CS 162 LAB #6 – Classes with Dynamic Memory & GDB Exercises

In order to get credit for the lab, you need to be checked off by the end of lab. You can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the lab before the next lab and get checked off with your instructor or TAs during their office hours. For extenuating circumstances, contact your lab TAs and the instructor.

This lab is worth 10 points total. Here's the breakdown:

- 6 points: Implement and test big three functions of one class (Menu)
- 4 points: debug programs using GDB and/or valgrind

This first part of the lab will continue to help you with handling dynamic memory in Assignment 3. Write the big three and test them to make sure they are working.

(6 pts) Part 1: Add copy constructor, assignment operator overload, and destructor to the appropriate classes

Step 1: Implement a non-default constructor in Menu class that allocates dynamic memory:

```
Menu (int size) {
      this->num_coffee = size;
      this->coffee_arr = new Coffee [size];
}
```

Then, create two Menu objects m1, m2 that will call the non-default constructor that you implemented above, i.e., in your main(), add the following two lines:

```
Menu m1(10);
Menu m2(5);
```

Compile and run your program with valgrind, and you will notice that your program has memory leaks due to the objects that you created above. Now implement the destructor to get rid of the memory leaks.

Function to implement:

```
~Menu(); //destructor
```

Step 2: Add the following line into your main():

```
m1 = m2;
```

Compile and run your program with valgrind. What did you notice?

The statement above assigns m2 to m1. However, we need deep copy instead of shallow copy since m1 and m2 contains dynamic memory. Now define and implement the overloaded assignment operator function to resolve the error.

Function to implement:

```
Menu& operator=(const Menu &); //assignment operator overload
```

Step 3: Add the following line into your main():

```
Menu m3 = m1;
```

Compile and run your program with valgrind. What did you notice?

The statement above creates a new object, m3, which is identical to m1, using a copy constructor. Similar to Step 2, we would need deep copy. Now define and implement copy constructor to resolve the error.

Function to implement:

```
Menu (const Menu &); //copy constructor
```

Make sure that you program does not have any memory leaks or memory errors before moving forward.

(4 pts in total) Part 3: GDB Debugging Exercises

Review the GDB tutorial from lab 5 and use GDB and/or valgrind to debug the following programs:

Step 0: Download and unzip the start code for this part:

wget https://classes.engr.oregonstate.edu/eecs/spring2023/cs162-010/labs/lab6.zip

unzip lab6.zip

(2 pts) Step 1: Use GDB to locate the segmentation fault

Use GDB to debug the programs `ex1.h, ex1.cpp, main.cpp`. The program tries to read information from the text file `file.txt` into the `Garage g` object. However, something in the code is broken, and the program crashes with a segmentation fault.

Answer the following questions in a text file called `ex1.txt`:

1. Run the program with GDB and/or valgrind, which line in which file generates the segmentation fault?
2. Use GDB commands that you learned from the previous lab to debug the program. What is the cause of this segmentation fault? What GDB commands did you use to find it? (Hint: all gdb commands you typed will be saved in `.gdb_history` file)
3. How did you fix the segmentation error?
4. In GDB, what command did you use to verify that the `Garage g` object stores all information from the `file.txt`? Is there another error that needs to be fixed?

(2 pts) Step 2: Use GDB to resolve the logic error

Use GDB to debug the program `ex2.cpp`. This program generates a small array of random integers and tries to sort it using insertion sort. However, something in the program is broken, and the array is not correctly sorted.

Answer the following questions in a text file called `ex2.txt`:

1. Use GDB commands that you learned from the previous lab to debug the program (i.e., run the program line by line). What is the cause of this logic error? What GDB commands did you use to find it? (Hint: all gdb commands you typed will be saved in `.gdb_history` file)
2. How did you fix the logic error?

Remember, you will not receive lab credit if you do not get checked off before leaving each lab. Once you have a zero on a lab, then it cannot be changed because we have no way of knowing if you were there or not.

Show your completed work and answers to the instructor or TAs for credit. You will not get points if you do not get checked off!

Submit your work to TEACH for our records **(Note: you will not get points if you don't get checked off with your instructor or a TA!!!)**

1. Create a **zip file** that contains all files you've created in this lab:
2. Transfer the tar file from the ENGR server to your local laptop.
3. Go to TEACH.
4. In the menu on the right side, go to **Class Tools → Submit Assignment**.
5. Select **CS162 Lab6** from the list of assignments and click "**SUBMIT NOW**"
6. Select your files and click the Submit button.