input
  break if (1)
  DB<3> s
main::(debug_me.pl:22):  $rad2deg = 180./$pi;       #convert radians to degrees
  DB<3> s
main::(debug_me.pl:24):  print "Cartesian to Polar Coorinate Conversion\n";
  DB<3> X rad2deg pi
$rad2deg = '57.2957795130823'
$pi = '3.14159265358979'
  DB<4> p 2. * $pi
6.28318530717959
  DB<5> c
Cartesian to Polar Coorinate Conversion
main::getUserInput(debug_me.pl:51):
51:             my($prompt) = @_;       #prompt to get user input
  DB<5>
main::getUserInput(debug_me.pl:52):
52:             local($UserInput);              #what the user enters
  DB<5>
main::getUserInput(debug_me.pl:55):
55:             do {
  DB<5>
main::getUserInput(debug_me.pl:56):
56:                     print ("$prompt> ");
  DB<5>
Enter a pair of values x,y> main::getUserInput(debug_me.pl:57):
57:                     $UserInput = <STDIN>;
  DB<5>
1.,2.
main::getUserInput(debug_me.pl:58):
58:                     chomp( $UserInput );    #remove newline character
  DB<5> c
main::(debug_me.pl:41):     $theta = $rad2deg * atan2( $y, $x);
```

```
  DB<5> s
main::(debug_me.pl:43):     printf ("Input (x,y) values (%.2g,%.2g)\n", $x, $y);
  DB<5> X r theta
$theta = '63.434948822922'
$r = '2.23606797749979'
  DB<6> c
Input (x,y) values (1,2)
Radius = 2.2
Theta = 63 (deg.)
gasster>
```

[BACK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 8:
UNIX System Functions
A Summary

NOTE

These functions are primarily based on using Perl in a Unix environment. Many of these functions may not be defined in your particular port of Perl. If there is a function with the same name, it may not work in exactly the same manner on your system as it might on a Unix system. As always read your release notes that came with your port of Perl.

This section of the notes draws heavily on the Perl manual.

PROCESS AND PROGRAM MANIPULATION

What is a process?

A process is an instance of a program running on your computer. In Unix processes are started by the shell (or another program that works through the shell) and are given a Process ID (pid or PID), that is used to track the process.

On a Unix system you can use the `ps` command to determine what processes are running:

```
gasster> ps -cjf
     UID   PID  PPID  PGID   SID  CLS PRI    STIME TTY      TIME COMD
 gasster 17360 16544 17360 16544   TS  34 14:07:23 pts/8    0:00 ps -cjf
 gasster 16544 16538 16544 16544   TS  48 14:00:45 pts/8    0:03 -tcsh
gasster>
```

Note that the PID of my shell is 16544, and when I did the ps -cjf command I forked another process with PID 17360 and PPID (parent pid) 16544, my shell pid.

---

eval EXPR
eval BLOCK

EXPR is parsed and executed as if it were a little Perl program. It is executed in the context of the current Perl program, so that any variable settings, subroutine or format definitions remain afterwards. The value returned is the value of the last expression evaluated, or a return statement may be used, just as with subroutines.

If there is a syntax error or runtime error, or a `die()` statement is executed, an undefined value is returned by `eval()` , and `$@` is set to the error message. If there was no error, `$@` is guaranteed to be a null string.

If EXPR is omitted, evaluates `$_`. The final semicolon, if any, may be omitted from the expression. Note that, since `eval()` traps otherwise-fatal errors, it is useful for determining whether a particular feature (such as `socket()` or `symlink()`) is implemented. It is also Perl's exception trapping mechanism, where the `die` operator is used to raise exceptions. If the code to be executed doesn't vary, you may use the `eval-BLOCK` form to trap run-time errors without incurring the penalty of recompiling each time. The error, if any, is still returned in `$@`.

Examples

```
# make divide-by-zero non-fatal
eval { $answer = $a / $b; }; warn $@ if $@;

# same thing, but less efficient
```

```
eval '$answer = $a / $b'; warn $@ if $@;

# a compile-time error
eval { $answer = };

# a run-time error
eval '$answer =';   # sets $@
```

---

system LIST

Does exactly the same thing as `exec LIST` except that a `fork` is done first, and the parent process waits for the child process to complete. Note that argument processing varies depending on the number of arguments. The return value is the exit status of the program as returned by the `wait()` call. To get the actual exit value divide by 256. See also exec.

---

fork

`fork` creates a new process just like the current one. It does a fork(2) system call. Returns the child pid to the parent process and 0 to the child process, or undef if the fork is unsuccessful. Note: unflushed buffers remain unflushed in both processes, which means you may need to set `$|` (`$AUTOFLUSH`) or call the `autoflush()` FileHandle method to avoid duplicate output. If you `fork()` without ever waiting on your children, you will accumulate zombies:

```
$SIG{CHLD} = sub { wait };
```

There's also the double-fork trick (error checking on `fork()` returns omitted);

```
unless ($pid = fork) {
    unless (fork) {
        exec "what you really wanna do";
        die "no exec";
        # ... or ...
        ## (some_perl_code_here)
        exit 0;
    }
    exit 0;
}
waitpid($pid,0);
```

---

pipe READHANDLE,WRITEHANDLE

Opens a pair of connected pipes like the corresponding system call. Note that if you set up a loop of piped processes, deadlock can occur unless you are very careful. In addition, note that Perl's pipes use stdio buffering, so you may need to set `$|` to flush your `WRITEHANDLE` after each command, depending on the application.

---

exec LIST

The `exec()` function executes a system command **AND NEVER RETURNS**. Use the `system()` function if you want it to return. If there is more than one argument in `LIST`, or if `LIST` is an array with more than one value, calls execvp(3) with the arguments in `LIST`. If there is only one scalar argument, the argument is checked for shell metacharacters. If there are any, the entire argument is passed to `/bin/sh -c` for parsing. If there are none, the argument is split into words and passed directly to execvp(), which is more efficient. Note: exec() (and system(0) do not flush your output buffer, so you may need to set `$|` to avoid lost output.

Examples

```
exec '/bin/echo', 'Your arguments are: ', @ARGV;
exec "sort $outfile | uniq";
```

If you don't really want to execute the first argument, but want to lie to the program you are executing about its own name, you can specify the program you actually want to run as an "indirect object" (without a comma) in front of the LIST. (This always forces interpretation of the LIST as a multi-valued list, even if there is only a single scalar in the list.) Example:

```
$shell = '/bin/csh';
exec $shell '-sh';          # pretend it's a login shell
```

or, more directly,

```
exec {'/bin/csh'} '-sh';    # pretend it's a login shell
```

---

syscall LIST

Calls the system call specified as the first element of the list, passing the remaining elements as arguments to the system call. If unimplemented, produces a fatal error. The arguments are interpreted as follows: if a given argument is numeric, the argument is passed as an int. If not, the pointer to the string value is passed. You are responsible to make sure a string is pre-extended long enough to receive any result that might be written into a string. If your integer arguments are not literals and have never been interpreted in a numeric context, you may need to add 0 to them to force them to look like numbers.

```
require 'syscall.ph';               # may need to run h2ph
syscall(&SYS_write, fileno(STDOUT), "hi there\n", 9);
```

Note
Perl only supports passing of up to 14 arguments to your system call, which in practice should usually suffice.

EXECUTION CONTROL

sleep EXPR
sleep

Causes the script to sleep for EXPR seconds, or forever if no EXPR. May be interrupted by sending the process a SIGALRM. Returns the number of seconds actually slept. You probably cannot mix `alarm()` and `sleep()` calls, since `sleep()` is often implemented using `alarm()`. On some older systems, it may sleep up to a full second less than what you requested, depending on how it counts seconds. Most modern systems always sleep the full amount. For delays of finer granularity than one second, you may use Perl's `syscall()` interface to access `setitimer(2)` if your system supports it.

---

alarm SECONDS

Arranges to have a SIGALRM delivered to this process after the specified number of seconds have elapsed. See `sleep()` above.

---

wait

Waits for a child process to terminate and returns the process ID number (pid) of the deceased process, or -1 if there are no child processes. The status is returned in `$?`.

---

waitpid PID,FLAGS

Waits for a *particular* child process to terminate and returns the pid of the deceased process, or -1 if there is no such child process. The status is returned in `$?`. If you say

```
use POSIX "wait_h";
```

```
...
waitpid(-1,&WNOHANG);
```

then you can do a non-blocking wait for any process. Non-blocking wait is only available on machines supporting either the waitpid(2) or wait4(2) system calls. However, waiting for a particular pid with FLAGS of 0 is implemented everywhere. (Perl emulates the system call by remembering the status values of processes that have exited but have not been harvested by the Perl script yet.)

UNIX SYSTEM LIBRARY EMULATION FUNCTIONS

Fetching User and Group Information

Unix systems provide information about each user in two files: the `/etc/passwd` and `/etc/group`.

`($gname, $gpasswd, $gid, $gmembers) = getgrent;`
getgrent reads the Unix user group definition file `/etc/group` and returns four componets from each entry. The function reads one line each time it is called. When the file is exhausted `getgrent` returns the empty list. The values returned are:

```
$gname = user group name
$gpasswd = user group password, if it exists
$gid = group ID number
$gmembers = group members
```

`setgrent()` and **`endgrent()`**
The setgrent() function rewinds to the beginning of the /etc/group file, allowing getgrent to read from the start of the file. endgrent tells Perl that you are done reading the /etc/group file and frees the memory used to store the file.

`($gname, $gpasswd, $gid, $gmembers) = getgrnam($name);`
`getgrname` allows you to retrieve the group file entry corresponding to a particular group name.

`($gname, $gpasswd, $gid, $gmembers) = getgrgid($id);`
`getgrgid` allows you to retrieve the group file entry corresponding to a particular group ID number.

Fetching Network Information

Unix systems provide information about the local network connections in the files: `/etc/networks` and `/etc/hosts`.

`($netname, $altnames, $addrtype, $net) = getnetent();`
getnetent enables you to step through the file /etc/networks, which lists the names and addresses of the networks your machine is on. The return values are:

```
$netname = name of the network
$altnames = list alternative names for the network (returned as
            a character string with names separated by a space)
$addrtype = address format type; default is AF_INET (address format internet )
$net = internet address of the network; represented as a string of four bytes;
       unpack into a scalar string using the unpack() function.
```

Example

```
#!/usr/local/bin/perl

print ("Networks this machine is connected to:\n");
while (($name, $altnames, $addrtype, $rawaddr) = getnetent()) {
        @addrbytes = unpack ("C4", $rawaddr);
        $address = join (".", @addrbytes);
```

```
        print ("$name, at address $address\n");
    }
```

($netname, $altnames, $addrtype, $addr) = getnetbyaddr($inaddr, $inaddrtype);
Retrieves a line from the /etc/networks file that matches the address specified by the address $inaddr. $inaddrtype is always AF_INET.

($netname, $altnames, $addrtype, $addr) = getnetbyname($inname);
Retrieves a line from the /etc/networks file that matches the address specified by the machine name $inname.

setnetent($ko); and endnetent;
The setnetent function rewinds the /etc/networks file, allowing you to start reading this file from the start. If the scalar $ko is non-zero, the file is not closed after getnetbyname or getnetbyaddr are called; allowing multiple reads. If $ko = 0, the file is closed.

The endnetent function tells perl you are done with the /etc/networks file, closes the file and frees memory used to store the related information.

($hostname, $altnames, $addrtype, $len, $addr) = gethostbyaddr($inaddr, $inaddrtype);
Reads the /etc/hosts (or the equivalent name server) for the host name specified by the particular Internet address.

Inputs

```
$inaddr = Internet address to search for, stored in the same four byte format
          used by getnetbyaddr.
$inaddrtype = the Internet address type, AF_INET at present is the only one understood
```

Return List

```
$hostname = the host name corresponding to the internet address specified by inaddr.altnames

$addrtype = the Internet address type, AF_INET at present is the only one understood

$addr = a list of addresses (main address and alternatives) corresponding to the node
named $name.  Each address is stored as 4 byte integer.

$len = length of $addr; always four times the number of addresses returned.
```

($hostname, $altnames, $addrtype, $len, $addr) = gethostbyname($inname);
Similar to the gethostbyaddr function, except that it searches the /etc/hosts file for a specified machine or internet site name.

($hostname, $altnames, $addrtype, $len, $addr) = gethostent();
Reads the /etc/hosts (or the equivalent name server) one entry at a time, allowing you to step through the file. Each call returns the same five element list returned by gethostbyaddr and gethostbyname.

sethostent($ko); and endhostent();
The sethostent function rewinds the /etc/hosts file, allowing you to start reading this file from the start. If the scalar $ko is non-zero, the file is not closed after getnetbyname or getnetbyaddr are called; allowing multiple reads. If $ko = 0, the file is closed.

The endhostent function tells perl you are done with the /etc/hosts file, closes the file and frees memory used to store the related information.

Login, User and Process Information

$loginmane = getlogin();
getlogin returns the user ID under which you are currently logged in. The user ID is retrieved from the file: /etc/utmp.

pgroup = getpgrp(PID);

Returns the current process group for the specified PID, 0 for the current process. Will raise an exception if used on a machine that doesn't implement getpgrp(2). If PID is omitted, returns process group of current process.

setpgrp(PID, GROUPID);

Sets the current process group for the specified PID, 0 for the current process. PID is the ID of the process whose group you want to change; GRPUPID is the process group ID you want this process to belong to. Will produce a fatal error if used on a machine that doesn't implement setpgrp(2).

parentID = getppid();

Returns the process id of the parent process.

LIST = getpwnam(USERNAME)

The getpwnam function retrieves a passwd record for the given user login name. This function reads the /etc/passwd file. Returns the empty list if no corresponding entry. This function returns a list with the following elements:

```
($username,   #user name
$password,    #encrypted password field
$userid,      #user ID number
$groupid,     #group ID number to which user belongs
$quota,       #quota and comment can mean different
$comment,     #things on different systems - check man pages
$infofield,   #character string with information about user
$homedir,     #directory when user first logs in
$shell        #Unix shell that user is in when first logged in
 ) =   getpwnam(username);
```

LIST = getpwuid(UID)

The getpwuid function is similar to the getpwnam function in that it retrieves a similar LIST structure, however it is keyed off of the user ID number field.

LIST = getpwent

The getpwent function reads a single entry from the /etc/passwd file. The returned LIST as the same structure as for the getpwnam function.

setpwent($ko); and endpwent();

These functions allow you to manipulate the /etc/passwd file (assuming you have the proper privileges). The setpwent function rewinds the passwd file allowing access from the start. If $ko is nonzero, the Perl interpreter assumes that the file will be accessed again and keeps it store in memory; if zero then any passwd file information stored in memory is released. The endpwent function closes the passwd file the releases any memory associated with it.

PRIORITY = getpriority(CATEGORY, ID);

Returns the current priority for a process, a process group, or a user. (See getpriority(2).) Will raise a fatal exception if used on a machine that doesn't implement getpriority(2).

setpriority(CATEGORY, ID, PRIORITY);

Sets the current priority for a process, a process group, or a user. (See setpriority(2).) Will produce a fatal error if used on a machine that doesn't implement setpriority(2).

(NAME, ALIASES, NUMBER) = getprotoent();

Read the system protocol database, usually in the file /etc/protocols. See the man page for the system function *getprotoent(3)* for more information on all of the protocal functions. A typical example of the /etc/protocols file is:

```
gasster> cat /etc/protocols
#ident  "@(#)protocols  1.2     90/02/03 SMI"   /* SVr4.0 1.1   */

#
# Internet (IP) protocols
#
ip      0       IP      # internet protocol, pseudo protocol number
icmp    1       ICMP    # internet control message protocol
ggp     3       GGP     # gateway-gateway protocol
```

```
tcp     6       TCP     # transmission control protocol
egp     8       EGP     # exterior gateway protocol
pup     12      PUP     # PARC universal packet protocol
udp     17      UDP     # user datagram protocol
hmp     20      HMP     # host monitoring protocol
xns-idp 22      XNS-IDP # Xerox NS IDP
rdp     27      RDP     # "reliable datagram" protocol
```

getprotobyname and getprotobynumber

These function provide additional methods to search through the system protocol database file.

setprotoent and endprotoent

These function provide methods to manipulate the system protocol database file.

getservent

This function reads the system services database file usually in `/etc/services`. A typical example of some of the system services is shown below (some entries deleted to save space). See the man pages for `getservbyname(3)` for more information on all of the system services functions.

```
gasster> cat /etc/services
#ident  "@(#)services   1.9     93/09/10 SMI"   /* SVr4.0 1.8   */

#
# Network services, Internet style
#
tcpmux          1/tcp
# echo          7/tcp
# echo          7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
netstat         15/tcp
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
name            42/udp          nameserver
whois           43/tcp          nicname         # usually to sri-nic
domain          53/udp
domain          53/tcp
hostnames       101/tcp         hostname        # usually to sri-nic
sunrpc          111/udp         rpcbind
sunrpc          111/tcp         rpcbind
#
# Host specific functions
#
tftp            69/udp
rje             77/tcp
finger          79/tcp
http            80/tcp
link            87/tcp          ttylink
supdup          95/tcp
pop-2           109/tcp                         # Post Office
pop3            110/tcp
ident           113/tcp         auth tap        # identify users
uucp-path       117/tcp
nntp            119/tcp         usenet          # Network News Transfer
ntp             123/tcp                         # Network Time Protocol
ntp             123/udp                         # Network Time Protocol
imap2           143/tcp                         # imapd
NeWS            144/tcp         news            # Window System
#
...and more...
```

getservbyname and getservbyport

These functions allow you to search the system services database file either by name or port number.

setservent and endservent
These function provide methods to manipulate the system services database file.

SOCKET FUNCTIONS

What is a socket?

The Perl function calls for dealing with sockets have the same names as the corresponding system calls in C, but their arguments tend to differ for two reasons: first, Perl filehandles work differently than C file descriptors. Second, Perl already knows the length of its strings, so you don't need to pass that information.

accept NEWSOCKET,GENERICSOCKET
Accepts an incoming socket connect, just as the accept(2) system call does. Returns the packed address if it succeeded, FALSE otherwise.

bind SOCKET,NAME
Binds a network address to a socket, just as the bind system call does. Returns TRUE if it succeeded, FALSE otherwise. NAME should be a packed address of the appropriate type for the socket.

connect SOCKET,NAME
Attempts to connect to a remote socket, just as the connect system call does. Returns TRUE if it succeeded, FALSE otherwise. NAME should be a packed address of the appropriate type for the socket.

getpeername SOCKET
find the other hend of a socket connection Returns the packed sockaddr address of other end of the SOCKET connection.

```
use Socket;
$hersockaddr    = getpeername(SOCK);
($port, $iaddr) = unpack_sockaddr_in($hersockaddr);
$herhostname    = gethostbyaddr($iaddr, AF_INET);
$herstraddr     = inet_ntoa($iaddr);
```

getsockname SOCKET
retrieve the sockaddr for a given socket and return the packed sockaddr address of this end of the SOCKET connection.

```
use Socket;
$mysockaddr = getsockname(SOCK);
($port, $myaddr) = unpack_sockaddr_in($mysockaddr);
```

getsockopt SOCKET,LEVEL,OPTNAME
Returns the socket option requested, or undefined if there is an error.

listen SOCKET,QUEUESIZE
Does the same thing that the listen system call does. Returns TRUE if it succeeded, FALSE otherwise.

recv SOCKET,SCALAR,LEN,FLAGS
Receives a message on a socket. Attempts to receive LENGTH bytes of data into variable SCALAR from the specified SOCKET filehandle. Actually does a C *recvfrom()*, so that it can returns the address of the sender. Returns the undefined value if there's an error. SCALAR will be grown or shrunk to the length actually read. Takes the same flags as the system call of the same name.

send SOCKET,MSG,FLAGS,TO
send SOCKET,MSG,FLAGS
Sends a message on a socket. Takes the same flags as the system call of the same name. On unconnected sockets you must specify a destination to send TO, in which case it does a C *sendto()*. Returns the number of characters sent,

or the undefined value if there is an error.

setsockopt SOCKET,LEVEL,OPTNAME,OPTVAL
Sets the socket option requested. Returns undefined if there is an error. OPTVAL may be specified as undef if you don't want to pass an argument.

shutdown SOCKET,HOW
Shuts down a socket connection in the manner indicated by HOW, which has the same interpretation as in the system call of the same name.

socket SOCKET,DOMAIN,TYPE,PROTOCOL
Opens a socket of the specified kind and attaches it to filehandle SOCKET. DOMAIN, TYPE and PROTOCOL are specified the same as for the system call of the same name. You should `use Socket;` first to get the proper definitions imported.

socketpair SOCKET1,SOCKET2,DOMAIN,TYPE,PROTOCOL
Creates an unnamed pair of sockets in the specified domain, of the specified type. DOMAIN, TYPE and PROTOCOL are specified the same as for the system call of the same name. If unimplemented, yields a fatal error. Returns TRUE if successful.

EXAMPLE: Internet TCP Clients and Servers

(From the Perl manual: perlipc section)

Use Internet-domain sockets when you want to do client-server communication that might extend to machines outside of your own system.

Here's a sample **TCP client using Internet-domain sockets:**

```
#!/usr/bin/perl -w
require 5.002;
use strict;
use Socket;
my ($remote,$port, $iaddr, $paddr, $proto, $line);
$remote  = shift || 'localhost';
$port    = shift || 2345;  # random port
if ($port =~ /\D/) { $port = getservbyname($port, 'tcp') }
die "No port" unless $port;
$iaddr   = inet_aton($remote)               || die "no host: $remote";
$paddr   = sockaddr_in($port, $iaddr);
$proto   = getprotobyname('tcp');
socket(SOCK, PF_INET, SOCK_STREAM, $proto)  || die "socket: $!";
connect(SOCK, $paddr)    || die "connect: $!";
while ($line = <SOCK>) {
    print $line;
}
close (SOCK)            || die "close: $!";
exit;
```

And here's a corresponding **server** to go along with it. We'll leave the address as INADDR_ANY so that the kernel can choose the appropriate interface on multihomed hosts:

```
#!/usr/bin/perl -Tw
require 5.002;
use strict;
BEGIN { $ENV{PATH} = '/usr/ucb:/bin' }
use Socket;
use Carp;
sub spawn;  # forward declaration
sub logmsg { print "$0 $$: @_ at ", scalar localtime, "\n" }
my $port = shift || 2345;
my $proto = getprotobyname('tcp');
```

```
socket(SERVER, PF_INET, SOCK_STREAM, $proto)        || die "socket: $!";
setsockopt(SERVER, SOL_SOCKET, SO_REUSEADDR, 1)     || die "setsockopt: $!";
bind(SERVER, sockaddr_in($port, INADDR_ANY))        || die "bind: $!";
listen(SERVER,5)                                    || die "listen: $!";
logmsg "server started on port $port";
my $waitedpid = 0;
my $paddr;
sub REAPER {
    $SIG{CHLD} = \&REAPER;  # loathe sysV
    $waitedpid = wait;
    logmsg "reaped $waitedpid" . ($? ? " with exit $?" : '');
}
$SIG{CHLD} = \&REAPER;
for ( $waitedpid = 0;
      ($paddr = accept(CLIENT,SERVER)) || $waitedpid;
      $waitedpid = 0, close CLIENT)
{
    next if $waitedpid;
    my($port,$iaddr) = sockaddr_in($paddr);
    my $name = gethostbyaddr($iaddr,AF_INET);
    logmsg "connection from $name [",
            inet_ntoa($iaddr), "]
            at port $port";
    spawn sub {
        print "Hello there, $name, it's now ", scalar localtime, "\n";
        exec '/usr/games/fortune'
            or confess "can't exec fortune: $!";
    };
}
sub spawn {
    my $coderef = shift;
    unless (@_ == 0 && $coderef && ref($coderef) eq 'CODE') {
        confess "usage: spawn CODEREF";
    }
    my $pid;
    if (!defined($pid = fork)) {
        logmsg "cannot fork: $!";
        return;
    } elsif ($pid) {
        logmsg "begat $pid";
        return; # i'm the parent
    }
    # else i'm the child -- go spawn
    open(STDIN,  "<&CLIENT")   || die "can't dup client to stdin";
    open(STDOUT, ">&CLIENT")   || die "can't dup client to stdout";
    ## open(STDERR, ">&STDOUT") || die "can't dup stdout to stderr";
    exit &$coderef();
}
```

DISCUSSION

This server takes the trouble to clone off a child version via `fork()` for each incoming request. That way it can handle many requests at once, which you might not always want. Even if you don't `fork()`, the `listen()` will allow that many pending connections. Forking servers have to be particularly careful about cleaning up their dead children (called *zombies* in Unix parlance), because otherwise you'll quickly fill up your process table.

We suggest that you use the `-T` flag to use taint checking (see the discussion on security later in this lecture) even if we aren't running setuid or setgid. This is always a good idea for servers and other programs run on behalf of someone else (like CGI scripts), because it lessens the chances that people from the outside will be able to compromise your system.

SYSTEM V INTERPROCESS COMMUNICATION (IPC) FUNCTIONS

The System V IPC functions refer to the interprocess communication capabilities used in Unix System V (five not the letter "V"). On a Unix system various process need to communicate with one another and Perl provides an interface to the system

level functions that peform these operations under SysV. The three types of communication objects are: message queues, shared memory and semaphores. This type of IPC is restricted to processes (which of course can be unrelated) on the same host machine. Other Unix systems can be configured to support SysV IPC, talk to your systems administrator for more information.

The SysV IPC functions provide access to some pretty detailed level access to the systems functionality, and are not really necessary for perform CGI programming tasks. Therefore I have provided only minimal information here. For a good introduction try: *Perl by Example*, Ellie Quigley, Prentice Hall, NJ (1995).

If you are serious about using SysV IPC I suggest you also consult the Perl module `IPC::SysV`.

In order to use the SysV IPC function you need to have the following Perl library header files available. If not you need to run the Perl h2ph program to create them. Read the Perl manual or on-line man pages and Quigley's book on how to do all this.

- `ipc.ph`
- `msg.ph`
- `sem.ph`
- `shm.ph`

Note that these files are created from the corresponding C language header files with the same name and `.h` extension. You will then have to load these into your Perl programs, as necessary, using the `require` function.

Most of the functions listed below require various arguments `(KEY, FLAGS, CMDs)` which not discussed here. Consult the above listed header files for more information and any documentation provided with your system on SysV IPC.

- **Message Queues**
  Message queues provide a mechanism for processes to send data to each other.

  msgget
  `msgid = msgget(KEY, FLAGS);`
  Similar to a file open, the msgget function sets up the message queue. Returns a message queue ID if sucessful, null string if failed.

  msgctl
  `msgctl(MSGID, CMD, ARG);`
  The `msgctl` function allows you to retrieve information about the message queue, set permission or remove the queue.

  msgsnd
  `msgsnd(MSGID, MESSAGE, FLAGS);`
  The `msgsnd` function adds a message to a queue associated with message ID `MSGID`.

  msgrcv
  `msgrcv(MSGID, VARIABLE, SIZE, TYPE, FLAGS);`
  The msgrcv function retrieves a message from the message queue associated with message ID `MSGID`. Returns true if successful, false if failed.

- **Semaphores**
  Semaphores are a mechanism for effciently managing and synchronizing shared resources on the system. If two processes are sharing memory and both try to write to the memory at the same time, the semaphore is used to lock the memory while one process is using it, and release the lock when it is done.

  semget
  `semid = semget(KEY, NSEMS, FLAG);`
  The semget function creates the semaphore and returns the corresponding ID number.

  semop
  semop(SEMID, SEMOPSTRING);
  The semop function is used to manipulate the semaphore. It determines whether a shared resource should be acquired or released by a process. SEMID is the ID number returned by `semget`. The `SEMOPSTRING` consists

of the following components each being stored as a short int (generally the pack function is used to create the SEMOPSTRING).

- The number of semaphores
- The semaphore operation
- The semaphore flags

semctl

```
semctl(SEMID, SEMCMD, SEMARG);
```

The senctl function enables you to set options and manipulate semaphores.

- **Shared Memory**

Shared memory allows multiple processes to access a common segment of memory. There is no fool proof method for controlling access to shared memory so one generally resorts to using semaphores. This is really not for the faint at heart, and you might ask your systems adminstrator about using this functionaly.

shmget

```
shmid = shmget(KEY, SIZE, FLAG);
```

The shmget function allows you to create shared memory. The SIZE argument is the size in bytes of the shared memory segment. The function returns the shared memory ID.

shmctl

```
shmctl(SHMID, SHMCMD, SHMARG);
```

The shmctl allows control over the options for shared memory segments. The shared memory is reference by the SHMID valued returned by shmget. The commands to affect shared memory are passed as SHMCMD and any arguements as SHMARG.

shmread

```
shmread(SHMID, RETVAL, POS, SIZE);
```

The shmread function is used to read data from the shared memory corresponding to SHMID. RETVAL is a scalar variable name used to store the data read from shared memory; POS is the number of bytes to skip over in shared memory; SIZE is the number of bytes to read.

shmwrite

```
shmwrite(SHMID, TEXT, POS, SIZE);
```

The shmwrite function is used to send data to the shared memory corresponding to SHMID. TEXT is the character string to write; POS is the number of bytes to skip over in shared memory; SIZE is the number of bytes to write.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 8]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 8: WHERE TO GO NEXT?

1. In this class we are going to discuss applications of Perl to CGI Programming. Perl is one of the most popular programming languages for WWW CGI programming.

2. Perl topics not covered in this course:

   A. **Databases**
   Perl has builtin functions (dbmopen() and dbmclose()) that allow you access to the standard Unix DBM databases files using associative arrays (see pp 428 of *TYPERL 5*).

   There are also extensive Perl modules for accessing a variety of databases types and formats. For example, see the AnyDBM_File Perl module documented in the perl manual or check the CPAN sites.

   B. **Perl Functions**
   There are many Perl functions we've not discussed. Read, or at least skim, the Perl manual to learn about these.

   C. **Object Oriented Perl (OOP) programming.**
   This is discussed in Ch. 19 of *TYPERL 5* , The Camel Book (Programming Perl, 2nd edition), the Perl manual and many other places. Should be part of an Advanced Perl Course.

   D. **Regular Expressions**
   You can never learn enough about regexp! There is a book on regexp coming out in the next 6 months published by O'Reilly & Associates.

3. **Resources**
   You have a good foundation to expand your knowledge of Perl!

   A. PROGRAM, PROGRAM AND PROGRAM in Perl to gain more experience with it!

   B. The Perl Journal

   C. News Groups and mailing lists

   D. Read The Perl Manual!

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 8]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 8
General Topics

---

1. **Perl Debugger Problem**

   Use the Perl debugger to *debug* (there are no bugs in the code as given) the Perl program given below.

   i. As a minimum you should make use of the following commands:

   ```
   s,b,p,l,w,L,X,H
   ```

   ii. You should set at least 3 break points and list them out.

   iii. Display values for each variable after it is assigned a value. Do this at least once.

   Record the debugging session and show the results.

