VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CSE 2003 – DATA STRUCTURES AND ALGORITHMS

WINTER 2017/18

SLOT – L39 + L40

FACULTY – PROF. GOPINATH M.P.

POLYNOMIAL IMPLEMENTATION USING LINKED LISTS

TEJAS JAMBHALE

17BCE0861

ABSTRACT -

Polynomials appear in a wide variety of areas of mathematics and science. For example, they are used to form polynomial equations, which encode a wide range of problems, from elementary word problems to complicated problems in the sciences; they are used to define **polynomial functions**, which appear in settings ranging from basic chemistry and physics to economics and social science; they are used in calculus and numerical analysis to approximate other functions. In advanced mathematics, polynomials are used to construct polynomial rings and algebraic varieties, central concepts in algebra and algebraic geometry.

This project is a practical implementation of the data structure "**Linked List**". A linked list is used to dynamically store user input of polynomial expressions and then various function are performed on polynomials depending on the user. For this, we follow the simple strategy:

- Make a polynomial abstract datatype using ***struct*** which basically implements a linked list.
- Write different functions for Creating (ie, adding more nodes to the linked list) a polynomial function, Adding two polynomials, Showing a polynomial expression, differentiate, multiply, subtract, negate, reverse, delete, extrema, solve, search.
- Finally write the main function with menu driven ability to perform various functions on polynomials depending on what the user wants.
- Apply this implementation on various real life problems to find solution and help us understand how data structures can be used to solve real life problems

INTRODUCTION –

A polynomial p(x) is the expression in variable x which is in the form ($ax^n + bx^{n-1} + \ldots + jx + k$), where a, b, c …., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.

A polynomial equation of degree 1 (n = 1) constitutes a linear equation. When n = 2, it is a quadratic equation; when n = 3, it is a cubic equation; when n = 4, it is a quartic equation; when n = 5, it is a quintic equation.

An important characteristics of polynomial is that each term in the polynomial expression consists of two parts:

One is the coefficient

Other is the exponent

Example:

$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 are its exponential value.

The manipulation of symbolic polynomials is a classic example of the use of list processing. We are able to represent any number of different polynomials as long as their combined size does not exceed our block of memory.

In general, the aim is to represent the polynomial where the nonzero coefficients and the exponents e, are nonnegative integers.

LINKED LIST –

A **linked list** is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

Link − Each link of a linked list can store a data called an element.

Next − Each link of a linked list contains a link to the next link called Next.

LinkedList − A Linked List contains the connection link to the first link called First.

Linked list can be visualized as a chain of nodes, where every node points to the next node.

As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

CREATING A NODE –

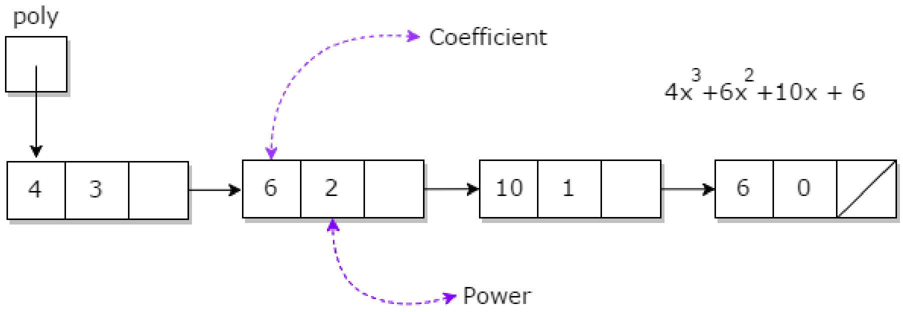
Structure is used rather than class to define Term to show that the data "members of Term" are public. A Polynomial Structure Node is defined to implement polynomials. Since a polynomial is to be represented by a list, there are two data parts and one address part. Each ListNode will represent a term in the polynomial. It is assumed that the coefficients are integers

A structure may be defined such that it contains two parts- one is the coefficient and second is the corresponding exponent. The structure definition may be given as shown below:

```
struct polynomial
{
        int coefficient;
        int exponent;
        polynomial *next;
};
```

This is the definition of the node class for a polynomial representation. The user can be asked to enter the terms in order of decreasing exponent and the coefficients and exponents can be stored in the coefficient and exponent fields of the node of a linked list.

$P(x) = 4x^3+6x^2+7x+9$



READ AND CREATING A POLYNOMIAL –

This function reads the input from the user. The Coefficient and exponent of each term is read. Create a node of type poly. Read value of coefficient and exponent from user and store in 'coeff' and 'pow' parts of the node. If more inputs are to be taken from the user then create a new node of existing polynomial and allocate space to it. Repeat above steps till no more input is given.

DISPLAYING A POLYNOMIAL –

This function prints the required polynomial when called. While next!=NULL all nodes in the polynomial are traversed and printed along with a variable x. While printing the constant term only coefficient is printed instead of x^0. Also if the coefficient is negative we add '–' before the term else '+' sign is added

ADDITION FUNCTION -

Consider addition of the following polynomials

5 x12 + 2 x9 + 4x7 + 6x6 + x3

7x8 + 2 x7 + 8x6 + 6x4 + 2x2 + 3 x + 40

The resulting polynomial is going to be

5x12 + 2 x9 + 7 x8 + 6 x7 + 14x6 + 6x4 +x3 +2x2 + 3 x + 40

Now notice how the addition was carried out. Let us say the result of addition is going to be stored in a third list. We started with the highest power in any polynomial. If there was no item having same exponent, we simply appended the term to the new list, and continued with the process.

Wherever we found that the exponents were matching, we simply added the coefficients and then stored the term in the new list.

If one list gets exhausted earlier and the other list still contains some lower order terms, then simply append the remaining terms to the new list.

