RAKOTOVAHINY
Thomas

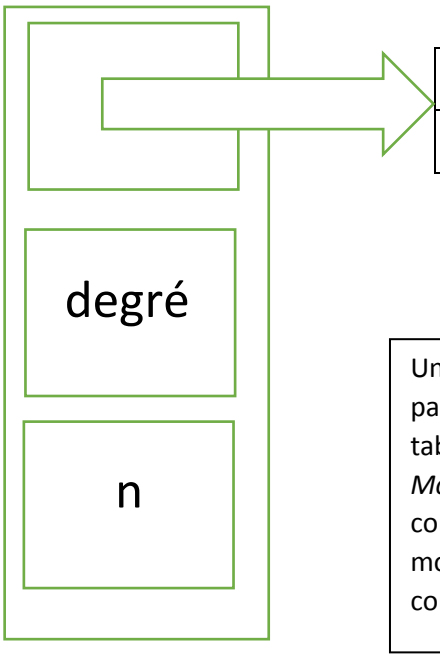
ANNICCHIARICO
Gianlucca

Projet Polynôme

Ce projet contient des fonctions permettant le stockage de polynômes en C et leurs traitements/calculs tels que la somme ou le produit.

Sommaire

I. Un polynôme en C



Coeff. 1	Coeff. 2
Degré	Degré

Etc.

Un polynôme en C est représenté par un pointeur qui pointe vers un tableau de monomes de type *Monome* qui est une structure contenant le coefficient d'un monôme et son degré correspondant.

II. La saisie et l'Affichage

Pour uniformiser le projet, tous les polynômes entrés par l'utilisateurs sont alloués dynamiquement.
Les tableaux de monômes sont aussi alloués dynamiquement dû au fait qu'on ne connaisse pas leur taille à l'avance.

La saisie commence par demander le *degré* du polynôme à l'utilisateur puis les coefficients de degré *degré* à *0*.
Une condition est insérée dans la boucle pour éviter le stockage d'un coefficient nul.

```
for (int i = p->degre; i >= 0; i--){
    printf("Saisir le coefficient du monome de degré %d : ", i);
    fscanf(stdin, "%d", &tmp);
    if (tmp != 0){
        monomes[n].coefficient = tmp;
        monomes[n].degre = i;
        n++;
    }
}
```

Ici la condition permet d'éviter l'insertion d'un monôme nul.

Après avoir vérifié que le polynôme n'est pas vide, on affiche dans l'ordre le degré du polynôme, le nombre de monômes puis l'ensemble des coefficients par degré croissant.

Par exemple : le polynôme $3x^3+7x^2+6$ sera affiché (3 3(6 0)(7 2)(3 3)).

```
void print_monome(const Monome *monomes, int n){
    for (int i = n - 1; i >= 0; i--){
        if (monomes[i].degre >= 0)
            fprintf(stdout, " (%d %d)", monomes[i].coefficient, monomes[i].degre);
    }
}
```

On insère aussi une condition pour éviter l'affichage de monôme nul.

III. La somme et le produit

Les fonctions *somme* et *produit* prennent en paramètre 2 polynômes afin d'effectuer l'opération correspondante.
Ils renvoient une adresse vers un polynôme alloué dynamiquement dans la fonction.

Pour la somme on effectue un parcours en parallèle des 2 polynômes et on compare les degrés des coefficients :

- Si le degré est égal on somme simplement les coefficients et on insère cette somme dans le tableau de monômes somme alloué dynamiquement
- Si le degré est différent on ajoute le monôme directement dans le tableau de monômes somme et on incrémente.

La condition d'arrêt de la boucle étant la taille des polynômes, il convient de rajouter les polynômes restants dans le polynôme somme si l'un est plus grand que l'autre puis on retourne l'adresse du polynôme somme.

Pour le produit on effectue simplement une *n* distributivité à l'aide d'une double boucle.

```
/** Calcul du produit */
    int k = 0; // parcours du polynôme produit
    for (int i = 0; i < p1->n ; i++){
        for (int j = 0; j < p2->n; j++){
            monomes[k].coefficient = p1->monomes[i].coefficient * p2->monomes[j].coefficient;
            monomes[k].degre = p1->monomes[i].degre + p2->monomes[j].degre;
            k++;
        }
    }
```

On obtient un polynôme qu'il convient de trier par degré décroissant à l'aide de la fonction *qsort* et à réduire à l'aide d'un boucle qui compare les coefficients 2 à 2 afin de comparer leur degré.

```
/** Réduction du polynôme obtenu */
    int tmp = p->n; // variable de réduction
    for (int i = 0; i < p->n ; i++){
        if (monomes[i].degre == monomes[i+1].degre){
            monomes[i].coefficient = monomes[i].coefficient + monomes[i+1].coefficient;
            for (int j = i; j < p->n; j++){
                monomes[j+1] = monomes[j+2];
            }
            tmp--;
        }
    }
```

On compare les monômes 2 à 2 pour sommer les coefficients de degré égaux puis "tirer" les monômes restants.
On décrémente ainsi la variable *tmp* qui deviendra la nouvelle taille du polynôme.

Note : La fonction de réallocation du polynôme faisant arrêter le programme a été mise en commentaire.

```
/*Monome *tmp_monomes = NULL;
printf("\n%d\n",tmp);
tmp_monomes = realloc(monomes, (tmp) * sizeof(Monome));
if (NULL == tmp_monomes){
    printf("Erreur réallocation\n");
    return NULL;
}else{
    monomes = tmp_monomes;
}*/
```

IV. La dérivée et la primitive

Les fonctions dérivée et primitive prennent en paramètres un pointeur de polynôme.

Pour la dérivée on applique la règle mathématique en multipliant le coefficient d'un monôme par son degré correspondant puis en réduisant le degré de 1.

```
for (int i = 0; i < p->n; i++){
    monomes[i].coefficient = (p->monomes[i].coefficient)*(p->monomes[i].degre);
    monomes[i].degre = p->monomes[i].degre - 1;
}
```

Pour la primitive il s'agit de diviser le coefficient par le degré + 1, puis ajouter 1 au degré.

```
for (int i = 0; i < p->n; i++){
    monomes[i].coefficient = (p->monomes[i].coefficient)/((p->monomes[i].degre) + 1);
    monomes[i].degre = p->monomes[i].degre + 1;
}
```

Ensuite on réutilise les fonctions de somme et produit pour calculer les dérivée d'un produit/somme de 2 polynômes.

```
Polynome * derivee_polynome_somme(const Polynome *p1, const Polynome *p2){
    return derivee_polynome(somme_polynome(p1,p2));
}

Polynome * derivee_polynome_produit(const Polynome *p1, const Polynome *p2){
    return derivee_polynome(produit_polynome(p1, p2));
}
```

V. Valeur d'un polynôme

Pour calculer la valeur d'un polynôme en un entier on applique la règle mathématique en faisant tourner une somme puis en retournant la valeur de cette somme.

```
int valeur_polynome(const Polynome *p, int x){
    int somme = 0;
    for (int i = 0; i < p->n; i++){
        somme = somme + p->monomes[i].coefficient * pow(x, p->monomes[i].degre);
    }
    return somme;
}
```

Pour conclure, ce projet nous a apporté beaucoup de connaissances et d’autonomie, notamment dans le respect du cahier des charges et de la programmation modulaire.

Les difficultés que nous avons rencontrées nous ont permis d’accroitre notre capacité à utiliser des structures ainsi que d'améliorer notre logique.