```
#!/usr/local/bin/perl -w
#
#-----------------------------------------------------------------#
#   Filename:   grades.pl
#
#   Algorithm Description:
#       A brief description of what the program is supposed
#       to do.
#
#   Input/Output/Restrictions:
#       input student name followed by 5 grades
#
#
#   Programmer:
#   from TYPERL 5, with slight
#   modifications by s. gasster
#
#   Revison History:
#       s. gasster  modified original code  9/12/96
#
#-----------------------------------------------------------------#
#
#
#
$prompt = <<EofP;

Enter student name followed by grades in
English history math science geography
^D to quit
EofP

$students = 0;
@subjects = ("English", "history", "mathematics",
             "science", "geography");
```



```
print "$prompt\n";
while ($line = <STDIN>) {
        $line =~ s/^\s+|\s+$//g;
        @words = split (/\s+/, $line);
        $students[$students++] = $words[0];
        for ($count = 1; $count <= 5; $count++) {
                $marks{$words[0].$subjects[$count-1]} =
                     $words[$count];
        }
print "$prompt\n";
}

# now print the failing grades, one student per line
foreach $student (sort (@students)) {
       $has_failed = 0;
        foreach $subject (sort (@subjects)) {
                if ($marks{$student.$subject} < 50) {
                        if ($has_failed == 0) {
                                $has_failed = 1;
                                print ("$student failed:");
                        }
                        print (" $subject");
                }
        }
        if ($has_failed == 1) {
                print ("\n");
        }
}
```

[TITLE PAGE| TABLE OF CONTENTS | HOMEWORK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 9: CGI PROGRAMMING

READING ASSIGNMENT:
CGI Programming on the WWW , Chapters 1-4, 12

HOMEWORK ASSIGNMENT:
Assignment No. 9 (Due 12/2/96).
I would like ALL homework turned in by the last class. Nothing will be accepted after the last class!

TOPICS

- CGI Overview (see Lecture 1) (revisited)
- Server Configuration (overview)
- Input to CGI
- Output from the CGI
- Forms and CGI
- Debugging and Testing CGI Applications
- **Review HTML on your own.**
  Consult the class links page for on-line references or any of a number of good books (Ch. 4 of course textbook)

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 10]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 9
CGI Programming

Due 12/2/96

Remember to watch out for:

- Spelling mistakes
- Getting paths to files correct
- Use correct directory/file path separators for your system
- Encode query string correctly when using CGI Lint
- Sending the correct HTTP header format
- Correctly loading/including any library files (common.pl or cgi-lib.pl)

---

1. GETTING STARTED

A. If you do not have access to an HTTP server skip to Item D.

B. If you have access to an HTTP server find out how it is configured; where are the log files; can you run CGI programs and if so from what directories.

C. Document this information including what type of server it is (Apache 1.1.1, NCSA 1.2, ...,etc.).

D. Download **CGI Lint**, either from the course web site or the O'Reilly ftp site. I've also put a sample `srm.conf` file on the course web site. Try anonyomous ftp to `ftp.muon.com` and look in `pub/ucla_perl`.

You might also want to grab the common.pl or cgi-lib.pl files. Review the documentation inside the CGI Lint Module; it wouldn't hurt to also just skim through the module. It should be quite readable to you at this stage.

E. For all of the problems below I want you to first test any CGI scripts using CGI Lint. Remember you will have to create two set of configuration files, for the GET and POST methods. For the POST method you may also need an additional data file. Also, don't forget the srm.conf file. You don't need a complete file, just the directives CGI Lint is looking for.

2. ABOUT THIS SERVER

A. Enter or download the About-This-Server CGI script (`ats.cgi - see ftp.muon.com`). Feel free to make any changes you need to so that it runs under your version of Perl, show me your version if you make changes.

B. Test it using CGI Lint. Show me the input file and all output files.

C. If you have access to an HTTP server run this CGI on your server. Show me the results (You can just print the page from your browser).

3. CGI FORM HANDLER

A. Implement an HTML form that has the following elements:
- User Name - text field
- User Email Address - text field
- City, State, Country - separate text fields
- Comment line - 60 character text field

B. Write a Perl CGI script that will parse this form input using the POST method. The CGI script should just return to the user the data he/she entered in the following format:

```
<H2 ALIGN=CENTER>Thank You for your Inputs</H2>
<P>
<HR>
Your Name:  username <BR>
Your Email Address:  email@address.com <BR>
Your Location: City, State, Country <BR>
Your Comment:  commentline <P>
Date: todaysdate&time <P>
<HR>
```

C. Test this script using CGI Lint. Show me all inputs and outputs.

OPTIONAL: If you have access to an HTTP server run it on the server.

[TITLE PAGE | TABLE OF CONTENTS | HOMEWORK | LECTURE 9]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 9: Server Configuration (Overview)

Why does a CGI programmer need to know this stuff?

The configuration of the server will tell you if you are even allowed to run CGI scripts on the server! It will also tell you where various log files are kept, which, as we shall see, is very useful in debugging CGI programs.

What does the server do?

- Accepts browser (client) requests
- Retrieves specified file(s) or executes appropriate CGI script
- Returns file contents or output from CGI script to client

Popular Servers

Unless otherwise indicated these servers run primarily on Unix platforms. Many of the Unix servers are publicly available and some of the commercial vendors may offer evaluation packages (check their web site).

- NCSA Server (National Center for Super Computing Applications
- Apache 1.1.1
- CERN Server (W3C)
- Netscape Family of servers
- WebStar (Mac)
- Microsoft Internet Information Server (IIS - Win/NT)
- WebSite from O'Reilly (Win95/NT)

Server Configuration

Given the large number of servers (and limited space/time) we will not cover the detailed configuration for all servers. We will provide a general overview for the Apache (http://www.apache.org) or NCSA (http://hoohoo.ncsa.uiuc.edu) type of server; many of the other server types have a similar configuration philosophy. As always, for your specific system and server RTFM!

What is server configuration?

Server configuration is the process of defining how the server will run on a given host platform. There are many parameters required by the server, each of these can be tailored for your system.

Servers are configured using **configuration files**. Rather than go through these files line-by-line, I've indicated some important directives in each file and provided sample copies.

`httpd.conf`
The server configuration file that specifies the basics of the server's operation. This is the main server configuration file.

`srm.conf`
The server resource configuration file, which specifies how the server should treat local resources when responding to a request.

With this document, you define the name space that users see of your http server. This file also defines server settings which affect how requests are serviced, and how results should be formatted.

`access.conf`
The access configuration file, which specifies what operations are allowed on what types of files and who has access.

This file defines server settings which affect which types of services are allowed, and in what circumstances.

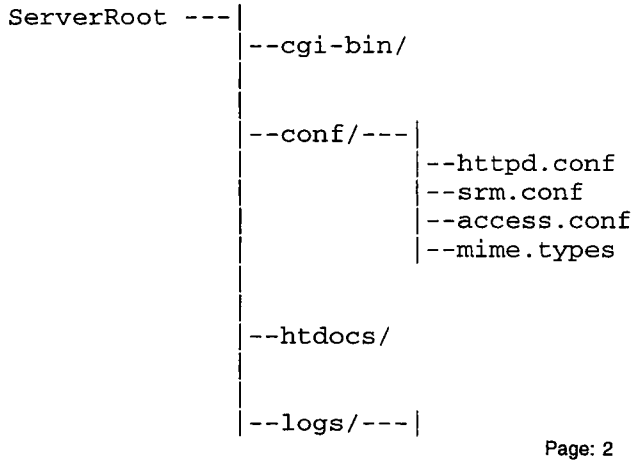
Each directory to which Apache has access, can be configured with respect to which services and features are allowed and/or disabled in that directory (and its subdirectories).

`mime.types`
The MIME types file; what MIME types should be associated with what file suffixes

NOTE:
Comment lines in these files start with # (seem familiar!).
Directives are not case sensitive, but by convention they are usually written with specific case conventions.

SERVER DIRECTORY TREE

Here is a typical layout of the httpd directory tree. This layout can be changed and modified by the systems administrator/webmaster, so talk to them about any specifies for your site.



```
|           |--access_log
|           |--error_log
```

`cgi-bin`
: CGI scripts are kept here. This is one of the places the server can look for scripts it is going to execute.

`conf`
: All the server configuration files go here

`htdocs`
: HTML documents and related information (graphics).

`logs`
: The server log files go here.

---

httpd.conf

The following directives are defined in the `httpd.conf` file.

```
# ServerRoot: The directory the server's config, error, and log files
# are kept in

ServerRoot /usr/local/etc/httpd

# ErrorLog: The location of the error log file. If this does not start
# with /, ServerRoot is prepended to it.

ErrorLog logs/error_log

# TransferLog: The location of the transfer log file. If this does not
# start with /, ServerRoot is prepended to it.

TransferLog logs/access_log
```

---

srm.conf

The following directives are defined in the `srm.conf` file.

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.

DocumentRoot /usr/local/etc/httpd/htdocs

# UserDir: The name of the directory which is appended onto a user's home
# directory if a ~user request is received.

UserDir public_html
```

```
# ScriptAlias: This controls which directories contain server scripts.
# Format: ScriptAlias fakename realname

#ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-bin/

# If you want to use server side includes, or CGI outside
# ScriptAliased directories, uncomment the following lines.

# AddType allows you to tweak mime.types without actually editing it, or to
# make certain files to be certain types.
# Format: AddType type/subtype ext1

# AddHandler allows you to map certain file extensions to "handlers",
# actions unrelated to filetype. These can be either built into the server
# or added with the Action command (see below)
# Format: AddHandler action-name ext1

# To use CGI scripts:
#AddHandler cgi-script .cgi

# To use server-parsed HTML files
#AddType text/html .shtml
#AddHandler server-parsed .shtml
```

---

access.conf

Probably the most important aspect of this file is the ability to control who has access to certain types of file and where. For example, you can turn off CGI script execution except in the `cgi-bin/` subdirectory of ServerRoot, thus all users must give their CGI scripts to the webmaster for inclusion into this directory (typically users don't have write access to the ServerRoot directory).

---

mime.types

The webmaster can add or subtract various types to/from this file depending on the user/customer needs. See our discussion of MIME types in Lecture 1. Here is just a short sample of some types found in the mime.types file.

```
audio/midi                      mid
application/pdf                 pdf
x-world/x-vrml                  wrl
application/mac-binhex40
application/postscript          ai eps ps
application/zip                 zip
application/x-tar               tar
audio/x-wav                     wav
audio/x-pn-realaudio            ram ra
image/gif                       gif GIF
image/jpeg                      jpeg jpg jpe
image/tiff                      tiff tif
text/html                       html htm HTM HTML
text/plain                      txt
text/richtext                   rtx
text/tab-separated-values       tsv
```

```
text/x-setext                   etx
video/mpeg                      mpeg mpg mpe mp2
video/quicktime                 qt mov
video/x-msvideo                 avi
video/x-sgi-movie               movie
xscript/movies                  xmz XMZ
xscript/files                   xst XST
application/x-httpd-php         pht phtml
audio/x-pn-realaudio-plugin     rpm
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 9]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 9: Input to CGI

CGI INPUT

- **Recall Client/Server Model**
  - Client (browser) sends request to server to access CGI program
  - Server executes CGI and makes available the following information
    1. client, server and user info
    2. form data (if appropriate)
    3. pathname information
- **How does your CGI program get its input?**
  - Environment Variables
  - Parsed HTML form input data
  - Additional path information
- **Environment Variables**

Much of this information is passed using **Environment Variables**. The environment variables can be accessed in Perl using the `%ENV` associative array.

- No HTML form is required to get the Environment information. The server sets these for you whether input was from a form or not. In fact the server doesn't really know whether the input was from a form or not.
- `HTTP_USER_AGENT` can be used to check the type of browser the client is using
- You can restrict access based on specific domains using the ENV-vars

```
REMOTE_ADDR
REMOTE_HOST
```

- Several of the ENV-vars are no longer set by most browsers. You used to be able to get someone's email address when they visited your website this way!!

```
REMOTE_USER
HTTP_REFERER
```

- Note that the server runs your CGI script with the `UID nobody`, not your `UID`! Therefore, none of your environment variable will be known to the script. Specifically, if you've defined any Perl libraries using the `PERLLIB` or `PERL5LIB` environment variables, they won't exist. Therefore, you might need to modify the `@INC` in your script. See the Perl Library Notes.

LIST OF CGI ENVIRONMENT VARIABLES

ENVIRONMENT VARIABLE	DESCRIPTION
GATEWAY_INTERFACE	CGI interface revision number used by server
SERVER_NAME	Server host name or IP address
SERVER_SOFTWARE	Name and version number of the server software that is answering the client request
SERVER_PROTOCOL	Name and revision of the information protocol the request came in with
SERVER_PORT	Port number of the host on which the server is running
REQUEST_METHOD	The method with which the information request was issued
PATH_INFO	Extra path information passed to the CGI program
PATH_TRANSLATED	Translated version of the information given by PATH_INFO
SCRIPT_NAME	The virtual path of the script being executed (e.g., /cgi-bin/program.pl)
DOCUMENT_ROOT	The directory from which Web documents are served
QUERY_STRING	Query information passed to the program
REMOTE_HOST	Remote hostname of the user making the request
REMOTE_ADDR	Remote IP address of the user making the request
AUTH_TYPE	Authentication method used to validate a user
REMOTE_USER	Authenticated name of the user
REMOTE_IDENT	The user making the request. *Only set* if NCSA *IdentityCheck* flag is enabled, and the client machine supports the RFC931 identification scheme (ident deamon).
CONTENT_TYPE	MIME type of the query data (e.g., *text/html*)
CONTENT_LENGTH	Length in bytes, or number of characters, of the data passed to the CGI program through STDIN
HTTP_FROM	The Email address of the user making the request. Most browsers do not support this variable.
HTTP_ACCEPT	List of the MIME types that the client can accept
HTTP_USER_AGENT	The browser the client is using to issue the request
HTTP_REFERER	The URL of the document that the client points to before accessing the CGI program

---

EXAMPLE: about_this_server.pl program

I rewrote this program using Here Doc formatting and subroutines in order to show how to better organize this program.

ats.pl

The HTML data produced by this script is:

```
    <HTML>
    <HEAD><TITLE>About this Server</TITLE></HEAD>
    <BODY><H1 ALIGN=CENTER>About this Server</H1>
    <HR ALIGN=CENTER>
    <B>Server Name:</B>     www.pacificnet.net <P>
    <B>Running on Port:</B> 80 <P>
    <B>Server Software:</B> Apache/1.1.1 <P>
    <B>Server Protocol:</B> HTTP/1.0 <P>
    <B>CGI Revision:</B>    CGI/1.1 <P>
    <B>Date:</B> Sun Nov 17 20:15:32 PST 1996
<P>
    <HR ALIGN=CENTER>
    </BODY></HTML>
```

OUTPUT

About this Server

Server Name: www.pacificnet.net

Running on Port: 80

Server Software: Apache/1.1.1

Server Protocol: HTTP/1.0

CGI Revision: CGI/1.1

Date: Sun Nov 17 20:15:32 PST 1996

NOTE: ats.pl

- HTTP HEADER - required by the server
- CGI Output is returned via STDOUT in this Perl script (more on this later), the server passes the script output along with the HTTP header back to the client (browser)

HTML FORM DATA

• HTML FORMS

HTML forms provide a method for the client to send data entered by the user to the server for handling by a CGI program. They provide a nice user interface, rather than requiring the user to encode the data.

The FORM tag has the METHOD attribute which can take the values: GET or POST.

```
<FORM ACTION="/cgi-bin/myscript.cgi" METHOD=GET>

....the rest of your form here....


<INPUT TYPE="submit" VALUE="Submit Form">
<INPUT TYPE="reset" VALUE="Reset Form">
</FORM>
```

• Query String

- Input data to CGI programs are transferred from the client to the server either as part of the URL (query string) or as part of the body of the HTTP request.
- The Query String is the method used to encapsulate the form data. The data is encoded with special characters being replaced by escaped values (see below).

• GET METHOD

- Data passed as part of the URL, in the Query String. The browser appends the information to the URL which specifies what CGI program to run using the ? character.

```
GET HTTP/1.0 /cgi-bin/guestbook.cgi?UserName=Sam%20Gasster&UserEmail=gasster%40abc.com
Accept: www/source
Accept: text/html
Accept: text/plain
User-Agent: Mozilla/3.0 (Macintosh; I; 68K)
```

- The server then stores all the data after the ? in the Environment variable QUERY_STRING
- The format for the Query String is a set of variable names and associated values separated by the = sign; and multiple sets separated by an & (ampersand). Special characters (spaces, tabs, newlines, ..., etc) are escaped as described below.

```
VarName1=Value1&VarName2=Value2
```

- The browser or the server may limit the amount of data (in bytes) that can be sent using the GET method.

• POST METHOD

- The POST method pass data to the server as part of the HTTP request body:

```
POST /cgi-bin/guestbook.cgi  HTTP/1.0
Accept: www/source
Accept: text/html
Accept: text/plain
User-Agent: Mozilla/3.0 (Macintosh; I; 68K)
Content-type: application/x-www-form-urlencoded
Content-length: 50


UserName=Sam%20Gasster&UserEmail=gasster%40abc.com
```

- POST uses the same format for the Query String that the GET method uses, it just gets sent to the server in a different manner.
- The server pass the HTTP body directly to the CGI program via STDIN, not using the `QUERY_STRING` Environment variable
- Requires additional HTTP header information: `Content-type` and `Content-length`
- Advantage of POST method is that there is no limit on the size of the Query String.

DATA ENCODING

- Since GET/POST methods encode the data into a single string we can't have any spaces or non-printing characters. This is especially true for the GET method where the data is sent as part of a URL.
- Need to convert special characters to simple printable encoded values.
- Encoding Method:

  A. Compute numerical `ASCII` value of character
  B. Convert numerical value to `HEX`
  C. Precede `HEX` value with a `%` sign

- Examples

CHARACTER	ASCII VALUE	HEX VALUE	ENCODED VALUE
space	32	20	%20
\t	9	09	%09
@	64	40	%40

EXTRA Path INFO

See *CGI Programming on the WWW* for discussion.

Decoding Form Input Data

This is discussed in LECTURE 9: Forms and CGI.

WARNING: SECURITY ALERT



- **ALWAYS THINK TWICE ABOUT EXECUTING ANY SYSTEM COMMAND IN A CGI PROGRAM!**
- You should always look for holes where a *clever cracker* could access your system, for example

```
....
    $query_string = $ENV{QUERY_STRING};
    print '$query_string';
....
```

Since any user can access your CGI program they could do so with the following value for $query_string:

```
rm -fr /
```

which could delete most if not all the files on the system.

- **REMEMBER**: don't assume that all inputs will be what you assumed they would be, always check for tainted CGI input.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 9]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

```
#!/usr/local/bin/perl -w
#
#------------------------------------------------------------
#   file: ats.pl
#
#   return server Environment variable to user
#   as an HTML document
#   adapted from:
#   CGI Programming on the World Wide Web, by S. Gundavaram
#
#   s. gasster  1996
#
#------------------------------------------------------------
#
#   declare subroutines
#
    sub PrintHeader;
    sub HTMLtop;
    sub HTMLbot;
    sub PrintServerENV;
#
#   define constants
#
$pagetitle   = "About this Server";
$pageheading = "About this Server";

#
#   execute the main body of the program
#
    PrintHeader;
    HTMLtop($pagetitle, $pageheading);
    PrintServerENV;
    HTMLbot;

#------------------------------------------------------------
#
#   define subroutines
#

# print the required HTTP content type header
sub PrintHeader {
    print "Content-type: text/html", "\n\n";
}

# print the top part of an HTML page

sub HTMLtop {
    local($pagetitle, $pageheading) = @_;

    print<<"End_of_Text";
    <HTML>
    <HEAD><TITLE>$pagetitle</TITLE></HEAD>
    <BODY><H1 ALIGN=CENTER>$pageheading</H1>
    <HR ALIGN=CENTER>
End_of_Text
}

# print some of the Environment variables

sub PrintServerENV {
```

```
    my ($date) = `date`;

    print<<"End_of_Text";
    <B>Server Name:</B>     $ENV{'SERVER_NAME'} <P>
    <B>Running on Port:</B> $ENV{'SERVER_PORT'} <P>
    <B>Server Software:</B> $ENV{'SERVER_SOFTWARE'} <P>
    <B>Server Protocol:</B> $ENV{'SERVER_PROTOCOL'} <P>
    <B>CGI Revision:</B>    $ENV{'GATEWAY_INTERFACE'} <P>
    <B>Date:</B> $date <P>
End_of_Text
}

# print the bottom part of an HTML page

sub HTMLbot {
    print<<"End_of_Text";
    <HR ALIGN=CENTER>
    </BODY></HTML>
End_of_Text
}
#
#-----------------------------------------------------------------
#

__END__
```

LECTURE 9: Output from the CGI

CGI OUTPUT

HTTP Preliminaries

- HTTP (HyperText Transfer Protocol) is the language that Web clients (browsers) and servers use to communicate.
- General format of HTTP transactions

  i. Client Request or Server Response line
  ii. Header Section (followed by a blank line)
  iii. Entity Body

Client Request Line

- Contains METHOD (GET or POST, there are other methods)
- URI indicating file or resource client is querying

  URI
  Uniform Resource Indicator - general term for all valid addressing schemes supported on the WWW. Most common is the URL Uniform Resource Locator. A URI consists of three fields:

  ```
  protocol://<domain name>/<requested file>
  ```

  - Valid protocols can be: FTP, WAIS, GOPHER, Telnet and HTTP.
  - Domain names specify host systems/servers and are usually of the form: `hoohoo.ncsa.uiuc.edu`
  - The requested file is the path specification to either a directory, file or CGI program

- HTTP Version Number

Example

```
GET /cgi-bin/myscript.cgi HTTP/1.0
```

Server Response Line

- HTTP Version Number
- Status code (number) of request
- Text of status code

Example

```
HTTP/1.0 200 OK
```

HTTP Headers

Four categories

General
Information not related to the client, server or HTTP.

Request
Preferred document formats and server parameters

Response
Information about the server sending the response

Entity
Information about the data being sent between the client and server

HEADER Syntax

a. Can be in any order (header lines)

b. Case-insensitive

c. Can extend over multiple lines by preceding each extra line by at least one space or tab

d. Header name followed by a colon (:)

e. Header block must end in a blank line

SOME VALID HTTP HEADERS (includes Netscape extensions)

HEADER	DESCRIPTION
`Cache-Control`	Specifies caching directives in comma-separated list. (ex: `no-cache`)
`Pragma:no-cache`	Turns document caching on and off. Replaced in `HTTP/1.1` by `Cache-Control`.
`Accept`	Specifies media types that the client prefers to accept (`MIME` *type/subtype*).
`Content-length`	Length in bytes of the output stream (implies binary data).
`Content-type`	MIME type of the output stream.
`Expires`	Date and time when the document is no longer valid and should be reloaded by the browser.
`Location`	Server redirection; specifies `URI` of new document location.
`Status`	Status of the request (cannot be sent as part of a complete header).
`Refresh`	Client reloads specified document.
`Set-Cookie`	Client stores specified data. Useful for keeping track of data between requests.

A complete list of HTTP headers can be found at
http://www.w3.org/hypertext/WWW/Protocols/HTTP/Object_Headers.html

Accept Types and Content Types

Client lets the server know what type of documents it can accept. This is determined by browser (client) type and configuration using helper applications. The server stores this information/data in the HTTP_ACCEPT Environment variable. You could, for example, test to see if the client can accept Shockwave files, and if not send them a *shock-less* web page.

Example

```
#!/usr/local/bin/perl

$gif_image = 'logo.gif';
$jpeg_image = 'logo.jpeg';
$plain_text = 'logo.txt';

$accept_types = $ENV{'HTTP_ACCEPT'};

if ($accept_types =~ m#image/gif#) {
    $logo = $gif_image;
} elsif ($accept_types =~ m#image/jpeg#) {
    $logo = $jpeg_image;
} else {
    $logo = $plain_text;
}

....rest of CGI script....
```

`Content-type` header also lets client know whats coming so it can figure out how to handle it.

EXAMPLE

```
#!/usr/local/bin/perl
#-------------------------------------------#
#   file: send_icon.cgi
#
#  simple perl script
#  to send the user the specified
#  icon
#
#   don't use with LARGE images
#
#  s. gasster  1996
#
#  after a script from
#  CGI Prog. on the WWW, S. Gundavaram
#-------------------------------------------#
#
```

```
$the_icon = "sun.gif";  # image icon to send

$gif_image = join ("/", $ENV{'DOCUMENT_ROOT'}, $the_icon);

if (open (ICON, "<$gif_image")) {
    $num_bytes = (stat ($gif_image))[7];

    print "Content-type: image/gif", "\n";
    print "Content-length: $num_bytes", "\n\n";

    print <ICON>;
    close (ICON);
} else {
    print "Content-type: text/plain", "\n\n";
    print "Sorry! I cannot open the file $gif_image!", "\n";
}

exit (1);
```

The result is:



Server Redirection

`Location Header` causes server to send the specified file. Can be used for sending static or existing documents, or docs on another server. Can also be used to notify client when a file has moved either temporarily or permanently.

See examples in *CGI Programming on the WWW* .

Expires and Pragma Headers

- `Expires:` *date*
  Specifies the date/time when document information may change or become invalid.

- `Pragma: no-cache`
  Replaced by Cache-Control header in HTTP/1.1

EXAMPLE

```
#!/usr/local/bin/perl
#
#-----------------------------------------------------------------#
#   Filename:   cache.cgi
#
#   Algorithm Description:
#       Test the Expires HTTP Header
#
#   Input/Output/Restrictions:
```

```
#       None/None/Don't run this program on Tuesdays.
#
#   Programmer:
#       S. Gasster
#
#   Revision History:
#       code started                Jan. 1, 1996
#       fixed minor syntax errors   Jan. 2, 1996
#
#-----------------------------------------------------------------#
#
BEGIN { unshift( @INC, "/home/gasster/perllib"); }
require "sdg_cgi_lib.pl";

$script_name = $ENV{'SCRIPT_NAME'};
$title = "Effects of Browser Caching";
$signature = "Sam was here!";

$otherheader = "Expires: Mon, 18 Nov 1996 00:02:33 PST";

chomp ($current_date = `date`);

&PrintHeader($otherheader);

&HTMLtop($title, $current_date);

print <<BODY;
  <P>
  <A HREF="$script_name">Click here to run again!</A>
  <P>
BODY

&HTMLbot($signature);

exit (1);
```

RESULT (cache_cgi.html)

The above page will not be reloaded unless you specifically do a RELOAD or until the page has expired.

STATUS Codes

- The first line of the HTTP server response indicates whether the client request was successful or not and why.
- As a CGI programmer, you can by-pass the server's default response headers and supply your own.
- Server Response Codes
- The complete list and discussion can be found at URL: http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTRESP.html

---

Complete (Non-Parsed) Headers

- CGI script can send partial HTTP header, but his requires the server to parse the output and complete the header.
- Some servers allow CGI output to be forward directly to the client without parsing, this however requires complete HTTP header.
- NCSA and CERN servers require programs that output complete headers to have the prefix: `nph-`

  `e.g., nph-test.cgi`
- Why do it?

  It can be slightly more efficient, requiring less server overhead. The exact amount of load reduction will depend on many factors such as document size and server load.

