What is Perl?

- Perl: Practical Extraction and Report Language
- An interpreted language by Larry Wall
  - similar to C, csh, sh, sed, awk
- "A shell for C programmers"- Larry Wall

Sriram Srinivasan (© 1999)

Aaargh - not another language !

- So, why should I use Perl
  - Much faster than shell script equivalents
  - Less syntax than all other tools of which it is a superset.
    - More consistent, and more options
  - "But I like the Unix small tool philosophy"
    - So do I - Perl fits in right there
    - Complements other tools - you don't *have* to use Perl alone
  - Portable across many platforms
    - Windows, Unix, DOS, Mac, Amiga, VMS, Cray
  - Easy to learn
- Why shouldn't I use Perl
  - No reason, really. Just kidding:-)
  - Need for fine-grained complex data structures, maximum performance. Use C.
  - If interfaces from Perl not available

Sriram Srinivasan (© 1999)

Sources of Information

- **Paper publications**
  - **Online reference documentation - all 500+ pages**
  - **"Learning Perl"- by Randal Schwartz, Tom Christiansen**
  - **"Programming Perl"- Larry Wall, Randal Schwartz, Tom Christiansen**
  - **"Advanced Perl Programming" - Sriram Srinivasan**
  - **"Effective Perl Programming" - Joseph Hall**
  - **"Perl Cookbook" - Tom Christiansen, Nathan Torkington**
- **Internet**
  - **http://www.perl.com/**
  - **comp.lang.perl.* - USENET newsgroups**
- **Perl modules (both built-in and from perl.com)**

Sriram Srinivasan (© 1999)

Perl programming: Essentials

print "hello world\n"; # An example

- **All statements end with a ";"**
- **A block of statements is enclosed in { .... }**
- **Anything following a # is treated as a comment**
- **Execute this script as**
  - `perl testfile.pl`
  - `perl -e 'print "hello world\n"'      # On unix systems`
  - `perl -e "print \"hello world\n\""   # In a dos window`

Example : Prime numbers

```
1: # This program prints prime numbers from 1 .. $max
2: $max = 100;
3: print ("1\n2\n"); # Cheating .. we know 1 and 2 are prime numbers !
4: for ($i = 3; $i <= $max; ++$i) {
5:     # skip if divisible by 2
6:     if ($i == int ($i/2) * 2) { #int () - truncates floating pt. no.
7:         next ;  # "continue"
8:     }
9:     $s = sqrt($i);
10:    if ($s == int ($s)) { next; }# If it is a perfect square, ignore
11:        $isPrime = 1;
12:    for ($j = 3; $j < $s; $j++) {
13:        if (int($i/$j) * $j == $i) {
14:            $isPrime = 0;
15:            last;  # "break"
16:        }
17:    }
18:    if ($isPrime) {print "$i \n";}
19:}
```

Data Types

- **Scalar**
  - **A scalar variable (single-valued variable) can contain string or a number**
  - **Prefixed by "$" –** `$x, $VariableNameCanBeAsLongAsYouWant`
- **Indexed Arrays / Lists**
  - **An <u>ordered</u> list of scalars (mixed string and numeric values)**
  - **Keys are integers (0 based)**
  - **Supports both array and list semantics**
  - **Prefixed by "@"**
- **Associative Arrays**
  - **An array of scalars (mixed string and numeric values)**
  - **Keys are strings. No concept of base, since no ordering.**
  - **Prefixed by "%"**
- **Barewords (no prefix symbol) – Treated as a string (scalar)**

Scalar data: Operations

- **Treated as a string or number depending on expression context**
  - **Numeric values stored internally as doubles**
  - **Strings are ascii, but Unicode if "use utf8" is invoked.**
- **Numeric Operations**
  - **Arithmetic operators:** `+ - / * += -= *= /= % ** ++ --`

```
$a = 123; $a++;
$b += 123; # Evaluated numerically
$b = $a + "456"; # "456"converted to number, and then added
```

  - **Math library functions:** log, exp, sin

```
$b = sin($a);   # or $b = sin $a;
```

  - **Comparison operators:** `==, <=, >=, <, >, <=>`

```
if ($a == 456) ... # Comparison evaluated numerically
$b = $a <=> 100;   # $b is -1, 0, or +1
```

String Operations

- **Concatenation – "."**
  - ♦ `$a = 123; $b = $a . "456"; # $b becomes "123456"`
- **String repetition – "x"**
  - ♦ `$a = "abc" x 3; # $a becomes "abcabcabc";`
- **String Comparison operators – eq, ne, lt, gt, le, ge, cmp**
  - ♦ `if ($a eq 456); # both sides converted to string before comparison`
- **Interpolation**
  - ♦ `$s = "and"; $str = "click $s clack"; # str => "click and clack"`
- **Extracting substrings: substr**

```
$a = substr("Hello World", 2, 5);  # $a gets "llo W"
```

  - ♦ **First character is at index 0.**
  - ♦ **Can be used as an "lvalue"**

```
substr ($str, 1, 4)  =  "abcde";
```

String Operations (contd.)

- **String indexing: index, rindex**

```
$a = index ("Hello World", "llo"); # $a gets a value of 2.
```

- **chop, chomp**
  - `chop ($a); # Chops and returns the last char in the string`
  - `chomp ($a); # eqvt. to chop($a) only if it is a '\n';`
- **Changing case**
  - `uc ($a); lc($a) #Converts entire string to upper(lower) case`
  - `$a = "abc\U$b\Edef";#\U operator upcases string until \E (or end)`
  - `ucfirst ($a);   #Upper cases the first character in $a. Or use \u`
  - **lc, lcfirst, or \L, \l operators for lower case**
- **Length of string**
  - `$x = length ($a);`

Basic Control Structures

- **Loops**
  - ```
    while ( condition is true ) {
          ....
    }
    ```
  - ```
    for ( initialization ;   condition   ;   increment )  {
          ....
    }
    ```
    - **identical to the "for"statement in C.**
  - **Condition == false if scalar is undefined or 0-length string or has value 0.**
- **Loop Terminators**
  - `next` **– same as "continue" in C.**
  - `last` **– same as "break" in C.**
- ```
  if ( ) {
  } elsif ( ... ) {
  } else {
  }
  ```
- **Always use curly braces**
  ```

Example : Reversing a string

