GOALS

- *At the completion of the course*

- We should

  * Understand basic functionality of perl
  * Understand syntax and semantics of Perl
  * Master perl by writing half a dozen scripts
  * Know how to use perl for system administration
  * Understand Sybperl routines and database manipulation

AGENDA

- *Day 1 Morning: Introduction and Data Types*
    * Chapters 1-5
- *Day 1 Afternoon: Control Structures, Pattern Matching*
    * Chapters 6-9
- *Day 2 Morning: Program Development*
    * Chaptrs 10-12
- *Day 2 Afternoon: SybPerl*

INTRODUCTIONS

- *Audience: Names, Background, Interests*
  - * What are your objectives ?
  - * Please Write on a sheet of paper.
- *Name:Shashi Shekhar*
  - * CS faculty at University of Minnesota
    - Email: shekhar@cs.umn.edu
  - * Graduate Education at U.C. Berkeley
    - Ph. D. (C. S.), 1989., M. S. (Bus. Adm.), 1989.
  - * Professional Activities
    - Review Panels at N.S.F. (DBES), N.A.S.A. (EOS DB)
    - Member IEEE, ACM, AAAI
  - * Research Interests:
    - Data and Knowledge Eng. (GIS, Neural Networks, Search)
    - Application in Finance (Bond Rating), Transportation etc.
    - List of (>70) Publications available
  - * Teaching Course and Seminars in
    - Databases, Software Eng., Neural Networks, Programming Languages

INTRODUCTIONS

- *Course Structure*
  - * Presentation and Discussion
  - * Exercises: Active Group Learning
  - * Course Evaluation: one minute papers
  - * Celeberate achievements

Perl5 : An Introduction

- *Supplement with Llama Book*
  - Learning Perl, R. L. Schwartz, O’Reilly & Associates

TOPICS IN Perl5

- *Introduction*
  - 1: What is Perl5?
  - 2: A Basic Perl5 Program
- *Data Types*
  - 3: Scalar Variables
  - 4: Array Variables
  - 5: Associative Arrays
- *Control Structures, Pattern Matching*
  - 6: File Handling
  - 7: Loops and Conditions
  - 8: String Matching
  - 9: Substitution and Translation
  - 10: Split
- *Program Development*
  - 11. Subroutines
  - 12. Other features (e.g. Report Writer)
- *Perspective*

CH#1: WHAT IS Perl5?

- *Practical extraction and reporting language*
  - Combining elements of C-shell, awk and other utilities
  - i.e. sh + awk + sed + tr + grep + ...
- *Perl5 is Free, NOT "public domain"*
  - README file that explains your license
- *Two publicly available books; comp.lang.perl*
- *better than shell script programming and awk or sed*
  - No shell initiation overhead
  - Can read and write binary data files
  - Can have many files for input or output at one time
  - It has a report writer
  - Has extended regular expressions
  - It has both linear and associative arrays
  - Has powerful defaults that simplify programming
  - It can process very large files without
    - record size limits

CH#1: WHAT IS Perl5?

- *Is better than C, C++, or Fortran*
  - No visible compile or link stages
  - Rich set of character string manipulation
  - Include X-windows features, tkperl, access to databases
- *Perl5 has some down points*
  - 1.7 times as long to execute as an equivalent C program
  - "gotcha’s" would confuse non-programmers

```
$num = 010;
print "The number is $num. \n";
```

- will print
- The number is 8

  - Honor system, rather than strict enforcement of rules

CH#2: A BASIC PROGRAM

- *A basic Perl5 program*

```
#!/usr/local/bin/perl
#
# A Program to do the obvious
#
print 'Hello world.';  #Print a message
```

- *Discussion*
  * The first line identifies the interpreter
  * Comments and statements
  * Simple printing
- *Q? How does one determine the path name for perl interpreter?*
  * On Unix
    which perl
    which perl5
    Solaris - #!/opt/bin/perl

CH#2: RUNNING THE PROGRAM

- *Type in the example program*
- *chmod u+x progname*
- *Type any of the following*
  - perl progname
  - ./progname
  - progname
- *Run the program with warnings*
  perl -w progname
- *Run the program with a debugger*
  perl -d progname
- *Perl first compiles and then it executes*

CH#3: SCALAR VARIABLES

- *Most basic kind is scalar variables, holds*

- both strings and numbers, completely interchangable

  * Example

    $priority = 9 ;

    $priority = 'high';

- *Variable names consist of numbers, letters and underscores*

- not start with numbers

  * $_ is special

  * Case sensitive, $a and $A are different

- *Operations and assignment*

  * Usual C arithmetic operators

    $a = 5 % 2; # Remainder of 5 divided by 2

    ++$a; # Increment $a and then return it

  * Strings

    $a = $b . $c; #Concatenate $b and $c

    $a = $b x $c; #$b repeated $c times

CH#3: SCALAR VARIABLES

- *Type in, and run the following perl script. Explain its output.*

```
#!/usr/local/bin/perl
$a = ’apples’;
$b = ’pears’;
print ’Concatenation : ’, $a.’ and ’.$b,  "\n" ;
print ’Single quotes : ’, ’$a and $b’ , "\n" ;
print ’Double quotes : ’, "$a and $b" , "\n";
```

- *Output*
  - Concatenation prints : apples and pears
  - Single quotes prints literally : $a and $b
  - Double quotes prints : apples and pears
- *Interpolation*
  - Double quotes force interpolation of any codes
  - i.e. substitute values of variables inside strings in " ... "

Rest of Scalars

- *Review chapter 2, pp 35-50 or 31-46 (Llama Book)*
- *Special Characters : Table 2-1 (pp. 38 or 34)*
  * newline, tabs, ...
  * capitalize first character or entire string
- *Comparison Operators : Table 2-2 (pp. 41 or 37)*
- *Operators : Table 2-3 (pp. 42-3 or 38-9)*
  * Q? Compare numeric operator symbols in Perl and those in C.
  - Find an operator which uses different symbols in Perl and in C.
- *Reading a scalar from standard input (pp. 49 or 45)*

```
$value = <STDIN> ;
chop($value); # chop operator on pp. 47 or 43
```

  * %Why does chop operator usually follow <STDIN> ?
- *Uninitialized values, error etc. : undef (pp. 51 or 46)*

SCALARS - CLASS EXERCISE

- *Ex. Write a Perl script which reads in a name from standard*
- input and prints greetings.

  * Example interaction

    >hello.pl

    May I know your name please ?:   Mary

    Hello Mary !

    Nice to See you.

    Bye Mary.

    ----

  * EXTRA: Ensure proper printing of the name.
  - The first character should be capitalized
  - and other characters should be in lower case.

  * EXTRA: Ensure that the number of dashes printed

  * is 4 + (number of characters in the name).

EXTRA EXERCISES

- *EXTRA Ex. 2-4, pp. 51 or 47 (Llama Book)*
- *EXTRA Ex. on Implicit conversion between numbers and strings.*

  * Type in, and run the following script and explain its output.

```
#!/usr/local/bin/perl
$num = "002" ; $num = $num + 1 ; print "$num \n" ;
$num = "002" ; $num++ ; print "$num \n" ;
$num = "02/05/1997" ; $num = $num + 1 ; print "$num \n" ;
$num = "02/05/1997" ; $num++ ; print "$num \n" ;
```

CH#4: ARRAY VARIABLES

- *Syntax: Prefixed by an @ symbol*

  @food = ("apples", "pears", "eels");

  @music = ("whistle", "flute");

- *Accessed by using indices starting from 0*

  * $food [2] returns eels.

  * Notice that the @ has changed to a $

- *Array assignments*

  @moremusic = ("organ", @music, "harp");

  @moremusic = ("organ", "whistle", "flute", "harp");

CH#4: ARRAY OPERATORS

- *Find the index of the last element of a list "$#food"*

  * Adding an element at the end

```
@food = ("apples", "pears", "eels");
$i = $#food ;  $food[ $i + 1 ] = "eggs" ;
```

- *Push : A simpler way of adding elements*

```
push(@food, "eggs");
push(@food, "eggs", "cookies");
push(@food, @morefood);
```

  * Push function returns length of the new list

- *Pop : Remove the last item and return it*

```
@food = ("apples", "pears", "eels");
$grub = pop(@food);  # Now $grub = "eels"
```

- *Other useful operators*

  * shift(), unshift() - pp. 59 or 54
  * sort() - pp. 60 or 55
  * reverse() - pp. 60 or 54
  * @manyLines = <STDIN> ; #pp. 61 or 56
  * chop(@manyLines) ; #pp. 60 or55

CH#4: ARRAY - Printing and Type Conversion Issues

- *Assign an array to a scalar variable*

  $f = @food;   #assigns the length of @food

  $f = "@food"; #turns the list into a string

- with a space between each element

- *Ex. Type in, and run the following script. Explain its output.*

```
#!/usr/local/bin/perl
@food = ( "pancakes", "eggs", "fruits" );
print @food."" , "\n" ;  # In a scalar context
print @food , "\n" ;  # By itself
print "@food" , "\n" ;  # embedded in double quotes
$" = ’ : ’ ;
print "@food" , "\n" ;  # again embedded in double quotes
```

CH#4: ARRAY - multiple assignments

- *Multiple assignments to scalar variables*

```
($a, $b) = ($c, $d); #Same as $a=$c; $b=$d
($a, $b) = @food; #$a and $b are the first two
                  #items of @food
($a, @somefood) = @food; #$a is first item of @food
                         #@somefood is a list of
                         #the others
(@somefood, $a) = @food; #@somefood is @food and
                         # $a is undefined. Why?
```

- *Arrays are greedy*

CH#4: ARRAY - CLASS EXERCISES

- *Review chapter 3, pp 53-62 or 48-57 (Llama Book)*
- *Ex. 3-1 / pp. 63 or 57*
- *Ex. 3-2 / pp. 63 or 57*
- *EXTRA Ex. Which behaviour from a. ... d.*

- following code fragment exhibits?

```
unshift(@myarray, pop(@myarray));
```

  * a. The first element is moved to the end
  * b. Both the first an last elements are removed.
  * c. The last element is moved to the beginning
  * d. The first and last elements are swapped.

