CS 161 Assignment #4 – Restaurant

Design Document due: Monday, 5/11/2020, 8 p.m. (Canvas)
Program and Written Answers due: Monday, 5/18/2020, 8 p.m. (Canvas)
Demonstrate by: Wednesday, 5/27/2020, 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 loops to perform repeated activities
- Use static and dynamic arrays to store multiple related variables
- Use functions to modularize code to increase readability and reduce repeated code

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://canvas.oregonstate.edu/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 May 27:** 30-point deduction. (Also, we cannot guarantee a late grading slot.)
- **Demo Late Assignments:** Must be done by May 27, 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 Restaurant

In this assignment, you will create a restaurant with a cuisine and menu of your choice. Will it be a Mexican restaurant? Chinese? Indian? Martian? (It does not have to be a real cuisine!) You also decide what dishes will be on the menu (these can also be made up) and what their prices will be (your choice!).

Your program will welcome the user to your restaurant and ask how many diners are in the party. Next, the program will show the menu (with prices) and ask for each diner's order. Once everyone is done ordering, the program will print out a final bill with each dish ordered and its price, then the final total (with a tip, if you like!).

Here is example output from my program (user input is highlighted):

```
Welcome to the Olympus Mons Restaurant!
 How many diners in your party (1-20)? 3
Diner 0, please make your choice:
0) Syrtis Salmon: $15.95
1) Chryse Chicken: $13.50
2) Pavonis Pork (spicy): $14.75
3) Meridiani Mix (vegetarian): $12.95
1
Diner 1, please make your choice:
0) Syrtis Salmon: $15.95
1) Chryse Chicken: $13.50
2) Pavonis Pork (spicy): $14.75
3) Meridiani Mix (vegetarian): $12.95
3
Diner 2, please make your choice:
0) Syrtis Salmon: $15.95
1) Chryse Chicken: $13.50
2) Pavonis Pork (spicy): $14.75
3) Meridiani Mix (vegetarian): $12.95
2

I hope your dinner was delicious!

The bill for your delightful meal:
-----------------------------------------
Diner 0) Chryse Chicken: $13.50
Diner 1) Meridiani Mix (vegetarian): $12.95
Diner 2) Pavonis Pork (spicy): $14.75
-----------------------------------------
Total amount: $41.20
-----------------------------------------
```

You can decorate your menu (make a border using _ | - etc.), use color codes in dish names, and so on! In strings, "\e[34m" changes the text to blue and "\e[0m changes it back to normal; more options here: https://misc.flogisoft.com/bash/tip_colors_and_formatting

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.
- Your menu should have at least 4 options to choose from.
- Use **static and dynamic memory allocation** to create and manage data arrays. Arrays with a fixed size that is known in advance (like the names of dishes on the menu, and the prices of the dishes) should use static allocation, while arrays with unknown size (like the dishes ordered by the diners) should be dynamically allocated.
- After all orders are taken, **calculate the total bill** for the group.
- Check all user inputs and let the user try again if the input is invalid. You can assume that the user will enter an integer, 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 restaurant theme, number of items on the menu, whether you will include a tip, and any other details.** For the assumptions, consider: what is the valid range of party sizes you will permit? Can a diner change his/her order? Can more than one diner order the same dish? If so, will you group the items together on the bill (e.g., "2 hamburgers ($6.95 each): $13.90") or will you list them separately? 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 6 test cases (3 for number of diners, 3 for menu selections)**. If you have more test cases, that's great! If you add more user interaction, include test cases for each one. 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/7847910

Part 2. **(85 pts)** Implement Your Restaurant

In this part, you will implement your restaurant 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. Full grading details are provided in the assignment rubric (on Canvas).

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

- Name your file `assign4_restaurant.cpp`.
- (5 pts) Choose appropriate data types for the information you need to store. **Just as in Assignment 1, include a justification and min/max for the chosen data type in a comment above the variable's declaration.**
- (10 pts) Allocate **static 1-D arrays** to store (1) the names of the dishes on your menu and (2) the price of each dish. (Use a **constant integer** (but not a global variable) for the number of dishes and use it to specify the size of each array.)
- (10 pts) Allocate a **dynamic 1-D array** to store the name of the dish each customer ordered. Since you do not know in advance how many guests will be in the group, dynamic memory allocation is a good solution.
- (5 pts) **Initialize** all array values before they are used.

- (5 pts) Free (delete) dynamically allocated memory when you are done using it. No memory leaks.
- (10 pts) Calculate the total bill correctly (use a loop).
- (5 pts) Output all prices with 2 decimal places ($3.90 instead of $3.9). See tips below.
- (10 pts) Use **loops** correctly to iterate over arrays.
- (10 pts) Use **functions** correctly to reduce code duplication and length.
  - Each function must have a comment header (see style guide).
  - Each function must use appropriate parameter and return data types.
  - There must be at least 3 functions (in addition to `main()`)
  - No function (including `main()`) should be more than 20 lines long, excluding comments and blank lines. If you find your functions are getting too long, break them into multiple smaller functions.
- (5 pts) Check **every** user input for validity. If a user enters an invalid input, tell them it was invalid and let them try again. Tip: use a do-while loop.
- (10 pts) Use good programming style: file header, comment blocks, **function headers**, 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://canvas.oregonstate.edu/courses/1770357/files/79125719/

Implementation tips:

- Break the problem down. You can start top-down by writing your `main()` method with "stubs" for each function call and get that working. A "stub" is a function that does nothing but return a default value. Once the steps are working (for loops, etc.), then fill in each stub, getting each one working before moving on to the next one.
  - Or you could start bottom-up by implementing one function at a time (based on your design) and then adding calls to your functions in `main()` when they are ready.
- 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.
- To control the number of decimal places when printing a floating point number, specify that you want "fixed-place" numbers with a particular precision. **This does not change the value, just how it is printed.** Example with 3 decimal places:

```
float g = 2.37653439;
cout << fixed << setprecision(3);   /* need only do this once */
cout << g << endl;
```

  This prints out `2.377`. You will need to `#include <iomanip>` .
  Note that this changes the behavior of `cout` for all subsequent floats as well (so you only need to do it once at the beginning of your program). If desired, you can reset the output stream to its original behavior with:

```
cout.setf(0, ios::fixed);
cout.precision(6);
```

- 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.
- 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).
- Remember the maximum line width of 80 characters.

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) (6 pts) Try each of your 6 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 6 test cases in your Design Document, make them up now to allow you to get full credit here.

(C) (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?

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

(E) (2 pts) Managing dynamically allocated memory takes more effort on the programmer's part because memory must be explicitly freed. Why not just use static memory allocation all the time? (When does dynamic allocation help us?)

(F) (2 pts) What was the hardest part of this assignment? Was it what you expected?

Part 4. Extra Credit (in your separate file)

- (**up to 1 pts**) After printing the bill, ask if the diners want to split the bill. If so, divide the total by the number of diners and report how much each diner should pay.
- (**up to 2 pts**) Add a loop so that your restaurant can handle more than one party. After taking orders and printing the bill for one party, ask if there is another party to dine, and keep going until there are no more parties. **Ensure that you free and re-allocate the dynamic array(s) for each new party so that there are no memory leaks.** Static arrays (for the menu and prices) do not have to be re-allocated.
- (**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) 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://canvas.oregonstate.edu/courses/1770357/assignments/7847913
(C) Remember to sign up with a TA to demo your assignment.
The last day to demo this assignment is 5/27/2020.