

# Programming Assignment 1

## CS 202 – Data Structures

Deadline: 14th February 2024, 11:55 PM

Lead TAs: Muhammad Muneeb Ahmed, Maira Kamal, Hashir Fida

### Plagiarism Policy:

1. Students must not share the actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students cannot copy code from the Internet.
4. Students must indicate any assistance they received.
5. All submissions are subject to automated plagiarism detection.
6. Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee

## Smart Pointers:

Smart pointers are class objects that behave like built-in pointers but also manage objects that you create with `new` so that you don't have to worry about when and whether to delete them - the smart pointers automatically delete the managed object for you at the appropriate time. They can prevent memory leaks and make code more readable and maintainable. In the industry, smart pointers are widely used in large-scale software projects to improve memory safety, exception safety, and code readability.

They are also used in many libraries and frameworks to manage resources and simplify resource allocation and deallocation.

To read an article explaining how to use smart pointers, click [here](#).

**In this assignment, you are not allowed to dynamically allocate memory. You need to make use of smart pointers to implement this assignment.**

The format for declaring a shared pointer in C++ is:

```
shared_ptr<Type> ptr_name(new Type(arguments));  
shared_ptr<Type> ptr_name = make_shared<Type>(arguments);
```

## Templates:

Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type. In C++ this can be achieved using template parameters. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow also types to pass to a function. These function templates can use these parameters as if they were any other regular type.

The format for declaring function templates with type parameters is:

```
template <class __identifier> function__declaration;  
template <typename __identifier> function__declaration;
```

For example, to create a template function that returns the greater one of two objects we could use:

```
template<class myType>  
myType GetMax(myType a, myType b)  
{  
    if (a > b)  
        return a;  
    else return b;  
}
```

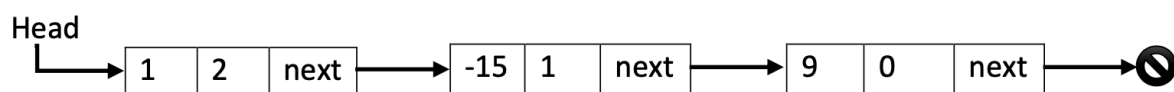
To learn more about templates, click [here](#).

## Task-1: Polynomial Calculator (50 Marks)

In this task, you will develop a Polynomial Calculator using C++ that will be used to perform addition, subtraction, and multiplication of two polynomials with one variable. In this part, you are expected to use the object-oriented programming fundamentals and linked list implementation. However, you are not allowed to use pre-built data structures of standard template library (STL).

You will be provided with a starter code containing class declarations and the main function. Your task is to complete the definition and implement the missing methods of the class. You may add new helper methods in the class, but you are not allowed to change/remove the headers of existing methods. The input/output format of the program should be the same as illustrated in the screenshots.

A polynomial expression can be stored in a Linked List by representing each term as a node of the linked list. Each node of the list contains the coefficient and exponent of each term, respectively. For example, the expression  $x^2 - 15x + 9$  can be stored in a Linked List as follows:



There are going to be two main classes to be implemented as described below:

1. LinkedList class
2. PolyCalculator class

## 1. LinkedList Class (5 marks)

In this part, you will be applying what you learned in class to implement different functionalities of a linked list efficiently. The basic layout of a linked list is given to you in the LinkedList.h file. The template ListItem in LinkedList.h represents a node in a linked list. The class LinkedList implements the linked list which contains pointers to head and tail and other function declarations. You are also given a file, test1.cpp for checking your solution. However, you are only allowed to make changes in the LinkedList.cpp file.

NOTE: When implementing functions, pay special attention to corner cases such as deleting from an empty list.

Member functions: Write implementation for the following methods as described here.

**a) LinkedList():**

- Simple default constructor.

**b) LinkedList(const LinkedList<T>& otherList):**

- Simple copy constructor, given pointer to otherList this constructor copies all elements from otherList to new list.

**c) void InsertAtHead(T item):**

- Inserts item at the start of the linked list.

**d) void InsertAtTail(T item):**

- Inserts item at the end of the linked list.

**e) void InsertAfter(T toInsert, T afterWhat):**

- Traverse the list to find afterWhat and insert the toInsert after it.

