CS 161 Assignment #2 – Text Adventure Game

Design Document due: Sunday, 1/19/2020, 11:59 p.m. (Canvas)
Design Peer Review due: Weds., 1/22/2020, 11:59 p.m. (Canvas)
Assignment due: Sunday, 1/26/2020, 11:59 p.m. (TEACH)
Demo/Grade Your Assignment by: 2/09/2020, 11:59 p.m. (In person)

Goals:

- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
  - Run tests and assess results to iteratively improve the program
- Use conditional statements to change program behavior based on user input
- Use random numbers to change program behavior without user control

Part 0. Sign Up for Your Demonstration

Every assignment in this course is graded by demonstrating your work for 10 minutes with a TA. You are required to **meet with a TA within the two weeks following the due date** to demo. Please sign up for your slot now (yes, before you begin the assignment!) to ensure your work will be graded. You can schedule a demo with a TA on the course website (http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020) in the far right column of the bottom table labeled "Grading Hours". You can select one of your lab TAs, or a different TA. Be sure to specify

- **Tip:** Pick an early slot before they are all taken! The earlier your demo, the sooner you get feedback, which will help you do even better on assignment 3. ☺
- **Demo Outside 2 Weeks:** Assignments that are demonstrated later than the acceptable time period will be receive a 50 point deduction. (Also, we cannot guarantee a late grading slot.)
- **Cancel/reschedule demo**: Must be done at least 24 hours in advance, or it will be considered a missed demo.
- **Missing a Demo:** If you miss your scheduled demo with a TA, you will receive a 10 point (one letter grade) deduction to that assignment for each demo missed.
- **No demo**: If you do not demonstrate your assignment, you will receive a 0.
- Submissions that do not compile on the ENGR servers will receive a 0 for the implementation part. Please test your code before submitting.

Your goal in the demonstration is to show your TA how to use your program, explain how it works, and answer questions. Be proud of your work and ready to show it off!

Part 1. (10 pts) Design a Text Adventure Game

Have you ever played the Oregon Trail game? (If not, you can try it out here: https://archive.org/details/msdos_Oregon_Trail_The_1990) It is a challenging adventure from Independence, Missouri, to Oregon's Willamette Valley. The player has to make decisions about supplies, navigation, and rations (and hunt for food) along the way. In this assignment, you will create your own short game that poses similar decisions and challenges for your players. (Note: your game need not have graphics and will be much shorter than the Oregon Trail game!)

You will choose what the theme and challenges will be in your game. Consider:

- **Setting**: Where will your game take place? In a city? A dorm room? On Venus?
- **Goal**: What is the player's goal? To reach a specific location? To find a specific object? To explore?
- **Scoring**: How will the player earn (or lose) points?

Here are the attributes your game must have:

- Compute the **player's score** and report the score after each decision and at the end of the game.
- Have **at least 2 paths** from start to end of the game that require the player to **make 3 choices** (some paths might end sooner and be shorter).
  For example, one "path" with 3 choices might be:
  - Welcome to Mythago Wood! You have 0 points.

    You are in a forest and see a cottage. Do you:
    (1) Knock on the door, or
    (2) Keep walking?
    [user enters "1"]
    You gain 10 points for a score of 10 points.

    When you knock on the door, it creaks open to reveal a smoky, dark interior with a fire burning in the fireplace and the delicious smell of stew cooking. A gruff voice says, "Come in. What did you bring me today?"
    You say:
    (1) "I brought you a bottle of wine."
    (2) "I'm hungry. Give me some of your stew."
    (3) "I'm sorry, I didn't bring anything today."
    [user enters "2"]
    You lose 6 points for a score of 4 points.

    A growl fills the cabin, and a strong man in well-used work clothes stands up from his bowl of stew. "No food for you! Now get out!"
    What do you do?
    (1) Steal the bowl of stew and run back outside.
    (2) Apologize deeply and explain that you have no money and are starving.
    [user enters "2"]
    You gain 5 points for a score of 9 points.

    The man calms down and offers you a fresh bowl of stew.

    The end! Your final score is 9 points.
- Your program would output a different result if the user entered different choices at each of the prompts.
- Have an **element of chance** that changes the player's outcome.
  Example 1: In your game the player might choose between eating lunch or going on a hike. If they eat lunch, your program could randomly determine whether they enjoy their meal (and gain points in their score) or they feel sick afterwards (and perhaps lose points in their score).
  Example 2: In your game, the player decides to buy a lottery ticket. You award them a

random number of points, chosen between 0 and 5, to simulate the random reward of the lottery ticket.
- If the player enters an invalid choice, the game should quit.
- Have fun and be creative!

