CSCI-1200 Data Structures — Spring 2022
Lab 4 — Memory Diagrams and Memory Debugging

Checkpoint 1 (Diagramming Dynamic Memory)
will be available at the start of Wednesday's lab.

Checkpoints 2 and 3 focus on using a memory debugger. It is highly recommended that you thoroughly read the instructions for Checkpoint 2 and Checkpoint 3 before starting.

Memory debuggers will be a useful tool in many of the assignments this semester, and in C++ development if you work with the language outside of this course. While a traditional debugger lets you step through your code and examine variables, a memory debugger instead reports memory-related errors during execution and can help find memory leaks after your program has terminated. The next time you see a "segmentation fault", or it works on your machine but not on Submitty, try running a memory debugger!

Please download the following 4 files needed for this lab:
http://www.cs.rpi.edu/academics/courses/spring22/csci1200/labs/04_memory_debugging/buggy_lab4.cpp
http://www.cs.rpi.edu/academics/courses/spring22/csci1200/labs/04_memory_debugging/first.txt
http://www.cs.rpi.edu/academics/courses/spring22/csci1200/labs/04_memory_debugging/middle.txt
http://www.cs.rpi.edu/academics/courses/spring22/csci1200/labs/04_memory_debugging/last.txt

Checkpoint 2

estimate: 20-40 minutes

For Checkpoint 2 of this lab, we will revisit the final checkpoint of the first lab of this course; only this time, we will heavily rely on dynamic memory to find the average and smallest number for a set of data from an input file. You will use a memory debugging tool such as DrMemory or Valgrind to fix memory errors and leaks in `buggy_lab4.cpp`. Make sure to download the provided .txt files as well.

First take a look at the code inside the *identifyMeanAndMin()* function. You will notice that the syntax used throughout the program may be a little different than what you're used to. Try and familiarize yourself with this syntax before you start working on the assignment.

What does the code `*(intArray + *numElements) = readInt;` do? What's going on inside the for loop? If you're stuck on this, ask a mentor or TA for help as soon as possible and refer back to your lecture notes on pointers and arrays.

Once you've done this, compile the program using:

```
g++ buggy_lab4.cpp -o buggy_lab4.out -g -Wall
```

Try running the program normally using: `./buggy_lab4.out`

You will notice that a segmentation fault occurs. Now run this program using either Valgrind or DrMemory. If running using Valgrind, remember to use `--leak-check=yes` or `--leak-check=full` .

Your memory debugger should give you more context as to why a segmentation fault is occurring, especially if you've compiled with -g. To complete this checkpoint, add or modify code in the areas marked Parts 1, 2,

and 3 to resolve all memory errors and leaks. As you are working on this, be sure to also think about the questions asked in Checkpoint 3.

It is highly recommended that you tackle one part at a time. For example, after adding a few lines of code to part 1, you will now receive different memory errors when you recompile and run the program using your memory debugger. Similarly, fixing all memory errors in part 2 will generate different memory errors that should be resolved in part 3.

In part 2 of `buggy_lab4.cpp`, the goal is to print out the contents of *intArray* in reverse order, while also calculating the sum of all elements in the array and keeping track of the smallest number encountered. Solutions that attempt to print the contents of the array in a different manner or end up with the wrong value for the smallest number found or sum won't be accepted.

Note that:

- You are only allowed to modify or add code when asked to. This would be between the comments that indicate parts 1, 2, and 3 inside `buggy_lab4.cpp`. Do not modify other parts of the code or create any helper functions.
- You are not allowed to declare new variables; the ones provided are more than enough. Hint: how do we create memory on the heap?
- You are not allowed to use additional data structures (like arrays or vectors).

You will receive no credit if you do not follow the above restrictions.

To receive credit for this checkpoint: Fix `buggy_lab4.cpp` so that it successfully prints out the average and smallest number for a given set of data and is free of all memory errors and leaks on your local machine. Submit `buggy_lab4.cpp` on Submitty and verify that there are no memory errors there as well and show a mentor or TA both results. Explain to a mentor TA what you added or modified in the program to resolve all memory errors.

Checkpoint 3 is on the following page.

Checkpoint 3

estimate: 15-25 minutes

As you work through Checkpoint 2, try and pay close attention the analysis given by DrMemory or Valgrind and think about the following:

1. How does the output from your memory debugger differ when you compile your program with the -g flag compared to when you leave it out?
2. How would you rewrite the for loop in part 2 to use the bracket [] operator instead of pointer syntax?
3. For DrMemory users, you would've encountered all of these errors in parts 1, 2, or 3 of Checkpoint 2:

   ```
   UNITIALIZED READ
   UNADDRESSABLE ACCESS
   INVALID HEAP ARGUMENT
   LEAK
   ```

4. For Valgrind users, the errors you will have seen are:

   ```
   Use of uninitialised value
   Invalid write
   Invalid read
   Conditional jump or move depends on uninitialised value(s)
   Mismatched free() / delete / delete []
   ? bytes in ? blocks are definitely lost in loss record ? of ?
   ```

5. What do you think each of these errors mean?

To receive credit for this checkpoint, discuss your answers to these questions with a TA or mentor.