CSCI 132 Data Structures—Lab #3

Introduction

In today's lab you will practice working with pointers and managing dynamic memory using `new` and `delete`. Part of this lab will requiring drawing diagrams with boxes and arrows.

The code for this lab is available in the directory: `~csci132/labs/lab3`

- Log onto your Logos account.
- Copy the code for lab3 into your labs folder by typing: `cp -r ~csci132/labs/lab3 ~/labs/`
- Change directories into your new lab3 directory by typing: `cd ~/labs/lab3/`

Goal

We have the following learning goals for today's labs.

- Getting used to pointers, dynamic memory, dynamic classes, and more advanced concepts such as pointers to pointers and 2D arrays.
- Sometimes you will see modified versions of your known data structures, where they have additional functionalities. You will see some such functionalities, and will be expected to use them.
- Writing client code with multiple classes and multiple data structures at the same time.

Diagram Memory

The `semester` program is a toy course enrollment system that holds data in some of the data structures you have studied, including ArrayBags, LinkedSets, and simple arrays. Compile and run the program using:

```
clang++ -Wall main.cpp Student.cpp Course.cpp -o semester
```

Examine the `main()` function within `main.cpp` and compare it to the output to orient yourself.

Answer the following questions on a separate sheet of paper (or on a tablet with a stylus)

- What local variables are declared in the `main()` function? List each variable's name and its data type. Only include variables that are in scope at the point in the code where `"READY!"` is printed, near the end. So don't include variables declared inside loops, for example. Hint: there are about half a dozen variables.
- Create a diagram showing the values of all of those local variables, again at the point in the code where `"READY!"` is printed, near the end. For simple values and static arrays, just draw a box (or a row of boxes) with the value. For pointers, your diagram should have an arrow leading to a box, or a clearly-indicated null pointer. For user-defined types (like class Student, class ArrayBag, class Course, etc.), show the inner private **contents** as well, which might in turn be pointers to yet further data. Careful: sometimes different variables will be pointing at the same objects, which you diagram must show as a single box with multiple arrows leading to it from different places. Hint: there will be many more than half a dozen boxes and arrows in your diagram.

Checkpoint 1:
Finish drawing the diagram and have it checked by the Professor or one of the TAs.

Working with Dynamic Memory

Now that we're familiar with the structure of the `Student` and `Course` classes and the arrangement of the dynamically allocated memory in our classes, we will update our `main` and add more functionality to our program.

To make the updates to our code more evident, we add the following three courses to `courses` in `main()`.

- `THEA 229 Virtual Realities`, enrollment: 16.
- `CSCI 328 Coding Music`, enrollment: 16.
- `VAST 105 Digital Arts Studio 1`, enrollment: 16.

In addition to adding these courses, update the number of courses each student is enrolled in so that they each have `5` courses (i.e., change the number in the corresponding calls to `enrollRandomCourses`).

Eliminate ArrayBag in favor of LinkedSet

The `courses` list in `main()` is declared as an `ArrayBag` containing pointers to `Course` objects. Often `ArrayBag` is cumbersome: its maximum size is quite small, and there is no way easy to iterate through the items in the `Bag`. Moreover, in this situation no course will appear more than once, so we don't need a `Bag`: a `Set` would do just as well, and the `Set` data type has some helpful methods that `Bag` doesn't have.

- If you look the `.h` files for `LinkedSet` and `ArrayBag`, you will notice that there are some public methods that we would normally not expect from a Set or Bag Abstract Data Type. In your `lab1.txt`, list down the methods that go beyond basic definitions of Sets and Bags.
- Change the code in `main.cpp` to use a `LinkedSet` of `Course` pointers rather than an `ArrayBag`. Change both the variable declaration, and modify the rest of the code so it compiles and runs as expected. Note: `ArrayBag` and `LinkedSet` have very similar, but not identical, interfaces. So you should only have to change a few lines of code. You can also simplify the code in `main()` converting from `ArrayBag` to a simple array, since `LinkedSet` has a `get()` method that makes it easy to iterate through.

Checkpoint 1.5:
You do not NEED to finish up to here to get in-lab checkpoints, but it is highly suggested that you target getting at least this much done in Day 1 of the Lab.

Making A New Friend

Judith makes friends easily. Add code at the end of `main()` to pick a new friend for Judith from among her classmates. Your code should:

- Pick a random course from Judith's schedule.
- Print the name of a random student from that course, formatted like: "Judith's new friend is ...", with a newline at the end.

Note: It might happen, randomly, that Judith picks herself. This is fine. It might even happen that Judith is the only student enrolled in the randomly-selected course.

Hint: You should not have to modify any existing code. Just add a few new lines of code to `main()`. Actually, this can be done with just one (long) line of new code.

Canceling a Course

Due to unforseen circumstances, there aren't enough classrooms to hold all six courses, so the College decides to cancel one class at random. Add code to the end of `main()` to do this. Your code should:

- Pick a random course to be deleted from among the list of courses.
- Remove it from the list of courses.
- Remove it from the schedule of all students enrolled in the course.
- Delete the course itself.
- Print the revised schedules for each of the students.

There should be no dangling references to the now-deleted `Course` object, and there should be orphaned objects that are unreachable from the variables in `main()`.

Note: There is currently no way to remove a course from a `Student`'s schedule. You will need to modify `Student` to add this functionality.

Favorite Course Nominations

Let's simulate an election where students vote on their favorite course. Add code to the end of `main()` to:

- For each student, pick a course at random from among the courses they are enrolled in. That becomes the student's vote for favorite course.
- Collect the votes in a single data structure. Hint: would `Set` or `Bag` make more sense here?
- For each course print a tally of how many votes that course received. Print these one per line, with the tally followed by the description (department, number, and title) of the course.

What to Turn In

Finish your answers to the checkpoints by the end of lab.

Submit your source code using the following command:

```
~csci132/bin/submit
```

Be sure that the program prologue for *each* of your source files contains your name, course, the date, and the purpose of the program or a description of the contents of the file (e.g., specification of the Student class).