```
1: $str = "gnimmargorP lreP";
2: $revStr = "";
3: while ($str) {
4:         $revStr .= chop ($str);
5: }
6: print "$revStr \n";
```

- **Remember the Perl motto : "There’s more than one way to do it"**

File I/O

- **Reading from files: `open`**

```
open (F, "/tmp/x");    # F is a filehandle - opened for read
open (FW, "> /tmp/x"); # open /tmp/x for writing
```

- **Reading from filehandle: `<filehandle>`**

```
$x = <F>; # reads one line at a time (with newline)
while ($x = <F>) {print $x};  # Types out entire file.
```

- **Writing: `print filehandle comma-separated-list`**

```
print FW "Name", "Sriram" # Or print FW ("Name", "Sriram")
printf FW "%d%s", $x, $y; # formatted OUTPUT. Note: no ","after FW
```

- **Closing**
  - `close (FW);`

- **STDIN, STDOUT, STDERR – standard filehandles**

Example: FIle I/O

```
1: if (!open (FW, "> /temp/myfile")) {
2:     print STDERR "Could not open file";
3:     exit(1);
4: }
5: $i = "Hello";
6: $j = 10;
7: print FW ("$i $j \n"); # or printf FW ("%s %d\n", $i, $j);
8: close(FW);
```

Exercise 1: Spell Checker

- **Write a spell checker**
- **Run it as perl spellcheck.pl**
- **Script should ask for the word to be checked.**
- **It should open the file "dictionary" to cross-check this word.**
- **The dictionary file has lots of valid words, one on each line, like this:**

```
Ababa
aback
abacus
abalone
abandon
abase
```

- **Case is not important.**
- **Dictionary is in sorted order.**

Opening pipes

- **Can set up command pipelines with open**

```
open (C, "/bin/cal 4 2001|");
while ($line = <C>) {
   print ">> $line";
}
close(C);
```

- **Output:**

```
>>     April 2001
>>   S  M Tu  W Th  F  S
>>   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 26 27 28
>>  29 30
>>
```

Perl debugger

- **Source code debugger**
  - **perl –d *script_file***
- **Commands**

<table>
<tr><th>Command</th><th>Description</th><th>Command</th><th>Description</th></tr>
<tr><td>s</td><td>Single step</td><td>p expr</td><td>evaluate and print perl expression</td></tr>
<tr><td>n</td><td>Single step, over subroutine calls</td><td>< command
> command</td><td>Define command before (or after) prompt</td></tr>
<tr><td>l [line] | [sub]</td><td>list line or subroutine</td><td>! number</td><td>history – redo cmd</td></tr>
<tr><td>b [line] [condition]</td><td>set breakpoints</td><td>command</td><td>Execute as perl stmt.</td></tr>
<tr><td>a [line] command</td><td>Set an action to be done before line is executed</td><td>/pattern/
?pattern?</td><td>Search fwd
Search bwd</td></tr>
</table>

On Regular Expressions

- **Problem: Search mail file for all subject lines containing "perl"**
  - **Pseudo code:**

```
open (F, "C:/eudora/in.mbx");
while ($line = <F>) {
     if ($line matches "Subject: ...junk ... perl") {
          print $line;
     }
}
```

    Note that you can’t use "`eq`**" in place of** *`matches`***.**

- **"Regular Expressions": Templates or patterns for string matching**
  - **Operators like** `eq`**,** `ne` **and functions such as** `substr`**,** `index` **do exact string comparisons.**
  - **Operators** `=~` **and** `!~` **do "matches" and "doesn’t match"**
  - **Regular expressions are used for fuzzy matches**

Regular Expressions: Contd.

- **Regular Expression meta-characters – standard ones**

<table>
<tr><td>^</td><td>At beginning of line</td><td>$</td><td>At end of line</td></tr>
<tr><td>.</td><td>Any character (except newline)</td><td>+</td><td>One or more of the previous character or expression</td></tr>
<tr><td>?</td><td>Zero or one of the previous character or expression.</td><td>[ ]</td><td>Match a set of characters</td></tr>
<tr><td>*</td><td>Zero or more of the previous character or expression</td><td>[^]</td><td>Match only if doesn’t belong to the set of characters specified</td></tr>
<tr><td>( )</td><td>Start a sub-expression</td><td>{m,n}</td><td>Matches m to n occurences of the previous character or expression.</td></tr>
<tr><td>|</td><td>Or</td><td>\</td><td>Treat next character as literal</td></tr>
</table>

-
```
if ($line =~ /^Subject:.*perl/) {
        print $line;
}
```

Perl additions

- **Pre-built sets**
  - **\b – Word boundary**
  - **\d, \D – any digit ([0-9]), not a digit ([^0-9])**
  - **\w, \W – alphanumeric character ([A-Za-z_0-9]), non alphanumeric character***
  - **\s – any visible space character (spaces, tabs, line-feeds)**
- **Patterns are treated as double quoted strings**
  - `if ($line =~ /\b$word\b/) {print "$word found"}`
- **Options – i, m, s, x**
  - **/i : Ignore case –** `if ($line =~ /\b$word\b/i) { .. }`
  - **/m : Treat string as multiple lines**
  - **/s : Treat string as single line**
  - **/x : Use extended regular expressions.**
  - **/o : Compile pattern once**

Regular Expressions: Memory

- **Use parentheses to extract portions of matched text**

```
if (!open (F, "Inbox")) {exit (1);}
while ($line = <F>) {
    if ($line =~ /^From: (\w+)\@weblogic.com/) {
        print $1, "\n";
    }

}
```

- **Extracted portions are available for use later in the same regex**

```
My red bat is red

My green ball is green

My red ball is red
```

  - ```
    if ($line =~ /^My (red|blue|green) (bat|ball) is \1/) {
         $color = $1; $object = $2;
    }
    ```
  - **\1 – \9 available within regular expressions; $1 – $9 outside**
    ```

Pattern Substitution

- **"If pattern exists in a string, substitute matched portion with another string"**
  - `$var =~ s/regex/substitute string/`
  - **Note that $var is modified, if its contents match regex**

- **Examples**

```
$str = "Mom & Pop";
$str =~ s/Pop/Me/ ; #$str becomes "Mom & Me"
$str =~ s/o[mp]/a/g; #global substitution "Ma & Pa";
$str =~ s/[mp]/b/gi; #global and ignore case -> "bob & bob"
$str =~ s/([A-z]+) & ([A-z]+)/$2 & $1/; #swap -> "Pop & Mom"
```

- **Additional substitution options**
  - **g – replace globally**
  - **e – Evaluate the right side as an expression**
  - **x – Extended regular expressions**

Translation

- **Supplying a character to character translation**
  - **tr**/*search list*/*replacement list*/*cds*
  - `$s = "zorba the greek"; $s =~ tr/gk/cp/; # "zorba the creep"`
  - `$s =~ tr/A-Z/a-z/; # lower case entire string`
  - `$s =~ tr/a-mn-z/n-za-m/ # Rot 13.`
- **The tr/// operator returns the count of translations**
  - `$count = ($s =~ tr/a-mn-z/n-za-m/);`
- **Options**
  - **/c : Complement the search list**
  - **/d : Delete found but unreplaced characters**
  - **/s : Squash duplicate replaced characters**

To summarize ...

