CS 161 Assignment #5 – The Showcase

Design Document due: Tuesday, 5/26/2020, 8 p.m. (Canvas)
Program and Written Answers due: Monday, 6/1/2020, 8 p.m. (Canvas)
No demonstration (graded offline using your README.txt as a guide)

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 3 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 Showcase

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 showcase: 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 dollars.

The items are stored in a showcase 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 showcase. Next, each item in the array is given randomly generated attribute contents, and the user starts with $0 in their bank account.

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

1. Show an individual item: prompt the user for a row and column, then output the data stored for that item (its attributes).
2. Sell an item: Prompt the user for a row and column and increase the user's bank account by the item's value, then reset the item's attributes to zero/empty values. This location can now be used to add items.
3. Buy an item: To add an item, the user must pay a fixed fee (you decide), and there must be an empty location to hold the new item. If the user's bank account contains enough funds, then prompt the user for a row and column. If this location is empty, generate random values for each attribute for the new item. (If it wasn't empty, re-prompt the

user for a location or allow them to cancel the transaction.) Decrease the user's bank account by the fixed fee.

4. <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? Compare two items?). 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 its name; you choose how to display items and the showcase as part of your design.
- Here, the "terraform planet" option would change the color of a planet to "green."
- User input is highlighted.

```
Welcome to the Planet Showcase!
 Enter number of rows:    3
 Enter number of columns: 3
|Jupiter|Lucia  |Sylmar  |
|Earth  |Pluto  |Sydney  |
|Mars   |Tweedle|Alleycat|
Total value of 9 items: $847.  Your bank account: $0.

Choose 1) Show, 2) Sell, 3) Buy, 4) Terraform, 5) Quit: 1
 Enter row:    2
 Enter column: 1
 Planet: name: Tweedle, value: $100, radius: 17256.3 km, color: green
|Jupiter|Lucia  |Sylmar  |
|Earth  |Pluto  |Sydney  |
|Mars   |Tweedle|Alleycat|
Total value of 9 items: $847.  Your bank account: $0.

Choose 1) Show, 2) Sell, 3) Buy, 4) Terraform, 5) Quit: 3
 You need $50 to buy a planet.  Please try again.

Choose 1) Show, 2) Sell, 3) Buy, 4) Terraform, 5) Quit: 2
 Enter row:    2
 Enter column: 1
 You sell Tweedle for $100.
|Jupiter|Lucia  |Sylmar  |
|Earth  |Pluto  |Sydney  |
|Mars   |_______|Alleycat|
Total value of 8 items: $747.  Your bank account: $100.

Choose 1) Show, 2) Sell, 3) Buy, 4) Terraform, 5) Quit: 3
You will be charged $50 to buy a planet.  Proceed (y/n)? y
 Enter row:    1
 Enter column: 0
That location is taken.  Please try again.
 Enter row:    2
 Enter column: 1
 New planet: name: Dee, value: $27, radius: 243.3 km, color: red
|Jupiter|Lucia  |Sylmar  |
|Earth  |Pluto  |Sydney  |
|Mars   |Dee    |Alleycat|
Total value of 9 items: $774.  Your bank account: $50.

Choose 1) Show, 2) Sell, 3) Buy, 4) Terraform, 5) Quit: 5
```

Here are the attributes your program must have:

- Use **loops** (for, while, or do-while) to avoid code duplication
- Use **functions** to define modules of code (and avoid code duplication). You must have **at least 3 functions** (in addition to `main()`). It is fine to have more.
- Define a **struct** to represent your items with **at least 4 member variables**.
- Use **a dynamic allocated 2D array** to store the showcase. Free the memory when done.
- Print the showcase, the total value, and the user's bank account balance at each step.
- Check all user inputs and let the user try again if the input is invalid. You can assume that the user will enter the data type requested, but you will need to check whether it is in the valid range.
- Employ good programming style and follow the course style guidelines.
- **See part 2 for additional specific requirements on the implementation (applies after you do the design document)**

Have fun and be creative!

Your first step is to **write a Design Document**. You can start on this right away. It does not require any programming; instead, it is your chance to plan out your program. Consult the Design Guidelines:
https://canvas.oregonstate.edu/courses/1770357/pages/design-document-guidelines

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 **your choice of items to include in the showcase, the list of attributes you will track, and any other details.** For the assumptions, consider: what is the min/max number of rows and columns you will permit? Etc.
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. Break the program into **individual pieces** that each perform one part of the task and will correspond to a function in C++.
   - **You should have a mini-flowchart for each function** (can be on the same diagram). See this example design document with functions: https://canvas.oregonstate.edu/courses/1770357/files/79399713

   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 for user input**. If you have more test cases, that's great! Each test case must have the **setting, user input, and expected result**.

