# Programming in Perl
# CSCI 2962
# MidTerm Exam

Read and follow ALL directions:

1) Turn off your cellular phones!

2) Clear your desks/tables of all notes, books, computers, calculators, handheld PDAs, etc.

3) Clearly print your name and RCS ID on the top of EACH page except the last.

4) If at all possible, move at least one seat away from any other student.

5) No talking or collaboration during this exam.

6) If you have a question, approach Paul and/or Justin in the front of the room.

7) You have until 6:00pm to complete the exam.

8) When completed, bring all pages of the exam to the table in front. Detach the final page and put it on the left desk (your left). Place the remaining pages on the right desk. (yes, your right).

9) Good luck.

10) Tomorrow morning, stop by the Union Admin office to pick up your free passes to the UPAC Cinema sneak preview of *Life as a House*.

11) There is no rule 12.

True or False. For each statement, circle either TRUE or FALSE. 1 point each

1) To make your program executable, the shebang must be at the top of your Perl code, but may come after any comments. TRUE [✓ FALSE]

2) $a = 5, $a = 7.5, $a = '1', $a = "Hi!!", and $a = '' (there is nothing between the two quotes) are all valid assignments in Perl. [✓ TRUE] FALSE

3) The print statement must take at least one argument. TRUE [✓ FALSE]

4) The statements `$foo = 4;` and `@foo = 4;` do the same thing. TRUE [✓ FALSE]

5) The first member of the array @ARGV is always the name of the Perl script. TRUE [✓ FALSE]

6) the keywords `for` and `foreach` mean the same thing, and are interchangeable. [✓ TRUE] FALSE

7) Unlike some other languages, Perl has no concept of a 'goto' statement. TRUE [✓ FALSE]

8) If you choose the front slash ( / ) as your delimiter, you do not need to type the 's' in the search-and-replace operator `s///`. TRUE [✓ FALSE]

9) Even if "@cs" has not been used as an array already, you cannot safely print out a string such as "My address is lallip@cs.rpi.edu." [✓ TRUE] FALSE

10) `push` and `unshift` mean the same thing, and are interchangeable, as are `pop` and `shift`. TRUE [✓ FALSE]

Short Answer  (2 points each)

11) Briefly explain the difference between `chop` and `chomp`:

Chop removes the last character from a string.  Chomp removes the last character if and only if it's a newline.

12)  What is meant by the phrase "variable interpolation"?

Replacing the name of a variable with the value of that variable within a double-quoted string.

13)  Name the three operators/functions we've seen so far that use regular expressions.

m//  s///   split()

14) Explain the difference between these two code snippets (ie, what does each do, and why are they different?)

```
    $a = $b || $c || $d;       $a = $b or $c or $d;
```

The first one assigns $a to the first true value of $b, $c, $d.  The second assigns $a to $b.  If $b is false, $c is evaluated.  if $b and $c are both false, $d is evaluated (however, $a *always* gets the value of $b).  They are different because or and || have different precedence.

15) Give the string-operator equivalents of each of the following numerical operators:  ==  >  <  >= <= !=  <=>

eq gt lt ge le ne cmp

16) In each of the following cases, describe what will happen if "file.txt" …

A) does exist (and contains data) and B) does not exist

```
open FILE, "file.txt" or die;
```

A)file.txt is opened for reading.  B) program exits.

```
open FILE, ">file.txt" or die;
```

A)contents of file.txt are erased, file opened for writing.  B)file.txt created and opened for writing

```
open FILE, ">>file.txt" or die;
```

A)file.txt opened for writing to end of file.  B)file.txt created and opened for writing

17) What is the difference in output between these two statements?

```
print “Hello World!  Where do you want to go today?\n”;
print ‘Hello World!  Where do you want to go today?\n’;
```

First one prints string followed by newline. Second prints string followed by a slash and letter n

18) Describe the two strings $str1 and $str2 after executing the following code:

```
$str1 = “/home/36/lallip/public_html/”;
$str2 = $str1;
$str1 =~ s#/#\\#;
$str2 =~ s*/*\\*g;
```

$str1 = “\home/36/lallip/public_html/”
$str2 = “\home\36\lallip\public_html\”

19) if `%my_hash` contains

(‘Jan’ => ‘January’, ‘Feb’ => ‘February’, ‘Mar’ => ‘March’, ‘Apr’ => ‘April’, ‘May’ => ‘May’, ‘Jun’ => ‘June’, ‘Jul’ => ‘July’, ‘Aug’ => ‘August’, ‘Sept’ => ‘September’, ‘Oct’ = ‘October’, ‘Nov’ => ‘November’, ‘Dec’ => ‘December’)

what are the contents of the array `keys %my_hash`?

