Lab 3: Conditional Statements

Due date: Monday, January 23, during the lab period.

Assignment Preparation

Lab type. This lab is a **pair programming assignment**. The pairs will be designated by the instructor at the beginning of the lab. Pair programming means that all work for the lab will be performed by two people *working together on one computer*.

Collaboration. Students work in pairs, and it is considered cheating, if members of the team (pair) do not work together. Communication between pairs during lab time is allowed, but no direct sharing of code is allowed.

Purpose. The lab allows you to practice the use of conditional statements. It also facilitates learning the use of boolean expressions as conditions in `if` and `switch` statements.

Programming Style. All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/∼cstaley/General/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a 10% penalty. Significant stylistic violations, epsecially those that make grading harder, may yield stricter penalties. Also note the the Lab 2 requirement for the content of the header comment in each file you submit applies **to each assignment** (lab, programming assignment, homework) in this course.

Testing and Submissions. Any submission that does not compile using the

```
gcc -ansi -Wall -Werror -lm
```

compiler settings will receive an automatic score of 0.

The Task

Note: Please consult the instructor if any of the tasks are unclear.

For this lab, you will write and submit a number of simple programs that test the use of the conditional statements in a number of situations.

Program 1: Revised Temperature Conversion

You shall revise your original temperature conversion program from **Lab 2** according to the following specifications.

TR1. Files. Your temperature conversion program shall now consist of two files. All the temperature conversion code you write shall be put in `converter.h` file. In addition, you shall also develop and submit the testing program `testConverter.c`, which will contain the `main()` function which includes your test cases.

TR2. Functions. You shall reimplement the same two functions as in **Lab 2**:

```
double toFahrenheit(double temperature);
double toCelsius(double temperature);
```

This time, however, the functions shall perform checks of the input parameter to establish that the input is valid.

TR3. toFahrenheit(). The `double toFahrenheit(double temperature)` function will take as input a temperature reading in degrees Celsius and convert it to degrees Fahrenheit (as before). Before doing so, however, it must ensure that the temperature reading is valid.

A temperature reading is valid if its value is greater than the value of the absolute zero. For the Celsius scale, absolute zero is -273.15.

For any input value that is less than the absolute zero value `toFahrenheit()` shall return a "dummy" value of -10000.

TR4. toFahrenheit(). The `double toCelsius(double temperature)` function will take as input a temperature reading in degrees Fahernheit and convert it to degrees Celsius (as before). Before doing so, however, it must ensure that the temperature reading is valid.

A temperature reading is valid if its value is greater than the value of the absolute zero. For the Fahernheit scale, absolute zero is -459.67.

For any input value that is less than the absolute zero value `toCelsius()` shall return a "dummy" value of -10000.

TR5. Testing. The `int main()` function of your `testConverter.c` file shall contain `checkit_double()` macros testing each of the two functions from `converter.h`. You shall include at least 10 tests per function. You can use some/all tests from **Lab 2**, but make sure that your tests cover all possible execution paths for each function.

Program 2: In or out?

You will write a function `checkCircle()` int check takes as input five `double` values: a center point of a circle in two dimensions and its radius and the coordinates of another 2D point.

Your function must check whether the point is inside or outside the sphere. Three situations are possible:

- The point is inside the circle. Your function shall return -1.
- The point is outside the circle. Your function shall return 1.
- The point is on the circle boundary. Your function shall return 0.

When writing the program, please do the following:

Recall from high-school math that given a circle of raduis r centered at point (x_0, y_0), the equation defining the inside of a circle is

$$(x - x_0)^2 + (y - y_0)^2 < r^2.$$

Similarly, the equation defining the sphere boundary is

$$(x - x_0)^2 + (y - y_0)^2 = r^2.$$

Recall also, that floating point numbers are hard to compare using C's "==" comparison operator. 49.99999 and 50.00001 may represent the same number, 50.0, obtained as a result of two different operations (e.g., 100.0/2.0 and 200.0/4.0). Therefore, in order to determine whether your point (x, y) is on the sphere boundary, it is not enough to compare $(x - x_0)^2 + (y - x_0)^2$ to r^2.

Instead, you will introduce a floating point constant `EPSILON` equal to 0.00001 in your program, and will compare the absolute difference between $(x - x_0)^2 + (y - y_0)^2$ and r^2 to it. If said difference is less than `EPSILON` then you consider the point to lie **exactly on the boundary**. You can also use similar comparison operations to test whether the point is inside or outside the sphere.