```
#!/usr/local/bin/perl
#
#   file: nph-test.cgi
#
$server_protocol = $ENV{'SERVER_PROTOCOL'};
$server_software = $ENV{'SERVER_SOFTWARE'};

print "$server_protocol 200 OK", "\n";
print "Server: $server_software", "\n";
print "Content-type: text/html", "\n\n";

print "OK, Here I go.\n<BR> I am going to count from 1 to 25!<P>", "\n";

for ($loop=1; $loop <= 25; $loop++) {
    print $loop, "<BR>\n";
    sleep (1);
}

print "All Done!", "\n";

exit (0);
```

RESULT

```
OK, Here I go.
I am going to count from 1 to 25!

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
16
17
18
19
20
21
22
23
24
25
All Done!
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 9]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 9:
Output from the CGI

HTTP SERVER RESPONSE CODE CATEGORIES

CODE RANGE	RESPONSE MEANING
100-199	Informational
200-299	Client Request Successful
300-399	Client Request Redirected; further action necessary
400-499	Client Request Incomplete
500-599	Server Errors

SAMPLE HTTP RESPONSE STATUS CODES

STATUS CODE	MESSAGE
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
301	Moved Permanently
302	Move Temporarily
303	See Other
304	Not Modified
400	Bad request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported

Mon Nov 18 00:01:47 PST 1996

Click here to run again!

Sam was here!

LECTURE 9: Forms and CGI

FORMS and CGI

Current HTML Forms specification

- HTML 2.0 and 3.0 are widely used, plus many of the NetScape Extensions
- HTML 3.2 Working Draft exists (see W3C at http://www.w3.org/ for more information.
- Limitations
  - Only text fields (not too bad since Perl can handle text to numeric conversions trivially)
  - User input not checked on the client side, CGI must check validity after user has submitted their input

HTML FORMS

I am assuming everyone in the class is familar with HTML. This is mainly for review. If you are not familar with HTML and Forms consult the list of references.

FORMS SUMMARY

Sending Data To The Server

What happens when you press [Submit Form]?

- x-www-form-urlencoded MIME Type
- Key/Value pairs are created:

```
key=value
```

- Each pair is separated by an & (ampersand)
- All special characters are translated to Hex codes

```
%20 ->  (space)
%21 -> !
```

- Fields with no entry get blank or null values

```
"key= "
```

This is very important - remember this when designing forms and the corresponding CGI program to check for blank values.

- Check boxes and Lists - these can have mulitple values which are simply repeated as Key/Value pairs, same Key, new Value.

GET vs POST

GET

- Form data is appended to the URL of the program when the client issues a request to the server
- Form data is then accessed on the server side using the QUERY_STRING ENV variable
- Allows you to access a CGI program without using a from - just include the data you want to send
- Caution: most servers limit the size, in bytes, of the URL-encoded data

POST

- Post request tells the server how much data is being transferred (Content-length) and this can be of any size, no limits

---

Parsing Form Data

We now understand how the form data is encoded and sent to our CGI script. The question now is how to decode the form data so that we can use it in our CGI script?

- Recall that for both `GET` and `POST` the form data is encoded into the query string format with the `Key=Value` (same as `VarName=Value`) pairs.
- This typically also includes `%HEX` encoded characters as well.
- Need a Perl function to decode this information.
- Requirements:
    - Determine how data was passed from the client. Use `REQUEST_METHOD` Environment variable.
    - `GET`: decompose `QUERY_STRING` and `PATH_INFO`
    - `POST`: decompose using `CONTENT_LENGTH` and query string data passed via `STDIN`
    - Split query string on `&` to separate `Key=Value` pairs
    - Decode `%HEX` characters
    - Create hash or associative array to hold the `Key=Value` pairs

Lucky for us several people have already done this work for us. You have several options, two of which I will discuss, when using Perl for CGI programming. The following library programs and Perl modules exist for handling CGI data input and output.

common.pl
: Simple Perl script for decoding form data. Written by S. Gundavaram. The funciton `parse_form_data` is used to extract the HTML form data.

cgi-lib.pl
: Perl function library Written by Steven E. Brenner. Very popular. The primary function is `ReadParse`. It reads in `GET` or `POST` data, converts it to unescaped text, and puts key/value pairs in `%in`, using "\0" to separate multiple selections Returns >0 if there was input, 0 if there was no input undef indicates some failure. Many *hard core* Perl programmers don't like this function since it not very *Perl 5ish*

CGI-lite.pm
: Written by Shishir Gundavaram. To quote from the module pod:

> The module can be used to handle and decode WWW form information. Both GET and POST requests can be processed. In the case of POST requests, the information can be one of two possible MIME types:
>
> ```
> application/x-www-form-urlencoded
> multipart/form-data
> ```
>
> This module is very light-weight, and can be thought of as an enhanced version of the old cgi-lib.pl library for Perl 4.0 by Steven Brenner.

CGI.pm
: Written by Lincoln Stein.

> This perl 5 library uses objects to create Web fill-out forms on the fly and to parse their contents. It is similar to `cgi-lib.pl` in some respects. It provides a simple interface for parsing and interpreting query strings passed to CGI scripts. However, it also offers a rich set of functions for creating fill-out forms. Instead of remembering the syntax for HTML form elements, you just make a series of perl function calls. An important fringe benefit of this is that the value of the previous query is used to initialize the form, so that the state of the form is preserved from invocation to invocation.

For the next severl lectures we'll use `cgi-lib.pl` and talk about the Perl CGI modules in later lectures.

Example

- modemspeed.html
- modemspeed.cgi

Decoding forms in other languages?

Read on your on...Hey this is a course on Perl!

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 9]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

OVERVIEW OF FORMS

FUNCTION	HTML CODE
Define Form	<FORM ACTION="URL" METHOD=GET|POST></FORM>
File Upload	<FORM ENCTYPE="multipart/form-data></FORM>
Input Field	<INPUT TYPE="TEXT|PASSWORD|CHECKBOX|RADIO| |MAGE|HIDDEN|SUBMIT|RESET">
Field Name	<INPUT NAME="***">
Field Value	<INPUT VALUE="***">
Checked?	<INPUT CHECKED> (checkboxes and radio boxes)
Field Size	<INPUT SIZE=?> (in characters)
Max Length	<INPUT MAXLENGTH=?> (in characters)
Selection List	<SELECT></SELECT>
Name of List	<SELECT NAME="***"></SELECT>
# of Options	<SELECT SIZE=?></SELECT>
Multiple Choice	<SELECT MULTIPLE> (can select more than one)
Option	<OPTION> (items that can be selected)
Default Option	<OPTION SELECTED>
Input Box Size	<TEXTAREA ROWS=? COLS=?></TEXTAREA>
Name of Box	<TEXTAREA NAME="***"></TEXTAREA>
Wrap Text	<TEXTAREA WRAP=OFF|VIRTUAL|PHYSICAL></TEXTAREA>

From: THE BARE BONES GUIDE TO HTML
by Kevin Werbach <barebones@werbach.com>
Version 2.0 HTML -- October 7, 1995

```
##++
##  This "mini" library contains the parse_form_data and
##  return_error subroutines. In some of the examples
##  described in the book (and in Examples.tar on the ORA
##  FTP site), I stated that you needed to use these subroutines,
##  but I didn't explicitly cover how to do this.
##
##  There are two ways to do so:
##
##     1. You can cut and paste these subroutines directly into
##        the program.
##
##     2. You can leave this file as is, and then insert the
##        following line at the top of your program:
##
##        require "common.pl";
##
##  Shishir Gundavaram
##  May 9, 1996
##--

sub parse_form_data
{
    local (*FORM_DATA) = @_;

    local ( $request_method, $query_string, @key_value_pairs,
           $key_value, $key, $value);

    $request_method = $ENV{'REQUEST_METHOD'};

    if ($request_method eq "GET") {
        $query_string = $ENV{'QUERY_STRING'};
    } elsif ($request_method eq "POST") {
        read (STDIN, $query_string, $ENV{'CONTENT_LENGTH'});
    } else {
        &return_error (500, "Server Error",
                       "Server uses unsupported method");
    }

    @key_value_pairs = split (/&/, $query_string);

    foreach $key_value (@key_value_pairs) {
        ($key, $value) = split (/=/, $key_value);
        $value =~ tr/+/ /;
        $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C", hex ($1))/eg;

        if (defined($FORM_DATA{$key})) {
        $FORM_DATA{$key} = join ("\0", $FORM_DATA{$key}, $value);
        } else {
            $FORM_DATA{$key} = $value;
        }
    }
}

sub return_error
{
    local ($status, $keyword, $message) = @_;

    print "Content-type: text/html", "\n";
    print "Status: ", $status, " ", $keyword, "\n\n";

    print <<End_of_Error;
```



```
<title>CGI Program - Unexpected Error</title>
<h1>$keyword</h1>
<hr>$message</hr>
Please contact $webmaster for more information.

End_of_Error

    exit(1);
}

1;
```

```
# Perl Routines to Manipulate CGI input
# S.E.Brenner@bioc.cam.ac.uk
# $Id: cgi-lib.pl,v 2.8 1996/03/30 01:36:33 brenner Rel $
#
# Copyright (c) 1996 Steven E. Brenner
# Unpublished work.
# Permission granted to use and modify this library so long as the
# copyright above is maintained, modifications are documented, and
# credit is given for any use of the library.
#
# Thanks are due to many people for reporting bugs and suggestions
# especially Meng Weng Wong, Maki Watanabe, Bo Frese Rasmussen,
# Andrew Dalke, Mark-Jason Dominus, Dave Dittrich, Jason Mathews

# For more information, see:
#     http://www.bio.cam.ac.uk/cgi-lib/

($cgi_lib'version = '$Revision: 2.8 $') =~ s/[^.\d]//g;


# Parameters affecting cgi-lib behavior
# User-configurable parameters affecting file upload.
$cgi_lib'maxdata    = 131072;    # maximum bytes to accept via POST - 2^17
$cgi_lib'writefiles =      0;    # directory to which to write files, or
                                 # 0 if files should not be written
$cgi_lib'filepre    = "cgi-lib"; # Prefix of file names, in directory above

# Do not change the following parameters unless you have special reasons
$cgi_lib'bufsize  =  8192;    # default buffer size when reading multipart
$cgi_lib'maxbound =   100;    # maximum boundary length to be encounterd
$cgi_lib'headerout =    0;    # indicates whether the header has been printed


# ReadParse
# Reads in GET or POST data, converts it to unescaped text, and puts
# key/value pairs in %in, using "\0" to separate multiple selections

# Returns >0 if there was input, 0 if there was no input
# undef indicates some failure.

# Now that cgi scripts can be put in the normal file space, it is useful
# to combine both the form and the script in one place.  If no parameters
# are given (i.e., ReadParse returns FALSE), then a form could be output.

# If a reference to a hash is given, then the data will be stored in that
# hash, but the data from $in and @in will become inaccessable.
# If a variable-glob (e.g., *cgi_input) is the first parameter to ReadParse,
# information is stored there, rather than in $in, @in, and %in.
# Second, third, and fourth parameters fill associative arrays analagous to
# %in with data relevant to file uploads.

# If no method is given, the script will process both command-line arguments
# of the form: name=value and any text that is in $ENV{'QUERY_STRING'}
# This is intended to aid debugging and may be changed in future releases

sub ReadParse {
  local (*in) = shift if @_;    # CGI input
  local (*incfn,                # Client's filename (may not be provided)
    *inct,                  # Client's content-type (may not be provided)
    *insfn) = @_;           # Server's filename (for spooled files)
  local ($len, $type, $meth, $errflag, $cmdflag, $perlwarn);
```



```
  # Disable warnings as this code deliberately uses local and environment
  # variables which are preset to undef (i.e., not explicitly initialized)
  $perlwarn = $^W;
  $^W = 0;

  # Get several useful env variables
  $type = $ENV{'CONTENT_TYPE'};
  $len  = $ENV{'CONTENT_LENGTH'};
  $meth = $ENV{'REQUEST_METHOD'};

  if ($len > $cgi_lib'maxdata) { #'
      &CgiDie("cgi-lib.pl: Request to receive too much data: $len bytes\n");
  }

  if (!defined $meth || $meth eq '' || $meth eq 'GET' ||
      $type eq 'application/x-www-form-urlencoded') {
    local ($key, $val, $i);

    # Read in text
    if (!defined $meth || $meth eq '') {
      $in = $ENV{'QUERY_STRING'};
      $cmdflag = 1;  # also use command-line options
    } elsif($meth eq 'GET' || $meth eq 'HEAD') {
      $in = $ENV{'QUERY_STRING'};
    } elsif ($meth eq 'POST') {
       $errflag = (read(STDIN, $in, $len) != $len);
    } else {
      &CgiDie("cgi-lib.pl: Unknown request method: $meth\n");
    }

    @in = split(/[&;]/,$in);
    push(@in, @ARGV) if $cmdflag; # add command-line parameters

    foreach $i (0 .. $#in) {
      # Convert plus to space
      $in[$i] =~ s/\+/ /g;

      # Split into key and value.
      ($key, $val) = split(/=/,$in[$i],2); # splits on the first =.

      # Convert %XX from hex numbers to alphanumeric
      $key =~ s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;
      $val =~ s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;

      # Associate key and value
      $in{$key} .= "\0" if (defined($in{$key})); # \0 is the multiple separator
      $in{$key} .= $val;
    }

  } elsif ($ENV{'CONTENT_TYPE'} =~ m#^multipart/form-data#) {
    # for efficiency, compile multipart code only if needed
$errflag = !(eval <<'END_MULTIPART');

    local ($buf, $boundary, $head, @heads, $cd, $ct, $fname, $ctype, $blen);
    local ($bpos, $lpos, $left, $amt, $fn, $ser);
    local ($bufsize, $maxbound, $writefiles) =
      ($cgi_lib'bufsize, $cgi_lib'maxbound, $cgi_lib'writefiles);


    # The following lines exist solely to eliminate spurious warning messages
    $buf = '';
```

```
    ($boundary) = $type =~ /boundary="([^"]+)"/; #";   # find boundary
    ($boundary) = $type =~ /boundary=(\S+)/ unless $boundary;
    &CgiDie ("Boundary not provided") unless $boundary;
    $boundary =  "--" . $boundary;
    $blen = length ($boundary);

    if ($ENV{'REQUEST_METHOD'} ne 'POST') {
      &CgiDie("Invalid request method for  multipart/form-data: $meth\n");
    }

    if ($writefiles) {
      local($me);
      stat ($writefiles);
      $writefiles = "/tmp" unless  -d _ && -r _ && -w _;
      # ($me) = $0 =~ m#([^/]*)$#;
      $writefiles .= "/$cgi_lib'filepre";
    }

    # read in the data and split into parts:
    # put headers in @in and data in %in
    # General algorithm:
    #   There are two dividers: the border and the '\r\n\r\n' between
    # header and body.  Iterate between searching for these
    #   Retain a buffer of size(bufsize+maxbound); the latter part is
    # to ensure that dividers don't get lost by wrapping between two bufs
    #   Look for a divider in the current batch.  If not found, then
    # save all of bufsize, move the maxbound extra buffer to the front of
    # the buffer, and read in a new bufsize bytes.  If a divider is found,
    # save everything up to the divider.  Then empty the buffer of everything
    # up to the end of the divider.  Refill buffer to bufsize+maxbound
    #   Note slightly odd organization.  Code before BODY: really goes with
    # code following HEAD:, but is put first to 'pre-fill' buffers.  BODY:
    # is placed before HEAD: because we first need to discard any 'preface,'
    # which would be analagous to a body without a preceeding head.

    $left = $len;
   PART: # find each part of the multi-part while reading data
    while (1) {
      last PART if $errflag;

      $amt = ($left > $bufsize+$maxbound-length($buf)
          ?  $bufsize+$maxbound-length($buf): $left);
      $errflag = (read(STDIN, $buf, $amt, length($buf)) != $amt);
      $left -= $amt;

      $in{$name} .= "\0" if defined $in{$name};
      $in{$name} .= $fn if $fn;

      $name=~/([-\w]+)/;  # This allows $insfn{$name} to be untainted
      if (defined $1) {
        $insfn{$1} .= "\0" if defined $insfn{$1};
        $insfn{$1} .= $fn if $fn;
      }

     BODY:
      while (($bpos = index($buf, $boundary)) == -1) {
        if ($name) {  # if no $name, then it's the prologue -- discard
          if ($fn) { print FILE substr($buf, 0, $bufsize); }
          else     { $in{$name} .= substr($buf, 0, $bufsize); }
        }
        $buf = substr($buf, $bufsize);
        $amt = ($left > $bufsize ? $bufsize : $left); #$maxbound==length($buf);
```



```
        $errflag = (read(STDIN, $buf, $amt, $maxbound) != $amt);
        $left -= $amt;
      }
      if (defined $name) {  # if no $name, then it's the prologue -- discard
        if ($fn) { print FILE substr($buf, 0, $bpos-2); }
        else     { $in{$name} .= substr($buf, 0, $bpos-2); } # kill last \r\n
      }
      close (FILE);
      last PART if substr($buf, $bpos + $blen, 4) eq "--\r\n";
      substr($buf, 0, $bpos+$blen+2) = '';
      $amt = ($left > $bufsize+$maxbound-length($buf)
          ? $bufsize+$maxbound-length($buf) : $left);
      $errflag = (read(STDIN, $buf, $amt, length($buf)) != $amt);
      $left -= $amt;


      undef $head;  undef $fn;
     HEAD:
      while (($lpos = index($buf, "\r\n\r\n")) == -1) {
        $head .= substr($buf, 0, $bufsize);
        $buf = substr($buf, $bufsize);
        $amt = ($left > $bufsize ? $bufsize : $left); #$maxbound==length($buf);
        $errflag = (read(STDIN, $buf, $amt, $maxbound) != $amt);
        $left -= $amt;
      }
      $head .= substr($buf, 0, $lpos+2);
      push (@in, $head);
      @heads = split("\r\n", $head);
      ($cd) = grep (/^\s*Content-Disposition:/i, @heads);
      ($ct) = grep (/^\s*Content-Type:/i, @heads);

      ($name) = $cd =~ /\bname="([^"]+)"/i; #";
      ($name) = $cd =~ /\bname=([^\s:;]+)/i unless defined $name;

      ($fname) = $cd =~ /\bfilename="([^"]*)"/i; #"; # filename can be null-str
      ($fname) = $cd =~ /\bfilename=([^\s:;]+)/i unless defined $fname;
      $incfn{$name} .= (defined $in{$name} ? "\0" : "") . $fname;

      ($ctype) = $ct =~ /^\s*Content-type:\s*"([^"]+)"/i;  #";
      ($ctype) = $ct =~ /^\s*Content-Type:\s*([^\s:;]+)/i unless defined $ctype;
      $inct{$name} .= (defined $in{$name} ? "\0" : "") . $ctype;

      if ($writefiles && defined $fname) {
        $ser++;
     $fn = $writefiles . ".$$.$ser";
     open (FILE, ">$fn") || &CgiDie("Couldn't open $fn\n");
      }
      substr($buf, 0, $lpos+4) = '';
      undef $fname;
      undef $ctype;
    }

1;
END_MULTIPART
  &CgiDie($@) if $errflag;
  } else {
    &CgiDie("cgi-lib.pl: Unknown Content-type: $ENV{'CONTENT_TYPE'}\n");
  }


  $^W = $perlwarn;
```

```
  return ($errflag ? undef :  scalar(@in));
}


# PrintHeader
# Returns the magic line which tells WWW that we're an HTML document

sub PrintHeader {
  return "Content-type: text/html\n\n";
}


# HtmlTop
# Returns the <head> of a document and the beginning of the body
# with the title and a body <h1> header as specified by the parameter

sub HtmlTop
{
  local ($title) = @_;

  return <<END_OF_TEXT;
<html>
<head>
<title>$title</title>
</head>
<body>
END_OF_TEXT
}


# HtmlBot
# Returns the </body>, </html> codes for the bottom of every HTML page

sub HtmlBot
{
  return "</body>\n</html>\n";
}


# SplitParam
# Splits a multi-valued parameter into a list of the constituent parameters

sub SplitParam
{
  local ($param) = @_;
  local (@params) = split ("\0", $param);
  return (wantarray ? @params : $params[0]);
}


# MethGet
# Return true if this cgi call was using the GET request, false otherwise

sub MethGet {
  return (defined $ENV{'REQUEST_METHOD'} && $ENV{'REQUEST_METHOD'} eq "GET");
}


# MethPost
# Return true if this cgi call was using the POST request, false otherwise

sub MethPost {
```



```
  return (defined $ENV{'REQUEST_METHOD'} && $ENV{'REQUEST_METHOD'} eq "POST");
}


# MyBaseUrl
# Returns the base URL to the script (i.e., no extra path or query string)
sub MyBaseUrl {
  local ($ret, $perlwarn);
  $perlwarn = $^W; $^W = 0;
  $ret = 'http://' . $ENV{'SERVER_NAME'} .
         ($ENV{'SERVER_PORT'} != 80 ? ":$ENV{'SERVER_PORT'}" : '') .
         $ENV{'SCRIPT_NAME'};
  $^W = $perlwarn;
  return $ret;
}


# MyFullUrl
# Returns the full URL to the script (i.e., with extra path or query string)
sub MyFullUrl {
  local ($ret, $perlwarn);
  $perlwarn = $^W; $^W = 0;
  $ret = 'http://' . $ENV{'SERVER_NAME'} .
         ($ENV{'SERVER_PORT'} != 80 ? ":$ENV{'SERVER_PORT'}" : '') .
         $ENV{'SCRIPT_NAME'} . $ENV{'PATH_INFO'} .
         (length ($ENV{'QUERY_STRING'}) ? "?$ENV{'QUERY_STRING'}" : '');
  $^W = $perlwarn;
  return $ret;
}


# MyURL
# Returns the base URL to the script (i.e., no extra path or query string)
# This is obsolete and will be removed in later versions
sub MyURL  {
  return &MyBaseUrl;
}


# CgiError
# Prints out an error message which which containes appropriate headers,
# markup, etcetera.
# Parameters:
#  If no parameters, gives a generic error message
#  Otherwise, the first parameter will be the title and the rest will
#  be given as different paragraphs of the body

sub CgiError {
  local (@msg) = @_;
  local ($i,$name);

  if (!@msg) {
    $name = &MyFullUrl;
    @msg = ("Error: script $name encountered fatal error\n");
  };

  if (!$cgi_lib'headerout) { #')
    print &PrintHeader;
    print "<html>\n<head>\n<title>$msg[0]</title>\n</head>\n<body>\n";
  }
  print "<h1>$msg[0]</h1>\n";
  foreach $i (1 .. $#msg) {
```

```
    print "<p>$msg[$i]</p>\n";
  }

  $cgi_lib'headerout++;
}


# CgiDie
# Identical to CgiError, but also quits with the passed error message.

sub CgiDie {
  local (@msg) = @_;
  &CgiError (@msg);
  die @msg;
}


# PrintVariables
# Nicely formats variables.  Three calling options:
# A non-null associative array - prints the items in that array
# A type-glob - prints the items in the associated assoc array
# nothing - defaults to use %in
# Typical use: &PrintVariables()

sub PrintVariables {
  local (*in) = @_ if @_ == 1;
  local (%in) = @_ if @_ > 1;
  local ($out, $key, $output);

  $output =  "\n<dl compact>\n";
  foreach $key (sort keys(%in)) {
    foreach (split("\0", $in{$key})) {
      ($out = $_) =~ s/\n/<br>\n/g;
      $output .=  "<dt><b>$key</b>\n <dd>:<i>$out</i>:<br>\n";
    }
  }
  $output .=  "</dl>\n";

  return $output;
}

# PrintEnv
# Nicely formats all environment variables and returns HTML string
sub PrintEnv {
  &PrintVariables(*ENV);
}


# The following lines exist only to avoid warning messages
$cgi_lib'writefiles =  $cgi_lib'writefiles;
$cgi_lib'bufsize    =  $cgi_lib'bufsize ;
$cgi_lib'maxbound   =  $cgi_lib'maxbound;
$cgi_lib'version    =  $cgi_lib'version;

1; #return true
```

MODEM SPEED SURVEY FORM

UserName

Email Address

Modem Speed

14.4 ○

28.8 ○

ISDN ◉

Submit Form | Reset Form

```
#!/usr/local/bin/perl
#
#-----------------------------------------------------------------#
#   Filename:   modemspeed.cgi
#
#   Algorithm Description:
#       Parse the user data from the modem speed survey form
#
#   Input/Output/Restrictions:
#       None/None/Don't run this program on Tuesdays.
#
#   Programmer:
#       s. gasster
#
#   Revison History:
#       code started                Jan. 1, 1996
#       fixed minor syntax errors   Jan. 2, 1996
#
#-----------------------------------------------------------------#
#
BEGIN { unshift( @INC, "/home/gasster/perllib"); }

$title = 'Modem Speed Survey';

#   libraries or modules
#
require "cgi-lib.pl";    #load the cgi-lib functions

&ReadParse(*input);     #extract the form data


# return data nicely formatted

print &PrintHeader;

print &HtmlTop($title);

# return the body of the HTML document

print<<End_of_Body;
    <H1 ALIGN=CENTER>MODEM SPEED SURVEY</H1>
    <P>
    <HR ALIGN=CENTER NOSHADE SIZE="2">
    <P>
    Thank you $input{'UserName'} for filling out the survey form<P>
    Your email address was given as:  $input{'UserEmail'} <P>
    Your Modem Speed is:  $input{'ModemSpeed'}
    <P>
    <HR ALIGN=CENTER NOSHADE SIZE="2">
End_of_Body

print &HtmlBot;
```

Lecture 9: Debugging and Testing CGI Applications

As with any type of computer programming you will encounter bugs and errors when working with CGI programming. The process of tracking down errors can be a little trickier since you will be dealing with send data over the network, parsing form input and interacting with the HTTP server. This section discusses some techniques that will hopefully make this process easier.

What to do first?

- Use modular programming. Using small simple functions to build your application will make debugging easier. You can test each function separately to make sure it does what you expect.
- Localize variables.
- Always check your script syntax and look for compiler warnings.

```
perl -cw mycgi.pl
```

Tracking down errors

When the server encounters an error in handling your CGI script either on input or output it will generate an error message.

- Run CGI Lint First!
- Server Log Files
  The server will write errors to the log file defined in the `srm.conf` file. Server access will be logged to the access_log file defined there as well.
- Make use of STDERR
  When the server is running your script any output directed to `STDERR` will go the the `ErrorLog` file.

