# COMP3512 - Lab Exercise 7 (Nov 27- Dec 1, 2017)

This is an exercise that you need to do on a computer. You'll need to commit and push your code to your GitLab repo, and submit for automated marking via Slack.

For this exercise, you will need to implement a *DoublyLinkedList* in C++ using smart pointers.

## 1. Project Setup

1. Open Lab7.sln in Visual Studio 2017
2. Add Node.h file to your project. (refer to Lab 1 if you don't know how)
3. Add the following content in the header file.

|  |
| --- |
| #pragma once  #include <memory>  namespace lab7  {  template<typename T>  class Node  {  public:  Node(std::unique\_ptr<T> data);  Node(std::unique\_ptr<T> data, std::shared\_ptr<Node<T>> prev);  std::unique\_ptr<T> Data;  std::shared\_ptr<Node<T>> Next;  std::weak\_ptr<Node<T>> Previous;  };  // Implement constructors here    } |

1. Add DoublyLinkedList.h file to your project. Add the following content in the header file.

|  |
| --- |
| #pragma once  #include <memory>  namespace lab7  {  template<typename T>  class Node;  template<typename T>  class DoublyLinkedList  {  public:  DoublyLinkedList();  void Insert(std::unique\_ptr<T> data);  void Insert(std::unique\_ptr<T> data, unsigned int index);  bool Delete(const T& data);  bool Search(const T& data) const;  std::shared\_ptr<Node<T>> operator[](unsigned int index) const;  unsigned int GetLength() const;  private:  std::shared\_ptr<Node<T>> mRoot;    // Add any other private variables/methods here  };  // Add implementations of public methods here.  } |

1. Note that the solution will not build because some methods are not returning the right types. You will have to implement them to be able to build the solution.

### Expected Behavior of Node class

* Node class is nothing but a pure data class that holds data type of T.
* It has reference to the next node, and its previous node. Notice the different types of smart pointers used for Previous and Next.
* There are two constructors. Inside those constructors, initialize mData and Previous accordingly.

### Expected Behavior of DoublyLinkedList class

* Insert method has two overloads. If index is not provided, a new node is added at the end of the list. If index is provided, a new node is inserted at that position,

Ex.

1 -> 2 -> 3 -> 4

After Insert(5, 2)

1 -> 2 -> 5 -> 3 -> 4

* Delete method finds data of type T and deletes that node. Return true if data was found in the list and deleted. Return false otherwise.
* Search method finds data of type T in the list. Return true if found, else false.
* Subscript operator ([]) returns the node at that position. If the index is not in the valid range (ie> index >= length) return nullptr.

Ex

DoublyLinkedList list = 1 -> 2 -> 3 -> 4

list[2] returns the node that contains 3.

* GetLength returns the current length of the list.

## 

## 

## 2. Implement All Class Functions Introduced in Step 1

* You can test your code using something like the following code in main.cpp

|  |
| --- |
| #include "DoublyLinkedList.h"  #include "Node.h"  using namespace lab7;  int main()  {  DoublyLinkedList<int> list;  list.Insert(std::make\_unique<int>(1));  list.Insert(std::make\_unique<int>(2));  list.Insert(std::make\_unique<int>(3));  list.Insert(std::make\_unique<int>(4));  list.Delete(3);  list.Insert(std::make\_unique<int>(1), 2);  bool bExists = list.Search(0);  unsigned int size = list.GetLength();  std::shared\_ptr<Node<int>> node = list[0];  return 0;  } |

## 3. Commit, Push and Ask for a Build

You know the drill :)

# 