Ecole Normale Supérieure de l'Enseignement Technique
Département mathématique Informatique
Filière : Ingénierie Informatique, Big Data & Cloud Computing 1

Compte Rendu de

TD1/TP1 – Algorithmique et Programmation en Langage C

Réalisé par :
ADIB Amine

Année universitaire 2019/2020

Partie 1 – Manipulations de base sur les polynômes

Explications des Choix

Pour stocker des données en mémoire, j'ai utilisé des listes chaînées, les listes chaînées représentent une façon d'organiser les données en mémoire de manière beaucoup plus flexible, Elles sont aussi très pratiques pour réarranger les données.

Chaque élément de la liste est un polynôme, il aura la forme de la structure suivante :

```
typedef struct Polynome
{
    int deg;
    float cof;
    struct Polynome *next;
}*Polynome;
```

La Saisie des données d'un polynôme

Créer un polynôme composée de plusieurs éléments de type polynôme

```
int main()
{
    Polynome p1;
    p1 = Cree();
```

J'ai déclaré une variable 'start' de type polynôme, l'utilisateur doit saisir le degré mono et après le coefficient, s'il n'y a plus d'autre élément à ajouter dans le un polynôme l'utilisateur va saisir -1.

La fonction AjouterDebut() prend en paramètre l'élément start(qui contient l'adresse du premier élément), le dégrée et le coefficient à stocker dans le nouvel élément que l'on va créer.

Pour simplifier ,j'ai insérer l'élément en début de liste. Pour mettre à jour correctement les pointeurs, j'ai fait les étapes suivant :

- Faire pointer le nouvel élément vers son futur successeur, qui est l'actuel premier élément de la liste.
- Faire pointer le pointeur premier vers nouvel élément.

Pour mettre à jour correctement les éléments de polynôme j'ai fait un tri, ce qui génère un polynôme, où l'élément suivant est plus grand que le précédent

```
Polynome Cree(){
    Polynome start = NULL;
    int d;
    float c;
    while(1){
        do
        {   printf("degree du monome courant (-1 pour finir) : ");
            scanf("%d",&d);
        }while (d < -1);
        if (d == -1)
            break;
        printf("coeff de x^%d : ",d);
        scanf("%f",&c);
        AjouterDebut(&start, d,c);
    }
    Sort(start);
    return start;
}
```

```
void AjouterDebut(Polynome *start_ref, int deg,float cof){
    Polynome ptr1 = (Polynome)malloc(sizeof(struct Polynome));
    ptr1->deg = deg;
    ptr1->cof = cof;
    ptr1->next = *start_ref;
    *start_ref = ptr1;
}
```

```
void swap(Polynome a,Polynome b){
    int temp = a->deg;
    a->deg = b->deg;
    b->deg = temp;
    float tmp = a->cof;
    a->cof = b->cof;
    b->cof = tmp;
}
```

```
void Sort(Polynome start){
    int swapped, i;
    Polynome ptr1;
    Polynome lptr = NULL;

    if (start == NULL)
        return;

    do
    {
        swapped = 0;
        ptr1 = start;

        while (ptr1->next != lptr)
        {
            if (ptr1->deg > ptr1->next->deg)
            {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    }
    while (swapped);
}
```

Afficher un polynôme

Pour bien visualiser ce que contient le polynôme, j'ai utilisé la fonction toString(), qui va retourner le polynôme sous forme de chaîne de caractère

```
char * poly = toString(p1);
    printf("%s \n",poly);
```

```
char * toString(Polynome l){
    char *buffer = malloc(sizeof(char)*2000);
    char *pnt = buffer;
    Polynome p;
    p = l;
```

```
    if(p->deg == 0 ){
        pnt += sprintf(pnt, "%6.2f ",p->cof);
        p = p->next;
    }else{
        pnt += sprintf(pnt, "%6.2f*x**%d ",p->cof,p->deg);
        p = p->next;
    }
    while (p)
    {
        if (p->cof > 0)
            pnt += sprintf(pnt, "+%6.2f*x**%d ",p->cof,p->deg);
        else
            pnt += sprintf(pnt, "%6.2f*x**%d ",p->cof,p->deg);
        p = p->next;
    }
    return buffer;
}
```

Evaluer la valeur du polynôme étant donnée la valeur de la variable x

```
double eval(double x,Polynome p){
    double r = 0;
    while (p)
    {
        r += pow(x,p->deg) * p->cof;
        p = p->next;
    }
    return r;
}
```

Changer le signe d'un polynôme en changeant le signe de tous ses coefficients

```
void changSign(Polynome a){
    while (a)
    {
        a->cof = -(a->cof);
        a = a->next;
    }
}
```