Server Error messages

Here are some sample server error messages.

```
[Sun Nov 17 20:13:31 1996] access to /exp/home/gasster/public_html/ats.cgi failed
for 207.171.20.52, reason: file permissions deny server execution
Can't locate sdg_cgi_lib.pl in @INC at /exp/home/gasster/public_html/cache.cgi line 23.

[Sun Nov 17 23:28:13 1996] access to /exp/home/gasster/public_html/cache.cgi failed
for 207.171.20.37, reason: Premature end of script headers
```

```
[Sun Nov 17 23:35:40 1996] access to /exp/home/gasster/public_html/cache.cgi failed
 for 207.171.20.37, reason: file permissions deny server execution

[Sun Nov 17 23:36:10 1996] access to /exp/home/gasster/public_html/cache.cgi failed
for 207.171.20.37, reason: malformed header from script. Bad header=    <HTML>
Undefined subroutine &main::printHeader called at /exp/home/gasster/public_html/
cache.cgi line 31.

[Sun Nov 17 23:41:16 1996] access to /exp/home/gasster/public_html/cache.cgi fai
led for 207.171.20.37, reason: Premature end of script headers
```

Access Log examples

```
207.171.20.52 - - [17/Nov/1996:20:13:31 -0800] "GET /~gasster/ats.cgi HTTP/1.0" 403 -
207.171.20.52 - - [17/Nov/1996:20:13:52 -0800] "GET /~gasster/ats.cgi HTTP/1.0" 200 397
207.171.20.52 - - [17/Nov/1996:20:15:32 -0800] "GET /~gasster/ats.cgi HTTP/1.0" 200 443
207.171.20.52 - - [17/Nov/1996:20:38:45 -0800] "GET /~gasster/summary.cgi HTTP/1.0" 200 1960
207.171.20.37 - - [17/Nov/1996:23:00:42 -0800] "GET /~gasster/send_icon.cgi HTTP/1.0" 200 1971
```

Common Errors

- Programmer Errors (that's you!)

  For the server to execute your script it must know the following:
  - Where is the script; is the path a valid CGI location?
  - What kind of program is the CGI script?
  - Does the server have permission to execute the script?
    The CGI scripts run by the server are typically run under the UID of *nobody* which is set in the `httpd.conf` file:

    ```
    # User/Group: The name (or #number) of the user/group to run httpd as.

    User nobody
    Group ftp
    ```

    - nobody has minimal privileges
    - Scripts must be world executable
    - Any files created by script may not have the permission you expect

    Example: CGI Script file permissions

    ```
    -rwxr--r-x   1 gasster  users       1807 Nov 17 20:26 ats.cgi
    -rwxr--r-x   1 gasster  users        902 Nov 18 00:04 cache.cgi
    -rw-r--r-x   1 gasster  users        748 Nov 15 22:24 check_browser.cgi
    -rwxr-xr-x   1 gasster  users       1244 Nov 16 11:16 modemspeed.cgi
    -rw-r--r-x   1 gasster  users        452 Nov 18 00:49 nph-test.cgi
    -rw-r--r-x   1 gasster  users        773 Nov 17 23:06 send_icon.cgi
    -rwxr-xr-x   1 gasster  users       1918 Nov 15 22:24 summary.cgi
    ```

- Malformed Headers (Error 500)

  You didn't send correct HTTP header back. Typically, it didn't get printed at all, or it was not followed by a blank line.

Programming and System Errors

- General Guidelines
  - Always check the return codes from system commands. Did they do what you expected?
    - `open`
    - `eval`
    - `system`
  - Open to a pipe
    - Checking the return status once the pipe filehandle is closed.
    - Check open status of pipe
    - Check return status from shell commands once the pipe is closed
    - Use `$?` after `close()` to check status
    - Trick using `kill()` function may not always work
  - Don't try to set ENV variables
    CGI is run as a child process and can not change its environment
  - Make use of the ErrorLog file as discussed above

Server Simulation and CGI Lint

It may be possible to test your CGI program without running the program using the HTTP server. This has the advantange of testing *off-line* from the server so that any problems won't affect the performance of the server and other users on your system. Many ISPs might not approve of using their server for hours of CGI script debugging if it impacts the performance for everyone else. Try to test your script as throughly as possible before putting it on-line.

In many cases it is possible to test the CGI script off-line from the server. To understand this we will review what the server actually does when it parses an HTML form or document. The process that we will describe has been nicely packaged by the author of our text book, Shishir Gundavaram, into a Perl program called CGI Lint. CGI Lint is discussed below, first the review of the httpd process.

- What does the server do?

  A. Receives a client request to serve a resource (file or execute a CGI script)
  B. Is request a CGI script?
  C. If YES: pass Environment variables and input stream to CGI via STDIN and wait for CGI output
  D. Send file requested or CGI output back to client

- To simulate this process we need only perform Step C. This requires that we:
  - Set Environment variables and provide any necessary inputs via STDIN
  - Catch CGI script output

- If we run the CGI under our UID, then we can set the Environment variables using the appropriate syntax for our Unix shell

- GET Method requires we set the QUERY_STRING, which we could do since we know how its encoded. Once the environment variables are set we just execute the CGI script from the command line

```
perl myscript.cgi
```

- POST Method requires we set the CONTENT_LENGTH, which we could do since we can write the input query string to a file and either count the number of characters by hand or use the Unix command:

```
wc -c post_data.txt
```

Once the environment variables are set we just execute the CGI script from the command line, piping the input data to STDIN

```
cat post_data.txt | perl myscript.cgi
```

In both cases we would expect the CGI output to go to STDOUT, unless we are writting it to a file from within the script or redirect it from the command line. This would all work, but it would be nice if some of this could be slightly automated, and maybe even have some additional error checking. Well someone has thought of this...**CGI Lint - a Perl Module.**

CGI Lint

- **CGI_Lint-1.0.pm** is a Perl module written by Shishir Gundarvarm. You should down load it from either the O'Reilly ftp site or the course ftp site (anonymous ftp to ftp.muon.com, pub/ucla_perl).

- CGI Lint automates the process described above, to some extent. I can think of a few things I would do to extend it since I'm so lazy!

- Requires
  - Configuration files for either GET or POST methods that contain the Environment variables
  - Location of server srm.conf file - located using SERVER_ROOT and conf directory name
  - Input data file for POST method
  - Path to script is formed from SERVER_ROOT and SCRIPT_NAME

- QUERY_STRING setup
  - You do not have to perform the %HEX encoding
  - However, if the query string does contain special characters such as %, & or = you will need to escape them with a \.

- My suggestion is to create your own local testing environment. From your home directory level create a conf/ directory and copy the server srm.conf file there. Create another directory (call it anything you like... cgitest/) to put your CGI scripts, CGI Lint configuration files and POST input files. If you put the CGI_Lint.pm in your perllib/ directory and have your path set to include that directory then you can execute the CGI_Lint file directly (assuming you set the execution permission).

CGI LINT EXAMPLE

Simple example of a form to get the User Name, User Email Address and the User's Modem Speed. The files required are:

- modemspeed.html (HTML Form)
- modemspeed.cgi (Perl CGI script)
- post_cgilint.conf (CGI Lint POST Method Config file)
- modemspeed.dat (CGI Lint POST Method input data file)
- srm.conf (Server resource configuration file)

- srm.conf (Server resource configuration file)

---

modemspeed.html

```
<HR>
<FORM ACTION="/cgibin/modemspeed.cgi" METHOD=POST>
<TABLE BORDER="0" CELLPADDING="2" CELLSPACING="2">
<TR>
        <TD><B>UserName</B></TD>
        <TD><INPUT TYPE="text" NAME=""></TD>
</TR>
<TR>
        <TD><B>Email Address</B></TD>
        <TD><INPUT TYPE="text" NAME=""></TD>
</TR>
<P>
<TR>
        <TD><B>Modem Speed</B></TD>
</TR>

<TR><TD ALIGN=LEFT>14.4 <INPUT TYPE="radio" NAME="ModemSpeed" VALUE="14.4" ></TD></TR>
<TR><TD ALIGN=LEFT>28.8 <INPUT TYPE="radio" NAME="ModemSpeed" VALUE="28.8" ></TD></TR>
<TR><TD ALIGN=LEFT>ISDN <INPUT TYPE="radio" NAME="ModemSpeed" VALUE="ISDN" CHECKED></TD></TR>
</TABLE>
<INPUT TYPE="submit" VALUE="Submit Form">
<INPUT TYPE="reset" VALUE="Reset Form">
</FORM>
<HR>
```

MODEM SPEED SURVEY FORM

---

modemspeed.cgi

Here is the CGI program with no errors. I'll add some errors so we can see what happens with CGI Lint. Note that it uses cgi-lib.pl to parse the input data, and several other functions from that library script.

```
#!/usr/local/bin/perl
#
#--------------------------------------------------------------#
#   Filename:   modemspeed.cgi
#
#   Algorithm Description:
#       Parse the user data from the modem speed survey form
#
#   Input/Output/Restrictions:
#       None/None/Don't run this program on Tuesdays.
#
#   Programmer:
#       s. gasster
#
#   Revison History:
#       code started                Jan. 1, 1996
#       fixed minor syntax errors   Jan. 2, 1996
#
```

```
#-------------------------------------------------------------#
#
BEGIN { unshift( @INC, "/home/gasster/perllib"); }

$title = 'Modem Speed Survey';

#   libraries or modules
#
require "cgi-lib.pl";    #load the cgi-lib functions

&ReadParse(*input);     #extract the form data


# return data nicely formatted

print &PrintHeader;

print &HtmlTop($title);

# return the body of the HTML document

print<<End_of_Body;
    <H1 ALIGN=CENTER>MODEM SPEED SURVEY</H1>
    <P>
    <HR ALIGN=CENTER NOSHADE SIZE="2">
    <P>
    Thank you $input{'UserName'} for filling out the survey form<P>
    Your email address was given as:  $input{'UserEmail'} <P>
    Your Modem Speed is:  $input{'ModemSpeed'}
    <P>
    <HR ALIGN=CENTER NOSHADE SIZE="2">
End_of_Body

print &HtmlBot;
```

---

post_cgilint.conf and modemspeed.dat

Since we are using the POST method we CGI Lint requires a conguration file to set the ENV variables and an input data file.

post_cgilint.conf

Note that I have left some of the Environment variables blank. That's ok, CGI Lint just assumes those values are not set.

Also note that CGI Lint will expect to find the `srm.conf` file in the directory `/home/gasster/conf`.

```
REQUEST_METHOD      =   POST
HTTP_ACCEPT         =   image/gif, image/x-xbitmap, image/jpeg, */*
SERVER_PROTOCOL     =   HTTP/1.0
REMOTE_ADDR         =   198.198.198.198
DOCUMENT_ROOT       =   /home/gasster
SERVER_ROOT         =   /home/gasster
GATEWAY_INTERFACE   =   CGI/1.1
SCRIPT_NAME         =   /public_html/modemspeed.cgi
SERVER_SOFTWARE     =   Apache/1.1.1
```

```
REMOTE_HOST         =   pm8-6.pacificnet.net
HTTP_USER_AGENT     =   Mozilla/3.0 (Macintosh; I; 68K)
SERVER_NAME         =   www.pacificnet.net
SERVER_PORT         =   80
PATH_INFO           =
PATH_TRANSLATED     =
AUTH_TYPE           =
REMOTE_USER         =
REMOTE_IDENT        =
CONTENT_TYPE        =
CONTENT_LENGTH      =
HTTP_REFERER        =
```

---

modemspeed.dat

```
UserName   = Sam
UserEmail  = gasster@xyz.com
ModemSpeed = ISDN
```

Running CGI Lint

Here is the results of running CGI Lint on my modemspeed.cgi program.

```
gasster> CGI_Lint perllib/post_cgilint.conf perllib/modemspeed.dat

It looks as though your script has no bugs (atleast, on the surface),
so here is the output you have been waiting for:

================================================================================
<html>
<head>
<title>Modem Speed Survey</title>
</head>
<body>
<h1>Modem Speed Survey</h1>
    <H1 ALIGN=CENTER>MODEM SPEED SURVEY</H1>
    <P>
    <HR ALIGN=CENTER NOSHADE SIZE="2">
    <P>
    Thank you Sam for filling out the survey form<P>
    Your email address was given as:  gasster@xyz.com <P>
    Your Modem Speed is:  ISDN
    <P>
    <HR ALIGN=CENTER NOSHADE SIZE="2">
</body>
</html>
================================================================================

gasster>
```

So all looks well. Let's break something in the CGI program and see what happens! Let's change the HTTP Header so that it only prints a single newline character after the `Content-type` header.

Code change in modemspeed.cgi

```
#  redefine PrintHeader to generate an error
#  for CGI Lint

print "Content-type: text/html\n";

#print &PrintHeader;
```

CGI Lint Output

```
gasster> CGI_Lint perllib/post_cgilint.conf perllib/modemspeed.dat

A serious error! Either you are not outputting a **BLANK** line after
the HTTP headers, *OR* you are trying to send invalid (or undefined)
HTTP headers. Please check the output of your script and try again.

gasster>
```

Using CGI Lint can save you time tracking down the problem and won't tie up the HTTP server with your debugging. I strongly suggest you learn how to use it. Its not perfect, but its a step in the right direction.

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 9]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 10: HYPERMEDIA, FORMS AND SECURITY

READING ASSIGNMENT:
CGI Programming on the WWW Ch. 5-7

HOMEWORK ASSIGNMENT:
Assignment No. 10 (Due: Last Class).

TOPICS

- Hypermedia Documents
- Advanced Form Applications
- CGI Security Issues
- Server Side Includes

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 11]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 10: ~~DEBUGGING~~, HYPERMEDIA, FORMS AND SECURITY

READING ASSIGNMENT:
CGI Programming on the WWW Ch. 5-7

HOMEWORK ASSIGNMENT:
Assignment No. 10 (Due: Last Class).

TOPICS

- Hypermedia Documents
- Advanced Form Applications
- CGI Security Issues
- Server Side Includes

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 11]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Introduction to Perl with Applications to CGI Programming

HOMEWORK ASSIGNMENT NO. 10
Guestbook CGI Module

DUE: Dec. 9, Last Class
(No Late Assignments Will Be Accepted!)

---

1. **Guestbook Module - CGI**

   As your final assignment you are to take your guestbook module and turn it into a guestbook CGI module. This should not be too difficult assuming you have completed the previous Guestbook assignments.

   Here are the requirements:

   A. The module name should be `Guestbook` and store it in a file called `Guestbook.pm`.

   B. The Guestbook module should contain, as a minimum, the following functions:

      - Function(s) to parse all of the required user inputs (you can use either cgi-lib.pl, common.pl or CGI.pm, or equivalent).
      - Function(s) to create new guestbook if one does not exist
      - Function(s) to add current user inputs to an existing guestbook
      - Function(s) to send the user a thank you page after completing the processing of their inputs.

      You may want to include more functions which these functions call. Also, feel free to use ANY EXISTING Perl module(s) that you find helpful, but you are not required to do this.

   C. Provide the necessary HTML form interface to your guestbook. This should include a form for users to add to the guestbook and a mechanism for the users to view the guestbook.

      You can implement this as a completely separate file or as part of your guestbook module (thus the module provides a function which is called as a CGI and returns the guestbook HTML form, similar to the method implemented in the textbook). As a minimum, your Guestbook should take as input:

      - User Name
      - User Email
      - User Location (city, state, country - either in single field or separate fields)
      - Comments

   D. Include the following security measures:
      - Assume all users inputs are tainted
      - Escape all HTML the user might enter; i.e., the HTML tags will show up, not what they refer to.
      - Limit size of user inputs; comment section should have a fixed limit on the size. 2000 bytes should be enough, but you can pick your own value.

For extra credit you can also try the following: If the user sends any tainted inputs return them a message stating that you are logging this with the webmaster and no longer allowing anyone from their domain to access your guestbook or guestbook form. Don't parse their inputs to the guestbook, but to a *tainted inputs* log file.

E. Demonstrate that your Guestbook works by either running it on a server or testing it stand alone using CGI Lint.

I want to see, as a minimum, the following items when you turn this assignment in:

- All source code: Perl and HTML
- Printed examples of what the user would see on their browser such as the Guestbook form, thank you page and the guestbook itself.

You should be able to print this from your browser.

[TITLE PAGE | TABLE OF CONTENTS | HOMEWORK | LECTURE 10]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 10: Hypermedia Documents

Introduction

Hypermedia is a term that refers to going beyond the usage of plain text or marked-up text (SGML or HTML) in the documents presented on the WWW. This includes graphics, sound and video [I haven't seen *smell-o-wave* on the WWW yet, but I'm sure someone's working on it!! ;-)].

The CGI definitions provided one of the earliest mechanisms for incorporating Additional data types into documents. CGI also provides a mechanism for creating dynamic documents; WWW pages with content that varies depending on the context, date/time or location (of client).

Chapter 6 CGI Examples

homepage_welcome.pl

Important points:

- Set time zone using TZ environment variable

```
$ENV{'TZ'} = 'PST';
```

- Extract current date and time
- Set greeting based on local time of day (how might you determine the user's time zone?
- Create formatted time string using

```
$time = sprintf ("%02d:%02d:%02d", $hour, $minutes, $seconds);
```

- Open a pipe to the Unix w command (who is logged in and what are they doing. The -h and -s switches supress the header and use the short format.

```
open(CHECK, "/usr/bin/w -h -s $webmaster |");

if (<CHECK> =~ /$webmaster/)
{
   $in_out = "I am currently logged in.";
} else {
   $in_out = "I just stepped out.";
}
```

Both of the next two examples use the HTTP Refresh Header to automatically cause the client to reload the document after a specified number of seconds. I have modified the versions that are provided by the author (slightly different format and fixed a few bugs and added comments).

date.pl

date.cgi: TRY IT!

```
#!/usr/bin/perl

# file: date.cgi

$delay = 5;  #refresh delay in seconds

# Get the current date using the
# Unix shell command

$date = `/bin/date`;

# print the HTTP header and date

  print <<End_of_Output;
Refresh:  $delay
Content-type: text/html


<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML><HEAD><TITLE>The Current Date</TITLE>
</HEAD>
<BODY>
<H2 ALIGN=CENTER>The Current Date</H2>
<P><HR ALIGN=CENTER NOSHADE SIZE="2"><P>
The current date/time are: $date
<P><HR ALIGN=CENTER NOSHADE SIZE="2"><P>
</BODY>
</HTML>
End_of_Output

exit(0);
```

---

fortune.pl

The CGI will print a fortune cookie very `$refresh_time` seconds.

fortune.cgi: TRY IT!

```
#!/usr/local/bin/perl

# set the refresh rate in seconds
$refresh_time = 10;

# grab a fortune cookie from the
# Unix shell fortune command

$fortune = `/usr/local/bin/fortune`;

# print the HTTP header and fortune
#
print <<End_of_Output;
Refresh: $refresh_time
Content-type: text/html

<HTML><HEAD><TITLE>YOUR FORTUNE</TITLE></HEAD>
```

```
<BODY>
Here is another fortune...
$fortune

</BODY></HTML>
End_of_Output
exit(0);
```

The next three examples show how one can create graphic object on-the-fly and include them in your documents.

Postscript Examples

The following Perl CGI script examples create graphics using the *Postscript* language, then use the *Ghostscript* program to convert the *Postscript* to *GIF* format (a bit of a kludge!). I had a problem getting this to work as the *gif8* driver is nolonger distributed with the *Ghostscript* program.

- ps_time.pl
- ps_clock.pl
- ps_load.pl

Important Points

- Buffering turned off using `$|=1;`
- Use of external program to convert *Postscript* to *GIF* format. These examples used the Ghostscript program, but any application that allows you to send input, using a pipe in Perl, to the application via `STDIN` and receive output via `STDOUT` would work. We also don't have to convert just to *GIF* , a program that would convert to *JPEG* (which Netscape can handle) would also have worked.
- HTTP Header set to graphic type.

GD Examples

The following examples all use the **gd graphics library** developed by Thomas Boutell. The library is written is C and is compilied into a stand-alone library on your system (hopefully your systems administrator has already done this for you). The library is callable from other C programs and fortunately for us, from Perl. To access the gd library from Perl you use the `GD.pm` Perl module written by Lincoln Stein.

- gd_clock.pl
- gd_counter.pl
- gd_load.pl
- gd_time.pl
- gd_clock.html (This is my example)

Important Points

- GD.pm requires at least Perl Version 5.002 (5.003 is best).

- Graphic counter example uses file lock function `flock()`
  The counter program keeps a running count of the number of visitors (human and otherwise) to the web page in a file. The file, `count.txt`, is locked while the program is accessing it for reading and up-dating. This prevents multiple processes from access the same file and causing problems. This is in general a useful technique when you want to access a file for up-dating via a CGI program and there is a chance that the server might get simultaneous requests for the script that access that file.

- We can call a CGI script that creates an image with the HTML `IMG` tag. (see below).

Example

gd_clock.html: TRY IT!

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML>
<HEAD>
<TITLE>GD Clock Example</TITLE>
</HEAD>
<BODY>
<H2 ALIGN=CENTER>GD Clock Example</H2>
<P>
This clock was created with the <B>gd_clock.cgi</B> program.
</P>
<CENTER>
<IMG SRC="http://www.pacificnet.net/~gasster/gd_clock.cgi" ALIGN=MIDDLE>
</CENTER>
</BODY>
</HTML>
```

Graphing and Plotting

In many applications one would like to produce sophisticated graphs and plots. One example might be a graph of the web site statistics compiled from the access log file.

- graph_gnuplot.pl [TRY IT]

  This program uses the gnuplot program to generate a graphic plot of the web server access. The default file format generated by gnuplot is ppm. The CGI script uses the utility ppmtogif to convert the gnuplot file to a gif file. The gif file is then displayed by the server.

- graph_pgperl.pl

  Similar in overall function to graph_gnuplot.pl, but this program uses the pgperl package. This is a Perl derivative that uses the Fortran plotting library developed for astronomical applications. Check it out.

Important Points

- Note the use of temporary files, to store the graphics, in the `/tmp` directory. The file names are generated using the process ID, which will result in a unique file name.

- The accesses to the server are binned into an array (`@time`) with 24 bins. The access_log file is opened read-only, and the following regular expression is used to extract the hour field for each server access:

```
while (<FILE>) {
```

```
if (m|\[\d+/\w+/\d+:([^:]+)|) {
    $time[$1]++;
}
```

which matches:

```
\[\d+/  ---> a [ followed by one or more digits, followed by a /
\w+/    ---> one or more alphanumeric (word) characters followed by a /
\d+:    ---> one or more digits followed by a colon
([^:]+) ---> one or more characters, not a colon, so it matches everything
             up-to the next colon
```

This works since the portion of the access log file that contains the date/time information is of the form:

```
[DD/MMM/YYYY:hh:mm:ss -xxxx]
```

- Note the use of `unlink` to delete the temporary plot files once the script is finished with them. Its bad form to leave your clutter in `/tmp`, since it is really not intended to be a scratch space.

Simple Animation

With the advent of `Java` and such programs as `Shockwave` the simple type of animation offered by CGI programming may seem a little *behind-the-times*.

CLIENT PULL

This technique uses the fact that we can tell the client (browser) to refresh (re-request a document) at specified intervals. In this case the client requests the data from the server on a period basis; the client is *pulling* the data from the server. See examples in the book.

SERVER PUSH

One implementation of the server push technique uses the MIME type `Content-Type: multipart/x-mixed-replace`. This tells the client that the server is going to be sending multiple data elements, each one replacing the previous one. Part of the specification requires providing a boundary separator for the data packets. See example below.

SERVER PUSH EXAMPLES

`nph-server_push.cgi`
TRY IT

`View_animation.html`
TRY IT

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 10]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

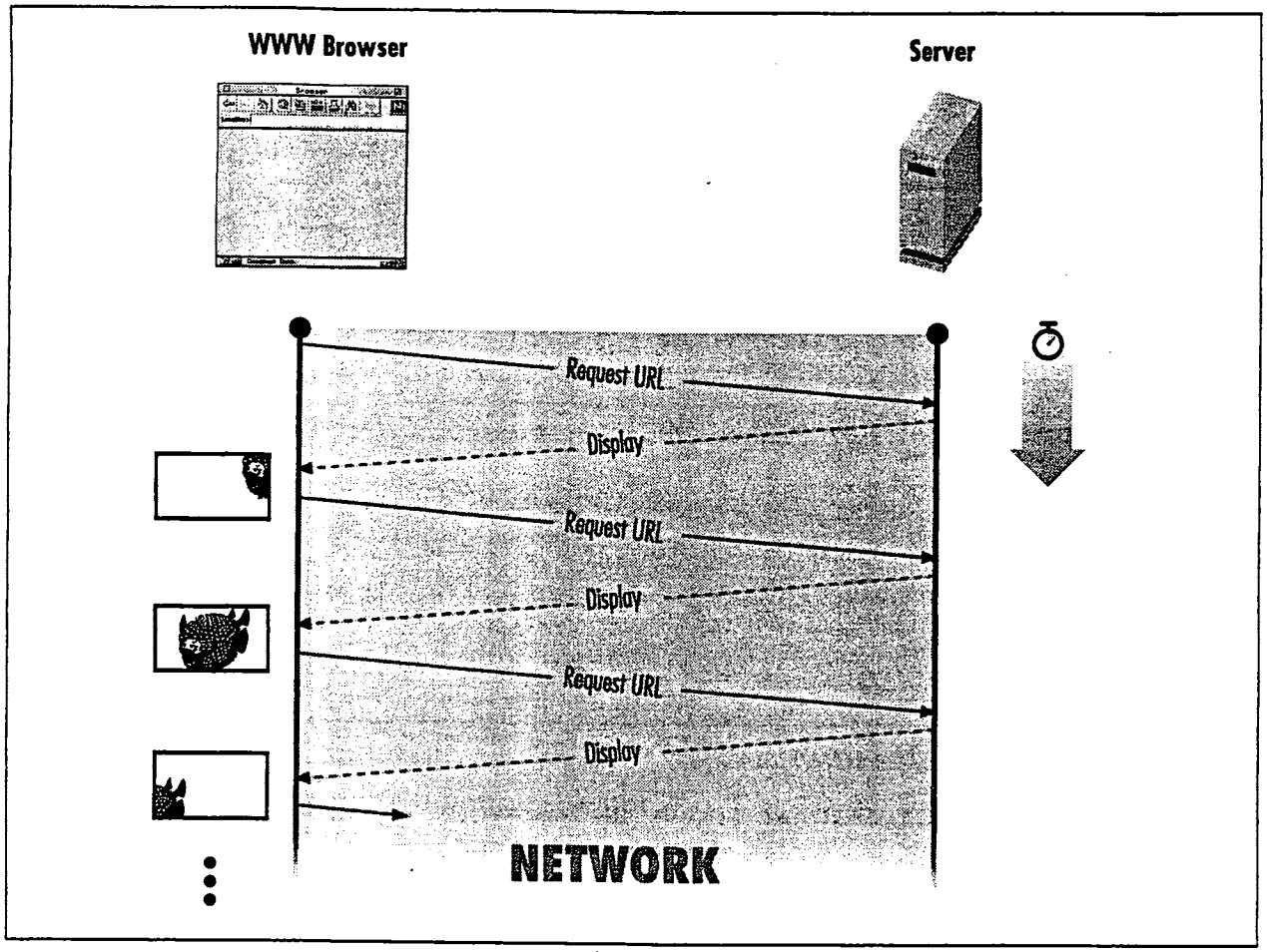


Figure 6-7. Animation using client pull

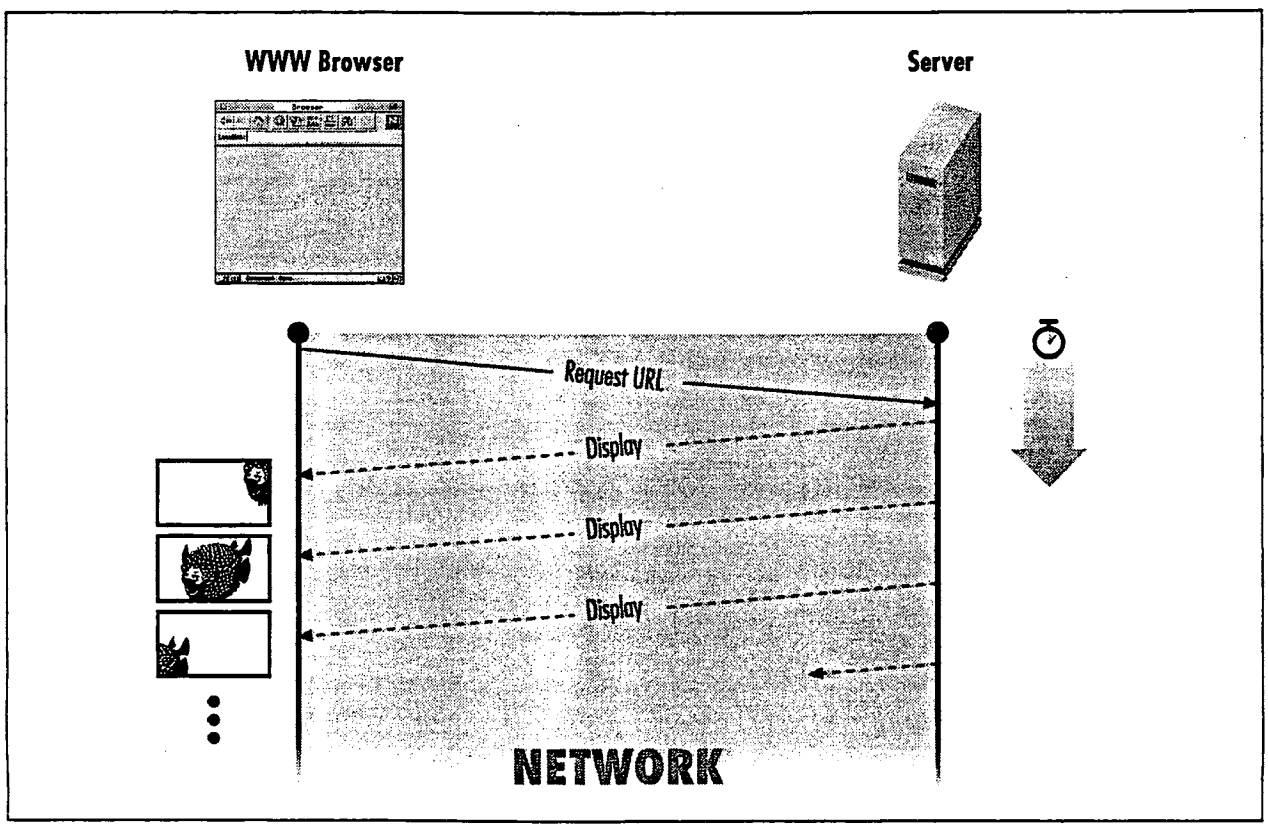


Figure 6-8. Animation using server push

Listing of gd_clock.pl

