1. Design a class Polynomial. A polynomial is a sequence of terms. a term is a pair (exponent, coefficient) where the exponent is a non-negative integer and the coefficient is a double-precision number.
   1. Write a class definition for a term. A single constructor is enough.
   2. Write a class definition for a polynomial. Given that polynomials have an arbitrary size, the proper implementation is by means of a linked list, i.e. each term has a pointer to the next term.
   3. Write a constructor for a polynomial, whose input is a term.
   4. Write operators that add a term to a polynomial, and that multiply a term and a polynomial.
   5. Write an operator that adds two polynomials.
   6. Write an operator that multiplies two polynomials.
   7. Write a destructor for the polynomial class.

Write an output routine for polynomials, overloading the << operator. Assume that the name of the variable is X

* #include <iostream>

Using namespace std;

classpolynomial

{

private:

structpolynomialnode

{

float coeff;

int exp;

polynomial node\*link ;

}\*p ;

public:

polynomial();

void polynomial\_append(float c, int e );

void display\_polynomial();

void polynomial\_multiply(polynomial&p1, polynomial&p2 );

void padd(float c, int e );

~polynomial();

};

polynomial::polynomial()

{

p =NULL;

}

voidpolynomial::polynomial\_append(float c, int e )

{

polynomialnode\*temp ;

temp= p ;

if( temp ==NULL)

{

temp=newpolynomialnode;

p =temp ;

}

else

{

while( temp -> link !=NULL)

temp= temp -> link ;

temp-> link =newpolynomialnode;

temp= temp -> link ;

}

temp->coeff= c ;

temp->exp= e ;

temp-> link =NULL;

}

voidpolynomial::display\_polynomial()

{

polynomialnode\*temp = p ;

int f =0;

while( temp !=NULL)

{

if( f !=0)

{

if( temp ->coeff>0)

cout<<" + ";

else

cout<<" ";

}

if( temp ->exp!=0)

cout<< temp ->coeff<<"x^"<< temp ->exp;

else

cout<< temp ->coeff;

temp= temp -> link ;

f =1;

}

}

voidpolynomial::polynomial\_multiply(polynomial&p1, polynomial&p2 )

{

polynomialnode\*temp1, \*temp2 ;

float coeff1, exp1 ;

temp1 =p1.p;

temp2 =p2.p;

if( temp1 ==NULL&& temp2 ==NULL)

return;

if( temp1 ==NULL)

p =p2.p;

else

{

if( temp2 ==NULL)

p =temp1 ;

else {

while( temp1 !=NULL)

{

while( temp2 !=NULL)

{

coeff1 = temp1 ->coeff\* temp2 ->coeff;

exp1 = temp1 ->exp+ temp2 ->exp;

temp2 = temp2 ->link ;

padd( coeff1, exp1 );

}

temp2 =p2.p;

temp1 = temp1 ->link ;

}

}

}

}

voidpolynomial::padd(float c, int e )

{

polynomialnode\*r, \*temp ;

temp= p ;

if( temp ==NULL|| c > temp ->exp)

{

r =newpolynomialnode;

r->coeff= c ;

r->exp= e ;

if( p ==NULL)

{

r-> link =NULL;

p =r ;

}

else

{

r-> link = temp ;

p =r ;

}

}

else

{

while( temp !=NULL)

{

if( temp ->exp== e )

{

temp->coeff+= c ;

return;

}

if( temp ->exp> c &&( temp -> link ->exp< c ||

temp-> link ==NULL))

{

r =newpolynomialnode;

r->coeff= c;

r->exp= e ;

r-> link =NULL;

temp-> link = r ;

return;

}

temp= temp -> link ;

}

r-> link =NULL;

temp-> link = r ;

}

}

polynomial:: ~polynomial()

{

polynomialnode\*q ;

while( p !=NULL)

{

q = p ->link ;

delete p ;

p =q ;

}

}

void main()

{

polynomial p1 ;

p1.polynomial\_append(3, 5);

p1.polynomial\_append(2, 4);

p1.polynomial\_append(1, 2);

cout<<"**\n** First polynomial: "<<endl;

p1.display\_polynomial();

polynomial p2 ;

p2.polynomial\_append(1, 6);

p2.polynomial\_append(2, 5);

p2.polynomial\_append(3, 4);

cout<<"**\n** Second polynomial: "<<endl;

p2.display\_polynomial();

polynomial p3 ;

p3.polynomial\_multiply( p1, p2 );

cout<<"**\n** Resultant polynomial: "<<endl;

p3.display\_polynomial();

}