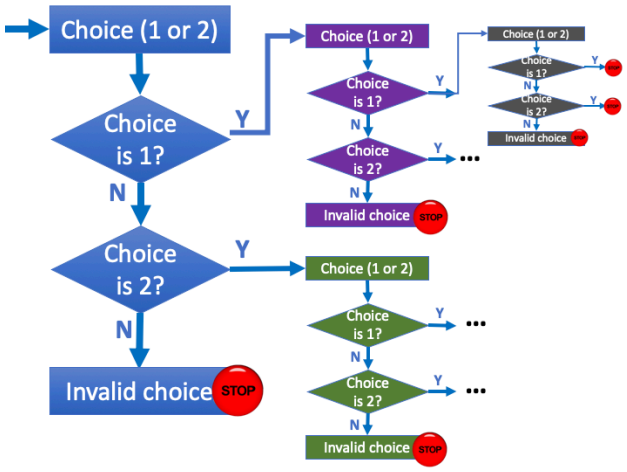
Your first step is to **write a Design Document** to plan your game. You can start on this right away. It does not require any programming; instead, it is your chance to plan your game. Consult the Design and Peer Review Guidelines:
http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/cs161-design-review-guidelines.pdf

and the sample Design Document:

http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/Example_Design_Document.pdf

Your Design Document must include three sections:

1. **Describe the problem** in your own words (what does your program need to do?) and any assumptions you are making to help make the problem more concrete. For this assignment, your design should explain the setting, goal, and scoring you have chosen.
2. **Devise a plan**: design your solution and show how it will work with either (1) pseudocode (not C++) and/or (2) a flowchart diagram. You can create your flowchart using software or sketch it on paper/whiteboard, take a picture, and insert it into your Design Document. **A flowchart is highly, highly recommended due to the branching nature of this program.**
   Partial example (all "…" should be filled in, in your design):



3. **Identify at least 8 test cases**. What kinds of input might a player try? (Think of typical inputs as well as all of the crazy and "wrong" things they might do! **The more tests you think of now, the better your program will be later.**) What do you want your program to do or output as a result? Since your program will have several places a player can provide input, you can list these prompts as distinct test cases. Example:

Test case setting	**Player input**	**Expected result**

Prompt: `It's 7 p.m. the night before your Physics midterm. What do you do?`
`(1) Review flash cards for the midterm`
`(2) Go see a movie`
	`1`	Increase score by 10 points

Prompt: `It's 7 p.m. the night before your Physics midterm. What do you do?`
`(1) Review flash cards for the midterm`
`(2) Go see a movie`
	`2`	Decrease score by 10 points

Prompt: `It's 7 p.m. the night before your Physics midterm. What do you do?`
`(1) Review flash cards for the midterm`
`(2) Go see a movie`
	`-1`	Output `"That is not a valid choice."` and quit the game (`return 1`).

Submit your Design Document on **Canvas** (not TEACH):
https://oregonstate.instructure.com/courses/1771939/assignments/7779087

Part 2. (10 pts) Provide a Peer Review of 2 Text Adventure Game Designs

You can work on this part in parallel with your work on Part 3. In this part, you will provide feedback on your classmates' designs (and receive feedback on your own design to help improve your final submission). **Your peer reviews do not affect the other student's grade.**

Peer reviews must be written **constructively and positively**. Your goal is to identify good aspects of the design, plus ways it can be improved. Negative, harsh, or mean comments are not appropriate and will cause you to lose points.

1. Describe the problem:
   - (1 pt) Write down something you like about this design. Is the game design very clearly written? Is the game setting something clever or fun?
   - (1 pt) Does the design explain the setting, goal, and scoring for the game? If one or more is missing, provide a positive, helpful suggestion for how it could be addressed. You don't have to fill in the answer (since it isn't your game), so instead you could write something like "Please say more about where the game takes place."
   - (1 pt) Does the document list assumptions? Do you have suggestions to clarify the assumptions?
   - (1 pt) If any parts are unclear, write down questions that will help the student clarify their description.

2. Devise a plan:
   - (2 pts) Can you understand the pseudocode or flowchart diagram? If hand-drawn, is the handwriting legible? Does it match the planned design? Are there parts that seem incomplete or confusing?
   - (1 pt) Write down at least one suggestion for how to improve the design. **There is always a way to improve any design!**

3. Test cases:
   - (2 pts) Do you understand the test cases, and do they match with the design as described? If so, write "All test cases look good." If not, suggest changes or improvements to clarify the test cases.
   - (1 pt) Propose one new test case not included by the student (e.g., use one of their "Test case settings" but change what the player types). You do not need to write the "Expected result" for this test case because that is up to the student to decide.

When you receive your peer reviews, **read them carefully and decide if you would like to modify your design (and implementation).** Often other people will spot issues you did not anticipate!

Part 3. (95 pts) Implement Your Text Adventure Game

In this part, you will implement your text adventure game in C++ following the design you have developed. **Note: It is normal (in fact, expected) that your design will evolve as you write the program and figure out new details.** The design provides a starting point, but you can deviate from it.

