CSCI 2720

Assignment 2 - **Circular and Doubly Linked Lists**

Due: **July 8, 2018 11:59PM**

This assignment has two parts**:** Implementing a **Sorted Circular Linked List** and a **Sorted Doubly Linked List**. These two data structures should be implemented as two separate programs. Each program must accept 5 basic commands as described later in this document.

Similar to the first assignment, you will create a class called ItemType in this assignment. ItemType will act as a wrapper that abstracts the actual data that both Circular Linked List and Doubly Linked List hold and perform operations on. In this case you can assume the actual data type to be integer.

Important Points

1. Create two folders named, CircularLinkedList and DoublyLinkedList. Each folder must contain ItemType.h, ItemType.cpp and Main.cpp. CircularLinkedList.h and CircularLinkedList.cpp must be in CircularLinkedList folder and DoublyLinkedList.h and DoublyLinkedList.cpp must be in DoublyLinkedList folder.
2. You will use the same input.txt provided in the first assignment.
3. You should not allow duplicate values to be stored in your data structures
4. You have to implement a less number of commands in this assignment than in assignment-1. You can find the specific details below.
5. There is one bonus question in this assignment. It is associated with the doubly linked list program.
6. Be sure to properly document all code with comments, and add your name above functions that you implement if you are in a group.
7. You must exactly follow the submission instructions at the end of the document.

ItemType.h class should have the following function declarations:
Public functions:

ItemType()
    Post-Condition: ItemType object is created.
void print()
    Pre-Condition: ItemType object has been initialized.
    Post-Condition: value instance variable is printed to standard-out.

void initialize(int number)

Pre-Condition: number parameter is initialized.

Post-Condition: the value instance variable is set to number.

int getValue() const

Pre-Condition: ItemType object has been initialized.

Post-Condition: return the value instance variable.

Private data member:

int value

ItemType.cpp should provide implementations for the above functions.

CircularLinkedList.h should be composed of struct called NodeType and the following function declarations:

NodeType should contain the members:

ItemType data;
NodeType *next;

public functions:

CircularLinkedList()

Post-Condition: List is created.

~CircularLinkedList()

Pre-Condition: List is created.

Post-Condition: all nodes are freed.

void insertItem(ItemType &item)

Pre-Condition: List exists.

Post-Condition: item is inserted into the list in sorted order. If the item exists don't insert the new item and print "Item already exists".

void deleteItem(ItemType &item)

Pre-Condition: List exists and item initialized.

Post-Condition: the node that contains item is removed from the list.

int lengthIs() const

Pre-Condition: List is initialized.

Post-Condition: return length instance variable.

void print()

Pre-Condition: List exists.

Post-Condition: items are printed to standard-out based on the
implementation in ItemType.

Private members and functions:

```
int length
NodeType *head
NodeType *current
```

CircularLinkedList.cpp should implement every struct and function listed in CircularLinkedList.h.

Main.cpp. You can use the following code snippet in your Main.cpp to read input from a text file.

```
#include <fstream>
#include <iostream>
CircularLinkedList list;
ItemType item;
int input;
std::fstream fs;
fs.open(argv[1], std::fstream::in);
if(fs.is_open())
{
        fs >> input;
        while(!fs.eof())
        {
                item.initialize(input);
                list.insertItem(item);
                fs >> input;
        }
}
else
{
        std::cout << "Failed to open the input file" << std::endl;
        return 0;
}
```

You should only provide the implementations for the following 5 commands in the program for the Circular Linked List. You must use the exact same command characters and instruction phrases in your programs.

Example Output:

```
insert (i), delete (d), length (l), print (p), quit (q)

Enter a command: i
Number to insert: 20
1 3 9 10 19 20 37 45 63 84 100

Enter a command: d
Number to delete: 2
Item not in list!
1 3 9 10 19 20 37 45 63 84 100

Enter a command: d
Number to delete: 63
1 3 9 10 19 20 37 45 84 100

Enter a command: p
1 3 9 10 19 20 37 45 84 100

Enter a command: q
Quitting program…

./main input.txt
Commands - insert (i), delete (d), length (l), print (p), quit (q)

Enter a command: k
Command not recognized. Try again.

Enter a command: z
Command not recognized. Try again.

Enter a command: q
Quitting program…
```

DoublyLinkedList.h should have a struct called NodeType and the following function declarations:

NodeType structure should contain the members:

```
ItemType data;
NodeType* next;
NodeType* back;
```

public functions:

DoublyLinkedList()
Post-Condition: List is created.
~DoublyLinkedList()
Pre-Condition: List is created.
Post-Condition: all nodes are freed.
int lengthIs() const
Pre-Condition: List is initialized.
Post-Condition: return length instance variable.
void insertItem(ItemType &item)
Pre-Condition: List exists.
Post-Condition: item is inserted into the list in sorted order. If the item exists do not insert the new item and print "Item already exists".
void deleteItem(ItemType &item)
Pre-Condition: List exists and item initialized.
Post-Condition: the node that contains item is removed from the list.
void print()
Pre-Condition: List exists.
Post-Condition: items are printed to standard-out based on the implementation in ItemType.

private members and functions:
NodeType *head
NodeType *tail

DoublyLinkedList.cpp should implement the struct and the functions listed in DoublyLinkedList.h.

