Spring 98 - Programming in Perl

Perl TEST - April 8, 1998

Instructor: Louis Ziantz

```
Question 1  [25 pts] : _______________
Question 2  [20 pts] : _______________
Question 3  [25 pts] : _______________
Question 4  [30 pts] : _______________
Total Score [100pts] : _______________
```

Read each question carefully. If you need more space, use the back of a sheet.

Your name:______________________________________

Your RCS userid:_________________________

Have you ever used HTML to create a webpage? ____________

Question 1 *[25 pts]*
Write a Perl script that computes the weekly pay for an employee. First prompt for an employee name. You may assume any name entered is valid. The script should then prompt for a numeric paycode, read in the paycode, prompt for additional data (given below) based on the paycode, read in that data, and compute and print pay (as indicated below).

The paycodes are as follows:

1 – hourly worker

- *prompt for and read:* hours worked
- *pay computed as:* \$10.00 * hours worked if hours worked <= 40; \$400 + (\$15.00 * hours beyond 40 worked) otherwise

2 – commission worker

- *prompt for and read:* gross weekly sales
- *pay computed as:* \$250 + 5.7% of gross weekly sales

other – invalid pay code

- print a message indicating that the pay code entered was invalid

For example, if `Bob Smith` is entered as the employee name, and `1` is entered as the paycode (hourly worker), then the script should prompt for hours worked (suppose `50` is entered). The script would then print `The weekly pay for Bob Smith is $550.` .

```
print("Enter an employee name: ");
chomp($name = <STDIN>);

print("Enter the employee's numeric pay code: ");
chomp($paycode = <STDIN>);

if ( $paycode == 1 ) { # employee is an hourly worker

  print("Enter hours worked: ");
  chomp($hours_worked = <STDIN>);

  if ( $hours_worked <= 40 ) { # no overtime
    $pay = 10.00 * $hours_worked ;
  } else { # compute pay with overtime
    $pay = 400.0 + (15.00 * ($hours_worked - 40.0));
  }
  print("The weekly pay for $name is \$$pay.\n");

} elsif ( $paycode == 2 ) { # employee is a commission worker

    print("Enter gross weekly sales: ");
    chomp($gross_sales = <STDIN>);

    $pay = 250.00 + 0.057 * $gross_sales;
    print("The weekly pay for $name is \$$pay.\n");
  } else { # invalid paycode

    print ("Invalid paycode entered.\n");

  }
```

Question 2 *[20 pts]*
For each part of this question, indicate what the snippet of Perl code prints below the code segment (assume that the `-w` flag is not used so no warnings will be printed):

(a)
```
$a = <STDIN>; # assume hello is entered from the keyboard
$a[0] = "there\n";
$a[1] = 'world\n';
chomp(@a);
print("$a@a");
```

```
hello
there world\n

# explanation:
# The read from STDIN includes a newline, and $a is never chomped, so
# hello appears on a line by itself.
# The array @a is chomped; however, location 1 was assigned using a
# single quoted string ('\n' is literally \n), so the \n was not
# removed from 'world\n'. However, the "\n" was removed from
# "there\n". Thus, the contents of @a appears on a single line.  The
# space between each element of the array is a result of variable
# interpolating it.
```

(b)
```
$b = 2 + "Three" . 4;
print("$b\n");
$c = 2;
$d = $c++;
print("$c $d\n");
$e = $c . $b x $d;
print("$e\n");


24
3 2
32424

# explanation:
# Each operator in in the first statement has the same precedence, so
# we proceed from left to right.  Because + forces numeric context,
# "Three" must be converted to a number; however, it doesn't start
# with any digits or other numeric characters, so it is converted to
# 0.  Now 2 + 0 is 2.  The . forces string context, so 2 becomes '2'
# and 4 becomes '4'.  Now, '2' . '4' concatenates the two strings into
# '24' which is assigned to $b and printed.  Next, the variable $c gets 2;
# $c++ first returns the existing value (2) to $d and then increments
# $c by 1.  Thus, $c becomes 3 and $d is 2.  Replacing variables with
# their values, the last statment then becomes $e = 3 . '24' x 2 The
# repetition operator (x) has a higher precedence than the
# concatenation operator (.).  As a result, '24' x 2 is done first,
# producing '2424'.  The value 3 is then converted to '3' and '3'
# . '2424' gives us '32424' which is what is printed by the last
# statement
```

