CS 161 Assignment #5 – Treasure Chest

Design Document due: Sunday, 3/1/2020, 11:59 p.m. (Canvas)
Design Peer Review due: Weds., 3/4/2020, 11:59 p.m. (Canvas)
Assignment due: Sunday, 3/8/2020, 11:59 p.m. (TEACH)

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 two-dimensional arrays to store information
- Define, allocate, access, update, and free your own data structures (structs)
- Manage dynamic memory: no memory leaks or dangling pointers
- Use functions to modularize code to increase readability and reduce repeated code

---

Part 0. No Demonstration for Assignment 5

Instead of a demo, you will write a README.txt file (see Part 4 for details). Your assignment will be graded offline, using the README as a guide for how to run and interact with the program.

Submissions that do not compile on the ENGR servers will receive a 0 for the implementation part. Please test your code before submitting! Comment out parts that do not work if necessary.

Part 1. (10 pts) Design a Treasure Chest

In this assignment, you will create a program that stores a representation of each item in a collection. You decide **what kind of items** will be stored in your treasure chest: books, DVDs, knitting needles, model cars, wine corks, etc. Pick something that interests you.

Next, decide what **attributes** each item will have: weight, color, number of pages, length, runtime in minutes, number of wheels, etc. You should pick at least 4 attributes, one of which is the item's value (in units of your choice).

The objects are stored in a treasure chest that your program will represent with a dynamically allocated two-dimensional array. The program begins by prompting the user for the desired size of the treasure chest.

The program will provide a menu that allows the user to:

1. Add an item: prompt the user for a row and column, then prompt the user for the value of each attribute for the item. Store the item's data at the appropriate place in the array.
2. Remove an item: prompt the user for a row and column; if an item is present, remove it (reset attributes to initial/empty values).
3. Generate a random item: prompt the user for a row and column and create a new item with randomly chosen values for its attributes.
4. Show an individual item: prompt the user for a row and column, then output the data stored for that item (its attributes).
5. <Your choice here>: give the user something else they can do to an item (Modify an attribute in a particular way? Copy an item from one location to another?). Your choice!

Example run:

- Here, the menu is displayed in a compact way; you choose how to display your menu.
- Here, each item is displayed with the first letter of its name; you choose how to display items and the treasure chest as part of your design.
- Here, the "terraform planet" option would change the color of a planet to "green."

```
Welcome to the Planet Treasure Chest!
 Enter number of rows: 3
 Enter number of columns: 3
|_|_|_|
|_|_|_|
|_|_|_|
Total value of 0 items: $0


1) Add planet, 2) Remove planet, 3) Add random planet, 4) Show planet,
5) Terraform planet, 6) Quit
Please make your choice: 1
 Enter row: 2
 Enter col: 1
 Enter name (string): Tweedle
 Enter value ($, float): 100
 Enter radius (float): 17256.3
 Enter color (string): brown
|_|_|_|
|_|_|_|
|_|T|_|
Total value of 1 item: $100

1) Add planet, 2) Remove planet, 3) Add random planet, 4) Show planet,
5) Terraform planet, 6) Quit
Please make your choice: 3
 Enter row: 1
 Enter col: 1
 Random planet: name: "Dee", value: $27, radius: 243.3 km, color: orange
|_|_|_|
|_|D|_|
|_|T|_|
Total value of 2 items: $127

1) Add planet, 2) Remove planet, 3) Add random planet, 4) Show planet,
5) Terraform planet, 6) Quit
Please make your choice: 4
 Enter row: 2
 Enter col: 1
 Planet: name: "Tweedle", value: $100, radius: 17256.3 km, color: brown
|_|_|_|
|_|D|_|
|_|T|_|
Total value of 2 items: $127

1) Add planet, 2) Remove planet, 3) Add random planet, 4) Show planet,
5) Terraform planet, 6) Quit
Please make your choice: 6
```