Submit your Design Document on **Canvas**:
https://canvas.oregonstate.edu/courses/1770357/assignments/7847914

Part 2. **(80 pts)** Implement Your Showcase

Implement your Showcase 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_showcase.cpp`
- (5 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 items (not from the stack, and not a 1D array).
- (5 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.
- (5 pts) Initialize the member variables for each item in your 2D array with randomly chosen values. (You probably want to make this a function.) 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 strings you have, minus 1, and use this random number to index into your list and assign the new item's member to it. *It is fine if more than one item gets the same value from this list.*
  - Random Boolean: `(rand()%2 == 0)`
- Implement a function for each of the following. Each function should take your 2D array and any dimensions (rows, cols) needed as arguments, then prompt the user for any additional inputs.
  1. (10 pts) Allow the user to sell an item (clear its values, but do not delete the memory. Increase the funds in the user's bank account appropriately.
  2. (10 pts) Allow the user to buy an item (if they have sufficient funds and there is an empty location to store it). Update the user's bank account appropriately.
  3. (5 pts) Allow the user to display an individual item (printing its member values).
  4. (10 pts) Your custom function (see above).
- Add helper functions as needed to avoid code duplication and to break the code into readable small sections (modularize).
- (5 pts) Any time the contents of the showcase change, print the showcase and report the total number of items, total value of all items, and the user's bank account balance.
- (5 pts) Handle user input: You can assume that the user will enter a value that is the correct **type** of data that you request, but you must check that it is in the valid range, 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: https://canvas.oregonstate.edu/courses/1770357/files/79125719/
  No function should be more than 20 lines long (*excluding lines that consist only of comments and whitespace or a single curly brace*). If you find your functions are getting too long, break them into multiple smaller functions.
  - Do not put more than one statement on each line.
  - Remember the maximum line width of 80 characters.

Implementation tips:

- **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.
- **Consult lecture slides, labs 7 and 8, Piazza, office hours, etc.**, for help.
- Use good memory management. Delete memory off the heap when you are done using it.
  - Use **valgrind** to check for memory leaks, and then fix them.
  - Use **gdb** to track down any errors (or step through your code).
- Use **indentation** (2-3 spaces) and **good comments** to keep track of the flow of execution in your program. Use vim's auto-indent capability (also, =G in command mode will indent from the current point to the end of the file).

Implementation don'ts:

- No use of global variables (-5), goto (-5), vectors (-10), or recursion (-10).
- No use of classes (-10), only structs.

Creativity time:

- Feel free to improve the visualization of your showcase, e.g., using color or borders.
- You can add more options to the menu - what else might a collector want to do with their items? (See Extra Credit section)

Part 3. **(10 pts)** README instructions

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

- Your name, ONID, class (CS161), 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 **compile and run** your program. Then, 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.

You can use vim or any text editor to write your README.txt. Here is an example README.txt file for a hypothetical assignment (not this one):

https://canvas.oregonstate.edu/courses/1770357/files/79776395

Your README.txt should show your menu and explain each item in the Instructions section.

Part 4. **(10 pts)** Analyze Your Work (in a Word/text file, section "Part 5")

(A) (4 pts) Try each of your 8 test cases and for each one, report the outcome of the test **(copy the table from your design document and add a column that shows "actual result")**. If the expected and actual result for a test case not the same, 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.

(B) (2 pts) How much time did this program take you to finish? How does it compare to the time you estimated in your design document?

(C) (2 pts) Paste the output you obtain from running valgrind on your final program.

(D) (2 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 7). 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 5. Extra Credit

- **(up to 2 pts)** Add another menu option (beyond the 4 that are required) that lets the user do something else with their collection. Document this in your README.txt file.
- (**up to 2 pts**) Ask another person (friend, family member, roommate) to try out your showcase. 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 2 pts**) Describe one improvement you would recommend to make your program better (more robust, more entertaining, more attractive, anything).

Submit Your Assignment Electronically

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

(B) Submit your C++ program (**.cpp** file), your **README.txt**, and your **PDF file with written answers** before the assignment due date, using **Canvas**. https://canvas.oregonstate.edu/courses/1770357/assignments/7847916