- *EXTRA Write code fragments using push, pop, shift, unshift*

- operators to simulate behaviours a. ... d.

CH# 5: FILE HANDLING

- *The basic perl program same as UNIX cat command*

```
#!/usr/local/bin/perl
#script file1.pl
# Open the password file, read it in,
# print it, and close it.
$file = '/etc/passwd'; # OR /var/yp/passwd
open(INFO, $file);     # Open the file
@lines = <INFO>;       # Read it into an array
close(INFO);           # Close the file
print @lines;          # Print the array
```

- *Open function*

  * First parameter is the filehandle

  * Second parameter is an expression denoting
  - the filename
  - To force shell expansion use angled brackets

CH# 5: FILE HANDLING

- *Print to a file already opened for output*
    - * print statement with an extra parameter
    - * print INFO "This line goes to the file.\n";

- *Specify a file for output, for appending, for input*

```
Open(INFO, $file);     # Open for input
Open(INFO, ">$file");  # Open for output
Open(INFO, ">>$file"); # Open for appending
Open(INFO, "<$file");  # Also open for input
```

- *Open the standard input (usually the keyboard)*

\- standard output (usually the screen)

```
Open(INFO, ’-’);   # Open standard input
Open(INFO, ’>-’);  # Open standard output
```

- *Formatted Print* - *like C language (pp. 82 or 74)*

```
printf "%.3d %s", $i, $line ;
```

CH# 5: FILE HANDLING

- *Diamond Operator* - *<> (pp. 82 or 73)*
  - * Reads the sequence of files given on command line.
  - * Consider myscript.pl given below:

```
#!/usr/local/bin/perl
# file myscript.pl
@a = <> ;
print @a;
```

  - * Following commands are equivalent in Unix:

```
myscript.pl file1 file2 file3
cat file1 file2 file3
```

CLASS EXERCISE

- *Review chapter 6, pp 79-82 or 72-75 (Llama Book)*

- *Ex. Write a perl script (file2.pl) which reads in a file*

- and prints each content with a # symbol at the beginning
- of each line.

  * Hint: Add one line to the above program (file1.pl).
  - Use the $" variable. DO not use loops.

- *EXTRA EX. 6-1/ PP. 82.*

- *EXTRA: What will the following code fragment do?*

```
print reverse <>;
```

- *EXTRA: Explain the output of the following script*

```
#!/usr/local/bin/perl
print( (2 + 3), ’hello’) ; print "\n" ;
print (2 + 3), ’hello’ ;
print "\n" ;
print 2 + 3, ’hello’ ;
print "\n" ;
```

CH# 6: CONTROL STRUCTURES

- *Overview: Similar to those in C, Pascal*
  - loop over a list - foreach
  - loop with tests - while, for, do-while, do-until
  - conditional - if elsif else

- *Foreach - go through each line of an array or other list*
  - Simple semantics, Robust w.r.t programming errors!

```
foreach $item (@food)   # Visit each element of array @food
{
    print "$item is yummy ! \n";
}
```

  - Revisiting last Exercise

```
open(INFO, '/etc/passwd');
chop( @lines = <INFO> );       # Read it into an array
close(INFO);
foreach $line (@lines) {
        print '#', $line , "\n" ; #Print # in front of each line!
}
```

CH# 6: CONTROL STRUCTURES - TESTS

- *Tests yield boolean value (True, False)*
  - Boolean type is simulated using scalars
  - False => 0, "", "000", undef, ...
  - True => 1, 2, -34, "abc", "false", ...

- *Boolean Operators to compare scalars*

```
$a == $b   # Is $a numerically equal to $b?
           # Beware: Don't use the = operator.
$a != $b   # Is $a numerically unequal to $b?
$a eq $b   # Is $a string-equal to $b?
$a ne $b   # Is $a string-unequal to $b?
```

- *Combining Booleans*

```
($a && $b)   # Is $a and $b true?
($a || $b)   # Is either $a or $b true?
!($a)        # is $a false?
```

CH# 6: CONTROL STRUCTURES - Loops with Tests

- *Loops with Tests: Pre-test, Post-test*
- *Pre-test: while condition { statements; }*
  * Syntax and semantics similar to C
  * Repeat a set of statements while condition is true
  * Statements not executed if condition is true at beginning
- *Example:*
  * Reads input and won't continue until correct password

```
#!/usr/local/bin/perl
print "Password? ";   # Ask for input
$a = <STDIN>;         # Get input
chop $a;              # Remove the newline at end
while ($a ne "fred")  # While input is wrong...
{
   print "sorry. Again? ";  # Ask again
   $a = <STDIN>;            # Get input again
   chop $a;                 # Chop off newline again
}
```

- *To test the opposite thing use the until*
  * until condition { statements; } ;

CH# 6: CONTROL STRUCTURES - Post test

- *Post-test: do { statements; } while condition ;*
  * Syntax and semantics similar to C
  * Repeat a set of statements while condition is true
  * Statements executed at least once
  - even if condition true at start

```
#!/usr/local/bin/perl
do
{
Print "Password? ";        # Ask for input
$a = <STDIN>;        # Get input
 chop $a;            # Chop off newline
}
while ($a ne "fred") ;  # Redo while wrong input
```

- *To test the opposite thing use the until*
  * do { statements; } until condition ;
  * Repeat a set of statements while condition is false

CH# 6: CONTROL STRUCTURES - for loop

- *"for" loop mimics C*

```
for (initialize; test; inc)
{
        statements ;
}
```

  * Has same semantics as

```
initialize;
while test {
        statements ;
        inc ;
}
```

- *The password example*

```
Print "Password? ";
for ( chop($a = <STDIN>); $a ne "fred" ; chop($a = <STDIN>) ) {
        Print "Password? ";
}
```

CH# 6 : CLASS EXERCISE

- *Review chapter 4, pp 65-70 or 58-65 (Llama Book)*

- *EX. Write a perl script (file3.pl) to read a file*

- one line at a time and print its content
- on standard output with a line number at the beginning.

  * You should get something like:

    1 root:oYpYXm/qRO6N2:0:0:Super-User:/:/bin/csh
    2 sysadm:*:0:0:System V Administration:/usr/admin:/bin/sh
    3 diag:*:0:996:Hardware Diagnostics:/usr/diags:/bin/csh
    etc

  * Hint: Reuse file2.pl as much as possible.

  * You may find it useful to use the foreach loop

```
foreach $line (@lines) {
...
}
```

  - Or the while loop

```
while ($line = <INFO>)
{
        ...
}
```

CH# 6 : EXTRA EXERCISES

- *EXTRA EX.: When you have done this see if you can alter*
- it so that line numbers are printed as 001, 002,
- ..., 009, 010, 011, 012, etc. To do this you
- should only need to change one line by inserting
- an extra four characters. Perl’s clever like that.

- *EXTRA: Ex. 4-3 / pp. 71 or 65*

- *EXTRA: Ex. 4-4 / pp. 71 or 65*

CH# 6 : CONDITIONALS

- *Two-alternative : if/then/else*

```
if ($a)
{
    print "The string is not empty \n";
}
else
{
    print "The string is empty \n";
}
```

- *Empty string is considered to be false*

- $a is for the string 0

- *Curly braces are mandatory!*

CH# 6 : CONDITIONALS

- *Multi- alternatives conditionals*

```
if (!$a)      # The ! is the not operator
{
     print "The string is empty \n";
}
elsif (length($a) == 1)  # If above fails, try this
{
     print "The string has one character \n";
}
elsif (length($a) == 2)  # If that fails, try this
{
     print "The string has two characters \n";
}
else                  # Now, everything has failed
{
     print "The string has lots of characters \n";
}
```

CH# 6 : CLASS EXERCISE

- *Review chapter 4, pp 65-70 or 58-65 (Llama Book)*

- *EX. Write a perl script script (file4.pl), which reads*
- a textfile and prints out all NON-BLANK lines
- with line number prefix. Blank lines are neither
- printed nor counted.

  * TESTCASE: Create a text file that contains some
  - text and a few blank lines.

  * HINTS: Reuse file3.pl as much as possible.
  - Change it to open a text file other than /etc/passwd.
  - Remember that when a line of the file is read in
  - it will still include a newline character at the end.

CH# 6 EXTRA EXERCISES

- *EXTRA: Write perl psuedo-code using a pre-test while loop*

\- to simulate

```
for( $i = 1; i < 10 ; $i++) {
        print $i , $i * $i, "\n" ;
}
```

- *EXTA: Ex. 9-1 / pp. 114 or 107.*

- *EXTRA: Read Chapter 9 and try the control structures discussed there.*

CH# 7 ASSOCIATIVE ARRAYS

- *Motivation: Consider the following applications:*

  * Identify the popular web-pages by listing
  - web-page URLs and frequency of access.

  * Identify the frequent users of a web-site by listing
  - email address and frequency of access.

  * Identify the login accounts under attack from hackers
  - by listing login and the frequency of failed passwords last month.

  * Identify the most ardent fans of a celebrity by listing
  - email address and frequency of sending email last year.

  * Relational Databases - store names and addresses of customers.