Your first step is to **write a Design Document** to plan your program. You can start on this right away, since it does not require any programming, just thinking.
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.
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.
    - **Include a description of your new data structure and its members.**
    - **You should have a separate section of pseudocode, or a flowchart, for each function**.
    - Things to consider in your design:
        - What if the user wants to add an item where one already exists?
        - What if the user tries to remove an item from an empty location?
3. **Identify at least 8 test cases**. Unlike previous assignments, these test cases need to describe what the current state of memory is (what is stored in the treasure chest) as part of the setting. Examples:

Test case setting	User input	Expected result
(1) Empty treasure chest.	`1`	Allow user to input values for a new item.
(2) Treasure chest is 3x3 and has only one item, in location [1][1]. User selects "2" (remove item).	Row: `2` Column: `2`	Output `"Nothing to remove at location [2][2]."` and prompt user to try again.
(3) Treasure chest is 3x3 and has only one item, in location [1][1]. User selects "2" (remove item).	Row: `5` Column: `2`	Output `"Invalid coordinates [5][2]."` and return to main menu.

Your tests must be different from the examples included above. The more tests you think of now, the better your program will be later.

Submit your Design Document on **Canvas** (not TEACH):
https://oregonstate.instructure.com/courses/1771939/assignments/7826749

Part 2. (10 pts) Provide a Peer Review of 2 Treasure Chest 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.**

Canvas will randomly assign you two other students' design documents to review. Please enter your review as **comments (not rubric)** on the Canvas page where their submission is.

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. (3 pts) Describe the problem:
   - Does the document list assumptions? Do you have suggestions to clarify the assumptions?
   - If any parts are unclear, write down questions that will help the student clarify their description.
2. (3 pts) Devise a plan:
   - 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?
   - Write down at least one suggestion for how to improve the design. **There is always a way to improve any design!**
3. (4 pts) Test cases:
   - 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.
   - Propose one new test case not included by the student (e.g., use one of their "Test case settings" but change what the user 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. (115 pts) Implement Your Treasure Chest

Implement your Treasure Chest 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.

Implementation requirements (get these basics working first):

- Name your file `assign5_treasure.cpp`
- (10 pts) Define your own struct to hold the item type you decided to model.
  - The struct should include at least 4 members, including the value of the item
  - Include comments to explain why each data type was chosen for each member.
  - Don't forget the semi-colon at the end of your struct declaration.
- (5 pts) Dynamically allocate a 2D array of your data type (struct) to store your treasure chest (not from the stack, and not a 1D array).
- (10 pts) Use good memory management.
  - Initialize all memory (static or dynamic) before it is used by the program.
  - Delete memory off the heap when you are done using it.
  - Set pointers to NULL if they are not pointing to a valid (allocated) memory location.
  - Use valgrind to check for memory leaks, and then fix them.
  - Memory leaks or dangling pointers will cause you to lose points.
- Implement a function for each of the following. Each function should take your 2D array and dimensions (rows, cols) as arguments, then prompt the user for any inputs needed.
  1. (10 pts) Allow the user to add a new item (prompt for its attributes and store them as struct members).
  2. (10 pts) Allow the user to remove an item.
  3. (10 pts) Allow the user to add a randomly generated item. We've worked with randomly generated integers. Here are some tips for other data types:

- Random float between 2.3 and 4.0 (inclusive): `((float)rand()%18)/10 + 2.3`
- Random name/color (or other string): initialize a static array of strings with color names you want to use. Then generate a random number from 0 to the number of color names, minus 1, and use this random number to index into your list and assign the new item's member to it.
- Random Boolean: `(rand()%2 == 0)`

4. (10 pts) Allow the user to display an individual item (printing its member values).
5. (10 pts) Perform the new action you designed.
6. Add helper functions as needed to avoid code duplication and to break the code into readable small sections (modularize).

