Question 1 (Regexes) *[20 pts]*

(a) Suppose we have the following assignment:

```
$_="chim chimery chim chimery chim chim chery";
```

Carefully circle the portion of the string matched by the following expressions (in other words, indicate the result of printing `$&`) :

i. `/chim.*chim/;`

```
chim chimery chim chimery chim chim chery
```

chim chimery chim chimery chim chim

ii. `/chim.*?chim/;`

```
chim chimery chim chimery chim chim chery
```

chim chim

(b) As part of a Perl script, a bank wishes to validate an account holder. The user must enter their account number, a blank space, and their mother's maiden name. The account number must start the line and the name must be the last thing on the line. The account number consists of a single uppercase letter, followed by a dash, followed by three digits, followed by a dash, followed by four digits. The name consists of an uppercase letter followed by one or more uppercase or lowercase letters. Assuming the data is stored in `$_` and that `$_` has been chomped, write the regular expression to correctly validate an account holder.

```
$acct_num = "[A-Z]-[0-9]{3}-[0-9]{4}";
$last_name = "[A-Z][a-xA-Z]+";
/^$acct_num $last_name$/;
```

(c) Suppose we wish to write a Perl script that reads from a file and detects lines that have double words, that is, two words that are exactly the same separated only by blank spaces. For example, a line of the file might read:

```
In my opinion, the the solution is to use Perl.
```

We wish to catch this as a possible typo by reusing a previously matched pattern in the regex. Note that any valid word boundary is allowed before the first `the` and after the second `the`, but we expect only one or more blank spaces in between them. Assuming the current line being processed is stored in `$_`, write the regular expression that will return true when double words are encountered.

```
/\b(\w+)\s+(\1)\b/;
```

Question 2 (Regexes) *[20 pts]*

(a) Write a regular expression that matches the following pattern in a string (assume the string is stored in `$_`):

- a slash followed by
- an asterisk followed by
- one or more characters that aren't newlines followed by
- an asterisk followed by
- a slash

```
/\/\*.+\*\//;
```

(b) Consider the following substitution:

```
s/([A-Z][a-z]+) ([A-Z][a-z]+)/$2 $1/;
```

i. Suppose `$_ eq "Betty Boop"` before the above substitution is applied. What is stored in `$_` afterwards?

```
Boop Betty
```

ii. Suppose `$_ eq "Leonardo DaVinci"` before the above substitution is applied. What is stored in `$_` afterwards?

```
Da LeonardoVinci
```

(c) Suppose we are writing a script to convert from Celsius to Fahrenheit or vice versa depending on the string entered by the user. For our purposes, a valid entry contains an integer composed of an optional plus or minus followed by one or more digits. Such an integer should be matched only if followed by one or more blank spaces in turn followed by a C or an F (uppercase only), which is followed by one or more whitespace characters. Since we want only the number to be stored in `$&`, the pattern after the integer should be enclosed in a lookahead anchor.

For example, given `"-20 F\n"`, we wish to match `"-20"`. Given `"-20\n"`, we would match nothing. Given `"-20 Fh\n"`, we would also match nothing.

Assuming the data is stored in `$_` and is unchomped, write the regular expression to do this.

```
$integer = "[+-]?[0-9]+";
/$integer(?= +[CF])/;
```

Question 3 (File Manipulation) *[20 pts]*
A file called backs.dat stores information on running backs in the following format:

```
<first-name> <last-name> <team> <carries> <yards><newline>
```

For example,

```
Jerome Bettis PIT 323 1460
```

Write a Perl script that first opens this file for reading, and a file called avgs.dat for writing (any existing data in the file should be overwritten). If either operation fails, the script should exit with an appropriate error message. Each line read from backs.dat should be separated so that all five fields are stored in separate variables. For each input line, a line of the following form should be output to avgs.dat:

```
<first-name> <last-name> <avg-yds-per-carry><newline>
```

Note that `<avg-yds-per-carry>` is computed as `$yards/$carries`. Continuing our above example, we would output

```
Jerome Bettis 4.52012383900929
```

After backs.dat has been processed in its entirety, both files should be closed. If a close fails, an error message should be produced and the script should exit.

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

$infilename = "backs.dat";
$outfilename = "avgs.dat";

open(IN, "<$infilename") or die ("Could not open $infilename for reading",
                                 " : $!\n");
open(OUT, ">$outfilename") or die ("Could not open $outfilename for writing",
                                   " : $!\n");

while( defined($line = <IN>) ) {

    chomp($line);

    ($first, $last, $team, $carries, $yards) = split(/ /, $line);

    $yds_per_carry = $yards/$carries;

    print OUT ("$first $last $yds_per_carry\n");

}

close(IN) or die ("Could not close $infilename : $!\n");
close(OUT) or die ("Could not close $outfilename : $!\n");
```

Question 4 (Subroutines) *[20 pts]*

(a) Write a subroutine called *plus_plus*. The routine may be passed an array or a list of scalar variables. It returns nothing. The routine should add 1 to every value of the array passed in. For example,

```
@numbers = (3, 2.1, 5);
plus_plus(@numbers); # @numbers is now (4, 3.1, 6)
```

You may assume that the list passed in consists of only numbers. The routine does not have to work for lists of literals.