- *Information is structured as*

  * Tables with pairs of < key , value >

  * Search : Given a key find associated value

  * Insert : add a new key with associated value

  * Update : Given a key, change its associated value

CH# 7 ASSOCIATIVE ARRAYS

- *Ex. A table of people and their ages*

```
%ages = ("Michael Douglas", 39,
        "Dirty Harry", 34,
        "Agness", 27,
        "Dogbert", "21 in dog years",
        "Ronnie", 108);
#Find the age of people
 $x = $ages{"Michael Douglas"};    # $x = 39
 print $ages{"Dirty Harry"};        # prints 34
```

- *Syntax:*
    * Associative Array Names Prefixed by a % sign
    * Element name starts with a $ because element is a scalar
    * Index is enclosed in curly braces

- *Operations* - *Insert, Delete, Update, Search*

```
$ages{ "Mary" } = 30 ; #insert (Mary, 30)
$ages{ "Agness" } = 50 ; #update
delete $ages{ "Agness" } ; #delete
#Search
if ( $ages{ "Agness" } ) { print "Found Agness \n"; }
else { print "DId not find Agness \n"; }
```

CH# 7 : CLASS EXERCISE

- *Q? Explain the output of the following script?*

```
#!/usr/local/bin/perl
 %a = ('One', 1, 'Two', 2, 'Three', 3, 'Four', 4) ;
 @k = keys (%a) ; print "keys are @k \n" ;
 @v = values (%a) ; print "values are @v \n" ;
 ($k, $v) = each (%a) ; print "A pair is ($k, $v)  \n" ;
 ($k, $v) = each (%a) ; print "Another pair is ($k, $v)  \n" ;
 ($k, $v) = each (%a) ; print "Another pair is ($k, $v)  \n" ;
```

- *Q? Explain the output of the following script?*

```
#!/usr/local/bin/perl
 %a = ('One, 1, 'Two', 2, 'Three', 3, 'Four', 4) ;
 @k = keys (%a) ;
 foreach $tmp (@k) { print "$tmp => $a{$tmp} \n" ; }
 foreach $tmp (sort @k) { print "$tmp => $a{$tmp} \n" ; }
 foreach ($t, $s) (%a) { print "$t => $s \n" ; }
```

CH# 7 : CLASS EXERCISE

CH# 7 : CLASS EXERCISE

- *EX. 5-1 / pp. 77 71.*
- *Ex. Examine the contents of predefined associative arrays in Perl.*

  * Print out the values associated with USER and DISPLAY

  * in Associative %ENV using code fragment:
  print "You are called $ENV{’USER’} and you are ";
  print "using display $ENV{’DISPLAY’} \n";

CH# 7 EXTRA EXERCISES

- *EXTRA Ex. Examine the order of the elements in an associative array*

  * by a code fragment such as

```
%ages = ("Michael", 39, "Harry", 34, "Agness", 27) ;
@tmp = %ages ; print "@tmp \n" ;
$ages{"Dogbert"} = 21;
@tmp = %ages ; print "@tmp \n" ;
$ages{"Ronnie"} = 98 ;
@tmp = %ages ; print "@tmp \n" ;
$ages{ "Mary" } = 30 ;
@tmp = %ages ; print "@tmp \n" ;
```

  * Q? Explain the output.

CH# 7 ASSOCIATIVE ARRAYS -Semantics

- *Type Conversions into a list array*

```
@info = %ages;     # @info is a list array. It
                   # now has 10 elements
$info[5];          # Returns the value 27 from
                   # the list array @info
%moreages = @info; # %moreages is an associative
                   # array. It is the same as % ages
```

- *Associative Arrays - Semantics*

  * Do not have any order to their elements

  * The hash function on keys may explain the order

  * Not meaningful to ask for the "first" or "second" elements!

- *Q? How does one get the number of pairs in an associative array?*

- *Q? How does one process all the pairs in an associative array?*

CH# 7 ASSOCIATIVE ARRAYS

- *"Keys" and "values" function*

  * Keys returns list of the keys
  * Values returns list of values in the same order
  * Keys and values called in a scalar context

  - return the number of key/value pairs

```
foreach $person (keys %ages)
{
   print "The age of $person is $ages{$person} \n";
}
foreach $age (values %ages)
{
   print "Somebody is $age \n";
}
```

- *"each" operator returns two element list*

- of a key and its value

```
while (($person, $age) = each(%ages))
{
      print "$person is $age \n";
}
```

CH# 7 CLASS EXERCISE

- *Review chapter 5, pp 73-77 or 66-70 (Llama Book)*

- *Ex. Write a perl script to simulate the "printnv"*

- command in Unix.

  * Hint: print all keys and values in %ENV array.

- *Ex. Try exercise 2 , chapter 5 (pp 77 or 70) in Llama Book.*

  * Hint: Identify the keys and values.

CH#8: STRING MATCHING

- *Most useful features of Perl:*
  - powerful string manipulation facilities
- *Regular expressions*
  - Contained in slashes matching with the =˜ operator
    $sentence =˜ /the/
  - RE is case sensitive
    $sentence = "The quick brown fox";
    $sentence !˜ /the/
- *$_ special variable*
  - Assign the sentence to the special variable
  - avoid using the match and non-match operators
- *Ex. Try the following perl script with file1.pl as input.*

```
#!/usr/local/bin/perl -n
# script regex1.pl
# test command:   regex.pl < file1.pl
if (/#/) {
        print ;
}
```

CH#8: STRING MATCHING - Special Characters

- *Position - beginning, end*

ˆ # The beginning of the line or string

$ # The end of the line or string

- *Examples*

ˆf # f at the beginning of a line

ˆftp # ftp at the beginning of a line

e$ # e at the end of a line

tle$ # tle at the end of a line

ˆ$ # A line with nothing in it.

- *Ex. Identify two strings to match with each of the patterns*

\- listed above with the help of the following script:

```
#!/usr/local/bin/perl -n
# script regex2.pl
chop ;
foreach $pattern ( ’#’, ’ˆf’, ’ˆftp’, ’e$’, ’tle$’, ’ˆ$’) {
        if (/$pattern/) { print "$_ Matches $pattern \n" ; }
}
```

- *Inside of the slashes are interpolated*

- *Q? How will you look for the email addresses of senders*

\- in emails you have received last month?

CH#8: STRING MATCHING - Special Characters

- *Optional characters, repeated characters.*

```
.   # Any single character except a newline
?   # Zero or one of the preceding character
*   # Zero or more of the preceding character
+   # One or more of the preceding character
{m, n} # m to n repetition of the preceding character
```

- *Example: Identify the strings matching the following patterns:*
  - Patterns: t.e, .*, und*, und{2,3}, und+, und?, ^und?$
  - Strings: the, tre, tale, tle, te
  - Strings: un, und, undd, unddd, undddd
  - Strings: "\n", "abc", "ab\n", "\n\n", ""

- *Ex. Explain the output.*

```
#!/usr/local/bin/perl
# script regex3.pl
chop ; $x = 'blah' ;
@patterns = ('und?', 'und*', 'und+', 'und{2,3}', '^und{2,3}$' );
#push(@patterns, "$x", "$x{3}", "${x}{3}", "($x){3}", "(${x}){3}" );
@strings = ('un', 'und', 'undd', 'unddd', 'undddd' );
#push(@strings, 'blah', 'blahhh', 'blahblahblah', 'xxx');
foreach $p (@patterns) {
  foreach $s (@strings) {
        if ($s =~ /$p/) { print "$p Matches $s based on $&\n" ; }
  }
```

}

CH#8: STRING MATCHING - Choice

- *Pattern CHOICE: A vertical bar | means "or"*

  jelly|cream # Either jelly or cream

  (eg|le)gs   # Either eggs or legs

  (da)+   # Either da or dada or dadada or...

- *Character Choice - Square brackets*

  [qjk]   # Either q or j or k

    * Q? Express [qjk] using pattern choice operator |.

- *Special characters with [..]*

- - indicates "between"
- ˆ at the beginning means "not"

  [ˆqjk]   # Neither q nor j nor k

  [a-z]   # Anything from a to z inclusive

  [ˆa-z]   # No lower case letters

  [a-zA-Z] # Any letter

  [a-z]+   # Any non-zero sequence of lower case letters

CH# 8: STRING MATCHING - Odds and Ends

- *Predefined characters and character groups*

  \n   # A newline

  \w   # Any alphanumeric (word) character.

  \W   # Any non-word character.

- *Suppressing meaning of a special character*

  * Prefix it by a backslash

    \| # Vertical bar

    \[# An open square bracket

Exercises

- *EX. Parts (a)-(d) of Ex. 1, Chapter 7, pp 98 or 91, Llama Book.*

- *EX. - Ex. 2, chapter 7, pp 98, or 91 Llama Book.*

- *EXTRA: Write a perl script to read in a file and numbers selected*

- lines containing a person's name.
- The program should print out every
- line, but it should only number those specified.

  * Note that a name may occur in many different ways:
  - (a) Full Name, e.g. President William Jefferson Clinton
  - (b) Improper Capitalization, e.g. President CLINTON
  - (c) Abbreviated name - President Clinton or W. J. C.
  - (d) nick name - Bill

  * Also note that names are word, i.e. do not match Williamson.
  - Hint: Use \b to detect word boundaries.

  * Do not worry about CONTEXT yet! In other words do not
  - differentiate between Bill Cosby and Bill yet.

CH#9: SUBSTITUTION AND TRANSLATION

- *Make substitutions based on matches*

  $sentence =~ s/minneapolis/Minneapolis/

  - * Do the same thing with the $_ variable

    s/Minneapolis/Minneapolis/

  - \- Result of this expression = number
  - \- of substitutions made

- *Options - global substitutions*

  s/minneapolis/Minneapolis/g

  - * i option (for "ignore case")

    s/minneapolis/Minneapolis/gi

- *Ex. Identify three distinct strings altered by the following script*

```
#!/usr/local/bin/perl -n
s/bill/Bill/gi;
print;
```

TRANSLATION

- *tr function is character by character translation*
  * Position sensitive!
  * Ex. Replace each a with e, each b with d
    $sentence =˜ tr/abc/edf/
- *Returns Count of characters translated*
  * Ex. Counting the number of asterisks
    $count = ($sentence =˜ tr/*/*/);
- *Special RE codes do not apply in the tr function*
  * However dash means "between"
- *Q? Describe effect of following script on strings on standard input ?*

```
#!/usr/local/bin/perl -n
#script tr.pl
$x = $_ ; $y = $_ ; $z = $_ ;
$x =˜ tr/a-z/A-Z/; print $x;
$y =˜ tr/a-zA-Z/A-Za-z/; print $y;
$z =˜ tr/$z/\u$z/; print $z;
```

- *More details : Chapter 15 (pp. 172 or 159)*

CLASS EXERCISE

- *Ex. Write a perl script switchcase.pl to switch the case*
- of each character in a textfile.
- Each lower case character should be converted to
- the corresponding upper case character.
- Each upper case character should be converted to
- the corresponding lower case character.