```
#!/usr/local/bin/perl

# load the local perl5 library
BEGIN { unshift( @INC, "/usr/local/lib/perl5"); }

use GD;   # load the perl module GD.pm

# turn off buffering
$| = 1;

# print the HTTP header
print "Content-type: image/gif", "\n\n";

# define some constants
# define clock face geometry and origin
# of the center
$max_length = 150;
$center = $radius = $max_length / 2;
@origin = ($center, $center);

# size of ticks, hour and minute hands
$marker = 5;
$hour_segment = $radius * 0.50;
$minute_segment = $radius * 0.80;

# conversion from degress to radians
$deg_to_rad = (atan2 (1, 1) * 4) / 180;

# create a new image object
$image = new GD::Image ($max_length, $max_length);

# define colors we want to use
$black = $image->colorAllocate (0, 0, 0);     # original background
$red = $image->colorAllocate (255, 0, 0);     # ticks and center
$green = $image->colorAllocate (0, 255, 0);   # hands
$blue = $image->colorAllocate (0, 0, 255);    # clock "face" circle

#
#  get current time
($seconds, $minutes, $hour) = localtime (time);

# convert time to angular position on clock face
$hour_angle = ($hour + ($minutes / 60) - 3) * 30 * $deg_to_rad;
$minute_angle = ($minutes + ($seconds / 60) - 15) * 6 * $deg_to_rad;

# draw circular clock face
$image->arc (@origin, $max_length, $max_length, 0, 360, $blue);


# make clock face tick marks
for ($loop=0; $loop < 360; $loop = $loop + 30) {
        local ($degrees) = $loop * $deg_to_rad;

        $image->line (  $origin[0] + (($radius - $marker) * cos ($degrees)),
                        $origin[1] + (($radius - $marker) * sin ($degrees)),
                        $origin[0] + ($radius * cos ($degrees)),
                        $origin[1] + ($radius * sin ($degrees)),
```

```
                        $red  );
}


# draw a line for the "BIG HAND"
$image->line (  @origin,
                $origin[0] + ($hour_segment * cos ($hour_angle)),
                $origin[1] + ($hour_segment * sin ($hour_angle)),
                $green  );

# draw a line for the "LITTLE HAND"
$image->line (  @origin,
                $origin[0] + ($minute_segment * cos ($minute_angle)),
                $origin[1] + ($minute_segment * sin ($minute_angle)),
                $green  );

# draw a red filled circle at the clock face center
$image->arc (@origin, 6, 6, 0, 360, $red);
$image->fill ($origin[0] + 1, $origin[1] + 1, $red);

#
#  make the gif transparent
#
$image->transparent($black);

print $image->gif;

exit(0);
```

Lecture 10: Advanced Form Applications

Introduction

Chapter 7 CGI Examples

- **Guestbooks**

  The WWW is full of web sites with guestbooks! There are many ways to implement this type of application. Gundavaram shows one example where as an *all-in-one* application. I don't like this format; I'd rather see separate applications for adding/up-dating the guestbook and for viewing the guestbook. This is just a matter of personel preference, as neither method is *better* than the other.

  Since you have all been working on your own guestbook applications I won't discuss this one in detail. Just highlight some of the interesting aspects.

- **Survey Forms**

  This is classic CGI example. The user fills out a standard form which is used to create a database of user responses. The data collected at any given time can be analyzed and the results provided in a summary format (statistical analysis); either tabular or graphical. This example shows how to provide a graphical summary of the survey results. We'll discuss this example in detail.

- **Quiz/Test**
  This is a very clever example since it uses a Perl script to handle a set of *extended* HTML tags, that the author has defined, to create the quiz. In addition to presenting and collecting the quiz input, the script can also grade the quiz.

FAVORITE ICE CREAM FLAVOR SURVEY

ice_cream.html
HTML form to provide a user interface to collect the survey data.

ice_cream.pl
Perl CGI program to parse the user's input to the survey and record their choice to the database.

ice_cream.dat
The survey database file.

pie.pl
Perl CGI program to read the survey database file and return the survey results to-date. The results are summarized either in tabular format or graphical format, depending on the client's browser type.

Important Points

- The main survery form (`ice_cream.html`) uses the `POST` method with the CGI script `ice_cream.pl`. This CGI script parses the user's favorite ice cream flavor selection; opens the `ice_cream.dat` survey database, reads the data, closes the file, up-dates the data, then writes it to the databases file. Finally a `Thank You` page is sent to the user, assuming there were no errors.

- The results of the survey are viewed using the CGI program `pie.pl`. This CGI script is call with the data file specified as part of the extra path variable (`PATH_INFO` environment variable).

```
http://www.xyz.com/~username/pie.cgi/ice_cream.dat
```

- The data file is then read using the subroutine `read_data_file`. The full path to the data file obtained from:

```
....
    if ($ENV{'PATH_INFO'}) {
        $poll_file = $document_root . $ENV{'PATH_INFO'};
....
```

This subroutine returns (using aliases) the count for each flavor; the name for each flavor; and the color corresponding to each flavor in the pie chart.

- The client browser type, obtained in the environment variable `HTTP_USER_AGENT` is used to determine whether to send the user a graphic or text version of the survey results.

- The subroutine `draw_pie` generates the pie chart and sends it to the user. This routine uses the Perl GD module. The text version is generated and returned using the subroutine `text_results`.

View Survey Results

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 10]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML>
<HEAD><TITLE>Ice Cream Survey</TITLE></HEAD>

<BODY TEXT="#000000" LINK="#0000FF" VLINK="#FF0000" ALINK="#AA1177" BGCOLOR="#E4BB94">

<H2 ALIGN=CENTER>Favorite Ice Cream Survey</H2>
<P><HR ALIGN=CENTER NOSHADE SIZE="2"><P>

<FONT SIZE="+1">
<FORM ACTION="http://www.pacificnet.net/~gasster/ice_cream.cgi" METHOD="POST">
What is your favorite flavor of ice cream?
<P>
<INPUT TYPE="radio" NAME="ice_cream" VALUE="Vanilla" CHECKED>Vanilla<BR>
<INPUT TYPE="radio" NAME="ice_cream" VALUE="Strawberry">Strawberry<BR>
<INPUT TYPE="radio" NAME="ice_cream" VALUE="Chocolate">Chocolate<BR>
<INPUT TYPE="radio" NAME="ice_cream" VALUE="Other">Other<BR>
<P>
<INPUT TYPE="submit" VALUE="Submit the survey">
<INPUT TYPE="reset"  VALUE="Clear your choice">
</FORM>
<P><HR ALIGN=CENTER NOSHADE SIZE="2"><P>
View the <A HREF="http://www.pacificnet.net/~gasster/pie.cgi/ice_cream.dat">current results </A>.

</FONT>
</BODY>
</HTML>
```

Vanilla::Strawberry::Chocolate::Other
13::21::43::30
red::yellow::blue::green

```
#!/usr/local/bin/perl5.001m

BEGIN { unshift( @INC, "/home/gasster/perllib"); }

require "common.pl";

$webmaster = "Sam\@muon\.com";
$document_root = "/home/gasster/public_html";
$ice_cream_file = "/ice_cream.dat";
$full_path = $document_root . $ice_cream_file;

$exclusive_lock = 2;
$unlock = 8;

&parse_form_data(*poll);
$user_selection = $poll{'ice_cream'};

if ( open (POLL, "<" . $full_path) ) {
    flock (POLL, $exclusive_lock);

    for ($loop=0; $loop < 3; $loop++) {
        $line[$loop] = <POLL>;
        $line[$loop] =~ s/\n$//;
    }

        @options = split ("::", $line[0]);
        @data    = split ("::", $line[1]);
        $colors  = $line[2];

        flock (POLL, $unlock);
        close (POLL);

        $item_no = 3;
        for ($loop=0; $loop <= $#options; $loop++) {
            if ($options[$loop] eq $user_selection) {
                $item_no = $loop;
                last;
            }
        }

        $data[$item_no]++;


        if ( open (POLL, ">" . $full_path) ) {
        flock (POLL, $exclusive_lock);

        print POLL join ("::", @options), "\n";
        print POLL join ("::", @data), "\n";
        print POLL $colors, "\n";

        flock (POLL, $unlock);
        close (POLL);

        print "Content-type: text/html", "\n\n";

        print <<End_of_Thanks;
<HTML>
<HEAD><TITLE>Thank You!</TITLE></HEAD>
<BODY>
<H1>Thank You!</H1>
<HR>
Thanks for participating in the Ice Cream survey. If you would like to see the
```



```
current results, click
<A HREF=" http://www.pacificnet.net/~gasster/pie.cgi${ice_cream_file}">here</A>.
</BODY></HTML>

End_of_Thanks

        } else {
            &return_error (500, "Ice Cream Poll File Error",
              "Cannot write to the poll data file [$full_path].");
        }

} else {
        &return_error (500, "Ice Cream Poll File Error",
            "Cannot read from the poll data file [$full_path].");
}

exit (0);
```

```
#!/usr/local/bin/perl5.001m

BEGIN { unshift( @INC, "/usr/local/lib/perl5"); }

use GD;

$webmaster = "Sam\@muon\.com";
$document_root = "/home/gasster/public_html";


&read_data_file (*slices, *slices_color, *slices_message);


$no_slices = &remove_empty_slices();


if ($no_slices == -1) {
    &no_data ();
} else {
    $nongraphic_browsers = 'Lynx|CERN-LineMode';
    $client_browser = $ENV{'HTTP_USER_AGENT'};

    if ($client_browser =~ /$nongraphic_browsers/) {
        &text_results ();
    } else {
        &draw_pie ();
    }
}

exit(0);

#---------- subroutine definitions ----------

sub no_data
{
    print "Content-type: text/html", "\n\n";

    print <<End_of_Message;
<HTML>
<HEAD><TITLE>Results</TITLE></HEAD>
<BODY>
<H1>No Results Available</H1>
<HR>
Sorry, no one has participated in this survey up to this point.
As a result, there is no data avilable. Try back later.
<HR>
</BODY></HTML>
End_of_Message
}

sub draw_pie
{
    local ( $legend_rect_size, $legend_rect, $max_length, $max_height,
        $pie_indent, $pie_length, $pie_height, $radius, @origin,
        $legend_indent, $legend_rect_to_text, $deg_to_rad, $image,
        $white, $black, $red, $yellow, $green, $blue, $orange,
        $percent, $loop, $degrees, $x, $y, $legend_x, $legend_y,
        $legend_rect_y, $text, $message);

    $legend_rect_size = 10;
    $legend_rect = $legend_rect_size * 2;
```



```
    $max_length = 450;

    if ($no_slices > 8) {
        $max_height = 200 + ( ($no_slices - 8) * $legend_rect );
    } else {
        $max_height = 200;
    }

    $pie_indent = 10;
    $pie_length = $pie_height = 200;
    $radius = $pie_height / 2;

    @origin = ($radius + $pie_indent, $max_height / 2);
    $legend_indent = $pie_length + 40;
    $legend_rect_to_text = 25;
    $deg_to_rad = (atan2 (1, 1) * 4) / 180;

    $image = new GD::Image ($max_length, $max_height);

    $white = $image->colorAllocate (255, 255, 255);
    $black = $image->colorAllocate(0, 0, 0);
    $red = $image->colorAllocate (255, 0, 0);
    $yellow = $image->colorAllocate (255, 255, 0);
    $green = $image->colorAllocate(0, 255, 0);
    $blue = $image->colorAllocate(0, 0, 255);
    $orange = $image->colorAllocate(255, 165, 0);

    grep ($_ = eval("\$$_"), @slices_color);

    $image->arc (@origin, $pie_length, $pie_height, 0, 360, $black);

    $percent = 0;
    for ($loop=0; $loop <= $no_slices; $loop++) {
        $percent += $slices[$loop];
        $degrees = int ($percent * 360) * $deg_to_rad;

        $image->line (  $origin[0],
                        $origin[1],
                $origin[0] + ($radius * cos ($degrees)),
                        $origin[1] + ($radius * sin ($degrees)),
                        $slices_color[$loop] );
    }

    $percent = 0;
    for ($loop=0; $loop <= $no_slices; $loop++) {
        $percent += $slices[$loop];
        $degrees = int (($percent * 360) - 1) * $deg_to_rad;

        $x = $origin[0] + ( ($radius - 10) * cos ($degrees) );
        $y = $origin[1] + ( ($radius - 10) * sin ($degrees) );

        $image->fill ($x, $y, $slices_color[$loop]);
    }

    $legend_x = $legend_indent;
    $legend_y = ( $max_height - ($no_slices * $legend_rect) -
            ($legend_rect * 0.75) ) / 2;

    for ($loop=0; $loop <= $no_slices; $loop++) {
        $legend_rect_y = $legend_y + ($loop * $legend_rect);
        $text = pack ("A18", $slices_message[$loop]);
```

```
        $message = sprintf ("%s (%4.2f%%)", $text, $slices[$loop] * 100);

        $image->filledRectangle (   $legend_x,
                                    $legend_rect_y,
                                $legend_x + $legend_rect_size,
                            $legend_rect_y + $legend_rect_size,
                                    $slices_color[$loop] );

        $image->string ( gdSmallFont,
                     $legend_x + $legend_rect_to_text,
                         $legend_rect_y,
                     $message,
                 $black );
    }

    $image->transparent($white);

    $| = 1;
    print "Content-type: image/gif", "\n\n";
    print $image->gif;
}

sub text_results
{
    local ($text, $message, $loop);

    print "Content-type: text/html", "\n\n";

    print <<End_of_Results;
<HTML>
<HEAD><TITLE>Results</TITLE></HEAD>
<BODY>
<H1>Results</H1>
<HR>
<PRE>
End_of_Results

    for ($loop=0; $loop <= $no_slices; $loop++) {
        $text = pack ("A18", $slices_message[$loop]);
        $message = sprintf ("%s (%4.2f%%)", $text,
                            $slices[$loop] * 100);

        print $message, "\n";
    }

    print "</PRE><HR>", "\n";
    print "</BODY></HTML>", "\n";
}

sub read_data_file
{
    local (*slices, *slices_color, *slices_message) = @_;
    local (@line, $total_votes, $poll_file, $loop, $exclusive_lock, $unlock);

    $exclusive_lock = 2;
    $unlock = 8;

    if ($ENV{'PATH_INFO'}) {
        $poll_file = $document_root . $ENV{'PATH_INFO'};
    } else {
        &return_error (500, "Poll Data File Error",
                "A poll data file has to be specified.");
    }
```



```
    if ( open (POLL, "<" . $poll_file) ) {
        flock (POLL, $exclusive_lock);

    for ($loop=0; $loop < 3; $loop++) {
                $line[$loop] = <POLL>;
                $line[$loop] =~ s/\n$//;
        }

        @slices_message = split ("::", $line[0]);
        @slices         = split ("::", $line[1]);
        @slices_color   = split ("::", $line[2]);

        flock (POLL, $unlock);
        close (POLL);

        $total_votes = 0;
        for ($loop=0; $loop <= $#slices; $loop++) {
            $total_votes += $slices[$loop];
        }

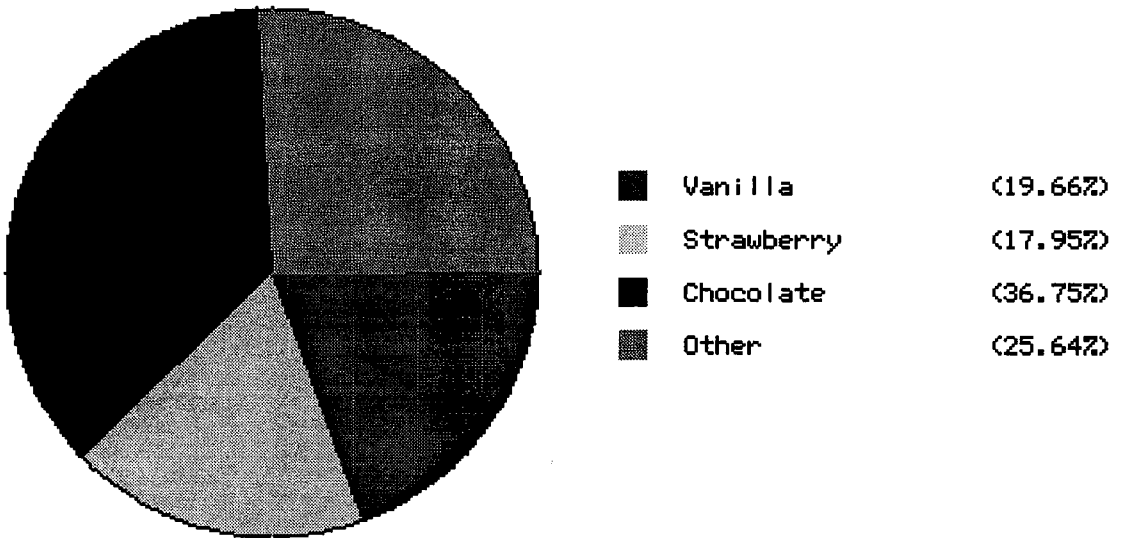
    if ($total_votes > 0) {
            grep ($_ = ($_ / $total_votes), @slices);
    }

    } else {
        &return_error (500, "Poll Data File Error",
                    "Cannot read from the poll data file [$poll_file].");
    }
}

sub remove_empty_slices
{
    local ($loop) = 0;

    while (defined ($slices[$loop])) {
        if ($slices[$loop] <= 0.0) {
            splice(@slices, $loop, 1);
            splice(@slices_color, $loop, 1);
            splice(@slices_message, $loop, 1);
        } else {
            $loop++;
        }
    }

    return ($#slices);
}
```


Vanilla (19.66%)
Strawberry (17.95%)
Chocolate (36.75%)
Other (25.64%)

Lecture 10: CGI Security Issues

Acknowledgement
The material in this lecture draws heavily on the work of several people. They have been kind enough to share this information on the WWW and I am passing it on you. They are: Paul Phillips, Lincoln Stein, and Jamie McCarthy.

Why should you be concerned about security?

Here is a very simple example of the type of problem that can occur with sloppy and in-secure CGI programming.

- Try the Example
- View the HTML file:bad_w.html
- View the CGI Script: showwho.cgi

What else might someone try?

- Mailing the password file to the attacker (unless shadowed)
- Mailing a map of the filesystem to the attacker
- Mailing system information from /etc to the attacker
- Starting a login server on a high port and telneting in
- Many denial of service attacks: massive filesytem finds, for example, or other resource consuming commands
- Erasing and/or altering the server's log files or any files the attacker can get to.
- Dumping a very large file into a textarea box. If your file system is close to full (which unfortunately is too often the case) this might crash the system, or at least create some very annoying problems.
- Also note that some versions of the HTTPd servers also have security holes. Check the on-line security FAQs and make sure you or your IAP are not using one of these!

What should I do?

1. Read the various security FAQ's and related information available on the WWW. Several books on CGI programming discuss these issues, although, in my opinion, they don't provide enough detail.
2. Understand what functions and commands are potentially dangerous in the programming language

you are using (Perl in our case).

3. **Always assume any user input data is tainted.** Even if you don't think there is a way for someone to enter a tainted command, assume they will/can and act accordingly.

4. Always use absolute paths, don't use relative paths, and don't include the currently directory (.) in your path.

5. Use Perl taint checking and untaint all user input. See below.

6. Don't let users enter arbitrary large amounts of data into your fields. Limit the size of text fields, check size of user input and if necessary truncate user input.

7. Use Perl's regular expressions and substitution operator to untaint and in general clean up your code. Here are two examples taken from a guestbook Written by Jamie McCarthy (jamie@nizkor.almanac.bc.ca):

```
##########################################################################
# stripMultiline
#
# Given multiline input (i.e. the comment textarea), make it all
# nice and neat:  no unwanted tags, no messiness, and automatically
# add <p> tags where it's double-spaced.
##########################################################################

sub stripMultiline
{
   local ($_) = @_;

      # limit incoming text to 16K -- no comment should be that long!
   if (length($_) > 16384) {
      $_ = substr($_, 0, 16384);
   }

                              # convert non-printable chars to " "
   tr#\000-\011\013-\014\015-\037# #;
   s#\r\n#\n#gm;              # some browsers send CRLF to separate lines;
   s#\n\r#\n#gm;              # convert that to a single newline

   if (!m#[<>]#) {            # if no HTML code,
      s#\n{2,}\s*#<p>#gm;     # convert blank lines to <p>
      s#\n\s+#<p>#gm;         # convert indented lines to <p>
   }
   s#<>##gm;                  # remove any empty tags: "<>"
   s# +$##gm;                 # remove any trailing whitespace
                              # strip the <> from <URL:xxx://xyz>
   s#<((URL.{1,3})?(http|ftp|gopher|telnet)://[^>\r\n]+)>#$1#gm;
   s#<[^ap/][^>]*>##gmi;      # remove all non-/ tags except <a...> and <p...>
   s#</[^a][^>]*>##gmi;       # remove all / tags except </a>
   s#<p[^>]*>#<p>#gmi;        # convert all <p xxx> tags into plain <p>
   s#\s+# #gm;                # convert all whitespace sequences to " "
   s#<p>\s*(?=<p>)##gmi;      # remove all empty paragraphs
   s#\s*<p>#\n\n<p>#gmi;      # put a double-space in front of <p> tags
   s#\A\n\n<p>#<p>#gm;        # except the first one
   s#<p>\s+#<p>#gm;           # remove paragraphs' initial whitespace
   if (!m#\A<p>#m) {          # if no initial <p>,
      $_ = "<p>" . $_;        # add one
   }
```

```
                                # make URLs clickable if they aren't already
    s#([^>"]|\A)
     ((http|ftp|gopher|telnet)://[\-\w/.:+~\#%?]+)
     ([^<"]|\Z)
     #
     $1<a href="$2">$2</a>$4
     #gmx;

    return $_;
}

########################################################################
# stripSingleline
#
# Given single-line input (i.e. the name), just do some basic
########################################################################

sub stripSingleline
{
    local ($_) = @_;

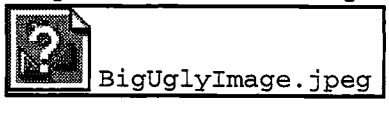
    tr#\000-\037# #s;       # convert non-printable chars to " "
    s#\s+# #gm;             # convert text to a single line
    s#^\s+# #g;             # with no leading whitespace
    s#\s+$# #g;             # and no trailing whitespace
    tr#<>##d;               # remove any < or > characters

    return $_;
}
```

EXAMPLE Input/Output from stripMultiline()

```
---->INPUT HTML
<MENU>
<LI>Use Perl's taint checking mode (<CODE>-T</CODE>) <P>
<LI>WWW Security FAQ: <A NAME="WWWFAQ" HREF="./wwwsf5.html">Safe Scripting in Perl
<LI>http://www.abc.com/bigfile.html
<LI><IMG SRC="BigUglyImage.jpeg" ALT="BigUglyImage.jpeg" ALIGN=MIDDLE>
</MENU>

---->What it would look like
```

- Use Perl's taint checking mode (-T)
- WWW Security FAQ: Safe Scripting in Perl
- http://www.abc.com/bigfile.html
-  BigUglyImage.jpeg

```
---->OUTPUT - cleaned up version
 Use Perl's taint checking mode (-T)

WWW Security FAQ: Safe Scripting in Perl
```

http://www.abc.com/bigfile.html

---

8. See the reference material for other examples.

CGI Security with Perl

- Use Perl's taint checking mode (`-T`)
- WWW Security FAQ: Safe Scripting in Perl

On-line References Regarding CGI Security

There are several sites on the WWW that discuss CGI security issues. Here are some good places to start. They also point you to other sources of information.

1. *The World Wide Web Security FAQ* , by Lincoln D. Stein, <lstein@genome.wi.mit.edu>

   This is a very comprehensive discussion of CGI security issues.
   For the latest version check the above URLs.

   - http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.txt (text only)
   - http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html (html)

2. Paul Phillips CGI Security Page:
   Paul's page provides links to several other cgi-security pages, plus Paul's own overview of CGI security. Good place to start.

   - http://www.cerf.net/~paulp/cgi-security/

3. The Perl Home Page also discusses and keeps track of security issues.

   - http://www.perl.com/perl

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 10]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

```
#!/usr/bin/perl

#
#        ---> WARNING <----
# ---> THIS IS VERY DANGEROUS <----
#
#  NOTE THAT IS SCRIPT IS AN EXAMPLE
#  OF WHAT NOT TO DO. DO NOT USE
#  THIS SCRIPT FOR ANY OPERATION CGI
#
#  CONSIDER YOURSELF WARNED!!!
#
# ---> THIS IS VERY DANGEROUS <----
#        ---> WARNING <----
#


BEGIN{
unshift(@INC, "/home/gasster/perllib")
}

require "common.pl";

&parse_form_data(*simple);
$user = $simple{'user'};

#
# ---> THIS IS VERY DANGEROUS <----
#
      $answer = `/usr/bin/w $user`;
#
# ---> THIS IS VERY DANGEROUS <----
#

$answer =~ s/\n/<BR>/gm;

print "Content-type: text/html", "\n\n";

print <<EOO;
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML>
<HEAD>
<TITLE>CGI Security Example: Bad w Results</TITLE>
</HEAD>

<BODY TEXT="#000000" LINK="#0000FF" VLINK="#FF0000" ALINK="#AA1177" BGCOLOR="#E4BB94">

<H2 ALIGN=CENTER>Is This What You Expected?</H2>
<P><HR ALIGN=CENTER NOSHADE SIZE="2"><P>
$answer

<P><HR ALIGN=CENTER NOSHADE SIZE="2"><P>
</BODY>
</HTML>
EOO

exit (0);
```

The World Wide Web Security FAQ

Safe Scripting in Perl

Q43: How do I avoid passing user variables through a shell when calling exec() and system()?

In Perl, you can invoke external programs in many different ways. You can capture the output of an external program using backticks:

```
$date = `/bin/date`;
```

You can open up a pipe to a program:

```
open (SORT, " | /usr/bin/sort | /usr/bin/uniq");
```

You can invoke an external program and wait for it to return with system():

```
system "/usr/bin/sort < foo.in";
```

or you can invoke an external program and *never* return with exec():

```
exec "/usr/bin/sort < foo.in";
```

All of these constructions can be risky if they involve user input that may contain shell metacharacters. For system() and exec(), there's a somewhat obscure syntactical feature that allows you to call external programs directly rather than going through a shell. If you pass the arguments to the external program, not in one long string, but as separate members in a list, then Perl will not go through the shell and shell metacharacters will have no unwanted side effects. For example:

```
system "/usr/bin/sort","foo.in";
```

You can take advantage of this feature to open up a pipe without going through a shell. By calling open on the magic character sequence |-, you fork a copy of Perl and open a pipe to the copy. The child copy then immediately exec's another program using the argument list variant of exec().

```
open (SORT,"|-") || exec "/usr/bin/sort",$uservariable;
while $line (@lines) {
  print SORT $line,"\n";
}
close SORT;
```

To read from a pipe without opening up a shell, you can do something similar with the sequence -|:

```
open(GREP,"-|") || exec "/usr/bin/grep",$userpattern,$filename;
while (<GREP>) {
  print "match: $_";
}
close GREP;
```

These are the form of open() you should use whenever you would otherwise perform a piped open to a command.

