CS 10 - Assignment 8

Collaboration Policy

Collaboration between students on programming assignments is **NOT** allowed under any circumstances - you may not even discuss your work with others; you may **NOT** get coding assistance, or copy code, from **ANY** outside source *(books, web sites, current or past students, your own previous submissions if you are repeating the course, etc.)*. We test every submission for copying and when we find it we treat it as flagrant academic dishonesty on the part of ***all*** parties involved, regardless of their roles as provider or consumer. The penalty is generally instant failure in the course, in addition to the usual sanctions applied by Student Judicial Affairs.

As you can see from the submission header below, we ask you to essentially take an oath that the code you submit is ***entirely your own work*.**

At the same time, we certainly understand very well that you will frequently need help completing your programming assignment, so we provide channels where you may get that help in a way which will strengthen your programming skills: instructor and TA office hours, and the discussion forums where you can ask all the questions you want about the assignment, and even help others with the understanding that you have gained *(but not, of course, with any actual code)*.

Assignment Submission Instructions

Your assignment must be submitted as a ***simple text file*** named ***main.cpp***
Files in ANY other format (MS Word document, etc.) or with ANY other name (main, main.doc, main.txt, etc.)
will ***not*** be graded.

Submit your work via the appropriate iLearn assignment link, making sure you use the correct link.

We strongly recommend that:

1. you submit at least *6 hours* before the deadline, even if you haven't completed the program - you can re-submit as often as you like, and we will only grade the final submission
2. once you have submitted your final attempt, go back and download it back to your system: then run it just to make absolutely sure that it really is the file you intended to submit!

You are budding professionals, and are expected to take full responsibility for abiding by the requirements of a contract.

The ***only reason we will ever accept*** for a missed deadline is if the system administrators inform me that either the iLearn system or the CS department servers were off-line for a significant period around the time of the deadline *(In which case we will probably have notified you of alternative procedures)*.

Remember to include the following header information at the top of your program
(DO NOT REMOVE THE // at the beginning of each line, these tell the compiler to ignore these lines)

```
// Course: CS 10 <quarter & year>
//
// First Name:
// Last Name:
// Course username: <enter the username you use to login in the lab>
// Email address: <enter your cs or UCR student email address here>
//
// Lecture Section: <e.g. 001>
// Lab Section: <e.g. 021>
// TA:
//
// Assignment: <assn1, hw2, lab3, etc.>
//
// I hereby certify that the code in this file
// is ENTIRELY my own original work.
//
// =================================================================
```

NOTE: This header MUST appear at the top of EVERY file submitted as part of your assignment
(don't forget to fill in *your* details and remove the <> brackets!!).

Copy & paste this header into your file then update personal details.

Assignment Specifications

For this assignment you will write a simple text adventure game. Within this game the player will travel from room to room choosing which door to traverse through. Once through a door the player cannot go back through the door that she came through.

Game Specifications

Every room in the game has 4 doors for the player to choose from, each associated with an **uppercase** direction: N, E, S or W. Three of the doors open up to another room, and the 4th door is an exit from the game. Of the three non-exit doors, one will send the player to a room containing a monster. Another will send the player to a room containing a genie. The third non-exit door will send the player to a room that has a picture.

The player will begin the game holding two types of items, bananas and oranges. The player should start out with 5 bananas and 3 oranges.

For simplicity. There are two possible alignments of the 4 doors. At the beginning of **each** turn this random alignment is decided. For clarification on implementation we will refer to these alignments as 0 and 1.

Alignment 0	**Alignment 1**
`N: Monster`	`N: Exit`
`S: Genie`	`S: Monster`
`E: Picture`	`E: Genie`
`W: Exit`	`W: Picture`

IMPORTANT: Only 1 random alignment is chosen per turn. If the user does not enter a valid directional door choice (N/E/S/W), keep asking for a direction but **DO NOT** get a new random alignment value.

