



# 2021 PROJET RAPPORT

## INFO Générale



MESRAR  
HAMZA



mesrarhamza48  
@gmail.com



ENSA  
KHOURIBGA

Mini-Projet de programmation C  
sous L'encadrement de :  
Mr.MOSTAFA SAADI

18-Mars-2021



# Plan de rapport :

- I. Introduction sur programmation C.
- II. Définition des en-tête et fichiers < .h > utilisé.
- III. Plan d'exercice 1 :
  - ✓ Que fait ce programme ?
  - ✓ Les fonctions constituent ce programme.
  - ✓ Main code de ce programme.
  - ✓ Conclusion.
- IV. Plan d'exercice 2 :
  - ✓ Que fait ce programme ?
  - ✓ Les fonctions constituent ce programme.
  - ✓ Sous-programme Q-9 à Q-12
  - ✓ Main code de ce programme.
  - ✓ Conclusion.
- V. Conclusion.

## I. Introduction sur programmation C :

Le langage C est un langage de bas niveau dans le sens où il permet l'accès à des données que manipulent les ordinateurs (Bits, octets, adresses) et qui ne sont pas souvent disponibles à partir de langages évolués tels que **Fortran**, **Pascal** ou **ADA**. Le langage C a été conçu pour l'écriture du système même d'exploitation (plus de 90% du noyau du système **UNIX** est écrit en langage C).

## II. Définition des en-tête et fichier < .h > utilisé :

Dans ce projet, nous utilisons six bibliothèques avec six en-têtes, cinq sont déjà définis en langage C, Et le dernier est un manuscrite, nous allons les définir une par une dans la partie suivante.

*Maintenant la question est de savoir qu'est-ce qu'un en-tête?*

Un fichier d'en-tête est un fichier avec une extension **.h** qui contient des déclarations de fonction C et des définitions de macro à partager entre plusieurs fichiers source. Il existe deux types de fichiers d'en-tête: les fichiers que le programmeur Écrit, et les fichiers fournis avec votre compilateur.

Nous avons commencé avec le premier type est les en-têtes qui viennent avec le compilateur, nous avons déjà dit que nous utilisons cinq en-têtes de ce type, allons-y pour les présenter:

- Le premier est **<stdio.h>** qui signifie Standard Input-Output. Il contient les informations relatives aux fonctions d'entrée / sortie.
- Le second **<stdlib.h>** est l'en-tête de la bibliothèque standard à usage général du langage de programmation C qui comprend des fonctions impliquant l'allocation de mémoire, le contrôle de processus, les conversions et autres.
- Et le troisième est **<math.h>** conçu pour les opérations mathématiques de base. Fonctions de bibliothèque mathématique qui opèrent sur des entiers, tels que **puissance**, **sqrt**, etc....
- Le quatrième est **<conio.h>** est un fichier d'en-tête C utilisé principalement par les compilateurs **MS-DOS** pour fournir des entrées / sorties de console.
- Le cinquième est **<windows.h>** est un fichier d'en-tête spécifique à Windows pour les langages de programmation C et C++ qui contient des déclarations pour toutes les



fonctions de l'API Windows telles que `windows("cls")`, `windows("close")`, etc...

Et maintenant nous allons définir l'en-tête écrit à la main qui a un nom est « `menu.h` » Et il contient 3 types de fonctions, le premier type est les fonctions de couleur, le deuxième type est les fonctions de menu et le troisième est une fonction unique pour le retard.

Voyons maintenant comment déclarer ce fichier d'en-tête. Tout d'abord, nous devons utiliser un `#include`, c'est une directive de préprocesseur. Et mettez le nom de l'en-tête entre `<>` si l'en-tête est fourni avec le compilateur mais si l'en-tête est écrit à la main, nous devrions utiliser `" "`.

```
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <windows.h>
```

En tête fournis avec  
le compilateur

```
#include "menu.h"  ← l'en-tête est écrit à la main.
#define DegreMax 20 ← déclaration des valeurs cts.
```

➤ Exemple des fonctions de "menu.h" :

```
void delay(int j)
{
    int i,k;
    for(i=0;i<j;i++)
        k=i;
}

// _____ COLORS _____

void red(void){
    system("color 0");
    printf("\033[0;31m");
}

void black(void){
    system("color 0");
    printf("\033[0;30m");
}

void green(void){
    system("color 0");
    printf("\033[0;32m");
}
```

### III. Traitement de l'exercice I :

#### Que fait ce programme ?

Le Programme de l'exercice 1 permet d'effectuer des calculs et traitement sur les polynômes.  
Ce programme contient plusieurs fonctions, chaque fonction joue un rôle C.-à-d. fait un calcul ou un traitement.

#### Les composants du Programme :

- ✓ Les en-tête
- ✓ Structure
- ✓ Fonctions
- ✓ Main code

#### Explication de chaque composant :

❖ **Les en-tête** (voir l'introduction précédent).

❖ **Structure :**

Dans ce programme, nous utilisons une structure avec **Pol** comme nom, et elle contient deux membres, le premier est un entier présent le degré du polynôme et le second est une table des **floats** pour stocker les coefficients du polynôme. (Voir la figure 1).


```
typedef struct {  
    int degre;  
    float coeff[DegreMax+1];    //Déclaration de structure utilisé  
} pol;
```

Figure 1

❖ **Les fonctions :**

1) **Void NulPoly (pol\* p) :**

Fonction qui permette de renvoyant un Polynôme Nul.