  * Hint: Use tr/// command.

- *EXTRA Ex. Write a perl script using s/// command to mimic the effect*

  * of script tr.pl on the previous slide.

- *EXTRA Ex. Can you write switchcase.pl using s/// ?*

REMEMBERING PATTERNS

- *Motivation*
  - * Finding repeated characters, words, etc.
  - * Match sensitive editing
- *Mechanism: Parenthesis in a patterns => initializes memory*
  - * Variables $1,...,$9
  - - RE codes \1,..., \9
  - * $1 = substring matched to subpattern within First (...)
- *Ex. Find three strings to trigger output from the following script:*

```
#!/usr/local/bin/perl -n
chop;
if (/(\b.+\b) \1/)
{ print "Found $1 repeated in $_ \n"; }
```

- *Explanation*
  - * Identify any words repeated. \b is a
  - - word boundary and .+ matches any
  - - non-empty string

REMEMBERING PATTERNS - Match Sensitive Editing

- *Ex. What is the output of the following script:*

```
#!/usr/local/bin/perl
$_ = "The United States of America";
s/([A-Z])/:\1:/g;
print "$_ \n";
```

- *Using multiple memories:*

  * Q? What does this script do ?

```
#!/usr/local/bin/perl -n
s/^(.)(.*)(.)$/\3 \2 \1/ ;
print ;
```

- *Swaps the first and last characters*

- a line in the $_, ignore spaces

REMEMBERING PATTERNS - W/O Parenthesis

- *Read-only variables $’, $& and $‘*

- find what was matched before, during and after the search.

$_ = "The United States of America";

/of/;

- all of the following are true. (Remember that
- eq is the string-equality test.)

$‘ eq "The United States "; #dollar backquote

$& eq "of";

$’ eq " America"; #dollar quote

CLASS EXERCISE

- *Ex. 7-1(e), pp. 98 or 91, Llama Book.*

- *Ex. Write a perl script piglatin.pl to take a textfile*

- with one word per line from the standard input
- and translate them into Pig-Latin.
- Sample interaction is shown below:

```
> piglatin.pl
jumping
beans
thunder
Pig-Latin: umpingjay eansbay underthay
```

- *EXTRA Ex. Write a perl script swapnames.pl to take a textfile*

- with one (lastname, firstname) pair per line
- from STDIN and swap last name and first name.
- Sample interaction is shown below:

```
> swapnames.pl
Clinton, Bill
Swapnames: Bill Clinton
```

EXTRA EXERCISES

- *EXTRA: Write a perl script to read in a textfile and*
- flag repeated character by parenthesizing them.
    * For example your program would produce a line
    * like the following among others:

      Amp L(ee), Jay M(oo)n, Bob Barker, etc.
      These f(oo)tball players contributed many ...

- *EXERCISE: Modify your program so that it accepts one argument*
- and counts only those lines with that string.
- It should also put occurrences of this string in parentheses.
- So

  > ./countlines the

    * will output something like this line among others:

      But (the) greatest quarterback of
      (the)m all was Joe Montana, who

    * HINT: The array @ARGV keeps command line arguments
    - $ARGV[0] is first and
    - $ARGV[1] is second and $ARGV[2] is etc.

CH# 10: SPLIT

- *Splits up a string, places it into an array*
  - * Field separator specified by a regular expression
  - * Works on the $_ variable unless otherwise specified
- *Example*

```
$info = "Douglas:Michael:Actor:55, Sunset Drive";
@personal = split(/:/, $info);
```

  * which has the same overall effect as

```
@personal = ("Douglas", "Michael", "Actor", "55, Sunset Drive");
```

- *Information stored in the $_ variable*

```
@personal = split(/:/);
```

- *EXAMPLES: Word split into characters, sentence into*
- words, paragraph into sentences:

```
@chars = split(//, $word);
@words = split(/ /, $sentence);
@sentences = split(/\./, $paragraph);
```

  * Null string is matched between each character

CH# 10: SPLIT

- *A family of separators, e.g. :, ::, :::, ...*

  * use the Regular expression to describe the family

```
$_ = "Montana:Joe::Quarter Back:::Forty Niners";
@personal = split(/:+/);
```

  * Is the same as

```
@personal = ("Montana", "Joe", "Quarter Back", "Forty Niners");
```

- *But:*

```
$_ = "Montana:Joe::Quarter Back:::Forty Niners";
@personal = split(/:/);
```

  * Would be like

```
@personal = ("Montana", "Joe", "", "Quarter Back",
             "", "", "Forty Niners");
```

CH# 10: CLASS EXERCISE

- *Exercise 3, Chapter 7, pp 98 or 91, Llama Book.*

  * Note 1: Unix operating system uses formatted files for
  - storing passwd, groups etc.
  - Split is used to split the lines
  - in these files. The format of password file is given
  - in Chapter 7, Llama Book.

  * Note 2: Modern unix systems do not keep entire password
  - files in /etc/passwd for security issues related to
  - external password cracker programs. You may try
  - ’ypcat passwd’ or ’niscat passwd.org_dir’ or related
  - commands to get a more interesting password file.

CH# 10: EXTRA EXERCISES

- *EXTRA: Ex. 7-4, pp. 98 or 91 (Llama Book).*

- *EXTRA: Ex. Write a perl script piglatin2.pl to take a textfile*
- with MULTIPLE words per line from the standard
- and translate them into Pig-Latin.
- Sample interaction is shown below:

  > piglatin.pl
  jumping beans
  Pig-Latin: umpingjay eansbay

- *EXTRA: Read first 3 pages of Chapter 15*
- for substring and index operations.

  * Try Ex. 15-1 (pp. 162, 2nd Ed)

CH# 11: SUBROUTINES

- *Allows the user to define their own functions*

  * May be placed anywhere

```
sub mysubroutine
{
    print "Not a very interesting routine \n";
    print "This does the same thing every time \n";
}
```

- *Called with an & character in front of the name:*

```
&mysubroutine;            # Call the subroutine
&mysubroutine($_);        # Call it with a parameter
&mysubroutine(1+2, $_); # Call it with two parameters
```

- *Q? What kind of error message will the example produce*

\- in a language like C or Pascal?

CH# 11: SUBROUTINES

- *Parameters passed as a list in the special array @_*

  @_ is array of arguments
  $_[0] is first argument
  $_[1] is second argument

- *Ex. Prints out the list that it was called with*

```
sub printargs
{
       print "@_ \n";
}
&printargs("perly", "king");    # Example prints "perly king"
&printargs("frog", "and", "toad"); # Prints "frog and toad"
```

- *Elements of @_ can be accessed w/ square brackets*

```
sub printfirsttwo
{
          $x = $_[0] ; $y = $_[1] ;
         # or $x = shift @_ ; $y = shift ;
   print "Your first argument was $x \n";
   print "and $y was your second \n";
}
```

- *Indexed scalars $_[0] and $_[1] have nothing*

\-   to do with scalar $_

* No fear of clash

CH# 11: SUBROUTINES

- *Returning values: result of a subroutine is*
  - * always the last thing evaluated
- *return statement is also available*

```
sub maximum
{
    if ($_[0] > $_[1])
    {
        return $_[0];
    }
    else
    {
        $_[1];
    }
}
$biggest = &maximum(37, 24);   # Now $biggest is 37
```

CH# 11: SUBROUTINES

- *Local or Private variables*

```
sub inside()
{
    my($a, $b);               # Make local variables
    ($a, $b) = ($_[0], $_[1]);   # Assign values
    $a =~ s/ //g;              # Strip spaces from
    $b =~ s/ //g;              #   local variables
    ($a =~ /$b/ || $b =~ /$a/);  # Is $b inside $a
                               #   or $a inside $b?
}

&inside("lemon", "dole money");  # true
```

- *local($x) ; create dynamically scoped local variable*
- *my($x) ; create lexically scoped local variable*
    - * Preferred for ease of semantics
    - * Also faster
    - * Use instead of local in Perl 5.

CH# 11: SUBROUTINES - Scoping rules

- *Q? Explain the output of the following code:*

```
#!/usr/local/bin/perl
sub  s1 { print ’In s1, $c = ’, "$c \n" ; }
sub s2 {
        local($c) ; $c = 2 ;  print ’s3::$c =’, "$c \n";
        print "s2 calls s1: "; s1;
}
sub s3 {
        my($c) ; $c = 3 ; print ’s3::$c = ’, "$c \n";
        print "s3 calls s1: "; s1;
}
$c = 1 ; print ’main::$c = ’, "$c  \n";
print "\nmain calls s1: "; s1;
print "\nmain calls s1: "; s2;
print "\nmain calls s1: "; s3;
```

CH#11: EXERCISES

- *Review chapter 8 (pp 99-105 or 92-100), Llama Book.*
- *Exercise 1: Try Ex. 1 from chapter 8 (pp 105 or 100), Llama Book.*
- *Exercise 2: Try Ex. 2 from chapter 8 (pp 105 or 100), Llama Book.*
- *Exercise 3: Try Ex. 3 from chapter 8 (pp 105 or 100), Llama Book.*
- *Exercise 4: Write a recursive function to compute*

- factorial of a given positive integer.
- Note factorial(N) = 1 if N <= 1.
- otherwise factorial(N) = N * factorial (N - 1)
- Use local variables.

