Lab 4: GDB, Lists, & Snake

CSCI 41

Starting this lab, you may work in pairs. You can assume that pairs are allowed for any future lab unless I explicitly say otherwise. Each of you must *still submit separately* ***to the autograder once you're done, and you must*** *include a comment in your code* ***saying who you worked with. Lawton can help you if you're unsure about how to efficiently transfer your code between yourselves.***

Feel free to post on Discord to search for a partner. If you want a partner but can't find one, Lawton can help.

Objectives:

- Get comfortable debugging with GDB
- Practice linked list concepts
- Play the snake game after all your hard work

Part 1: Get the starter code

First, `cd` into the folder where you want to store your class files (e.g., `~/labs`). Copy the starter code there with the following line:

```
gimme lab04@csci41
```

This will make a lab04 folder in whatever directory you were in.

Part 2: Practice debugging with GDB

This part is optional and won't be turned in, but learning GDB will be a big help when you implement your list ADT.

Compile segfault.cpp with `make segfault`, and notice that it includes the `-g` compiler flag.

Load the `segfault` program inside of gdb with `gdb ./segfault`—then follow the process that I demonstrated in class to locate and debug the errors.

Part 3: Implement list.cpp

You will be implementing the List class from list.h—it uses the linked list data structure to build a list of `Point2D`s (which for all intents and purposes you can pretend are `int`s). The comments in list.cpp should guide you through the process.

You can test that your implementation is correct by running `make linkedListTests` and running the resulting `linkedListTests` program.

Part 4: Play snake with the fruits of your labor

This part is also optional, but fun!

After you get your List class working and passing all the tests, you can play the snake game that I implemented in snake.cpp. Compile with `make snake`. The game uses a List to keep track of where all the pieces of the snake's body are. My code isn't the cleanest, but I'd be happy to explain it to you in office hours if you'd like!

It's technically not a complete game, since the snake is allowed move on top of itself, etc.—can you fix that?

Part 5: Submit your code

When you're satisfied with your solutions, you can submit them to the autograder for grading. **Do not submit and assume you got 100%—you may be unpleasantly surprised. Always check your grade.**

From your solution directory, submit your code to the autograder using the following command on the terminal:

```
turnin lab04@csci41 list.cpp
```

The autograder will grade your code within a minute or so. If it's not working, please yell at Lawton to fix it. Run `view-grades` to look at your grades on the terminal, or go to the class website to view them there. You may resubmit as much as you want before the due date—just follow this same process after you've updated your programs.

Rubric

Rubric Item	**Points (18 pts total)**
The tests in linkedListTests.cpp run and pass	12 pts (2 for each test)
Your List class implementation does not leak memory (test yourself with `valgrind --leak-check=full ./linkedListTests`)	6 pts