# COMP 322
## Winter Semester 2018

INSTRUCTOR: DR. CHAD ZAMMAR
chad.zammar@mcgill.ca

## Assignment 2: Exploring classes and dynamic memory management.

## Due date: 12 March 2018, 11:59 PM.

### Before you start:

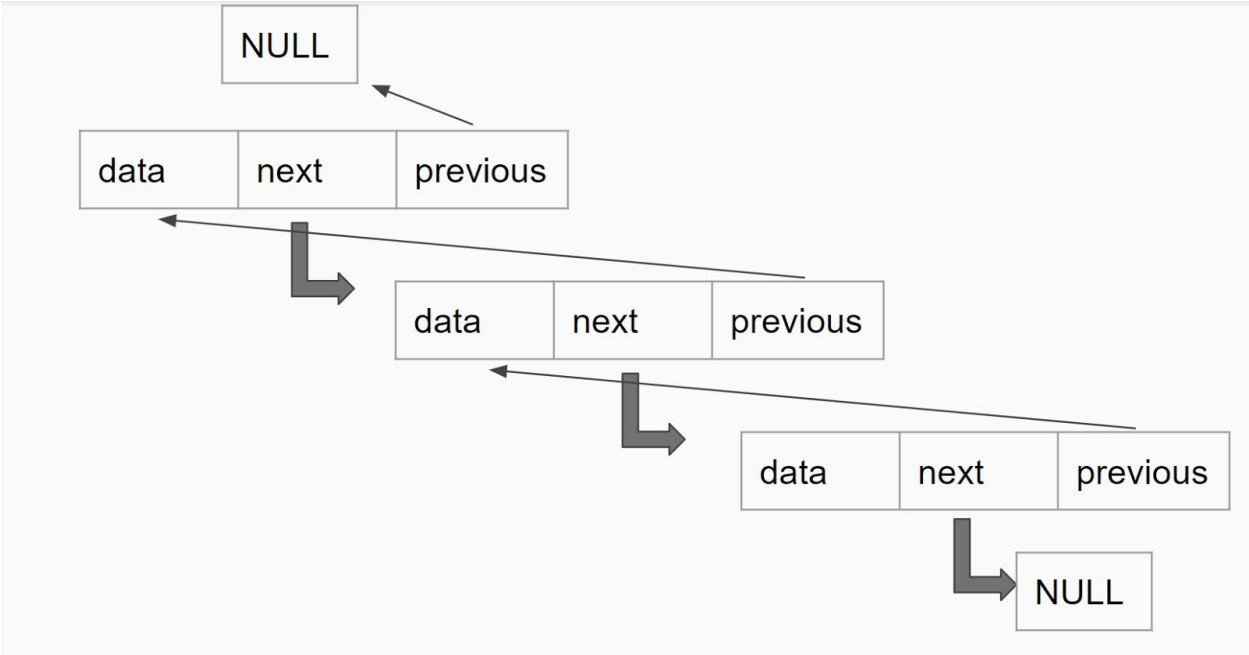
- Collaboration and research for similar problems on the internet are recommended. However, your submission should reflect individual work and personal effort.
- Some of the topics may not be covered in class due to our limited time. You are encouraged to find answers online. You can also reach to your instructor or TAs for guidance.
- Please do submit your assignment before the due date to avoid penalties or worse risking your assignment being rejected.
- Submit only .cpp and .h files. If some questions require you to develop and explain, you can embed your text in a C++ comment style.
- Make sure your code is clear and readable. Readability of your code as well as the quality of your comments will be graded.
- No submission by email. Submit your work to mycourses.
- Be happy when working on your assignment, because a happy software developer has more inspiration than a sad one :).

## Object Oriented Linked List Implementation

Arrays have their limitations. Their size should be known in advance during compile time. Inserting a new element in an array is a costly operation because it involves the shifting of elements. Resizing of an array is also very costly operation because it involves the creation of a new array and the copying of all the elements to the new structure. These limitations can be avoided by using a Linked List data structure.

In this assignment you will be designing and implementing a Linked List Class to manage a **doubly linked list**.

A linked list data structure is a collection of elements called nodes. Each element can hold data along with a pointer to the next element. Doubly linked list structures hold two pointers in each element instead of one. The first pointer points to the next element, and the second one points to the previous element in the list.



Let's start by designing and implementing a doubly linked list data structure step by step.

Note that we will need to define two classes. One that defines a node structure and one that manages the doubly linked list structure and behavior.

## Question 1 (5 pts)

Create a class called **Node** having the following data members and methods:

- **Int** data member to hold the **data** in each element
- A pointer to **Node** to hold the pointer to the **next** element
- A pointer to **Node** to hold the pointer to the **previous** element
- A default constructor **Node()** that initializes the data members appropriately
- A destructor **~Node()** that makes sure all dynamically allocated memory was appropriately deleted (if any)
- A personalized constructor that will create a node and assign its data and pointers to the given values passed as arguments **Node( int data, Node* next, Node* previous )**

You can make all members public for simplicity. However, if you want to make the data members private, make sure to provide methods permitting the interaction with the data like Setters and Getters for example.