 Le code ce fonction :

NB : ce bloc d'instruction si juste des affectations.

Le degre veut la valeur **-1** et en peut mettre tous les coefficients à la valeur **0**.



```
void NulPoly(pol* p){
    int i;
    p->degre = -1;
    for(i=0;i<DegreMax+1;i++){
        p->coeff[i] = 0;
    }
}
```

☞ L'exécution sera comme Ça :

```
P(X) =
-----
Process exited after 0.04304 seconds with return value 0
Press any key to continue . . .
```

2) **int** **verif** (pol p) :

Est une fonction qui permet de vérifier si un polynôme est NULL ou non :

- Si le polynôme est nul la fonction **return** la valeur **1**.
- Si le polynôme n'est pas un polynôme nul la fonction **return** la valeur **0**.

Un polynôme est non nul s'il existe au moins un coefficient différent de **0**, ou degré vaut **-1**.

☞ Le code de cette fonction :

```
int verif(pol p){
    int i,bol=1;
    for(i=0;i<p.degre+1;i++){
        if(p.coeff[i] != 0){
            bol = 0;
            break;
        }
    }
    return bol;
}
```



Le bol sera initialisé par 1 et en vérifiant s'il existe un coefficient non nul s'il existe, en affecte 0 à la variable bol et sortir avec break, et return bol.

🔗 L'exécution sera comme Ça :

```
Entrer le polynome P :
entrer le degre de polynome (max 20) : 3
entrer l'coeff A0 : 0
entrer l'coeff A1 : 0
entrer l'coeff A2 : 0
entrer l'coeff A3 : 0

P(X) = 0.00 * X^0 + 0.00 * X^1 + 0.00 * X^2 + 0.00 * X^3

verification en cours...

Le polynome P est NULL!
-----
Process exited after 5.026 seconds with return value 0
Press any key to continue . . .
```

3) **void** SaisiePoly(pol \*p) ET **void** voirPoly(pol p) :

🔗 **void** SaisiePoly(pol \*p) :

Une fonction de saisie des polynômes qui demande premièrement le degré de polynôme et après demande les coefficients de polynôme un par un et stocker chaque valeur dans la structure pol. Le degré de polynôme va stocker dans le membre p.degre Et les coefficients vont stocker dans le tableau coeff par p.coeff[i].

🔗 **void** voirPoly(pol p) :

Une procédure qui prend un polynôme en paramètre et qui affiche le contenu de chaque membre mais elle est bien organisé et plus lisible.



### Le code de deux fonctions :

```
void SaisiePoly(pol *p){  
    int i;  
  
    printf("entrer le degre de polynome (max 20) : ");  
    scanf("%d",&p->degre);  
    for(i=0;i<p->degre+1;i++){  
        printf("entrer l'coeff A%d : ",i);  
        scanf("%f",&p->coeff[i]);  
    }  
}  
  
void voirPoly(pol p){  
    int i;  
    printf("P(X) = ");  
    for(i=0;i<p.degre+1;i++){  
        printf("%.2f * X^%d ",p.coeff[i],i);  
        if(i<p.degre)  
            printf(" + ");  
    }  
}
```

### L'exécution sera comme Ça :

```
La saisie et l'affichage d'un Polynome :  
Le Polynome sera de type : A0*X^0 + A1*X^1 + ..... + An*X^n .!  
  
entrer le degre de polynome (max 20) : 3  
entrer l'coeff A0 : -4.5  
entrer l'coeff A1 : 2  
entrer l'coeff A2 : 0.36  
entrer l'coeff A3 : 1  
  
P(X) = -4.50 * X^0 + 2.00 * X^1 + 0.36 * X^2 + 1.00 * X^3  
-----  
Process exited after 15.58 seconds with return value 0  
Press any key to continue . . . _
```

#### 4) **void voirPoly\_2(pol p, float x) :**

Cette fonction permet de calculer  $P(x)$  d'un certain  $X$  donné en paramètre.  
Le polynôme et aussi passe en paramètre.






Le principe de calcul est de calculer la somme de  $A_i \times X^i$  avec  $i$  à 0 jusqu'à degré de  $P$ .

Tq :

- ◆  $A_i$  est le coefficient.
- ◆  $i$  est le degré Du coefficient  $A_i$
- ◆  $X$  est le nombre entré.

 Le code de fonction :

```
void voirPoly_2(pol p, float x){  
    int i;  
    float s=p.coeff[0];  
  
    for(i=1;i<=p.degre;i++){  
        s = s + (p.coeff[i] * pow(x,i));  
        //printf("%f\t",s);  
    }  
  
    printf("P(%.2f) = %.2f",x,s);  
}
```

 L'exécution sera comme Ça :

```
Calculation des images de X :  
  
entrer le degre de polynome (max 20) : 2  
entrer l'coeff A0 : -1  
entrer l'coeff A1 : 0  
entrer l'coeff A2 : 1  
  
P(X) = -1.00 * X^0 + 0.00 * X^1 + 1.00 * X^2  
  
entre un nombre X : 5  
P(5.00) = 24.00  
-----  
Process exited after 16.39 seconds with return value 0  
Press any key to continue . . .
```

5) **void AddPoly(pol p1, pol p2) :**

Est une fonction qui permette de calculant le produit de deux polynômes  $P$  et  $Q$ .



