Name: M. Anas Baig **Subject: Data Structure & Algorithms**

Enrollment No.: 01-134152-037 **Section: BS(CS)-4A**

BAHRIA UNIVERSITY ISLAMABAD, PAKISTAN

Assignment # 2

Data Structures and Algorithms

March, 14th 2017

SOLUTION:

C++ Concepts Used:

Following C++ concepts are used in solving the problem:

- Operator Overloading (used in Polynomials Addition and Multiplication).
- Recursion (used in evaluating exponent at base i.e. $\mathbf{y^x}$).
- Linked Lists. (to store Polynomial Equations).
- String and String Functions.
- File Handling. (to read and write data from files).

Functionalities\Features of Program:

Following functionalities are provided in the program:

- Could **write Polynomial Equations to File**.
- Could **read Polynomial Equations from File.**
- Could **Add Two Polynomial Equations** by using **'+'** operator directly.
- Could **Multiply Two Polynomial Equations** by using **'*'** operator directly.
- Could **Evaluate any Polynomial Equation** at a certain given point.
- Could also **Add and Multiply Multiple (more than two) Polynomial Equations** using **'+'** and **'*'** operator directly at once i.e. **P1+P2+P3** can be solved at once.

node.h File:

```
#pragma once
class node
{
public:
    int coefficient;
    int degree;
    node *next;
public:
    node(void);
};
```

node.cpp File:

```
#include "node.h"

node::node(void)
{
}
```

list.h File:

```
#include "node.h"

#pragma once
class list
{
public:
    node *head;
public:
    list(void);
    bool isEmpty(); //empty check
    void addNode(int, int); //insertion of polynomial equation node
    list operator + (list); //addition
    list operator * (list); //product
    int power(int, int); //to solve y^x
    int solve(int); //to evaluate polynomial equation at a given point
    void display(); //displays polynomial equations
};
```

list.cpp File:

```
#include "list.h"
#include <math.h>
#include <stdio.h>
#include "node.h"
#include <iostream>
using namespace std;

