钱立坤(Qian Likun) 15310116001
Charles-Qian@outlook.com

孙 皓(Sun Hao) 15310116005
sh9339@163.com

Le rapport du projet de langage C

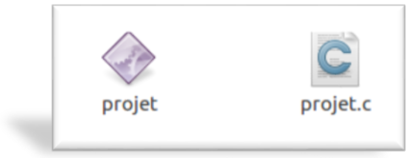
Mercredi. 04.01.2017

Sommaire

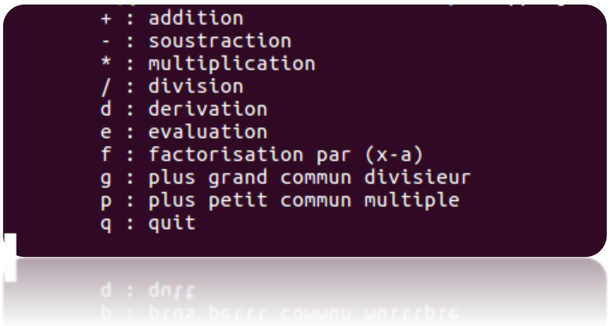
I La description de notre programme

Ce projet est de realiser un ensemble de fonctions per mettantant la manipulation de polynômes. Il comprend deux parties, comprendu le fichier source **projet.c** et **projet** après traduire.

Comme:



L'utilisateur peut voir l'interface d'ouverture de la procédure, comme ci – dessous:



Le menu d'utilisation est:

--Initialisation;

--Lecture et ecriture;

--Evaluation de p(x) par la methode de Horner pour un x donne;

--Addition;

--Soustraction;

--Multiplication;

--Derivation;

--Division euclidienne;

--Factorisation par (x-α), α un entier donne;

--plus grand commun diviseur de deux polynômes;

--plus petit commun multiple de deux polynômes.

C'est le début de la procédure de menu, vous pouvez effectuer l'opération que vous voulez. Par exemple, vous voulez calculer la **sommation**, et vous avez besoin d'écrire + , ensuite vous écrivez les deux polynômes selon la suggestion.

D'autres procédures de fonctionnement sont le même chose, on ne peut pas expliquer un par un.

II La pensée de notre programme

Comme on ne peut pas utiliser **Pointer** et **les structures et unions**, on ne utilise que les déclarations essentiels, comme **if** , **while** , **for** etc. . On utilise les énoncés itératifs.

Notre programme comprend la fonction principal et les routines, chacune opération est réalisé par appel des routines. Dans laquelle, tous les polynômes sont donnés dans le tableau **degre** homologue et la variable **q** pour obtenir une fonction de l'utilisation de multiples. Ensuite, l'autres routines représente respectivement une fonction. Enfin, ce sont les fonctions **écriture** et **double_écritures** . Ils peuvent **printf** les polynômes qui sont calculés. La fonction principale appelle les routines.

La fonction **lecture** , c’est le plus capital. Comme on écrit les chaînes de caractères, le système ne peut pas distinguer un indice et un coefficient .On a une circulation continue et constante de jugement pour atteindre cet objectif. D’abord, on sépare chacun monôme dans le polynôme et place les monômes dans un tableau bidimensionnalité par un ligne, donc dans chacun ligne, il y a un monôme. Pour chaque monôme, détermine les signes positifs et négatifs, le coefficient de détermination du coefficient de valeur absolue et de l'indice. Finalement, on mise en ordre par le degré en croissant.

En raison de temps urgent et notre capacité limitée, on ne peut pas écrire **debug** , mais s’il y a une erreur , on la trouve dans le polynôme d’entré. L’autre, des déclarations de rigueur sont limitées à une plage particulière, et ils doivent encore être améliorées.

III La description des tableaux et des fonctions

3.1 La fonction principal

La fonction principal comprend un énoncé **switch** , et il offre la liste des fonctions comme addition, soustraction, multiplication, division, dérivation, évaluation, factorisation, le plus grand commun diviseur, et le plus petit commun multiple etc. .Comme:

```
switch(a)
  {
  case '+':{lecture(p1,1);lecture(p2,2);addition(p1,p2,result);
      printf("p1(x) + p2(x) = ");ecriture(result);}                  break;

  case '-':{lecture(p1,1);lecture(p2,2);soustraction(p1,p2,result);
      printf("p1(x) - p2(x) = ");ecriture(result);}                  break;

  case '*':{lecture(p1,1);lecture(p2,2);multiplication(p1,p2,result);
      printf("p1(x) * p2(x) = ");ecriture(result);}                  break;

  case 'd':{lecture(p1,0);derivation(p1,result);ecriture(result);}   break;

  case 'e':{lecture(p1,0);evaluation(p1);}                           break;

  case '/':{lecture(p1,1);lecture(p2,2);division_euclidienne(p1,p2,r1,r2);
      printf("AN: p1(x) = (");double_ecriture(r1);
      printf(")*p2(x) + (");double_ecriture(r2);printf(")");}        break;

  case 'g':{lecture(p1,1);lecture(p2,2);pgcd(p1,p2,result);
      printf("Le plus grand commun diviseur est:");double_ecriture(result);}break;

  case 'p':{lecture(p1,1);lecture(p2,2);ppcm(p1,p2,result);
      printf("Le plus petit commun mutiple est:");double_ecriture(result);} break;

  case 'f':{lecture(p1,0);factorisation(p1);}                        break;

  case 'q':                                                          break;
```

3.1.1 double result [CMONOME]

Ce tableau représente le dernier résultat après calculer.

3.1.2 double p1 [CMONOME]

C'est le premier polynôme dans les deux polynômes que vous écrivez.

PS:Pour la fonction utilisant deux polynômes, il montre ***P1(x)= , P2(X)=*** *.*

3.1.3double p2 [CMONOME]

C'est le deuxième polynôme dans les deux polynômes que vous écrivez.

PS:En même, pour la fonction utilisant un polynôme, il ne montre que ***p(X)=*** *.*

3.2 void lecture (double COEFF_ORDONNE[CMONOME], int q)

C'est la fonction qui comprend le tableau **COEFF_ORDONNE[CMONOME]** et le variable **q** , et elle montre le dernier polynôme après calculer quand vous écrivez.

3.2.1char CLASSI [CMONOME][LMONOME]

C'est le tableau **char** pour classifier les monômes dans le polynôme par un ligne (#define CMONOME, LMONOME . **CMONOME** est le nombre des monômes dans un polynôme donc il y a **CMONOME** lignes. **LMONOME** est la longueur d'un monôme.)pour fonctionner.

