CSE30: Lab #8 – Linked list

Overview

Although arrays are used extensively in this class, they suffer from a major problem in that you, as the programmer, need to know in advance the number of elements required by your program. There are many situations, where this information is not available (e.g., reading lines from a file, where the number of lines is not given to you in advance) and you have to dynamically increase the size of your arrays. Although there are different ways of achieving this dynamical increase, the most popular is to implement a **Linked List**. A Linked List is simply a set of nodes, where each node is composed of a **value** (the value of the element) and a **pointer** that points to the next node. Using the pointer methodology allows the programmer to create new nodes by using the **new** keyword. In this lab, we will implement a Linked List as a class (the class declaration (.h file) is given on CatCourses) and make sure it works (using two main files <**Exercise1.cpp** and **Exercise2.cpp**> provided on CatCourses).

Getting started

Create a new directory in your main development directory (probably on Desktop/CSE30) called Lab_8. Try to use the terminal on your own, without getting help from the TA to setup the new directories (try to learn/remember the terminal commands).

The g++ syntax to compile classes is slightly different than for a single program comprised of a main (and potential functions):

```
g++ class1.h class1.cpp class2.h class2.cpp mainSource.cpp -o executable
```

where:

- g++ is the compiler (installed on your Linux system) of C++ source files,
- ***mainSource.cpp*** is the source file for your main program (main function),
- ***class1.h*** is the class declaration for your 1st class,
- ***class1.cpp*** is your 1st class definition,
- ***class2.h*** is the class declaration for your 2nd class,
- ***class2.cpp*** is your 2nd class definition (and so on...),
- -o tells the compiler that you want to give the executable its own name, and
- ***executable*** is the name you want to give your program.

As an example, if we have a main source file called ***Exercise1.cpp***, the class declaration called ***LinkedList.h***, the class definition called ***LinkedList.cpp***, and want to create an executable called ***aTestProgram***, you would type:

```
g++ LinkedList.h LinkedList.cpp Exercise1.cpp -o aTestProgram
```

Assuming that your program compiled successfully (i.e. no errors were found), you can run your program as you normally would by typing "***./aTestProgram***" in the terminal/console.

Good coding practices (worth 2 points!)

Writing code that is understandable by humans is as important as being correct for compilers. Writing good code will help you complete the code, debug it and … get good grades. It is very important to learn as soon as possible, because bad habits are hard to get rid of and good habits become effortless. Someone (guess who) reads your code will be in a better mood if it is easy to understand … leading to better grades! This lab will include 2 points (10% for code quality):

- Explanations with comments
- Meaningful names
- Indenting of blocks { } and nesting …
- Proper use of spaces, parentheses, etc. to
- Visible, clear logic
- One / simple statements per line
- Anything that keeps your style consistent

(Exercise 1)

In this part of the lab, you will be implementing the basic functions for a Linked List, which are provided in the class declaration **LinkedList.h** (on CatCourses). In other words, you have to **create and implement the class implementation in a file called LinkedList.cpp**. The main file you have to use for this lab is also provided on CatCourses (**Exercise1.cpp**). ***DO NOT modify the class declaration (LinkedList.h) or main file (Exercise1.cpp)***. Looking at the class declaration, you will find that a **Node** is defined as a structure comprised of a value (***val***, of type **int**) and a pointer to the next node (***next***, of type **Node pointer**). You will also notice (under ***private***) that you will be keeping track of your Linked List using two Node pointers, ***first*** and ***last***. The ***first*** pointer should always point to the **first element** of your Linked List and the ***last*** pointer should always point to the **last element** of your Linked List. If the Linked List is empty, the *first* and *last* pointers should both point to **NULL**. If there is only one element in your Linked List, both ***first*** and ***last*** will point to that element.