```

```
list::list(void)
{
    head = '\0';
}

bool list::isEmpty()
{
    if( head == '\0' ) //condition for empty list
    {
        return true;
    }
    else
    {
        return false;
    }
}

void list::addNode(int coefficient, int degree) //logic of addNode() is based on 'insert at end'
{
    node *ptr = new node; //node creation
    ptr->coefficient = 0;
    ptr->degree = 0;
    ptr->next = '\0';

    ptr->coefficient = coefficient; //assigning values
    ptr->degree = degree; //assigning values

    if( isEmpty() ) //empty list
    {
        head = ptr;
    }
    else
    {
        node *temp = head;

        while ( temp->next != '\0' ) //traverse to the last node
        {
            temp = temp->next;
        }

        temp->next = ptr;
    }
}

list list::operator + (list l) //operator '+' overloaded for addition
{
    list r; //to store addition result in that list

    node *t1 = head; //point to 1st list's head
    node *t2 = l.head; //point to 2nd list's head

    while( t1 != '\0' || t2 != '\0' ) //keep loop running until any list's head != '\0'
    {
        if( t1 != '\0' && t2 != '\0') //if pointer to both of the lists is not reached at end
        {
            if( t1->degree == t2->degree ) //multiplies only those terms with same degree
            {
```

```
                r.addNode( ( t1->coefficient + t2->coefficient), t1->degree);
                t1 = t1->next; //moves first list to next
                t2 = t2->next; //moves second list to next
            }
            else if( t1->degree > t2->degree) //if degree of first node > second node
            {
                r.addNode(t1->coefficient, t1->degree);
                t1 = t1->next; //moves first list to next
            }
            else if( t1->degree < t2->degree) //if degree of first node < second node
            {
                r.addNode(t2->coefficient, t2->degree);
                t2 = t2->next; //moves second list to next
            }
        }
        else if( t1 == '\0') //if first list is reached at end
        {
            r.addNode(t2->coefficient, t2->degree);
            t2 = t2->next; //moves second list to next because as is still remaining
        }
        else if( t2 == '\0') //if second list is reached at end
        {
            r.addNode(t1->coefficient, t1->degree);
            t1 = t1->next; //moves first list to next because as is still remaining
        }
    }
    return r; //returning resultant list to main
}

list list::operator * (list l)
{
    //=================Below Module will multiply first not full solve=============
    list r; //to store mutiplication result in that list

    node *t1 = head; //point to 1st list's head
            node *t2 = l.head; //point to 2nd list's head

            while ( t1 != '\0' )
            {
                while( t2 != '\0' )
                {
                    r.addNode( ( t1->coefficient * t2->coefficient ), ( t1->degree + t2->degree ) );
                    t2 = t2->next;
                }
                t1 = t1->next;
                t2 = l.head;
            }
            //=================Module below will now solve the multiplied solution further====================
            node *t = r.head; //point to list's head
            int max = t->degree;
            list res;

            while (t != '\0') //calculates maximum degree in the list
            {
                if( t->degree > max )
                {
                    max = t->degree;
                }
                t = t->next;
```

```
            }

            t = r.head;

            for( int i=max; i>=0; i--) //it will solve the terms which have same degree in the whole equation
            {
                int coefficient = 0;
                int degree = 0;

                while(t != '\0' )
                {
                    if(t->degree == i)
                    {
                        coefficient = coefficient + t->coefficient; //it adds coefficients of those terms which have same degree
                        degree = t->degree;
                    }
                    t = t->next;
                }
                t = r.head;
                if(coefficient != 0 )
                {
                    res.addNode(coefficient, degree); //it adds nodes sequentially to list 'res'
                }
            }
            return res;
        }

        int list::power(int base, int exponent) //recursion is used to calulate "base^exponent"
        {
            if(exponent != 0)
            {
                if( exponent == 1 ) //base case
                {
                    return ( base );
                }
                else //general case
                {
                    return ( base * power( base, exponent-1) );
                }
            }
            else //condition if "exponent == 0" i.e. x^0
            {
                return (1);
            }
        }

        int list::solve(int point) //function to solve equation
        {
            node *temp = head;
            int sum = 0;
            double multiplication;

            while( temp != '\0' )
            {
                multiplication = ( (temp->coefficient)*( power( point, temp->degree ) ) );
                sum = sum + multiplication;
```

```
                temp = temp->next;
            }
            return sum;
        }

        void list::display() //function to display list
        {
            node *temp = head;

            while ( temp != '\0' ) //traverse until reached at end
            {
                cout<<temp->coefficient<<"x^"<<temp->degree;

                if(temp->next == '\0') //just to eliminate '+' sign at end of the equation
                {
                }
                else
                {
                    cout<<"  +  ";
                }
                temp = temp->next;
            }
        }
```