Par exemple, vous écrivez :

5x^2+2x^4

Utilisant ce tableau, il est comme ça :

5x^2

2x^4

Ce tableau est très important dans ce programme.Il est le basic de ce programme.

j(type int)est le premier indice de ce tableau

k(type int)est le deuxième indice de ce tableau

3.2.2 char OCCA [8000]

C'est le tableau occasionnel, sa longueur est 8000. Il amasse les caractères qui représent le polynôme.

i (type int)est l'indice de ce tableau

L(type int)est la longueur de ce tableau

3.2.3 double COUNT [LMONOME]

Ce tableau compte le nombre des éléments par un ligne, et donne cette valeur.

Par exemple, la longueur de **25x^12** est 6, l'élément dans **COUNT[0]** est 6.

x(type **int**)est l'indice de ce tableau.

3.2.4 int PLUS_MOINS [CMONOME]

Ce tableau amasse les signes positives et negatives pour représenter le signe de chacun monomer et montre +1 ou -1.

Par exemple, **+ - = -** ; **+ + = +** etc.

```
case '-' :
  {
    if(CLASSI[j][k+1]=='+')
      PLUS_MOINS[pm]=-1;
    else if(CLASSI[j][k+1]=='-')
      PLUS_MOINS[pm]=1;
    else
      PLUS_MOINS[pm]=-1;
  }
  ; break;
case '+' :
  {
    if(CLASSI[j][k+1]=='+')
      PLUS_MOINS[pm]=1;
    else if(CLASSI[j][k+1]=='-')
      PLUS_MOINS[pm]=-1;
    else
      PLUS_MOINS[pm]=1;
  }
  ; break;
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
case '0':
case 'x': PLUS_MOINS[pm]=1 ;break;
}
```

pm(type int) est l'indice de ce tableau.

3.2.5char COEFF [CMONOME][LMONOME]

C'est le tableau **coefficient** en début.
jcoe, **kcoe**(type int)sont les indices de ce tableau.

3.2.6 double COEFF2 [CMONOME][LMONOME]

C'est le tableau **coefficient** (type **double**), transforment les caractères dans COEFF [CMONOME][LMONOME] pour les donnés **int** , c'est pour mieux calculer.
Comme:

```
COEFF2[jcoe][kcoe]=COEFF[jcoe][kcoe];
COEFF2[jcoe][kcoe]=COEFF2[jcoe][kcoe]-48;
```

3.2.7 double REELCOEFF [CMONOME]

C'est le tableau **coefficient** après combines, c'est à dire, PLUS_MOINS[CMONOME] et COEFF2[CMONOME][LMONOME] sont combinés. Il montre chacun coefficient par un monôme exactement.
Comme:

```
for(jcoe=0;jcoe<J;jcoe++)
  {
    for(kcoe=0;kcoe<COUNT_2[x];kcoe++)
      {
        REELCOEFF[reel]=REELCOEFF[reel]+COEFF2[jcoe][kcoe]*pow(10,(COUNT_2[x]-kcoe-1));

      }
    REELCOEFF[reel]=REELCOEFF[reel]*PLUS_MOINS[pm];
    pm++;
    reel++;
    x++;
  }
```

On obtient les réels coefficients après calculer.
reel(type **int**) représente le nombre des monômes.

3.2.8 double COUNT_2 [LMONOME]

Ce tableau compte le nombre des éléments par un ligne dans le tableau COEFF2.

3.2.9 int INDICE [CMONOME]

Ce tableau représente le degré d'un monôme.
ind(type **int**) est l'indice de ce tableau
trans(type **int**) est la quantité intermédiaire
Comme:

```
trans=CLASSI[j][k+1];
trans=trans-48;
INDICE[ind]=INDICE[ind]+trans*pow(10,(COUNT[x]-k-2));
```

3.2.10 int COEFF_ORDONNE [CMONOME]

Ce tableau représente le tableau **coefficient** après calculer. En même temps, il est placé par degré.
Comme:

```
COEFF_ORDONNE[INDICE[x]]=COEFF_ORDONNE[INDICE[x]]+REELCOEFF[x];
```

q est l'indice de ce tableau.

3.3 void evaluation(double coeff[CMONOME])

C'est la fonction qui donne la valeur. Sa variable est le polynôme que vous écrivez.

3.4 voidaddition(doublecoeff1[CMONOME], doublecoeff2[CMONOME], doubleresult[CMONOME])

C’est la fonction **sommation** , comprendu les deux polynômes que vous écrivez et le résultat après calculer.

3.5 voidsoustraction(doublecoeff1[CMONOME], doublecoeff2[CMONOME], doubleresult[CMONOME])

C’est la fonction **soustraction** , comprendu les deux polynômes que vous écrivez et le résultat après calculer.

3.6 voidmultiplication(doublecoeff1[CMONOME], doublecoeff2[CMONOME], doubleresult[CMONOME])

C’est la fonction **multiplication** , comprendu les deux polynômes que vous écrivez et le résultat après calculer.

3.7 void derivation(double coeff[CMONOME], double result[CMONOME])

C’est la fonction **dérivation** , comprendu le polynôme que vous écrivez et le résultat après calculer.

3.8 void division_euclidienne(double coeff1[CMONOME], double coeff2[CMONOME], double result[CMONOME], double reste[CMONOME])

C’est la fonction **division euclidienne** , comprendu les deux polynômes que vous écrivez, le résultat après calculer et le reste par chacun calcule.

max1 est le maximal exponent de monôme dans le tableau **coeff1**.
max2 est le maximal exponent de monôme dans le tableau **coeff2**.
X est l’indice des tableaux.

3.8.1 double centre[CMONOME]

C’est la quantité intermédiaire pour donner la valeur.

3.8.2 double quotient[CMONOME]

C’est le deuxième polynôme que vous écrivez.

3.8.3 double mu[CMONOME]

C’est le produit de **result** et **double quotient[CMONOME]** par une circulation.

3.8.4 double dividende[CMONOME]

C’est le premier polynôme que vous écrivez.

3.9 voidpgcd(doublecoeff1[CMONOME], doublecoeff2[CMONOME], doublequotient2[CMONOME])

C’est la fonction **le plus grand commun diviseur** , comprendu les deux polynômes que vous écrivez et le résultat après calculer.

3.9.1 double reste[CMONOME]

Le tableau **reste** représente le dividende, donc le polynôme qui a un supérieur degré. **max1** est le maximal exponent.

