ENSA KHOURIBGA | ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

2023 - 2024

ALGORITHME AVANCEE

PROJET : CALCULATRICE DE POLYNOMES

REALISER PAR :

- FATIH MOHAMED-AMINE
- BOUCHTA OTHMANE

ENCADRER PAR :

- MR DARGHAM ABDELMAJID

SOMMAIRE

- INTRODUCTION
- OBJECTIF
- METHODE ET ANALYSE
- LE CODE ET L’EXPLICATION
- CONCLUSION

INTRODUCTION :

DANS LE CADRE DE NOTRE PREMIÈRE ANNÉE DU CYCLE D'INGÉNIEUR EN GÉNIE INFORMATIQUE À L'ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES DE KHOURIBGA, NOTRE MISSION CONSISTAIT À RÉALISER UN PROJET EN ALGORITHMIQUE AVANCÉE. L'OBJECTIF PRINCIPAL DE CE PROJET ÉTAIT D'IMPLÉMENTER UNE CALCULATRICE DE POLYNÔMES CAPABLE D'EFFECTUER LES OPÉRATIONS DE BASE SUR CES DERNIERS. NOUS AVONS PRIS EN CONSIDÉRATION QUE LES COEFFICIENTS DES POLYNÔMES POUVAIENT ÊTRE SOIT DES ENTIERS, SOIT DES FRACTIONS (À DÉFINIR ULTÉRIEUREMENT). DE PLUS, NOUS AVONS CHOISI DE REPRÉSENTER LES POLYNÔMES SOUS FORME DE LISTES CHAÎNÉES.

DANS CE RAPPORT, NOUS PRÉSENTERONS INITIALEMENT L'ANALYSE DU PROBLÈME AINSI QUE LES MÉTHODES UTILISÉES POUR RÉSOUDRE CHAQUE PARTIE. ENSUITE, NOUS FOURNIRONS UN GUIDE DÉTAILLÉ DU PROGRAMME ÉLABORÉ.

L’OBJECTIF

CE PROJET D'ALGORITHMIQUE AVANCÉE A POUR BUT : CALCULATRICE DE POLYNÔMES EN UTILISANT CE QUE NOUS AVONS APPRIS DANS DANS LE COURS POUR AIDER L'UTILISATEUR A CALCULER UN POLYNÔMES PAR UNE APPLICATION CONSOLE QUE OFFRE LES FONCTIONNALITÉS SUIVANTES :

- LA SAISIE ET L'AFFICHAGE
- L'ADDITION, LA SOUSTRACTION, LE PRODUIT ET LA PUISSANCE.
- L'AFFECTATION DE POLYNOMES
- LA D´ERIVATION ET L'INTEGRATION.
- L'´EVALUATION DE P(X) POUR UN REEL X DONN´E.

METHODE ET ANALYSE

CE PROJET VISE À MANIPULER ET EFFECTUER DES OPÉRATIONS SUR DES POLYNÔMES DONT LES COEFFICIENTS SONT DE TYPE ENTIER OU FRACTION. AFIN D'ASSURER UNE ORGANISATION CLAIRE ET LISIBLE DU CODE, NOUS AVONS CRÉÉ UN FICHIER SOURCE EN LANGAGE C++. DANS CE FICHIER, NOUS AVONS DÉFINI UNE STRUCTURE APPELÉE "POLYNÔME", REGROUPANT LES DONNÉES CARACTÉRISANT UN POLYNÔME TELLES QUE SON DEGRÉ, SES COEFFICIENTS, ET UN POINTEUR "NEXT" POINTANT VERS UN NOUVEAU MONÔME. POUR LE STOCKAGE DES DEGRÉS DES POLYNÔMES, NOUS UTILISONS UN TABLEAU DE TYPE INT.