CH#11: EXTRA EXERCISES

- *Q? Write a perl script to read in a file with*
- one number per line and output it in a
- numerically sorted order.

  * Note: The sort operator on arrays uses lexicographic order.
  - You will need to define a numeric comparison routine
  - and pass it to the sort operator as shown in
  - Chapter 15 (pp. 168-172 or 153-162) under advanced sorting.

CH#12: Advanced Features - REPORT WRITER

- *Define how your page should look with*
  - In a single command
- *Takes care of page breaks, page numbers*

```
format STDOUT_TOP =
        Projects Using Too Much Disk      page @##
$%
   Project      Owner     Last Used       Cost
--------------  --------  ------------- -----------
.
format STDOUT =
@|||||||||||  @<<<<<<<  @>>>>>>>>>>>> @#######.##
$project,       $owner,   $lastUsed,    $cost
.
while (<>)
{
    ($project, $owner, $lastUsed, $cost) = split;
    write;
}
```

CH#12: Advanced Features - REPORT WRITER

- *Explanations*

_TOP   indicates a heading
.   ends a format description
$%   is the page number
@<<<<   is a left justified field
@|||| is a centered field
@>>>>   is a right justified field
@###.##   is a right justified, two decimal number

- *Special Variables*

$= is maximum number of lines per page
$- is number of lines in current page

- *No footers in Perl due to variable #lines per format!*

CH#12: EXERCISES

- *Review chapter 11 (pp 125-137 or 116-128), Llama Book.*
- *Exercise 1: Try Ex. 1 from chapter 11 (pp 138 or 128)*
- *Exercise 2: Try Ex. 2 from chapter 11 (pp 138 or 128)*
- *Exercise 3: Try Ex. 3 from chapter 11 (pp 138 or 128)*

CH#12: Advanced Features - DEBUGGER

- *Perl has a built-in debugging system*
- *Use the debugger*
  * #! /usr/local/bin/perl -d
  * commands
- *When you run the debug program it will start*

\- in debug mode

  * Debugging commands

    h help on debugger

    s step

    c continue to next break

    c <line> continue until line <line>

    n next (does not step into subroutines)

    l <range> list program statements in the <range>

    b <line> sets a breakpoint at line <line>

    p <expr> prints <expr> which is usually a variable

Advanced Perl

- *(i) Operating Systems Programming*
  * CH# 13: File tests
  * CH# 14: Directory Manipulation
  * CH# 15: Process management, Exception Handling
- *(ii) Perl 5*
  * CH# 16: Pointers and structures
  * CH# 17: Object-oriented programming features
  * awk-like BEGIN and END sections
  * Additional regular expression features
- *(iii) Common Gateway Protocol*
  * CH# 18: HTML and CGI
  * CH# 19: Generating dynamic web pages using Perl
- *(iv) Database Interface, e.g. Sybperl, Oraperl, ...*
- *(v) Project*
  * Processing the logfile from Web server
  * Processing EDI file

CH# 13: File Handles and File Tests

- *Source: Ch.11 (pp. 115-123 or 116-128) Llama Book*
- *What Is a File Handle?*
  * an I/O connection
  * e.g. STDIN, STDOUT, STDERR
  * Note: no prefix before the name
  * May be confused with reserved words
  * Recommendation: use all capital letters

CH# 13: Operations on Files:

- *1. Open(FILEHANDLE, "myfile")*

  * => create a file handle

- *2. <FILEHANDLE> reads from "myfile"*

- *3. print FILEHANDLE, "something"*

  * => writes to "myfile"

- *4. Close(FILEHANDLE) => closes "myfile"*

- *5. @info = stat("myfile") => information on "myfile"*

- *6. if (-r "myfile") { ... } => Test file properties*

CH# 13: Operations open and close

- *close(FILEHANDLE)*
  - * Closes FILEHANDLE
  - * Implicit close() at exiting perl script
  - * and reopening the filehandle
- *open (FILEHANDLE, "filename")*
  - * Open a new filehandle (besides STDIN, ...)
  - * returns false iff error
  - * permissions can be specified, e.g.
    open(FILEHANDLE, ">filename") - write
    open(FILEHANDLE, ">>filename")- append
- *die("message")*
  - * prints "message", program name, location (line#)
    open(FILE, "myfile") || die("can’t open myfile"); #e. g.
  - * die("message\n") will not print program name, line#

CH# 13: The -x File Tests

- *Motivating Risk: Opening a file for writing*

\- can destroy an existing file with same name

 * Q? How ensure that we do write to an existing file ?

- *Use file tests, e.g.*

```
$x = "/etc/passwd" ;
if (-e $x) {  #does /etc/passwd exist?
        #good
} else {
        print "How could we login?"
}
```

- *Tests will work with filenames or FILEHANDLEs*

- *Other File Tests (see Table 10-1, pp 119-120 or 113)*

 * Boolean: -r, -w, -x, d, -f

 * Others:

 - (-s "filename") returns length in bytes
 - -M, -A -C return number of days since file was
 - modified, accessed etc.

CH# 13: The stat() and lstat() Operations

- *Like the stat() command in Unix*

  * Takes a filename or a FILEHANDLE

  \- Default argument is $_

  * Returns either undef or

  \- an array with 13 elements

  * See unix manpage for stat(2)

- *Example:*

  ($dev, $ino, $mode, $nlink, $uid, $gid, $rdev,
  $size, $atime, $mtime, $ctime, $blksize, $blocks) = stat("filename");

- *Examine the output of 'ls -l' in Unix.*

- *Examine the man page for 'stat' in Unix.*

CH# 13: The _ FILEHANDLE

- *Analogy: default scalar $_, special array @_*

- *Note: Each file test with a file or a filehandle*

- is a system call to operating system.

  * Thus scripts with multiple file tests can be slow

- *Alternative:*

  * _ Provides efficiency in case of mutiple file tests

```
if ( (-r $file) && (-w _)) {
        print "$file is both readable and writable";
}
```

- *Perl will use the stats() from previous system-call*

- for file tests on _ FILEHANDLE.

CH# 13: EXERCISES:

- *Ex. 10-3 (pp. 123 or 115, Llama book)*
- For a given list of filenames in your directory,
- Display which are readable, which are writable
- and which are executable.

- *Ex. 10-4 (pp. 123 or 115, Llama book)*
- For a given list of filenames in your directory,
- Display which one is the oldest.

  * Hint: use stat()

- *Ex. Write a perl script to mimic unix command*
  "ls -l filename".

  * Sample output is enclosed:
    -rwxr--r--   1 sashi   1200 Aug 19 13:29 first.pl

  * You do not need need to convert time fields to date
  - unless you are familiar with the library DATE.pl

CH# 14: DIRECTORY ACCESS

- *Chapter 12 (pp. 139-143 or 129-133), Llama Book*
- *Overview*

  * Moving Around - chdir("mydirectory")

  * Globbing, e.g. @a = </etc/host*>
  - Analogy: ls /etc/host*

  * Directory Handles

  * Opening, Closing, Reading directories:
  - opendir(), closedir() , readdir()

CH# 14: The Directory Tree

- *Unix Directory Structure:*
  * Hierarchical, like a tree
  * Root Directory is called /
  * Children of root (try ’ls -F /’) are /etc, /bin, ...
  * Absolute pathnames : /usr/local/bin/perl
  - shows all directories on path from root to ’perl’
- *Present working directory*
  * Ex. Try ’pwd’ in Unix.
  * Each process has a working directory!
- *Relative filenames:*
  * Files in "pwd" first.pl, ./first.pl, ...
  * Files in parent directory: ../filename
  * Files in home directory: ˜sashi/first.pl, ˜/first.pl
  * Relative Names are convenient!!!

CH# 14: Moving Around the Unix Directory Tree

- *Unix has ’cd’ command, e.g.*

```
pwd
cd /
pwd
ls
```

- *Perl has chdir() operator*
    - Takes 1 argument, i.e. directory name
    - Returns TRUE iff successful in changing directory
    - Will not affect "pwd" of parent process (shell)!

- *Example:*

```
print "where do you want to go? ";
chop($destination = <STDIN>);
if (chdir $destination) {
        #we got there
} else {
        #we did not get there
}
```

CH# 14: GLOBBING

- *What is Globbing in Unix?*
  - Try ’ls *’ or ’cat *.pl’ or ’echo *.pl’ with Unix shell.
  - Shell converts solitary *
  - to a list of all files in present working directory
- *Globbing in Perl*
  - Syntax: * inside angular brackets
  - Semantics:

```
@a = </etc/* /etc/.*> ; #list of files in "/etc" directory
while ($aFile = </etc/*>)  #next file in "/etc"
 { print "one of files is $aFile \n"}
```

  - Returns undef (FALSE) in last iteration
- *Multiple patterns ok, i.e. @a = <*.pl *.c *.h>*
- *Interpolation, $w = ’/etc’ ; @a = < $w/* ${x} >*
  - Similar to Regular Expressions
  - Except for special characters, e.g. \

CH# 14: Directory Handles

- *What is a directory handle?*

  * Name for interacting with a directory e.g.DIR

  * Separate namespace relative to FILEHANDLES

- *Opening, Reading, Closing Directories*

```
opendir(ETC, "/etc")  || die "no /etc directory?" ;
while ($filename = readdir(ETC) ) {#scalar 1 per loop
        print "$name\n" ;
}
closedir(ETC);
opendir(ETC, "/etc")  || die "no /etc directory?" ;
@filenames = readdir(ETC); #array context
foreach $name (sort @filenames) {
        print "$name\n" ;
}
closedir(ETC);
```

- *Comparison with GLOB*

  * readdir() - Give all filename including

  \- those beginning with '.'