3.9.2 double quotient[CMONOME]

Le tableau **quotient** représente le diviseur, donc le polynôme qui a un inférieur degré. **max2** est le maximal exponent.

3.9.3 double reste2[CMONOME]

Le tableau **reste2** représente le reste.

3.9.4double quotient2[CMONOME]

Le tableau **quotient2** représente le diviseur dont on a besoin.

3.10 voidppcm(doublecoeff1[CMONOME], doublecoeff2[CMONOME], double result[CMONOME])

C’est la fonction **le plus petit commun multiple** , comprendu les deux polynômes que vous écrivez et le résultat après calculer.

3.10.1 double mu[CMONOME]

mu représente le produit de deux polynômes.

3.10.2 double pgcd12[CMONOME]

C’est le tableau **le plus petit commun multiple** on a défini.

3.10.3 double reste[CMONOME]

Le tableau **reste** représente le reste.

3.11 void factorisation(double coeff[CMONOME])

C’est la fonction **factorisation**, comprendu le polynôme que vous écrivez.

3.11.1int maxresoudre[100]

Ce tableau amasse les diviseurs de coefficient qui a le maximal exponent.

3.11.2 int minresoudre[100]

Ce tableau amasse les diviseurs de coefficient qui a le minimum exponent.

3.11.3 int coeffint[CMONOME]

C’est le tableau polynôme **coefficient** que vous écrivez.

3.11.4 double racine_pro[100]

C’est le tableau qui amasse les racines possibles.

3.11.5 double m[100]

C’est le type **double** de **int maxresoudre[100]**.

3.11.6 double n[100]

C’est le type **double** de **int minresoudre[100]** .

3.11.7 double reel[100]

C’est le tableau qui amasse les racines par calculer.

3.11.8 double reel2[100]

reel2 est le tableau **reel** qui a effacé les racines réitératifs.

3.11.9 double deriva[CMONOME]

La quantité intermédiaire dans le dérivation.

3.11.10 double final[100]

C’est le dernier tableau qui amasse les racines réels après calculer.

3.11.11 double ecri[CMONOME]

C’est le tableau qui donne les valeurs.

3.11.12double f[CMONOME]

C’est le tableau qui amasse le polynôme f(X).

3.12 void ecriture(double result[CMONOME]) (void double_ecriture(doubleresult[CMONOME]))

Cette routine <printf> le tableau dans le terminal, pour les fonctions **division euclidienne** , **le plus grand commun diviseur** et **le plus petit commun multiple** , il faut <printf> par **type double** , donc void **double_écriture** est sûr les trois. Sa variable est le tableau **result** après chacun fonction.

IV Le procédé d’exécution de programme

Voici le processus d'exécution de programme détaillé:

4.1 Lecture et écriture

On sait l’utilisateur peut donner un polynôme discrétionnairement, donc la première étape de cette procédure est le problème d'un polynôme d'entrée.

Ici on défini **char CLASSI [CMONOME][LMONOME]**, **char OCCA [8000]**, **double COUNT [LMONOME]**, **int PLUS_MOINS [CMONOME]**, **char COEFF [CMONOME][LMONOME]**, **double COEFF2 [CMONOME][LMONOME]**, **double REELCOEFF [CMONOME]**, **double COUNT_2 [LMONOME]**, **int INDICE [CMONOME]**, **int COEFF_ORDONNE [CMONOME]**, il y a 10 tableaux en totale et une fonction **void lecture** .

D’abord, initialiser les tableaux.

```
for (x = 0; x < CMONOME; ++x) /*shuzu de chushihua*/
  INDICE[x] = 0;

for (x = 0; x < CMONOME; ++x)/*shuzu de chushihua.*/
  REELCOEFF[x] = 0;


/*shuzu de fuchuzhi.*/
for(j=0;j<CMONOME;j++)
  for(k=0;k<LMONOME;k++)
    CLASSI[j][k] = '|';

/*zhengfuhao shuzu de chushihua.*/
for(pm=0;pm<CMONOME;pm++)
  PLUS_MOINS[pm]=1;
```

PS:

*1. Défini le tableau **INDICE** , le tableau **REELCOEFF** et initialiser les tableaux égale à 0 ; initialiser le tableau **CLASSI** égale à | (Il n'y a pas de signification réelle) ; initialiser **PLUS_MOINS** égale à 1.*

*2 .x représente l’indice du tableau **coefficient** et le tableau **degré** ; j, k représentent les deux indices dans le tableau **classifier** ; pm représente l’indice du tableau **PLUS__MOINS** .*

Ensuite, on determine la valeur de **q** (q determine la situation de la fonction dans ce polynôme vous ecrivez). Par exemple, *pour la fonction utilisant deux polynômes, il montre* ***P1 (x)=*** , ***P2(X)=*** , mais *pour la fonction utilisant un polynôme, il ne montre que* ***p(X)=*** . **getchar** garantit le premier donne de polynôme

Comme:

```
if(q==0)
  printf("P(x) = ");
else
  printf("P%d(x) = ",q);

if(q==1 || q==0)
  ch=getchar();

for(x=0;x<100;x++)
  {
    if(x>0)
      {
        if(q==0)
          printf("P(x) = ");
        else
          printf("P%d(x) = ",q);
        if(q==1 || q==0)
          ch=getchar();
      }
```

ch=getchar *est donné les polynômes réellement.*

Fonctionne dans le terminal :

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:-
P1(x) = 2
P2(x) = 4
```

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:e
P(x) =
```

Effacer l'espace:

```
while((ch = getchar()) != '\n')
  {
    if(ch != ' ')
      {
        OCCA[i]=ch;
        i++;
        L=i;
      }
  }
```

PS:

1.On sélectionne par énoncé ***while*** *jusqu'à* ***\n*** *, la donne s'arrête,* ***while*** *s'arrête aussi.*

2.Dans chacune circulation, saute ***l'espace****, donne l'autres éléments dans* ***ch****, et donne chacun élément de* ***ch*** *dans le tableau* ***OCCA*** *.*

Après effacer l'espace, vérifier s'il y a une erreur dans **lecture**.

Comme:

```
j=0;
k=0;
x=0;
for(i=0;i<L;i=i)
  {
    if(OCCA[i]=='+'||OCCA[i]=='-')
      {
        CLASSI[j][k]=OCCA[i];
        i++;
        k++;
      }
    else
      {
        do{
          CLASSI[j][k]=OCCA[i];
          i++;
          k++;
        }
        while((OCCA[i]!='+' && OCCA[i]!='-') && i<L);
```

PS:

1. Défini les variables:

__j__(type __int__) est le premier indice dans le tableau __classifier__ ;

__k__(type __int__) est le deuxième indice dans le tableau __classifier__ ;

__i__ (type __int__) est l'indice du tableau __occasionnel__ ;

L(type int) est la longueur du tableau __occasionnel__ .