**f) ListItem \*getHead():**

- Returns the pointer to the head of the list.

**g) ListItem \*getTail():**

- Returns the pointer to the tail of the list.

**h) ListItem \*searchFor (T item):**

- Returns a pointer to the item if it is in the list, returns null otherwise.

**i) void deleteElement(T item):**

- Find the element item and delete it from the list.

**j) void deleteHead():**

- Delete the head of the list.

**k) void deleteTail():**

- Delete the tail of the list.

**l) int length():**

- Returns the number of nodes in the list.

## 2. PolyCalculator Class (45 marks)

You need to implement the following functionality as part of this class:

### a) void PolyCalculator::input()

This method should ask for two polynomial expressions to be provided by the user as illustrated in the below screenshot of the terminal, which invokes this method.

```
>input
Enter first Polynomial expression: 4x^2-2x^2+3x^1+2x^0
Enter second Polynomial expression: 3x^2-2x^1+1x^0
```

### b) void PolyCalculator::display()

This method should display the two polynomial expressions previously loaded to the calculator as illustrated below.

```
>display
Exp1: +2x^2+3x^1+2x^0
Exp2: +3x^2-2x^1+1x^0
```

### c) void PolyCalculator::add(PolyCalculator<int>& p1)

This method should add the two polynomial expressions previously loaded to the calculator (i.e., the values of attributes: list1 and list2) and print out the result as illustrated below. The result should be stored in the attribute list3.

```
>add
Exp1 + Exp2 = +5x^2+1x^1+3x^0
```

**d) void PolyCalculator::sub(PolyCalculator<int>& p1)**

This method should subtract the two polynomial expressions previously loaded to the calculator (i.e., the values of attributes: list1 and list2) and print out the result as illustrated below. The result should be stored in the attribute list3.

```
>sub
Exp1 - Exp2 = -1x^2+5x^1+1x^0
```

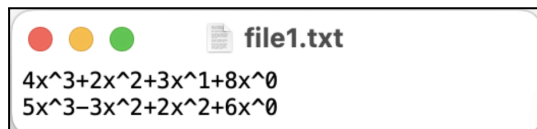
**e) void PolyCalculator::mul(PolyCalculator<int>& p1)**

This method should multiply the two polynomial expressions previously loaded to the calculator (i.e., the values of attributes: list1 and list2) and print out the result as illustrated below. The result should be stored in the attribute list3.

```
>mul
Exp1 * Exp2 = +6x^4+5x^3+2x^2-1x^1+2x^0
```

**f) void PolyCalculator::readData(string filename)**

This method should read and load two polynomial expressions from a given file. The existing LinkedLists/expressions must be cleared before the new data is stored into them. The expressions in the file might not necessarily be sorted or simplified. The following is an example of a valid input file.



A screenshot of a text editor window titled "file1.txt". The window contains two lines of text, each representing a polynomial expression. The first line is "4x^3+2x^2+3x^1+8x^0" and the second line is "5x^3-3x^2+2x^2+6x^0". The text is in a monospaced font.

Below is a sample output of the *read* command, that invokes the readData method:

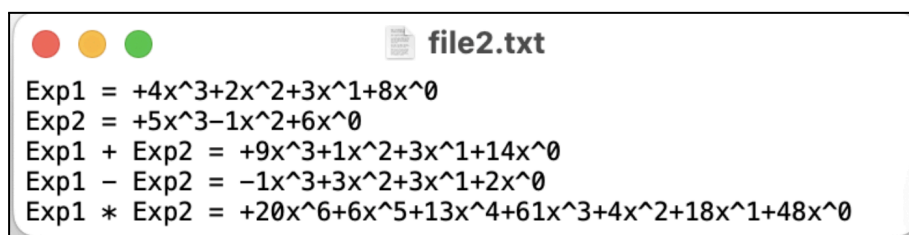


```
>read file1.txt
Exp1: +4x^3+2x^2+3x^1+8x^0
Exp2: +5x^3-1x^2+6x^0
```

**g) void PolyCalculator::writeData(string filename)**

This method should write the currently loaded polynomial expressions and the result of their addition, subtraction, and multiplication into a given file. The contents of the file must be overwritten if the file already exists.