Door Actions

The game should ask the player to choose a door to step into. If the player steps into a room with the monster inside, the monster will steal all of the bananas and oranges the player is holding. If the player steps into the room with the genie inside, the genie will give the player 2 additional bananas and 1 additional orange. If the player steps into the room with the picture inside, the game should just draw the picture to the screen but no other reward is granted. Finally, if the player steps through the door to the exit, the game should end. When the game ends, give the player her combined and individual scores. The combined score is the total number of bananas and oranges the player is holding when finding the exit.

Functions

You must write a function for each door. When the player chooses a door in a certain direction (N/E/S/W) -- and only one of those values -- the program should call the function for the door in that direction or end the program if she chooses the door to the exit. Some of these functions will use reference variables for all or some of their parameters.

You MUST use these function names and **use the specified number and types of parameters** for each function. Here is what each function must do:

- `monsterRoom`: Returns nothing, takes two integers both by reference and a string by value. The function reduces the values of the variables storing the number of bananas and oranges to 0. This function should also tell the player what monster attacked them.

- `genieRoom`: Returns nothing, takes two integers both by reference. Increase the values of the variables storing the number of bananas and oranges by the appropriate number as per the game specification. The function should also output the message telling the player what she encountered in the room and what happened.

- `pictureRoom`: Takes no parameters and returns nothing. This function must output the proper announcement and draw a picture to the screen. We have provided a `drawPicture` function for your `pictureRoom` function to call the provided function actually does the picture drawing. Your function should also tell the player what happened.

  Here is the `drawPicture` function, you should copy and paste (make sure spaces are copied as well, for those having problems with that) it prior to the `pictureRoom` function call where it is utilized:

```
void drawPicture()
{
    cout << "       _--~~--_" << endl;
    cout << "     /~/_|  |_\\~\\" << endl;
    cout << "    |____________|" << endl;
    cout << "    |[][][][][][]|" << endl;
    cout << "  __| __         |__" << endl;
    cout << " |  ||. |   ==   |  |" << endl;
    cout << "(|  ||__|   ==   |  |)" << endl;
    cout << " |  |[] []  ==   |  |" << endl;
    cout << " |  |____________|  |" << endl;
    cout << " /__\\            /__\\" << endl;
    cout << "  ~~              ~~" << endl;
    cout << endl;
}
```

Implementation Hints

- You will need variables to store the number of bananas and oranges.
- Follow every input statement by outputting a blank link through an output statement.
- Some of the functions will use reference variables for all or some of their parameters.
- **DO NOT** use global variables
- Before the adventure begins, ask the user for their name and the scariest monster they can think of, storing each as a string. You will pass the monster name to the associated function when the monster attacks the player.

Testing Specifications

As with all games we wish them to be as unpredictable as possible. However, when testing to verify the program is in working condition we must have a measure of predictability. To achieve similar results to the ones shown within the input and output samples you should utilize a random seed value of 500 (`srand(500);`). This is for testing purposes only! You should **maintain the unpredictability on turn-in** and submit with with `srand(time(NULL))` as the seed call.

Sample Inputs and Outputs

Due to their length they are included at the end of the specification.

Testing Your Solution

Testing this program will involve unit testing as well as full program output testing. Output and input files will be provided on iLearn for your benefit. Please download the files to test your program. These tests are just a few of the possibilities, we expect you to to test other possibilities on your own as well as implementing the functions as defined in the specification. Recall the `diff` command is utilized to compare two files, which allows comparison of a solution's output and your program's output.

```
diff -Bbiau my.out solution.out
```

You should not do this until you have your program complete. If your program has an infinite loop, you will generate excessively large files and experience many problems.

Step 1 - *Compile your program*: `[user@well]$ g++ main.cpp`

Step 2 - *Execute With Input Redirection*: `[user@well]$ ./a.out < input.txt`
DO NOT skip Step 2, this will assure no infinite loops exist.
If program does not execute to completion, fix your program and go to Step 1.

Step 3 - *Use Input & Output Redirection*: `[user@well]$ ./a.out < input.txt > my.out`
As you can see, output redirection is similar except the right angle bracket is used and points to the file where output should go.