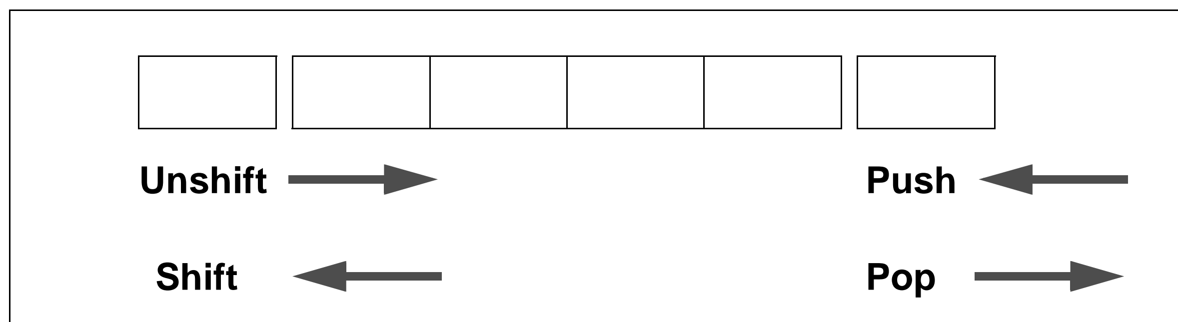
- **Scalar data types introduced**
- **Functions to manipulate strings of characters**
- **Regular Expressions**
- **Substitution**
- **File I/O, pipes**

Arrays of scalars

- **Arrays are multi-valued (as opposed to scalar data types)**
  - `@a = ("123", 3, 4.44455, '11122', $b);`
  - `@b = @a;`
- **Accessing one element – each element is a scalar**
  - **Starting index is 0.**
  - `$x = $a[2]; # $x gets 4.44455`
  - **Note the '$' in $a[2] – each element in an array is a scalar, hence the '$'**
  - **This is a scalar assignment – scalar to scalar**
- **Array slices – accessing multiple elements**
  - `@tempArray = @a [1, 2, 4]; # @tempArray gets (3, 4.44455, $b)`
  - `($a, $b, $c) = @a [1,2,5];  ($a, $b, $c) = ($d, 12, $e);`
- **Assigning to a scalar variable**
  - `$x = @a; # $x gets the count of elements.`
  - **This is a list to scalar assignment**

Arrays/Lists: Operations

- **Insertion, Removal of elements**
  - **push** (@aList, @bList); # Appends @bList to @aList
  - **push** (@aList, "123", 3333, 'a', $x);
  - **pop** (@aList); # removes and returns last element from @aList
  - **unshift** (@aList, "234 ", 4444); # Prepends list to @aList
  - **shift** (@aList); #removes and returns first element from @aList



- **Rearranging elements**
  - **sort** @aList; # Returns an alphabetically sorted list
  - **reverse** (@aList); # returns a new list with the order reversed

More control structures

- **foreach**
  - ```
    foreach $e (@mylist) {
         print $e;
    }
    ```
- **Equivalent constructs**
  - `print "DEBUG" if $debugging; # Note, no braces.`
  - `do { $x = $x**3 } while $x < 100;`
  - `until ($x > 100) { $x = $x**3 }  # eqvt. to "while (!condition)"`
  - `unless ($x < 10) { } # eqvt. to "if (!condition)"`
- **||, && as control structures**
  - `open (F, "/tmp/x") || die "no such file \n";`

    Second argument of "||"evaluated only if the first part returned false (a non-zero value)
- **There is no switch/case statement**
    ```

Associative Arrays

- **Hash table implementation**
  - **Stores both key and value. Key is always a string, and value can be any scalar.**

USA	Washington
India	New Delhi

  - **Variables identified by "%" prefix. Indexing operator is "{ }"**

- **Examples**
  - `$capitalOf{"USA"} = "Washington"; %copy = %capitalOf;`
  - `$populationOf{"India"} = 45;`
  - `$aa[123] = 45;` **# Gotcha – "aa " here is a list, not an associative array.**

- **Accessing one element – each element is a scalar**
  - `$x = $bb{"abc"};`
  - **This is a scalar assignment – scalar to scalar**

- **Hash slice: Accessing multiple elements at a time**
  - `@capitals = @capitalOf{"USA", "India"}; # Note @ prefix.`

Associative Arrays: Operations

- **keys**
  - `@list = keys (%array);`
- **values**
  - `@list = values (%array);`
  - `print (sort (values %array));`
- **each**
  - **`"each"` returns a (key, value) pair every time it is called. At the end it returns an empty list, and resets itself.**
  -
```
while (($key, $value) = each %myArray) {  #Note - list assignment
     print "$key: $value \n";
}
```

Conversion from one data type to another

- **Creating an array from a scalar and vice-versa**
  - **split: extracting arrays from scalars**

```
@a = split (/,\s+/, "abc, def, ghi"); # Comma separated list
($user, $password) = split (/:/, $line); # $line from /etc/passwd
```

  - **join: combining elements of an array to yield a scalar**

```
$s = join ("::", ("abc", "def", "ghi"));
$s = join ("::", @l);
```

- **An associative array from an array or list and vice-versa**
  - **Assigning list to associative array: each pair of values is treated as a key and a value.**

```
%capitalOf = ("USA", "Washington", "Colombia", "Bogota");
```

  - `@names = %capitalOf;  # "Flattens" the table in no particular order of keys.`

Context

- **Most functions work with lists and scalars.**
  - `chop (@foo); # Chops the last character of all elements in foo.`
- **The left hand side specifies a LIST or SCALAR context**
  - **Automatic conversion to required context**

