Programming in Perl
Lecture 1 Examples

A full list of these BACKSLASH ESCAPEs may be found in *Learning Perl*, p. 34. Some examples:

```
"\a"     Alarm(Bell)
"\n"     Newline
"\t"     Tab

    Various Perl escape sequences can alter the case of subsequent
strings.

"\l"  The following character is converted to lowercase
"\u"  The following character is converted to uppercase
"\L"  All subsequent characters are converted to lowercase
      up to a \E character
"\U"  All subsequent characters are converted to uppercase
      up to a \E character
"\Q"  All non-alphanumeric characters are prepended with a backslash
"\E"  Used with \L, \U, and \Q
```

Escaping `$` and `@`:

```
print "The price is $100.\n";
The price is .<newline>
# Perl assumes $100 is a scalar variable with a default value of
# "".

print "The price is \$100.\n";
The price is $100.<newline>

print "The price is $", 100, ".\n";
The price is $100.<newline>
# Doesn't immediately preceed a valid variable name.
```

Precedence of operators discussed so far (full table in *Learning Perl*, pp. 38-39):

```
Assoc         Operator
-----         --------
Right         **       ( 2 ** 3 ** 2  # is 2 ** (3 ** 2) = 512)
Right         + - (unary)
Left          * / % x (multiplication)
Left          + - . (addition)
```

Example of conversion between numbers and strings:

```
      if we turn -w off, what is
      2 . "E" . 1 + 20 - "10+5"
      # first of all, everything is at the same precedence level,
      #   so we just go left to right
      # 2 converted to "2"
      # "2" . "E" is "2E"
      # "2E" . 1
      # 1 is converted to "1"
      # "2E" . "1" is "2E1"
      # "2E1" + 20
      # "2E1" is converted to 20
      # 20 + 20 is 40
      # 40 - "10+5"
      # "10+5" is converted to 10
      # 40 - 10 is 30
```

```
#!/usr/local/bin/perl -w
# PERL STRING INCREMENT
# By Darren Lim 8/30/97

# Description: This program is a demonstration of Perl string increments.
# Procedure: Various strings are incremented via the AutoIncrement operator.
#            Each one is then printed to the screen with various results.
# Input:       None
# Output:      The various contents of incremented strings.


  $str1 = "Hello World";
  print ++$str1, "\n";          #Can not handle spaces; prints 1<newline>
  $str2 = "HelloWorld";
  print ++$str2, "\n";          #prints HelloWorle<newline>
  $str3 = "zz9";
  print ++$str3, "\n";          #prints aaa0
  $str4 = "aZ";
  print ++$str4, "\n";          #prints bA
```

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

# PERL VARIABLE INTERPOLATION EXAMPLE
# By Louis Ziantz

# Description: This program is a demonstration of scalar variable
#             interpolation in Perl.
# Procedure:   A number of variables are used inside double-quoted
#             strings, and the results are printed.
# Input:       None
# Output:      Various messages using interpolated variables.

# (see next page)
```

```
$num = 5;
$friend = "John Smith\n";
$month = "Octo";
print "$num\n"; # prints 5<newline>
print $friend;  # prints John Smith<newline>
print "Can I have \$$num?\n";
# prints  Can I have $5?<newline>
print "The month is $monthber.\n";
# prints The month is .<newline>
# because it thinks $monthber is a scalar variable
print "The month is ${month}ber.\n";
# prints The month is October.<newline>
print 'The month is ${month}ber.\n';
# prints The month is ${month}ber.\n
$test = "double interpolation?";
$insert = '$test';  # $insert is '$test', not "double interpolation?"
print $insert;
# prints $test<newline>
```

Output (of interpolation example):

```
% var.interp.plx
Identifier "main::test" used only once: possible typo at ./var.interp.plx line 19.
Identifier "main::monthber" used only once: possible typo at ./var.interp.plx line 10.
5
John Smith
Can I have $5?
Use of uninitialized value at ./var.interp.plx line 10.
The month is .
The month is October.
The month is ${month}ber.\n
$test
```

```
# PERL EXAMPLE OF CASE SHIFT ESCAPES
# By Louis Ziantz

# Description: This program is a demonstration the use of
#              case shift backslash escape character sequences.
# Procedure:   Case shift escapes are applied to a number of
#              strings, and the results are printed.
# Input:       None
# Output:      Various strings that have their case modified.