LE FICHIER SOURCE COMPREND ÉGALEMENT LES PROTOTYPES DE TOUTES LES PROCÉDURES ET FONCTIONS NÉCESSAIRES TOUT AU LONG DU PROGRAMME. ENFIN, UNE FONCTION "MAIN" A ÉTÉ MISE EN PLACE POUR ÊTRE EXÉCUTÉE, CONSTITUANT AINSI LE POINT DE DÉPART DE L'EXÉCUTION DU PROGRAMME

LE CODE

DANS CETTE PARTIE NOUS ALLONS EXPLICITER CERTAINES PARTIES DÉLICATES DE NOTRE CODE :

```
typedef struct s_coeff
{
    int a;
    int b;
}   t_coeff;
```

VOICI NOTRE STRUCTURE FRACTION OÙ A' REPRÉSENTE LE NUMÉRATEUR ET 'B' REPRÉSENTE LE DÉNOMINATEUR

```c
typedef struct s_polynome
{
    int degree;
    t_coeff *coeff;
    struct s_polynome *next;
}   t_polynome;
```

VOICI NOTRE STRUCTURE T_POLYNOME OÙ NOUS AVONS DEFINI 3 CHAMPS , LE DEGRE ET LE COEFFICIENT DU MONOME , ET UN POINTEUR NEXT VERS LE MONOME SUIVANT

```c
static size_t   count_words(char const *s, char c)
{
    size_t  i;
    size_t  count;

    i = 0;
    count = 0;
    while (s[i])
    {
        while (s[i] && s[i] == c)
            i++;
        if (s[i])
        {
            if (i == 0 || s[i - 1] == c)
                count++;
            i++;
        }
    }
    return (count);
}
```

- CETTE FONCTION PERMET DE COMPTER LE NOMBRE DE MOTS DANS UNE CHAINE DE CARACTÈRES

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
static size_t   calc_len(char const *s, char c)
{
    size_t  i;
    size_t  count;

    count = 0;
    i = 0;
    while (s[i] != c && s[i])
    {
        i++;
        count++;
    }
    return (count);
}
```

- CETTE FONCTION PERMET DE CALCULER LA LONGUEUR D'UN MOT DANS UNE CHAINE DE CARACTÈRES

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
static char *fill_subs(char const **s, char c)
{
    size_t  i;
    char    *subs;
    size_t  len;

    while (**s == c && **s)
        (*s)++;
    len = calc_len(*s, c);
    subs = (char *) malloc (sizeof (char) * (len + 1));
    if (!subs)
        return (NULL);
    i = 0;
    while (i < len)
    {
        subs[i] = **s;
        i++;
        (*s)++;
    }
    subs[i] = '\0';
    return (subs);
}
```

- CETTE FONCTION PERMET DE REMPLIR UNE SOUS-CHAINE DE CARACTÈRES
- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
char    **ft_split(char const *s, char c)
{
    size_t  nbr_words;
    char    **strings;
    size_t  i;

    i = 0;
    if (!s)
        return (NULL);
    nbr_words = count_words(s, c);
    strings = (char **) malloc (sizeof(char *) * (nbr_words + 1));
    if (!strings)
        return (NULL);
    while (i < nbr_words)
    {
        strings[i] = fill_subs(&s, c);
        if (!strings[i])
        {
            while (i > 0)
                free(strings[--i]);
            free (strings);
            return (NULL);
        }
        i++;
    }
    strings[i] = NULL;
    return (strings);
}
```

- CETTE FONCTION PERMET DE SÉPARER UNE CHAINE DE CARACTÈRES EN PLUSIEURS SOUS-CHAINES

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
void    lstadd_back(t_polynome **lst, t_polynome *new)
{
    t_polynome  *curr;

    if (new == NULL)
        return ;
    if (*lst == NULL)
    {
        *lst = new;
        return ;
    }
    curr = *lst;
    while (curr->next != NULL)
    {
        curr = curr->next;
    }
    curr->next = new;
}
```