```
open (F, "address.doc");
@list = <F>;    # @list gives the <> operator a list context, and
                # ends up slurping in the entire file, one row per
                # list element ($list[0] contains the first line)
close(F);
```

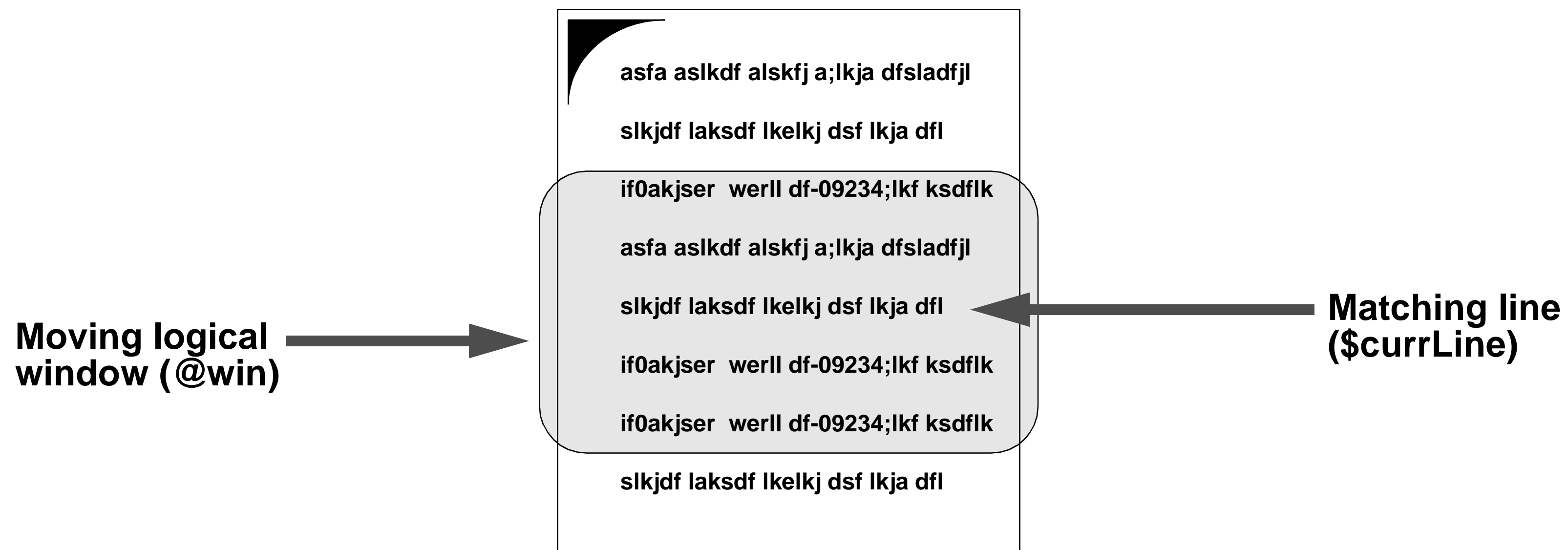
- **Sub contexts**
  - **Scalars can be treated as numbers, character strings, or binary strings depending upon the function.**

Functions

- **Declaring a function**
  - ```
    sub func {
          .....
    }
    ```
- **Calling a function**
  - `func(); myProc ($x, "hello"); # old style - &func;`
  - `myProc($x, "hello")`
- **Argument processing (inside a function)**
  - **Function always sees parameters passed to it in @_ array**
  - **That is, in $_[0], $_[1], $_[3] etc.**
- **Local Variables**
  - `my ($count); #lexical scoping`
  - `local (@tempArray); #dynamic scoping`
  - `my ($a, $b, @c) = @_; # Useful pattern to give names to args`
    ```

Example: Context grep

- **cgrep – Search for a regular expression in a file, and print it along with a couple of lines before and after it (context)**



Example: Context grep

```
1: use strict;
2: # This script is similar to egrep, and in addition to displaying all
3: # those lines that match, it displays a couple of lines above and below
4: # it (the context).Example: perl cgrep.pl -2 incl /usr/include/*.h
5: my (@win, $line, $fileName, $pattern);
6: my $cLines = 3;
7:
8: if ($ARGV[0] =~ /^-(\d*)/) {
9:     $cLines = $1;
10:    shift (@ARGV);
11:}
12:$pattern = shift @ARGV;
13:while ($fileName = shift @ARGV) {
14:    open (F, $fileName) || die("Could not open $fileName: $! \n");
15:    print ("=" x 70, "\n", $fileName, "\n", "=" x 70, "\n");
16:    # @win maintains the context: the previous $cLines, the current
17:    # line and the next couple of lines. The current line is always the
18:    # middle element of @win. Prime this array.
19:    my $i;
```

```
20:     for ($i = 0; $i < $cLines; $i++) {$win[$i] = "";}
21:     $win[$cLines] = <F>; # if cLines==2,$win[2] is first line
22:     for ($i = 0; $i < $cLines; $i++) {
23:         $line = <F>;
24:         push (@win, $line); # Prime $win[3],[4] with next lines
25:     }
26:     # Now, $win[2], [3],[ 4] have the first three lines of  the file.
27:     while ($line = $win[$cLines]) {
28:         if ($line =~ /$pattern/o) {
29:             #Print lines before the center line
30:             for ($i = 0; $i < $cLines; $i++) {print $win[$i];}
31:             chop($line); # Print the center line
32:             print("$line <<<<<<<<<<<<<<<<<<<<<<<<\n");
33:             #Print lines after the center line
34:             for ($i = 1 ; $i <= $cLines; $i++) {
35:                 print $win[$cLines+$i];
36:             }
37:             print "-" x 70, "\n";
38:         }
39:         shift (@win); #Move top line out of window ...
40:         $line = <F>;
41:         if (defined($line)) {
```

```
42:             push (@win, $line); # ... and append new line from file
43:         } else {
44:             push (@win, "");
45:         }
46:     }
47:     close (F);
48:}
```

- **Sample Output**

```
C:\temp>perl -w cgrep.pl -2 chop cgrep.pl
==============================================================
cgrep.pl
==============================================================
      Print lines before the center line
     for ($i = 0; $i < $cLines; $i++) {print $win[$i];}
      chop($line); # Print the center line<<<<<<<<<<<<<<<<<<
      print("$line <<<<<<<<<<<<<<<<<<<<<<<<\n");
      #Print lines after the center line
--------------------------------------------------------------
```

