**Perl Part 2 Data Structures**

|  |  |
| --- | --- |
| **Command Argument Lists**  **@ARGV** Perl receives arguments in the @ARGV array.  **$ARGV[***subscript*] $ARGV[0] is the first argument in that array.  **$#ARGV** Contains the subscript of the highest populated element of @ARGV.  **<ARGV>** References a file handle which can be used to sequentially read all the files associated with @ARGV.  Note:  **$\_** Set by many commands and is the default when not specified. It is similar to $0 in awk.    Personally, I prefer more **explicit** variable name usage, but since this has wide usage, it is presented here.  **$ARGV** When processing input from multiple files as shown to the right, you may need to reference the current file. With this type of loop $ARGV has the name of the current file. | **Example 2-1**: Read and print all the contents of the files referenced on the command line.  $ vi catAll  #!/usr/bin/perl -w  die "no files specified\n" if ($#ARGV < 0);  while ( $\_ = <ARGV> )  {  print $\_;  }  $ perl catAll junk1 junk2 junk4  Inside junk1 ... line1  Inside junk1 ... line2  Inside junk2 ... line1  Inside junk2 ... line2  Inside junk2 ... line3  Inside junk4 ... line1  Inside junk4 ... line2  Inside junk4 ... line3  # Same as above, even the print can default to $\_  # Personally, I would rather be more explicit.  $ vi catAllv2  #!/usr/bin/perl -w  die "no files specified\n" if ($#ARGV < 0);  while ( <ARGV> ) {  print;  } |
| **More array processing**  **shift** @*array* returns the first element of an array and shifts the array downward.  **push** @*array,value* pushes (appends) a value onto the end of an array.  **join** *sep,*@*array* joins a list of values into a single string with elements separated by *sep*.  **sort** @*array* returns a new list from the sorted values of the array  **grep** *pat*,@*array* returns a new list containing elements that matched the pattern. The pattern can also be file test expressions.  **reverse** @*array* returns a new list with the values in reverse order  Assuming @pets = ('cat', 'dog', 'bird', 'bat'):   |  |  | | --- | --- | | **Operation** | **Returned Value** | | sort @pets | ('bat', 'bird', 'cat', 'dog') | | grep(/at/, @pets) | ('cat', 'bat') | | join(':', @pets) | 'cat:dog:bird:bat' | | reverse @pets | ('bat', 'bird', 'dog', 'cat') |   Effect on @pets:   |  |  | | --- | --- | | **Operation** | **Effect on @pets** | | push @pets,"fish" | ('cat', 'dog', 'bird', 'bat', 'fish') | | shift @pets | ('dog', 'bird', 'bat') | | **Example 2-2**: rotate the command arguments to the left  $ vi rotateL  #!/usr/bin/perl  die "no arguments to rotate\n" if ($#ARGV < 0);  $save = shift @ARGV;  push @ARGV, $save;  print "@ARGV\n";  $ perl rotateL one two three  two three one  **Example 2-3**: Sorts the command arguments and print the name of each on a separate line  $ vi example2-3  #!/usr/bin/perl  $txt = join("\n", sort(@ARGV));  print "$txt\n";  $ perl example2-3 one two three four  four  one  three  two  **Example 2-4**: same as 2-3, but only print valid text files in reverse sorted order.  We can use grep(-f && -t, @ARGV) to get only valid text files.  $ vi example2-4  #!/usr/bin/perl  print join("\n", reverse (sort(grep(-f && -t, @ARGV)))), "\n";  $ perl example2-4 junk1 junk3 junk4 junk2  junk4  junk2  junk1 |
| **sort**  The **sort** function sorts the contents of a list using a specified comparison returning the sorted result. It does not modify the list.  sort *comparison list*  If *comparison* is not included, it sorts in ascending order.  The examples show popular forms of the *comparison.* | **Example 2-5**: sort an array in ascending, descending, and ignoring case  $ vi example2-5  my @colors = ("red", "Orange", "yellow", "green", "blue");  my @sorted;  print "Original:\t@colors \n";  @sorted = sort @colors; # defaults to ascending  print "Ascending:\t@sorted \n";  @sorted = sort {$a cmp $b} @colors; # explicitly ascending  print "Ascending:\t@sorted \n";  @sorted = sort {$b cmp $a} @colors; # explicitly descending  print "Descending:\t@sorted \n";  @sorted = sort {lc($a) cmp lc($b)} # compared as lower case  @colors; # ascending  print "Ignore case ascending:\t@sorted \n";  $ example2-5  **Output:**  Original: red Orange yellow green blue  Ascending: Orange blue green red yellow  Ascending: Orange blue green red yellow  Descending: yellow red green blue Orange  Ignore case ascending: blue green Orange red yellow |