- (10 pts) Display the collection (using a visualization of your choice) each time the menu is printed.
- (5 pts) Display the total value of the collection and number of items each time the menu is printed. These should be freshly computed, each time the menu is printed, by iterating over the current 2D array (tip: this is a good place to create a function).
- (10 pts) Handle user input:
  - You can assume that the user will enter a value that is the correct **type** of data that you request. When reading in member values for your struct, prompt the user with the type you are expecting (see example above).
  - When there is a **range** constraint (valid values) on what the user can input (e.g., *menu choice, row/column coordinates, positive number for "radius", etc.)*, you must check that the input has a valid value and either prompt the user to re-enter an input if it isn't valid (Tip: Use a do-while loop) or return to the main menu.
- (15 pts) Employ good programming style and follow the course style guidelines: http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/cs161-style-guidelines.pdf
  - No function should be more than 25 lines long (*excluding lines that consist only of comments and whitespace*). If you find your functions are getting too long, break them into multiple smaller functions. Your `main()` method can be longer, but you are encouraged to split off sections of `main()` into smaller functions, which makes your code easier to read and easier to debug.
  - Do not put more than one statement on each line.

Implementation don'ts:

- No use of global variables, goto, or recursion.
- No use of classes, only structs.
- Don't try to solve the whole program at once. Start with adding support for one menu option at a time. Get that option working (and tested), then move on to the next one. Use your design as a guide.

Creativity time:

- Feel free to improve the visualization of your treasure chest. You can use color, larger spaces to hold/display items, etc.
- When you show a single item, is there a fun way you could render it (beyond just outputting values?). You can use punctuation marks to draw shapes. Bigger items could take up more screen space. Items with a color could be printed in that color. Etc.
- You can add more options to the menu - what else might a collector want to do with their items? (See Extra Credit section)

Part 4. (10 pts) README instructions

Instead of a demo, you will write a README.txt file that includes the following information:

- Your name, ONID, section (CS161-020), assignment number, and due date

1. **Description**: One paragraph advertising what your program does (for a user who knows nothing about this assignment, does not know C++, and is not going to read your code). Highlight any special features.
2. **Instructions**: Step-by-step instructions telling the user how to run your program. Each menu choice should be described. If you expect a certain kind of input at specific steps, inform the user what the requirements are. Include examples to guide the user.
3. **Limitations**: Describe any known limitations for things the user might want or try to do but that program does not do/handle.
4. **Extra credit**: If your program includes extra credit work (more options, luck event, etc.), describe it here for the user.

Part 5. (15 pts) Analyze Your Work (in a Word/text file, section "Part 5")

(A) (2 pts) As you worked to implement your Treasure Chest, 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.)

(B) (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.

(C) (4 pts) Introduce a bug in your program that causes it to crash when it runs (e.g., invalid access into your 2D array). Compile your program with -g and then run your program in gdb (see Lab 8). When the program crashes, copy and paste the error that is generated into your document. Then generate a stack trace and copy and paste this stack trace into your document.

- Does gdb's output lead you to the right program line where you introduced the bug?
- Write down the order of function calls that led to the place where the program crashed. (e.g., func2 -> func2 -> func3 …)

Part 6. Extra Credit

- **(up to 2 pts)** Add another menu option (beyond the 5 that are required) that lets the user do something else with their collection. Document this in your README.txt file.
- (**up to 2 pts**) Add a luck aspect to the game. Each time the user makes a menu choice, generate a random number to see if a **luck event** (something outside the user's control) happens (you choose how likely this is to happen). Your luck event could be good or bad. Examples (these are ideas; you only have to implement one, not all):

- One of their items breaks (is destroyed). If so, notify the user and update the treasure chest to remove that item.
- One of their items doubles in value. If so, notify the user and update the item's value member.
- Your idea here!

To receive these extra credit points, describe and document your luck event in your README.txt file.

- (**up to 2 pts**) Ask another person (friend, family member, roommate, coffee shop stranger) to try out your program. In "Part 6" of your Word/text file (created in Part 5), 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 (in "Part 6" of your Word/text file) one improvement you would recommend to make your program better (more robust, more entertaining, more attractive, anything).

Submit Your Assignment Electronically

(A) 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.

(B) Ensure that your program (1) compiles, (2) does not generate run-time errors, (3) has no memory leaks.

(C) Submit your C++ program (.cpp file, not your executable), your **README.txt** file, and your Word/text file (**must be converted to a PDF file**) with written answers (Part 5 and optionally Part 6) before the assignment due date, using **TEACH**.