Faire l’addition de deux polynômes P et Q fournis en paramètres

```
Polynome add(Polynome a,Polynome b){
    Polynome start = NULL;

    while (a && b)
    {
        if (a->deg == b->deg)
        {
            AjouterDebut(&start, a->deg,a->cof + b->cof);
            a = a->next;
            b = b->next;
        }
        else if (a->deg > b->deg)
        {
            AjouterDebut(&start,b->deg,b->cof);
            b = b->next;
        }else{
            AjouterDebut(&start,a->deg,a->cof);
            a = a->next;
        }
    }
    if (a!=NULL)
    {
        while (a)
        {
            AjouterDebut(&start,a->deg,a->cof);
            a = a->next;
        }

    }if(b!=NULL)
    {
        while (b)
        {
            AjouterDebut(&start,b->deg,b->cof);
            b = b->next;
        }
    }
    Sort(start);
    return  start;
}
```

Faire la soustraction de deux polynômes

```
Polynome sub(Polynome a,Polynome b){
    Polynome start = NULL;

    while (a && b)
    {
        if (a->deg == b->deg)
        {
            if ( (a->cof - b->cof) != 0)
                AjouterDebut(&start, a->deg,a->cof - b->cof);
            a = a->next;
            b = b->next;
        }
        else if (a->deg > b->deg)
        {
            AjouterDebut(&start,b->deg,-(b->cof));
            b = b->next;
        }else{
            AjouterDebut(&start,a->deg,a->cof);
            a = a->next;
        }
    }
    if (a!=NULL)
    {
        while (a)
        {
            AjouterDebut(&start,a->deg,a->cof);
            a = a->next;
        }

    }if(b!=NULL)
    {
        while (b)
        {
            AjouterDebut(&start,b->deg,-(b->cof));
            b = b->next;
        }
    }
    Sort(start);
    return  start;
}
```

Faire la comparaison de deux polynômes

```
bool cmp(Polynome a,Polynome b) {
    while (a && b) {
        if (a->deg == b->deg) {
            if (a->cof == b->cof) {
                a = a->next;
                b = b->next;
            }
        }
    }
    if (b==NULL && a == NULL){
        return true;
    }
    else
        return false;
}
```

Calculer la dérivée d'un polynôme

```
Polynome drv(Polynome a){
    Polynome start = NULL;
    if (a->deg == 0)
        a = a->next;
    while (a){
        AjouterDebut(&start, a->deg - 1 ,a->cof * a->deg);
        a = a->next;
    }
    Sort(start);
    return start;
}
```

Tracer la courbe représentative du polynôme dans un intervalle I=[a b] en utilisant le grapheur Gnuplot

```
void trace(int nbarg,...){

    char buf[2000];
    char *pf=buf;
    va_list ap;
    va_start(ap,nbarg);
    pf += sprintf(pf,"gnuplot -p -e \"plot [0:10]");
    for (int i = 0; i < nbarg; i++)
    {
        Polynome p = va_arg(ap,Polynome);
        pf += sprintf(pf,"%s",toString(p));
        if (nbarg > i+1 )
            pf += sprintf(pf,",");
    }
    pf += sprintf(pf,";\"");
    va_end(ap);
    puts(buf);
    system(buf);
}
```

Partie 2 : Calcul numérique d'une racine d'un polynôme

```
void racin(int k,double x,Polynome p){
    Polynome pd;
    pd = drv(p);
    char * p1 = toString(p);
    char * p2 = toString(pd);
    printf("\n\n");
    printf("p(x) = %s\n",p1);
    printf("p'(x) = %s\n",p2);
    for (int i = 0; i <= k; i++)
    {
        printf("K=%d\tx=%.9f\tp(x)=%.9f\n",i,x,eval(x,p));
        x = x - (eval(x,p) / eval(x,pd));
    }
}
```

un exemple de calcul de racine par la méthode de Newton-Raphson

```
p(x) =   1.00 +  1.00*x**1 +  2.00*x**2  -2.00*x**3
p'(x) =   1.00 +  4.00*x**1  -6.00*x**2
K=0     x=1.000000000   p(x)=2.000000000
K=1     x=3.000000000   p(x)=-32.000000000
K=2     x=2.219512195   p(x)=-8.795693620
K=3     x=1.772561953   p(x)=-2.082180082
K=4     x=1.579079697   p(x)=-0.308782285
K=5     x=1.538687685   p(x)=-0.012062920
K=6     x=1.536976780   p(x)=-0.000021160

Process finished with exit code 0
```