- CETTE FONCTION PERMET D'AJOUTER UN ÉLÉMENT À LA FIN D'UNE LISTE CHAINÉE

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
char    *ft_substr(char const *s, unsigned int start, size_t len)
{
    unsigned int    i;
    char            *subs;

    if (!s)
        return (NULL);
    if (len > strlen(s) - start)
        len = strlen(s) - start;
    subs = (char *)malloc(len + 1);
    if (!subs)
        return (NULL);
    i = 0;
    while (i < len && s[start])
    {
        subs[i] = s[start];
        start++;
        i++;
    }
    subs[i] = '\0';
    return (subs);
}
```

- CETTE FONCTION PERMET DE SUBSTR UNE CHAINE DE CARACTÈRES
- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
void display_rules(void)
{
    printf("\t\t\t\t\t\x1b[32m(*) LET:     To Enter Your Polynom\n");
    printf("\t\t\t\t\t\x1b[32m(*) SET:     To Edit Your Polynom\n");
    printf("\t\t\t\t\t\x1b[32m(*) DISPLAY: To Display Your Polynom\n");
    printf("\t\t\t\t\t\x1b[32m(*) ADD:     Add Your Polynoms\n");
    printf("\t\t\t\t\t\x1b[32m(*) SUB:     Substract Your Polynoms\n");
    printf("\t\t\t\t\t\x1b[32m(*) MUL:     Multiple Your Polynoms\n");
    printf("\t\t\t\t\t\x1b[32m(*) POW:     POW Of Your Polynom\n");
    printf("\t\t\t\t\t\x1b[32m(*) AFFECT:  Affectation Of A P To The Other\n");
    printf("\t\t\t\t\t\x1b[32m(*) DER:     Derivation Of A Polynom\n");
    printf("\t\t\t\t\t\x1b[32m(*) INT:     Integration Of A Polynom\n");
    printf("\t\t\t\t\t\x1b[32m(*) EVAL:    Evaluation Of A Polynom\n");
    printf("\t\t\t\t\t\x1b[32m(*) EXIT:    To Exit The Program\n\n");
}
```

- CETTE FONCTION AFFICHE LES RÈGLES DU PROGRAMME, ELLE EST APPELÉE LORS DE L'EXECUTION DE LA COMMANDE "HELP"

- LA COMPLEXITÉ DE CETTE FONCTION EST O(1)

```c
char    *valid_cmd(char *cmd)
{
    if (strcmp(cmd, "LET") && strcmp(cmd, "SET") && strcmp(cmd, "DISPLAY") && strcmp(cmd, "ADD") && strcmp(cmd, "SUB") &&
    strcmp(cmd, "MUL") && strcmp(cmd, "POW") && strcmp(cmd, "AFFECT") && strcmp(cmd, "DER") && strcmp(cmd, "INT") &&
    strcmp(cmd, "EVAL") && strcmp(cmd, "EXIT") && strcmp(cmd, "Help"))
    {
        printf("\x1b[31mError: Command Not Found\n");
        return NULL;
    }
    return cmd;
}
```

- CETTE FONCTION PERMET DE VÉRIFIER SI LA COMMANDE ENTRÉE PAR L'UTILISATEUR EST VALIDE OU NON
- LA COMPLEXITÉ DE CETTE FONCTION EST O(1)

```c
void free_terminal(char **terminal)
{
    int i = 0;
    t_polynome *tmp = NULL;
    while (terminal[i])
    {
        free(terminal[i]);
        i++;
    }
    free(terminal);
}
```

- CETTE FONCTION LIBERE LA MÉMOIRE ALLOUÉE POUR LA FONCTION FT_SPLIT
- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
void free_p1(t_polynome **p1)
{
    t_polynome *tmp = NULL;
    while (*p1)
    {
        tmp = *p1;
        *p1 = (*p1)->next;
        free(tmp);
    }
}
```

- CETTE FONCTION PERMET DE LIBÉRER LA MÉMOIRE ALLOUÉE POUR LA LISTE

- CHAINÉE P1 LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
void free_p2(t_polynome **p2)
{
    t_polynome *tmp = NULL;
    while (*p2)
    {
        tmp = *p2;
        *p2 = (*p2)->next;
        free(tmp);
    }
}
```

