CS 161 Assignment #2 – Text Adventure Game

Design Document due: Monday, 4/13/2020, 8 p.m. (Canvas)
Program and Written Answers due: Monday, 4/20/2020, 8 p.m. (Canvas)
Demonstrate by: Wednesday, 4/29/2020, 8 p.m. (Zoom appointment)

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

Please sign up for your demonstration now (yes, before you begin the assignment!) to ensure your work will be graded. You can schedule a demo with a TA on Canvas (https://oregonstate.instructure.com/courses/1770357/pages/grading-slash-demo-signup). You can select one of your lab TAs, or a different TA. **Use your OSU email address to sign up.**

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. ☺

Brief reminders (see Syllabus and Assignment 1 for more details):

- **Demo After April 29:** 30-point deduction. (Also, we cannot guarantee a late grading slot.)
- **Demo Late Assignments:** Must be done by April 29, even if the submission was late.
- **Cancel/Reschedule Demo**: At least 24 hours in advance, or it will be a missed demo.
- **Missing a Demo:** 10-point deduction 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 (user input is highlighted):
  -
```
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?
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."
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.
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.
```

- The program outputs different results if the player makes different choices.
- 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 print an error message and quit. You can assume that the user will type an integer (not a floating point number, string, etc.), but it might not be a valid number for your options.
- 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 Guidelines:
https://oregonstate.instructure.com/courses/1770357/pages/design-document-guidelines

and the example Design Document:
https://oregonstate.instructure.com/files/79099302

Your Design Document must include three sections:

1. **Understanding the problem:** Describe the **goal** 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, you should explain **the setting, goal, and scoring** you have chosen.
2. **Devise a plan**: Design your solution and show how it will work with a **flowchart** diagram. You can create your flowchart using software or sketch it on paper/whiteboard, scan it in (preferred) or take a picture, and insert it into your Design Document. Also include your **strategy** for how you will approach and complete the implementation, including your estimate of the time it will take.
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**:
https://oregonstate.instructure.com/courses/1770357/assignments/7847900

Part 2. **(85 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).
- (10 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.
- (20 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-else** statements in C++:

```
if (choice_1 == 1) {
   if (choice_2 == 1) {
      if (choice_3 == 1) {
         …
      } else if (choice_3 == 2) {
         …
      } else /* invalid choice 3 */ {
         …
      }
   } else if (choice_2 == 2) {
      …
   } else /* invalid choice 2 */ {
   …
   }
}
```

- (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 style guidelines: https://oregonstate.instructure.com/files/79125719/

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 65 of 85 possible points (if you satisfy all other requirements)!
- Do not use global variables (-5 pts) or goto (-5 pts). They are not needed, and they tend to create bugs that will take extra time for you to correct.
- Do not add functions, other than main(), nor classes/objects/structs. Keep it simple!
- 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 a single line:
  ```
  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 "
       << "man in well-used work clothes stands "
       << "up from his bowl of stew. " << endl;
  ```
  Note where the **semi-colons** are and where they are not.
- Use an indentation of 2-3 spaces. If you use 5 or more spaces per indentation, you will likely have trouble keeping to the 80-character line limit with nested if-then statements.
- 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, and include a **default** case to catch invalid choices.
- To print quotation marks, use a backslash (escape character) before " inside the string:
  ```
  cout << "This is a \"quoted string\"." << endl;
  ```

Part 3. **(15 pts)** Analyze Your Work (in a separate file)

(A) (1 pt) Write down the demonstration timeslot you signed up for. Include the name of the TA, the date, the time, and the Zoom link.

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

(C) (3 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 (with **setting, input, and expected result**). (If you didn't think of any new test cases while implementing, create one now.)

(D) (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.

Part 4. Extra Credit (in your separate file)

- (**up to 1 pt**) Ask another person (friend, family member, roommate) to try out your game. You can do this remotely using Zoom/Skype etc. to share your screen, run your program, and have them tell you what inputs to type. In a section called "Extra Credit" in 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 1 pt**) Describe one improvement you would recommend to make your program better (more robust, more entertaining, more attractive, anything).
- **(up to 1 pt**) Modify your program to ask the player for their name at the start of the game, store it in a string, and use the name when reporting the current score, e.g.:
  `Kiri, you gain 5 points for a score of 9 points.`
- **(up to 2 pts**) Modify your program to achieve **one** of these options, using a **do-while** loop:
  - Allow the user to play again if they want (at the end of the game, ask if they want to replay, and if so, start over).
  - Instead of quitting when the player enters an invalid choice, allow them to keep trying until they enter a valid choice. (You don't have to do this for every choice, but demonstrate it for at least one.)

Submit Your Assignment Electronically

(A) Convert the file containing your written answers to Part 3 and (optionally) Part 4 into PDF format.
(B) Submit your C++ program (**.cpp** file) and your **PDF file with written answers** before the assignment due date, using **Canvas**.
https://oregonstate.instructure.com/courses/1770357/assignments/7847904
(C) Remember to sign up with a TA to demo your assignment.
The last day to demo this assignment is 4/29/2020.