CH# 14: EXERCISES

- *Ex. 12-1, pp. 143 or 133, Llama Book*
- *Ex. 12-2, pp. 143 or 133, Llama Book*
- *Ex. Write a perl script to mimic ’ls -F directory’.*

  * Sample output is enclosed:

```
Mail/            ex4-4.pl~*    hello.pl~*    pl/
PERL/            ex4.pl*       index.html    pop.pl*
adv-util-files/  ex4.pl~*      introducti    pop.pl~*
```

- *Ex. Write a perl script to mimic ’ls -R directoryname’.*

CH# 15: PROCESS MANAGEMENT

- *Chapter 14, pp. 153-164 or 142-152, Llama Book*
- *Overview: Executing shell commands*
  - * (i) backquotes
    $now = ‘date‘;
  - * (ii) system() command
    system("date > datefile");
  - * (iii) opening processes as files
    open(LP, "| lpr -Pcs"); print LP, @a; close(LP);
  - * (iv) fork-exec-wait protocol
    if (fork) { ... } else { exec("date > datefile") ; }

CH# 15: BACKQUOTES

- *Syntax: backquotes around commands, i.e. ‘command‘*
- *Semantics: Takes a command and*
  * Launches /bin/sh command
  * returns a scalar or an array
  * Synchronous: Parent waits for child to finish!

```
$now = ‘date‘ ; #scalar context
#$now becomes "Fri Aug 21 12:00:00 EDW 1996"
@a = ‘who‘ ; #array context
#@a has 1 element for each line from "who"
```

- *About the Child Process:*
  * STDOUT is redirected to parent.
  * STDIN, STDERR are inherited.
- *Redirecting, e.g. STDERR*

```
$errormessage = ‘rm fred 2>&1‘;
```

CH# 15: Using Processes as Filehandles

- *Syntax*

  * vertical bar ’|’ differentiates processes from files
    open(LS, "ls *.pl|"); # STDOUT linked to parent::LS
    open(CAT, "| cat"); #STDIN linked to parent::CAT

- *Semantics:*

  * STDIN or STDOUT of child is linked to parent

  * Asynchronous except for pipe read/write

  * Idea: Use processes like files
  - by open, read/write, close protocol.

  * Analogy: popen library routine

CH# 15: Processes as File Handles - contd.

- *Example:*

```
open(LS, "ls *.pl|");
open(CAT, "| cat > training.report");
@scripts = <LS> ; $numScripts = @scripts;
print CAT, "Dear Boss,\n";
print CAT, "We developed $num scripts in Perl class.";
print CAT, "\nA list is enclosed:\n" ;
foreach $s (@scripts) {
      $lines = ‘wc $s‘  = 0;
       print CAT "Script $s - $lines lines\n" ;
}
close(LS) ; close(CAT);
```

- *Q? What will the program do if*

- "cat > training.report" is replaced by
- mail $emailaddress, e.g. mail president@whitehouse.gov

CH# 15: Summary

- *See Table 14-1, pp 160 or 152*
- *Executing shell commands*
  - (i) Synchronous - with output coming back to a variable
    $now = ‘date‘;
  - (ii) Synchronous - STDI/O copies from parent process
    system("date > datefile");
  - (iii) Asynchronous - with input or output from parent
    open(LP, "| lpr -Pcs"); print LP, @a; close(LP);
    open(LS, "ls *.pl | "); @a = <LS>; close(LS);
  - (iv) Asynchronous - user selected STDI/O...
    if (fork) { ... } else { exec("date > datefile") ; }

CH# 15: EXERCISES:

- *Ex. 14-1, pp. 164 or 152 , Llama Book*
- *Ex. 14-2, pp. 164 or 152 , Llama Book*

CH# 15: Sending and Receiving Signals

- *What is a signal (exception)?*

  * External events, e.g. control-C

  * Internal errors, e.g. divide-by-zero

  * Requires immediate attention!

  * Perl has predefined exceptions:
  - e.g. INT => control-C

  * (see manpage signal for Unix/C)

- *Exception Handlers:*

  * Is a user defined subroutine.

  * Is associated with exceptions.

  * Is called implicitly by runtime system
  - whenever associated exceptions occur.

CH# 15: Exception Handlers in Perl

- *%SIG associative array in perl*

  * Holds the association between

  \- exceptions and handler routines.

- *Associating Handlers to Exceptions:*

```
$count  = 2;
$SIG{'INT'} = 'my_handler' ;
while ( $count> 0 ) {
        print "Waiting for $count control-Cs\n" ;
}
$SIG{'INT'} = 'DEFAULT'; #restore default action

sub my_handler {
        print "Subroutine my_handler invoked\n";
        print "We probably saw a control-C\n" ;
        $count--;
}
```

CH# 15: EXERCISES

- *Exploring the exception handler in Perl.*

- *Ex. 1a.: Print names of various exceptions (aka signals)*
- supported by perl interpreter. (Hint: check %SIG)

- *Ex. 1b.:Test the exception handler in the example on the*
- previous slide.

- *Ex.2. Read section on "Sending and Receiving signals"*
- (pp. 161-3 or 150-2, Llama Book).

  * Write a perl script which traps
  - control-C (SIGINT). Its exception handler routine should
  - ask the user if the program should quit or continue.

- *Ex. 3. Write a perl script to determine the behaviour of*
- Perl exception handler when an exception arrives inside the handler.
- Hint: print a message and then wait for a control-C
- inside the handler.

CH# 16: POINTERS / REFERENCES in Perl 5

- *Overview:*
  - Hard and Symbolic References
  - References to Arrays and Scalars
  - Passing arrays to subroutines by reference
  - References to subroutines
- *What is a Reference?*
  - A pointer to something, i.e. a variable
  - An address to a value
  - Useful in creating sophisticated data-structures,
    - e.g. multi-dimensional arrays

CH# 16: Hard and Symbolic References

- *Symbolic Reference (Perl 4, 5)*

  * Contains name of a variable for symbol table lookup

  * Useful for creating variable names
  - and addressing them AT RUNTIME

- *Ex.: $$language* --> *$perl, if $language is "perl"*

- If the variable does not exist, it will be created!!!

- *Example:*

```
$language = "perl" ;
$perl = "pretty" ;
print "${language} \n" ; #prints perl
print "${perl} \n" ; #prints pretty
print "$$language \n" ; #prints  pretty
```

- *Like macro substitution like make, sh*

```
$freeway = 55; $highway = 45;
$r = "free" ; $s = "high" ;
print "${r}way \n" ;
print "${s}way \n";
```

CH# 16: Hard References (Perl 5.0)

- *Refers to variable’s value directly.*
  - * Like pointers in ’C’
  - * Holds Actual addresses in memory
  - * Keeps track of Reference counts
  - * If the variable does not exist, it is not created .

```
#!/usr/local/bin/perl5
$perl = "pretty";
$pointer = \$perl ; #get address of $perl
print ’Address of $perl is ’, "$pointer\n";
print ’Deferenced $pointer is ’, "$$pointer\n";
```

- *Backslash Operator (\)*
  - * Similar to the ’address-of’ operator (& in C)
  - * Returns the memory address of a variable
  - * Creates a new reference to the value
  - * Contains a hexadecimal address and type of value
- *Deferencing ($)*
  - * Takes a hard reference (e.g. $$pointer)
  - * Returns value stored at the location pointed to

CH# 16: Pointers to Arrays

- *Work the Same Way as pointers to scalars*

```
#!/usr/local/bin/perl5
$pointer = \@ARGV; #address-of
print "array ARGV is located at $pointer\n";
$numArgs = @$pointer; #dereference
print "There are $numArgs arguments.\n";
$i = 0;
foreach $x ( @$pointer ) {
        print "Element $i  =  $$pointer[$i++] \n";
}
```

- *Pointers to associative arrays are similar*

```
#!/usr/local/bin/perl5
$pointer = \%ENV; #address of
print "array ENV is located at $pointer\n";
$numArgs = @$pointer; #dereference
print "There are $numArgs environment variables.\n";
@vars = keys %$pointer ;
$x = $vars[0];
print "$x  = $$pointer{$x } \n";
```

CH# 16: Multidimensional Arrays

* Creates reference to an anonymous array
* Elements can be retrieved by -> operator
* Example:

```
$pointer = ['a', 'b', 'c'] ;
$p = [ 'a', 'b', ['1', '2'], ['3', '4'] ] ; #2-d array
$i = 0; $j = 2;
print $pointer->[$i];
print $p->[$j][$i] ;
```

- *Syntax variations*

```
# equivalent calls
# print "${$pointer}{$i}"
# print "$pointer->{$i}"
# these wont' print:
# print "${pointer{$i}}"
#                 -- references array with no first element
# ${$pointer->{$i}} -- same reason
```

CH# 16: Parameter Passing By Reference

- *Consider the subroutine with side-effect*
  - e.g. chop, reverse, sort, ...
  - Example Code of swap:

```
$a = 2 ; $b = 3;
swap($a, $b);
print "a is $a, and b is $b\n";
sub swap {
        local($x, $y) = @_;
        $tmp = $x ; $x = $y ;$y = $tmp;
}
```

- *Vocabulory*
  - Formal parameters: @_ or $x, $y;
  - Actual Parameters: $a, $b for the call
- *Parameters usually passed by VALUE Copy*
  - Actuals and formals have own memory locations!
  - i.e. $_[0] = $a ; $_[1] = $b ; during &swap($a, $b)
- *Problem: Actual parameters can't be changed by*

- changing formal parameters!!!

CH# 16: Parameter Passing By Reference

- *Solution: Use pointers to actual parameters*
    * Pass pointers to actual variable
    * Actual parameters can be changed from subroutine
    - via dereferencing formal parameters.