An even more obscure feature allows you to call an external program and lie to it about its name. This is useful for calling programs that behave differently depending on the name by which they were invoked.

The syntax is

```
system $real_name "fake_name","argument1","argument2"
```

For example:

```
$shell = "/bin/sh"

system $shell "-sh","-norc"
```

This invokes the shell using the name "-sh", forcing it to behave interactively. Note that the real name of the program must be stored in a variable, and that there's no comma between the variable holding the real name and the start of the argument list.

There's also a more compact syntax for this construction:

```
system { "/bin/sh" } "-sh","-norc"
```

Q44: What are Perl taint checks? How do I turn them on?

As we've seen, one of the most frequent security problems in CGI scripts is inadvertently passing unchecked user variables to the shell. Perl provides a "taint" checking mechanism that prevents you from doing this. Any variable that is set using data from outside the program (including data from the environment, from standard input, and from the command line) is considered tainted and cannot be used to affect anything else outside your program. The taint can spread. If you use a tainted variable to set the value of another variable, the second variable also becomes tainted. Tainted variables cannot be used in eval(), system(), exec() or piped open() calls. If you try to do so, Perl exits with a warning message. Perl will also exit if you attempt to call an external program without explicitly setting the PATH environment variable.

You turn on taint checks in version 4 of Perl by using a special version of the interpreter named "taintperl":

```
#!/usr/local/bin/taintperl
```

In version 5 of perl, pass the -T flag to the interpreter:

```
#!/usr/local/bin/perl -T
```

See below for how to "untaint" a variable.

Q45: OK, I turned on taint checks like you said. Now my script dies with the message: "Insecure $ENV{PATH} at line XX" every time I try to run it!

Even if you don't rely on the path when you invoke an external program, there's a chance that the invoked program might. Therefore you need to include the following line towards the top of your script whenever you use taint checks:

```
$ENV{'PATH'} = '/bin:/usr/bin:/usr/local/bin';
```

Adjust this as necessary for the list of directories you want searched. It's *not* a good idea to include the current directory (".") in the path.

Q46: How do I "untaint" a variable?

Once a variable is tainted, Perl won't allow you to use it in a system(), exec(), piped open, eval(), backtick command, or any function that affects something outside the program (such as unlink). You can't use it even if you scan it for shell metacharacters or use the tr/// or s/// commands to remove metacharacters. The only way to untaint a tainted variable is by performing a pattern matching operation on it and extracting the matched substrings. For example, if you expect a variable to contain an e-mail address, you can extract an untainted copy of the address in this way:

```
$mail_address=~/([\w-.]+\@[\w-.]+)/;
$untainted_address = $1;
```

Q47: I'm removing shell metacharacters from the variable, but Perl still things it's tainted!

See the answer to the question above. The only way to untaint a variable is to extract substrings using a pattern matching operation.

Q48: Is it true that the pattern matching operation `$foo=~/$user_variable/` is unsafe?

A frequent task for Perl CGI scripts is to take a list of keywords provided by the remote user and to use them in a patttern matching operation to fetch a list of matching file names (or something similar). This, in and of itself, isn't dangerous. What is dangerous is an optimization that many Perl programmers use to speed up the pattern matching operation. When you use a variable inside a pattern matching operation, the pattern is recompiled every time the operation is invoked. In order to avoid this expensive recompilation, you can provide the "o" flag to the pattern matching operation to tell Perl to compile the expression once:

```
foreach (@files) {
```

```
    m/$user_pattern/o;

}
```

Now, however, Perl will ignore any changes you make to the user variable, making this sort of loop fail:

```
foreach $user_pattern (@user_patterns) {
   foreach (@files) {
      print if m/$user_pattern/o;
   }
}
```

To get around this problem Perl programmers often use this sort of trick:

```
foreach $user_pattern (@user_patterns) {
   eval "foreach (\@files) { print if m/$user_pattern/o; }";
}
```

The problem here is that the eval() statement involves a user-supplied variable. Unless this variable is checked carefully, the eval() statement can be tricked into executing arbitrary Perl code. (For example of what can happen, consider what the eval statement does if the user passes in this pattern: `"/; system 'rm *'; /"`

The taint checks described above will catch this potential problem. Your alternatives include using the unoptimized form of the pattern matching operation, or carefully untainting user-supplied patterns. In Perl5, a useful trick is to use the escape sequence \Q \E to quote metacharacters so that they won't be interpreted:

```
print if m/\Q$user_pattern\E/o;
```

Q49: My CGI script needs more privileges than it's getting as user "nobody". How do I run a Perl script as suid?

First of all, do you really need to run your Perl script as suid? This represents a major risk insofar as giving your script more privileges than the "nobody" user has also increases the potential for damage that a subverted script can cause. If you're thinking of giving your script root privileges, think it over extremely carefully.

You can make a script run with the privileges of its owner by setting its "s" bit:

```
chmod u+s foo.pl
```

You can make it run with the privileges of its owner's group by setting the s bit in the group field:

```
chmod g+s foo.pl
```

However, many Unix systems contain a hole that allows suid scripts to be subverted. This hole affects only scripts, not compiled programs. On such systems, an attempt to execute a Perl script with the suid bits set will result in a nasty error message from Perl itself.

You have two options on such systems:

1. You can apply a patch to the kernel that disables the suid bits for scripts. Perl will detect these bits nevertheless and do the suid function safely. See the Perl faq for details on obtaining this kernel patch. This faq can be found at:

   ftp://rtfm.mit.edu/pub/usenet-by-group/comp.lang.perl/

2. You can put a C wrapper around the program. A typical wrapper looks like this:

```
#include <unistd.h>
void main () {
execl("/usr/local/bin/perl","foo.pl","/local/web/cgi-bin/foo.pl",NULL);
}
```

   After compiling this program, make it suid. It will run under its owner's permission, launching a Perl interpreter and executing the statements in the file "foo.pl".

Another option is to run the server itself as a user that has sufficient privileges to do whatever the scripts need to do. If you're using the CERN server, you can even run as a different user for each script. See the CERN documentation for details.

[BACK]

Lincoln D. Stein, lstein@genome.wi.mit.edu
Whitehead Institute/MIT Center for Genome Research
Last modified: Thu Nov 7 15:50:47 EST 1996

LECTURE 10:
Server Side Includes

- Provides a simple short cut to adding dynamic functionality to Web pages
- Directives to the server which are included directly into your HTML code
- Not supported by all servers

  Supported on Apache, NCSA and Netscape, on others you can try the fakessi.pl to emulate SSI (but ask yourself first if its really worth it and if there isn't a better way to do what you are trying to do).

  - Many System Administrators turn this option off for security reasons
  - Can load down the server if there are lots of SSI requests

SERVER SIDE INCLUDE DIRECTIVES

COMMAND	PARAMETER	DESCRIPTION
echo	var	Inserts value of speicall SSI variable as well as other environment variables
include		Inserts text of file into current document
	file	Pathname relative to current directory
	virtual	Virtual path to a document on the server
fsize	file	Inserts the size of a specified file
flastmod	file	Inserts the last modification date and time for a specified file
exec		Executes external programs and inserts the output into the current document
	cmd	Command or application on the host (server)
	cgi	CGI program on the host to execute
config		Modifies various aspects of the Server Side Include
	errmsg	Default error message return to client
	sizefmt	Format for size of the file (bytes or kilobytes)
	timefmt	Format for date

Server Configuration

For those servers that support SSI the do so by specific directives in the server configuration files. This is usually done by the webmaster or systems administrator for you server. Many webmasters consider SSI a potential mechanism for crackers to gain un-authorized access to their system and have not allowed SSI to be used. If you really need they might make a case-by-case exception if you can convince them that you won't do anything insecure.

Sample Configuration Directives for SSI

Apache 1.1.1 HTTPd Server

- srm.conf file

```
# To use server-parsed HTML files
AddType text/html .shtml
AddHandler server-parsed .shtml
```

- access.conf file

```
<Directory /usr/local/etc/httpd/htdocs>

# This may also be "None", "All", or any combination of "Indexes",
# "Includes", "FollowSymLinks", "ExecCGI", or "MultiViews".

# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you (or at least, not yet).

Options Includes ExecCGI Indexes FollowSymLinks
</Directory>
```

SSI DIRECTIVES: SYNTAX and EXAMPLES

- **Basic syntax**

```
<!--#command parameter="argument"-->
```

- **Echo Environment Variables**

```
<!--#echo var="DATE_LOCAL"-->
```

- **Document Templates**
  Use this to include information that is common to many different web pages or to up-date a page periodically

```
<!--#include file="signature.html"-->

<!--#include virtual="/public/signature.html"-->
```

- **File Statistics**

Let the user know the size of a file (that way they can decide if they want to download it or not. You can also let the user know when the file was last modified so they can decide if it is time to get the latest version of your report.

```
<!--#fsize file="report.ps"-->

<!--#flastmod file="report.ps"-->
```

- **Executing External Commands**
  Allows execution of commands on the server, and can be **VERY DANGEROUS**!

VERY DANGEROUS DO NOT DO THIS

```
<!--#exec cmd="/usr/ucb/finger me@myhost.com"-->
```

Potential Problem
Many sites have guestbooks which allow users to enter text or comments. Which are then saved to a file as HTML. A user could enter the following text:

```
<!--#exec cmd="/bin rm -fr /"-->
```

which could be executed as part of the HTML.

- **Executing CGI Programs**

```
<!--#exec cgi="mycgifile.pl"-->
```

`cgi` can only output text in this case. Thus the `Content-type` must be either `text/plain` or `text/html`.

Random link Example

In your HTML file you would include the SSI directive:

```
<!--#exec cgi="random.cgi"-->
```

where the file `random.cgi` is given here:

```
#!/usr/local/bin/perl
# file: random.cgi

@URL = ("http://www.ora.com",
        "http://www.digital.com",
        "http://www.ibm.com",
        "http://www.radius.com");

srand (time | $$);

$number_of_URL = $#URL;
$random = int (rand ($number_of_URL));

$random_URL = $URL[$random];
```

```
print <<End_of_Output;
  Content-type: text/html

  <A HREF="$random_URL">Click here for a random Web site!</A>

End_of_Output


exit (0);
```

Sample Output

```
  Content-type: text/html

  <A HREF="http://www.ibm.com">Click here for a random Web site!</A>
```

Counting Days Until... Example

```
#!/usr/local/bin/perl

require "timelocal.pl";
require "bigint.pl";

($chosen_date = $ARGV[0]) =~ s/\s*//g;

if ($chosen_date =~ m|^(\d+)/(\d+)/(\d+)$|) {
    ($month, $day, $year) = ($1, $2, $3);

    $month -= 1;

    if ($year > 1900) {
        $year -= 1900;
    }

    $chosen_secs = &timelocal (undef, undef, undef, $day, $month, $year);

    $seconds_in_day = 60 * 60 * 24;
    $difference = &bsub ($chosen_secs, time);
    $no_days = &bdiv ($difference, $seconds_in_day);
    $no_days =~ s/^(\+|-)//;

    print $no_days;
    exit(0);

} else {
    print " [Error in date format] ";
    exit(1);
}
```

- **Tailoring SSI Input**
  - Use config SSI command to fine tune the format
    - Error messages

- File size information
- Date/time format

```
<!--#config errmsg="Hey, Yea You, you screwed up!"-->

<!--#config sizefmt="OPTION"-->
        where OPTION = abbrev   (nearest Kbyte)
                       byte              (Bytes)

<!--#config timefmt="timeformatcodes"-->
(see table below for time format codes)
```

SSI TIME FORMAT CODES

FORMAT	VALUE	EXAMPLE
%a	Day of week, abbreviated	Sun
%A	Day of week	Sunday
%b	Month Name, abbreviated	Mar
%B	Month name	March
%d	Date, no leading zeros	1 not 01
%D	Date as "%m/%d/%y"	8/13/96
%e	Date, leading zeros	01
%H	hour (24-hour clock)	15
%I	hour (12-hour clock)	3
%j	Decimal day of the year	257
%m	month number	10
%M	Minutes	08
%p	am | pm	a.m.
%r	Time as "%I:%M:%S AM|PM"	06:15:33 PM
%S	Seconds	09
%T	24 hour time	16:55:33
%U	Weak of the year (also %W)	39
%w	Day of week, number	05
%y	Year of the century	96
%Y	Year	1996
%Z	Time Zone	PST

EXAMPLE USING MULTIPLE SSI COMMANDS

This example combines several of the features discussed in this lecture. The HTML file, `current_doc.shtml`, is shown below, along with the address file `address.shtml`.

- `current_doc.shtml`
- `address.shtml`

TRY IT

- **Common Errors**

    1. Don't forget the # sign
    2. Don't add any extra spaces between the - and the #
    3. Always check that the server is configured to allow SSI (check the server config files or ask you IAP or webmaster

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 10]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

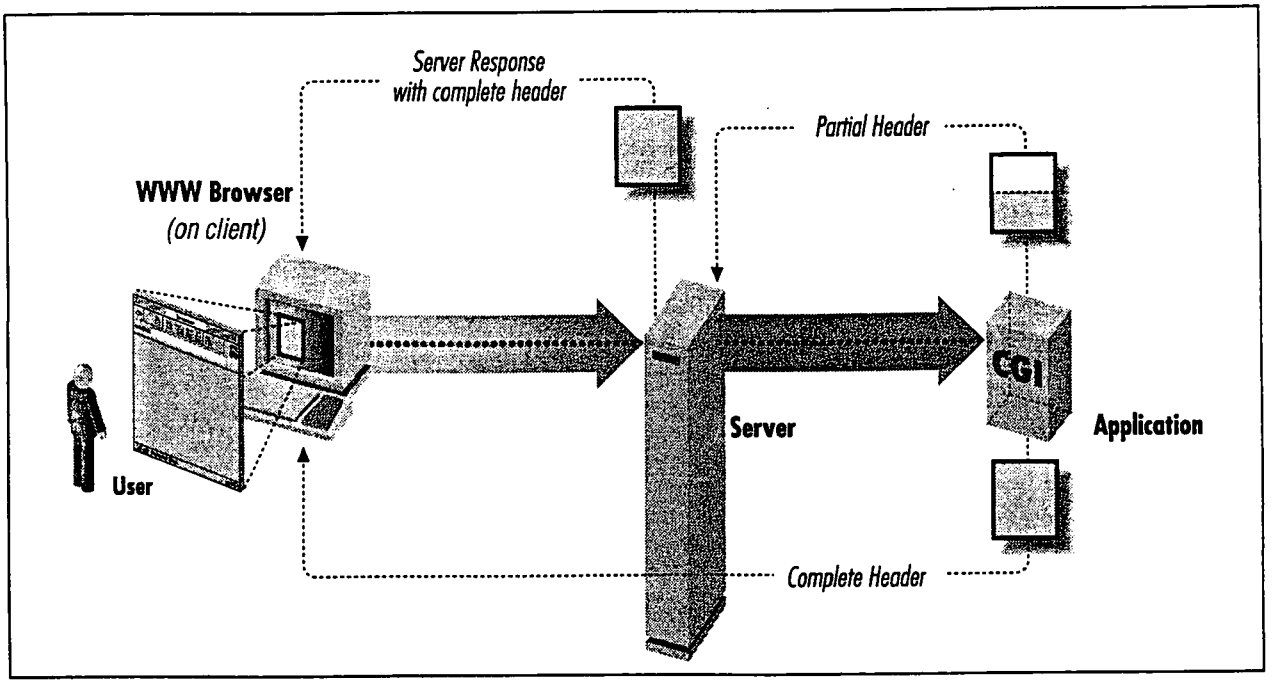


Figure 3-2. Partial and complete headers

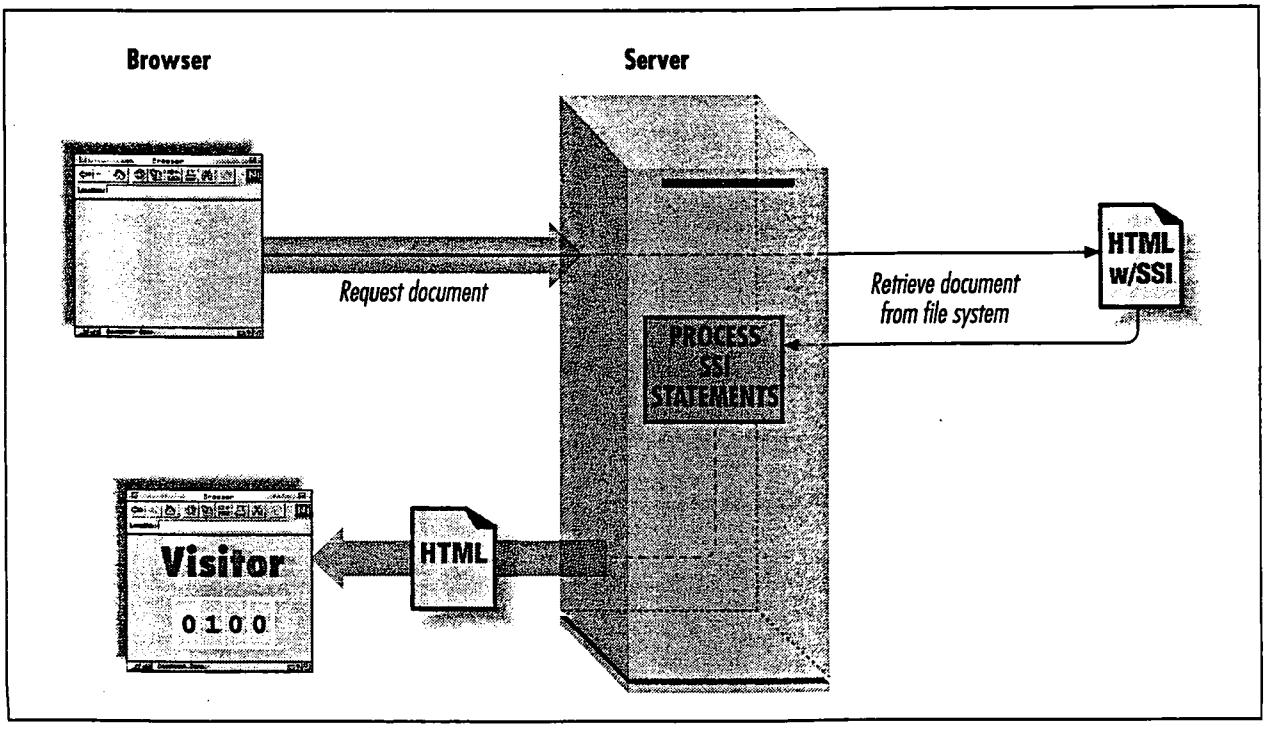


Figure 5-1. Server Side Includes

Listing of current_doc.shtml

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML>
<HEAD>
<TITLE>File Summary: SSI Example</TITLE>
</HEAD>

<BODY TEXT="#000000" LINK="#0000FF" VLINK="#FF0000" ALINK="#AA1177" BGCOLOR="#E4BB94">

<H2 ALIGN=CENTER>File Summary: SSI Example</H2>
<P><HR ALIGN=CENTER NOSHADE SIZE="2"><P>
Welcome to the Server Side Include Example Page!<P>
We are running on the <B> <!--#echo var="SERVER_NAME"--></B> server.
<BR>You are visiting us from: <B><!--#echo var="REMOTE_HOST"--></B>.

<P>
The document you are viewing is titled:<BR>
<B><!--#echo var="DOCUMENT_NAME"--></B>,<BR>
and you can access it a later time by opening the URL to:
<I><!--#echo var="DOCUMENT_URI"--></I>. <P>
Please add this to your bookmark list.
<P>
The following users are currently logged in:<BR>
<!--#exec cmd="/usr/bin/who -q"--><P>
<HR>
Document last modified on <!--#echo var="LAST_MODIFIED"-->.

<P>
<!--#include file="address.shtml"-->
</BODY>
</HTML>
```

Listing for address.shtml

```
<HR SIZE="2" NOSHADE ALIGN="CENTER">
<ADDRESS>
<PRE>
Sam Gasster                              Muon Software, Inc.
Los Angeles, CA.                               (310)444-1234
gasster@pacificnet.net
</PRE>
<P>
<HR SIZE="2" NOSHADE ALIGN="CENTER">
<P>
<!--#config timefmt="%A, %D at %r"-->
Up-dated: <!--#echo var="LAST_MODIFIED"-->.
</ADDRESS>
```

LECTURE 11: CGI APPLICATIONS FORMS AND GATEWAYS

READING ASSIGNMENT:
CGI Programming on the WWW Ch. 8,9,10

HOMEWORK ASSIGNMENT:
Finish any homework you've not done!

TOPICS

- Multiple Form Interaction
- Gateways, Databases, and Search/Index Utilities
- Gateways to Internet Information Servers

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 12]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

LECTURE 11: Multiple Form Interaction

Introduction

- What do `STATE` and `MAINTAIN STATE` refer to?

  `STATE`
  : This refers to information about a client/server interaction, such as what documents were requested by the client, what form data were passed from the client to the server, and any CGI scripts.

  `MAINTAIN STATE`
  : This refers to the ability to track and record the state information for a given client/server interaction.

- One of the main problems with HTTP is the inability to maintain state between documents. The HTTP protocol was developed primary so that the WWW servers could just sit back and serve up document requests, not worrying about who asked for what and when. As the WWW developed the need for state information became evident and several *ad-hoc* solutions have been developed. The best solution is yet to be defined, and modifications to the HTTP protocol to include state tracking are underconsideration.

- How to get around this problem?

  - Hidden fields (`HTML FORMS`)
  - CGI Side Include (CSI)
  - Netscape Persistent Cookies

HIDDEN FIELDS

- HTML <INPUT> TAG used with the <FORM> TAG.

```
This is a hidden Field
<INPUT TYPE="hidden" NAME="HD_01" VALUE="Something Goes Here">
```

- Not visible on client browser, but visible via VIEW SOURCE. Not for passing secure data.

- Utility

  - Selected Hidden Field data can be copied from one form to another in a dynamic manner.
  - Allows tracking forms and user inputs between forms.

- Advantages

  - No temporary files need be written
  - Does not use Netscape Cookies (not supported on all clients - yet)

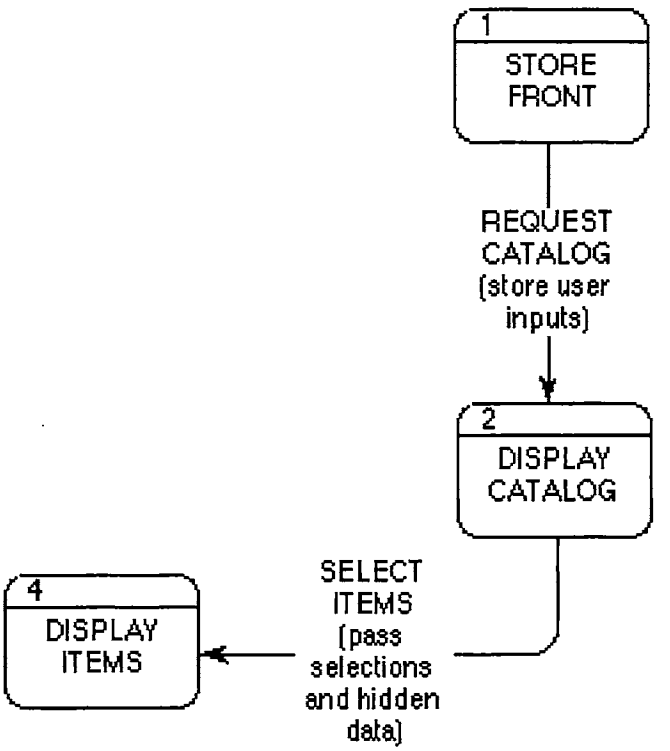
- Disadvantages
    - Forms can get quite long and contain many hidden fields
    - More complex script required to handle hidden data

HIDDEN FIELDS: ILLUSTRATIVE EXAMPLE

SIMPLE SHOPPING CART

This example illustrates the use of hidden fields and the concept of an on-line shopping service.

- **Task-Action Diagram**



- **Facilities**

Store Front
: Provides initial interface to catalop shopping via `books.html`. Get initial user information (name and email address) and request catalog.

Catalog Display and Processing
: Perl script (shopping.pl) is used to parse the catalog file (catalog.html) and user selections.

Display Selections
: The same perl script also displays the user selections.

- **TRY IT**

shopping.pl -- main program

Functions

`display_file()`
: Prints the file specified by the additional path information (`PATH_INFO`) to `STDOUT` preceeded by the HTTP header (text/html).

`parse_form_data(*STATE)`
: Parses form data sent using the `POST` method and returns it using the typeglob or alias `*STATE`

`parse_file()`
: Takes the data from previous forms and inserts it into the current form using `HIDDEN` fields.

`thank_you()`
: Thanks the user for using the shopping program and displays the results of their selection

`return_error()`
: Error handler used if scripts are incorrectly requested or requested with incorrect data.

```
if ($request_method eq "GET") {
        if ($form_file) {
                &display_file ();
        } else {
                &return_error (500, "CGI Shopping Cart Error",
                        "An initial form must be specified.");
        }
} elsif ($request_method eq "POST") {

        &parse_form_data (*STATE);

        if ($form_file) {
                &parse_file ();
        } else {
                &thank_you ();
        }
} else {
        &return_error (500, "Server Error",
                        "Server uses unsupported method");
}
```

CGI SIDE INCLUDES

The tecniques of CGI Side Includes (CSI) is similar to SSI, except the user provides a CGI program to process and parse the CSI tags, rather than the HTTP server.

- Embed special codes or tags into the HTML document using a syntax similar to SSI

```
<!--#command argument=parameter-->
```

- HTML documents are sent to the client via a CGI parsing program that reads the CSI tags and replaces them with the required information.
- Allows insertion of dynamic information in otherwise static documents
- Can be quite cumbersome to handle for large numbers of interacting documents.

- Additional processing time may slow document retrieval

CSI Parsing Script Example

This subroutine is taken from *CGI Programming on the WWW,* page 196-197. The line numbers were inserted form discuss only. Remove them if you want to use this script.

pseudo_ssi.pl

Discussion

The main function of `pseduo_ssi` is to parse the CSI statements within the HTML file. The `pseudo_ssi` function was written for the specific application illustrated in the textbook. However, it does provide a general idea of how to parse the CSI statements. The key element is the pattern match statement to find the CSI statements and extract the command, arguement and parameters.

Multiple Form Example: TV/Movie Survey

Sample Session

Every HTML page viewed by the user is parsed by the CGI script `survey.pl`. This script handles the CSI statements as well as parsing the form data. This example is discussed in *CGI Programming on the WWW* .

A. Start survey
Need a primary link to get everything started. In this case I use localhost since I am running the server locally.

```
http://localhost/cgi-bin/survey.pl/start_survey.html
```

TRY IT

This calls the CGI program `survey.pl` with the extra path information set to the HTML file we wish to parse, `start_survey.html`.

