

algorithme avancé II

Le traitement des polynômes

à coefficients entiers

$$\begin{array}{r} \underline{3x^2 - 3x^4 - 5 + 2x} + \underline{2x^2 - x} \\ \hline 5x^2 - 3x^4 - 5 + x \end{array}$$

Realise par LAAFIA AICHA N 53

Encadre par Mr.Dargham

Ensa khouribga 2019-2020

Remerciement :

Avant d'entamer ce rapport, on tient à remercier dieu dans un premier temps, puis nos parents qui n'ont jamais cessé de nous encourager et de nous porter le soutien nécessaire tout au long de nos études. Nous saisissons également cette occasion pour exprimer nos sincères remerciements et notre extrême gratitude envers toutes les personnes ayant contribué, de près ou de loin, à ce projet. Nous souhaitons tout d'abord remercier Mr. DARGHAM pour nous avoir encadré tout au long des étapes qui nous ont permis d'aboutir au final à la réalisation de notre projet : elle a toujours été disponible, à l'écoute, elle a répondu à nombreuses de nos questions et s'est intéressée à l'avancement de nos travaux. Son soutien nous a été d'une grande aide et nous a encouragé à mener ce projet à terme.

Sommaire :

1. Introduction
2. PARTIE THEORIQUE : les algorithmes.
3. PARTIE PRATIQUE : les Codes en C.
4. CONCLUSION.

Introduction

L'objectif de ce projet est manipulation des polynômes c'est à dire Réaliser un module pour le traitement des polynômes à coefficients entiers en utilisant les listes chaînées. Le module doit fournir les opérations suivantes : créer un polynôme, afficher un polynôme sous le format usuel, calculer la somme, la différence et le produit de deux polynômes, calculer l'opposé et la puissance n -ème d'un polynôme, calculer la division euclidienne (quotient et reste) d'un polynôme par un autre, calculer la dérivée d'un polynôme, calculer une primitive d'un polynôme, calculer le plus grand diviseur commun de deux polynômes, calculer la valeur d'un polynôme en un entier x , tester si un entier x est un zéro d'un polynôme.

PARTIE THEORIQUE : les algorithmes

Il a d'abord fallu créer les structures de données utiles à la manipulation des polynômes :

J'ai commencé par créer la structure de donnée d'un monôme qui est composé d'un entier représentant le degré et d'un float représentant le coefficient.

```
type Structure mon :  
    degre : Entier;  
    coef  : float;  
fin Structure mon;  
  
type Structure mon * monome;
```

J'ai par la suite créer la deuxième structure de donnée celle du polynôme : qui est composé de la valeur d'un monôme et d'un pointeur suivant :

```
type Structure elem :  
    valeur : monome;  
    suivant : Structure ↑elem;  
fin Structure poly;
```

J'ai ensuite créer les différents sous-programmes demandés dans le sujet :

- `creerMonome` : Réel x Entier \rightarrow Monome

Crée un monôme à partir son coefficient et son degré,

```
Fonction      creerMonome(r : float, e : Entier ) : monome
Var    temp : monome;
Debut
    allouer(temp);
    temp.degre := e;
    temp.coef := r;
    Retourner (temp);
fin
```

- `creerPolynome` : \rightarrow Polynome

Crée un polynôme vide

```
Fonction      creerPolynome() : polynome
Debut
    Retourner(NULL);
fin
```

-
- ajouterMonome : Polynome x Monome \rightarrow Polynome

Ajoute un monôme à un polynôme.

```
Fonction      ajouterMonome(p : polynome, m : monome) : polynome
Var          temp : polynome; temp2 : polynome; p1 : polynome;
Debut
    p1 := p;
    allouer(temp);
    temp.valeur := m;
    temp.suivant := NULL;
    Si (p = NULL) Alors
        p := temp;
    Sinon
        Tant que ((p1.suivant != NULL) AND ((p1.valeur.degre)>(m.degre))) Faire
            p1 := p1.suivant;
        FinTQ
        Si (p1.valeur.degre = m.degre) Alors
            p1.valeur.coef := (m.coef) + (p1.valeur.coef);
        Sinon
            Si ((p1.valeur.degre)<(m.degre)) Alors
                allouer(temp2);
                temp2.valeur := p1.valeur;
                temp2.suivant := p1.suivant;
                p1.valeur := m;
                p1.suivant := temp2;
            Sinon
                p1.suivant := temp;
            finSi
        finSi
    Retourner(p);
fin
```