On a trois cas pour l'addition de polynôme :

- \* 1<sup>er</sup> cas : Si le degre de P1 supérieur à degre de P2.
- \* 2<sup>ème</sup> cas : Si le degre de P2 supérieur à degre de P1.
- \* 3<sup>ème</sup> cas : Si le degre de P1 égale à degre de P2.

Le degre de la somme de deux polynôme égale à le plus grand degre entre le degre de P et Q.

On Pose  $S(X) = P(X) + Q(X)$  et  $k = \sup(\deg P, \deg Q)$ .

On applique la relation :  $S(X) = \sum_{i=0}^k (A_i + B_i) \times X^i$


Tq :

- ◆  $A_i$  est les coefficient de P.
- ◆  $B_i$  est les coefficient de Q.
- ◆  $k$  egale a le  $\max(\deg P, \deg Q)$ .

 Voici le code de cette fonction :

```
void AddPoly(pol p1, pol p2){  
  
    pol p;  
    int i;  
  
    if(p1.degre > p2.degre){  
        p.degre = p1.degre;  
        p.coeff[p1.degre] = p1.coeff[p1.degre];  
        for(i=0;i<p.degre;i++){  
            p.coeff[i] = (p1.coeff[i]+p2.coeff[i]);  
        }  
    }else if(p1.degre == p2.degre){  
        p.degre = p1.degre;  
        for(i=0;i<p.degre+1;i++){  
            p.coeff[i] = (p1.coeff[i]+p2.coeff[i]);  
        }  
    }else{  
        p.degre = p2.degre;  
        p.coeff[p2.degre] = p2.coeff[p2.degre];  
        for(i=0;i<p.degre;i++){  
            p.coeff[i] = (p1.coeff[i]+p2.coeff[i]);  
        }  
    }  
    voirPoly(p);  
}
```



 L'exécution sera comme Ça :

```
L'addition des Polynome :  
  
Entrer Le Polynome P :  
entrer le degre de polynome (max 20) : 2  
entrer l'coeff A0 : 1  
entrer l'coeff A1 : 2  
entrer l'coeff A2 : 3  
  
P(X) = 1.00 * X^0 + 2.00 * X^1 + 3.00 * X^2  
  
Entrer Le Polynome Q :  
entrer le degre de polynome (max 20) : 1  
entrer l'coeff A0 : -1  
entrer l'coeff A1 : 2  
  
P(X) = -1.00 * X^0 + 2.00 * X^1  
  
La polynome Somme est :  
P(X) = 0.00 * X^0 + 4.00 * X^1 + 3.00 * X^2  
-----  
Process exited after 24.1 seconds with return value 0  
Press any key to continue . . .
```

6) **void** MultPoly(pol p1, pol p2) :

Cette fonction est une procédure qui calculant le produit de deux polynôme P et Q.  
Le principe de calcul par applique la relation de produit de deux polynômes C.-à-d. :

On pose :

$$k = \deg(P) + \deg(Q) = n + q \quad \text{avec : } n = \deg(P) \\ \text{et } q = \deg(Q)$$

On sait que La degre de produit de deux polynômes égale à la somme de degre c.-à-d. :

$$\deg(P \times Q) = \deg(P) + \deg(Q) \quad \text{qui egale à } k.$$

Puis on applique la relation suivante :

$$(P \times Q) = \sum_{i=0}^{n+q} \left( \sum_{j=0}^i A_j \times B_{i-j} \right) \times X^i$$

Avec :

- $A_i$  est les coefficients de  $P$ .
- $B_i$  est les coefficient de  $Q$ .
- $X^i$  est le variable.
- $n$  est le degre de  $P$ .
- $q$  est le degre de  $Q$ .

La méthode utilisé pour le code C est de stocké  $(\sum_{j=0}^i A_j \times B_{i-j})$  sur un tableau  $C[i]$ , Puis ce tableau sera contient les coefficients de Polynôme produit et en fin, j'ai utilisé la fonction d'affichage simple en prend chaque Coefficient  $C[i]$  on multiplions par  $X^i$  Avec  $i$  allant de  $0$  à  $n + q = \text{deg}(P) + \text{deg}(Q)$ .


 Voici le code de fonction :

```
void MultPoly(pol p1, pol p2){
    pol p;
    int i,j,d=p1.degre+p2.degre;
    float C[d],s;

    for(i=0;i<d+1;i++){
        s=0;
        for(j=0;j<i+1;j++){
            s+=(p1.coeff[j]*p2.coeff[i-j]);
        }
        C[i] = s;
    }

    printf("P(X) = ");
    for(i=0;i<d+1;i++){
        printf("%.2f * X^%d",C[i],i);
        if(i<d)
            printf(" + ");
    }
}
```

NB : Le code source contient plusieurs commentaires qui explique tous les choses.

 Le fichier code source est uploader dans le dossier de projet.