2. Sépare par les signes positives et negatives.

Donne le nombre des elements de ce ligne dans le tableau COUNT[x], quand il y a retour à la ligne.

```
if((OCCA[i]=='+'||OCCA[i]=='-') && i<L)
  {
    j++;
    COUNT[x] = k;
    x++;
  }
```

PS:Quand fonctionne __do-while__ , il est à la maximal longueur, donne dans __COUNT__ directement .

Sépare chacun signe positif ou négatif par ligne.

```
pm=0;
for(j=0;j<CMONOME;j++)
  {
    for(k=0;k<LMONOME;k++)
      {
        switch(CLASSI[j][k])
          {
          case '-' :
            {
              if(CLASSI[j][k+1]=='+')
                PLUS_MOINS[pm]=-1;
              else if(CLASSI[j][k+1]=='-')
                PLUS_MOINS[pm]=1;
              else
                PLUS_MOINS[pm]=-1;
            }
            ; break;
          case '+' :
            {
              if(CLASSI[j][k+1]=='+')
                PLUS_MOINS[pm]=1;
              else if(CLASSI[j][k+1]=='-')
                PLUS_MOINS[pm]=-1;
              else
                PLUS_MOINS[pm]=1;
            }
            ; break;
          case '1':
          case '2':
          case '3':
          case '4':
          case '5':
          case '6':
          case '7':
          case '8':
          case '9':
          case '0':
          case 'x': PLUS_MOINS[pm]=1 ;break;
          }
        pm++;
        k=LMONOME;
      }
  }
```

PS:

*1. Initialise **PLUS_MOINS***

2.Détermine le signe

3. Par exemple, +- = -; ++ = + etc.

```
x=0;

jcoe=0;kcoe=0;

for(j=0;j<CMONOME;j++)
  {
    for(k=0;k<LMONOME;k++)
      {
        if (CLASSI[j][k]=='|')
          {
            k=LMONOME;
            jcoe--;}
        else if((CLASSI[j][k]=='+' || CLASSI[j][k]=='-') && (CLASSI[j][k+1]!='+' && CLASSI[j][k+1]!='-'&&CLASSI[j][k+1]!='x'))
          {
            do
              {
                COEFF[jcoe][kcoe] = CLASSI[j][k+1];
                kcoe++;
                COUNT_2[x]=kcoe;
                k++;
              }
            while(CLASSI[j][k+1]!='x'&& CLASSI[j][k+1]!='|');
          }
        else if((CLASSI[j][k]=='+' || CLASSI[j][k]=='-') && (CLASSI[j][k+1]=='+' || CLASSI[j][k+1]=='-')&&(CLASSI[j][k+2]!='x'))
          {
            do
              {
                COEFF[jcoe][kcoe] = CLASSI[j][k+2];
                kcoe++;
                COUNT_2[x]=kcoe;
                k++;
              }
            while(CLASSI[j][k+2]!='x' && CLASSI[j][k+2]!='|');
          }
```

PS:

*Il y a trios cas comprendu **x** .*

***jcoe**, **kcoe** représentent le premier indice et le deuxième dans le tableau **coeff** .*

*1.Comme k=0, tous les éléments dans **CLASSI** sont < | >, c'est à dire tous les éléments dans cette ligne sont < | > , il signifie **jcoe** - - .*

*2.Quand le premier élément dans **classifier** est le signe, et le deuxième élément n'est pas le signe ou **x** , donc le deuxième élément est le coefficient dont on a besoin.*

*3.Quand le premier et le deuxième éléments sont tout le signe, mais le troisième n'est pas **x** , donc le troisième est le coefficient dont on a besoin.*

Quand le premier élément dans **classifier** est constante ou **x** , donne **+1** dans **PLUS_MOINS** .

```
        else if(CLASSI[j][k]=='1'||CLASSI[j][k]=='2'||CLASSI[j][k]=='3'||CLASSI[j][k]=='4'||CLASSI[j][k]=='5'||CLASSI[j][k]=='6'||CLASSI[j][k]=='7'||CLASSI[j][k]=='8'||
CLASSI[j][k]=='9'||CLASSI[j][k]=='0')
          {
            do
              {
                COEFF[jcoe][kcoe] = CLASSI[j][k];
                kcoe++;
                COUNT_2[x]=kcoe;
                k++;
              }
            while(CLASSI[j][k]!='x'&&CLASSI[j][k]!='|');
          }
        else if((CLASSI[j][k]=='x')||(CLASSI[j][k]=='+'&&CLASSI[j][k+1]=='x')||(CLASSI[j][k]=='-'&&CLASSI[j][k+1]=='x')||(CLASSI[j][k]=='+'&&CLASSI[j][k+1]=='-'&&CLASSI
[j][k+2]=='x')||(CLASSI[j][k]=='-'&&CLASSI[j][k+1]=='+'&&CLASSI[j][k+2]=='x')||(CLASSI[j][k]=='+'&&CLASSI[j][k+1]=='+'&&CLASSI[j][k+2]=='x')||(CLASSI[j][k]=='-'&&CLASSI[j]
[k+1]=='-'&&CLASSI[j][k+2]=='x'))
          {
            COEFF[jcoe][kcoe]='1';
            COUNT_2[x]=1;
          }
        k=LMONOME;
      }
```

*PS:1.Quand l'element dans **classifier** est constante, il est le coefficient dont on a besoin.*

*2.Quand l'élément est **x** **+x** **-x** **+-x** **-+x** **++x** **--x** , le coefficient est égale à 1.*

Vérifie le tableau classification.,

Comme:

```
x=0;
J=jcoe;
for(jcoe=0;jcoe<J;jcoe++)
  {
      for(kcoe=0;kcoe<COUNT_2[x];kcoe++)
        {
          COEFF2[jcoe][kcoe]=COEFF[jcoe][kcoe];
          COEFF2[jcoe][kcoe]=COEFF2[jcoe][kcoe]-48;
        }
      x++;
      |
  }
```

Maintenant!

Le tableau **PLUS_MOINS** est le signe positif et négatif pour chacun monome.

Le tableau **COUNT** est le nombre des elements dans chacun ligne.