main.cpp File:

```
#include "list.h"
#include "node.h"
#include "conio.h"
#include <string>
#include <cstdlib>
#include <sstream>
#include "fstream"
#include <iostream>
using namespace std;

void stringToList(string readList, list *l) //converts polynomial string to list
{
    readList += '+'; //added at end of string to evaluate till end
    string coefficientString; //to store coefficient string
    string degreeString; //to store degree string
    int lengthCoefficient; //to store coefficient length
    int lengthDegree; //to store degree string
    int coefficientStart = 0; //coefficient start position in string
    int coefficientEnd = 0; //coefficient end position in string
    int degreeStart = 0; //degree start position in string
    int degreeEnd = 0; //degree end position in string
    int coefficientNumber; //to store coefficient value
    int degreeNumber; //to store degree value

    for(int i=0; i<readList.length(); i++)
    {
        if( readList[i] == 'X' || readList[i] == 'x' ) //char == 'x' or 'X' it will take previous value as coefficient
        {
            coefficientEnd = i;
            lengthCoefficient = ( coefficientEnd - coefficientStart );
```

```
            coefficientString = readList.substr( coefficientStart, lengthCoefficient );

            stringstream convert(coefficientString);
            convert>>coefficientNumber;
        }

        if( readList[i] == '^' ) //if char == '^' it will set start position for degree at this point
        {
            degreeStart = ( i+1 );
        }

        if( readList[i] == '+' ) //if char == '+' it will take previous value as degree

        {
            degreeEnd = i;
            lengthDegree = ( degreeEnd - degreeStart );
            degreeString = readList.substr( degreeStart, lengthDegree );

            stringstream convert( degreeString );
            convert>>degreeNumber;

            coefficientStart = (i+1);
            l->addNode(coefficientNumber, degreeNumber);
        }
    }
}

void main()
{
    list list1;
    list list2;
    list additionList;
    list multiplicationList;
    int value;

    //==============GETTING USER INPUT AND, WRITING POLYNOMIAL EQUATION TO FILE==========
    string writeList; //to store polynomial equation to string

    ofstream file_write("polynomial.txt");

    cout<<"Enter Polynomial Equation-1:"<<endl;
    getline( cin, writeList );
    cout<<endl;
    file_write<<writeList; //writing plynomial equation-1 to file

    file_write<<endl;

    cout<<"Enter Polynomial Equation-2:"<<endl;
    getline( cin, writeList );
    cout<<endl;
    file_write<<writeList; //writing plynomial equation-2 to file

    file_write.close();
    cout<<"Writing equations to File. . ."<<endl;

    //==========================READING POLYNOMIAL EQUATION FROM FILE==================
    ifstream file_read("polynomial.txt");
```

```
    string readList1; //to store polynomial equation-1
    string readList2; //to store polynomial equation-2

    file_read>>readList1; //reading polynomial equaton-1 to string from file
    file_read>>readList2; //reading polynomial equaton-2 to string from file
    cout<<endl<<"Reading equations from File. . ."<<endl;

    //=====================CONVERTING POLYNOMIAL EQUATION STRING TO LIST==============
    stringToList(readList1, &list1);
    stringToList(readList2, &list2);

    //========================DISPLAYING POLYNOMIAL EQUATION 1 & 2===================
    cout<<endl<<"POLYNOMIAL EQUATION-1:"<<endl;
            cout<<"======================"<<endl;
            list1.display();
            cout<<endl;
            cout<<endl<<"POLYNOMIAL EQUATION-2:"<<endl;
            cout<<"======================"<<endl;
            list2.display();
            cout<<endl;

            //=================ADDING POLYNOMIAL EQUATION 1 & 2 AND DISPLAYING RESULT=============
            additionList = list1 + list2;
            cout<<endl<<"ADDITION RESULT:"<<endl;
            cout<<"================"<<endl;
            additionList.display();
            cout<<endl;

            //=============MULTIPLYING POLYNOMIAL EQUATION 1 & 2 AND DISPLAYING RESULT=============
            multiplicationList = list1 * list2;
            cout<<endl<<"MULTIPLICATION RESULT:"<<endl;
            cout<<"======================"<<endl;
            multiplicationList.display();
            cout<<endl;

            //========================EVALUATING POLYNOMIAL AT GIVEN POINT==============
            cout<<endl<<"Enter value of X to evaluate Polynomial-1:"<<endl;
            cin>>value;
            cout<<endl<<"POLYNOMIAL EQUATION-1 EVALUATION AT X = "<<value<<":"<<endl;
            cout<<"=========================================="<<endl;
            cout<<list1.solve(value);
            cout<<endl;
            cout<<endl<<"Enter value of X to evaluate Polynomial-2:"<<endl;
            cin>>value;
            cout<<endl<<"POLYNOMIAL EQUATION-2 EVALUATION AT X = "<<value<<":"<<endl;
            cout<<"=========================================="<<endl;
            cout<<list2.solve(value);

            getch();
        }
```

DISPLAY CONSOLE OUTPUT:

```
C:\Users\Muhammad Anas Baig\Documents\Visual Studio 2010\Projects\Assignent2\Debug\Assign...
Enter Polynomial Equation-1:
9x^5+2x^3+3x^1

Enter Polynomial Equation-2:
8x^3+6x^2+4x^1

Writing equations to File. . .

Reading equations from File. . .

POLYNOMIAL EQUATION-1:
======================
9x^5  +  2x^3  +  3x^1

POLYNOMIAL EQUATION-2:
======================
8x^3  +  6x^2  +  4x^1

ADDITION RESULT:
================
9x^5  +  10x^3  +  6x^2  +  7x^1

MULTIPLICATION RESULT:
======================
72x^8  +  54x^7  +  52x^6  +  12x^5  +  32x^4  +  18x^3  +  12x^2

Enter value of X to evaluate Polynomial-1:
1

POLYNOMIAL EQUATION-1 EVALUATION AT X = 1:
==========================================
14

Enter value of X to evaluate Polynomial-2:
4

POLYNOMIAL EQUATION-2 EVALUATION AT X = 4:
==========================================
624
```

TEXT FILE OUTPUT:

polynomial.txt File:

```
polynomial - Notepad
File  Edit  Format  View  Help
9x^5+2x^3+3x^1
8x^3+6x^2+4x^1
```