```
sub plus_plus {

    my($elem);

    foreach $elem (@_) {
       $elem++;
    }
}
```

(b) Write a subroutine called *absolute*. The routine may be passed an array or a list of numbers, and it returns a list consisting of the absolute values of the numbers passed in. For example,

```
@numbers = (-3, -4.5, 6, 2.2);
@abs_vals = absolute(@numbers); # @abs_vals is (3, 4.5, 6, 2.2)
@abs_vals = absolute(4, -5.6, -11); # @abs_vals is (4, 5.6, 11)
```

Recall that the absolute value of a positive number x is simply x and the absolute value of a negative number x is $-x$. Do not make use of the Perl built-in function `abs`. You will need to create a private array to store the list of absolute values. The list passed in should NOT be modified (you may assume it contains only numbers).

```
sub absolute {

    my(@start_list) = @_;
    my(@return_list);
    my($elem);
    my($i) = 0;

    foreach $elem (@start_list) {
       if ( $elem >= 0 ) {
           $return_list[$i] = $elem;
       } else {
           $return_list[$i] = - $elem;
       }
       $i++;
   }

   return @return_list;
}
```

Question 5 (<u>Hard References</u>) *[20 pts]*

(a) Write a subroutine *search_and_replace* that accepts the following parameters:

   i. a (hard) reference to a hash
   ii. a scalar search value
   iii. a replacement scalar value

For each key in the hash referenced by the first argument, the corresponding value should be compared with the scalar search value. If the value equals (`eq`) the scalar search value passed in to the function, it should be replaced with the replacement scalar value, that is, the value corresponding to the current key being processed should be overwritten with the replacement value. Every such value matched should be replaced. None of the keys should be modified. The function should return nothing. For example,

```
# the curly brace defines a reference to an anonymous hash
$hash_ref = {
  "Van Halen" => "David Lee Roth",
  "Twisted Sister" => "Dee Schnider",
  "U2" => "Bono",
  "the News" => "Huey Lewis"
};
search_and_replace($hash_ref, "David Lee Roth", "Sammy Hagar");
# this subroutine call replaces every occurrence of the value
# "David Lee Roth" with "Sammy Hagar" in the hash referenced by $hash_ref
```

```
sub search_and_replace {

    my($hash_ref) = $_[0];
    my($search_val) = $_[1];
    my($replacement_val) = $_[2];

    my($key);

    # actually, there are more efficient solutions to this problem,
    # such as using each and exiting the loop if a match was found,
    # but something like this was acceptable
    foreach $key (keys(%$hash_ref)) {
        if( $$hash_ref{$key} eq $search_val ) {
            $$hash_ref{$key} = $replacement_val;
        }
    }
}
```

(b) Consider the following definition:

```
%lead_singers = (
  "Van Halen" => "David Lee Roth",
  "Twisted Sister" => "Dee Schnider",
  "U2" => "Bono",
  "the News" => "Huey Lewis"
);
```

Given the subroutine that you defined above, call *search_and_replace* so that the value `"Bono"` is replaced with `"the Edge"` in the hash `%lead_singers`.

```
search_and_replace(\%lead_singers, "Bono", "the Edge");
```

Question 6 (<u>CGI</u>) *[20 pts]*
Consider the following Perl Code:

```
use strict;
use CGI qw(:standard);
# emit initial HTML tags
print header(), start_html("Language Preference");
# print a <H1> header in HTML
print h1("Language Preference");
if(param()) {
  # process form
} else {
  # generate form
}
# emit closing HTML tags
print end_html();
```

Fill in the blanks in the above code to write a CGI script that both generates and processes a simple form.

(a) (**generate form**) First, print a horizontal rule (`hr()`) and start the form. An HTML paragraph `p()` should be printed. Within this paragraph should be the string `"Enter name: "` and a `textfield()`. The name associated with the `textfield()` should be `"user_name"`. A `scrolling_list()` should be emitted within another HTML paragraph. The `scrolling_list()` should be associated with the name `"language"` and acceptable values returned by the widget should be the strings `"C"`, `"C++"`, `"Java"`, `"Lisp"`, and `"Perl"`. All other parameters to `scrolling_list()` may be omitted and default values used. Print one more HTML paragraph. Within it include an option to `submit()` (associated with the name `"engage"`) and an option to `reset()` (associated with the name `"clear"`). Finally, end the form and print another horizontal rule.

```
print hr(), start_form();

print p("Enter name: ", textfield( -NAME => "user_name"));
print p(scrolling_list( -NAME => "language",
-VALUES => [ qw(C C++ Java Lisp Perl) ] ) );
print p(submit( -NAME => "engage" ), reset( -NAME => "clear" ) );

print end_form(), hr();
```

(b) (**process form**) The `"user_name"` and `"language"` entered by the user should be queried using the appropriate `CGI.pm` function and stored in `$name` and `$language` respectively. Print a horizontal rule, and within an HTML paragraph include a string which indicates that

`"The favorite language of $name is $language."`

To finish, print another horizontal rule.

```
my($name) = param("user_name");
my($language) = param("language");

print hr();

print p("The favorite language of $name is $language.");

print hr();
```