Le tableau **COEFF** est le coefficient dans chacun monome par chacun ligne.(type **char**)

Comme :

```
reel=0;pm=0;x=0;

for(reel=0;reel<J;reel++)
    {
      REELCOEFF[reel]=0;
    }

reel=0;

for(jcoe=0;jcoe<J;jcoe++)
  {
    for(kcoe=0;kcoe<COUNT_2[x];kcoe++)
      {
        REELCOEFF[reel]=REELCOEFF[reel]+COEFF2[jcoe][kcoe]*pow(10,(COUNT_2[x]-kcoe-1));

      }
    REELCOEFF[reel]=REELCOEFF[reel]*PLUS_MOINS[pm];
    pm++;
    reel++;
    x++;
  }
```

PS:

*1. **reel**(type **int**) représente le nombre des monômes.*

*2.**REELCOEFF** le tableau **coefficient** réel, **J** est l'indice de **REELCOEFF**.*

*3. L'initialisation de **REELCOEFF** égale à 0, puis à COEFF2 ensemble, **count2** est inférieure à la longueur de chaque monômes, **kcoe** - -. Le nombre est multipé 1, 10 et 100, selon la place.*

*4. Chaque élément **REELCOEFF** qui a donné des symboles après multiplier par **PLUS__MOINS** .*

Maintenant, on a reussi à obtenir le tableau **degre** homologue.
Comme:

```
for(ind=0;ind<LMONOME;ind++)
  INDICE[ind]=0;

ind=0;x=0;

for(j=0;j<J;j++)
  {
    for(k=0;k<LMONOME;k++)
      {
        if(CLASSI[j][k]=='^')
          {
            while(k+1<COUNT[x])
              {
                trans=CLASSI[j][k+1];
                trans=trans-48;
                INDICE[ind]=INDICE[ind]+trans*pow(10,(COUNT[x]-k-2));
                k++;
              }
            break;
          }
        else if(CLASSI[j][k]=='x' && CLASSI[j][k+1]== '|')
          {
            INDICE[ind]=1;
            break;
          }
        else
          INDICE[ind]=0;

      }
    ind++;
    x++;
  }
```

PS:

*1. **ind** est l'indice de **INDICE***

*2. **trans**(type **int**)est la quantité intermédiaire.*

*3.Quand le premier élément dans **classifier** est **x** et le deuxième est < | > , donc le monôme est **x** , le degré est égale à 1.*

4.Sinon, il n'y a que une constante, le degré est égale à 0.

Combine les coefficients et les degrés, comme:

```
for(x=0;x<CMONOME;x++)
    COEFF_ORDONNE[x]=0;

for(x=0;x<CMONOME;x++)
  COEFF_ORDONNE[INDICE[x]]=COEFF_ORDONNE[INDICE[x]]+REELCOEFF[x];

for(x=0;x<CMONOME;x++)
  printf("%2.0f",COEFF_ORDONNE[x]);

printf("\n");
```

PS:

*1. **COEFF_ORDONNE** représente le tableau **coefficient** après calculer. En même temps, il est placé par degré.*

*2.**%2.0f** représente les fractions décimale nécessaires.*

Lecture est fini maintenant.

4.2 Initialisation sur les polynômes

Cette routine calcule le resultat par **L'axiome de Horner** en utilisant la valeur que vous donnez.

Donne le maximal exponent de **coeff** dans **max**

```
for(x=0;x <CMONOME;x++)
    if(coeff[x]!=0)
    max=x;
```

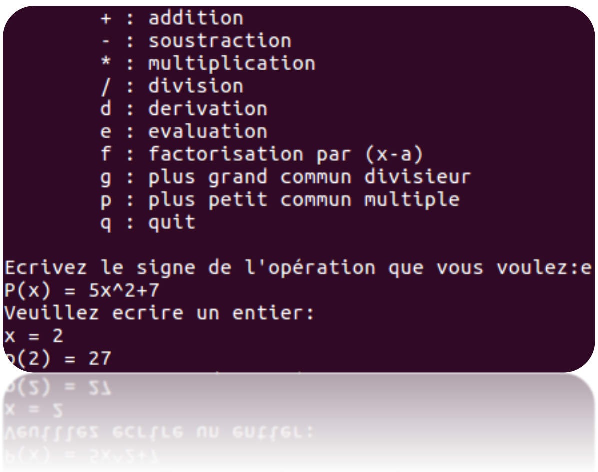
Par le maximal exponent, multiplie et additionne de l'intérieur vers l'extérieur

```
x=max;
if(max 0)
  result=coeff[x]*t+coeff[x-1];
  for(x=max-1;x =1;x--)
     result=result*t+coeff[x-1];
  printf("p(%d) = %d", t, result);
```

Si l'entrée est une constante, exporte directement.

```
else if(max==0)
  result=coeff[0];
  printf("p(%d) = %d", t, result);
```

Fonctionne en terminal:



4.3 Addition sur les polynômes

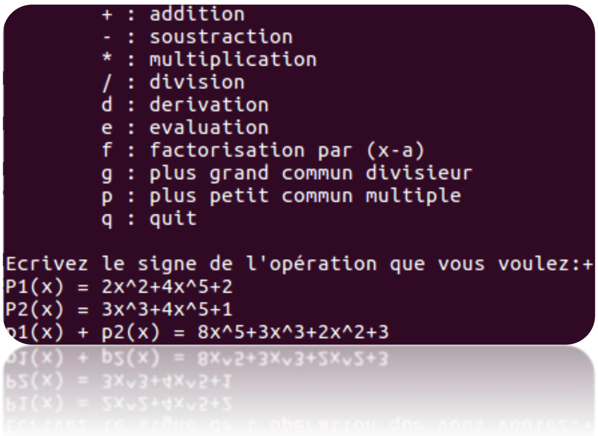
Cette routine additionne les elements homologues dans tous les deux tableaux, et donne la valeur dans **resultat** .

```
void addition(double coeff1[CMONOME],double coeff2[CMONOME],double result[CMONOME])
{
  int x=0;
  for(x=0;x<CMONOME;x++)
    result[x]=0;

  for(x=0;x<CMONOME;x++)
    result[x]=coeff1[x]+coeff2[x];
```

PS : L'addition des elements homologues.