Step 4 - *Compare the files with `diff`*: `[user@well]$ diff -Bbaiu my.out solution.out`

```
--- mine.out         2012-11-10 17:22:41.582953000 -0800
+++ solution.out     2012-11-10 17:22:29.870525000 -0800
@@ -19,7 +19,7 @@
Phrase: he---
-uessed so far: he
+Guessed so far: he
Wrong guesses left: 7
Enter a guess:
```

If there are differences between the two files they will be shown. The first two lines show a key for reading the output. The character preceding the file denotes the character that will precede lines in differences for that file. In this example the Lines preceded by a `‘-’` correspond to the `mine.out` file, similarly the `‘+’` belongs to solution output. If a line is not preceded by a key character then it exists in both.

If no output is produced by the `diff` the files match.

Step 5 - *Lots of differences*: `[user@well]$ diff -Bbaiu mine.out solution.out | less`

Always try Step 4 before Step 5. If there are lots of differences it can be hard to read in the terminal. In that case we send the output of the command to a text reading program such as `less`. This will allow you to use the arrow keys to navigate up and down the output.

Press `‘q’` to quit the `less` program and return to the terminal line.

Marking Guidelines

- Make sure your code compiles.
- Make sure your output matches the format of the examples: spacing, spelling, etc.
- We utilize the `diff` command to grade in some portions
- We will unit test your functions so make sure they meet the specification as they will be tested individually (separated from your program)

Basic Style Guidelines

- **indentation** - follow proper 4 space indentation as you go into loops or branches
- **spacing** - blank lines should be used to separate logic blocks
- **no line wraps** - no line of code should have more than 80 characters
- "**meaningful**" variable **names**
- **comments** - each logical block of code should have a brief explanatory comments

Monster then Exit Input

```
Kris
Pinhead
N
N
```

Monster then Exit Output

```
Please enter your name:
Name your scariest monster:
Kris, you are in a room with 4 doors.
You are carrying 5 bananas and 3 oranges.

Pick a door to enter by typing the direction it is in (N/E/S/W):
WATCH OUT!!
Pinhead attacks you and steals all of your bananas and oranges.

Kris, you are in a room with 4 doors.
You are carrying 0 bananas and 0 oranges.

Pick a door to enter by typing the direction it is in (N/E/S/W):
You found the exit!
Your score is 0 (0 bananas and 0 oranges).
Bye bye!!!
```

Whole Game Input

```
Kris
Pinhead
N
a
b
E
W
W
```

Whole Game Output

```
Please enter your name:
Name your scariest monster:
Kris, you are in a room with 4 doors.
You are carrying 5 bananas and 3 oranges.

Pick a door to enter by typing the direction it is in (N/E/S/W):
WATCH OUT!!
Pinhead attacks you and steals all of your bananas and oranges.

Kris, you are in a room with 4 doors.
You are carrying 0 bananas and 0 oranges.

Pick a door to enter by typing the direction it is in (N/E/S/W):
Pick a door to enter by typing the direction it is in (N/E/S/W):
Pick a door to enter by typing the direction it is in (N/E/S/W):
!!POOF!!
A genie pops out and grants you 2 bananas and 1 orange.

Kris, you are in a room with 4 doors.
You are carrying 2 bananas and 1 oranges.

Pick a door to enter by typing the direction it is in (N/E/S/W):
You found a picture!

       _--~~--_
     /~/_|  |_\~\
    |____________|
    |[][][][][][]|
  __| __         |__
 |  ||. |   ==   |  |
(|  ||__|   ==   |  |)
 |  |[] []  ==   |  |
 |  |____________|  |
 /__\            /__\
  ~~              ~~

Kris, you are in a room with 4 doors.
You are carrying 2 bananas and 1 oranges.

Pick a door to enter by typing the direction it is in (N/E/S/W):
You found the exit!
Your score is 3 (2 bananas and 1 oranges).
Bye bye!!!
```