<u>For subtraction, the polynomial to be subtracted is first negated and then same procedure is followed</u>

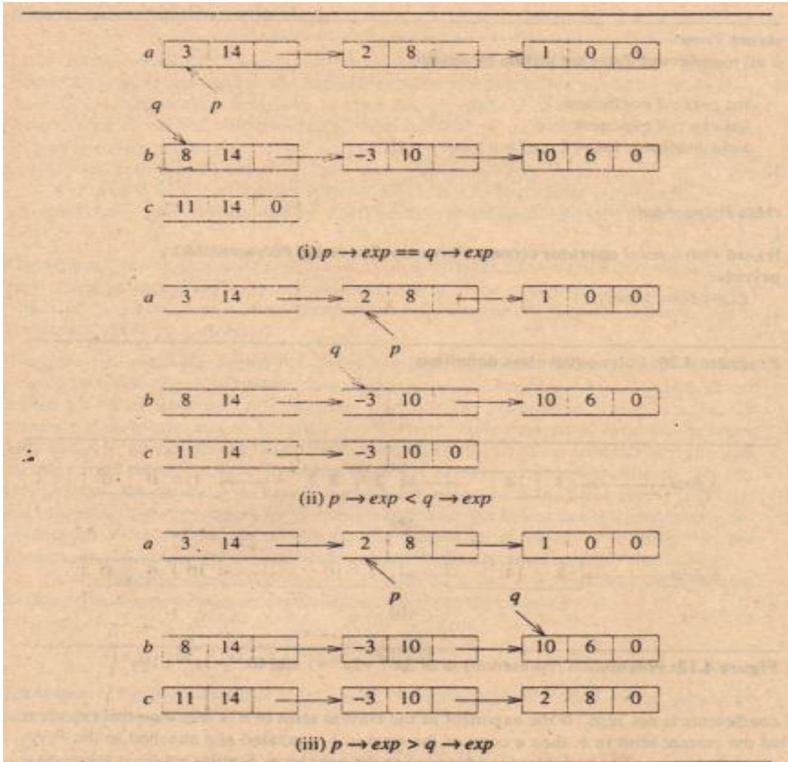


Figure 4.19: Generating the first three terms of $c = a + b$

- After having scanned two polynomials we need to traverse the entire linked list representing a polynomial for every node of the other polynomial.
- If the exponents match then we add the two coefficients and put them into a third list.

- If they do not match then the node is put into the third list.
- Next when one of the lists has been exhausted then the remaining nodes of the second polynomial whose exponent did not match any exponent from first list need to be put into the third resultant list.

MULTIPLICATION FUNCTION -

Multiplication of two polynomials requires manipulation of each node such that the exponents are added up and the coefficients are multiplied.

After each term of first polynomial is operated upon with each term of the second polynomial, then the result has to be added up by comparing the exponents and adding the coefficients for similar exponents and including terms as such with dissimilar exponents in the result.

- Input the multiplicand and multiplier
- Set both the polynomial in descending order of the coefficient
- Multiply each node of multiplicand with each node of the multiplier (multiplication of the coefficient part and addition of the exponent part) and add them into a newly formed linked list in descending order
- Coefficient having the same exponent value are added up with each other in the list and no two nodes have the same exponent value.
- Then the product is to be displayed in a proper way in the form of ax^n+bx^n-1+…
- Certain points to be noted before displaying a polynomial: Any coefficient with value 0 must not be displayed, 1x^n+2x^n-1 must not be displayed … node having coefficient value 1 must be displayed as x^n, node with exponent value 0 must be displayed as x not x^0, 1x^n-2x^n-1 format should be maintained not standard like errors 1x^n+-2x^n-1 should come up.

EVALUATION OF POLYNOMIAL –

Evaluation of polynomial requires multiplication of coefficient with given value of x to the power of the exponent of x at that term

Polynomials can be evaluated by simply taking the value of x at which the polynomial is to be evaluated. This value of x is raised to the power of the exponent of x of the term and this result is then multiplied with the coefficient to give value of that term at given value of x. Similarly this is calculated for all terms in the polynomial and all results are added up to give the final result.

- Insert the required polynomial to be evaluated
- Enter the value of x at which the polynomial is to be evaluated at. This value of x can be any real number
- While we traverse each term in the entire polynomial, set result as value of x raised to power of exponent multiplied by the coefficient. Once NULL is reached the execution is stopped and the result is displayed.

DERIVATIVE OF POLYNOMIAL –

This function finds the derivative of the polynomial with respect to the variable x. For a term with coefficient ‘a’ and exponent ‘n’, derivative is given by a(n)x^n-1. Likewise all nodes are visited and derivative is found for each to get a new polynomial. The derivative of n degree polynomial has degree n-1. The calculated derivative can be used to find extremum values and monotonicity of polynomial

- Input the polynomial for which derivative is to be calculated. Create another polynomial to store the result of derivation
- Traverse the entire matrix till NULL is not reached.
- Set the a pointer on result polynomial. Set the value of current node Coefficient as the product of given polynomials coefficient and exponent
- Set the value of current exponent as given polynomials exponent-1.
- After setting value of coefficient and exponent add a new node to the result polynomial and set pointer to the new node. Continue the traversal

Derivative of a polynomial has many applications in science and maths. Many equations in Physics use derivatives. We can find maxima and minima of polynomials using derivative. Derivative gives the rate of change of the polynomial over time and is used in graphing

INTEGRAL OF POLYNOMIAL –

Calculating the integral of a polynomial is similar to the derivative. We need to find the integral of each term in the polynomial and add the up to get the final integral. To find integral of a term we divide the coefficient of the term with exponent+1 and set the exponent as exponent +1.

- Input the polynomial for which integral has to be calculated. Create another polynomial to store the result of integration
- Traverse the entire polynomial till NULL is not reached
- Set the coefficient of resultant polynomial as coefficient of given polynomial divided by exponent+1
- Set the exponent of result polynomial as given polynomial exponent +1
- After setting the value of coefficient and exponent, add a new node to the result polynomial and allocate space. Continue the traversal to find integral of remaining terms

OTHER FUNCTIONS –

Number of terms- Count_Terms() –

The total number of terms in the polynomial is calculated. Each node is traversed while next!=NULL and count is incremented

Delete – Del() –

This functions deletes any term from the polynomial

Degree of Polynomial- Degree()-

The **degree** of a polynomial in one variable is the largest exponent in the polynomial. This can be easily implemented by traversing the list and finding largest exponent

Reverse – Rev()-

This function reverses the polynomial in ascending or descending order

Maxima or Minima of quadratic polynomial- Extrema()-

This function finds the max or min of a quadratic polynomial. A quadratic polynomial has a parabolic graph. The value of –b/2a gives us the value of maxima or minima where b is the coefficient of term with exponent 1 and a term with exponent 2. if a>0 then this value is the minima and if a<0 then this value is the maxima

