LAB SESSION 2
POLYNOMIAL LIST

1. OBJECTIVE

The objectives of Lab 2 are

(1) to introduce on the concepts of class interface and implementation in C++,
(2) to demonstrate how to use linked list for representing polynomial.

2. CLASS INTERFACE AND IMPLEMENTATION

For the sake of convenience, C++ allows (and suggests) developers to separate interface and implementation parts when developing a class. Listing 1 illustrates the separation. In this listing, the interface for class List is declared first. Note that, the parameters of its methods are declared by only the data type. For example, the method void addFirst(int) is about to receive an input of type int and returns nothing.

The implementation of all methods in the class List can be declared after that. Note that, the method should be prefixed by the class name and a double colon (::) and the parameter names should be declared. For example, the method addFirst is implemented as void List::addFirst(int newdata).

```
//just an entry in the list, a "struct++" in fact
class Node {
      public:
       int data;
       Node* next;
};

//interface part
class List {
     private:
            int count;
            Node* pHead;
     public:
            List();
            void addFirst(int);
            void display();
            ~List();
};

//implementation part
List::List() {pHead=NULL;}

void List::addFirst(int newdata) {
      Node* pTemp = new Node;
      pTemp->data = newdata;
      pTemp->next = pHead;
      pHead = pTemp;
      count++;
}

void List::display() {
      Node* pTemp = pHead;
      while (pTemp!=NULL)    {
            cout << pTemp->data;
            pTemp = pTemp->next;
      }
}

List::~List()  {
      Node* pTemp = pHead;
      while (pTemp!=NULL) {
            pTemp = pTemp->next;
            delete pHead;
            pHead = pTemp;
      }
}
```

Listing 1

3. USE LINKED LIST to REPRESENT POLYNOMIAL

As described in Tutorial 2, linked list can be used effectively to represent polynomials. For example, to create a list representing the polynomial of $5x^4 + x^2 + 1$, a piece of code can be developed as described in Listing 2.

```
void main() {
      IntList intList;
      intList.addFirst(5);
      intList.addFirst(0);
      intList.addFirst(2);
      intList.addFirst(0);
      intList.addFirst(1);
      intList.display();
}
```

Listing 2

As another example, in Listing 3 is the implementation of a method *addConstant*, which adds a constant to a polynomial.

```
void List::addConstant(int nConst) {
      Node* pTemp = pHead;
      while (pTemp->next!=NULL) pTemp = pTemp->next;;
      pTemp->data += nConst;
      return;
}
```

Listing 3

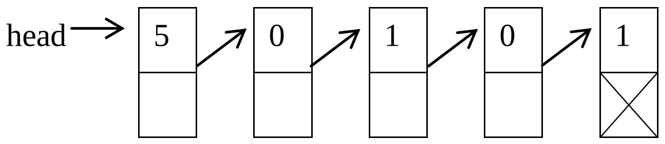
4. EXERCISES

In this work, you are provided seven files: *List.h*, *List.cpp*, *Poly.cpp*, *Stack.h*, *Stack.cpp*, *Queue.h*, and *Queue.cpp*. You can see that the .h file contains the interface part and *.cpp* the implementation part of the class List introduced above. File *Poly.cpp* contains the main program.

Consider the file *List.cpp* attached. Use this initial code to accomplish the following tasks.

Required Exercises

4.1. Develop the *main* function in the file *Poly.cpp* in order to build a linked list representing the following polynomial:



4.2. Implement the method *display* in file *List.cpp* and use it to display the lists built in Exercise 4.1.

4.3. Implement the incomplete methods of *addContant()* and *addPoly*. Write some pieces of code in the *main* function to test your implemented methods.

4.4. Develop method *printPoly* to display the contents of list as polynomial. For example, the list {3,5,0,8} will be displayed as 3x^3 + 5x^2 + 8.

4.5. Implement simple Stack and simple Queue using Linked List

a. Stack has methods: *Stack*, *push*, *pop*, *~Stack*

b. Queue has methods: *Queue*, *~Queue*, *enQueue*, *deQueue*

4.6. Develop method *reverseList* that reverses the order of elements on list using additional non-array variables and

a. one additional stack
b. one additional queue

4.7. Develop method *appendList* that receives another linked queue and appends the input queue to the end of the current queue. The input queue will be empty afterward. Write some pieces of code in the *main* function to test your implemented methods.

4.8. Develop the method *getIntersection* of class List that find intersection of two List and return new List (result). Write some pieces of code in main function to test your implemented method. Example:

List A: 10 20 30 40 50 60 70

List B: 10 30 50 70 90 110 130

Intersection of A and B: 10 30 50 70

4.9. Develop the method *getUnion* of class List that find intersection of two List and return new List (result). Write some pieces of code in main function to test your implemented method. Example:

List A: 10 20 30 40 50 60 70

List B: 10 30 50 70 90 110 130

Union of A and B: 10 20 30 40 50 60 70 90 110 130

Advanced Exercises

4.10. Develop the method *divisionPoly* of class *List* to implement the operation $f \setminus f_2$ as stated in Tut2.

For convenience, you may assume that f_2 is a factor of f, i.e. $f(x) = f_2(x)*g(x)$. Moreover, there would be no rounding required when performing the division among the coefficients (i.e the coefficients are always divisible in the division).

-- End --