(‘Jan’, ‘Feb’, ‘Mar’, ‘Apr’, ‘May’, ‘Jun’, ‘Jul’, ‘Aug’, ‘Sept’, ‘Oct’, ‘Nov’, ‘Dec’)
(not necessarily in that order)

20) What are the three keywords that can be used within a while, until, for, or foreach loop to control the flow of the code? (Do not bother describe what they do – just list them).

next, last, redo

What’s Wrong Here? Each of the following chunks of code do not work (ie, either they do not compile, or do not have the intended effect). State why the code does not work as is, and explain how to correct it. (5 points each)

21)

```
#!/usr/local/bin/perl –w
#Store all the lines in all three files into one array
@files = (“file1.txt”, “file2.txt”, “file3.txt”);
for ($i = 0; $i<$#files; $i++){
    open (FILE, $files[$i]) ||
        die “Cannot open $files[$i]\n”;
    @lines = <FILE>;
    close FILE;
    push @all_lines, @lines;
}
```

$#files is last index of @files. Change to $i<=$#files, or $i<@files, or

$i<$#files+1

22)

```
#!/usr/local/bin/perl –w
#This program gets a sentence from the user and prints
#each word in the sentence on a separate line.
print “Enter a sentence:\n”;
$string = <STDIN>;
@words = split ($string, / /); #(space between the /’s)
foreach (@words){
    print “$_\n”;
}
```

arguments to split are backwards. change to `split(/ /, $string)`

23)

```
#!/usr/local/bin/perl –w

#....
#(assume some piece of code has filled @array with many
#(but an unknown number) of integers.  We want to
#compute the average of all positive numbers, and stop
#when a number equals –999. Assume there is at least
#one positive integer before –999.
for $num (@array){
    if ($num < 0){
        if ($num == -999){
            last;
        }
        redo;
    } else {
        $sum+=$num;
        $count++;
    }
}
$avg = $sum/$count;
#pretend the print statement is all on one line…
print “The avg of + numbers before –999 is ”.$avg.“\n”;
```

`redo` will not get next value of $num.  Change to `next;`

What’s the output? For each code chunk below, give the output of the code.

24) 6 points

```
@foo = (1..10);
foreach $bar (@foo){
    if ($bar % 2 == 0){
        $bar /= 2;
    } else {
        $bar *= 2;
    }
}
print “@foo\n”;
```

2 1 6 2 10 3 14 4 18 5

25) 7 points

```
$i = 0;
{
    $i += 5;
    $i *= 2;
    $i += 3;
    if ($i > 50){
        print (“\$i = $i\n”);
        last;
    }
    $i--;
    redo;
}
```

$i = 85

26) 7 points

```
$string = “These:are:some:fields\n”;
@fields = split (/:/, $string);
$new_string = join (“#”, “”, @fields);
print $new_string;
#hint: think of line 3 as    join(“#”, (“”, @fields));
```

#These#are#some#fields

Code it! Write a small program to do each of the following tasks (do not concern yourself with error checking):

27) Prompt the user for a number and then prompt for a string. Print the string that number of times on separate lines. Do not use any looping structures. 10 points

```
print “Enter a string\n”;
chomp ($s = <STDIN>);
print “Enter a number\n”;
chomp ($n = <STDIN>);
print “$s\n” x $n;
```

28) Read an unknown number of strings from the command line, and print each string out in the opposite order. Change all upper-case letters to lower-case, and all lower-case letters to upper-case. Do not use the `reverse` keyword. 10 points

```
for ($i = $#ARGV; $i>=0; $i--){
        $ARGV[$i] =~ tr/a-zA-Z/A-Za-z/;
        print “$ARGV[$i] ”;
}
print “\n”;
```

29) Prompt the user for one word at a time. User will enter the word ‘done’ (case insensitive) when all words are finished being entered. Print each word entered and the number of times it was entered. 15 points

```
while ($word !~ /\bdone\b/i){
        print “Enter a word\n”;
        chomp($word = <STDIN>);
        $hash{$word}++;
}
foreach $word (keys %hash){
        print “$word : $hash{$word}\n”;
}
```

**Do NOT write your name on this page.**

MidSemester course evaluation

When you have completed the exam, please fill out this evaluation. When you've completed the evaluation, detach this page and bring it to the front of the room along with the exam. Put this evaluation on the left desk, and the exam on the right desk.

1) What is your major? ______________________________

2) What is your class year (circle one)? Frosh Soph Junior Senior Grad other________

3) How much Perl did you know before registering for this class?

○ None at all

○ Less than we've covered so far

○ Just about what we've covered up to now

○ More than we've covered so far

○ I'm an expert

4) Please rate the following aspects of the class from 1 (not at all helpful) to 5 (very helpful), as it applies to your learning of the Perl language

| | | | | | | |
|---|---|---|---|---|---|---|
| Class Lectures | N/A | 1 | 2 | 3 | 4 | 5 |
| Class Notes posted online | N/A | 1 | 2 | 3 | 4 | 5 |
| Homework | N/A | 1 | 2 | 3 | 4 | 5 |
| Paul's office hours | N/A | 1 | 2 | 3 | 4 | 5 |
| Justin's office hours | N/A | 1 | 2 | 3 | 4 | 5 |
| email communication with Paul and/or Justin | N/A | 1 | 2 | 3 | 4 | 5 |
| Course website | N/A | 1 | 2 | 3 | 4 | 5 |
| The midterm exam you just took | N/A | 1 | 2 | 3 | 4 | 5 |

5) Approximately how many of the first 6 lectures did you attend? _______

6) What do you expect your grade in the course to be? A B C D F

7) What one thing would you suggest Paul continue doing as he has been?

8) What one thing would you suggest Paul change about the course?