```
/**
 * Devoir libre Listes doubelement chainées et application sur les polynomes
réels
 *
 * Dans ce programme on va implémenter une liste deoublement chainée
 * et son application pour representer et gérer des poly réels.
 *
 * Bimone : TRAORE Souleymane
 *          RACHIDI OUSSAMA
 * Date : 2023-04-18
 * version : 1.0
 */

/**
 * Définition des types
 */
 #include<iostream>
 #include <cstddef>
 #include<math.h>
 #include<cstring>

 using namespace std;

typedef struct {
    double coefficient ;
    int exposant ;
} Monome ;

typedef Monome TypeListe;

typedef struct cellule {
    TypeListe donnee ;
    struct cellule * suivant ;
    struct cellule * precedent ;
} Cellule ;


typedef struct {
    Cellule * debut ; // pointeur sur le début
    Cellule * fin ; // pointeur sur le fin
} Liste;

typedef Liste Polynome;


//////////////  Partie 1. Gestion d'une liste doublement chainée
```

```
/**
 * Fonction pour créer et initialiser une liste vide
 */
Liste * init_liste (){
    Liste * p = new Liste;
    p->debut = NULL;
    p->fin = NULL;
    return p;
}

/**
 * Fonction pour ajouter une valeur au début
 */
void ajouter_debut (Liste * L, TypeListe valeur){
    Cellule * p = new Cellule;
    p->donnee = valeur;
    p->suivant = L->debut;
    p->precedent = NULL;
    L->debut->precedent = p;
    L->debut = p;
}

/**
 * Fonction pour supprimer au début
 */
void supprimer_debut (Liste * L){
    if (L->debut != NULL){
        Cellule * p = L->debut;
        L->debut = L->debut->suivant;
        delete p;
        L->debut->precedent = NULL;
    }
}



/**
 * Fonction pour ajouter une valeur au fin
 */

void ajouter_fin (Liste * L, TypeListe valeur){
    Cellule * p = new Cellule;
    p->donnee = valeur;
    p->precedent = L->fin;
    p->suivant = NULL;
    L->fin->suivant = p;
    L->fin = p;
}
```

```
//        ----------fonction pour supprimer a la fin de la liste--------------
-

void supprimer_fin(Liste *L){
    if(L->fin != NULL){
        Cellule * p = L->fin;
        L->fin = L->fin->precedent;
        delete p;
        L->fin->suivant = NULL;
    }
}


//  ------------------------fin de la  Partie 1. Gestion d'une liste
doublement chain?e----------------------------




//------------------Debut de la partie 2: Polynome reel avec la liste
doublement---------------------


//                 ---------fonction initialiser un plynome------

Polynome * init_polynome (){
    Polynome * p = new Polynome;

    p->debut = NULL;
    p->fin = NULL;
    return p;
}


//           ------------fonction lire()-------------

Polynome *lire(){
    int exp;
    double coef;
    Polynome* p = init_polynome();
    Cellule * t = p->debut;

    cout<<"Donner le couple coefficients et exposants (0 ou expposant negatif
pour arreter ): ";
    while(1){
        cin>>coef >>exp;
        if (coef==0 || exp<0){
```

```
            break;
        }

        Monome m = {coef, exp};
        Cellule * b = new Cellule;
        b->donnee = m;
        b->precedent = t;
        b->suivant = NULL;
        if (t != NULL) {
           t->suivant = b;
        } else {
            p->debut = b;
        }
            t = b;
        }

    p->fin = t ;
    return p;
}

//             --------------------fonction affichage--------------------------
------

void afficher(Polynome *p) {
    Cellule *c = p->debut;
    for (c = p->debut; c != NULL; c = c->suivant) {
        // if(c->donnee.coefficient >= 0) {
        cout << c->donnee.coefficient << "x^" << c->donnee.exposant<<" + " ;

    }
   cout<<" 0 "<<endl;

}

//           ---------la fonction degre()--------------

int degre(Polynome *p){
    cellule *c = p->fin;
    int M=0;
    Monome m;

    while(c!=NULL){
        m=c->donnee;
        if(m.exposant> M){
            M=m.exposant;
        }
    c=c->precedent;
}
return M;
```

```
}


void ajouter_monome(Polynome *p, double coef, int exp) {
    Cellule *n = new Cellule;
    n->donnee.coefficient = coef;
    n->donnee.exposant = exp;
    if (p->debut == NULL) {
        p->debut = n;
        p->fin = n;
        n->suivant = NULL;
        n->precedent = NULL;
    }
    else {
        Cellule *a = p->debut;
        while (a != NULL && a->donnee.exposant > exp){
            a = a->suivant;
        }
        if (a == NULL){
            n->suivant = NULL;
            n->precedent = p->fin;
            p->fin->suivant = n;
            p->fin = n;
        }
        else if (a->donnee.exposant < exp) {
            n->suivant = a;
            n->precedent = a->precedent;
            a->precedent = n;
            if (n->precedent == NULL) {
                p->debut = n;
            } else {
                n->precedent->suivant = n;