Fonctionne en terminal:



```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:+
P1(x) = 2x^2+4x^5+2
P2(x) = 3x^3+4x^5+1
p1(x) + p2(x) = 8x^5+3x^3+2x^2+3
```

4.4 Soustraction sur les polynômes

Cette routine soustrait les elements homologues dans tous les deux tableaux, et donne la valeur dans **resultat** .

```
void soustraction(double coeff1[CMONOME],double coeff2[CMONOME],double result[CMONOME])
{
  int x=0;
  for(x=0;x<CMONOME;x++)
    result[x]=0;

  for(x=0;x<CMONOME;x++)
    result[x]=coeff1[x]-coeff2[x];
```

PS : La soustraction des elements homologues.

Fonctionne en terminal:

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:-
P1(x) = 3x^7+4x^2-8
P2(x) = 2x^2+7x-9
p1(x) - p2(x) = 3x^7+2x^2-7x+1
```

4.5 Multiplication sur les polynômes

Cette routine multiplie les elements homologues un par un dans tous les deux tableaux, et donne la valeur dans **resultat** .

```
void multiplication(double coeff1[CMONOME],double coeff2[CMONOME],double result[CMONOME])
{
  int x1=0;
  int x2=0;
  int x;

  for(x=0;x<CMONOME;x++)
    result[x]=0;

  for(x1=0;x1<CMONOME;x1++)
    {
      for(x2=0;x2<CMONOME;x2++)
        result[x1+x2]=result[x1+x2]+coeff1[x1]*coeff2[x2];
    }
```

*PS:Le tableau **result** comprend les deux éléments homologues.*

Fonctionne en terminal:

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:*
P1(x) = 2x^3+4x-7
P2(x) = 5x^5+6x^2+3
p1(x) * p2(x) = 10x^8+20x^6-23x^5+30x^3-42x^2+12x-21
```

4.6 Dérivation sur les polynômes

Cette routine donne chacun coefficient du tableau que vous ecrivez dans chacun coefficient qui a le degre inférieur 1 à lui, et multiplie le degre de ce terme pour la dérivation.

```c
void derivation(double coeff[CMONOME],double result[CMONOME])
{
  int x=0;
  for(x=0;x<CMONOME;x++)
    result[x]=0+coeff[x+1]*(x+1);

  result[CMONOME-1]=0;

  printf("p'(x) = ");
}
```

PS:

1.Faire le derivation pour un monôme, c'est à dire, tire son degré avant son coefficient et multiplie avec son coefficient; son nouveau degré est inférieur 1 à le premier.

2. Le tableau resultat après le calcule dernier, le dernier terme n'est pas initialise.

Fonctionne en terminal:

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:d
P(x) = 5x^6+8x^4+2-16x
p'(x) = 30x^5+32x^3-16
```

4.7 Division euclidienne

1.Pour les constantes, le division euclidienne est facile.

Par exemple:

Le premier :1515/600, on obtient 2...315;

Le deuxième :600/315, on obtient 1...285;

Le troisième :315/285, on obtient 1...30;

Le quatrième :285/30, on obtient 9...15;

Le cinquième :30/15, on obtient 2.

2.Pour les polynômes:

f(x)=p(x)*g(x)+r(x), selon la formule, dans le division euclidienne, remplace f(x) par g(x), et remplace g(x) par r(x), jusqu'au maximal exponent de r(x) est inférieur a le premier.

p(x) est le tableau **centre** , ce tableau est comme le compteur additif.

g(x) est le tableau **quotient** , quotient(diviseur)

r(x) est le tableau **reste** , reste

f(x) est le tableau **dividendei** , dividendei

Trouve le maximal exponent respectivement dans **coeff1** et **coeff2** .

```
for(x=0;x<CMONOME;x++)
  {
    if(coeff1[x]!=0)
      max1=x;
  }

for(x=0;x<CMONOME;x++)
  {
    if(coeff2[x]!=0)
      max2=x;
  }
```

Maintenant, on a le division par 2 pour 1 selon la demande.

Initialiser le tableau.

```
for(x=0;x<CMONOME;x++)
  {
    /*Initialiser le tableau.*/
    centre[x]=0;
    quotient[x]=0;

    /*D'abord, <coeff1> est comme residu et dividendei.*/
    dividende[x]=coeff1[x];
    reste[x]=coeff1[x];

    /*<coeff2> est comme quotient.*/
    quotient[x]=coeff2[x];
  }
```

*PS: D'abord, **coeff1** est comme residu et dividendei. **coeff2** est comme quotient.*

Avec le division euclidienne, **max1** baissera jusqu'à il est inférieur à **max2**.

```
while(max1>=max2)
  {
    /*Le dividendei divise par le diviseur font le quotient.*/
    centre[max1-max2]=centre[max1-max2]+reste[max1]/quotient[max2];

    /*Cumule le quotient pour obtenir le resultat.*/
    result[max1-max2]=result[max1-max2]+centre[max1-max2];

    /*On a obtenu le dividendei pour la circulation la prochaine fois.*/
    multiplication(result,quotient,mu);
    soustraction(dividende,mu,reste);

    /*Déterminer si le residu est égale à zéro.*/
    max1=0;
```

PS:

1. ***centre*** *est le quotient que le maximal exponent de f(x)divise g(x)et + +.*

2.Initialise ***centre*** *après donner chaque fois.*

3. Appel de ***void soustraction*** *et* ***void multiplication*** *.*

4. ***mu*** *est le produit de* ***result*** *et* ***g(x)*** *.*

5. Déterminer si le residu est nul.

```
for(x=0;x<CMONOME;x++)
  {
    centre[x]=0;

    /*-------------------------------------------------*/
    /*Cette étape est afin d'éviter des erreurs en langage C.
     *L'erreur moyenne est inférieur a 0.00000000001.
     *Cette situation sera apparaît dans l'autres routines.
     */
    if(reste[x]<0.00000000001 && reste[x]>-0.0000000000)
      reste[x]=0;
    /*-------------------------------------------------*/

    if(reste[x]!=0)
      max1=x;
```

PS: L'étape d'évaluation de précision afin d'éviter que la question de l'ordinateur d'erreurs, dans les opérations suivantes peut également avoir une situation similaire.

Fonctionne en terminal:

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:/
P1(x) = 3x^4+5x+7
P2(x) = 2x^2-9
AN: p1(x) = (1.50x^2+6.75)*p2(x) + (5.00x+67.75)
```

4.8 Factorisation par (x-α)

Cette routine factorise le polynôme que vous ecrivez par calculer, le diviseur du terme qui a le maximal exponent et le diviseur du terme qui a le minimum exponent, pour trouver les racines possibles, ensuite crible, les racines reels (comprendu les racines binaires, triplex......)