Gotchas

- **Scalar vs. Array – Using "$"and "@"properly.**
- **Strings and Numbers – comparison operators**
  - **String comparisons – eq, ne, le, etc.**
  - **Numeric comparisons – ==, <, > etc.**
- **Always run "perl -w" on new scripts.**
  - **Will catch reads from uninitialized variables**

Exercise 2 (5 minutes)

- **Given a (tab separated) TV trivia file,**

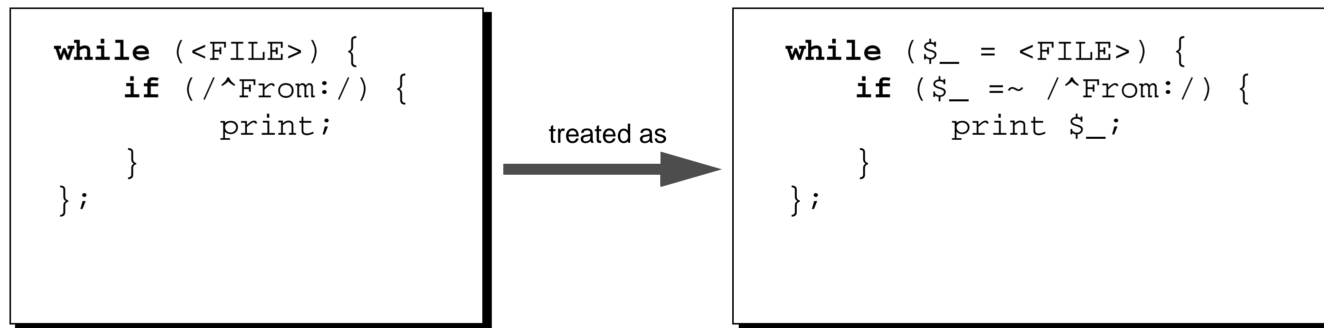
```
Jason Alexander         Seinfeld
Alex Trebek             Jeopardy
Lisa Kudrow             Friends
```

sort by last name and print.

Perl Programming: The next few steps

Built-in/Default Variables

- **$_**
  - **Most Perl operations either operate on, and/or return "$_", by default**



- **@ARGV – arguments passed to the script**
  - **Access as $ARGV[0], $ARGV[1] etc.**
- **%ENV – array of environment variables.**
  - `$p = $ENV {"PATH"};`
- **@_**

Quoting

- **Double quotes**
  - **Variables expanded, and special meaning given to \n, \t, \r etc.**
  - `print "Total Quantity = $qty \n"; # $qty interpolated`
- **Single quotes**
  - **No expansion of variables ("interpolation"), and no special meaning**
- **Back quotes**
  - **Executes programs and captures their output:** `$x = `/bin/ls` # $x gets a list of files`
  - **Back quotes treated like double quotes**

Quoting: Contd.

- **Using other quoting symbols for regular expressions, and substitution expressions**
  - **Can use `m"From:"` instead of** `/From:/`
  - **For substitutions, the first character after "s" is taken as the delimiter - `s/xxx/yyy/` can be written as `s#xxx#yyy#`**
  - **Typically, you'd choose a character that isn't likely to occur in the regular expression.**
- **Quoting words.**
  - **Instead of saying** `('goodbye', 'cruel', 'world', '@#$#$#')`**, you could say** `qw(goodbye cruel world @#$#$#)`
  - **qw extracts words out of a line, with white-space characters as delimiters**

Example: Quoting

```
1: # This code shows the different types of quoting mechanisms
2: # It prints out an indented calendar, btw ..
3: $dt = `date`; # date prints out, say, "Tue Feb  6 21:29:08 PST 1996"
4: chop ($dt);
5: @splitDt = split (m{ +},$dt);
6: $year = $splitDt[5]; #Extract the last word - "1996"
7: $header = "This is the calendar for $year ....\n";
8:
9: # Trying to act smart here ! Capture the output of `cal 1996`
10:# into $cal_output, then globally substitute the first character
11:# of each line by 10 spaces + that character.
12:$cal_output = $header . `cal $year`; # Capturing cal's output
13:# Indent every line by 30 spaces
14:$indent = " " x 10;
15:# Note the 'm' option to the substitute
16:$cal_output =~ s|^(.)|$indent$1|mg; # "|" used as delimiting symbol
17:print $cal_output;
```

Argument Processing

- **Arguments to a perl script in the @ARGV array.**
- **Processing the @ARGV array**
  - ```
    foreach $a (@ARGV) {
         ....
    }
    ```
  - **The `shift` operator shifts the ARGV array by default (or @_ inside a sub)**
- **<> operator**
  - **Equivalent to treating all arguments as files, and reading them one after another**
    - ```
      while ($a = <>) { # a simple "cat" equivalent
          print $a;
      }
      ```
  - **Defaults to STDIN**
  - **Beware of array contexts – all files slurped in**
    - ```
      @a = <>; # Check if you really want to do this
      ```
  - **Built-in filehandle ARGV set to currently open file**
      ```

File processing

- **File tests**
  - ```
    if (-e "/etc/passwd") { ..} # File /etc/passwd exists?
    ```
  - ```
    $a = (-d $x) && (-w "$x/$y"); # =1, if $x is a dir and
                                  # $x/$y is writable
    ```
- **stat**
- **chown, chmod, unlink, rename ..**
- **filename wild cards.**
  - ```
    @files = <*.bak>; #  <> syntax overloaded
    ```
  - ```
    unlink <*.o>; # <> works as a wild-expander where it counts.
    ```

Directory Access

- **opendir ()**

```
opendir (D, "/tmp") || die "No such directory\n";
```

- **readdir ()**

```
while ($f = readdir(D)) {
    if (-d $f) {
        print "$f    .... Directory \n";
    } else {
        print "$f \n";
    }
}
```

- **mkdir, chdir**

```
$f = "/tmp/xx";
mkdir ($f);
chdir ($f);
```

Example: Pretty printed directory tree

```
1: # This script prints out a directory tree
2: # Invoke as "perl dirtree.pl [directory name]
3: $dirName = ".";
4: if (@ARGV) {
5:     $dirName = shift @ARGV;
6: }
7:
8: $currLevel = 0; # current indentation level
9: PrintDir ($dirName);
10:
11:sub PrintDir {
12:    my ($dirName) = @_;
13:    my (@files);
14:
15:    $spaces = " " x ($currLevel * 4); # Spacing for indenting
16:    print ("$spaces$dirName\n");
17:
18:    $spaces = " " x (++$currLevel * 4);
19:    if (!opendir (D, $dirName)) {
```

Example: Pretty printed directory tree (contd.)