🔗 L'exécution sera comme Ça :

```
Le Produit des Polynome :

Entrer Le Polynome P :
entrer le degre de polynome (max 20) : 1
entrer l'coeff A0 : 1
entrer l'coeff A1 : 1

P(X) = 1.00 * X^0 + 1.00 * X^1

Entrer Le Polynome Q :
entrer le degre de polynome (max 20) : 1
entrer l'coeff A0 : -1
entrer l'coeff A1 : 1

P(X) = -1.00 * X^0 + 1.00 * X^1

La polynome Produit est :
                P(X) = -1.00 * X^0 + 0.00 * X^1 + 1.00 * X^2
-----
Process exited after 7.986 seconds with return value 0
Press any key to continue . . .
```

❖ Main CODE :

Le main Code ou fonction main est une fonction permet de tester les fonctions précédent ainsi que simplifier les choses, et l'exécution de chaque fonction à part.

La forme général ou syntaxe du fonction main il vient Comme suit :

```
int main () {

    Déclaration des variables ;

    Bloc d'instruction ;

    Return 0 ;

}
```



## Main code de ce programme :

```
int main(){

    pol A,B;
    int i,choix;
    float x;

    menu1(); //procEDURE pour afficher Le menu (pr  d  fini dans "menu.h")
    scanf("%d",&choix); //lire un choix entrer l'utilisateur
    system("cls"); // fonction d  finie dans windows.h> pour videz l'  cran
    switch(choix){
        case 1: // case 1 Pour saisie et affichage des Polynomes
            printf("La saisie et l'affichage d'un Polynome :*\n");
            printf("Le Polynome sera de type : A0*X^0 + A1*X^1 + ..... + An*X^n .!\n\n");
            SaisiePoly(&A);
            printf("\n");
            voirPoly(A);
            break;
        case 2: // case 2 qui permet de calculer P(x) d'un certain X
            printf("Calcul des images de X : \n\n");
            SaisiePoly(&A);
            printf("\n");
            voirPoly(A);
            printf("\n\n\t\t");
            printf("entre un nombre X : ");
            scanf("%f",&x);
            voirPoly_2(A,x);
            break;
        case 3: // case 3 pour l'addition des polynomes
            printf("L'addition des Polynome : \n\n");
            SaisiePoly(&A);
            printf("\n");
            voirPoly(A);
            printf("\n\n");
            SaisiePoly(&B);
            printf("\n");
            voirPoly(B);
            printf("\n\nLa polynome Somme est : \n\t\t");
            AddPoly(A,B);
            break;
        case 4: // case 4 pour la multiplication des polynomes
            printf("Le Produit des Polynome : \n\n");
            SaisiePoly(&A);
            printf("\n");
            voirPoly(A);
            printf("\n\n");
            SaisiePoly(&B);
            printf("\n");
            voirPoly(B);
            printf("\n\nLa polynome Produit est : \n\t\t");
            MultPoly(A,B);
            break;
        case 5: // case 5 Pour Les informations personnel
            printf("\n\n\n\t\t\t\t\tHAMZA MESRAR");
            break;
        case 0: // case 0 pour Sortir est fermer Le Program
            printf("\n\n\t\t\t\t\tMESRAR HAMZA vous remercie de votre visite! _____");
            getch();
            break;
        default : // pour incorrect case
            main(); // ex  cution fonction main a nouveau
    }

    return 0;
}
```



### ^ Explication de ce code :

Ce code consiste à quatre choses : déclaration de variable, fonction menu, écrire et lire à l'écran, et finalement une boucle **switch () case**.

#### ◆ Déclaration des variables :

J'ai utilisé deux variables de type structure pol pour stocker les informations de deux polynômes, un variable entier « choix » pour prendre le choix d'utilisateur, et un **float x** pour utiliser dans la fonction voirPoly\_2().

#### ◆ Fonction menu () :

 Code de fonction :

```
void menu1(void){  
    blue();  
    printf("\n\n\t\t\t");printf("*****\n");  
    printf("\t\t\t");printf("*\t\tPolynme Section\t\t*\n");  
    printf("\t\t\t");printf("*****\n\n\n");  
    printf("\t\t\t");printf("1)- Saisir et affichage un polynome.\n");  
    printf("\t\t\t");printf("2)- L'image d'un X par un polynome.\n");  
    printf("\t\t\t");printf("3)- Addition de deux polynomes.\n");  
    printf("\t\t\t");printf("4)- Multiplication de deux Polynomes.\n");  
    printf("\t\t\t");printf("5)- Developpeur Informations.\n");  
    printf("\t\t\t");printf("0)- Sortir.!\n");  
    printf("\n\n\n\t\t\t");printf("____Entrer votre choix : ");  
    white();  
}
```

- \* Est une procédure d'affichage prédéfini dans « menu.h » (menu.h est un en-tête écrit à la main) consiste juste par des printf pour afficher un menu lisible (voir la figure en bas).
- \* Fonction blue() et white() si juste deux fonctions pour changer la couleur display en l'écran en de console.
- \* NB : les deux fonctions précèdent son prédéfini dans l'en-tête "menu.h"