- $\text{supprimerMonome} : \text{Polynome} \times \text{Entier} \rightarrow \text{Polynome}$

Supprime dans un polynôme, le monôme de degré donné.

```
Fonction      supprimerMonome(p : polynome,e : Entier ) : polynome
Var    p1,temp : polynome;
Debut
    /*On suppose le polynome non null*/
    p1 := p;
    Si (p != NULL) Alors
        Si (p.valeur.degre = e) Alors
            p := p.suivant;
            Liberer (p1);
        Sinon
            temp := p.suivant;
            Tant que (temp!= NULL) Faire
                Si (temp.valeur.degre = e) Alors
                    p1.suivant := temp.suivant;
                    Liberer (temp);
                finSi
                p1 := temp;
                temp := temp.suivant;
            FinTQ
        finSi
    finSi
    Retourner (p);
fin
```

- Copier : copier des fonctions

```
Fonction      copier(p : polynome) : polynome
Var    p1 : polynome; m : monome;
Debut
    p1 := creerPolynome();
    Tant que (p) Faire
        m := 0;
        allouer(m) ;
        m.degre := (p.valeur.degre);
        m.coef := (p.valeur.coef);
        p1 := ajouterMonome(p1,m);
        p := p.suivant;
    FinTQ
    Retourner (p1);
fin
```


- additionner : Polynome x Polynome \rightarrow Polynome

Calcule la somme de deux polynômes

```

Fonction      additionner(p1 : polynome, p2 : polynome) : polynome
Var      p,q1,q2 : polynome;
         monome m;
Debut
    q1 := p1;
    q2 := p2;
    p := creerPolynome();
    Tant que ((q1) AND (q2)) Faire
        m := 0;
        allouer (m);
        Si (q1.valeur.degre = q2.valeur.degre) Alors
            m.coef := (q1.valeur.coef) + (q2.valeur.coef);
            m.degre := q1.valeur.degre;
            p := ajouterMonome(p,m);
            q1 := q1.suivant;
            q2 := q2.suivant;
        Sinon
            Si ((q1.valeur.degre) > (q2.valeur.degre)) Alors
                m.coef := (q1.valeur.coef);
                m.degre := q1.valeur.degre;
                p := ajouterMonome(p,m);
                q1 := q1.suivant;
            Sinon
                m.coef := (q2.valeur.coef);
                m.degre := q2.valeur.degre;
                p := ajouterMonome(p,m);
                q2 := q2.suivant;
            finSi
        finSi
    FinTQ
    Si (q1=NULL AND q2!= NULL) Alors
        Tant que (q2) Faire
            m := 0;
            allouer (m);
            m.coef:=(q2.valeur.coef);
            m.degre:=q2.valeur.egre;
            p:=ajouterMonome(p,m);
            q2:=q2.suivant;
        FinTQ
    Sinon
        Si (q1 != NULL AND q2=NULL) Alors
            Tant que (q1) Faire
                m:=0;
                allouer (m);
                m.coef :=(q1.valeur.coef);
                m.degre := q1.valeur.degre;
                p := ajouterMonome(p,m);
                q1 := q1.suivant;
            FinTQ
        finSi
    finSi
    Retourner (p) ;
fin

```

- multiplier : Polynome x Polynome \rightarrow Polynome

Calcule le produit de deux polynômes

```
Fonction      multiplier(p1 : polynome,p2 : polynome) : polynome
Var          p,q1,q2 : polynome;
            |
            | m : monome;
Debut
    q1 := copier(p1);
    q2 := copier(p2);
    p := creerPolynome();
    Tant que (p2) Faire
        m := 0;
        allouer (m);
        m.degre := (p1.valeur.degre)+(p2.valeur.degre);
        m.coef := (p1.valeur.coef)*(p2.valeur.coef);
        p := ajouterMonome(p,m);
        p2 := p2.suivant;
    FinTQ
    Tant que (q1) Faire
        q1 := q1.suivant;
        Tant que (q2) Faire
            m := 0;
            allouer (m);
            m.degre := (q1.valeur.degre)+(q2.valeur.degre);
            m.coef := (q1.valeur.coef)*(q2.valeur.coef);
            p := ajouterMonome(p,m);
            q2 := q2.suivant;
        FinTQ
    FinTQ
    Retourner (p);
fin
```