```
<FORM ACTION="/cgi-bin/survey.pl?cgi_cookie=<!--#insert var="COOKIE"-->
&cgi_form_num=<!--#insert var="NUMBER"-->"
METHOD="POST">
```

- request initial survey form using `GET` method
- name of HTML page is appended to URL as extra path information (`PATH_INFO`)
- date is inserted using CSI statement
- Cookie is initialized using name of `REMOTE_HOST` and time data
- convert Cookie to HTTP accetable format
- parse `$form_file` and print to `STDOUT` (sending it back to client)

```
$cookie = join ("_", $ENV{'REMOTE_HOST'}, time);
$cookie = &escape($cookie);
&pseudo_ssi ($form_file, $cookie, $type, $form_num);
```

B. Submit Initial survey form (`start_survey.html, POST`)

- `start_survey.html` is parsed and sent to the client with the CSI data inserted (in this case the date/time). This form has several fields (shown here)
  - User Name

    - User Email
    - Survey selection: TV shows or Movies

  and uses the POST method with the following ACTION attribute (line breaks inserted for readability):
  - select survey type (TV or Movie)
  - write initial form data to file on `/tmp`
  - parse new survey form and increment form number
  - send form to user

C. User submits first survey form (`POST`)
  - User selects survey options and submits
  - Note query string data attached to URL to track temporary file for this user and form number
  - write first set of survey data to file on `/tmp`
  - increment form number
  - parse and send final form to user

D. User submits last survey form (`POST`)
  - User selects survey options and submits
  - Note query string data attached to URL to track temporary file for this user and form number
  - write first set of survey data to file on `/tmp`
  - increment form number
  - parse and send final form to user

E. Return Thank You page to user
  - Thank you page formatted and parsed by survey.pl
  - delete temporary file from `/tmp`

NETSCAPE PERSISTENT COOKIES

- What are Netscape Cookies?
  - Method of storing information about the current client/server session on the client side.
  - Initially developed by Netscape and only supported by Netscape Navigotor browser
  - Users file on client side to store data
  - Uses additional HTTP response headers to write (Set-Cookie) and send (Cookie) data
  - Netscape Cookie Specification

Sample Cookie File

```
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file!  Do not edit.

cgi.netscape.com        FALSE   /       FALSE   946684799       NETSCAPE_VERIFY c65ff94b,c6dd7da9
.infoseek.com   TRUE    /       FALSE   858835466       InfoseekUserId  420CE696D44317FBEAEC3EF7C50D536D
.netscape.com   TRUE    /       FALSE   946684799       NETSCAPE_ID     c65ffb1e,c701644d
.disney.com     TRUE    /       FALSE   946684799       DISNEY  205.162.154.411882683488776863
.adobe.com      TRUE    /       FALSE   946684799       INTERSE 207.171.20.5510854849286808102
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 11]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

```
#!/usr/local/bin/perl

$webmaster = "shishir\@bu\.edu";
$document_root = "/home/shishir/httpd_1.4.2/public";
$request_method = $ENV{'REQUEST_METHOD'};
$form_file = $ENV{'PATH_INFO'};
$full_path = $document_root . $form_file;

$exclusive_lock = 2;
$unlock = 8;

if ($request_method eq "GET") {
    if ($form_file) {
        &display_file ();
    } else {
        &return_error (500, "CGI Shopping Cart Error",
            "An initial form must be specified.");
    }
} elsif ($request_method eq "POST") {
    &parse_form_data (*STATE);

    if ($form_file) {
        &parse_file ();
    } else {
        &thank_you ();
    }
} else {
    &return_error (500, "Server Error",
                        "Server uses unsupported method");
}

exit (0);

sub display_file
{
    open (FILE, "<" . $full_path) ||
        &return_error (500, "CGI Shopping Cart Error",
            "Cannot read from the form file [$full_path].");

    flock (FILE, $exclusive_lock);

    print "Content-type: text/html", "\n\n";
    while (<FILE>) {
        print;
    }

    flock (FILE, $unlock);
    close (FILE);
}

sub parse_file
{
    local ($key, $value);

    open (FILE, "<" . $full_path) ||
        &return_error (500, "CGI Shopping Cart Error",
            "Cannot read from the form file [$full_path].");

    flock (FILE, $exclusive_lock);

    print "Content-type: text/html", "\n\n";
```



```
    while (<FILE>) {

        if (/<\s*form\s*.*>/i) {
            print;

            foreach $key (sort (keys %STATE)) {
                $value = $STATE{$key};
                print <<End_of_Hidden;
<INPUT TYPE="hidden" NAME="$key" VALUE="$value">
End_of_Hidden
            }

        } else {
            print;
        }

    }

    flock (FILE, $unlock);
    close (FILE);

}

sub thank_you
{
    local ($key, $value);

    print <<Thanks;
Content-type: text/html

<HTML>
<HEAD><TITLE>Thank You!</TITLE></HEAD>
<BODY>
<H1>Thank You!</H1>
Thank you again for using our service. Here are the items
that you selected:
<HR>
<P>

Thanks

    foreach $key (sort (keys %STATE)) {
        $value = $STATE{$key};

            $key =~ s/^\d+\s//;

        if ($value =~ /\0/) {
            print "<B>", $key, "</B>", "<BR>", "\n";
            $value =~ s/\0/<BR>\n/g;
            print $value, "<BR>", "\n";
        } else {
            print $key, ": ", $value, "<BR>", "\n";
        }
    }

    print "<HR>", "\n";
        print "</BODY></HTML>", "\n";
}

sub parse_form_data
{
    local (*FORM_DATA) = @_;
```

```
    local ($query_string, @key_value_pairs, $key_value, $key, $value);

    read (STDIN, $query_string, $ENV{'CONTENT_LENGTH'});

    @key_value_pairs = split (/&/, $query_string);

    foreach $key_value (@key_value_pairs) {
        ($key, $value) = split (/=/, $key_value);

    $key   =~ tr/+/ /;
    $value =~ tr/+/ /;

    $key   =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C", hex ($1))/eg;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C", hex ($1))/eg;

        if (defined($FORM_DATA{$key})) {
            $FORM_DATA{$key} = join ("\0", $FORM_DATA{$key}, $value);
        } else {
            $FORM_DATA{$key} = $value;
        }
    }
}

sub return_error
{
    local ($status, $keyword, $message) = @_;

    print "Content-type: text/html", "\n";
    print "Status: ", $status, " ", $keyword, "\n\n";

    print <<End_of_Error;

<title>CGI Program - Unexpected Error</title>
<h1>$keyword</h1>
<hr>$message</hr>
Please contact $webmaster for more information.

End_of_Error

    exit(1);
}
```

Lecture 11
Listing of pseudo_ssi.pl

```
1       sub pseudo_ssi
2       {
3               local ($file, $id, $kind, $number) = @_;
4               local ($command, $argument, $parameter, $line);
5
6               $file = $document_root . $file;
7
8               open (FILE, "<" . $file) ||
9                       &return_error (500, "CGI Network Survey Error",
10                              "Cannot open: form [$number], file [$file].");
11
12              flock (FILE, $exclusive_lock);
13
14              print "Content-type: text/html", "\n\n";
15
16              while (<FILE>) {
17                  while ( ($command, $argument, $parameter) =
18                          (/<!--\s*#\s*(\w+)\s+(\w+)\s*=\s*"?(\w+)"?\s*-->/io) ) {
19
20                      if ($command eq "insert") {
21                          if ($argument eq "var") {
22                              if ($parameter eq "COOKIE") {
23                                  s//$id/;
24                              } elsif ($parameter eq "DATE_TIME") {
25                                  local ($time) = &get_date_time();
26                                  s//$time/;
27                              } elsif ($parameter eq "NUMBER") {
28                                  s//$number/;
29                              } elsif ($parameter eq "SURVEY") {
30                                  s//$kind/;
31                              } else {
32                                  s///;
33                              }
34                          } else {
35                              s///;
36                          }
37                      } else {
38                          s///;
39                      }
40                  }
41
42                  print;
43
44              }
45
46              flock (FILE, $unlock);
47              close (FILE);
48      }
```

[BACK]

PERSISTENT CLIENT STATE HTTP COOKIES

Preliminary Specification - Use with caution

Note:
This material was obtained directly from the Netscape WWW site at http://www.netscape.com/newsref/std/cookie_spec.html. It was not marked with any copyright, but I am assuming it probably is copyrighted by Netscape. I have only made small formatting modifications for purposes of this course.
- Sam Gasster

INTRODUCTION

Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications.

OVERVIEW

A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store. Included in that state object is a description of the range of URLs for which that state is valid. Any future HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server. The state object is called a **cookie**, for no compelling reason.

This simple mechanism provides a powerful new tool which enables a host of new types of applications to be written for web-based environments. Shopping applications can now store information about the currently selected items, for fee services can send back registration information and free the client from retyping a user-id on next connection, sites can store per-user preferences on the client, and have the client supply those preferences every time that site is connected to.

SPECIFICATION

A cookie is introduced to the client by including a **Set-Cookie** header as part of an HTTP response, typically this will be generated by a CGI script.

Syntax of the Set-Cookie HTTP Response Header

This is the format a CGI script would use to add to the HTTP headers a new piece of data which is to be stored by the client for later retrieval.

```
Set-Cookie: NAME=VALUE ; expires=DATE ;
path=PATH ; domain=DOMAIN_NAME ; secure
```

NAME=VALUE
This string is a sequence of characters excluding semi-colon, comma and white space. If there is a

need to place such data in the name or value, some encoding method such as URL style %XX encoding is recommended, though no encoding is defined or required.

This is the only required attribute on the **Set-Cookie** header.

expires=*DATE*

The **expires** attribute specifies a date string that defines the valid life time of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out.

The date string is formatted as:

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

This is based on RFC 822, RFC 850, RFC 1036, and RFC 1123, with the variations that the only legal time zone is **GMT** and the separators between the elements of the date must be dashes.

expires is an optional attribute. If not specified, the cookie will expire when the user's session ends.

Note: There is a bug in Netscape Navigator version 1.1 and earlier. Only cookies whose **path** attribute is set explicitly to "/" will be properly saved between sessions if they have an **expires** attribute.

domain=*DOMAIN_NAME*

When searching the cookie list for valid cookies, a comparison of the **domain** attributes of the cookie is made with the Internet domain name of the host from which the URL will be fetched. If there is a tail match, then the cookie will go through **path** matching to see if it should be sent. "Tail matching" means that **domain** attribute is matched against the tail of the fully qualified domain name of the host. A **domain** attribute of "acme.com" would match host names "anvil.acme.com" as well as "shipping.crate.acme.com".

Only hosts within the specified domain can set a cookie for a domain and domains must have at least two (2) or three (3) periods in them to prevent domains of the form: ".com", ".edu", and "va.us". Any domain that fails within one of the seven special top level domains listed below only require two periods. Any other domain requires at least three. The seven special top level domains are: "COM", "EDU", "NET", "ORG", "GOV", "MIL", and "INT".

The default value of **domain** is the host name of the server which generated the cookie response.

path=*PATH*

The **path** attribute is used to specify the subset of URLs in a domain for which the cookie is valid. If a cookie has already passed **domain** matching, then the pathname component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request. The path "/foo" would match "/foobar" and "/foo/bar.html". The path "/" is the most general path.

If the **path** is not specified, it as assumed to be the same path as the document being described by the header which contains the cookie.

secure

If a cookie is marked **secure**, it will only be transmitted if the communications channel with the host is a secure one. Currently this means that secure cookies will only be sent to HTTPS (HTTP over SSL) servers.

If **secure** is not specified, a cookie is considered safe to be sent in the clear over unsecured channels.

Syntax of the Cookie HTTP Request Header

When requesting a URL from an HTTP server, the browser will match the URL against all cookies and if any of them match, a line containing the name/value pairs of all matching cookies will be included in the HTTP request. Here is the format of that line:

```
Cookie: NAME1=OPAQUE_STRING1 ; NAME2=OPAQUE_STRING2 ...
```

Additional Notes

- Multiple **Set-Cookie** headers can be issued in a single server response.
- Instances of the same path and name will overwrite each other, with the latest instance taking precedence. Instances of the same path but different names will add additional mappings.
- Setting the path to a higher-level value does not override other more specific path mappings. If there are multiple matches for a given cookie name, but with separate paths, all the matching cookies will be sent. (See examples below.)
- The expires header lets the client know when it is safe to purge the mapping but the client is not required to do so. A client may also delete a cookie before it's expiration date arrives if the number of cookies exceeds its internal limits.
- When sending cookies to a server, all cookies with a more specific path mapping should be sent before cookies with less specific path mappings. For example, a cookie "name1=foo" with a path mapping of "/" should be sent after a cookie "name1=foo2" with a path mapping of "/bar" if they are both to be sent.
- There are limitations on the number of cookies that a client can store at any one time. This is a specification of the minimum number of cookies that a client should be prepared to receive and store.
    - 300 total cookies
    - 4 kilobytes per cookie, where the name and the OPAQUE_STRING combine to form the 4 kilobyte limit.
    - 20 cookies per server or domain. (note that completely specified hosts and domains are treated as separate entities and have a 20 cookie limitation for each, not combined)

  Servers should not expect clients to be able to exceed these limits. When the 300 cookie limit or the 20 cookie per server limit is exceeded, clients should delete the least recently used cookie. When a cookie larger than 4 kilobytes is encountered the cookie should be trimmed to fit, but the name should remain intact as long as it is less than 4 kilobytes.
- If a CGI script wishes to delete a cookie, it can do so by returning a cookie with the same name, and an **expires** time which is in the past. The path and name must match exactly in order for the expiring cookie to replace the valid cookie. This requirement makes it difficult for anyone but the originator of a cookie to delete a cookie.
- When caching HTTP, as a proxy server might do, the **Set-cookie** response header should never be cached.
- If a proxy server receives a response which contains a **Set-cookie** header, it should propagate the **Set-cookie** header to the client, regardless of whether the response was 304 (Not Modified) or 200 (OK).

  Similarly, if a client request contains a Cookie: header, it should be forwarded through a proxy, even if the conditional If-modified-since request is being made.

EXAMPLES

Here are some sample exchanges which are designed to illustrate the use of cookies.

First Example transaction sequence:

Client requests a document, and receives in the response:

```
Set-Cookie: CUSTOMER=WILE_E_COYOTE; path=/; expires=Wednesday, 09-Nov-99 23:12:40
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE
```

Client requests a document, and receives in the response:

```
Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001
```

Client receives:

```
Set-Cookie: SHIPPING=FEDEX; path=/foo
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001
```

When client requests a URL in path "/foo" on this server, it sends:

```
Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001; SHIPPING=FEDEX
```

Second Example transaction sequence:

Assume all mappings from above have been cleared.

Client receives:

```
Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

When client requests a URL in path "/" on this server, it sends:

```
Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001
```

Client receives:

```
Set-Cookie: PART_NUMBER=RIDING_ROCKET_0023; path=/ammo
```

When client requests a URL in path "/ammo" on this server, it sends:

```
Cookie: PART_NUMBER=RIDING_ROCKET_0023; PART_NUMBER=ROCKET_LAUNCHER_0001
```

NOTE: There are two name/value pairs named "PART_NUMBER" due to the inheritance of the "/" mapping in addition to the "/ammo" mapping.

LECTURE 11: Gateways, Databases, and Search/Index Utilities

What is a Gateway?

Gateway

A computer system that transfers data between normally incompatible applications or networks. It reformats the data so that it is acceptable for the new network (or application) before passing it on.

-From: *The Whole Internet User's Guide & Catalog* , by Ed Krol, 2nd ed., O'Reilly & Associates, Inc. (1996).

Gateways are used to interconnect a variety of services on the Internet and WWW. In this Lecture we will discuss gateways to:

- Databases
- Email
- Search index utilites
- Information servers

Database Overview

Database Gateway Applications

- Phonebook
- Personnel or Employee Records
- Shopping Catalog
- Documentation

Example: Employee Phonebook Database

Although this is a relatively simple example it illustrates the key elements of developing and using a database gateway. It would not be difficult to expand this example and provide a complete on-line phonebook gateway for your company's intranet.

The phonebook example below uses the Sprite module provided by Gundavaram.

FILES

phone_form.html

HTML form to access phone book. By entering partial data the use will be able to retrieve all database fields matching the selected criteria.

db_phone.pl (or .cgi)
Perl script for parsing search requests and accessing database using the Sprite module.

phone.db
The phone book database. Simple text delimited ASCII database.

Some Sample Search Results

- Single Last Name Only: `Levine`

Last	First	Job Title	Department	EMail	Phone
Levine	Julia	Administrativ	Administrat	julia	(617)555-3056

- Area Code Only: `(617)`

Last	First	Job Title	Department	EMail	Phone
Supra	John	System Operat	Systems	jsupra	(617)555-1578
Painton	Todd	Network Engin	Systems	tpainton	(617)555-6530
Martin	Robert	Sales Represe	Sales	martinr	(617)555-7406
Levine	Julia	Administrativ	Administrat	julia	(617)555-3056
Keenan	Jeff	Manager	Software	jeffk	(617)555-7769
Nets	Laurie	Group Leader	Development	lnets	(617)555-9962

- First Letter of First Name: `J`

Last	First	Job Title	Department	EMail	Phone
Supra	John	System Operat	Systems	jsupra	(617)555-1578
Levine	Julia	Administrativ	Administrat	julia	(617)555-3056
Keenan	Jeff	Manager	Software	jeffk	(617)555-7769

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 11]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Lecture 11
Database Overview

What is a Database (DB)?

Database
Most generally, any clearly identified collection of data.

DBMS - Database Management System
Collection of hardware and software that organizes and provides access to a database. The functionality normally includes the ability to: create, list/print, select parts or all, up-date or add and delete.

RDBMS - Relational Database Management System
A database organizationa scheme that treats files as tables of data in which rows represent fixed length records and columns represent fields. Some of the data items in one type of record can refer to records of a different type. Relational DBs allow the user to link or cross-reference information.

SQL - Structure Query Language
Used with relational databases, allows the user to present near-English queries to the DB manager in order to view the database in a variety of different ways.

Note:
The above definitions were all adopted from *Webster's NewWorld Dictionary of Computer Terms*, 5th ed.

What kinds of Databases are there?

There are many different kinds of databases, implemented on a variety of platforms, under a variety of operating systems (Unix, PC, Mac, etc.).

Hierarchial
A collection of data in which the individual elemets are organized as an inverted tree with a series of nodes connected by branches.

Network
Similar to a Hierarchial database, except that each child node may have more than one parent.

Relational
A database where the data is represented as a table containing columns and rows. This seems to be one of the most popular database types and the only one we'll really discuss in this course.

Object-Oriented
Used in OO programming, the data can be stored as objects.

If you are really interested in learning about this stuff check on-line for more information, read a database book and look at the Perl manpages and related books for more information.

Database Access in Perl

There are basically two methods for working with databases in Perl.

1. Implement the DB entirely in Perl
2. Use Perl as an interface to an exsiting DB

We've already seen simple example of Option 1, with the homework problems for manipulating address books, congress.db and the web server log file. You can design the database yourself, specifying the field delimiters (often a colon :), the field data types (text or numerical; not much difference in Perl) and the DB record structure (what fields constitute each record).

The Standard Perl Library Database Interfaces

The following modules come with the standard distribution of Perl and are part of the standard Perl library.

Perl Module	Function
AnyDBM_File	Provides a frameworkd to access a variety of DBMs
DB_File	Access to Berkeley DB files
GDBM_File	Access the GNU database files
NDBM_File	Berkeley Unix data files
ODBM_File	Access ODBM database files
SDBM_File	Access SDBM database files

There are also several other Perl programs and modules for accessing databases. For example, from the textbook mentions the following Perl derviatives (running under Perl 4):

- oraperl - Accessing Oracle databases
- sybperl - Accessing Sybase databases

Note that these have been superceeded in Perl 5 with:

- DBD::Oracle -- an Oracle 7 interface for Perl 5, by Tim Bunce
- sybperl 2.0 -- The Sybase extensions for Perl5.

The textbook, *CGI Programming on the WWW* provides a mini-SQL database environment using a module called Sprite. See the attached documentation. Sprite is designed to access simple text delimited databases, but provides considerable flexibility when it comes to database query. If you have little or no experience with database environments and want an easy introduction to using databases with Perl (and CGI) then I recommend trying Sprite.

[BACK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

AnyDBM_File.pm Perl Manual Page

NAME

AnyDBM_File - provide framework for multiple DBMs
NDBM_File, ODBM_File, SDBM_File, GDBM_File - various DBM implementations

SYNOPSIS

```
use AnyDBM_File;
```

DESCRIPTION

This module is a *pure virtual base class* -- it has nothing of its own. It's just there to inherit from one of the various DBM packages. It prefers ndbm for compatibility reasons with Perl 4, then Berkeley DB (See DB_File), GDBM, SDBM (which is always there--it comes with Perl), and finally ODBM. This way old programs that used to use NDBM via dbmopen() can still do so, but new ones can reorder @ISA:

```
@AnyDBM_File::ISA = qw(DB_File GDBM_File NDBM_File);
```

Note, however, that an explicit use overrides the specified order:

```
use GDBM_File;
@AnyDBM_File::ISA = qw(DB_File GDBM_File NDBM_File);
```

will only find GDBM_File. Having multiple DBM implementations makes it trivial to copy database formats:

```
use POSIX; use NDBM_File; use DB_File;
tie %newhash,  DB_File, $new_filename, O_CREAT|O_RDWR;
tie %oldhash,  NDBM_File, $old_filename, 1, 0;
%newhash = %oldhash;
```

DBM Comparisons

Here's a partial table of features the different packages offer:

	odbm	ndbm	sdbm	gdbm	bsd-db
Linkage comes w/ perl	yes	yes	yes	yes	yes
Src comes w/ perl	no	no	yes	no	no
Comes w/ many unix os	yes	yes[0]	no	no	no
Builds ok on !unix	?	?	yes	yes	?
Code Size	?	?	small	big	big
Database Size	?	?	small	big?	ok[1]
Speed	?	?	slow	ok	fast
FTPable	no	no	yes	yes	yes
Easy to build	N/A	N/A	yes	yes	ok[2]
Size limits	1k	4k	1k[3]	none	none
Byte-order independent	no	no	no	no	yes
Licensing restrictions	?	?	no	yes	no

[0]

on mixed universe machines, may be in the bsd compat library,which is often shunned.

[1]

Can be trimmed if you compile for one access method.

[2]

See DB_File. Requires symbolic links.

[3]

By default, but can be redefined.

SEE ALSO

dbm(3), ndbm(3), DB_File(3)

Documentation for Sprite.pm

NAME

Sprite - Perl 5.0 module to manipulate text delimited databases.

SYNOPSIS

```
use Sprite;

$rdb = new Sprite ();

$rdb->set_delimiter ("Read", "::");
$rdb->set_delimiter ("Write", "::");

$rdb->set_os ("UNIX");

$rdb->sql (<<Query);
    .
    .
    .
Query

$rdb->close ();
$rdb->close ($database);
```

DESCRIPTION

Here is a simple database where the fields are delimted by commas:

```
Player,Years,Points,Rebounds,Assists,Championships
...
Larry Joe Bird,12,28,10,7,3
Michael Jordan,10,33,6,5,3
Earvin Magic Johnson,12,22,7,12,5
...
```

Note: The first line must contain the field names (case sensitive).

Supported SQL Commands

Here are a list of the SQL commands that are supported by Sprite:

select - retrieves records that match specified criteria:

```
select col1 [,col2] from database
    where (cond1 OPERATOR value1)
    [and|or cond2 OPERATOR value2 ...]
```

The '*' operator can be used to select all columns.

The *database* is simply the file that contains the data. If the file is not in the current directory, the path must be specified.

Sprite does *not* support multiple tables (or commonly

knows as "joins").

Valid column names can be used where [cond1..n] and [value1..n] are expected, such as:

Example 1 :

```
select Player, Points from my_db
    where (Rebounds > Assists)
```

The following SQL operators can be used: =, <, >, <=, >=, <> as well as Perl's special operators: =~ and !~. The =~ and !~ operators are used to specify regular expressions, such as:

Example 2 :

```
select * from my_db
    where (Name =~ /Bird$/i)
```

Selects records where the Name column ends with "Bird" (case insensitive). For more information, look at a manual on regexps.

update - updates records that match specified criteria.

```
update database set (cond1 OPERATOR value1)[,(cond2 OPERATOR value2)...]*
   where (cond1 OPERATOR value1)
   [and|or cond2 OPERATOR value2 ...]
```

* = This feature was added as of version 3.1.

Example :

```
update my_db
    set Championships = (Championships + 1)
    where (Player = 'Larry Joe Bird')
```

```
update my_db
    set Championships = (Championships + 1),
        Years = (12)

    where (Player = 'Larry Joe Bird')
```

delete - removes records that match specified criteria:

```
delete from database
    where (cond1 OPERATOR value1)
    [and|or cond2 OPERATOR value2 ...]
```

Example :

```
delete from my_db
    where (Player =~ /Johnson$/i) or
          (Years > 12)
```

alter - simplified version of SQL-92 counterpart
Removes the specified column from the database. The other standard SQL functions for alter table are not supported:

```
alter table database
    drop column column-name
```

Example :

```
alter table my_db
    drop column Championships
```

insert - inserts a record into the database:

```
insert into database
    (col1, col2, ... coln)
values
    (val1, val2, ... valn)
```

Example :

```
insert into my_db
    (Player, Years, Points, Championships)
values
    ('Kareem Abdul-Jabbar', 21, 27, 5)
```

Note : You do not have to specify all of the fields in the database! Sprite also does not require you to specify the fields in the same order as that of the database.

Note : You should make it a habit to quote strings.

METHODS

Here are the four methods that are available:

set_delimiter

The set_delimiter function sets the read and write delimiter for the the SQL command. The delimiter is not limited to one character; you can have a string, and even a regexp (for reading only).

Return Value

None

set_os

The set_os function can be used to notify Sprite as to the operating system that you're using. Valid arguments are: "UNIX", "VMS", "MSDOS", "NT" and "MacOS". UNIX is the default.

Return Value

The previous OS value

sql The sql function is used to pass a SQL command to this module. All of the SQL commands described above are supported. The *select* SQL command returns an array containing the data, where the first element is the status. All of the other other SQL commands simply return a status.

Return Value

1 - Success

0 - Error

close

The close function closes the file, and destroys the database object. You can pass a filename to the function, in which case Sprite will save the database to that file.

Return Value

None

EXAMPLES

Here are two simple examples that illustrate some of the functions of this module:

Example 1

```
#!/usr/local/bin/perl5

use Sprite;

$rdb = new Sprite ();

# Sets the read delimiter to a comma (,) character. The delimiter
# is not limited to one character; you can have a string, or even
# a regexp.

$rdb->set_delimiter ("Read", ",");

# Retrieves all records that match the criteria.

@data = $rdb->sql (<<End_of_Query);

    select * from /shishir/nba
        where (Points > 25)

End_of_Query

# Close the database and destroy the database object (i.e $rdb).
# Since we did not pass a argument to this function, the data
# is not updated in any manner.

$rdb->close ();

# The first element of the array indicates the status.

$status = shift (@data);
$no_records = scalar (@data);

if (!$status) {
    die "Sprite database error. Check your query!", "\n";
} elsif (!$no_records) {
    print "There are no records that match your criteria!", "\n";
    exit (0);
} else {
    print "Here are the records that match your criteria: ", "\n";

    # The database returns a record where each field is
    # separated by the "\0" character.
```

```
        foreach $record (@data) {
            $record =~ s/\0/,/g;
            print $record, "\n";
        }
    }
```

Example 2

```
    #!/usr/local/bin/perl5

    use Sprite;

    $rdb = new Sprite ();
    $rdb->set_delimiter ("Read", ",");
    # Deletes all records that match the specified criteria. If the
    # query contains an error, Sprite returns a status of 1.

    $rdb->sql (<<Delete_Query)
                || die "Database Error. Check your query", "\n";

        delete from /shishir/nba
            where (Rebounds <= 5)

    Delete_Query

    # Access the database again! This time, select all the records that
    # match the specified criteria. The database is updated *internally*
    # after the previous delete statement.

    # Notice the fact that the full path to the database does not
    # need to specified after the first SQL command. This
    # works correctly as of version 3.1.

    @data = $rdb->sql (<<End_of_Query);

        select Player from nba
            where (Points > 25)

    End_of_Query

    # Sets the write delimiter to the (:) character, and outputs the
    # updated information to the file: "nba.new". If you do not pass
    # an argument to the close function after you update the database,
    # the modified information will not be saved.