            }
        }
        else {  //
            if (a->donnee.coefficient + coef != 0) {
                a->donnee.coefficient += coef;
                delete n;
            }
            else {
                a->precedent->suivant = a->suivant;
                if (a->suivant != NULL) {
                    a->suivant->precedent = a->precedent;
                } else {
                    p->fin = a->precedent;
                }
                delete a;
                delete n;
            }
```

```
        }
    }
}


//----------------La fonction evaluer polynome----------------


float evaluer(Polynome *p, float x) {
    float resultat = 0.0;
    Cellule *cellule = p->debut;
    while (cellule != NULL) {
        Monome monome = cellule->donnee;
        resultat += monome.coefficient * pow(x, monome.exposant);
        cellule = cellule->suivant;
    }
    return resultat;
}

//----------la fonction evaluuer_horner--------------------------

double evaluer_horner(Polynome *p, double x){
    double resultat = 0.0;
    Cellule * cellule = p->debut;
    resultat = 0.0;
    cellule = p->fin;
    while (cellule!= NULL) {
        resultat = cellule->donnee.coefficient + (x * resultat);
        cellule = cellule->precedent;
    }
    return resultat;
}


//------la fonction supprimer_monomes------

void supprimer_monome(Polynome *p, int exp) {
    Cellule *n = p->debut;
    while (n != NULL) {
        if (n->donnee.exposant == exp) {
            if (n == p->debut) {
                p->debut = n->suivant;
                if (p->debut != NULL) {
                    p->debut->precedent = NULL;
                }
            } else {
                n->precedent->suivant = n->suivant;
                if (n->suivant != NULL) {
                    n->suivant->precedent = n->precedent;
```

```
                }
            }
            delete n;
            break;
        }
        n = n->suivant;
    }
    cout<<"Le polynome apres suppression du monome devient: ";
}


//---------La fonction somme de deux polynome------------

Polynome* somme(Polynome* poly1, Polynome* poly2) {
    Polynome* somme = init_polynome();
    Cellule* cel1 = poly1->debut;
    Cellule* cel2 = poly2->debut;
    while (cel1 != NULL || cel2 != NULL) {
        if (cel1 == NULL) {
            ajouter_monome(somme, cel2->donnee.coefficient, cel2-
>donnee.exposant);
            cel2 = cel2->suivant;
        } else if (cel2 == NULL) {
            ajouter_monome(somme, cel1->donnee.coefficient, cel1-
>donnee.exposant);
            cel1 = cel1->suivant;
        } else if (cel1->donnee.exposant > cel2->donnee.exposant) {
            ajouter_monome(somme, cel1->donnee.coefficient, cel1-
>donnee.exposant);
            cel1 = cel1->suivant;
        } else if (cel1->donnee.exposant < cel2->donnee.exposant) {
            ajouter_monome(somme, cel2->donnee.coefficient, cel2-
>donnee.exposant);
            cel2 = cel2->suivant;
        } else {
            ajouter_monome(somme, cel1->donnee.coefficient + cel2-
>donnee.coefficient, cel1->donnee.exposant);
            cel1 = cel1->suivant;
            cel2 = cel2->suivant;
        }
    }
    return somme;
}

//----------------fonction opposer()-----------

Polynome* oppose(Polynome* p) {
    Cellule* temp = p->debut;
    while (temp != NULL) {
```

```
        temp->donnee.coefficient = -temp->donnee.coefficient;
        temp = temp->suivant;
    }
    return p;
}


//---------La fonction produits------------

Polynome* produit(Polynome* p1, Polynome* p2) {
    Polynome* resultat = init_polynome();
    Cellule* temp1 = p1->debut;
    while (temp1 != NULL) {
        Cellule* temp2 = p2->debut;
        while (temp2 != NULL) {
            float coeff_resultat = temp1->donnee.coefficient * temp2-
>donnee.coefficient;
            int degre_resultat = temp1->donnee.exposant + temp2-
>donnee.exposant;
            ajouter_monome(resultat, coeff_resultat, degre_resultat);
            temp2 = temp2->suivant;
        }
        temp1 = temp1->suivant;
    }
    return resultat;
}



//------------------------------la fonction soustraction--------------------
-------------------------

Polynome* soustraction(Polynome* poly1, Polynome* poly2) {
    Polynome* p = somme(poly1, oppose(poly2));
    return p;

}
//-----------------la fonction supprimer poly---------

void supprimer_polynome(Polynome* p) {
    Cellule* n = p->debut;
    while (n != NULL) {
        Cellule* temp = n;
        n = n->suivant;
        delete temp;
    }
    delete p;
}
```

```
//---------La fonction division eclud------------


Polynome * division_euclidienne(Polynome * A, Polynome * B) {
    Polynome * quotient = init_liste();