- $\text{mderiver} : \text{Monome} \times \text{Entier} \rightarrow \text{Monome}$

Calcule la k mdriver-ième dérivé d'un monôme, en donner une version récursive (mdriverR) et une autre itérative (mdriverI)

```
Fonction          mderiver (m : monome ,e : Entier) : monome
Var      d : Entier;
Debut
    d := m.degre;
    Si (m.degre != 0) Alors
        Tant que (m.degre != (d-e)) Faire
            m.coef := (m.coef)*(.degre);
            m.degre := (m.degre) - 1;
        FinTQ
        Si (m.degre <= 0 AND m.coef=0) Alors
            m.degre := 0;
            m.coef := 0;
        finSi
    finSi
    Retourner(m);
fin

Fonction          mdriverR (m : monome ,e : Entier) : monome
Debut
    Si (m.degre <=0 AND m.coef=0) Alors
        m.degre := 0;
        m.coef := 0;
    Sinon
        Si (e != 0) Alors
            m.coef := (m.coef)*(.degre);
            m.degre := (m.degre) - 1;
            mdriverR(m,e-1);
        finSi
    finSi
    Retourner (m);
fin
```

- pderive r: Polynome x Entier \rightarrow Polynome

Calcule la k ième dérivé d'un polynôme

```
Fonction          pderiver(p : polynome,e : Entier) : polynome
Var      p1 : polynome;
Debut
    p1 := p;
    Tant que (p1!=NULL) Faire
        |      p1.valeur := mderiver(p1.valeur,e);
        |      p1 := p1.suivant;
    FinTQ
    Retourner(p);
fin

Procédure          ecrireMI(m : monome)
Debut
    Afficher(m.coef,m.degre,"(%f,%d),");
fin

Procédure          ecrirePolynomeR(polynome p)
Debut
    Si (p.suivant=NULL) Alors
        |      ecrireMI(p.valeur);
    Sinon
        |      ecrireMI(p.valeur);
        |      ecrirePolynomeR(p.suivant);
    finSi
fin
```

-
- `ecrireMonome : Monome \rightarrow \emptyset`

Affiche un monôme, en donner une version itérative (`ecrireMI`)

```
Procedure                               ecrireMI(m : monome)
Debut
    Afficher(m.coef,m.degre,"(%f,%d),");
fin
```

- `ecrirePolynome : Polynome \rightarrow \emptyset`

Affiche un polynôme

```
Procedure                               ecrirePolynomeR(polynome p)
Debut
    Si (p.suivant=NULL) Alors
        |   ecrireMI(p.valeur);
    Sinon
        |   ecrireMI(p.valeur);
        |   ecrirePolynomeR(p.suivant);
    finSi
fin

Procedure                               ecrirePolynome(polynome p)
Var      p1 : polynome;
Debut
    p1 := p;
    Tant que (p1) Faire
        |   ecrireMI(p1.valeur);
        |   p1 := p1.suivant;
    FinTQ
    Afficher("\n");
fin
```

Vous pouvez ajouter un sous-programme permettant de créer un polynôme par interaction avec l'utilisateur (lecture du coefficient et du degré de chaque monôme) en utilisant les opérations `creerPolynome` et `ajouterMonome`.

```
Fonction          creationPolynome() : polynome
Var      m : monome;
        e : Entier;
        r : float;
        p : polynome;
Debut
    m := 0;
    e := 0;
    r;
    p;
    p := creerPolynome();
    Afficher("Veuillez entrer le degré du Monome de ce polynome : ");
    Afficher(e,"%d");
    Afficher("Veuillez entrer le coefficient du Monome de ce polynome : ");
    Ecrire(&r,"%f",);
    m.coef := r;
    m.degre := e;
    p := ajouterMonome(p,m);
    Retourner(p);
fin
```

