CSE221: Data Structures
Programming Assignment 1
Due on Sunday 3 October, 23:59

Antoine Vigneron

1 Introduction

This assignment is on *circularly linked lists*. Circularly linked lists are briefly presented in Lecture 6, and presented in much more details in Section 3.4.1 of the textbook. The goal of this assignment is to implement a more complete version of the circular linked list than what is presented in the textbook. In particular:

- We want the data structure to be *generic* in the sense that we will use *templates*, so that a circular linked list can store elements of any types in its nodes. For instance, we will be able to construct a circular linked list of integers and a circular linked list of strings as follows:

```
CircleList<int> l1;
CircleList<string> l2;
```

- We will add a few more member functions that are not given in the textbook, as well as overloaded operators.
- For some member functions, we will throw an exception when some illegal operations are performed, such as accessing the front element of an empty list.

2 Circularly linked lists

A circularly linked list is similar to a singly linked list. The main difference is that the last node, called the *back* of the list, now points to the first node, called the *front* of the list. (See Figure 1.)

Again we refer to Section 3.4.1 of the textbook for a detailed explanation. It is important to note that there is a typo at the bottom of page 129: The descriptions of the back() method and the front() method have been exchanged. The function back() returns the element pointed by the cursor, and the function front() returns the next one.

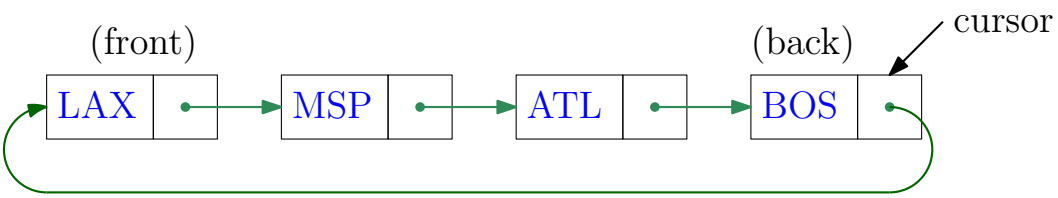

Figure 1: A circularly linked list of strings.

3 Implementation

Again, your circularly linked list class should be templated, allowing you to store any type of objects in its nodes. See Section 3.4.1 of the textbook, or the end of Lecture 5 slides, for more information on templates.

The whole code should be placed in a file `circlelist.h`. It is important to use this precise name so that we can automatically check your program correctness. You should not use any other file, so in particular, all the member functions definitions should be in `circlelist.h`. One reason for doing so is that there are technical issues with templated classes is we separate its declaration and definition into a `.h` and a `.cpp` file.

You should implement the data structure from scratch. For instance, you are not allowed to use the STL. So your `circlelist.h` file is only allows to include iostream, i.e. you can write `#include <iostream>`, but you cannot include any other file.

In order for us to be able to check automatically your code, the class definition should respect some rules. First, the general organization of your `circlelist.h` file should be as follows. Make sure to use the same class names, otherwise we will not be able to check your code automatically.

```
#ifndef CIRCLELIST
#define CIRCLELIST
#include <iostream>
using namespace std;

template <typename E>
class CircleList;

template <typename E>
class CNode {               // Circularly linked list node
// ...
}

template <typename E>
class CircleList {           // Circularly linked list
// ...
}

// ...

#endif
```

3.1 Basic member functions

You should first implement the member functions below, which are presented in the textbook.

```
CircleList();                    // default constructor
bool empty() const;              // returns true if list is empty
const E& back() const;           // element at the cursor
const E& front() const;          // element after the cursor
void advance();                  // moves cursor forward
void add(const E& e);            // adds a node after the cursor
void remove();                   // removes the node after cursor
```

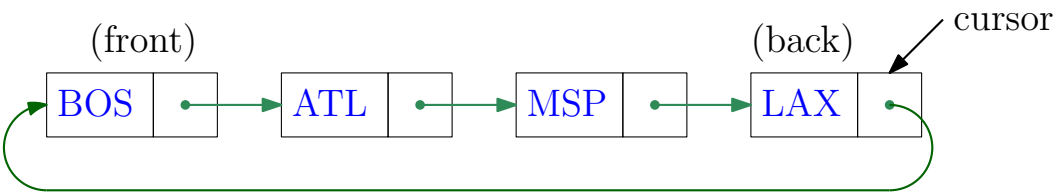

Figure 2: The list obtained after reversing the list from Figure 1.