Change **coeff** pour le tableau **int** , car il faut être **entier** pour le reste.

coeffint[x]=coeff[x];

Calcule le maximal exponent et le minimum exponent.

min=x;

max=x;

Calculer tous les diviseurs et donne le valeur.

```
for(x=0;x<abs(coeffint[max]);x++)
  {
    if((coeffint[max]%(x+1))==0)
      {
        maxresoudre[i]=abs(coeffint[max]/(x+1));
        i++;
      }
  }
i=0;
for(x=0;x<abs(coeffint[min]);x++)
  {
    if((coeffint[min]%(x+1))==0)
      {
        minresoudre[i]=abs(coeffint[min]/(x+1));
        i++;
```

PS:

*1. **abs** représente la valeur absolue de coefficient.*

*2.Donne les diviseurs du maximal exponent et du minimum exponent dans le tableau **maxresoudre** , **minresoudre** .*

Efface 0 dans le tableau de diviseurs.

```
/*Efface 0 dans le tableau de diviseurs.*/
i=0;
for(x=0;x<CMONOME;x++)
  {
    if(maxresoudre[x]!=0)
      {
        maxresoudre[i]=maxresoudre[x];
        if(i!=x)
          maxresoudre[x]=0;
        i++;
      }
  }
j=0;
for(x=0;x<CMONOME;x++)
  {
    if(minresoudre[x]!=0)
      {
        minresoudre[j]=minresoudre[x];
        if(j!=x)
          minresoudre[x]=0;
        j++;
```

m, n sont les deux tableux **double** pour le division.

m[x]=maxresoudre[x];

n[x]=minresoudre[x];

*PS:Change **type int** par **type double** pour diviser.*

Divise les diviseurs pour obtenir les racines possibles.

racine_pro[x]=n[j]/m[i];

PS:Ce tableau est le tableau qui a tous les racines possibles.

Trouve les racines réels :

*result=result+coeff[a]*pow(racine_pro[x], a)*

Répète le même travail :

reel[i]=-racine_pro[x];

Donne les racines qui ont été éffacés les réitératifs de **reel** dans le tableau **reel2** .

if(reel[x]!=reel2[a])

reel2[i]=reel[x];

*L=i;(La longueur de **reel2**)*

Donne **coeff** dans **dervia** pour le debut du dérivation. Parce qu'une racine satisfait si le polynôme, ne peut pas satisfaire dérivé de polynôme; au contraire, si une racine de satisfaire et de satisfaire un polynôme, dérivé de polynôme, cette racine au moins double racine.

for(x=0;x CMONOME;x++)

deriva[x]=coeff[x];

Donne y=0 quand la circulation commence, quand ne donne pas, **y** ne augment pas, ensuite la circulation sera cessé.

```
y=0;

for(x=0;x<L;x++)
  {
    for(a=0;a<CMONOME;a++)
      result=result + deriva[a]*pow(reel2[x],a);

    if(result> -0.00000000001 && result< 0.00000000001)
      {
        final[i]=reel2[x];
        i++;
        y++;
      }
    result=0;
```

Faire le derivation et donne les racines dans le tableau.

deriva[a]=deriva[a+1](a+1);*

deriva[CMONOME-1]=0;

On peut trouver tous les racines qui ne sont pas 0 par les mesures. Puis, on determine le nombre de 0

result=0; y=1;

*PS:Ici, i n'est pas initialise, car **final** a besoin de donner les valeurs ensuite.*

for(x=0;x <CMONOME;x++)
deriva[x]=coeff[x];
*result =result + deriva[a]*pow(0, a);*
for(a=0;a< CMONOME;a++)
deriva[a]=deriva[a+1](a+1);*

PS:Faire le dérivation pour les polynômes de l'original, puis stockée dans le tableau d'origine.

deriva[CMONOME-1]=0;

On doit montrer par (x-α), donc le coefficient de **x** egale a 1

ecri[1]=1

*PS:ecri[1] représente tous les coefficients de **x** est égale à 1.*

```
if(L==0)
  {
    printf("Ce polynome est non-factorisation: ");
    ecriture(coeff);
  }
else
  {
    printf("le polynome par factorisation est: \n");

    for(x=0;x<L;x++)
      {
        ecri[0]=-final[x];

        printf("(");
        double_ecriture(ecri);
        printf(")*");
```

*PS:Si la longueur de **reel2** est égale à 0, ce polynôme est non-factorisation.*

Trouve les termes restants après choisir les termes cartésiens.

f[x]=coeff[x];
ecri[0]=-final[x];
division_euclidienne(f, ecri, mu, rest);

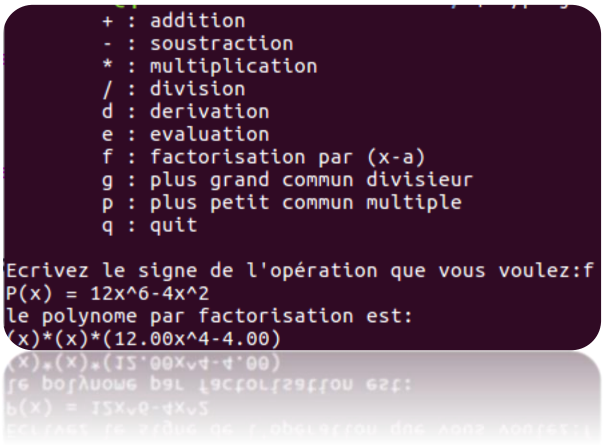
PS:

*1.Donne les éléments du tableau **coeff** dans f(x)*
*2.Le tableau **final** représente les racines réels.*
*3.Appel de la fonction **void division_euclidienne** , trouve les termes restes.*
f[a]=mu[a];

<Printf> les termes restants.

printf("(");
double_ecriture(mu);
printf(")");

Fonctionne en terminal:


+ : addition
- : soustraction
* : multiplication
/ : division
d : derivation
e : evaluation
f : factorisation par (x-a)
g : plus grand commun divisieur
p : plus petit commun multiple
q : quit
Ecrivez le signe de l'opération que vous voulez:f
P(x) = 12x^6-4x^2
le polynome par factorisation est:
(x)*(x)*(12.00x^4-4.00)

4.9 Le plus grand commun diviseur de deux polynômes

Cette routine peut calculer le plus grand commun diviseur de deux polynômes par le division euclidienne

Comme l'axiome: f(x)=p1(x)*g(x)+r(x), f(x) et g(x) ont le commun diviseur avec g(x) et r(x).