(c)
```
@nums = (1 .. 6);
@odds = (0, 2, 4);
@indices = @odds;
foreach $index (@indices) {
  $index = $index + 1;
}
print("@indices\n");
$indices[3] = $#odds;
print("@indices\n");
print("@nums[@indices]\n");
```

```
1 3 5
1 3 5 2
2 4 6 3

# explanation:

# The array @nums is assigned (1,2,3,4,5,6).  The array @indices is
# set equal to @odds, which is (0, 2, 4).  Each element of @indices is
# then incremented by 1 (the $index is an alias to the element
# locations), giving (1, 3, 5) which is printed.  Location 3 of
# @indices then gets the INDEX of the last element of @odds.  Since
# @odds has three elements and subscripting starts at 0, the index of
# its last element is 2.  Now @indices is (1, 3, 5, 2) which is
# printed.  The last print uses an array slice to select the elements
# of the array @nums in locations 1, 3, 5, and 2 (the contents of
# @indices), resulting in 2 4 6 3 being printed.
```

Question 3*[25 pts]*
Write a Perl script that uses the diamond operator to read the contents of files specified on the command line. Assume each file contains one word to a line. Once a line is read, it should be lowercased and added to an array. After all files have been exhausted, the array should be sorted in descending order based on the ASCII collating sequence. One line should then be printed for each entry in the array. Each line should consist of the current line number being printed (starting at 1), a tab, and the array entry. (*Hint:* note that the standard method of sorting in Perl sorts in ascending order. I would suggest sorting the array normally and then using another Perl builtin function to change the ordering.)

```
$lines = 0;

# diamond operator automatically processes files specified on command line
while(<>) {

  $array[$lines++] = "\L$_"; # lowercase each string read

}

# both functions only return lists; @array is unchanged unless we
# overwrite its contents as done here
@array = sort(@array);  # sort in ascending order
@array = reverse(@array); # reverse order to descending

# print each array location
for($i = 1; $i <= $lines; $i++) {

    print("$i\t$array[$i-1]");

}
```

Question 4 *[30 pts]*
Given the following arrays:

```
@classes = ('First', 'Second', 'Third', 'Fourth');
@dates = ('Mar. 18', 'Mar. 25', 'Apr. 1', 'Apr. 8');
```

(a) Assign the same values as in the first assignment above to `@classes` using the quote word function.

```
@classes = qw(First Second Third Fourth);
```

(b) Using the contents of `@classes` as keys and the contents of `@dates` as values, create a hash called `%class_to_date` with a single line of Perl code.

```
# hash slice used; @ must appear before class_to_date, not $
@class_to_date{@classes} = @dates;
```

(c) Use a list literal composed of `=>` operators and literal strings to initialize a hash called `%topics` with keys and values given the following table:

Key	Value
First	scalars
Second	arrays
Third	hashes
Fourth	regexes

```
%topics = (
       First => 'scalars' ,
       Second => 'arrays' ,
       Third => 'hashes' ,
       Fourth => 'regexes' ,
  );
```

(d) Remove the element of the %topics hash that corresponds to the fourth Perl class.

```
delete $topics{Fourth};
```

(e) Print the number of elements in the %topics hash using no more than two Perl statements.

```
# keys in scalar context returns number of pairs in hash
# this is not the only acceptable answer
$size = keys(%topics);
print($size, "\n");
```

(f) Iterate through the hash %topics using the each function. For each element of %topics, use the key returned by each to index %class_to_date and obtain a date. Using this date and the value returned from each (the topic), print one line of the following form for each element of %topics:

```
On <date>, we learned about <topic>.
```

Thus, this piece of code should print

```
On Mar. 18, we learned about scalars.
On Apr. 1, we learned about hashes.
On Mar. 25, we learned about arrays.
```

```
while(($class, $topic) = each(%topics)) {

    $date = $class_to_date{$class};

    print("On $date, we learned about $topic.\n");

}
```