```
20:         warn "Problem with \"$dirName\": $! \n";
21:         --$currLevel;
22:         return 1;
23:     }
24:     @files = readdir(D);
25:     close(D);
26:     foreach $f (@files) {
27:         if (($f eq ".") || ($f eq "..")) {
28:             next;
29:         }
30:         if (-d "$dirName/$f"){
31:             PrintDir ("$dirName/$f");  # Recursive call
32:         } else {
33:             print ("$spaces$f\n");
34:         }
35:     }
36:     --$currLevel;
37:     return 0;
38:}
```

Run-time code – Using eval

- **Used to produce code on-the-fly, and execute it**
- `eval $str` **– Treating $str as a little perl program**
- **Examples**
  - Simple example:

```
$str = '$c = $a + $b'; #Note single quotes
$a = 10; $b = 20;
eval $str;
print $c;
```

  - **Evaluating perl one-liners**

```
while ($line = <>) {
     eval $line;
     if ($@) {print $@};
}
```

Example: Using eval to rename multiple files

- **Usage: rename.pl perl-expression [files]**

```
1: #!/opt/bin/perl
2: $code = shift (@ARGV);
3: foreach $name (@ARGV) {
4:     $oldname = $name;
5:     # This is the heart of the script - a perl expression
6:     # is being supplied at run time to be evaluated
7:     eval ($code);
8:     if ($oldname ne $name) {
9:         rename ($oldname, $name);
10:    }
11:}
```

- **Usage: (command line invocation)**
  - `rename.pl '$name =~ s/\.c$/.c.bak/' *`
  - ```
    rename.pl 'print "$name --> ${name}.bak \n"; \
                          $name = $name . ".bak" ; ' *
    ```
    ```

eval (continued)

- **Rewriting "rename" – taking advantage of "$_"**

```
1: $code = shift (@ARGV);
2: die "Usage: rename perl-expression [files]\n" if ($code eq "");
3: foreach (@ARGV) {
4:     $oldname = $_;
5:     eval ($code);
6:     die $@ if $@;
7:     rename ($oldname, $_) unless ($oldname eq $_);
8: }
```

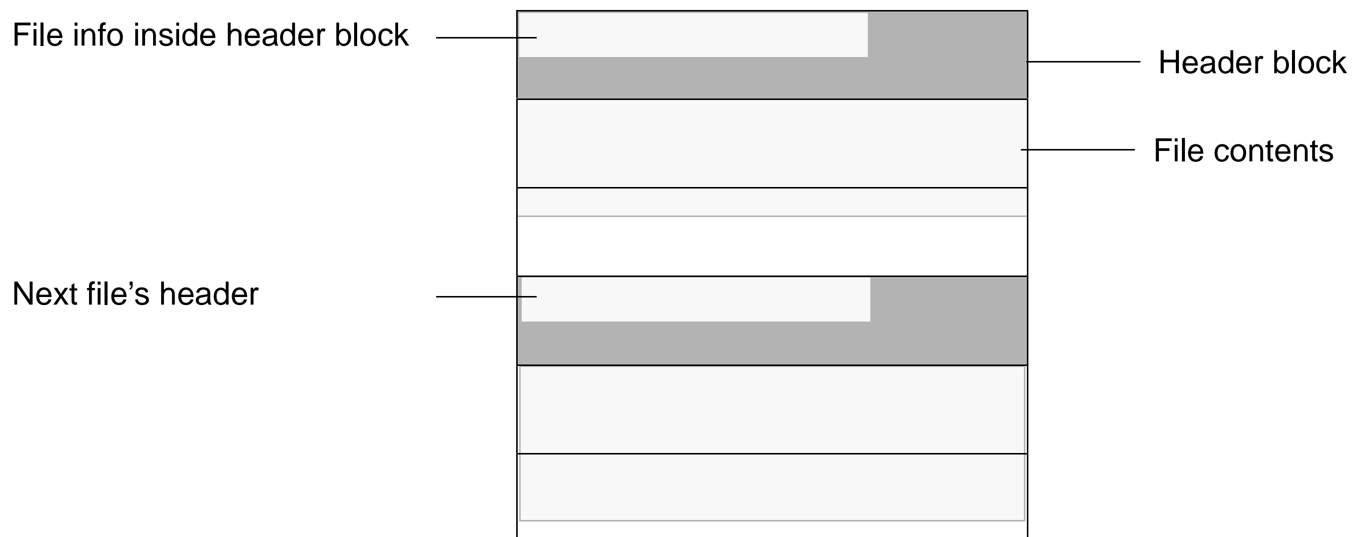
- **Usage simpler for simple operations**
  - `perl rename.pl "s/.c/.c.bak/" *`
- **Error checking with special variables.**
  - **$@ contains a compilation error, if an invalid perl expression was supplied.**

Binary Data

- **perl strings can take binary values**
- **pack, unpack**
  - **Similar to sprintf except used for binary strings**
  - ```
    $a = -10; $b = "Hello"; $c = 4.333;
    $d = pack ('i a10 d', $a, $b, $c);
    ($a, $b, $c) = unpack ('i  a10 d', $d);
    ```
  - **Each template element gobbles up one value**
- **Exercise: Contrast between split/join, pack/unpack, sprintf**
    ```

Example: Modifying tar files

```
csh> tar cf  x.tar  a.c  b.c  g.h
```



Each block is 512 bytes long

Each header gets its own block. The header information is binary.

The file contents get as many integral number of blocks as required

Example: Reading a tar file header (contd.)