Roots of linear and quadratic polynomial- Root_linear() and Root_Quadratic()–

This function finds root of linear by using formula x=-b/a where x is root. For quadratic it uses the formula

–b+sqrt(b^2-4ac)/2a and –b-sqrt(b^2-4ac)/2a to give the roots of the equation

Monotonicity of quadratic polynomial- Mon()–

This can be found by finding point of maxima or minima and depending on value of a decides where the function is increasing or decreasing

Finding Coefficient- coeff()-

The coefficient of any term can be found by searching the list for required exponent and corresponding coefficient is printed

Negating- Neg()-

Each node in the polynomial is visited and the coefficient is multiplied by -1 to find the negation

Multiplying by constant – MultiConst()–

Each node in the polynomial is visited and the coefficient is multiplied by the given constant

CODE IMPLEMENTATION -

Project.cpp - Code::Blocks 16.01

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

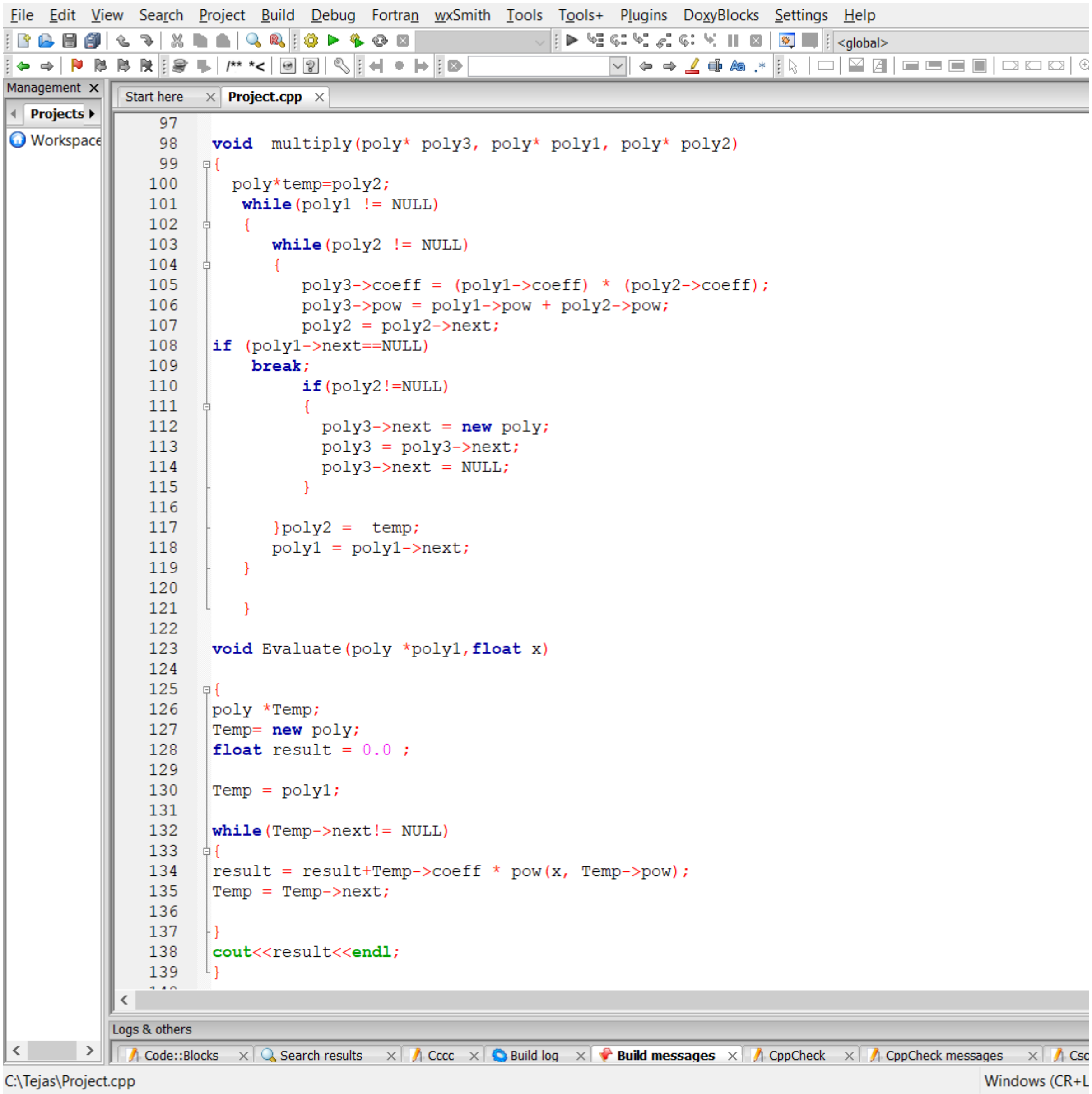
```cpp
/*tejas jambhale 17BCE0861*/
#include <iostream>
#include <math.h>
using namespace std;

struct poly{
    float coeff;
    int pow;
    poly *next;
};

void create(poly *node)
{
 char ch;
 cout<<"Enter the polynomial"<<endl;
  cout<<"Enter coefficient and exponent of each term"<<endl;
 do
 {

  cout<<"\n enter coeff:";
  cin>>node->coeff;
  cout<<"\n enter power:";
  cin>>node->pow;
  node->next=new poly;
  node=node->next;
  node->next=NULL;
  cout<<"\n continue(y/n):";
  cin>>ch;
 }
 while(ch=='y' || ch=='Y');
}

void show(poly *node,int flag)
{
    if (flag==0)
cout<<"The entered polynomial is"<<endl;
 while(node->next!=NULL)
 {
     if (node->pow!=0)
  cout<<node->coeff<<"x^"<<node->pow;
  else
    cout<<node->coeff;
  node=node->next;
  if(node->next!=NULL)
  {
      if (node->coeff>0)
          cout<<"+";
      else
        cout<<"";
  }
```