Below is an example of a file generated by *the writeData* method that is invoked by the write command from User-Interface (UI). For simplicity and to remove redundancy, you may also create and invoke a helper method for this method.



```
Exp1 = +4x^3+2x^2+3x^1+8x^0
Exp2 = +5x^3-1x^2+6x^0
Exp1 + Exp2 = +9x^3+1x^2+3x^1+14x^0
Exp1 - Exp2 = -1x^3+3x^2+3x^1+2x^0
Exp1 * Exp2 = +20x^6+6x^5+13x^4+61x^3+4x^2+18x^1+48x^0
```

**h) bool PolyCalculator<T>::parse(string str, PolyCalculator<int>& p1)**

This method takes as an argument a polynomial expression stored in a string and a linked list object. This method should parse and store the expression in the given linked list. It should check if the given string is a valid polynomial expression or not. In case the expression is invalid, the method should return false. The method should identify, extract, and process each term of the polynomial expression from the given string and insert it into the given linked list. A valid polynomial term in the given string will always be of the following form:

**<coef>x^<exp>**

**coef:** The coefficient of a term of the polynomial expression. It can be a positive or negative integer number. The positive sign for the first term of a polynomial expression is optional. (see examples below).

**x:** The polynomial variable. It will always be x in lower-case.

**exp:** The exponent of a term of the polynomial expression. It is always a non-negative number without a sign.

**Following are some examples of valid polynomial expressions:**

- $12x^1 + 3x^0$
- $+2x^1 - 1x^2 + 5x^0$
- $-2x^2 + 2x^2 + 3x^1 - 4x^0$
- $2x^1$

**Following are some examples of invalid polynomial expressions:**

- $2x^{12}x^3$  // no operator between two terms
- $2x^$  // exponent is missing
- $2x^1 + 4^x0 +$  // the expression is not properly terminated.
- $3x^1++3x^0$  // multiple operators

The coefficient of a term in the expression can be a positive or negative integer number, whereas the exponent cannot be a negative value. Please also note that this method is not responsible for sorting or simplifying the polynomial expression. It is encouraged that you write a helper function that does this job.

For example, if an expression  $1x^2 - 15x^1 + 9x^0 + 2x^2$  is passed to the parse function along with a linked list object, the content of the Linked List should be structured as follows by the parse function.

The *parse()* method should return true if the expression is valid and false otherwise.

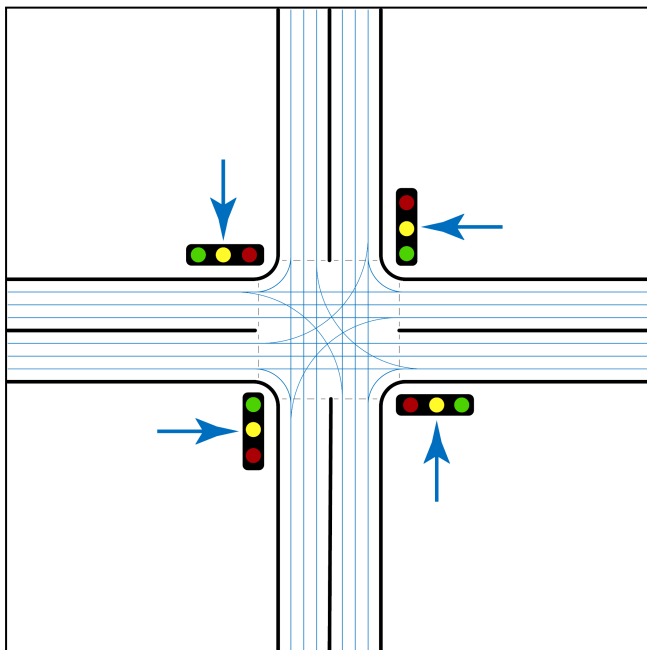
## Task-2: Traffic Signal Simulator (50 marks)

In this task, you are required to implement a traffic signal simulator as explained below:

There are four directions where the cars may be coming from at a junction. The junction has traffic signals for all incoming queues. At one time, one of the signals will be ON for a period  $t$  seconds. In parallel, traffic is coming into all queues almost randomly. So, order and time of arrival of individual cars is random but each car is noticed as a separate entity and its **arrival** and **departure** times are noted. You are required to implement the following variations of the simulator:

- A) Configure signals of each side for an equal amount of time and run the simulator.
- B) The signal should decide time for each side intelligently with an attempt to reduce the average waiting time in all queues (a decent attempt in this direction will help you earn the credit)

Please note that your simulator should run infinitely (as is the case in any real world scenario)



You will be using a queue data structure for the implementation of this part.