PARTIE PRATIQUE

les Codes en C

Notre projet contient :

- **polynome.h,**
- **polynome.c,**
- **main.c,**

• Polynome.h

le fichier d'entête doit contenir les sous-programmes, les types, les constantes et les variables fournis par la bibliothèque,

```
1
2  /* * Fichier source décrivant les structures et les différentes fonctions des polynomes *
3  typedef struct mon{
4      int degre;
5      float coef;
6  }mon;
7  typedef mon*monome;
8
9  typedef struct elem{
10     monome valeur;
11     struct elem *suivant;
12 }poly;
13
14 typedef poly * polynome;
15
16
17 monome creerMonome(float r, int e);
18 polynome creerPolynome();
19 polynome ajouterMonome(polynome p, monome m);
20 polynome supprimerMonome(polynome p, int e);
21 polynome additionner(polynome p1, polynome p2);
22 polynome copier(polynome p);
23 polynome multiplier(polynome p1, polynome p2);
24 monome mderiver (monome m,int e);
25 monome mderiverR(monome m, int e);
26 polynome pderiver(polynome p, int e);
27 void ecrireMR(monome m);
28 void ecrireMI(monome m);
29 void ecrirePolynomeR(polynome p);
30 void ecrirePolynome(polynome p);
31 polynome creationPolynome();
32
```


● Polynome.c

le fichier source doit contenir le corps des différents sous-programmes proposés

```
2 //le fichier source doit contenir le corps des différents sous-programmes proposés
3 #include "polynome.h"
4 #include <stdio.h>
5 #include <stdlib.h>
6
7
8 monome creerMonome(float r, int e){
9     // Crée un monôme à partir son coefficient et son degré,
10    monome temp;
11    temp=(monome*)malloc(sizeof(mon));
12    temp->degre=e;
13    temp->coef=r;
14    return temp;
15 }
16
17
18
19 polynome creerPolynome(){
20     // Crée un polynôme vide
21     return NULL;
22 }
23
24
25 polynome ajouterMonome(polynome p, monome m){
26     /*ce programme ajoute un monome dans un polynome tout en le gardant trié*/
27     polynome temp;
28     polynome temp2;
29     polynome p1=p;
30     temp=(polynome*)malloc(sizeof(polynome));
31     temp->valeur=m;
32     temp->suivant=NULL;
33     if (p==NULL)
34         p=temp;
35     else{
36         while ((p1->suivant !=NULL) && ((p1->valeur->degre)>(m->degre))){
37             p1=p1->suivant;
38         }
39         if (p1->valeur->degre==m->degre)
40             p1->valeur->coef=(m->coef)+(p1->valeur->coef);
41         else{
42             if((p1->valeur->degre)<(m->degre)){
43                 temp2=(polynome*)malloc(sizeof(polynome));
44                 temp2->valeur=p1->valeur;
45                 temp2->suivant=p1->suivant;
46                 p1->valeur=m;
47                 p1->suivant=temp2;
48             }
49             else
50                 p1->suivant=temp;
51         }
52     }
53     return p;
54 }
55 }
```

```
57  polynome supprimerMonome(polynome p, int e){
58      /*On suppose le polynome non null*/
59      polynome p1=p;
60      polynome temp;
61      if (p!=NULL){
62          if(p->valeur->degre==e){
63              p=p->suivant;
64              free(p1);
65          }
66          else{
67              temp=p->suivant;
68              while(temp!=NULL){
69                  if (temp->valeur->degre==e){
70                      p1->suivant=temp->suivant;
71                      free(temp);
72                  }
73                  p1=temp;
74                  temp=temp->suivant;
75              }
76          }
77      }
78      return p;
79  }
80
```

```

83
84 polynome additionner(polynome p1, polynome p2){
85     //Calcule la somme de deux polynômes
86     polynome p;
87     polynome q1=p1;
88     polynome q2=p2;
89     p=creerPolynome();
90     while((q1 && (q2))){
91         monome m=0;
92         m=(mon*)malloc(sizeof(mon));
93         /*Utilisation des monomes pour ensuite les ajouter dans le polynome à l'aide de la fonction ajouterpolynome(polynome p, int e)*/
94         if (q1->valeur->degre==q2->valeur->degre){
95             m->coef=(q1->valeur->coef) + (q2->valeur->coef);
96             m->degre=q1->valeur->degre;
97             p=ajouterMonome(p,m);
98             q1=q1->suivant;
99             q2=q2->suivant;
100         }
101         else{
102             if ((q1->valeur->degre) > (q2->valeur->degre)){
103                 m->coef=(q1->valeur->coef);
104                 m->degre=q1->valeur->degre;
105                 p=ajouterMonome(p,m);
106                 q1=q1->suivant;
107             }
108             else{
109                 m->coef=(q2->valeur->coef);
110                 m->degre=q2->valeur->degre;
111                 p=ajouterMonome(p,m);
112                 q2=q2->suivant;
113             }
114         }
115     }
116     if (q1==NULL && q2!=NULL){
117         while (q2){
118             monome m=0;
119             m=(mon*)malloc(sizeof(mon));
120             m->coef=(q2->valeur->coef);
121             m->degre=q2->valeur->degre;
122             p=ajouterMonome(p,m);
123             q2=q2->suivant;
124         }
125     }
126     else{
127         if (q1!=NULL && q2==NULL){
128             while (q1){
129                 monome m=0;
130                 m=(mon*)malloc(sizeof(mon));
131                 m->coef=(q1->valeur->coef);
132                 m->degre=q1->valeur->degre;
133                 p=ajouterMonome(p,m);
134                 q1=q1->suivant;
135             }
136         }
137     }
138     return p;
139 }
140