```
struct tarheader {
    char name[100];
    char mode[8];
    char uid[8];
    char gid[8];
    char size[12];
    char mtime[12];
    char chksum[8];
    char linkflag;
    char linkname[100];
} dbuf;
```

```
(   $name,
    $mode,
    $uid,
    $gid,
    $size,
    $mtime,
    $chksum,
    $linkflag,
    $linkname) =
unpack ("A100 a8 a8 a12 a12 a8 a1 a100",
$data);
```

Example : tarmodify (contd.)

- **Sample usage – append .bak to every file**

```
csh> ls /tmp
test1  test2 test3
csh> tar cf x.tar test*
csh> tarmodify '-s/$/.bak/' < x.tar > y.tar
```

- **Change "test1" to "1.t" inside a tar file**

```
csh> tarmodify  '-s/test(\d+)/\1.t/' < x.tar > y.tar
csh> tar tf y.tar
1.t
2.t
3.t
```

Example : tarmodify (contd.)

```
1: ##########################################################################
2: # tm - A script to modify names inside tar files.                        #
3: # Usage:                                                                 #
4: #   tm -s<Perl substiution command>   < input.tar > output.tar           #
5: # Examples :                                                             #
6: #  tm -s/.c$/.c.bak/ < x.tar > y.tar (Renames all .c files to.c.bak     #
7: #  tm -s#^/## (Makes all absolute path names relative, by kicking out    #
8: #                  the leading "/".                                      #
9: # Note:                                                                  #
10:#  The first character after -s is used as the delimiter.                #
11:#  Anything that can be done in a perl substitution expression can be    #
12:#  done (such as ignoring case) etc.                                     #
13:##########################################################################
$BLOCKSIZE = 512;
14:if (@ARGV != 1) {
15:    Usage();
16:}
17:foreach (@ARGV) {
18:    if (/^-h/) {
```

```
19:        Usage();
20:    } elsif (/^-s(.*)/) {
21:        $cmd = $1;
22:    }
23:}
24:Usage() if ($cmd eq "");
25:while (($n = ProcessFileHeader()) != -1){
26:    for ($i = 0; $i < $n; $i++) {
27:        ReadBlock();
28:        WriteBlock();
29:    }
30:}
31:sub ProcessFileHeader {
32:    my $n;
33:    if (!(ReadBlock())) {
34:        return -1;
35:    }
36:    ($name, $mode, $uid, $gid, $size, $mtime, $chksum,
      $linkflag, $linkname) =
37:         unpack ('A100 a8 a8 a8 a12 a12 a8 a1 a100', $data);
38:    if ($name eq "") {
39:        WriteBlock();
```

```
40:         return 1; #One more null block to follow.
41:     }
42:     PrintHeader() if $debugging;
43:     if (eval("\$name =~ s$cmd")) {
44:         substr ($data, 0, 100) = pack ("a100", $name); # Replace name
45:         #recalculate Checksum;
46:         $s = sprintf ("%lo", CheckSum());
47:         substr ($data, 148, 8) = pack ('a7 x', $s);
48:     }
49:     $n = int (oct($size) / $BLOCKSIZE);
50:     (++$n) if (oct($size) % $BLOCKSIZE);
51:     WriteBlock();
52:     return $n;
53:}
54:sub ReadBlock {
55:     return sysread (STDIN, $data, $BLOCKSIZE);
56:}
57:sub CheckSum {
58:     substr ($data, 148, 8) = " " x 8; #Blank out checksum
59:     @byteArray = unpack("c*", $data);
60:     $sum = 0;
61:     foreach (@byteArray) {
```

```
62:         $sum += $_;
63:     }
64:     return $sum;
65:}
66:sub WriteBlock {
67:     my ($l);
68:     $l = length ($data);
69:     if ($l) {
70:         die "LENGTH OF BLOCK ($l) != $BLOCKSIZE"
71:             if ($l != $BLOCKSIZE) ;
72:         return syswrite(STDOUT, $data, $BLOCKSIZE);
73:     }
74:     return 0;
75:}
76:sub Usage {
77:     print STDERR "\n\n Usage:\n\t$0 -s/from pattern/to pattern/\n";
78:     print STDERR "\t(Any character can be used in place of '/')\n\n";
79:     exit (1);
80:}
81:sub PrintHeader {
82:     print STDERR "name     = $name \n";
83:     print STDERR "mode     = $mode \n";
```

```
84:     print STDERR "uid      = $uid \n";
85:     print STDERR "gid      = $gid \n";
86:     print STDERR "size     = $size \n";
87:     print STDERR "mtime    = $mtime \n";
88:     print STDERR "chksum   = $chksum \n";
89:     print STDERR "linkflag = $linkflag \n";
90:     print STDERR "linkname = $linkname \n";
91:}
```

Command line options

- **Some important command line options**
  - **-v** print out version
  - **-w** issue warnings about error-prone constructs
  - **-d** run script under debugger
  - **-e** for single line expressions
  - **-n** loop around input, like sed
  - **-p** same as above, but print out each line
  - **-i** edit in place
  - **-a** autosplit every input line

Example: pgrep

```
1: # Usage: pgrep <regular expression> - finds files in path which match
2: # the given regular expression. This script only works on Unix.
3:
4: $regexp = shift (@ARGV) || die "usage: $0 regexp\n";
5:
6: # The PATH environment variable has path names separated by a ":".
7: foreach $dir (split(/:/,$ENV{'PATH'})) {
8:     #chdir returns 1 on success. If not successful, continue.
9:     if (! chdir($dir)) { next; }
10:    foreach $f (<*>) {
11:        # -f : "is a file", -x : "is an executable"
12:        if (($f =~ /$regexp/o) && (-f $f) && (-x $f))  {
13:            print "$dir/$f\n";
14:        }
15:    }
16:}
```

pgrep (contd.)

- **Sample output**

```
csh> pgrep man

/bin/catman
/bin/man
/opt/X11R5PL25/bin/xman
/usr/openwin/bin/timemanp
/usr/openwin/bin/xman
```

- **Exercise: Deal with relative paths in PATH.**

Example: today

- **Sample output**

```
csh> today

   March 1999             April 1999              May 1999
 S  M Tu  W Th  F  S   S  M Tu  W Th  F  S    S  M Tu  W Th  F  S
    1  2  3  4  5  6               1  2  3                      1
 7  8  9 10 11 12 13   4  5  6  7  8  9 10    2  3  4  5  6  7  8
14 15 16 17 18 19 20  11 12 13 14 15 16 17    9 10 11 12 13 14 15
21 22 23 24 25 26 27  18 19 20 21 22 23 24   16 17 18 19 20 21 22
28 29 30 31           25 26 27 28 29 30      23 24 25 26 27 28 29
```

Example: today (contd.)