$$\begin{cases} f(x)=g(x)q_1(x)+r_1(x), & \partial(r_1(x))<\partial(g(x)), \\ g(x)=r_1(x)q_2(x)+r_2(x), & \partial(r_2(x))<\partial(r_1(x)), \\ r_1(x)=r_2(x)q_3(x)+r_3(x), & \partial(r_3(x))<\partial(r_2(x)), \\ \cdots\cdots\cdots\cdots\cdots\cdots & \\ r_{k-2}(x)=r_{k-1}(x)q_k(x)+r_k(x), & \partial(r_k(x))<\partial(r_{k-1}(x)), \\ r_{k-1}(x)=r_k(x)q_{k+1}(x)+0, & r_{k+1}(x)=0. \end{cases} \tag{1.4.1}$$

Lemme de processus ici n'est pas introduit, les lecteurs peuvent explorer.

```
int x=0;
int va=0;
int i=0;

double reste2[CMONOME];
double reste[CMONOME];
double quotient2[CMONOME];
double quotient[CMONOME];

int max1=0;
int max2=0;
```

PS:

*1. **va** est une quantité intermédiaire pour determiner le dernier résultat.*

*2.Le tableau **reste** représente le dividende, donc le polynôme qui a le supérieur degré, **max1** est son plus grand degré； le tableau **quotient** représente le diviseur, donc le polynôme qui a le inférieur degré, **max2** est son plus grand degré； le tableau **reste2** représente le reste； le tableau **quotient2** représente le plus grand commun diviseur dont on a besoin finalement.*

D'abord, détermine les maximal exponents dans les deux polunomes.

```
if(coeff1[x]!=0)
    max1=x;
if(coeff2[x]!=0)
    max2=x;
```

Donne le polynôme qui a un supérieur dégre dans **reste** comme le dividende.

```
for(x=0;x<CMONOME;x++)
  reste2[x]=reste[x];
while(va==0)
  {
    for(x=0;x<CMONOME;x++)
      result[x]=quotient[x];


    division_euclidienne(reste,quotient,quotient2,reste2);
```

Ici, on détemine si chacun élément dans **reste2** est tout égale à 0.
Comme:

```
for(x=0;x<CMONOME;x++)
  {
    if(reste2[x]<0.00000000001 && reste2[x]>-0.00000000001)
      reste2[x]=0;
  }
for(x=0;x<CMONOME;x++)
  {
    if(reste2[x]!=0)
      i=i;
    else
      i++;

    if(i==CMONOME)
      va=1;
```

*PS : **va=1** est la condition d'arrêt, si on manque une fois de **i++** , donc finalement la longueur de **i** est inférieur à CMONOME, alors va=0, la circulation n'est pas arrêtée. Quand calculer tous les fois **i++** , la longueur de **i** est égale à CMONOME, ici **va** est égale à 1, la circulation sera arrêtée. i=0 (Initialise i).*

Quand il y a une valeur qui n'est pas égale à 0, on donne une nouvelle valeur et fait une circulation **while** à nouveau.

Comme:

```
if(va==0)
  {
   for(x=0;x <CMONOME;x++)
  {
     reste[x]=quotient[x];
     quotient[x]=reste2[x];
 }
```

Fonctionne en terminal:

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:g
P1(x) = 12x^6+4x^4
P2(x) = 2x^2
Le plus grand commun diviseur est:2.00x^2
```

4.10 Le plus petit commun multiple de deux polynômes

Cette routine utilise l'axiome: le plus petit commun multipe est égale à le quotient entre le produit de deux polynômes et le plus grand commun diviseur.

Calcule le plus grand commun multipe de deux polynômes.

pgcd(coeff1, coeff2, pgcd12)

Calcule le produit de deux polynômes.

multiplication(coeff1, coeff2, mu)

Le produit divise par le plus grand commun multipe.

division_euclidienne(mu, pgcd12, result, reste)

*PS:Ici, le residu **reste** doit être 0.*

Fonctionne en terminal:

```
        + : addition
        - : soustraction
        * : multiplication
        / : division
        d : derivation
        e : evaluation
        f : factorisation par (x-a)
        g : plus grand commun divisieur
        p : plus petit commun multiple
        q : quit

Ecrivez le signe de l'opération que vous voulez:p
P1(x) = 15x^6
P2(x) = 5x^2
Le plus petit commun mutiple est:15.00x^6
```

V Instructions d’utilization

1、Donnez les polynômes dont la variable est une lettre minuscule **x** , les coefficients sont tout entiers.
2、Donnez le terme **degré** par ^ . Arrêtez par **\n** .
3、Ne pas ajoutez de parenthèses.
4、Le maximal exponent des polynômes est inférieur à 100, la longueur par un monôme est inférieur à 20. (Vous pouvez modifier cette longueur par **#define**)
5、Utilisant la division euclidienne, on utilise P(1) divise P(2) tacitement.
6、Il n’y a que deux fractions décimale dans la division euclidienne et le plus grand commun diviseur.
7、Donnez les signes dans le menu que nous vous offrons.
8、Donnez **q** pour quitter.

VI La conclusion

1. Ce qui a été fait

Par ce projet, nous avons renforcé les points de connaissances suivants:

--Initialisation;
--Lecture et ecriture;
--Evaluation de p(x) par la methode de Horner pour un x donne;
--Addition;
--Soustraction;
--Multiplication;
--Derivation;
--Division euclidienne;
--Factorisation par (x-α),α un entier donne;
--plus grand commun diviseur de deux polynômes;
--plus petit commun multiple de deux polynômes.

2. Les difficultés que nous avons rencontrées

a. En réalité, après donner l'instruction dont vous avez besoin, on doit confirmer par **\n**, mais **\n** sera donné dans le tableau **getchar**, comme le premier caractère dans la circulation,en même temps, \n est la fin de donner,donc il saute le premier polynôme, directement à l'entrée du deuxième polynomial.

b. Dans le processus de programmation, on donne le polynôme par le tableau **char**, et on montre le coefficient par **int** finalement, nous rencontrons les difficultés dans le processus réel de coefficient polynomial extraites.

c. On ne sait pas bien les maths de connaissances comme Division euclidienne, Factorisation par (x-α), l'algorithme de plus grand commun diviseur de deux polynômes, l'algorithme de plus petit commun multiple de deux polynômes etc.

3. Les points qui peuvent être améliorés

a. Ce programme dans le processus d'entrée de polynômes, ne peut trouver qu'une erreur, mais pas tous les erreurs.

b. Factorisation est seulement validé pour les réels, pas pour les nombres complexes.

c. Le programme n'est pas assez simple et lisible.