### **Queue: (5 marks)**

Member functions: Write implementation for the following methods as described here.

**a) Queue():**

- Simple base constructor.

**b) Queue(const Queue<T>& otherQueue):**

- Copy all elements of other Queues into the new queue.

**c) void enqueue(T item):**

- Add items to the end of the queue.

**d) T front():**

- Returns an element at the front of the queue without deleting it.

**e) T dequeue():**

- Returns and deletes elements at the front of the queue.

**f) int length():**

- Returns the count of the number of elements in the queue.

**g) bool isEmpty():**

- Return true if there is no element in the queue, false otherwise.

## Traffic Signal: (45 marks)

You need to implement the following classes and their member functions in this part:

### 1. Class Car()

This class will have a car's id, arrival, and departure time. This will also have a static integer variable to store the time for which the program is running (each time when a new instance of car is made or a destructor is called, the time increments). The declaration and initialization of this variable is given in the starter code.

(To read more about static variables in a class, click [here](#).)

You need to think of necessary member functions for this class yourself.

### 2. Class TrafficSignal()

This represents each side of the junction. There will be 4 sides: North, South, East, West.

This has the following member functions (to be implemented):

- a. TrafficSignal(string direction); //Constructor
- b. void addCar (Car car); // To enqueue the car in the queue
- c. void runSignal(int signalTime); //Running the signal for some specified time. This time is fixed for part A and has to be calculated in part B. Here you need to display the id, direction and departure time for each of the cars.

Hint: Everytime a car is *dequeued*, it is leaving the junction so needs to be *destroyed*.

### 3. Class Junction()

This represents the entire traffic signal system.

The following functions need to be implemented for this class:

- a. Junction(int signalTime) //Constructor
- b. void addCar(Car car) //To randomly assign a car to one of the TrafficSignals
- c. void runSimulation() //To infinitely run the simulation

Finally, here is a breakdown of how your simulation is going to work and some tasks to perform:

1. Before running the simulation, you need to enqueue 20 cars randomly in the 4 mentioned directions (TrafficSignals)
2. The simulation should run infinitely but all 4 traffic signals should have run once in a cycle before it starts again. After every cycle, you need to display the average wait time for cars during that cycle.

**Note:** Wherever necessary you need to make getters and setters. You may add variables and helper functions to classes as needed but **DO NOT** change the given declarations.

**IMPORTANT NOTE:** Write the time complexities with all the functions that you have implemented. Failure to do so will result in negative marking.

You have 2 cpp files which are exactly the same. However, in TrafficSimulatorB.cpp, you need to have signalTime variable for every TrafficSignal. The implementation for this is on you.

**(Hint: Make use of Average wait time and length of queue.)**

## Running and Testing Code:

To run the code, you have to use g++ compiler. You have 2 test files at your disposal:

1. Test1.cpp: Test for LinkedList.cpp.
2. Test2.cpp: Test for queue.cpp.

Write your implementations in the .cpp files and then run the test 1 and test 2 using the following commands:

Compile: `g++ test1.cpp -std=c++11`  
Run: `./a.out`

## Submission guidelines:

Zip the complete folder and use the following naming convention:

PA1\_<roll number>.zip

For example, if your roll number is 25100067 then your zip file name should:

PA1\_25100067.zip

All submissions must be uploaded on LMS before the deadline.

You are allowed 5 "free" late days during the semester (that can be applied to one or more assignments; the final assignment will be due tentatively on the final day of classes, i.e., before the dead week and cannot be turned in late. The last day to do any late submission is also the final day of classes, even if you have free late days remaining).

If you submit your work late for any assignment once your 5 "free" late days are used, the following penalty will be applied:

- 10% for work submitted up to 24 hours late
- 20% for work submitted up to 2 days late
- 30% for work submitted up to 3 days late
- 100% for work submitted after 3 days (i.e., you cannot submit assignments more than 3 days late after you have used your 5 free late days.