```
1: # today - print out last month, this month, and next month from
2: #         cal program, with today in reverse video (vt100 hardcoded)
3: # demonstrates how to get a data feed from more than one process
4: $SO = "\033[7m"; #VT 100 encoding for bold start
5: $SE = "\033[m";  #VT 100 encoding for bold end
6:
7: ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
           localtime (time);
8: $mon++;                    # month is 0 based
9: $year += 1900;             # Possible Y2k hole !!
10:# select next year and prev year "wrapping"at 12 months
11:$nmon = $mon + 1; $nyear = $year;
12:if ( $nmon == 13) { $nmon = 1; $nyear++;}
13:$pmon = $mon - 1; $pyear = $year;
14:if ( $pmon ==  0) { $pmon = 12; $pyear--;}
15:
```

Example: today (contd).

```
16:#start three cal processes for each month
17:open ( PREV, "cal $pmon $pyear |");
18:open ( CUR, "cal $mon $year |");
19:open ( FOLL, "cal $nmon $nyear |");
20:
21:while (! ((eof(PREV) && eof(CUR) && eof(FOLL)))) {
22:    chop($prev = <PREV>);
23:    chop($foll = <FOLL>);
24:    chop($cur = <CUR>);
25:    $len = 22;
26:    if ($cur =~ s/\b$mday\b/$SO$mday$SE/) {# hilite today's date
27:        $len += 7;  # Accounting for the highlight escape characters.
28:    }
29:    # printf format depends on len .. so generate the format first.
30:    $fmt = sprintf("%%-20s  %%-%ds  %%-20s\n", $len);
31:    printf ($fmt, $prev, $cur, $foll);
32:}
```

Example: makedepend

- **Recursively prints out all include file dependencies in C/C++ files**
  - **File main.c:**
    ```
    #include <termio.h>
    #include <main.h>
    ```
  - `makedepend main.c` **gives**

    ```
    /usr/include/termio.h /usr/include/sys/ioccom.h /usr/include/sys/
    termios.h /usr/include/sys/stdtypes.h /usr/include/sys/ttydev.h /
    usr/include/sys/ttycom.h main.h
    ```
- **Does not handle ifdef'ed sections**
- **Handles exclude patterns**
  - `makedepend -Etty main.c` **gives**

    ```
    /usr/include/termio.h /usr/include/sys/ioccom.h /usr/include/sys/
    termios.h /usr/include/sys/stdtypes.h main.h
    ```

Example: makedepend (continued)

```
1: $debugging = 0; # Set it to 1, to get debug output
2: foreach (@ARGV) {
3:     if (/^-I(.*)/) {
4:         push(@includeDirs, $1."/"); # list of include directories
5:     } elsif (/^-E(.*)/) {
6:         push(@excludePats, $1); # list of exclude patterns
7:     } else {
8:         push (@sourceFiles, $_); #list of source files
9:     }
10:}
11:#add default dir. to list of include dirs
12:push(@includeDirs, "/usr/include/");
13:foreach $f (@sourceFiles) {
14:    if (! -e $f) {
15:        print STDERR "$f does not exist\n"; next;
16:    } else {
17:        print ParseIncludes($f), "\n";
18:    }
```

Example: makedepend (continued)

```
19:}
20:
21:sub ParseIncludes {
22:    my($f) = @_; #This routine parses the file in $f
23:    my (@buf);
24:    my ($retval);
25:    $retval = "";
26:    $indentLevel++; # for printing nested levels of include files
27:    print STDERR "  " x $indentLevel, "Parsing $f\n"if $debugging;
28:    open (F, $f);
29:    @buf = <F>; # Read the entire file in ...
30:    close (F);
31:    foreach $line (@buf) {
32:        # extract the file name from a line like #include <file.h>
33:        if ($line =~ /^\s*#include\s*[<"](.*)[>"]/o) { # Note the "/o"
34:            $found = 0;
35:            $inc = $1;
36:            # the pathname is absolute if it exists as-is.
37:            if (-e ($fullPathName = $inc)) {
```

Example: makedepend (continued)

```
38:                     $found = 1;
39:             } else {
40:                 #relative Pathname - prepend each include dir. one by
41:                # one and check if the full path name exists.
42:                 foreach $dirs (@includeDirs) {
43:                     if ( -e ($fullPathName = $dirs.$inc)) {
44:                         $found = 1;
45:                         last;
46:                     }
47:                }
48:             }
49:             if (!$found) {
50:                 print STDERR "$inc not found\n";
51:             } else {
52:                 $exclude = 0;
53:                 #Should this filename be excluded ?
54:                 foreach $e (@excludePats) {
55:                    if ($fullPathName =~ /$e/) {
56:                     $exclude = 1;
```

Example: makedepend (continued)

```
57:                         last;
58:                     }
59:                 } #foreach
60:                 if (!$exclude) {
61:                     if (!($fileAlreadyAnalysed{$fullPathName})) {
62:                         $fileAlreadyAnalysed{$fullPathName} = 1;
63:                         $retval .= $fullPathName . " " .
64:                                        ParseIncludes($fullPathName);
65:                     } else {
66:                         print STDERR "   " x ($indentLevel+1),
67:                           "$fullPathName already seen\n"if $debugging;
68:                     }
69:                 }
70:             }
71:         }
72:     }
73:     $indentLevel--;
74:     return $retval;
75:}
```

Example: makedepend (continued)

- **What next?**
  - **Support ifdef, ifndef. Support all standard makedepend options – –**
  - **Output lines should not exceed 80 characters**

Answer for Exercise 1 : A spell checker

```
1: print ("What word do you want to verify ?   ");
2: $word = <STDIN>;
3: $lcword = lc($word);
4: if (!open (F, "dictionary")) { # Use /usr/dict/words on unix sytems
5:     print ("Dictionary not found \n");
6:     exit(1);
7: }
8: while ($dictWord = <F>) {
9:     $result = ($lcword cmp lc($dictWord));
10:    if ($result == 0) {
11:        chomp($word);
12:        print ("'$word' is spelled correctly \n");
13:        exit(0);
14:    } elsif ($result == -1) {
15:        last; # dictionary is in sorted order; no need to check further
16:    }
17:}
18:chomp($word);
19:print ("'$word' does not exist in the dictionary \n");
```