Bonus Question:
You should remove the ItemType.h and ItemType.cpp from the doubly linked list program. After that, you should convert the sorted doubly linked list to use C++ templates to be able to store any data type. Then, you should change your Main.cpp file to support storing int, float and std::string in your generic doubly linked list depending on an input that you take from the user at the very beginning of the program. User should be able to enter "i" for int, "f" for float or "s" for std::string. Please see the following example output.

```
./main input.txt
Enter list type (i - int, f - float, s - std:string): s
```

In the above example, the user has provided 's' as the input. In this case, you should initialize your generic doubly linked list to store std:string values. Also, you should read the values in the text file as std::string values and enter them to the list as std::string. In addition to that, insert, delete and print commands should also support std::string inputs. In the same way, if the user provided 'i' or 'f' your program should be able to work with int or float respectively. User is responsible for selecting an appropriate type for a given program run depending on the provided input file and the values that are expected store in the doubly linked list. See the following example output for the case of storing std::string values.

```
./main string-input.txt
Enter list type (i - int, f - float, s - std:string): s

insert (i), delete (d), length (l), print (p), quit (q)


Enter a command: p
Craig Dern Ford Goodman Macy


Enter a command: i
String to insert: Willis
Craig Dern Ford Goodman Macy Willis


Enter a command: i
String to insert: Aba
Aba Craig Dern Ford Goodman Macy Willis

Enter a command: d
String to delete: John
Item not in list!
Aba Craig Dern Ford Goodman Macy Willis

Enter a command: d
Number to delete: Ford
Aba Craig Dern Goodman Macy Willis

Enter a command: p
Aba Craig Dern Goodman Macy Willis

Enter a command: q
Quitting program…
```

You should only provide the implementations for the following 5 commands in the program for the Doubly Linked List. You must use the exact same command characters and instruction phrases in your programs.

Example Output:

```
insert (i), delete (d), length (l), print (p), quit (q)

Enter a command: i
Number to insert: 20
1 3 9 10 19 20 37 45 63 84 100

Enter a command: d
Number to delete: 2
Item not in list!
1 3 9 10 19 20 37 45 63 84 100

Enter a command: d
Number to delete: 63
1 3 9 10 19 20 37 45 84 100

Enter a command: p
1 3 9 10 19 20 37 45 84 100

Enter a command: q
Quitting program…
```

Grading Rubric:

Circular Linked List –	**35%**
Insert() –	10%
Delete() –	10%
Print() –	5%
Length() –	5%
~CircularLinkedList() –	5%
Doubly Linked List –	**35%**
Insert() –	10%
Delete() –	10%
Print() –	5%
Length() –	5%
~DoublyLinkedList() –	5%
Common Implementations–	**30%**
Following the specification –	5%
Main.cpp (Reading input and handling commands) –	10%
ItemType.h and ItemType.cpp –	10%
Readme, Comments and Makefile –	5%

Bonus Question- 20%

Compiling the Program (Same as assignment - 1):

A Makefile should compile your code into program called "main".

Your program should run with the following command syntax:

```
./main <input file name>
```

Commands to run and compile the code should be documented in the Readme File clearly.

Code that fails to compile or the code that compiles but fails to run will receive a grade of a zero.

Late Submission Policy:

Except in the cases of serious illness or emergencies, projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score
- If submitted 24–48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project.

Students unable to submit a project due to a serious illness or other emergency should contact the instructor as soon as possible before a project's deadline. Based upon the circumstances, the instructor will decide an appropriate course of action.

Submission Notes:

You must include your full name and university email address in your Readme.txt file. If your are doing the project in a group of two, list the full names and the email addresses of all two group members. If you are in a group you must also describe the contributions of each group member in the assignment.

Submit the following files on Nike:

First Folder named- CircularLinkedList containing

- ItemType.h
- ItemType.cpp
- CircularLinkedList.h
- CircularLinkedList.cpp
- input.txt
- Main.cpp
- Makefile

Second Folder named- DoublyLinkedList containing

- ItemType.h
- ItemType.cpp
- DoublyLinkedList.h
- DoublyLinkedList.cpp
- Main.cpp

- input.txt
- Makefile

One Readme file with all necessary descriptions.

Group Submissions

If you are working in a group of two, your submission folder must be names as below,

LastName1_LastName2_assignment2

When you are submitting the folder you should use the submit command as below,

```
submit LastName1_LastName2_assignment2 cs2720a
```

If you are working in a group, **you must submit the assignment only once** from the Nike account of one of the members in the group.

Individual Submissions

If you are working individually, your submission folder must be named as below,

LastName_**assignment2**

Submission command should be used as below

submit LastName_**assignment2** cs2720a

****For all the submissions, the folder naming and the readme is very important in order to be able to locate your assignment and grade it.**

When using the submit command, first set your current directory of the terminal to the immediate parent of the assignment folder.

For example if your assignment is in the following directory,

```
/documents/csci2720/assignments/LastName_assignment2
```

Before executing the submit command, execute:

```
cd /documents/csci2720/assignments/
```