- *Example:*

```
$a = 2 ; $b = 3;
swapWpointers(\$a, \$b);
print "a is $a, and b is $b\n";
sub swapWpointers {
  local($x, $y) = @_;
  $tmp = $$x ; $$x = $$y ;$$y = $tmp;
}
```

CH# 16: EXERCISES

- *Ex. 1. Test the swap() and swapWpointers()*
- routines from previous slides.

- *Ex. 2. Write a Perl5 subroutine which takes two arrays,*
- say @a and @b, as arguments. It assigns the reverse
- of first array (i.e. reverse(@a)) to second argument @b.

  * Hint: use parameter passing by reference.

- *Ex. 3. Create a 2 dimension array to store a multiplication*
- table among umbers 0 thru'5. Script should print out the
- table on STDOUT in a nice format.

- *Ex. 4. Write a Perl5 subroutine mychop() which*
- takes a string and removes its last character
- via side effect. Analogy - chop operator.

CH# 17 Objects In Perl5

- *Object oriented progrmming in Perl5 is based on*
    - * Packages, Modules
    - * Pointers
    - * Anonymous Variables
- *It is a honor systems*
    - * You can simulate object orietned programming
    - * Interpreter will not enforce it!
    - * Do not choose perl 5 if OOP is all you need!
- *Simultion Tools*
    - * Reference - a pointer
    - * Object - a pointer that knows its class
    - * Class - a package with methods to manipulate objects
    - * Method - a subroutine inside an object or a class
    - * Encapsulation, Inheritance, Polymorphism - Packages
    - * Instance variables - Anonymous variables

CH# 17 Objects In Perl5 - Packages

- *Package is a group of names*
  * Can include variables, subroutines
  * Default package is ’main’
  * User defined names are always inside some package
  * Only global names are %ENV, %SIG, @ARGV etc.
- *Scope Resolution Operator (::)*
  * PackageName::variableName
  * PackageName::procedureName
- *Example*

```
package Jamaica;
$plant = "coconut" ;

package main;
$plant = "palm" ;
print main::$plant , Jamaica::$plant ;
```

CH# 17 Objects In Perl5 - Packages

- *Packages vs. Files*

  * A File may contain many packages

  * A pakage may be spread across many files

```
#file: file1.pl
package A;
$x = 2 ; $y = 3;
#file: file2.pl
package A;
$s = 2 ;
```

- *Symbol tables for packages* - *%::*

  * Contains names, values

```
#!/usr/local/bin/perl
#file: symboltable.pl
package A;
 $a = 1; $c = 2;
 while (($k, $v) = each(%::)) { print "A::$k = $v \n"; }
package main;
 $c = 3 ; $d =4 ;
 while (($k, $v) = each(%::)) { print "main::$k = $v \n"; }
```

CH# 17 Objects In Perl5 - Modules

- *Module = a pacakge defined in a file with same name*

  #file name = box.pm

  pakage box;

  #definitions

- *Use ststement* - *use box;*

  * imports all names defined in module box

  \- infile box.pm

CH# 17 Objects In Perl5 - Bless()

- *Object in Perl5 = a reference that knows its class*
- *bless(Reference R, Package P)*

  * Make the variable (i.e. object) pointed to Reference
  \- an intance of class P

- *All constructors must bless!*
- *Example Class Interval with Instance Variables & Methods*

```
package Interval;
sub new {
        my $type = shift; my %params = @_ ; my $self = {};
        $self->{High} = $params{High} ;
        $self->{Low} = $params{Low} ;
        return bless $self, $type;
}
#virtual method, get object name as first argument
sub find {
        my $self = $_[0]; my $key = $_[1];
        return $self->{$key};
}
```

CH# 17 Objects In Perl5 - Simple Example

```
package main;

# exercise the class Interval
#by creating instances $a and $d and
#call methods new() and find()
#
$a = Interval->new( High => 42, Low => 11 );
$d = Interval->new( High => 22, Low => 23 );
print 'For instance $a ',"High=$a->{High}" ;
print "and Low=$a->{Low}\n" ;
$value = $d->find(High) ;
print 'Value with key High in $d is ' , $value, "\n" ;
$value = Interval::find($d, "Low");
print 'Value with key Low in $d is ' , $value, "\n" ;
```

- *Q? What happens when $d is created using*

  $d = Interval::new(High => 22, Low => 23);

- :: won't prepend name of class to @_ for new()

CH# 17 Objects In Perl5 - Exercises

- *Q? Which is not a valid constructor?*

(a) sub new { bless {} }

(b) sub initialize { my $self = {} ; bless self; }

(c) sub new { my $self = {} ; return self ; }

(d) sub new {my $self = []; return bless $self ; }

- *Q? Which is an incorrect call to the new() method?*

$unit = new Interval(High => 2, Low => 1);

$unit = Interval::new(High => 2, Low => 1) ;

$unit = Interval->new(High => 2, Low = 1) ;

$unit = Interval::new(’Interval’, ’High’, 2, ’Low’, 1) ;

- *Q? Define an instance method Interval::midpoint*

- which returns the average of Low and High.
- Test the method with two different instances.

- *EXTRA Q? Provide a different implementation of*

- class Interval using reference to arrays instead of
- reference to a hash.

CH# 17 Objects In Perl5 - Inheritance

- *A Class inherits from classes listed in its @ISA array*

```
package realInterval; @ISA = qw(Interval) ;
sub new {
        my $class = shift; my $self {} ; bless $self;
}
sub set_High {
        my $self = shift; $self->{'High'} = 'open';
}
sub set_Low {
        my $self = shift; $self->{'Low'} = 'open';
}
package main;
$a = new realInterval;
$a->set_High( 5 );
$a->set_Low( 2 );
print $a->find('High') ; #inherited
```

CH# 17: Objects

- *Q? Study the Box class given on next few slides.*

- Implement a Square class
- as a derived class of box.

  * Class Box has instance variable xLL, yLL, xUR, yUR,
  - instance methods display(), area(), display2()
  - and class method new().

  * Class Square should have
  - instance variable xLL, yLL, height
  - instance methods display(), and area()
  - and class method new().

CH# 17: boxdemo.pl

```
#!/usr/local/bin/perl5
package Box;

sub new {
	my $type = shift; #first paramemter has class name
	my %params = @_ ; my $self = {};
	$self->{xLL} = $params{xLL} ; #create instance variables
	$self->{yLL} = $params{yLL} ;
	$self->{xUR} = $params{xUR} ;
	$self->{yUR} = $params{yUR} ;
	print "Box::new() created an object with ";
	print "\nxUR = $self->{xUR}, yUR = $self->{yUR} ";
	print "xLL = $self->{xLL}, yLL = $self->{yLL} \n";
	return bless $self, $type;#make reference to be an object
}
```

CH# 17: boxdemo.pl contd.

```
#virtual method, get object name as first argument
sub area {
	my $self = $_[0];
	my $width = $self->{xUR} - $self->{xLL}  ;
	my $height = $self->{yUR} - $self->{yLL}  ;
	print 'Box::area(): area = ',  $width * $height, "units \n";
	return $width * $height ;
}

sub display {
	my $self = $_[0];  my ($key, $value);
	print "Box::display() ";
	while ( ($key, $value) = each (%$self) ) {
		print "$key = $value, ";
	}
	print "\n";
}
```

CH# 17: boxdemo.pl contd.

```
package main;

print "This program is a Demostration of objects in Perl5 \n" ;
print "Package box represent a class box \n" ;
print " Main package creates box instance $a and $b \n" ;
print "  to test the instance methods and class methods \n" ;
print " \n" ;

$a = Box->new( xLL => 0, yLL => 0, xUR => 3, yUR => 6 );
Box::display($a);
Box::area($a);

print " \n" ;
$b = Box->new( xLL => 0, yLL => 0, xUR => 3, yUR => 4 );
$b->display();
$b->area();
```

CH# 18: HTML and CGI

- *Terminology*

  * World wide web (WWW) = collection of computers
  - speaking HyperText Transfer Protocol (HTTP)

  * Web server - supplies web pages

  * Web browsers - clients requesting web pages

- *Web Page Formatting*

  * HyperText Marking Language (HTML)

  * Plain ASCII text file

  * With tags to desribe display properties

```
<HEAD><TITLE>test.cgi</TITLE></HEAD>
<BODY>
<H1>Point size for Header H1 is large</H1>
<H3>Point size for Header H3 is not so large</H3>
<P>
<A HREF="http://www.cs.umn.edu"> Click here! </A>
This is another paragraph on fonts. <BR>
There are three types of fonts, including
<I> Italics </I> , and <B> Bold </B>
</P>
</BODY>
```

- *HTML Tutorials on web*
  * http://www.stars.com - library of books, faqs
  * http://www.jmarshall.com - quick tutorial
  * http://www.msi.umn.edu

CH# 18: HTML and CGI

- *Motivation for CGI*

  * (i) Forms - a subset of HTML to allow users
  - to supply information in a structured manner.

  * (ii) Gateways - programs to access information
  - that is ordinarily accessable via Web

  * (iii) Virtual Documents - Creating the document
  - on the fly, customized to user requests.