```

```

polynome copier(polynome p){
    // fonction auxiliaire qui nous permet de copier une polynome 'p' ,pour ne pas repeter le meme code dans plusieurs fonctions
    polynome p1;
    p1=creerPolynome();
    while(p){
        monome m=0;
        m=(monome*)malloc(sizeof(mon));
        m->degre=(p->valeur->degre);
        m->coef=(p->valeur->coef);
        p1=ajouterMonome(p1,m);
        p=p->suitant;
    }
    return p1;
}

polynome multiplier(polynome p1, polynome p2){
    // Calcule le produit de deux polynômes
    polynome p;
    polynome q1;
    polynome q2;
    q1=copier(p1);
    q2=copier(p2);
    p=creerPolynome();
    while (p2){ /*Cette boucle sert à multiplier le premier Monome de p1 avec chacun des monomes de p2*/
        monome m=0;
        m=(monome*)malloc(sizeof(mon));
        m->degre=(p1->valeur->degre)+(p2->valeur->degre);
        m->coef=(p1->valeur->coef)*(p2->valeur->coef);
        p=ajouterMonome(p,m);
        p2=p2->suitant;
    }
    while (q1){ /*ici utilisation d'une copie de p1 dans q1 */
        /* Cela sert à mutliplier tous les autres Monomes de p1 (sauf le premier) avec chacun des monomes de p2*/
        q1=q1->suitant; /*Passage directement au deuxième élément de p1 car le 1er a déjà été fait dans la 1ère boucle*/
        while (q2){ /*Utilisation d'une copie de p1 et p2 car sinon cela ne fonctionne pas*/
            monome m=0;
            m=(monome*)malloc(sizeof(mon));
            m->degre=(q1->valeur->degre)+(q2->valeur->degre);
            m->coef=(q1->valeur->coef)*(q2->valeur->coef);
            p=ajouterMonome(p,m);
            q2=q2->suitant;
        }
    }
    return p;
}

```

```

monome mderiver (monome m,int e){
    //fonction itérative auxiliaire pour ne pas repeter le meme code dans plusieurs fonctions de dérivation
    //quand degre=0? ->supprimer le monome ??
    int d;
    d=m->degre;
    if (m->degre!=0){
        while (m->degre!=(d-e)){
            m->coef=(m->coef)*(m->degre);
            m->degre=(m->degre)-1;
        }
        if (m->degre<=0 && m->coef==0){
            m->degre=0;
            m->coef=0;
        }
    }
    return m;
}