Files to submit. Place your `int checkCircle()` function into a `circle.h` file. Additionally, create a C program that uses `checkit_int()` macro to test this function. Create at least 10 tests and include them in your `int main()`. Make sure all execution paths are covered.

Examples. Consider the following functions calls. All calls assume the following order of inputs: x_0, y_0, r, x, y.

```
checkCircle(0,0,2,1,1);
checkCircle(3,4,1,10,-10);
checkCircle(1,1,1,0,1);
```

The first call shall return -1 (point $(1, 1)$ is inside the circle of radius 2 centered at $(0, 0)$). The second call shall return 1 (point $(10, -10)$ is outside the circle of radius 1 centered at $(3, 4)$). The third call shall return 0 (point $(0, 1)$ lies on the boundary of the circle of radius 1 centered at $(1, 1)$).

Program 3: Limerick or Haiku?

For this assignment you shall create a file `poetry.h` which implements a series of Limerick or Haiku games. Each game shall be implemented by one function in the file.

Limerick or Haiku game. A Limerick or Haiku game is a simple decision procedure that has the following properties:

- A set of input parameters. These parameters (usually numbers) are used in the decision procedure.
- A set of outcomes. A typical game has two outcomes, which we denote as a haiku and limerick, although more complex version of the game may involve more outcomes.
- A set of rules. The rules specify the conditions on the input parameters that must meet in order for the game to end in a specific outcome.

A *function implementing an instance of Limerick or Haiku game* shall take as input a specified set of input arguments/parameters and use their values to determine the outcome of the game according to the specified rules. The output generated by this function will be either a printout or a haiku or a limerick — as the game outcome directs.

LHR1. Functions to implement. For each set of game rules you will implement two functions. One function shall return a `char` value specifying the game outcome. The other function shall be a `void` function which shall print either a limerick or a haiku based on the game outcome.

The actual gameplay will be coded up in the first function (the one returning the `char` value). This way, you can test the gameplay using `checkit_char()` macros prior to completing the full assignment. The second function essentially is providing a convenient *wrapper* around it, in order to print the poems. Functin names will be specified when individual game instances are described. In what follows we refer to the functions that return `char` values as gameplay functions, while the `void` functions are referred to as wrapper functions.

Essentially, the wrapper function takes the input parameters, calls the gameplay function, checks the returned result and based on it, prints the necessary piece of poetry.

LHR2. Poetry functions. You shall implement two general functions

```
void printHaiku();
void printLimerick();
```

`printHaiku()` shall print the haiku used in our Haiku or Limerick games. `printLimerick()` shall print the limerick. The text of both the haiku and the limerick is provided below.

These functions will simplify the code in the wrapper functions - this way, your program shall have only one place where each poem is printed. This makes it easier to replace one or both poems used in the games.

LHR3. Haiku and Limerick. Use the following poetry as output of your program.

Haiku: as the haiku use the following[1]:

```
encrypt shall you
is the cipher secure?
i do not agree
```

Limerick: as the limerick use the following[2]:

```
A young man from Pacific Northwest
Once decided to pass Turing test...
"I blame John von Neumann,
That I can't prove I'm human",
He confessed after failing his quest.
```

Print two empty lines after the last line of each poem.

Game 1. The first Haiku or Limerick game you will implement has the following rules. The game has three `int` input parameters referred to in what follows as `first`, `second` and `third`.

Among the rules outlined below, the rule with the lowest number always has a precedence. (For example if Rule 1 is ""`first`" is odd" and Rule 2 is "`second` is even", input 3,4,5 should be dealt with according to Rule 1).

Rule 1. If `first` is the largest number of the three, output the **limerick**.

Rule 2. If `second` is greater than `third`, output the **haiku**.

Rule 3. If `first + third` is even, output the **limerick**.

Rule 4. Output the **haiku**

Function names. Use the following functions for this game (the first function is the wrapper, the second function is the gameplay function):

```
void lhGame01(<Input parameters>);
char lhGameplay01(<Input parameters>);
```

(replace `<Input parameters>` with appropriate function parameter declarations in each case).

`char lhGamePlay01()` shall return `'l'` if the limerick it to be printed, otherwise it shall return `'h'`.