3.2 Additional member functions

We ask you to implement a few more member functions, which we describe in more details.

- The size function

  ```
  int size() const;                    // number of elements in the list
  ```

  which returns the number of elements in the list. For instance, in Figure 1, the size is 4.

- The find function

  ```
  bool find(E e);                      // returns true if e is in the list,
                                       // and places cursor at e
  ```

  which returns true if element e is in the list, and false otherwise. In addition, if e is in the list, the cursor is moved to its first occurrence. For instance, in Figure 1, if we call `find("MSP")`, then the the function will return `true`, after moving the cursor two positions ahead.

- The reverse function

  ```
  void reverse();                      // reverse the list
  ```

  which reverses the order of the elements recorded in the list. (See Figure 2.) In particular, the back and the front elements are exchanged.

- The copy constructor

  ```
  CircleList(const CircleList<E> & L);  // copy constructor
  ```

  that creates a new linked list by copying entirely the list L node by node. In particular, if the list L is modified at a later point, this modification should not affect the newly created list.

3.3 Overloaded operators

You should also overload the following two operators:

- The << operator that prints the content of a list. There should be a single space between any two printed elements. So if list `l` is as pictured in Figure 1, then `cout << l;` should print `LAX MSP ATL BOS`.

- The equality operator ==. If lists L1 and L2 store the same elements in the same order, and with the same cursor position, then $L1 == L2$ is equal to `true`, and it is equal to `false` otherwise.

The code below shows an example of the copy constructor and the two overloaded operators:

```
#include "circlelist.h"
#include <string>

int main(){
  CircleList<int> l1;
  l1.add(1); l1.add(2); l1.add(3); l1.add(4);
  cout << l1 << endl;          // prints 4 3 2 1
  CircleList<int> l2=l1;       // calls the copy constructor
  cout << (l1==l2) << endl;    // l1==l2 is true, so it prints 1
  return EXIT_SUCCESS;
}
```

The output is:

```
4 3 2 1
1
```

4 Exceptions handling

The member functions `back()`, `front()` and `remove()` should throw an exception when they are called on an empty list.

You should not redefine your own exceptions as we did in class, but rather throw a standard `runtime_error`, so that our program checker will catch it. (More precisely, `std::runtim_error`.) You can choose the error message it throws.

5 Testing your program

You should already have, or soon have, access to this course server. There have been some technical issues that we are now resolving. In the meantime, you can first write your program under your favorite environment and test it later on our server.

A program `test1.cpp` is provided with this handout which makes simple tests of the basic member functions. To check your program, first upload the files `test1.cpp`, `Makefile` and your `circlelist.h` file to the server. (It should also work on most computers under linux.) Make sure the files are in the same directory, then type:

```
make
./test1
```

It should either print out an error message, or let you know that your code passed the test.

A program that tests your code much more extensively will be posted next week. In particular, we will check whether your program causes any memory leaks, so make sure that you deallocate all dynamically allocated memory, as explained in class.

6 Submission

You should submit your code by simply uploading your `circlelist.h` file to your home directory on the server.

7 Grading

Your program will be graded automatically. Make sure that the provided test programs compile and run without error on your code. If not, it will make it considerably more difficult to grade your program, and there will be a 10% penalty.

Late submissions up to one day after the deadline will be penalized 15%, and 30% two days after the deadline. After that, your assignment will not be graded (hence you get 0).

The grading program we will use in the end may differ from the ones that have been/will be provided.

8 Other rules

The general rules for the CSE department, given at the beginning of the semester (see course information folder), apply to this assignment. You may contact one of our two TAs for help:

- Seonghyeon Jue (`shjue@unist.ac.kr`)
- Hyeyun Yang (`gm1225@unist.ac.kr`)

9 Typos and bugs

If you find typos in this handout, or bugs in the code that is being supplied, please send email to the TAs or to me.