    Polynome * reste = init_liste();
    ajouter_debut(reste, A->debut->donnee);

    while(reste->debut != NULL && reste->debut->donnee.exposant >= B->debut-
>donnee.exposant) {
        double coeff_quotient = reste->debut->donnee.coefficient / B->debut-
>donnee.coefficient;
        int exp_quotient = reste->debut->donnee.exposant - B->debut-
>donnee.exposant;
        Monome monome_quotient = {coeff_quotient, exp_quotient};
        ajouter_debut(quotient, monome_quotient);

        Polynome * mult = init_liste();
        for (Cellule * p = B->debut->suivant; p != NULL; p = p->suivant) {
            double coeff = -1 * coeff_quotient * p->donnee.coefficient;
            int exp = exp_quotient + p->donnee.exposant;
            Monome monome = {coeff, exp};
            ajouter_debut(mult, monome);
        }

        Polynome * nouveau_reste = init_liste();
        Polynome *S1= somme(reste,mult);
        somme(S1, nouveau_reste);
        supprimer_polynome(reste);
        reste = nouveau_reste;
        supprimer_polynome(mult);
    }

    supprimer_polynome(reste);
    return quotient;
}



//////////////  Fonction principale Main

int main(){

//creation et initialisation
    Polynome *p;
    cout << "Voulez-vous créer un nouveau Polynome [y/n] : ";
    char reponse;
    cin >> reponse;
    if(reponse == 'y') {
```

```
        p = lire();
        cout << "Le Polynome est : ";
        afficher(p);
    }
//ajouter

    Polynome *pa;
        cout << "Voulez-vous ajouter un nouveau monome au ploynome P(x) [y/n]
: ";
        char ra;
        int coef,expo;
        cin >> ra;
        if(ra == 'y') {
            cout<<"Donner le coefficient: "<<endl;
            cin>>coef;
            cout<<"Donner le exponente: "<<endl;
            cin>>expo;
            ajouter_monome(p, coef, expo);
            cout << "Le Polynome apres l'ajout est : ";
            afficher(p);
        }

//supprimer
    cout << "Voulez-vous suuprimer un monome [y/n] : ";
    char re;
    cin >> re;
    if(re == 'y') {
        int exp;
        cout<<"Veuillez donner le dgre du monome a supprimer:";
        cin>>exp;
        supprimer_monome(p,exp);
        cout << "Le Polynome deveient : ";
        afficher(p);
    }

//sommer
 Polynome *p1,*p2,*sum;
    cout << "Voulez-vous sommer deux Polynomes [y/n] : ";
    char rep;
    cin >> rep;
    if(rep == 'y') {
        p1 = lire();
        p2 = lire();
        cout << "Le Polynome est P1(x)= ";
        afficher(p1);
        cout << "Le Polynome est P2(x)= ";
        afficher(p2);

        sum =somme(p1,p2);
```

```
        cout << "La somme de P1 et P2 est : ";
        afficher(sum);
    }

//soustraction
    Polynome *p3,*p4,*soust;
    cout << "Voulez-vous soustraire deux Polynomes [y/n] : ";
    char c;
    cin >> c;
    if(c == 'y'){
        p3 = lire();
        p4 = lire();
        cout << "Le Polynome est P3(X)= ";
        afficher(p3);
        cout << "Le Polynome est P4(X)= ";
        afficher(p4);

        soust =soustraction(p4,p3);
        cout << "La difference de P1 et P2 est : ";
        afficher(soust);
        cout<<"Le degree de P3est: ";
        cout << degre(p3) << endl;
        cout<<"Le degree de P3est: ";
        cout << degre(p4) << endl;
    }

   //opposer

    Polynome *p7,*o_p;
    cout << "Voulez-vous chercher l'opposer d'un Polynomes [y/n] : ";
    char co;
    cin >> co;
    if(co == 'y'){
        p7 = lire();

        cout << "Le Polynome est P7(X)= ";
        afficher(p7);
        o_p = oppose(p7);
        cout << "L'oppose de P7  est : ";
        afficher(o_p);

    }

//produit
    Polynome *p8,*p9,*prd;
    cout << "Voulez-vous multiplier deux Polynomes [y/n] : ";
    char cp;
    cin >> cp;
    if(cp == 'y'){
```

```
        p8 = lire();
        p9 = lire();
        cout << "Le Polynome est P8(X)= ";
        afficher(p8);
        cout << "Le Polynome est P9(X)= ";
        afficher(p9);
        prd = produit(p8,p9);
        cout << "Le produit de P8 et P9  est : ";
        afficher(prd);

    }

//evaluation
    Polynome *pv;
    float val;
    float x;
    cout << "Voulez-vous evaluer un polynome [y/n] : ";
    char cv;
    cin >> cv;
    if(cv == 'y'){
        pv = lire();
        cout << "Le Polynome est Pv(X)= ";
        afficher(pv);
        cout<<"Donner la valeur de X: ";
        cin>>x;
        val = evaluer(pv,x);
        cout << "La valeur de Pv est : ";
        afficher(pv);

    }

            delete p;
            delete p1;
            delete p2;
            delete p3;
            delete p4;
            delete p7;
            delete p8;
            delete p9;
            delete prd;
            delete pv;


    return 0;
}
```