- *Examples*

  * http://www.lycos.com
  - search results return as a hypertext document

  * http://www.cosy.sbg.ac.at/rec/guestbook
  - Form based application to allow leaving messages

CH# 18: HTML and CGI

- *Dynamic Web Pages needs*

  * (i) A program to generate HTML formatted text
  - e.g. a CGI script

  * (ii) A web server to executes the program
  - when the page is visited

- *Common Gateway Interface (CGI)*

  * A data transfer protocol b/w Web servers and programs

  * Three ways to transfer data
  - (i) Command line argument
  - (ii) via standard input
  - (iii) thru' environment variables

- *Getting CGI to find your Perl5 script*

  * (i) Location: Special directory, e.g. cgi-bin, htbin
  - Specified in httpd.conf file for httpd web serversa

    Exec /cgi-bin/* /project/WWW/scripts/*

  * (ii) Permission: run as user 'nobody'
  - assuming your script is executable by all

CH# 18: HTML and CGI

- *Example hello.cgi*

  * display hello world message in colors

  * Pupose: Confidence Booster

```
#!/usr/local/bin/perl
#filename - hello.cgi
print "Content-type: text/html\n\n";
print "<HTML> <HEAD><TITLE>hello.cgi</TITLE></HEAD>
<BODY BGCOLOR= \"000000\">
<FONT COLOR=GREEN SIZE=1> Hello World </FONT><BR>
<FONT COLOR=RED SIZE=2> Hello World </FONT><BR>
<FONT COLOR=WHITE SIZE=3> Hello World </FONT><BR>
<FONT COLOR=BLUE SIZE=4> Hello World </FONT><BR>
<FONT COLOR=CYAN SIZE=5> Hello World </FONT><BR>
<FONT COLOR=MAGENTA SIZE=6> Hello World
</FONT><BR>
<FONT COLOR=YELLOW SIZE=7> Hello World
</FONT><BR>
</BODY> </HTML> " ;
```

CH# 18: HTML and CGI

- *Example test.cgi*
  - * display a predefined HTML document
  - * Pupose: Confidence Booster

```
#!/usr/local/bin/perl
#test.cgi
print <<EOF
Content-type: text/html

<HEAD><TITLE>test.cgi</TITLE></HEAD>
<BODY>
<H1>Font size for header H1 is large</H1>
<H3>Point size for Header H3 is not so large</H3>
<H6>Point size for Header H6 is very small</H6>
This is the first sentences of first paragraph
in the body of the html document.
Long sentences are wrapperd automatically.
<P>
This is another paragraph on fonts. <BR>
There are three types of fonts, including
<I> Italics <\I> , and
<B> Bold <\B> </P>
</BODY>
EOF
```

CH# 19: Generating dynamic web pages using Perl5

- *Dynamic Creation of HTML pages*

  * Using Data from a file

```
#!/usr/local/bin/perl
#s.cgi - generate HTML to display contents of s.dat
print "Content-type:text/html \n\n";
print "<BODY>\n";
open(IN,'s.dat');
while ($line=<IN>)
{
        ($name,$address)=split(/:/,$line);
        print <<XXX
        <H1>$name</H1>
        <H3>$address</H3>
        XXX
}
close(IN);
print"</BODY>";
```

- *Sample s.dat*

Sam Malone:58 Rainbow Dr.
Diane:251 Oakwood Dr.
Carla:932 Minnehaha Avenue
Clifford:365 Silver Lake Road

CH# 19: Generating dynamic web pages using Perl5

- *Example : envdump.cgi*

  * prints environment via HTML formatting

```
#!/usr/local/bin/perl
# envdump.pl
print <<EOF
Content-type: text/html

<HEAD><TITLE>Environment variables</TITLE></HEAD>
<BODY><H1>Environment variables</H1>
<H5>
EOF
;

@vars = sort keys(%ENV);
foreach $var (@vars)
{
        print "<P> $var=$ENV{$var}";
}
print "</H5></BODY>";
```

CH# 19: EXERCISES

- *Ex. There are several errors in the following CGI script.*

\- Identify the serious one.

```
#!/usr/bin/perl
print "<TITLE> The Error Page </TITLE>\n" ;
print "<H1> Hello $ENV{REMOTE_USER} </h1> \n" ;
print "\n" ;
```

- *Ex. Your CGI Perl5 Script isn’t being executed.*

\- What’s a possible cause?

  * (i) Script not in a directory "approved" by Web server
  * (ii) Script has a syntax error
  * (iii) Script is not world executable
  * (iv) Script is world writable
  * (v) All of the above.

CH# 19: Getting data About Client

- *gbook2.htm*

```
<HTML>
<HEAD>
<TITLE>Guestbook</TITLE>
</HEAD>
<BODY>
<H2>Guest book program</H2>
<H3>
For list of visitors, click
<A HREF="http://www.technology.umn.edu/~studentj/gbook2.cgi">
here</A>
</H3>
</BODY>
</HTML>
```

CH# 19: Getting data About Client

- *Q? How does gbook2.cgi get client name and machine name?*

- *gbook2.cgi*

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

print "Content-type: text/html\r\n\r\n" ;
print "<TITLE> Guest Book Version II </TITLE>" ;
print "\n" ;
print "<H1> Guest Book Version II </H1>" ;
print "\n" ;
print '<BR><H3>Back to <A
HREF=http://www.technology.umn.edu/~stu
dentj/gbook2.htm> ';
print 'Cover Page</A>.';
print "</H3>\n" ;
print "<H4> Using environment variables \n";
print "to log visitor information. </H4>" ;
print "\n" ;
```

CH# 19: Getting data About Client

```
print '<BR> This page has been visited by : ', "\n" ;

$file = 'guest2.dat' ;
open(DATA, $file);
while (<DATA>) { chop; print "<BR> $_"; print "\n" ; }

open (DATA, ">>$file") ;
if ( $ENV{'REMOTE_USER'} ) {
	print DATA "$ENV{'REMOTE_USER'}\@$ENV{'REMOTE_HOST'}\n";
} elsif ($ENV{'REMOTE_HOST'}) {
	print DATA "someone\@$ENV{'REMOTE_HOST'}\n";
} else {
	print DATA "someone\@somemachine \n" ;
}

print "\n" ;
close DATA;
```

CH# 19: Getting data from Forms

- *File: gbook3.htm*

```
<TITLE> Guest Book </TITLE>
<H3> Cover page for guest book - version 1. </H3>
<P>
For getting and updating the list of visitors
<A HREF="http://www.cs.umn.edu/˜shekhar/gbook2.cgi">
Click here </A>!
<P>
Please fill out the following questionnaire
to help improve this site.
<FORM ACTION="http://www.technology.umn.edu/˜studentj/gbook3.cgi">
I am:
<MENU>
<INPUT TYPE="radio" NAME="mood" VALUE="very happy" CHECKED>very happy
<INPUT TYPE="radio" NAME="mood" VALUE="happy">happy
<INPUT TYPE="radio" NAME="mood" VALUE="satisfied">satisfied
<INPUT TYPE="radio" NAME="mood" VALUE="not satisfied">not satisfied
</MENU>
with this site.
<P>
<INPUT TYPE="submit"><INPUT TYPE="reset">
```

</FORM>

CH# 19: Getting data from Forms

- *Q? How does gbook3.cgi get the input from form gbook3.htm*

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

$args = $ENV{'QUERY_STRING'} ;
%argPairs = split(/[&=]/, $args);
$mood = $argPairs{'mood'};
$mood =~ s/\+/ /g;  # replace + by space
$happiness = $mood ;
print "Content-type: text/html\n\n";

print "<TITLE> Feedback </TITLE>" ; print "\n";
print "<H3> Feedback </H3>" ; print "\n";
print "<HR>" ; print "\n";
print "So you felt $happiness with this page.\n" ;

if ($happiness eq 'very happy') {
      print 'Thank you. Please visit again. !<BR>' ;
} elsif ($happiness eq 'happy') {
      print 'Glad you enjoyed it!<BR>' ;
} elsif ($happiness eq 'satisfied') {
      print 'Hmmm. Can you suggest an improvement? <BR>' ;
} else {
      print 'Who invited you to this site?. <BR>' ;
}
```

CH# 19: EXERCISES

- *Ex. Write a CGI script, count.pl, in Perl5 to maintain*

- the count of its invocation from Web.

  * Create a HTML file to display a simple page
  - display "Guest book program" as a header
  - and "How many people have clicked here?"

  * Clicking on ’here’ on the page should
  - trigger the script count.cgi, which will
  - display the count with label "Number of visits to this page".

  * The count should be stored in a file.

- *EXTRA Ex. Provide a breakdown of count by visitors.*

- Consider processing log of gbook2.cgi for this.

- *EXTRA Ex. Add a button to reset the count.*

- *Read Chapter 19 (pp. 180-209) from Llama Book (2nd Ed.)*

- *Ex. 19-1, pp. 209 (2nd Ed. Llama Book)*

- *Ex. 19-2, pp. 209 (2nd Ed. Llama Book)*

CH# 19 Perl5/WWW Resources

- *cgi.pm library (Chpater 19, Llama Book)*
- *http://www.oac.uci.edu/indiv/ehood/perlWWW*
  - * Eight categories or Perl5 programs
  - * Applications, browsers, converters, developers, ...
- *CGI.pm* - *a popular and useful library*
  - * http://www-genome.wi.mit.edu/ftp/pub/software/WWW/cgi-docs.html
- *The CGI Module Set from CPAN*
  - * Six modules - base, request, miniSvr, response, ...

CONCLUSIONS

- *Practical extraction and reporting language*
  * Combining elements of C-shell, awk and other
  - utilities
- *Perl5 is Free, NOT "public domain"*
  * README file that explains your license
- *Two publicly available books; comp.lang.perl*
- *better than shell script programming and awk or sed*
  * No shell initiation overhead
  * Can read and write binary data files
  * Can have many files for input or output at one time
  * It has a report writer
  * Has extended regular expressions
  * It has both linear and associative arrays
  * Has powerful defaults that simplify programming
  * It can process very large files without
  - record size limits

CH # 20 PROJECTS - Logfiles from Web

- *(i) Produce reports from a weblog*
- *(ii) Produce reports from a email file*
- *(iii) Produce reports from a EDI file*

CONCLUSIONS

- *Is better than C, C++, or Fortran*
  - No visible compile or link stages
  - Rich set of character string manipulation
  - Include X-windows features, tkperl, access to databases
- *Perl5 has some down points*
  - 1.7 times as long to execute as an equivalent C program
  - "gotcha’s" would confuse non-programmers

```
-        $num = 010;
-        print "The number is $num. \n";
-   will print
-        The number is 8
```