C:\Tejas\Project.cpp

ADDITION FUNCTION -

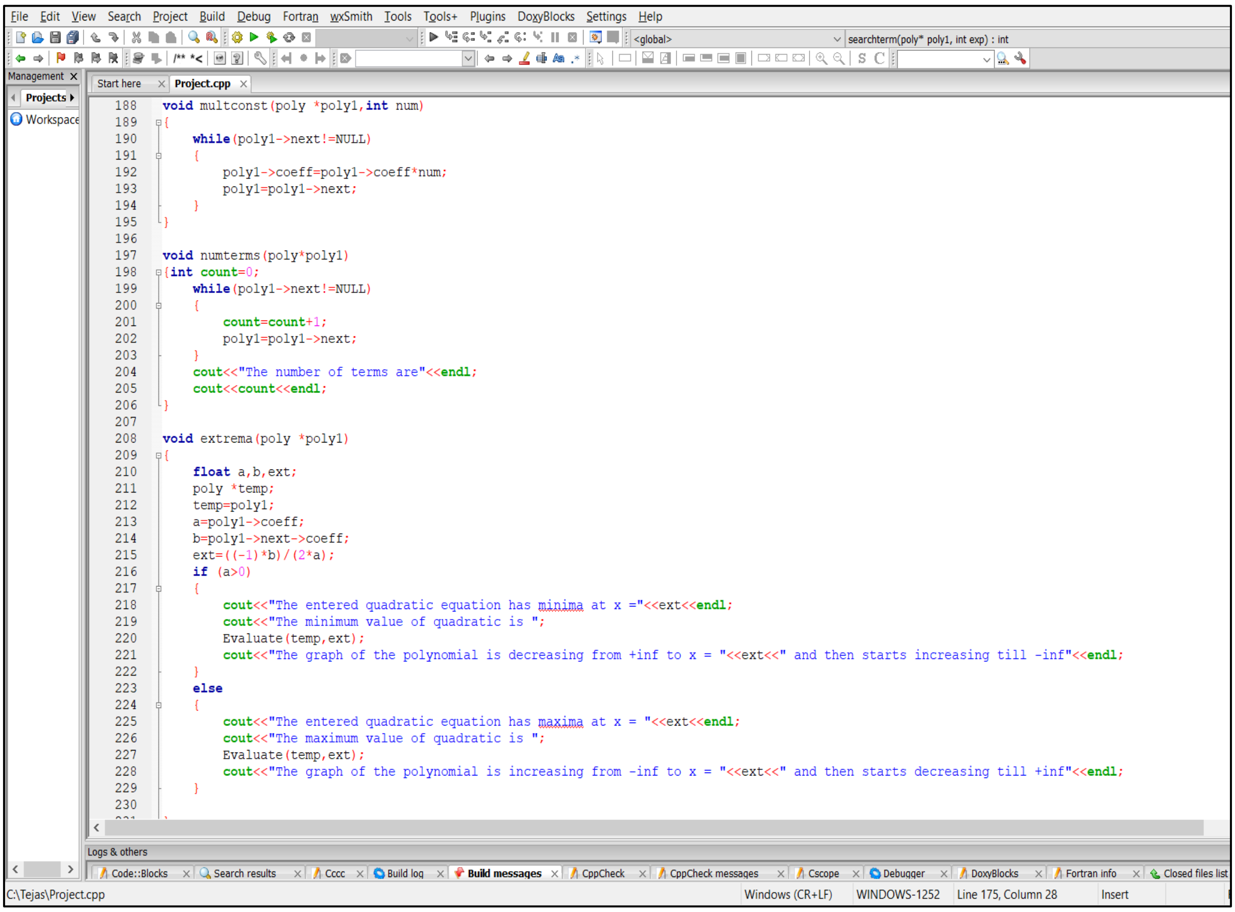
```
void polyadd(poly *poly1, poly *poly2, poly *poly3)
{
while(poly1->next && poly2->next)
    {
        if(poly1->pow > poly2->pow)
        {
            poly3->pow = poly1->pow;
            poly3->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        else if(poly1->pow < poly2->pow)
        {
            poly3->pow = poly2->pow;
            poly3->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        else
        {
            poly3->pow = poly1->pow;
            poly3->coeff = poly1->coeff+poly2->coeff;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }

        poly3->next = new poly;
        poly3 = poly3->next;
        poly3->next = NULL;
    }
while(poly1->next || poly2->next)
    {
        if(poly1->next)
        {
            poly3->pow = poly1->pow;
            poly3->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if(poly2->next)
        {
            poly3->pow = poly2->pow;
            poly3->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        poly3->next = new poly;
        poly3 = poly3->next;
        poly3->next = NULL;
    }
}
```