$fred = "fred";
$bigfred = "\U$fred";
print "$bigfred\n";
# prints FRED<newline>
$bigbarney = "BARNEY";
$barney = "\L$bigbarney";
$capbarney = "\u\L$bigbarney"; # or "\u$barney"
print "$barney $capbarney $bigbarney\n";
# prints barney Barney BARNEY<newline>
```

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

# PERL READING SCALARS FROM STDIN EXAMPLE
# By Louis Ziantz

# Description: This program is a demonstration reading scalar
#              variables from STDIN.
# Procedure:   Two values are read from STDIN and they are echoed.
# Input:       Name (string) and age (number).
# Output:      Input is echo printed.

print "Enter first name: ";
$name = <STDIN>;
print "Enter age: ";
$age = <STDIN>;
print "$name is your name and you are $age years old.\n";
```

Output (of reading scalars from STDIN example):

```
% stdin.plx
Enter first name: Bart
Enter age: 10
Bart
 is your name and you are 10
 years old.
```

Builtin functions and parentheses:

```
$a = 1;
# suppose we want to add 2 to the value of $a and replicate
# the result 5 times to print it
print 2 + $a x 5 , "\n";
# prints 11113<newline> due to precedence
print (2+1) x 5 , "\n";
# prints 3 (no newline) because print prints 3, returns
# 1 (sucessful) which is converted to a string and
# replicated 5 times, but the result is not stored or
# printed anywhere
$t = print (2+1) x 5 , "\n";
print("$t");
# prints 311111 (no newline); the 3 is from the first print;
# the 11111 is from $t
print ((2+1) x 5 , "\n");
# this prints 33333<newline> as expected

# suppose we want to read in a character and replicate it 5
# times
print("Enter a character: ");
$d = <STDIN>;
$b = chomp $d x 5; # taken as chomp ($d x 5)
# the above is a comiler error because it expects
# a memory location that it can modify
$c = chomp($d) x 5;
```

Examples of array assignment:

```
# array on lhs
@empty = ();
@nums = (1, 2, 3);
@numbers = (1 .. 5);
@nums = @numbers;     # @nums is resized to hold contents of @numbers
@two = 2;             # 2 is promoted to (2) and assigned
@stooges1 = qw(Larry Moe);
@stooges2 = (@stooges1, "Curly");
@stooges3 = ("Shemp", @stooges2, "Curly Joe");
# note: this is still a one-dimensional array

#list literal on lhs
($a, $b, $c) = ("a", "b", "c");  # $a gets "a", $b gets "b", $c gets "c"
($var1, $var2) = ($var2, $var1);  # swap values of $var1 and $var2
($d, @hold) = ($a, $b, $c);  # $d gets "a"; @hold gets ("b", "c")
($e, @hold) = @hold;  # $e gets "b"; @hold gets ("c")

# if the number of elements being assigned > number of variables on
# lhs, excess elements are discarded
($x, $y, $z) = (1 .. 5);  # $x gets 1, $y gets 2, and $z gets 3
# if the number of elements being assigned < number of variables on
# lhs, execess lvalues are assigned undef
($s, $t, $u) = ("one", "two");  # $s gets "one", $t gets "two", and $u gets undef

# array variable in a literal list should be placed at the end of the list
($a, $b, $c) = ("a", "b", "c");
($d, @hold, $x) = ($a, $b, $c, 1, 3, 4);
# $d is "a"; @hold is ("b", "c", 1, 3, 4); $x is undef

# important to remember size of an array on the rhs
($a1, $a2, $a3, $a4, $a5, $a6) = (@hold, $d);
# works because @hold has 5 elements, so $d is assigned to $a6
# if @hold has 6 elements, variable from $d is never assigned
```

Examples of using a slice:

```
# the subscript for a slice is also a list

@array = (1, 2, 3);
@slice = @array[0,1];       # @slice gets (1, 2)
@array[0,1] = @array[1,0];  # swap first two array elements
@array[0,1] = @slice;       # first two elements set back to
                            #   starting values

# can take a slice of a literal list

@kids = (qw(george elroy jane judy))[1,3];  # @kids is ("elroy", "judy")
$a = 2;
($daughter) = (qw(george elroy jane judy))[$a+1];  # $daugher is now "judy"
@jetsons = qw(george elroy jane judy);
@reorder = (0, 2, 3, 1);
@ordered = @jetsons[@reorder];  # @ordered is qw(george jane judy elroy)
```

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

# METHODS OF ACCESSING LAST ARRAY ELEMENT IN PERL
# By Louis Ziantz

# Description: This program is a demonstration different ways
#              to access the last element of an array.
# Procedure:   The last element of an array is accessed using
#              three different methods.
# Input:       None
# Output:      The size and last element of the an array.

@strings = ("first", "second", "last");
$size = @strings;       # returns size of array

print("There are $size elements in the \@strings array.\n");
print("The last element of the array is $strings[$size - 1].\n");
print("The last element of the array is $strings[$#strings].\n");
print("The last element of the array is $strings[-1].\n");
```

Output (of accessing last array element example):

```
% ar.elem.plx
There are 3 elements in the @strings array.
The last element of the array is last.
The last element of the array is last.
The last element of the array is last.
```

Push vs. Unshift:

```
@list = (1, 2, 3);
push(@list, 4, 5);
# produces the same list as
push(@list, 4);
push(@list, 5);
# either way yields (1, 2, 3, 4, 5)
#
@list = (1, 2, 3);
unshift(@list, 4, 5);
# produces (4, 5, 1, 2, 3)
# while
unshift(@list, 4);
unshift(@list, 5);
# produces (5, 4, 1, 2, 3)
```