| **grep**  The **grep** function provides some of the power of the grep utility to perl.  **grep** (*expression*, *list*)  **grep** iterates through the list:   * Assigns an element to $\_ * Executes *expression*. * Includes the result in a resulting list.   **grep** {*codeBlock*} *list*  **grep** iterates through the list:   * Assigns an element to $\_ * Executes the *codeBlock.* * Includes the result in a resulting list. | **Example 2-6**: use of Perl grep  $ vi example2-6  #!/usr/bin/perl -w  use strict;  my @words = ("grep", "can", "match", "regular", "expressions", "strings", "iterator", "anything");  print "Original Words\n\t", join(':', @words), "\n";  # The following statement finds the long words.  my @longWords = grep {(length($\_) > 6)} @words;  print "Long Words:\n\t", join(':',@longWords), "\n";  # We can apply a pattern to produce a list of the elements that  # matched the pattern.  my @threeConsonantLongWords = grep (/[^aeiou]{3}/, @longWords);  print "Words containing three adjacent consonants:\n\t";  print join(':', @threeConsonantLongWords), "\n";  # Suppose our command arguments is a list of files. The following grep  # includes a file in the result if it is a text file.  print "ARGV:\n\t";  print join(':', @ARGV), "\n";  my @textFiles = grep (-f && -T, @ARGV);  print "Text files:\n\t";  print join(':', @textFiles), "\n";  $ mkdir Djunk  $ perl example2-6 file\* Djunk  Original Words:  grep:can:match:regular:expressions:strings:iterator:anything  Long Words:  regular:expressions:strings:iterator:anything  Words containing three adjacent consonants:  expressions:strings:anything  ARGV:  file1:file4:file5:Djunk  Text files:  file1:file4:file5 |
| **Printing Lists**  Assume @list = ('a', 'b', 'c', 'd');   |  |  |  | | --- | --- | --- | | Statement | Output | Newline? | | print "@list\n" | a b c d | yes | | print @list; | abcd | no | | print '@list\n'; | @list\n | no | | print join("\n",@list), "\n"; | a  b  c  d | yes | |  |
| **splice**  The splice function removes or replaces a subset of an array.  **splice** (*array, offset, count*)  removes *count* elements from the specified array beginning with *offset.*  **splice** (*array, offset, count, replaceList*)  replaces *count* elements from the specified array beginning with *offset* with the elements in *replaceList.* | **Example 2-7**: use of splice  $ vi example2-7  #!/usr/bin/perl -w  @fruitM = ('apple', 'orange', 'banana', 'grape', 'pear');  splice(@fruitM, 2, 2);  print "Remove 2 2:", join(":", @fruitM, "\n");  @fruitM = ('apple', 'orange', 'banana', 'grape', 'pear');  splice(@fruitM, 2, 2, ("clark", "maynard", "slavin"));  print "Replace:", join(":", @fruitM, "\n");  $ perl example2-7  Remove 2 2:apple:orange:pear:  Replace:apple:orange:clark:maynard:slavin:pear: |
| **Hash Array**  A hash array in Perl provides a lookup capability based on a key value. It is declared using a **%**. Instead of using square brackets "[*subscript*]" to reference an element, hash array elements are referenced using curly braces "{*key*}".  To reference the value for a key, use:  **$***hashArray***{***key***}**  Some special values:  keys %*hashArray* - list of the **keys**  values %*hashArray* - list of the **values**  Since a hash array hashes to get to the values, the list of keys or values will not be returned in key order.  If you reference a hash array element that doesn't exist, you do not get a runtime error; instead, it returns **undef** which is treated as either an empty string or zero. | **Example 2-8**: hash array for getting long form of month from abbreviation  $ vi example2-8  #!/usr/bin/perl -w  my %monthH = (  "jan" => "January",  "feb" => "February",  "mar" => "March",  "apr" => "April",  "may" => "May",  "jun" => "June",  "jul" => "July",  "aug" => "Auguest",  "sep" => "September",  "oct" => "October",  "nov" => "November",  "dec" => "December"  );  my $one = "feb";  print "Long form of $one is $monthH{$one}", "\n";  $monthH{"aug"} = "August";  my @keyArray = keys %monthH;  print "keys are @keyArray\n";  my @valArray = values %monthH;  print "values are @valArray\n";  $ perl example2-8  Long form of feb is February  keys are jul oct apr jan mar nov feb aug sep dec jun may  values are July October April January March November February August September December June May |
| **Hash Arrays Built-in Perl**  **%ENV** contains the current environment variables.  **%SIG** contains the current signal actions (i.e., signal handling subroutines). For example, when you press CTRL-C, a SIGINT (signal interrupt) is signaled. The default action is to interrupt your program, causing it to abort. |  |
| **Exercise:** given a list of files as input, count the occurrence of each word in all the files. Show the words in sorted ascending order. (See the output to the right.)  Do we know how to read the text from multiple files as command arguments? Look at example 2-1  How do we get the words? Look at example 1-6 | $ vi exwc.pl  #!/usr/bin/perl -w  my %wcH;  die “no files specified\n” if ($#ARGV < 0); // will be on exam while ( $line = <ARGV> ) {  my @words = split (/[ ,.!?:]/, $line);  foreach $word (@words) {  $wcH{ lc $word } += 1 if $word ne “”;  } }  my $word;  foreach $word (sort keys %wcH) {  print “$word: $wcH{$word}\n; }  $ perl exwc.pl file4 file5  a: 6  and: 1  bad: 2  bought: 1  but: 1  cat: 4  catastrophe: 1  cats: 1  dogs: 1  fun: 1  get: 1  getting: 1  had: 1  he: 1  hoped: 1  i: 3  in: 1  it: 3  john: 1  nap: 1  needed: 1  poodle: 1  raining: 1  really: 1  smelly: 1  so: 2  stepped: 1  that: 2  the: 1  tired: 1  to: 1  very: 2  was: 5  weather: 1  what: 1 |
|  |  |

**©2017 Larry W. Clark,** UTSA CS students may make copies for their personal use