MULTIPLICATION AND EVALUATION FUNCTION –

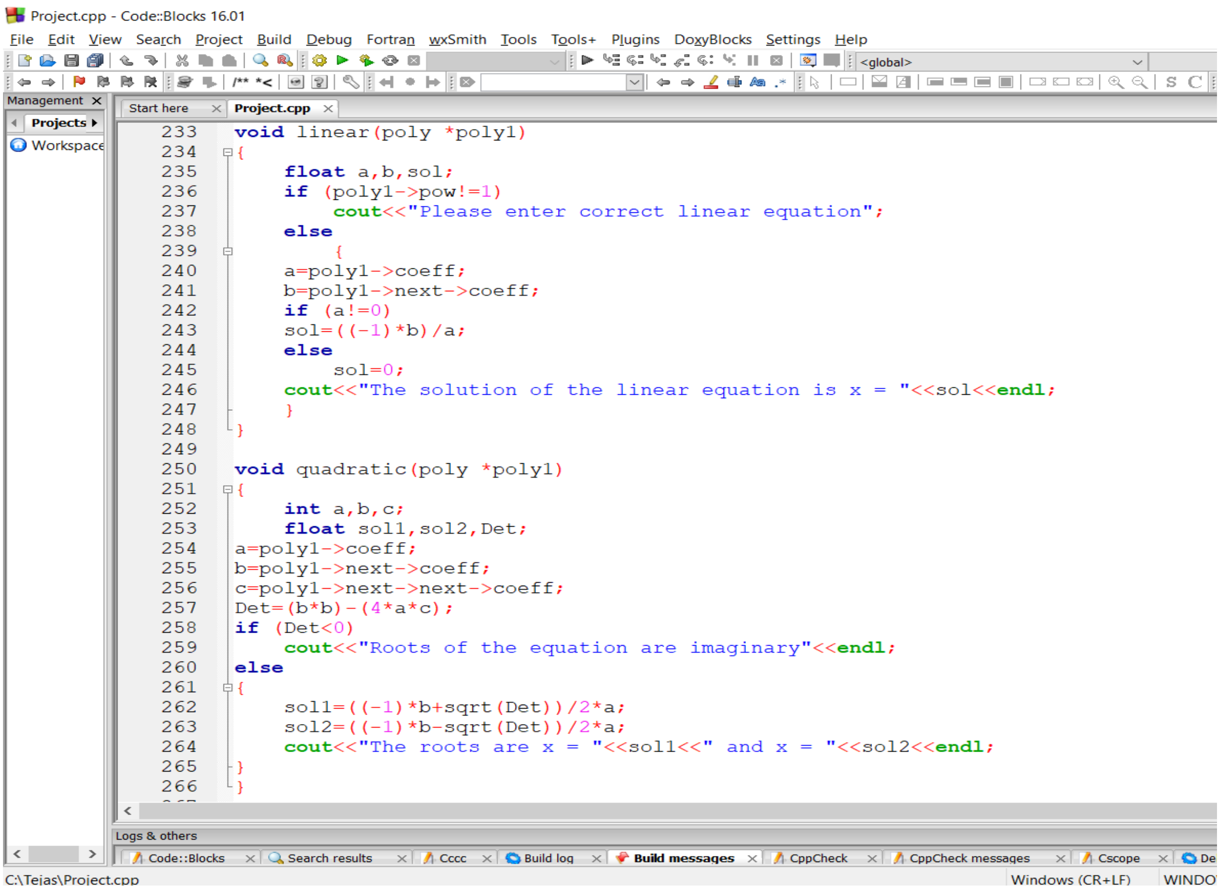


DERIVATION, DEGREE AND SEARCHING–

```cpp

void derivative(poly* poly2, poly* deri)
{
  poly* ptr;
  ptr = poly2;
  while(ptr->next!=NULL)
  {
     deri->coeff = (ptr->coeff)*(ptr->pow);
     if(ptr->pow == 0)
     {
        deri->pow = 0;
     }
     else
        deri->pow = (ptr->pow)-1;
     deri->next = NULL;
     ptr = ptr->next;
     deri->next = new poly;
     deri = deri->next;
     deri->next = NULL;
  }
}

void degree(poly *poly1)
{
    int deg;
    deg=poly1->pow;
    cout<<"The degree is"<<endl;
   cout<<deg<<endl;
}


int searchterm(poly *poly1,int exp)
{
    while(poly1->next!=NULL)
    {
        if(poly1->pow==exp)
           {
               cout<<poly1->coeff<<"x^"<<poly1->pow<<endl;
               return poly1->coeff;
                break;
           }
        else
            poly1=poly1->next;

    }
    cout<<"The term is not present in entered polynomial"<<endl;
}

```

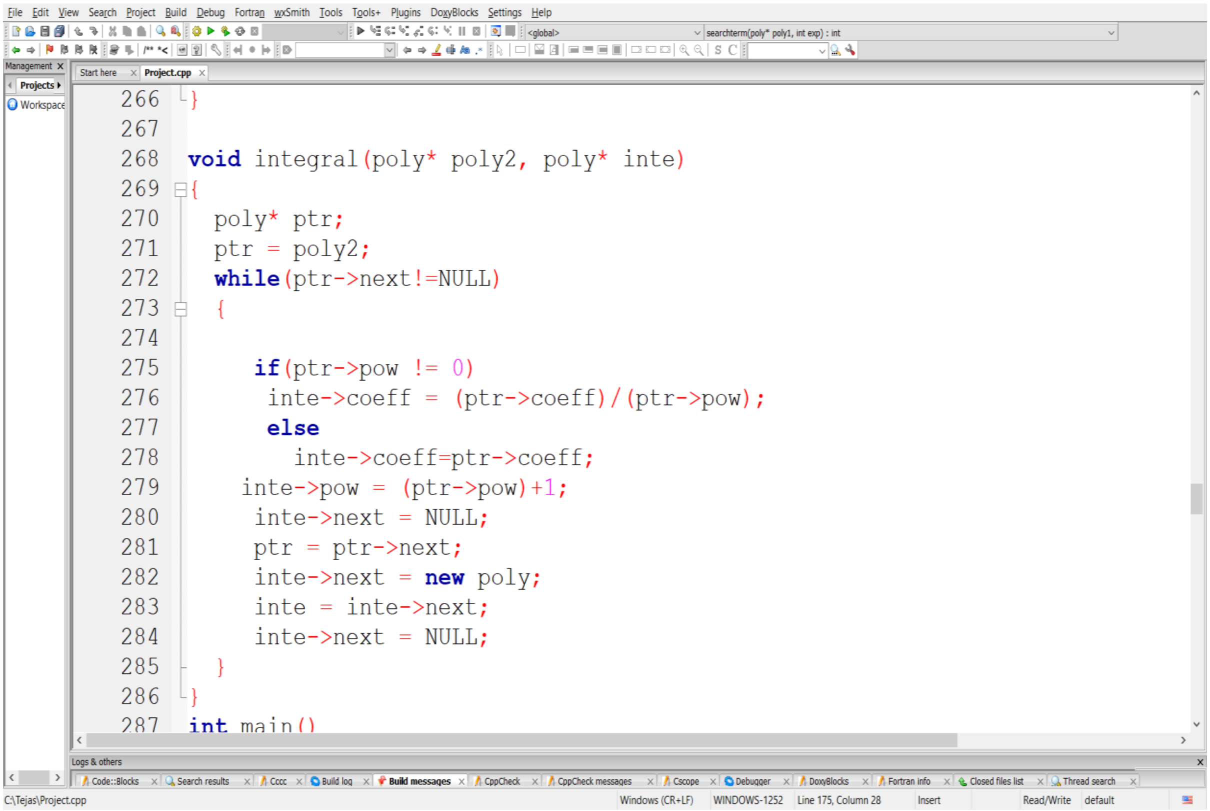


```
void multconst(poly *poly1,int num)
{
    while(poly1->next!=NULL)
    {
        poly1->coeff=poly1->coeff*num;
        poly1=poly1->next;
    }
}

void numterms(poly*poly1)
{int count=0;
    while(poly1->next!=NULL)
    {
        count=count+1;
        poly1=poly1->next;
    }
    cout<<"The number of terms are"<<endl;
    cout<<count<<endl;
}

void extrema(poly *poly1)
{
    float a,b,ext;
    poly *temp;
    temp=poly1;
    a=poly1->coeff;
    b=poly1->next->coeff;
    ext=((-1)*b)/(2*a);
    if (a>0)
    {
        cout<<"The entered quadratic equation has minima at x ="<<ext<<endl;
        cout<<"The minimum value of quadratic is ";
        Evaluate(temp,ext);
        cout<<"The graph of the polynomial is decreasing from +inf to x = "<<ext<<" and then starts increasing till -inf"<<endl;
    }
    else
    {
        cout<<"The entered quadratic equation has maxima at x = "<<ext<<endl;
        cout<<"The maximum value of quadratic is ";
        Evaluate(temp,ext);
        cout<<"The graph of the polynomial is increasing from -inf to x = "<<ext<<" and then starts decreasing till +inf"<<endl;
    }
```



```
void linear(poly *poly1)
{
    float a,b,sol;
    if (poly1->pow!=1)
        cout<<"Please enter correct linear equation";
    else
        {
    a=poly1->coeff;
    b=poly1->next->coeff;
    if (a!=0)
    sol=((-1)*b)/a;
    else
        sol=0;
    cout<<"The solution of the linear equation is x = "<<sol<<endl;
     }
}

void quadratic(poly *poly1)
{
    int a,b,c;
    float sol1,sol2,Det;
a=poly1->coeff;
b=poly1->next->coeff;
c=poly1->next->next->coeff;
Det=(b*b)-(4*a*c);
if (Det<0)
    cout<<"Roots of the equation are imaginary"<<endl;
else
{
    sol1=((-1)*b+sqrt(Det))/2*a;
    sol2=((-1)*b-sqrt(Det))/2*a;
    cout<<"The roots are x = "<<sol1<<" and x = "<<sol2<<endl;
}
}
```