There are some specific requirements for your implementation **(read this carefully)**:

- Name your file `assign2_game.cpp`.
- (10 pts) Choose appropriate data types for the information you need to store (e.g., player score, player choices, random numbers). Think about what you learned during Assignment 1, in lecture, and in your reading about different data type ranges. **Just as in Assignment 1, include a justification and min/max for the chosen data type in a comment above the variable's declaration.**
- Player choices must be based on numbers (not letters or words).
- Output a blank line between each interaction so it is easy for the player to read (see above example).
- (15 pts) Use **nested if/then/else statements** and/or **switch statements** to respond to the player's inputs and direct the program's flow through your game.
- (25 pts) As stated above, your program must have **at least 2 paths** from start to end that require the player to **make 3 choices**.
- (10 pts) Track player score and output it when the game ends.
- (10 pts) Check for valid player inputs. If a player enters an invalid input, tell them it was invalid and quit the game (e.g., `return 1` from `main()`) immediately so they can try again. Use the "`else`" clause in an if/then/else statement or the "`default`" clause in a switch statement to catch invalid inputs.

Here is an example of **nested if/then** statements in C++:

```
if (choice_1 == 1) {
   …
   if (choice_2 == 1) {
      …
      if (choice_3 == 1) {
         …
      else if (choice_3 == 2) {
         …
      }
      else /* invalid choice */ {
      }
   }
   else if (choice_2 == 2) {
      …
```

```
    }
    else /* invalid choice */ {

    …
    }
}
```

- (10 pts) Use `rand()` to generate random numbers in your game. Note that you can obtain a random number from `x` to `y` (inclusive) with `rand()%(y-x+1) + x`. Remember to `#include <cstdlib>` so `rand()` will work and to seed the random number generator (see Lab 2).
- (15 pts) Use good programming style: comment header, comment blocks, informative variable names, appropriate indentation, lines no more than 80 characters long, blank lines between code sections, correct spelling (in output and comments), etc. Refer to the guidelines: http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/cs161-style-guidelines.pdf

Tips (read this section multiple times):

- Break the problem down. Start by getting your program working for one player decision, then add more. Notice that even with a single (working) if/then/else statement, you can get 70 of 95 possible points (if you satisfy all other requirements)!
- Use **indentation** and **good comments** to keep track of which choice you are working on inside your program.
- Remember the maximum line length of 80 characters. Instead of
  ```
  cout << "A growl fills the cabin, and a strong man in well-used
  work clothes stands up from his bowl of stew. " << endl;
  ```
  (line wraps to the next line), use
  ```
  cout << "A growl fills the cabin, and a strong" << endl
       << "man in well-used work clothes stands " << endl
       << "up from his bowl of stew. " << endl;
  ```
  Note where the **semi-colons** are and where they are not.
- If a choice has more than 2 options, you may find it more convenient to use a **switch** statement (instead of **if/then**). If you use **switch**, remember to use **break;** at the end of each case.
- To print quotation marks, use the escape character before " inside the string:
  ```
  cout << "This is a \"quoted string\"." << endl;
  ```
- Before you submit, read the peer reviews you received and ensure that you have addressed any issues they identified. This will help you improve your score on the assignment.

Part 4. (15 pts) Analyze Your Work (in a Word/text file, section "Part 4")

(A) (2 pts) Were 8 test cases enough to cover all possible user inputs for your game? If not, why not?

(B) (2 pts) As you worked to implement your game, you probably thought of new test cases after your Design Document was already submitted. List one here. (If you didn't think of any new test cases while implementing, create one now.)

(C) (9 pts) Try each of your 9 test cases (there are 9 now) and for each one, report whether or not the behavior is as expected. If not, state whether (1) your design has evolved and

now the expected behavior is different (state what the new expected behavior is) or (2) this test helped you find (and fix) a bug in your program.

- If you didn't have 8 test cases in your Design Document, make them up now to allow you to get full credit here.

(D) (2 pts) List one useful suggestion or idea you got from your peer reviews. If you didn't receive peer reviews, or didn't receive useful comments, instead list a new idea that came to you while you were working on your implementation that led to a change in your design. Example:
"While working on my if/then statements, I realized that offering 5 options for choice 2 was going to take too long to implement, so I reduced it to 2 options."

Part 5. Extra Credit (write in the Word/text file, section labeled "Part 5")

- (**up to 2 pts**) Ask another person (friend, family member, roommate, coffee shop stranger) to try out your game. In "Part 5" of your file, copy/paste their full interaction (program outputs and what they typed). Did the program crash or behave in a way you did not expect? Describe anything you found surprising.
- (**up to 2 pts**) Describe one improvement you would recommend to make your program better (more robust, more entertaining, more attractive, anything).
- **(up to 2 pts**) Modify your program so that instead of quitting when the player enters an invalid choice, the program asks them to try again with a valid choice. That is, the program should inform the payer that they entered an invalid choice, repeat the prompt, and keep doing this until a valid choice is made. Consider using a **do/while** loop for this capability.

Submit Your Assignment Electronically

(A) Submit your C++ program (.cpp file, not your executable) and your Word/text file (**must be converted to a PDF file**) with written answers (Part 4 and optionally Part 5) before the assignment due date, using **TEACH**.

(B) Remember to sign up with a TA on the course website to demo your assignment. **The deadline to demo this assignment is 02/09/2020.**