```

```

monome mderiverR(monome m, int e){
    //Pourquoi pas , une version récursive de mderiver
    if (m->degre<=0 && m->coef==0){
        m->degre=0;
        m->coef=0;
    }
    else{
        if (e!=0){
            m->coef=(m->coef)*(m->degre);
            m->degre=(m->degre)-1;
            mderiverR(m,e-1);
        }
    }
    return m;
}

```

```

polynome pderiver(polynome p, int e){
    //Calcule la k ième dérivé d'un polynôme
    polynome p1=p;
    while (p1!=NULL){
        p1->valeur=mdriver(p1->valeur,e);
        p1=p1->suivant;
    }
    return p;
}

```

```

void ecrireMI(monome m){
    //afficher un monome
    printf("(%.f,%d)",m->coef,m->degre);
}

void ecrirePolynomeR(polynome p){
    //afficher une polynome "la version récursive"
    if (p->suivant==NULL)
        ecrireMI(p->valeur);
    else{
        ecrireMI(p->valeur);
        ecrirePolynomeR(p->suivant);
    }
}

void ecrirePolynome(polynome p){
    // //afficher une polynome "la version itérative"
    polynome p1=p;
    while(p1){
        ecrireMI(p1->valeur);
        p1=p1->suivant;
    }
    printf("\n");
}

polynome creationPolynome(){
    /*Vous pouvez ajouter un sous-programme permettant de
    créer un polynôme par interaction avec l'utilisateur (lecture du coefficient et du degré de chaque monôme)
    en utilisant les opérations creerPolynome et ajouterMonome.*/
    monome m=0;
    int e=0;
    float r;
    polynome p;
    p=creerPolynome();
    printf("Veuillez entrer le degré du Monome de ce polynome : ");
    printf("%d",e);
    printf("Veuillez entrer le coefficient du Monome de ce polynome : ");
    scanf("%.f",&r);
    m->coef=r;
    m->degre=e;
    p=ajouterMonome(p,m);
    return p;
}

```


• main.c, le programme principal

```
1  #include "polynome.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main(){
6
7      monome m;
8      polynome p;
9
10     /*test du sous-programme creerMonome qui retourne un monome*/
11     float r;
12     int e;
13     printf("\n Test du sous-programme creerMonome \n\n");
14     printf("Veuillez entrer le coefficient du Monome : ");
15     scanf("%f",&r);
16     printf("Veuillez entrer le degré du Monome : ");
17     scanf("%d",&e);
18     m=creerMonome(r,e);
19     ecrireMI(m);
20     printf("\n");
21     /*Permet d'afficher le monome créer et en même temps de tester la fonction d'affichage d'un monome*/
22
23     /*Le sous-programme CreerPolynome retourne un polynome NULL*/
24     printf("\n Test du sous-programme creerPolynome \n\n");
25     p=creerPolynome();
26     printf("Le polynome NULL a été créé \n");
27
28
29     /*Le prochain test : sous-programme ajouterMonome qui permet l'ajout d'un monome dans un polynome*/
30     printf("\n Test du sous-programme ajouterMonome \n\n");
31     p=ajouterMonome(p,m);
32     ecrirePolynome(p);
33     /*ajout du monome m dans le polynome p*/
34
35
36     /*Le prochain sous-programme tester est supprimerMonome*/
37     printf("\n Test du sous-programme supprimerMonome \n\n");
38     p=supprimerMonome(p,e);
39     /*ici on supprime le monome créer dans le premier sous-programme CreerMonome et ajouter dans le polynome p juste avant*/
40     ecrirePolynome(p);
41
42     /*Test du sous-programme additionner qui additionne 2 polynomes ensemble*/
43     printf("\n Test du sous-programme additionner \n\n");
44
45     return 0;
46 }
```

CONCLUSION

Ce Mini-projet nous a permis de mettre en évidence les connaissances que nous avons apprises au cours du «Module C» et aussi de savoir l'utilité des méthodes analytique ainsi que la programmation pour résoudre des problèmes et aussi faire face à la pression et à des situations proches de celles que nous pourrions rencontrer dans le milieu professionnel. De plus, le fait de mener un projet en partant de zéros était une expérience enrichissante et

motivante pour nous.