INTEGRATION FUNCTION –



```
}

void integral(poly* poly2, poly* inte)
{
  poly* ptr;
  ptr = poly2;
  while(ptr->next!=NULL)
  {

     if(ptr->pow != 0)
      inte->coeff = (ptr->coeff)/(ptr->pow);
      else
        inte->coeff=ptr->coeff;
    inte->pow = (ptr->pow)+1;
     inte->next = NULL;
     ptr = ptr->next;
     inte->next = new poly;
     inte = inte->next;
     inte->next = NULL;
  }
}
int main()
```

```cpp
}
int main()
{
    do
    {
        int option;
        poly *poly1,*poly2,*poly3;
          poly1=new poly;
        poly2=new poly;
        poly3=new poly;
        cout<<"***************POLYNOMIAL IMPLEMENTATION***************"<<endl<<endl;
        cout<<"Choose an option"<<endl<<endl;
        cout<<" 1. Addition of two polynomials"<<endl;
        cout<<" 2. Subtraction of two polynomials"<<endl;
        cout<<" 3. Multiplication of two polynomials"<<endl;
        cout<<" 4. Evaluating a Polynomial"<<endl;
        cout<<" 5. Derivative of a Polynomial"<<endl;
        cout<<" 6. Integral of a polynomial"<<endl;
        cout<<" 7. Find Degree of Polynomial"<<endl;
        cout<<" 8. To Search for a term in a Polynomial"<<endl;
        cout<<" 9. Find negation of a polynomial"<<endl;
        cout<<" 10. Multiply the polynomial with a constant"<<endl;
        cout<<" 11. Count the number of terms in the polynomial"<<endl;
        cout<<" 12. Find Extrema (MAX or MIN) and Monotonicity of Quadratic Polynomial"<<endl;
        cout<<" 13. Solve a linear Polynomial"<<endl;
        cout<<" 14. Solve a Quadratic polynomial"<<endl;
        cout<<" 15. Exit"<<endl<<endl;
        cin>>option;
```

Project.cpp - Code::Blocks 16.01

```cpp
        case 1:
            create(poly1);
            show(poly1,0);
            create(poly2);
            show(poly2,0);
            polyadd(poly1,poly2,poly3);
            cout<<"Sum of two polynomials entered is"<<endl;
            show(poly3,1);
            break;

        case 2:
            create(poly1);
            show(poly1,0);
            create(poly2);
            show(poly2,0);
            multconst(poly2,-1);
            polyadd(poly1,poly2,poly3);
            cout<<"Difference of two polynomials entered is"<<endl;
            show(poly3,1);
            break;
        case 3:
            create(poly1);
            show(poly1,0);
            create(poly2);
            show(poly2,0);
            multiply(poly3,poly2,poly1);
            cout<<"Product of two polynomials entered is"<<endl;
            show(poly3,1);
            break;
        case 4:
            create(poly1);
            show(poly1,0);
            float num;
            cout<<"Enter value at which polynomial is to be evaluated"<<endl;
            cin>>num;
            cout<<"The value of the polynomial at x = "<<num<<" is ";
            Evaluate(poly1,num);
            break;
```

C:\Tejas\Project.cpp

Project.cpp - Code::Blocks 16.01

```cpp
            case 5:
                create(poly1);
                show(poly1,0);
                derivative(poly1,poly3);
                cout<<"The derivative of entered polynomial is"<<endl;
                show(poly3,1);
                break;
            case 6:
                create(poly1);
                show(poly1,0);
                integral(poly1,poly3);
                cout<<"The integral of the entered polynomial is"<<endl;
                show(poly3,1);
                break;
            case 7:
                create(poly1);
                show(poly1,0);
                degree(poly1);
                break;
            case 8:
                create(poly1);
                show(poly1,0);
                int exp;
                cout<<"Enter exponent of term to be searched"<<endl;
                cin>>exp;
                searchterm(poly1,exp);
                break;
            case 9:
                create(poly1);
                show(poly1,0);
                multconst(poly1,-1);
                cout<<"The negation is"<<endl;
                show(poly1,1);
                break;
            case 10:
                create(poly1);
                show(poly1,0);
                cout<<"Enter constant to be multiplied"<<endl;
                int num1;
                cin>>num1;
                multconst(poly1,num1);
                cout<<"The result is"<<endl;
                show(poly1,1);
                break;
            case 11:
```