## Question 2 (15 pts)

Create a class called **DLLStructure** (for Doubly Linked List Structure) having the following data members and methods:

- A pointer to the first element of the list
- A pointer to the last element of the list
- A default constructor that initializes an empty list
- A destructor that makes sure all elements in the list are being destroyed (calling **delete operator** for each element in the list)

- A personalized constructor that will create a list from an array passed as argument, meaning that the list will have the same number of elements as the array and each element will have its data assigned to the value from the corresponding array element (you may need to pass the array's size as an argument as well). Make sure to call the **new operator** to dynamically create a **new Node** element for each value in the array.

Make sure that all of your data is declared private and your methods are declared public.

The class declaration may look something like this:

```
class DLLStructure
{
public:
    DLLStructure();
    DLLStructure(int array[], int size);
    ~DLLStructure();

private:
    Node* first;
    Node* last;
};
```

## Question 3 (5 pts)

Implement a method in **DLLStructure** that will loop over all the elements in the list and print their values to the screen. The signature of the method will be

**void DLLStructure ::PrintDLL()**

To test your DLLStructure class, you may enter the following calls in your main function:

int array[5] = {11, 2, 7,  22, 4};

DLLStructure dll(array, 5); // note that 5 is the size of the array

dll.printDLL();

## Question 4 (10 pts)

Implement a method in **DLLStructure** that will insert a new element in the linked list after the first occurence of the value provided by the first argument and having the value provided in the second argument. The signature of the method will be

**void DLLStructure ::InsertAfter( int valueToInsertAfter, int valueToBeInserted)**

Test code:

```
int array[5] = {11, 2, 7,  22, 4};

DLLStructure dll(array, 5); // note that 5 is the size of the array

dll.InsertAfter(7, 13); // To insert 13 after the first occurence of 7

dll.printDLL(); // the output should be: 11, 2, 7, 13, 22, 4
```

## Question 5 (5 pts)

Can you use the method **InsertAfter** to implement another method **InsertBefore** that inserts a new element in the linked list before the first occurence of the value provided by the first argument and having the value provided in the second argument. The signature of the method will be

**void DLLStructure ::InsertBefore( int valueToInsertBefore, int valueToBeInserted)**

Remember that this method should call InsertAfter in its implementation.

Test code:

```
int array[5] = {11, 2, 7,  22, 4};

DLLStructure dll(array, 5); // note that 5 is the size of the array

dll.InsertAfter(7, 13); // To insert 13 after the first occurence of 7

dll.InsertBefore(7, 26); // To insert 26 before the first occurence of 7
```

dll.printDLL(); // the output should be: 11, 2, 26, 7, 13, 22, 4

## Question 6 (10 pts)

Implement a method in **DLLStructure** that will delete the first occurence of the value provided as argument

**void DLLStructure ::Delete(int value)**

Remember that you have to update the previous element's next pointer, as well as the next element's previous pointer to reflect the new list after deleting the appropriate value.

Test code:

int array[5] = {11, 2, 7,  22, 4};

DLLStructure dll(array, 5); // note that 5 is the size of the array

dll.InsertAfter(7, 13); // To insert 13 after the first occurence of 7

dll.InsertBefore(7, 26); // To insert 26 before the first occurence of 7

dll.printDLL(); // the output should be: 11, 2, 26, 7, 13, 22, 4

dll.Delete(22);

dll.printDLL(); // the output should be: 11, 2, 26, 7, 13, 4

## Question 7 (10 pts)

Implement a method in **DLLStructure** that will sort the elements in ascending order

**void DLLStructure ::Sort()**

Test code:

int array[5] = {11, 2, 7,  22, 4};

DLLStructure dll(array, 5); // note that 5 is the size of the array

dll.Sort();

dll.printDLL(); // the output should be:  2, 4, 7, 11, 22

## Question 8 (5 pts)

Implement a method in **DLLStructure** that will return TRUE if the list is empty and FALSE otherwise

**bool DLLStructure ::IsEmpty()**

## Question 9 (5 pts)

Implement the following getter methods  GetHead and GetTail in **DLLStructure** that will return the value of the first and last node respectively

**int DLLStructure ::GetHead()**

**int DLLStructure ::GetTail()**

## Question 10 (5 pts)

Implement GetSize method that will return the number of elements present in the list

**int DLLStructure ::GetSize()**

What would be the best implementation of GetSize if this method is requested very often? In other terms, how can we avoid looping over the elements of the list each time the GetSize method gets called?

## Question 11 (10 pts)

Implement GetMax and GetMin methods that will return the maximum and minimum data values present in the list

**int DLLStructure ::GetMax()**

**int DLLStructure ::GetMin()**

What would be the best implementation of GetMax and GetMin if these methods are requested very often? In other terms, how can we avoid looping over the elements of the list each time the GetMax or GetMin method gets called?

## Question 12 (15 pts)

What will happen if we rely on the default copy constructor that is provided by the compiler to copy the elements of a list into a new one?

If the default copy constructor  is not reliable (and you should explain why), can you provide a copy constructor for the class DLLStructure that given a reference to a DLLStructure object, would copy all its elements to a newly created DLLStructure.

**DLLStructure& DLLStructure::DLLStructure( DLLStructure& dlls)**

Test code:

```
int array[5] = {11, 2, 7,  22, 4};

DLLStructure dll1(array, 5); // note that 5 is the size of the array

DLLStructure dll2 (dll1);

dll2.printDLL(); // the output should be:  11, 2, 7,  22, 4
```