🔗 Le menu s'affiche comme ça :

```
*****
*                               *
*           Polynme Section     *
*                               *
*****

1)- Saisir et affichage un polynome.
2)- L'image d'un X par un polynome.
3)- Addition de deux polynomes.
4)- Multiplication de deux Polynomes.
5)- Developpeur Informations.
0)- Sortir.!!

_____Entrer votre choix : _
```

💧 Écrire et lire à l'écran le choix d'utilisateur :

Pour faire ça et très simple on utilise juste un printf pour afficher un message et scanf pour stocker le choix d'utilisateur dans une variable.

```
Printf ("_____Entrer votre choix : ");
Scanf ("%d", &choix);
```

💧 Boucle switch ( ) case :

**switch()** Est une instruction nous permettons d'exécuter un bloc de code parmi de nombreuses alternatives. Pour chaque **case** ont exécuté un bloc d'instruction défèrent, par le choix de l'utilisateur (voir le code en haut).





## IV. Traitement de l'exercice II :

### + Que fait ce programme ?

Le programme de l'exercice 2 permet d'effectuer des calculs et traitement sur les points et les vecteurs du plan. Ce programme contient plusieurs fonctions, chaque fonction fait un calcul ou un traitement.

### + Les composants du Programme :

- ✓ Les en-tête
- ✓ Structures
- ✓ Fonctions
- ✓ Main code

### + Explication de chaque composant :

#### ❖ Les en-tête :

Si les même en-tête que j'ai utilisée dans le premier programme la seule différence est la déclaration de constant.

En utilisé `#define Max_Pts 100` dans ce programme.

#### ❖ Les Structures :

Dans ce Programme, j'ai utilisé trois structures, une pour stocker les informations d'un point, et l'autre pour stocker les informations d'un vecteur, et la dernière pour stocker deux informations [le nom d'un point et la distance entre ce point et l'origine 0 du repère]

```
struct point {char Nom ; float x ; float y ;} ;  
typedef struct point spt ;  
  
struct vecteur {float x ; float y ;} ;  
typedef struct vecteur svect ;  
  
struct Distance {char Nom ; float dis ;} T1,tab ;  
typedef struct Distance sdis ;
```

Le 1<sup>er</sup> pour les points.

La 2<sup>ème</sup> pour les vecteurs.

La 3<sup>ème</sup> pour Les distance (utilisable dans sous-programme)



## ❖ Les fonctions :

1) spt saisie(**void**) ET **void** afficher(spt P) :

✂ spt saisie(**void**) :

Une fonction de saisie des points avec aucun paramètre a donné, qui demande trois paramètres à entrer par l'utilisateur [le nom d'un point, et l'abscisse x, et l'ordonnée y]  
Et stocker ces valeurs dans la structure spt.  
Puis **return** une variable de type spt prêt à l'emploi.

✂ **void** afficher(spt P) :

Une procédure qui prend un point en paramètre d'entrer et qui affiche le contenu de chaque membre de structure spt P, mais bien organisé et plus lisible.

📄 Le code de deux fonction :

```
spt saisie(void){
    spt P;


    printf("Donnez le nom de point : ");
    scanf("%1s",&P.Nom);
    //scanf ("%s^[^\\n]");
    printf("Donnez X : ");
    scanf("%f",&P.x);
    printf("Donnez Y : ");
    scanf("%f",&P.y);
    printf("\n");

    return P;
}

void afficher(spt P){
    printf("Le point %c = ( %.2f , %.2f )\n",P.Nom,P.x,P.y);
}
```

NB : on utilise **%1s** à la place de **%c** pour éviter les Commons erreurs par exemple : (scanf avec **%c** exécuté une seule fois dans une boucle).




 L'exécution sera comme ça :

```
Saisir et affichage d'un point :
Donnez le nom de point : M
Donnez X : 0.236
Donnez Y : -2.5

Le point M = ( 0.24 , -2.50 )

-----
Process exited after 8.446 seconds with return value 0
Press any key to continue . . .
```

2) svect ss(**void**) ET void aff(svect v) :

 svect ss(**void**) :

Une fonction de saisie des vecteurs avec aucun paramètre a donné, qui demande deux paramètres à entrer par l'utilisateur [l'abscisse x d'un vecteur, et l'ordonnée y d'un vecteur]  
Et stocker cette valeur dans la structure svect.  
Puis **return** une variable de type svect prêt à l'emploi.

 void aff(svect v) :

Une procédure qui prend un vecteur en paramètre d'entrer et qui affiche le contenu de chaque membre de structure svect v, mais bien organisé et plus lisible.

 Le code de deux fonctions :

```
svect ss(void){
    svect V;

    printf("Donnez X : ");
    scanf("%f",&V.x);
    printf("Donnez Y : ");
    scanf("%f",&V.y);

    return V;
}

void aff(svect v){
    printf("Les cordonner de vecteur est : ( %.2f , %.2f )\n",v.x,v.y);
}
```

 L'exécution sera comme ça :

```
Saisie et affichage d'un vecteur :  
Donnez X : 27.32  
Donnez Y : 6  
  
Les cordonner de vecteur est : ( 27.32 , 6.00 )  
  
-----  
Process exited after 5.646 seconds with return value 0  
Press any key to continue . . .
```

3) **float** distance(spt p1, spt p2) :

Cette fonction permet de calculer la distance entre deux points A et B, passant par paramètre à la fonction, par appliquant la relation :

$$\|AB\| = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

En calculons la distance.