- CETTE FONCTION PERMET DE LIBÉRER LA MÉMOIRE ALLOUÉE POUR LA LISTE CHAINÉE P1
- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
int new_degree(int *degs, int degree)
{
    int i = 0;
    while (i < 100)
    {
        if (degs[i] == degree)
            return 0;
        i++;
    }
    return 1;
}
```

- FONCTION VÉRIFIE SI LE DEGRÉ DEGREE EST DÉJÀ PRÉSENT DANS CE TABLEAU. SI LE DEGRÉ EXISTE, LA FONCTION RENVOIE 0 (FAUX) INDIQUANT QUE LE DEGRÉ N'EST PAS NOUVEAU. SI LE DEGRÉ N'EST PAS TROUVÉ DANS LE TABLEAU, LA FONCTION RENVOIE 1 (VRAI), INDIQUANT QUE LE DEGRÉ EST NOUVEAU.

- LA COMPLEXITÉ DE CETTE FONCTION EST O(1)

```c
t_polynome* optmz_polynom(t_polynome* p) {
    t_polynome* result = NULL;
    t_polynome* current = p;

    while (current) {
        t_polynome* temp = current->next;
        while (temp) {
            if (current->degree == temp->degree) {
                current->coeff->a += temp->coeff->a;
                current->coeff->b += temp->coeff->b;
                t_polynome* to_delete = temp;
                current->next = temp->next;
                free(to_delete->coeff);
                free(to_delete);
                temp = current->next;
            } else {
                temp = temp->next;
            }
        }
        t_polynome* simplified = malloc(sizeof(t_polynome));
        if (!simplified) {
            return NULL;
        }
        simplified->degree = current->degree;
        simplified->coeff = current->coeff;
        simplified->next = result;
        result = simplified;
        current = current->next;
    }
    t_polynome* reversed = NULL;
    while (result) {
        t_polynome* temp = result->next;
        result->next = reversed;
        reversed = result;
        result = temp;
    }
    return reversed;
}
```

- CETTE FONCTION PERMET D'OPTIMISER LE POLYNOME ENTRÉ PAR L'UTILISATEUR
- /LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
t_polynome* create_polynome(char* str, t_polynome* p)
{
    int i = 0;
    int j = 0;
    int len = strlen(str);
    int degree = 0;
    int coeff_a = 0;
    int coeff_b = 0;
    int sign = 1;

    while (str[i])
    {
        while (str[i] == ' ')
            i++;
        if (str[i] == '+' || str[i] == '-')
        {
            if (str[i] == '-')
                sign = -1;
            i++;
        }
        while (str[i] && str[i] >= '0' && str[i] <= '9')
        {
            coeff_a = coeff_a * 10 + (str[i] - '0');
            i++;
        }
        if (!coeff_a)
            coeff_a = 1;
        if (str[i] == '/')
            i++;
```

```
while (str[i] && str[i] >= '0' && str[i] <= '9')
{
    coeff_b = coeff_b * 10 + (str[i] - '0');
    i++;
}
if (!coeff_b)
    coeff_b = 1;
t_polynome* node = malloc(sizeof(t_polynome));
if (!node)
    return (NULL);
node->coeff = malloc(sizeof(t_coeff));
if (!node->coeff)
    return (NULL);
node->coeff->a = coeff_a * sign;
node->coeff->b = coeff_b;
if (str[i] == 'X')
{
    if (str[i + 1] == '^')
    {
        i += 2;
        while (str[i] && str[i] >= '0' && str[i] <= '9')
        {
            degree = degree * 10 + (str[i] - '0');
            i++;
        }
        node->degree = degree;
    }
    else
    {
        node->degree = 1;
        i++;
    }
```

```
        else
            node->degree = 0;
        node->next = NULL;
        if (p == NULL)
            p = node;
        else
        {
            t_polynome* tmp = p;
            while (tmp->next != NULL)
                tmp = tmp->next;
            tmp->next = node;
        }
        degree = 0;
        coeff_a = 0;
        coeff_b = 0;
        sign = 1;
        i++;
    }
    return optmz_polynom(p);
}
```