In this part of the lab, you will need to implement the following functions for the **LinkedList** class:

- **Default Constructor**: initializes the first and last variables, the list is empty.
- **Destructor**: deletes the entire Linked List, if not already empty.
- **insertAtBack(int)**: inserts a new element (i.e., a new Node) at the **end** of the list.
- **removeFromBack()**: removes the **last** element of the list. The function should return true if an element was successfully removed and false otherwise.
- **print()**: prints the entire list. See the example below for the format when printing the list.

- **isEmpty()**: returns whether or not the list is empty (true if it is empty, false otherwise).
- **size()**: returns the size of the list. You will need to navigate through the entire list and count how many elements are in the list.
- **clear()**: same function as the Destructor, deletes the entire list if not already empty.

Note:

- a pointer does not point to anything unless you 1) use the reference operator (&) in front of a variable, 2) you dynamically allocate memory for it (using the **new** operator), or 3) you assign it (set its value to) to another pointer
- every time you want to create a node, you will have to use the **new** operator
- every time you allocate memory dynamically for a pointer using the **new** keyword, you have to make sure that you de-allocate the memory once you do not need the data anymore. This can be done using **delete** (in the ***destructor***, ***clear*** and ***remove*** functions).

Example:

```
The first list is empty!
The second list is empty!
The size of the first list is: 0
The size of the second list is: 0
Here is the first list: [1,2,3,4,5]
Here is the second list: []
Here is the first list: [1,2,3,4,5]
Here is the second list: [25]
Here is the first list: [1,2,3,4]
Here is the second list: []
Here is the first list: []
Here is the second list: [-5,0,5,10]
The size of the first list is: 0
The size of the second list is: 4
The first list is empty!
The second list is NOT empty...
Here is the second list: [-5,0,5,10]
Successfully removed an item from the list...
Here is the second list: [-5,0,5]
Successfully removed an item from the list...
Here is the second list: [-5,0]
Successfully removed an item from the list...
Here is the second list: [-5]
Successfully removed an item from the list...
Here is the second list: []
COULD NOT remove an item from the list!
Here is the second list: []
COULD NOT remove an item from the list!
```

```
The size of the first list is: 0
The size of the second list is: 0
The first list is empty!
The second list is empty!
```

(Exercise 2)

In this part of the lab, you will finish implementing your Linked List by implementing two more functions. You should use a new main file (**Exercise2.cpp**), which you can find on CatCourses . Again, ***DO NOT modify the class declaration (LinkedList.h) or main file (Exercise2.cpp)***.

The two new functions are as follows (use your ***LinkedList*.cpp** file from Exercise 1, and **DO NOT** create a new LinkedList.cpp file):

- **insertAtFront(int)**: inserts a new element (i.e., a new Node) at the **beginning** of the list.
- **removeFromFront()**: removes the **first** element of the list. The function should return true if an element was successfully removed and false otherwise.

Example:

```
The first list is empty!
The second list is empty!
The size of the first list is: 0
The size of the second list is: 0
Here is the first list: [5,4,3,2,1]
Here is the second list: []
Here is the first list: [5,4,3,2,1]
Here is the second list: [25]
Here is the first list: [4,3,2,1]
Here is the second list: []
Here is the first list: []
Here is the second list: [10,5,0,-5]
The size of the first list is: 0
The size of the second list is: 4
The first list is empty!
The second list is NOT empty...
Here is the second list: [10,5,0,-5]
Successfully removed an item from the list...
Here is the second list: [5,0,-5]
Successfully removed an item from the list...
Here is the second list: [0,-5]
Successfully removed an item from the list...
Here is the second list: [-5]
Successfully removed an item from the list...
Here is the second list: []
COULD NOT remove an item from the list!
Here is the second list: []
```

```
COULD NOT remove an item from the list!
The size of the first list is: 0
The size of the second list is: 0
The first list is empty!
The second list is empty!
```

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your **LinkedList.h**, **LinkedList.cpp**, **Exercise1.cpp**, **Exercise2.cpp** , and a list of Collaborators.
- Documentation (in a text file) of code you used from the internet. You may want to cut-and-paste the original code as well.