CS 15 Lab 2: LinkedLists

Introduction

This week, we are working with a linked list of `Planet` objects. Like arrays, linked lists are linear data structures which store elements. Unlike arrays, the elements of a linked list are not stored together at a contiguous location in memory. Thus, you can't access the `nth` element of a linked list by directly indexing. Instead, a linked list is a collection of `Node` objects. Each `Node`, in addition to containing some data (in our case a `Planet` object), contains a pointer to another `Node`. Details will be discussed in lecture.

For the lab, however, you will be required to implement the functions to insert elements into the front and back of the list!

Getting Started

- You should have a `cs15m1` directory within your home directory. If you don't, create it now with `mkdir cs15m1`, and move inside it with `cd cs15m1`.
- Create a lab2 directory inside `cs15m1` and move into it.
- Copy the starter code: `cp /comp/15m1/files/lab02/* ./`
- If you're working on your local system, move the files there.
- Questions: What does the . mean? Do you know how to look at the files in the course files directory and your directory and make sure you got them all?
- Remember that a good policy when programming is to write a little, compile, test, repeat.

- In `CS 15`, we'll use make and an associated `Makefile` to compile and link programs. You don't need to know how to write `Makefile` just yet, just how to use them to build a program (type `make planet-driver`)
- Execute the program you built by running `./planet-driver`
- Once you are done with the lab you can submit the files with provide. The precise command is at the bottom of this handout. Remember that this time you need to submit a `README` file together with all of the source code.

Part 0: Understanding the Starter Code

Whenever you are given starter code, it's important that you take a few minutes to review it! Some introduction is given below.

Planet Class

In this lab, the elements we are working with will be `Planet`s! For our purposes, a `Planet` object contains a name (a `std::string`) and a distance from the sun in millions of miles (an `int`). The `Planet` class is completely written for you. Take a quick look at both `Planet.h` and `Planet.cpp` before continuing on.

LinkedList class

Regarding the `LinkedList` class, the starter code provides a complete `LinkedList` header file (`LinkedList.h`), and a partially completed driver file `LinkedList.cpp`. It would be well worth your time to review these files. Start with the header file. Notice the Node struct definition within the `private` section.

```
...
 private:
        struct Node {
            Planet info;
            Node   *next;
            void    print(std::ostream &out) {
                info.print(out);
            };
        };
...
```

Recall that contents of a `struct` object are public by defaut. However, because this `struct` is defined within the `private` section of the `LinkedList` class, its fields are only accessible by way of `LinkedList` objects. In other words, the member functions of the `LinkedList` can access directly the member fields of a `Node`.

Also, take note of the `Node *front` object in the private member section of the `LinkedList` class. This pointer is the starting point for your actual linked list.

As for the implementation file, take a quick look over it. Some functions (like `print` are defined for you - you would be wise to review and make use of these functions to help test and write your own code!

The portions of the lab you have to complete are marked with **TODO** (details are in **The Lab** section of this document).

Driver file

Your starter code also includes a partially completed driver file - `planet-driver.cpp`. Reviewing the function headers of the already completed functions should be enough for you to understand what they do - although if you'd like a review of using `std::cin` and/or `std::cout`, take a look at the code.

The portions of the lab you have to complete are marked with **TODO** - (details are in **The Lab** section of this document).

The Lab

In this assignment we will create two Linked Lists of `Planet` objects - one list will be created by pushing the `Planet`s to the front; the other will be created by pushing the `Planet`s to the back of the list.

Each `Planet` has a name, and a distance value, which represents the its approximate distance from the sun (in millions of miles). The main driver is mostly implemented for you. When you execute it, it first asks for the number of `Planet`s to be entered. Then, for that number, it asks for the user to type in the name of a `Planet` along with its associated distance from the sun.

As each `Planet` information is entered, the program creates a `Planet` object. All of the entered `Planet`s are stored in an array named `planet_array`.

Part I: Inserting at the Front

As you have seen in class, for simple linked lists it's easiest to insert a new element at the front of the list, because it's fast and doesn't require keeping track of the back of the list. Your first task is to implement the `pushAtFront()` function in `LinkedList.cpp`, and then to update `planet-driver.cpp` such that the Planet objects are pushed to the front list.

Note! you do **not** need to create the `LinkedList` class from scratch: use the one that exists in the starter code. If you have not already, please read the the **Part 0** section of this document before continuing.

When you're done implementing `pushAtFront()` and modifying the driver, the output of your program should look something like this:

```
Please enter the number of planets: 3
Enter: planet_name planet_distance
mars 4

Enter: planet_name planet_distance
earth 3

Enter: planet_name planet_distance
mercury 1

Part 1: PushAtFront list of planets:

LinkedList of size 3
--------------------
mercury 1
earth 3
mars 4
```

Part 2: Inserting at the Back

Let's insert planets directly at the back of the list. Open the file `LinkedList.cpp` and look for function `void pushAtBack(Planet)`. Fill in the implementation of this function. Then update the driver file to correctly push the Planets to the back of the list.

When you're done, the output of your program should look something like this:

```
Please enter the number of planets: 3
Enter: planet_name planet_distance
mercury 1

Enter: planet_name planet_distance
earth 3

Enter: planet_name planet_distance
mars 4

Part 1: pushAtFront list of planets:

LinkedList of size 3
--------------------
mars 4
earth 3
mercury 1


Part 2: pushAtBack list of planets:

LinkedList of size 3
--------------------
mercury 1
earth 3
mars 4
```

Part 3: Valgrind / Destructor

At this point, you should run `valgrind` on your code! You might be surprised to find some memory leaks / errors! To fix these, implement the `destructor` of the `LinkedList`. When finished, make sure there are no leaks/errors with `valgrind`.

JFFE: Reversing the List

This is a great task to do to thoroughly test your knowledge of Linked Lists! Your job here is to implement the reverse function of the `LinkedList`.

For this assignment you can modify `LinkedList.h` to add additional private member functions. However, you should keep the list as singly-linked, with no tail pointer. Your changes must not change the interface or break any of the other code in the lab.

You should understand all the code in this lab. Read the code that you are given. Do you have questions? Ask! You should be able to do any of this, for example, on an exam.

Part 4: Submitting Your Lab

You need to submit all code files (even those that you did not modify). In addition, from this assignment onwards you also have to provide a README file (reference the style guide for what should be included in your README, lab README's can be brief). Please submit your code as discussed in the class.

A Note on Lab Grading

Labs are for hands-on practice in a supervised setting so you can develop your skill. They are designed to be low-pressure and fun. Come to lab, do your best, submit your work at the end of the period (your best effort given the time), and you will get a high score. If for some reason you miss your lab session you can go to another one (provided there is space, see the course syllabus). If you cannot attend any session for a whole week do it in your own so you will gain the skills. See the syllabus for course policies on labs.