C:\Tejas\Project.cpp

Project.cpp - Code::Blocks 16.01

```cpp
                break;
            case 11:
                create(poly1);
                show(poly1,0);
                numterms(poly1);
                break;
            case 12:
                create(poly1);
                show(poly1,0);
                extrema(poly1);
                break;
            case 13:
                create(poly1);
                show(poly1,0);
                linear(poly1);
                break;
            case 14:
                create(poly1);
                show(poly1,0);
                quadratic(poly1);
                break;
            case 15:
                return 0;
        }
        cout<<endl<<endl;
    }
    while(1);
}
```

C:\Tejas\Project.cpp

OUTPUT -

C:\Tejas\Project.exe

```
****************POLYNOMIAL IMPLEMENTATION****************

Choose an option

 1. Addition of two polynomials
 2. Subtraction of two polynomials
 3. Multiplication of two polynomials
 4. Evaluating a Polynomial
 5. Derivative of a Polynomial
 6. Integral of a polynomial
 7. Find Degree of Polynomial
 8. To Search for a term in a Polynomial
 9. Find negation of a polynomial
 10. Multiply the polynomial with a constant
 11. Count the number of terms in the polynomial
 12. Find Extrema (MAX or MIN) and Monotonicity of Quadratic Polynomial
 13. Solve a linear Polynomial
 14. Solve a Quadratic polynomial
 15. Exit
```

OUTPUT FOR ADDITION -

C:\Tejas\Project.exe

```
1
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:4

 enter power:3

 continue(y/n):y

 enter coeff:7

 enter power:2

 continue(y/n):y

 enter coeff:12

 enter power:1

 continue(y/n):y

 enter coeff:9

 enter power:0

 continue(y/n):n
The entered polynomial is
4x^3+7x^2+12x^1+9
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:34

 enter power:2

 continue(y/n):y

 enter coeff:6

 enter power:1

 continue(y/n):n
The entered polynomial is
34x^2+6x^1
Sum of two polynomials entered is
4x^3+41x^2+18x^1+9
```

OUTPUT FOR MULTIPLICATION

C:\Tejas\Project.exe

```
3
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:3

 enter power:2

 continue(y/n):y

 enter coeff:4

 enter power:1

 continue(y/n):y

 enter coeff:9

 enter power:0

 continue(y/n):n
The entered polynomial is
3x^2+4x^1+9
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:7

 enter power:3

 continue(y/n):y

 enter coeff:8

 enter power:0

 continue(y/n):n
The entered polynomial is
7x^3+8
Product of two polynomials entered is
21x^5+28x^4+63x^3+24x^2+32x^1+72
```

OUTPUT FOR EVALUATION -

C:\Tejas\Project.exe

```
4
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:7

 enter power:4

 continue(y/n):y

 enter coeff:12

 enter power:3

 continue(y/n):y

 enter coeff:10

 enter power:2

 continue(y/n):y

 enter coeff:7

 enter power:0

 continue(y/n):n
The entered polynomial is
7x^4+12x^3+10x^2+7
Enter value at which polynomial is to be evaluated
4
The value of the polynomial at x = 4 is 2727
```

OUTPUT FOR DERIVATION -

C:\Tejas\Project.exe

```
5
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:3

 enter power:4

 continue(y/n):y

 enter coeff:5

 enter power:2

 continue(y/n):y

 enter coeff:16

 enter power:1

 continue(y/n):y

 enter coeff:7

 enter power:0

 continue(y/n):n
The entered polynomial is
3x^4+5x^2+16x^1+7
The derivative of entered polynomial is
12x^3+10x^1+160
```

OUTPUT FOR INTEGRATION -

C:\Tejas\Project.exe

```
6
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:8

 enter power:3

 continue(y/n):y

 enter coeff:12

 enter power:2

 continue(y/n):y

 enter coeff:4

 enter power:1

 continue(y/n):y

 enter coeff:14

 enter power:0

 continue(y/n):n
The entered polynomial is
8x^3+12x^2+4x^1+14
The integral of the entered polynomial is
2x^4+4x^3+2x^2+14x^1
```

OUTPUT FOR SEARCH -

C:\Tejas\Project.exe

```
8
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:5

 enter power:3

 continue(y/n):y

 enter coeff:17

 enter power:2

 continue(y/n):y

 enter coeff:6

 enter power:1

 continue(y/n):y

 enter coeff:9

 enter power:0

 continue(y/n):n
The entered polynomial is
5x^3+17x^2+6x^1+9
Enter exponent of term to be searched
1
6x^1
```

OUTPUT FOR MULTIPLY BY CONSTANT -

C:\Tejas\Project.exe

```
10
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:7

 enter power:4

 continue(y/n):y

 enter coeff:15

 enter power:3

 continue(y/n):y

 enter coeff:5

 enter power:2

 continue(y/n):y

 enter coeff:4

 enter power:1

 continue(y/n):y

 enter coeff:6

 enter power:0

 continue(y/n):n
The entered polynomial is
7x^4+15x^3+5x^2+4x^1+6
Enter constant to be multiplied
4
The result is
28x^4+60x^3+20x^2+16x^1+24
```

OUTPUT FOR EXTREMAS -

C:\Tejas\Project.exe

```
12
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:12

 enter power:2

 continue(y/n):y

 enter coeff:4

 enter power:1

 continue(y/n):y

 enter coeff:-2

 enter power:0

 continue(y/n):n
The entered polynomial is
12x^2+4x^1-2
The entered quadratic equation has minima at x =-0.166667
The minimum value of quadratic is -2.33333
The graph of the polynomial is decreasing from +inf to x = -0.166667 and then starts increasing till -inf
```

OUTPUT FOR SOLUTION OF LINEAR EQUATION -

C:\Tejas\Project.exe

```
13
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:3

 enter power:1

 continue(y/n):y

 enter coeff:54

 enter power:0

 continue(y/n):n
The entered polynomial is
3x^1+54
The solution of the linear equation is x = -18
```

OUTPUT FOR SOLUTION OF QUADRATIC EQUATION -

C:\Tejas\Project.exe

```
14
Enter the polynomial
Enter coefficient and exponent of each term

 enter coeff:4

 enter power:2

 continue(y/n):y

 enter coeff:6

 enter power:1

 continue(y/n):y

 enter coeff:-2

 enter power:0

 continue(y/n):n
The entered polynomial is
4x^2+6x^1-2
The roots are x = 4.49242 and x = -28.4924
```

TIME COMPLEXITY	
FUNCTION	**USING LINKED LIST**
ADDITION	O(m+n)
MULTIPLICATION	O(mn)
EVALUATION	O(n)
SEARCH	O(n)
DELETION	O(1)
DEGREE	O(1)
NUM OF TERMS	O(n)
EXTREMA	O(n)
DIFFERENTIATION	O(n^2)
lINEAR	O(1)
QUADRATIC	O(1)
MULTCONST	O(n)

COMPARING LINKED LISTS IMPLEMETATION AND ARRAY IMPLEMENTATION OF POLYNOMIAL -

A polynomial can be represented in an array or in a linked list by simply storing the coefficient and exponent of each term.