- CETTE FONCTION PERMET DE CRÉER UNE LISTE CHAINÉE QUI CONTIENT LES COEFFICIENTS ET LES DEGRÉS DU POLYNOME ENTRÉ PAR L'UTILISATEUR

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
void display_polynome(t_polynome *p, char name)
{
    t_polynome *tmp = p;
    int started = 0;
    int frac = 0;
    printf("\t\t\t\t\t\t<< %c = ", name);
    while (tmp)
    {
        frac = !(tmp->coeff->a % tmp->coeff->b) ? (tmp->coeff->a / tmp->coeff->b) : 0;
        if (tmp->coeff->a >= 0 && started)
            printf("+");
        if (tmp->coeff->a >= 0 && !started)
            started = 1;
        if (frac)
            printf("%d", frac);
        else
        {
            printf("%d", tmp->coeff->a);
            if (tmp->coeff->b > 1)
                printf("/%d", tmp->coeff->b);
        }
        if (tmp->degree > 0)
            printf("X");
        if (tmp->degree > 1)
            printf("^%d", tmp->degree);
        tmp = tmp->next;
        frac = 0;
    }
    printf(" >>\n\n\n");
}
```

- CETTE FONCTION PERMET D'AFFICHER LE POLYNOME ENTRÉ PAR L'UTILISATEUR

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
t_polynome *add_polynomes(t_polynome *p1, t_polynome *p)
{
    t_polynome *res = NULL;
    t_polynome *tmp = p1;
    t_polynome *tmp2 = p;
    t_polynome *node = NULL;
    while (tmp && tmp2)
    {
        node = malloc(sizeof(t_polynome));
        if (!node)
            return (NULL);
        node->coeff = malloc(sizeof(t_coeff));
        if (!node->coeff)
            return (NULL);
        if (tmp->degree == tmp2->degree)
        {
            node->degree = tmp->degree;
            node->coeff->a = (tmp->coeff->a * tmp2->coeff->b) + (tmp2->coeff->a * tmp->coeff->b);
            node->coeff->b = tmp->coeff->b * tmp2->coeff->b;
            lstadd_back(&res, node);
            tmp = tmp->next;
            tmp2 = tmp2->next;
        }
        else if (tmp->degree > tmp2->degree)
        {
            node->degree = tmp->degree;
            node->coeff->a = tmp->coeff->a;
            node->coeff->b = tmp->coeff->b;
            lstadd_back(&res, node);
            tmp = tmp->next;
        }
```

```
        else
        {
            node->degree = tmp2->degree;
            node->coeff->a = tmp2->coeff->a;
            node->coeff->b = tmp2->coeff->b;
            lstadd_back(&res, node);
            tmp2 = tmp2->next;
        }
    }
    while (tmp)
    {
        node = malloc(sizeof(t_polynome));
        node->degree = tmp->degree;
        node->coeff->a = tmp->coeff->a;
        node->coeff->b = tmp->coeff->b;
        lstadd_back(&res, node);
        tmp = tmp->next;
    }
    while (tmp2)
    {
        node = malloc(sizeof(t_polynome));
        node->degree = tmp2->degree;
        node->coeff->a = tmp2->coeff->a;
        node->coeff->b = tmp2->coeff->b;
        lstadd_back(&res, node);
        tmp2 = tmp2->next;
    }
    return optmz_polynom(res);
}
```