    $rdb->set_delimiter ("Write", ":");
    $rdb->close ("nba.new");

    # The first element of the array indicates the status.

    $status = shift (@data);
    $no_records = scalar (@data);

    if (!$status) {
        die "Sprite database error. Check your query!", "\n";
    } elsif (!$no_records) {
        print "There are no records that match your criteria!", "\n";
        exit (0);
    } else {
        print "Here are the records that match your criteria: ", "\n";
```

```
        # The database returns a record where each field is
        # separated by the "\0" character.

        foreach $record (@data) {
            $record =~ s/\0/,/g;
            print $record, "\n";
        }
    }
```

ADVANTAGES

Here are the advantages of Sprite over mSQL by David Hughes available on the Net:

Allows for column names to be specified in the update command:

Perl's Regular Expressions allows for powerful pattern matching

The database is stored as text. Very Important! Information can be added/modified/removed with a text editor.

Can add/delete columns quickly and easily

DISADVANTAGES

Here are the disadvantages of Sprite compared to mSQL:

Speed . No where close to mSQL! Sprite was designed to be used to manipulate very small databases (~1000-2000 records).

Does not have the ability to "join" multiple tables (databases) during a search operation. This will be added soon!

RESTRICTIONS

1 If a value for a field contains the comma (,) character or the field delimiter, then you need to quote the value. Here is an example:

```
insert into $database
(One, Two)
values
('$some_value', $two)
```

The information in the variable $some_value *might* contain the delimiter, so it is quoted -- you can use either the single quote (') or the double quote (").

2 All single quotes and double quotes within a value must be escaped. Looking back at the previous example, if you think the variable $some_value contains quotes, do the following:

```
$some_value =~ s/(['"])/\\$1/g;
```

3 If a field's value contains a newline character, you need to convert the newline to some other character (or string):

```
$some_value =~ s/\n/<BR>/g;
```

4 If you want to search a field by using a regular expression:

```
select * from $database
    where (Player =~ /Bird/i)
```

the only delimiter you are allowed is the standard one (i.e /../). You *cannot* use any other delimeter:

```
select * from $database
    where (Player =~ m|Bird|i)
```

5 Field names can only be made up of the following characters:

"A-Z", "a-z", and "_"

In other words,

```
[A-Za-z_]
```

6 If your update value contains parentheses, you need to escape them:

```
$rdb->sql (<<End_of_Query);

 update my_db
     set Phone = ('\\(111\\) 222 3333')
     where (Name = /Gundavaram\$/i)

End_of_Query
```

Notice how the "$" (matches end of line) is escaped as well!

SEE ALSO

RDB (available at the Metronet Perl archive)

REVISION HISTORY

v3.1 - June 18, 1996
Added the following features:

+ As of this version, Sprite allows you to update multiple fields with a single *update* command. See the *Supported SQL Commands* section above.

+ You can execute your scripts with the following:

```
#!/usr/local/bin/perl5 -wT

use strict;
```

Sprite no longer generates the "Use of uninitialized value..." errors.

+ For records that don't contain quotes or escaped strings, Perl's split is used to dramatically speed up database loading.

+ The *set_os* function will allow you to set the operating system that you're using.

+ Added a "require 5.002" as Sprite fails on versions of Perl older than 5.002 with the following error:

    "Bad *free*() ignored at Sprite.pm..."

  Fixed the following bugs:

+ If you call the *close* method with a database as an argument *without* opening a database first, Sprite will warn you as opposed to wiping out the database, as was the case in earlier versions of Sprite.

+ Sprite no longer chops off the trailing "0" on records.

+ The *drop column* works as it should.

+ You can safely escape parentheses in the *update* command.

+ Extra spaces between field names in the *select* command and values in the *update* command no longer cause fatal errors.

+ In earlier versions of Sprite, if you opened two databases that were located in different directories, *but* with the same name, Sprite incorrectly assumed that it was the same database. As a result, the second database would never be loaded.

+ Can be used on the Mac OS and Windows NT.

v3.01 - March 5, 1996
    Fixed a bug in *parse_expression* subroutine so that it recognizes the "_" character as valid in field names.

v3.0 - Febraury 20, 1996
    Totally re-wrote parser; works reasonably well even in the worst case scenarios.

v2.0 - November 23, 1995
    Fixed *numerous* errors in parsing, and added pod style documentation.

v1.5 - September 10, 1995
    Created Perl 5 module instead of a command-line interface.

v1.0 - September 7, 1995
    Initial Release

ACKNOWLEDGEMENTS

I would like to thank the following for finding bugs and offering suggestions:

Dave Moore (dmoore@videoactv.com)

Shane Hutchins (hutchins@ctron.com)

Josh Hochman (josh@bcdinc.com)

Barry Harrison (barryh@topnet.net)

Lisa Farley (lfarley@segue.com)

Loyd Gore (lgore@ascd.org)

Tanju Cataltepe (tanju@netlabs.net)

COPYRIGHT INFORMATION

Copyright (c) 1995, 1996 by Shishir Gundavaram
All Rights Reserved

Permission to use, copy, and distribute is hereby granted, providing that the above copyright notice and this permission appear in all copies and in supporting documentation.

Employee Database

Welcome to the CGI Corporations's Employee Search Form. You can use this to find information about one of our employee. Enter as much information as possible to narrow down the search.

Last Name:	
First Name:	
Job Title:	
Department:	
EMail Address:	
Phone Number:	

Submit the search | Clear all fields

Listing of db_phone.cgi

Note:
This code is copyrighted, see *CGI Programming on the WWW* for more information. I have indicated the changes I made to get the script to work on my ISP in red (light font if printed). I've also add a few comments.

```
#!/usr/local/bin/perl

# add my perl library path to the @INC array.

BEGIN { unshift(@INC, "/home/gasster/perllib"); }

# load the Sprite module for parse SQL commands on our DB
use Sprite;

# load the standard set of functions to parse HTML form data

require "common.pl";


$webmaster = "gasster\@pacificnet\.net";
$query = undef;

# parse the form data

&parse_form_data(*FORM);

# define the database fields
$fields = '(Last|First|Job_Title|Department|EMail|Phone)';

# extract a query string from the form input data
# this query string will be sent to Sprite which
# will access the phone database

foreach $key (keys %FORM) {
    if ( ($key !~ /\b$fields\b/o) || ($FORM{$key} =~ /[^\w\-\(\) ]/) ) {
        &return_error (500, "CGI Corporation Employee Database Error",
                "Invalid Information in Form.");
    } else {
        if ($FORM{$key}) {
                $FORM{$key} =~ s/(\W)/\\$1/g;

            $query = join (" and ", $query,
                "($key =~ /$FORM{$key}/i)");
        }
    }
}

if ($query) {
    $query =~ s/^ and //;
} else {
    &return_error (500, "CGI Corporation Employee Database Error",
                "No query was entered.");
```

```
}

# call the Sprite module to access
# phone.db, the phone book database
# and parse the current query
#

$rdb = new Sprite ();
$rdb->set_delimiter ('Read', ',');

@data = $rdb->sql (<<End_of_Query);
    select * from phone.db
    where $query
End_of_Query

$status = shift (@data);
$no_elements = scalar (@data);

# if error let user know else send them the formatted
# query results

if (!$status) {
    &return_error (500, "CGI Corporation Employee Database Error",
                "Sprite Database Error!");
} elsif (!$no_elements) {
    &return_error (500, "CGI Corporation Employee Database Error",
                            "The record you specified does not exist.");
} else {
    print <<End_of_HTML;
Content-type: text/html

<HTML>
<HEAD><TITLE>CGI Corporation Employee Directory</TITLE></HEAD>
<BODY>
<H1>CGI Corporation Employee Directory</H1>
<HR><PRE>

End_of_HTML

    $~ = "HEADING";
    write;

    $~ = "EACH_ENTRY";

    foreach (@data) {


#  I don't think this line is needed
#  so I commented it out and everything works fine
#
#           s/([^\w\s\0])/sprintf ("&#%d;", ord ($1))/ge;

        ($last, $first, $job, $department, $email, $phone) =
            split (/\0/, $_, 6);
        write;
    }

    print "</PRE>", "\n";
    print "<HR>";
        print "</BODY></HTML>", "\n";
}
```

```
$rdb->close ();

# done and exit

exit (0);

# use the Perl report formatting syntax
# to output search results
# and I bet you thought nobody ever used
# this stuff!!


format HEADING =
Last       First      Job Title      Department   EMail       Phone
----       -----      ---------      ----------   -----       -----
.

format EACH_ENTRY =
@<<<<<<<<  @<<<<<<<<  @<<<<<<<<<<<<  @<<<<<<<<<<  @<<<<<<<<<  @<<<<<<<<<<<<<<
$last,     $first,    $job,          $department, $email,     $phone
.
```

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

Listing of phone.db

```
Last,First,Job_Title,Department,EMail,Phone
Supra,John,System Operator,Systems,jsupra,(617) 555-1578
Painton,Todd,Network Engineer,Systems,tpainton,(617) 555-6530
Martin,Robert,Sales Representative,Sales,martinr,(617) 555-7406
Levine,Julia,Administrative Assistant,Administration,julia,(617) 555-3056
Keenan,Jeff,Manager,Software,jeffk,(617) 555-7769
Nets,Laurie,Group Leader,Development,lnets,(617) 555-9962
```

LECTURE 11: Gateways to Internet Information Servers

Prelude: Review material on sockets.

Perl Socket Library from *CGI Programming on the WWW*

What are Internet Information Servers?

Network News (USENET News)
A set of newsgroups accessible on the internet via the NNTP (Network News Transfer Protocol).

Archie
A system that allows you to search indexes to locate files on public servers.

Gopher
The Internet Gopher allows you to browse for resources on the Internet using menu based interfaces. You access text based information and read it on-line; download binary files and even Telnet to other servers or gateways (e.g., University of California Library servers).

WAIS (Wide Area Information Service)
WAIS is a distributed text-searching system available on the Internet and through the WWW. WAIS is based on a standard that allows one computer to ask another to perform a search. In each case a WAIS library must be created for these searches.

Accessing IIS using Perl

Archie Access Example

The Perl script archie.pl discussed in the textbook provides a typical example of a CGI gateway to one of the IIS.

The basic functionality of this CGI script (`archie.pl`) is as follows:

- Call script using `GET` method; Archie Gateway Form is returned to the user by the script.
  View the Archie Gateway form.
- Fill out form, submit using `POST` method. Script parses form inputs and pipes query to Archie server.
- Results of archie query are wrapped in html and sent back to the user.
  View search results

Checking URLs

The textbook discusses a Perl subroutine that can be used to check the validity of a URL. There are several requirements to be able to use this script, primarily having to do with accessing the sockets.pl library (see handouts and textbook).

Basically, the concept behind this subroutine is that we can open a TCP/IP socket to the specified web server and send it a request. The HTTP responds header returned by the server is check for the text 200 (meaning success), else an error or failure status is returned by this subroutine.

Note that this routine does not handle other errors with any detail. Basically, if the connection retrieves a success flag of 200 then all is well, otherwise flag an error (return FALSE). You could modify this routine to handle other options, such as unable to make a connection or name not in the local DNS, more gracefully and providing more information. Try it out.

```
require "socket_lib.pl";

sub check_url
{
        local ($url) = @_;
        local ($current_host, $host, $service, $file, $first_line);

        if (($host, $service, $file) =
                ($url =~ m|http://([^/:]+):{0,1}(\d*)(\S*)$|)) {

                chop ($current_host = `/bin/hostname`);
                $host = $current_host if ($host eq "localhost");

                $service = "http" unless ($service);
                $file = "/"       unless ($file);

                &open_connection (HTTP, $host, $service) || return (0);

                print HTTP "HEAD $file HTTP/1.0", "\n\n";

                chop ($first_line = <HTTP>);

                if ($first_line =~ /200/) {
                        return (1);
                } else {
                        return (0);
                }

                close (HTTP);
        } else {
                return (0);
        }
```

[TITLE PAGE | TABLE OF CONTENTS | LECTURE 11]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

```
#!/usr/bin/perl

BEGIN
{
     unshift(@INC, "/mnt2/guide2/muon/perllib");
}


require "common.pl";

$webmaster = "Sam Gasster (muon\@muon\.com)";

#where is the archie program on your server?
$archie = "/usr/contrib/bin/archie";

$error = "CGI Archie Gateway Error";
$default_server = "archie.rutgers.edu";
$timeout_value = 180;

%servers = ( 'ANS Net (New York, USA)',         'archie.ans.net',
             'Australia',                       'archie.au',
             'Canada',                          'archie.mcgill.ca',
             'Finland/Mainland Europe',         'archie.funet.fi',
             'Germany',                         'archie.th-darmstadt.de',
             'Great Britain/Ireland',           'archie.doc.ac.ac.uk',
             'Internic Net (New York, USA)',    'ds.internic.net',
             'Israel',                          'archie.ac.il',
             'Japan',                           'archie.wide.ad.jp',
             'Korea',                           'archie.kr',
             'New Zealand',                     'archie.nz',
             'Rutgers University (NJ, USA)',    'archie.rutgers.edu',
             'Spain',                           'archie.rediris.es',
             'Sweden',                          'archie.luth.se',
             'SURANet (Maryland, USA)',         'archie.sura.net',
             'Switzerland',                     'archie.switch.ch',
             'Taiwan',                          'archie.ncu.edu.tw',
             'University of Nebraksa (USA)',    'archie.unl.edu' );

$request_method = $ENV{'REQUEST_METHOD'};

if ($request_method eq "GET") {
    &display_form ();

} elsif ($request_method eq "POST") {
    &parse_form_data (*FORM);
    $command = &parse_archie_fields ();

    $SIG{'ALRM'} = "time_to_exit";
    alarm ($timeout_value);
    open (ARCHIE, "$archie $command |");

    $first_line = <ARCHIE>;

    if ($first_line =~ /(failed|Usage|WARNING|Timed)/) {
        &return_error (500, $error,
            "The archie client encountered a bad request.");
    } elsif ($first_line =~ /No [Mm]atches/) {
        &return_error (500, $error,
        "There were no matches for <B>$FORM{'query'}</B>.");
    }

    print "Content-type: text/html", "\n\n";
```



```
    print "<HTML>", "\n";
    print "<HEAD><TITLE>", "CGI Archie Gateway", "</TITLE></HEAD>", "\n";
    print "<BODY>", "\n";
    print "<H1>", "Archie search for: ", $FORM{'query'}, "</H1>", "\n";
    print "<HR>", "<PRE>", "\n";

    while (<ARCHIE>) {
        if ( ($host) = /^Host (\S+)$/ ) {
            $host_url = join ("", "ftp://", $host);
            s|$host|<A HREF="$host_url">$host</A>|;

            <ARCHIE>;

        } elsif (/^\s+Location:\s+(\S+)$/) {
            $location = $1;
            s|$location|<A HREF="${host_url}${location}">$location</A>|;
        } elsif ( ($type, $file) = /^\s+(DIRECTORY|FILE).*\s+(\S+)/) {
            s|$type|<I>$type</I>|;
            s|$file|<A HREF="${host_url}${location}/${file}">$file</A>|;
        } elsif (/^\s*$/) {
            print "<HR>";
        }

        print;
    }

    $SIG{'ALRM'} = "DEFAULT";
    close (ARCHIE);

    print "</PRE>";
    print "</BODY></HTML>", "\n";
} else {
    &return_error (500, $error, "Server uses unspecified method");
}

exit (0);

sub time_to_exit
{
    close (ARCHIE);

    &return_error (500, $error,
        "The search was terminated after $timeout_value seconds.");
}

sub parse_archie_fields
{
    local ($query, $server, $type, $address, $status, $options);

    $status = 1;

    $query = $FORM{'query'};
    $server = $FORM{'server'};
    $type = $FORM{'type'};

    if ($query !~ /^\w+$/) {
        &return_error (500, $error,
            "Search query contains invalid characters.");
    } else {
        foreach $address (keys %servers) {
            if ($server eq $address) {
                $server = $servers{$address};
```

```
                $status = 0;
            }
        }

        if ($status) {
            &return_error (500, $error,
                "Please select a valid archie host.");
        } else {
            if ($type eq "cs_sub") {
                $type = "-c";
            } elsif ($type eq "ci_sub") {
                $type = "-s";
            } else {
                $type = "-e";
            }

            $options = "-h $server $type $query";

            return ($options);
        }
    }
}

sub display_form
{
    local ($archie);

    print <<End_of_Archie_One;
Content-type: text/html

<HTML>
<HEAD><TITLE>Gateway to Internet Information Servers</TITLE></HEAD>
<BODY>
<H1>CGI Archie Gateway</H1>
<HR>
<FORM ACTION="/cgibin/archie.cgi" METHOD="POST">
Please enter a string to search from: <BR>
<INPUT TYPE="text" NAME="query" SIZE=40>
<P>
What archie server would you like to use (<B>please</B>, be considerate
and use the one that is closest to you): <BR>
<SELECT NAME="server" SIZE=1>

End_of_Archie_One

    foreach $archie (sort keys %servers) {
        if ($servers{$archie} eq $default_server) {
            print "<OPTION SELECTED>", $archie, "\n";
        } else {
            print "<OPTION>", $archie, "\n";
        }
    }

    print <<End_of_Archie_Two;
</SELECT>
<P>
Please select a type of search to perform: <BR>
<INPUT TYPE="radio" NAME="type" VALUE="exact" CHECKED>Exact<BR>
<INPUT TYPE="radio" NAME="type" VALUE="ci_sub">Case Insensitive Substring<BR>
<INPUT TYPE="radio" NAME="type" VALUE="cs_sub">Case Sensitive Substring<BR>
<P>
<INPUT TYPE="submit" VALUE="Start Archie Search!">
```



```
<INPUT TYPE="reset"  VALUE="Clear the form">
</FORM>
<HR>
</BODY>
</HTML>

End_of_Archie_Two
}
```

CGI Archie Gateway

Please enter a string to search from:

What archie server would you like to use (**please**, be considerate and use the one that is closest to you):

Rutgers University (NJ, USA)

Please select a type of search to perform:

- (•) Exact
- ( ) Case Insensitive Substring
- ( ) Case Sensitive Substring

Start Archie Search! Clear the form

Archie search for: Perl

Host fpms.fpms.ac.be
Location: /pub
DIRECTORY drwxr-xr-x 6144 Aug 7 01:16 Perl

Host pascal.zedat.fu-berlin.de
Location: /pc/win95/network/winsock-1
DIRECTORY drwxr-xr-x 72 Jun 14 17:31 Perl

Host irz301.inf.tu-dresden.de
Location: /.4.1/vol4/atari/Programming/Languages
DIRECTORY drwxr-xr-x 512 May 2 08:33 Perl

Host rcs1.urz.tu-dresden.de
Location: /pub/soft/win95/mirror/winsock-1
DIRECTORY drwxr-xr-x 8192 Jun 23 05:26 Perl

Host ftp.urz.uni-heidelberg.de
Location: /pub/linux/mirror.redhat/contrib/by-group/Development/Libraries
DIRECTORY drwxr-xr-x 512 Jul 8 17:40 Perl
Location: /pub/net/winsock/winsock-1/Windows95
DIRECTORY drwxr-xr-x 1024 Jul 31 02:30 Perl
Location: /pub/net/winsock/winsock-1/WindowsNT
DIRECTORY drwxr-xr-x 512 Jul 31 02:26 Perl

Host warum.uni-mannheim.de
Location: /systems/windows/win32/win95-winsock
DIRECTORY drwxr-xr-x 1536 Jul 31 02:00 Perl
Location: /systems/windows/win32/winnt-winsock
DIRECTORY drwxr-xr-x 512 Jun 20 03:09 Perl

Host rzbsdi01.uni-trier.de
Location: /pub/pc/mirrors/papa.indstate.edu/Windows95
DIRECTORY drwxr-xr-x 1536 Aug 12 22:00 Perl
Location: /pub/pc/mirrors/papa.indstate.edu/WindowsNT
DIRECTORY drwxr-xr-x 512 Jun 29 00:08 Perl
Location: /pub/unix/systems/linux/sunsite/distributions/redhat/contrib/by-group/Devel
DIRECTORY drwxr-xr-x 512 Jul 19 10:41 Perl

Host sunsite.auc.dk
Location: /mirrors/ftp.redhat.com/pub/contrib/by-group/Development/Libraries
FILE -rwxrwxrwx 39 Jul 27 08:59 Perl

Host scotch.csua.berkeley.edu
Location: /pub/Help-sessions
FILE -rw-rw-r-- 7677 Apr 17 1995 Perl

```
#!/usr/local/bin/perl

##++
##      Socket Library v2.0
##      Last modified: November 23, 1995
##
##      Copyright (c) 1995 by Shishir Gundavaram
##      All Rights Reserved
##
##      E-Mail: shishir@ora.com
##
##      Permission  to  use,  copy, and distribute is hereby granted,
##      providing that the above copyright notice and this permission
##      appear in all copies and in supporting documentation.
##--

require "sys/socket.ph";

sub open_connection
{
    local ($socket, $remote_host, $remote_service) = @_;
    local ($current_host, $current_address, $remote_address,
           $remote_port_number, $current_port, $remote_port, $protocol);

    $current_host = &get_current_host ();
    $current_address = &get_packed_address ($current_host);

    $remote_address = &get_packed_address ($remote_host);
    $remote_port_number = &get_port_number ($remote_service);

    $current_port = &create_socket_structure (0, $current_address);
    $remote_port  = &create_socket_structure ($remote_port_number,
                        $remote_address);

    $protocol = (getprotobyname ("tcp"))[2];

    socket  ($socket, &AF_INET, &SOCK_STREAM, $protocol) || return (0);
    bind    ($socket, $current_port)                     || return (0);
    connect ($socket, $remote_port)                      || return (0);

    &unbuffer_socket ($socket);

    return (1);
}

sub listen_to_port
{
    local ($socket, $service_or_number) = @_;
    local ($current_host, $current_address, $protocol,
           $specified_port_number, $specified_port);

    $current_host = &get_current_host ();
    $current_address = &get_packed_address ($current_host);

    $protocol = (getprotobyname ("tcp"))[2];

    if (defined ($service_or_number)) {
        $specified_port_number = &get_port_number ($service_or_number);
    } else {
        $specified_port_number = 0;
    }
```



```
    $specified_port = &create_socket_structure ($specified_port_number,
                          $current_address);

    socket  ($socket, &AF_INET, &SOCK_STREAM, $protocol) || return (0);
    bind    ($socket, $specified_port)                   || return (0);
    listen  ($socket, 5)                                 || return (0);

    unless (defined ($service_or_number)) {
        $specified_port_number = &get_socket_port ($socket);
    }

    return ($specified_port_number);
}

sub accept_connection
{
    local ($connection, $socket) = @_;
    local ($IP_number, $IP_name);

    accept ($connection, $socket) || return (0);

    &unbuffer_socket ($connection);

    $IP_number = &where_from ($connection);
    $IP_name = &IP_number_to_name ($IP_number);

    return ($IP_name, $IP_number);
}

sub strip_control_chars
{
    local ($input) = @_;

    $input =~ s/[\000-\037]//g;
    return ($input);
}

sub close_connection
{
    local ($connection) = @_;

    &flush_socket ($connection);
    close ($connection);
}

###############################################################################

sub get_current_host
{
    local ($host);

    unless ($host = `hostname`) {
        $host = `uname -n`;
    }

        chop ($host);

    return ($host);
}

sub create_socket_structure
{
```

```
    local ($port_number, $packed_address) = @_;
    local ($socket_template, $structure);

    $socket_template = "S n a4 x8";

    $structure =
         pack ($socket_template, &AF_INET, $port_number, $packed_address);

    return ($structure);
}

sub get_port_number
{
    local ($service_or_port) = @_;
    local ($port_number);

    if ($service_or_port !~ /^\d+$/) {
        $port_number =
             (getservbyname ($service_or_port, "tcp"))[2];
    } else {
        $port_number = $service_or_port;
    }

    return ($port_number);
}

sub get_socket_port
{
    local ($socket_handle) = @_;
    local ($socket_template, $socket_address, $port_number);

    $socket_template = "S n a4 x8";

    $socket_address = getsockname ($socket_handle);
    $port_number = (unpack ($socket_template, $socket_address))[1];

    return ($port_number);
}

sub unbuffer_socket
{
    local ($socket) = @_;
    local ($current_handle);

    $current_handle = select ($socket);
    $| = 1;
    select ($current_handle);
}

sub flush_socket
{
    local ($socket) = @_;
    local ($current_handle);

    ##++
    ##  Based on the code from the flush.pl library
    ##--

    $current_handle = select ($socket);
    $| = 1;
    print "";
    $| = 0;
```



```
    select ($current_handle);
}

sub get_packed_address
{
    local ($IP_name_or_number) = @_;
    local ($packed_address, @decimal_address);

    if ($IP_name_or_number =~ /^\d+\.{3}\d+$/) {
        @decimal_address = split (/\./, $IP_name_or_number);
        $packed_address = pack ("C4", @decimal_address);
    } else {
        $packed_address = (gethostbyname ($IP_name_or_number))[4];
    }

    return ($packed_address);
}

sub packed_address_to_IP_number
{
    local ($packed_address) = @_;
    local (@decimal_address, $IP_number);

    @decimal_address = unpack ("C4", $packed_address);
    $IP_number = join (".", @decimal_address);

    return ($IP_number);
}

sub where_from
{
    local ($socket) = @_;
    local ($socket_template, $internet_address, $packed_address,
           $IP_number);

    $socket_template = "S n a4 x8";
    $internet_address = getpeername ($socket);
    $packed_address = (unpack ($socket_template, $internet_address))[2];
    $IP_number = &packed_address_to_IP_number ($packed_address);

    return ($IP_number);
}

sub IP_number_to_name
{
    local ($IP_number) = @_;
    local ($packed_IP_address, $IP_name);

    $packed_IP_address = &get_packed_address ($IP_number);
    ($IP_name) = gethostbyaddr ($packed_IP_address, 2);

    return ($IP_name);
}

sub IP_name_to_number
{
    local ($IP_name) = @_;
    local ($packed_IP_address, $IP_number);

    $packed_IP_address = &get_packed_address ($IP_name);
    $IP_number = &packed_address_to_IP_number ($packed_IP_address);
```

```
    return ($IP_number);
}

1;
```

LECTURE 12: ADVANCED CGI APPLICATIONS

READING ASSIGNMENT:
CGI Programming on the WWW: Ch. 11

HOMEWORK ASSIGNMENT:
NONE! Your projects are due today!

TOPICS

- Advanced and Creative Applications
- CGI and Perl resources on the Internet.
- Discussion of Class projects.

[TITLE PAGE | TABLE OF CONTENTS]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

HOMEWORK ASSIGNMENT NO. 5

1. What is wrong with the following program?

```
#!/usr/local/bin/perl

for ($count =1; $count<= 10; $count++) {
        &print_ten( $count );
}

sub print_ten{
        local ($mulitplier) = @_;
        for ($count =1; $count<= 10; $count++) {
                $printval = $multiplier * 10 + $count;
                print ("$printval\n");
        }
```

2. Write a subrountine that will return the following HTML code:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<HTML>
<HEAD>
        <TITLE>PAGE TITLE</TITLE>
</HEAD>
<BODY>
        <H1>MY FAVORITE HEADING</H1>
</BODY>
</HTML>
```

The text *PAGE TITLE* and *MY FAVORITE HEADING* are defined by arguements passed to the subroutine. Thus you should be able to call the subroutine in the following manner:

```
$pagetitle = "PAGE TITLE";
$heading1 = "MY FAVORITE HEADING";

$mypage = &gethtml($pagetitle, $heading1);
```

3. XXXX

4. XXXX

HOMEWORK ASSIGNMENT NO. 6

Working With Patterns and Regular Expressions

1. In this exercise you will write a perl script that will perform the following tasks:
   - Read in a whole number with up to 36 digits and print out the English equivalent
   - Remove any non-digit characters, such as commas, tabs or blanks
   - Allow for the number zero as: 0, 00, 000, ..., etc.
   - Test your program and show the results. Does it handle the following numbers:
     000
     123456789
     999, 89, 7654321
     (note there is a tab between the second comma and the 7.

To get you started I have provided the following code fragment:

```
#!/usr/local/bin/perl
#
#       Perl script to convert numeric input
#       to English equivalent.
#
#       define the strings used in printing

@digitword = ("", "one", "two", "three", "four", "five",
        "six", "seven", "eight", "nine",);

@digit10word = ("","ten", "twenty", "thirty", "forty", "fifty",
        "sixty", "seventy", "eighty", "ninety");

@teenword = ("ten", "eleven", "twelve", "thirteen", "fourteen", "fifte
        "sixteen", "seventeen", "eighteen", "nineteen");

@groupword = ("", "thousand", "million", "billion", "trillion", "quadr
        "quintillion", "sextillion", "septillion", "octillion",
        "novillion","decillion" );

#
#       YOU FILL IN THE REST!
#


#
#       end of script
#
```

```
__END__
```

2. What does the following perl script do? Show examples of input and output.

```
#!/usr/local/bin/perl

@input = <STDIN>;
$count = 0;
while ($input[$count] ne "") {
        $input[$count] =~ s/^[ \t]+//;
        $input[$count] =~ s/[ \t]+\n$/\n/;
        $input[$count] =~ s/[ \t]+/ /g;
        $count++;
}
print ("Formatted text:\n");
print (@input);
#
#       end of script
#

__END__
```

3. Write a perl script to do blah.

4. Write patterns that match the following:
   - Three or more lower case letters (a-z).
   - Either the string four or the number 4.
   - Any letter followed by a vowel, followed by the same letter (e.g., sas).

[TITLE PAGE| TABLE OF CONTENTS | HOMEWORK]

© 1996 Samuel D. Gasster, ALL RIGHTS RESERVED.

COPYRIGHT 1995 – 2008 ©
SAMUEL GASSTER
J. C. KERN