Each node will need to store the exponent and the coefficient for each term. It often does not matter whether the polynomial is in x or y. This information may not be very crucial for the intended operations on the polynomial. Thus we need to define a node structure to hold two integers, exp and coeff

However, for any polynomial operation, such as addition or multiplication of polynomials , you will find that the linked list representation is more easier to deal with.

In a polynomial all the terms may not be present, especially if it is going to be a very high order polynomial.

Consider,

5 x12 + 2 x9 + 4x7 + 6x5 + x2 + 12 x

Now this 12th order polynomial does not have all the 13 terms (including the constant term).It would be very easy to represent the polynomial using a linked list structure, where each node can hold information pertaining to a single term of the polynomial.

Compare this representation with storing the same polynomial using an array structure.

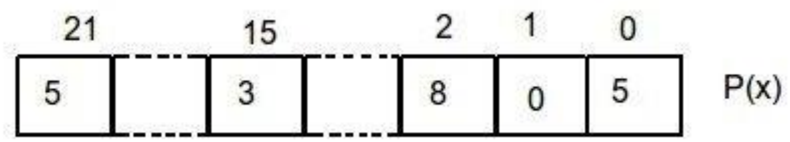
In the array we have to have keep a slot for each exponent of x, thus if we have a polynomial of order 50 but containing just 6 terms, then a large number of entries will be zero in the array.

You will also see that it would be also easy to manipulate a pair of polynomials if they are represented using linked lists. For tasks such as addition, deletion, multiplication linked list makes it considerably easier.

Using array for polynomial operations will consume a lot of space. For solving polynomial equations, you need to have two parallel data structures. Index of both the data structures move forward to perform the desired operation. Let's take an example.

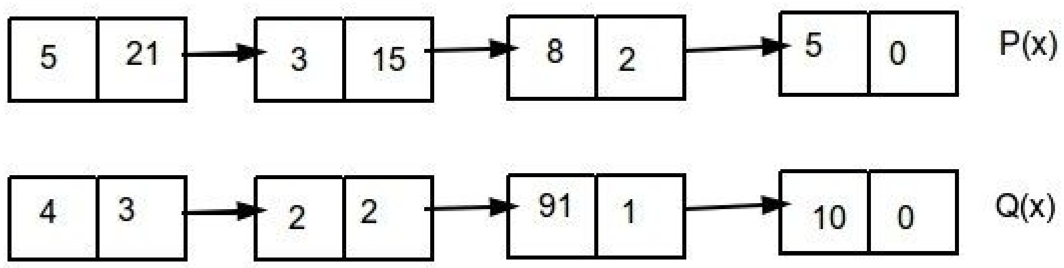
$P(x) = 5x21+3x15+8x2+5$
$Q(x) = 4x3+2x2+91x+10$
We need to find $G(x) = P(x) + Q(x)$

This is the array representation of both the Polynomials.



As you can see here, for storing just 4 values, N values are used (N being the highest power). Now we'll have one pointer which will move from index 21 to 0 adding elements of both the arrays. If we use polynomials with higher power there will be a considerable amount of **memory wastage**.

Now look at the linked list representation of the same polynomials.



The first element in each node is the data value whereas the second one is the power. Its clear that there is very less amount of memory space used also the computation will be less. Following is the structure node architecture.

Other problems with using arrays are

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, upper limit is rarely reached.

2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.
For example, suppose we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040, …..].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

Linked lists are preferable over arrays when:
a) You need constant-time insertions/deletions from the list (such as in real-time computing where time predictability is absolutely critical)

b) You don't know how many items will be in the data structure

ADVANTAGES OF LINKED LISTS –

Dynamic Data Structure

Linked list is a dynamic data structure so it can grow and shrink at runtime by allocating and deallocating memory. So there is no need to give initial size of linked list.

Insertion and Deletion

Insertion and deletion of nodes are really easier. Unlike array here we don't have to shift elements after insertion or deletion of an element. In linked list we just have to update the address present in next pointer of a node.

No Memory Wastage

As size of linked list can increase or decrease at run time so there is no memory wastage. In case of array there is lot of memory wastage, like if we declare an array of size 10 and store only 6 elements in it then space of 4 elements are wasted. There is no such problem in linked list as memory is allocated only when required.

Implementation

Data structures such as stack and queues can be easily implemented using linked list

DISADVANTAGES OF LINKED LISTS

- Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.
- More memory is required to store elements in linked list as compared to array. Because in linked list each node contains a pointer and it requires extra memory for itself.
- Arrays have better cache locality that can make a difference in performance.
- In singly linked lists we cannot traverse in reverse direction. If we want to visit previous node we need start from the first node and traverse from beginning

CONCLUSION –

- Polynomial implementation has many applications in various field like physics, economics, mathematics, computer science, business, biology. Polynomials are often used to show trends or are formed to analyse problems. The equation of a line ax+b is one of the most basic equations in maths
- Since polynomials are used to describe curves of various types, people use them in the real world to graph curves. For example, roller coaster designers may use polynomials to describe the curves in their rides. Combinations of polynomial functions are sometimes used in economics to do cost analyses,
- Polynomials can also be used to model different situations, like in the stock market to see how prices will vary over time. Business people also use polynomials to model markets, as in to see how raising the price of a good will affect its sales. Additionally, polynomials are used in physics to describe the trajectory of projectiles. Polynomial integrals (the sums of many polynomials) can be used to express energy, inertia and voltage difference, to name a few applications.