Et en fin **return** la distance sous la forme d'un **float**.

 Le code de la fonction :

```
float distance(spt p1, spt p2){  
    return (sqrt(pow((p1.x-p2.x),2)+pow((p1.y-p2.y),2)));  
}
```

#en code C :

NB : après la fonction en va faire une saisie de deux points pour le passer à la fonction pour calculer la distance

Puis en faire un printf avec **%f** pour afficher le contenu retourner par la fonction.



🔗 L'exécution sera comme ça :

```
Calculation des Distance entre deux points :
Donnez le nom de point : A
Donnez X : 2
Donnez Y : 0

Le point A = ( 2.00 , 0.00 )

Donnez le nom de point : B
Donnez X : 3
Donnez Y : -1

Le point B = ( 3.00 , -1.00 )

La distance Entre A et B est : 1.41
-----
Process exited after 21.84 seconds with return value 0
Press any key to continue . . . _
```

4) **void Deplacer(spt\* p, float dx, float dy) :**

Est une fonction permet de déplacer un point sur le plan (Ox, Oy) par un déplacement élémentaire dx et dy passant par paramètre à la fonction en plus d'un point A de type spt\*.

On sait que pour déplacer un point il se fait d'ajouté dx à x et ajouté dy à y.  
D'où : on fait  $x+=dx$  et  $y+=dy$ .

📄 Voici le code de fonction :

```
void Deplacer(spt* p, float dx, float dy){
    p->x+=dx;
    p->y+=dy;

    printf("Les nouveau cordonner de %c est : ( %.2f , %.2f )",p->Nom,p->x,p->y);
}
```



NB : on a le point est un pointeur alors en utilisé -> à la place de.

☞ L'exécution sera comme ça :

```
Deplacement des point :
Entrer Les cordonner de point :
Donnez le nom de point : M
Donnez X : -4
Donnez Y : 1.3

Le point M = ( -4.00 , 1.30 )

    entrer la valeur que vous allez deplacez Dans l'axe Ox : 2.3
    entrer la valeur que vous allez deplacez Dans l'axe Oy : 1.7

    Les nouveau cordonner de M est : ( -1.70 , 3.00 )
-----
Process exited after 19.75 seconds with return value 0
Press any key to continue . . .
```

5) **float** ProdScal(svect v1, svect v2) :

Cette fonction permet de calculer le produit scalaire entre deux vecteurs V1 et V2, passant par paramètre à la fonction, par appliquant la relation :

On pose :  $\vec{v}_1(x_1, y_1)$  et  $\vec{v}_2(x_2, y_2)$

$$\vec{v}_1 \cdot \vec{v}_2 = (x_1 \times x_2) + (y_1 \times y_2)$$


En calculons le produit scalaire.

Et en fin **return** le résultat sous la forme d'un **float**.


#en C code :

NB : après la fonction en va faire une saisie de deux vecteurs pour le passer à la fonction pour calculer le produit scalaire. Puis en faire un printf avec %f pour afficher le contenu retourner par la fonction.



 Le code de la fonction :

```
float ProdScal(svect v1, svect v2){  
    return ((v1.x*v2.x) + (v1.y*v2.y));  
}
```

 L'exécution sera comme ça :

```
Calculation de produit scalaire des vecteur :  
Entrer Les cordonner du 1er Vecteur :  
Donnez X : 2  
Donnez Y : 1.5  
  
Les cordonner de vecteur est : ( 2.00 , 1.50 )  
  
Entrer Les cordonner du 1eme Vecteur :  
Donnez X : -3.2  
Donnez Y : 4  
  
Les cordonner de vecteur est : ( -3.20 , 4.00 )  
  
Le produit scalair de deux vecteur est : -0.40  
-----  
Process exited after 10.98 seconds with return value 0  
Press any key to continue . . .
```

6) `int Colineaires(spt p1, spt p2, spt p3) :`

Cette fonction permet de vérifie si trois points du plan passant comme un paramètre à la fonction est colinéaire, Si oui la fonction **return** la valeur 1, Sinon la fonction **return** la valeur 0. Pour faire ça on applique la relation :  
On a trois points A, B, et C sont Colinéaire si :  
Soit les vecteurs BA et CA compris par les points A, B, et C entré.  
si  $x_{BA} \times y_{CA} = y_{BA} \times x_{CA}$  alors les trois points sont colinéaire.

J'ai juste copie cette relation a le bloc code comme tu vois dans la figure en bas.