Game 2. The second game has the following rules. The game has four `int` input parameters referred to in what follows as `first`, `second` and `third` and `fourth`.

Among the rules outlined below, the rule with the lowest number always has a precedence. (For example if Rule 1 is ""`first`" is odd" and Rule 2 is "`second` is even", input 3,4,5,6 should be dealt with according to Rule 1).

[1] http://dissertationhaiku.wordpress.com/2009/12/29/computer-science-7/

[2] Sorry for inflicting this on you, but this one is mine.

Rule 1. If either first or second is even and both third and fourth are odd, output the haiku.

Rule 2. If both first and second are odd, output the limerick.

Rule 3. If fourth is greater than the average of the other three values, output the haiku.

Rule 4. If third is the smallest of the four numbers, or if second is the greatest, output the limerick.

Rule 5. Output the haiku.

Function names. Use the following functions for this game (the first function is the wrapper, the second function is the gameplay function):

```
void lhGame02(<Input parameters>);
char lhGameplay02(<Input parameters>);
```

(replace <Input parameters> with appropriate function parameter declarations in each case).

char lhGamePlay02() shall return 'l' if the limerick it to be printed, otherwise it shall return 'h'.

Game 3. The third game has the following rules. The game has three char input parameters referred to in what follows as first, second and third.

Among the rules outlined below, the rule with the lowest number always has a precedence. (For example if Rule 1 is ""first" is an 'a'" and Rule 2 is "second is not a 'b'", input 'a','c','b' should be dealt with according to Rule 1).

We assume that the inputs are lowercase letters of English alphabet.

Rule 1. If exactly one of the three inputs is a vowel[3], output the limerick.

Rule 2. If all inputs are consonants and at least one of them comes from the bottom half of the alphabet ('n' --- 'z'), output the haiku.

Rule 3. If at least two inputs are the same character, output the limerick

Rule 4. If first is not a 'w' but at least one other character is a 'w', or if third is either 'x', 'y' or 'z', output the haiku.

Rule 5. Output the limerick.

Note. A convenient way simplify this one, is to create a function isVowel() which checks if a char value is a vowel and use it.

Function names. Use the following functions for this game (the first function is the wrapper, the second function is the gameplay function):

```
void lhGame03(<Input parameters>);
char lhGameplay03(<Input parameters>);
```

(replace <Input parameters> with appropriate function parameter declarations in each case).

char lhGamePlay03() shall return 'l' if the limerick it to be printed, otherwise it shall return 'h'.

[3]For the purpose of this assignment, vowels are 'a', 'e', 'i', 'o', 'u' and 'y'.

What to submit. You will submit the `poetry.h` file containing the declarations and the definitions of all the functions discussed above (note that for this assignment you are allowed to create any other functions that might help you). In addition, you will also create and submit a C program `testPoetry.c`, which will contain a single function, `int main()`. The body of `main()` shall consist of tests of the three Limerick or Haiku games. A single test is a call to the approriate `lhGameOX()` function with a specific set of input parameter values. You shall include at least 10 tests per game. Generally speaking, your tests shall cover every execution path.

Submission.

Who submits. Each pair should submit only one set of files! However, all files must be submitted by the same person (otherwise, it will be very hard for me to collect them all).

Files to submit. You shall submit the following files: `circle.h`, `testCircle.c`, `converter.h`, `testConverter.c`, `poetry.h`, `testPoetry.c` and `team.txt`.

team.txt file. Your `team.txt` file shall contain the list of students in your team (pair). For each student, the file shall list the name, and the Cal Poly loginId. E.g., if I were on a team with Clark Turner, we would submit the following `team.txt` file:

```
Alex Dekhtyar, dekhtyar
Clark Turner, csturner
```

Submission procedure. You will be using `handin` program to submit your work. The procedure is as follows:

- `ssh` to `unix1` (`unix2`, `unix3` or `unix4`) (`unix1.csc.calpoly.edu`).
- Execute the `handin` command:

  ```
  > handin dekhtyar lab05 <files>
  ```

  Please, DO NOT submit binary files.

Grading

The grade for the lab is formed as follows:

Converter	20%
Circle	30%
Poetry	50%

Any submitted program that does not compile earns 0 points.

All programs will be checked for style conformance. Any style violation will be noted. The program will receieve a 10% penalty.