- CETTE FONCTION PERMET D'ADDITIONNER DEUX POLYNOMES

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
t_polynome *sub_polynoms(t_polynome *p1, t_polynome *p)
{
    t_polynome *res = NULL;
    t_polynome *tmp = p1;
    t_polynome *tmp2 = p;
    t_polynome *node = NULL;
    while (tmp && tmp2)
    {
        node = malloc(sizeof(t_polynome));
        if (!node)
            return (NULL);
        node->coeff = malloc(sizeof(t_coeff));
        if (!node->coeff)
            return (NULL);
        if (tmp->degree == tmp2->degree)
        {
            node->degree = tmp->degree;
            node->coeff->a = (tmp->coeff->a * tmp2->coeff->b) - (tmp2->coeff->a * tmp->coeff->b);
            node->coeff->b = tmp->coeff->b * tmp2->coeff->b;
            lstadd_back(&res, node);
            tmp = tmp->next;
            tmp2 = tmp2->next;
        }
        else if (tmp->degree > tmp2->degree)
        {
            node->degree = tmp->degree;
            node->coeff->a = tmp->coeff->a;
            node->coeff->b = tmp->coeff->b;
            lstadd_back(&res, node);
            tmp = tmp->next;
        }
```

```
        else
        {
            node->degree = tmp2->degree;
            node->coeff->a = tmp2->coeff->a;
            node->coeff->b = tmp2->coeff->b;
            lstadd_back(&res, node);
            tmp2 = tmp2->next;
        }
    }
    while (tmp)
    {
        node = malloc(sizeof(t_polynome));
        node->degree = tmp->degree;
        node->coeff->a = tmp->coeff->a;
        node->coeff->b = tmp->coeff->b;
        lstadd_back(&res, node);
        tmp = tmp->next;
    }
    while (tmp2)
    {
        node = malloc(sizeof(t_polynome));
        node->degree = tmp2->degree;
        node->coeff->a = tmp2->coeff->a;
        node->coeff->b = tmp2->coeff->b;
        lstadd_back(&res, node);
        tmp2 = tmp2->next;
    }
    return optmz_polynom(res);
}
```

- CETTE FONCTION PERMET DE SOUSTRAIRE DEUX POLYNOMES
- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
t_polynome *mul_polynoms(t_polynome *p1, t_polynome *p)
{
    t_polynome *res = NULL;
    t_polynome *tmp = p1;
    t_polynome *tmp2 = p;
    t_polynome *node = NULL;
    while (tmp)
    {
        tmp2 = p;
        while (tmp2)
        {
            node = malloc(sizeof(t_polynome));
            if (!node)
            return (NULL);
            node->coeff = malloc(sizeof(t_coeff));
            if (!node->coeff)
                return (NULL);
            node->degree = tmp->degree + tmp2->degree;
            node->coeff->a = tmp->coeff->a * tmp2->coeff->a;
            node->coeff->b = tmp->coeff->b * tmp2->coeff->b;
            lstadd_back(&res, node);
            tmp2 = tmp2->next;
        }
        tmp = tmp->next;
    }
    return optmz_polynom(res);
}
```

- CETTE FONCTION PERMET DE MULTIPLIER DEUX POLYNOMES

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
t_polynome *der_polynom(t_polynome *p)
{
    t_polynome *res = NULL;
    t_polynome *tmp = p;
    t_polynome *node = NULL;
    while (tmp)
    {
        node = malloc(sizeof(t_polynome));
        if (!node)
            return (NULL);
        node->coeff = malloc(sizeof(t_coeff));
        if (!node->coeff)
            return (NULL);
        node->degree = tmp->degree - 1;
        node->coeff->a = tmp->coeff->a * tmp->degree;
        node->coeff->b = tmp->coeff->b;
        lstadd_back(&res, node);
        tmp = tmp->next;
    }
    return optmz_polynom(res);
}
```

- CETTE FONCTION PERMET DE DERIVEE UNE POLYNOMES

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