### Le code de fonction :

Ces lignes pour  
calculer les  
coordonnées du  
vecteur utilisé

```
int Colineaires (spt p1, spt p2, spt p3){  
  
    svect u1,u2;  
    int bol=0;  
  
    u1.x = p2.x-p1.x;  
    u1.y = p2.y-p1.y;  
    u2.x = p3.x-p1.x;  
    u2.y = p3.y-p1.y;  
  
    if((u1.x*u2.y) == (u1.y*u2.x))  
        bol = 1;  
  
    return bol;  
}
```

### L'exécution sera comme ça :

```
Entrer Les cordonner du 1er point :  
Donnez le nom de point : A  
Donnez X : 0  
Donnez Y : 4  
  
Le point A = ( 0.00 , 4.00 )  
  
Entrer Les cordonner du 2eme point :  
Donnez le nom de point : B  
Donnez X : 0  
Donnez Y : 12.53  
  
Le point B = ( 0.00 , 12.53 )  
  
Entrer Les cordonner du 3eme point :  
Donnez le nom de point : C  
Donnez X : 0  
Donnez Y : -15  
  
Le point C = ( 0.00 , -15.00 )  
  
Les Trois points est colineaires!  
-----  
Process exited after 40.39 seconds with return value 0  
Press any key to continue . . .
```






7) **void AffichEquatCartesienne**(spt p1, spt p2) :

Est une procédure prend deux points A et B en paramètre et affiche l'équation cartésienne de la droite (AB).


On a l'Equation d'un droit (AB) est :  $y = a \times x + b$

$$Tq : a = \frac{(y_B - y_A)}{(x_B - x_A)}$$

et  $b = y_A - (a \times x_A)$  Ou  $b = y_B - (a \times x_B)$  sont les mêmes.

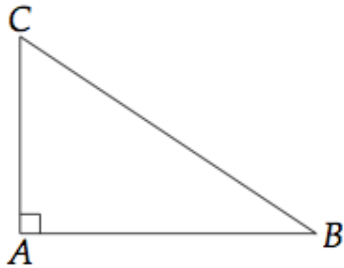
 Voici le code de fonction :

```
void AffichEquatCartesienne (spt p1, spt p2){  
    float a,b;  
  
    a = (p2.y-p1.y)/(p2.x-p1.x);  
    b = p1.y - (a*p1.x);  
  
    printf("L'equation cartesienne de la droite (%c%c) est : Y = %.2f*X + %.2f",p1.Nom,p2.Nom,a,b);  
}
```

 L'exécution sera comme ça :

```
Entrer Les cordonner du 1er point :  
Donnez le nom de point : M  
Donnez X : 1  
Donnez Y : 0  
  
Le point M = ( 1.00 , 0.00 )  
  
Entrer Les cordonner du 2eme point :  
Donnez le nom de point : N  
Donnez X : -2  
Donnez Y : 3  
  
Le point N = ( -2.00 , 3.00 )  
  
L'equation cartesienne de la droite (MN) est : Y = -1.00*X + 1.00  
-----  
Process exited after 20.95 seconds with return value 0  
Press any key to continue . . .
```





8) `int triangleRectangle(spt p1, spt p2, spt p3) :`

Cette fonction permet de vérifier si trois points du plan passant comme un paramètre à la fonction est former un triangle rectangle, Si oui la fonction **return** la valeur 1, Sinon la fonction **return** la valeur 0.

Pour faire ça on applique le théorème de Pythagore :

Si on a :  $BC^2 = AB^2 + AC^2$  Alors le triangle ABC est rectangle en A.

Mais dans le code on applique ce théorème sur chaque point par ce que ne connaît pas la tête de triangle.

 Voici le code de fonction :

```
int triangleRectangle(spt p1, spt p2, spt p3){
    int bol=0;
    float a,b,c;

    a = distance(p1,p2);
    b = distance(p2,p3);
    c = distance(p1,p3);
    //printf("%f\t%f\t%f\n",a,b,c);
    if( pow(b,2) == (pow(a,2)+pow(c,2)+0.1)){
        bol=1;
    }else if(pow(c,2) == (pow(a,2)+pow(b,2))){
        bol=1;
    }else if(pow(a,2) == (pow(b,2)+pow(c,2))){
        bol=1;
    }

    return bol;
}
```

NB : avant la fonction on utilise la fonction de saisie pour créer trois points dans le plan, et après la fonction on utilise une condition **if** pour connaître quoi **return** la fonction 1 ou 0. Puis un **printf** pour afficher "Le Triangle est Rectangle!" si la valeur est 1 et le contraire si la valeur est 0.



🔊 L'exécution sera comme ça :

```
Entrer Les cordonner du 1er point :  
Donnez le nom de point : A  
Donnez X : 0  
Donnez Y : 0  
  
Le point A = ( 0.00 , 0.00 )  
  
Entrer Les cordonner du 2eme point :  
Donnez le nom de point : B  
Donnez X : 0  
Donnez Y : 3  
  
Le point B = ( 0.00 , 3.00 )  
  
Entrer Les cordonner du 3eme point :  
Donnez le nom de point : C  
Donnez X : -3  
Donnez Y : 0  
  
Le point C = ( -3.00 , 0.00 )  
  
Le Triangle est Rectangle!  
-----  
Process exited after 25.02 seconds with return value 0  
Press any key to continue . . .
```

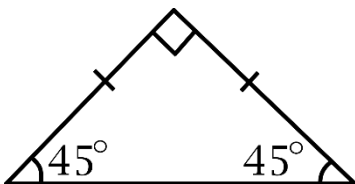
9) `int triangleIsocele(spt p1, spt p2, spt p3) :`


Cette fonction permet de vérifie si trois points du plan passant comme un paramètre à la fonction est former un triangle isocèle, Si oui la fonction **return** la valeur 1, Sinon la fonction return la valeur 0.