```c
t_polynome *int_polynom(t_polynome *p)
{
    t_polynome *res = NULL;
    t_polynome *tmp = p;
    t_polynome *node = NULL;
    while (tmp)
    {
        node = malloc(sizeof(t_polynome));
        if (!node)
            return (NULL);
        node->coeff = malloc(sizeof(t_coeff));
        if (!node->coeff)
            return (NULL);
        node->degree = tmp->degree + 1;
        node->coeff->a = tmp->coeff->a;
        node->coeff->b = tmp->coeff->b * (tmp->degree + 1);
        lstadd_back(&res, node);
        tmp = tmp->next;
    }
    return optmz_polynom(res);
}
```

- CETTE FONCTION PERMET DE INTEGRER UNE POLYNOMES

- LA COMPLEXITÉ DE CETTE FONCTION EST O(N)

L'EXECUTION DU PROGRAMME

```
                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom

                                    Enter <Help> To See The Rules
                                (*) LET:      To Enter Your Polynom
                                (*) SET:      To Edit Your Polynom
                                (*) DISPLAY: To Display Your Polynom
                                (*) ADD:      Add Your Polynoms
                                (*) SUB:      Substract Your Polynoms
                                (*) MUL:      Multiple Your Polynoms
                                (*) POW:      POW Of Your Polynom
                                (*) AFFECT:   Affectation Of A P To The Other
                                (*) DER:      Derivation Of A Polynom
                                (*) INT:      Integration Of A Polynom
                                (*) EVAL:     Evaluation Of A Polynom
                                (*) EXIT:     To Exit The Program

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom

                                    Enter <Help> To See The Rules

>> LET P=3/4X^2+4X+2
                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom

                                    Enter <Help> To See The Rules

>> DISPLAY P
                                    << P = 3/4X^2+4X+2 >>


                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom

                                    Enter <Help> To See The Rules

>> LET Q=2X^2+1/2X+3
                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom

                                    Enter <Help> To See The Rules

>> DISPLAY Q
                                    << Q = 2X^2+1/2X+3 >>
```

```
>> SUB P,Q
                                        << R = -9/4X^2-5X1 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> ADD P,Q
                                        << R = 15/4X^2+9X+5 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> MUL P,Q
                                        << R = 9/4X^4+21/4X^3+6/4X^2+6X^3+14X^2+4X+9X^2+21X+6 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> POW P,2
                                        << R = 9/16X^4+6/4X^3+9/4X^2+6/4X^3+4X^2+6X+9/4X^2+6X+9 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> EVAL P FOR X = 3
                                        << R = 36 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>>
```

```
                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> DER P
                                        << R = 6/4X+2+0 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> INT Q
                                        << R = 1X^3+7/2X^2+2X >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> AFFECT P,Q
                                        << Q = 3/4X^2+2X+3 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> SET Q=4
                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
>> DISPLAY Q
                                        << Q = 4 >>

                    Hi There, Welcome To Your Small App, We Will Help You To Handle Your Polynom
                                        Enter <Help> To See The Rules
```

CONCLUSION

GRÂCE À CE PROJET, NOUS AVONS ACQUIS UNE EXPÉRIENCE PRÉCIEUSE DANS LA RÉALISATION D'UN PROGRAMME EN C DÉDIÉ AU CALCUL DES POLYNÔMES ET À LA RÉALISATION DE DIVERS TRAITEMENTS SUR EUX. CETTE INITIATIVE A POUR OBJECTIF DE SIMPLIFIER LE CALCUL DES POLYNÔMES POUR LES UTILISATEURS. NOUS AVONS PARTICULIÈREMENT SOULIGNÉ L'IMPORTANCE DES STRUCTURES, UTILISANT LES LISTES CHAÎNÉES, POUR DÉFINIR DE MANIÈRE EFFECTIVE LES COMPOSANTS DES POLYNÔMES. CES CONCEPTS ONT ÉTÉ ABORDÉS AU COURS DE NOTRE PROGRAMME D'ÉTUDES, METTANT EN LUMIÈRE L'IMPORTANCE DE TRAVAILLER EN BINÔME POUR OPTIMISER LA COMPRÉHENSION ET L'EFFICACITÉ DANS LA RÉSOLUTION DE PROBLÈMES COMPLEXES.