On sait qu'un triangle isocèle si deux de ces cotes est égale c.-à-d. :

$$AB = AC \Rightarrow \text{distance}(A,B)=\text{distance}(A,C).$$

Le principe de ce code est de calcule les deux distances et fait une condition **if** si les deux distances sont égales, la fonction **return** la valeur 1, Sinon la fonction **return** la valeur 0.



 Voici le code de fonction :

```
int triangleIsocele(spt p1, spt p2, spt p3){  
    int bol=0;  
    float P2P1,P2P3;  
  
    P2P1 = distance(p2,p1);  
    P2P3 = distance(p2,p3);  
  
    if(P2P1 == P2P3)  
        bol=1;  
  
    return bol;  
}
```

✗ Comme le NB de fonction précédent !

 L'exécution sera comme ça :

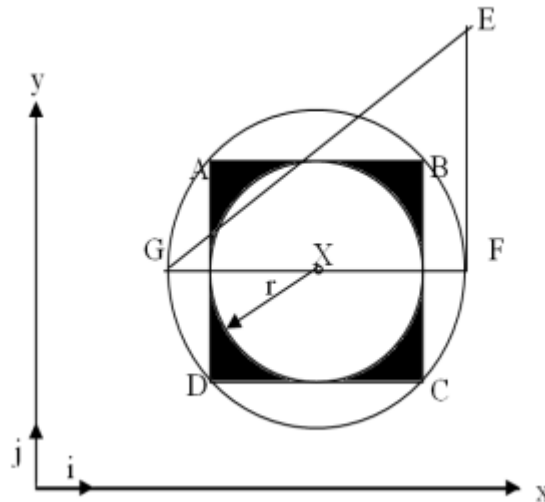
```
Entrer Les cordonner du 1er point :  
Donnez le nom de point : a  
Donnez X : 0  
Donnez Y : 3  
  
Le point a = ( 0.00 , 3.00 )  
  
Entrer Les cordonner du 2eme point :  
Donnez le nom de point : b  
Donnez X : 0  
Donnez Y : 0  
  
Le point b = ( 0.00 , 0.00 )  
  
Entrer Les cordonner du 3eme point :  
Donnez le nom de point : v  
Donnez X : -3  
Donnez Y : 0  
  
Le point v = ( -3.00 , 0.00 )  
  
Le Triangle est Isocele!  
-----  
Process exited after 19.65 seconds with return value 0  
Press any key to continue . . .
```



### ❖ Sous-programme Q9-Q12 :

Ce sous-programme est un programme complémentaire à l'exercice II qui traite les question 9 à 12. J'ai choisi de fait tout seul pour bien organiser les choses.

Premièrement le programme est à propos de l'étude de ce schéma :



En forme de 4 questions :

- Question d'analyse qui dit, comment peut tracer ce schéma à partir de deux points Z et Q, et le rayon  $r$ , avec détermination de ces points.
- Calcul de la surface de triangle GEF avec une fonction `float SurfaceGEF(spt Z, spt Q, float r)`, qui `return` la surface.
- Une fonction qui permet de trier un tableau de 100 points selon la distance des points à l'origine 0 du repère.
- Tracer ce schéma à partir de les deux point Z et Q (trouver dans a), et le rayon  $r$ .

Nous commençons par la démonstration de premier question et trouve les deux points Z et Q.

### ✍ Démonstration :

- ☒ Dans ce schéma on connaît juste le rayon  $r$ .





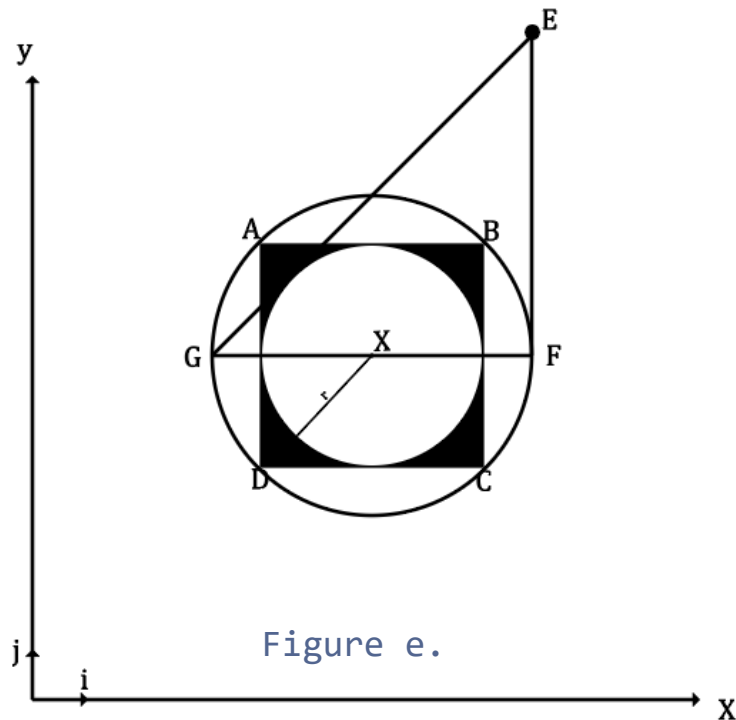


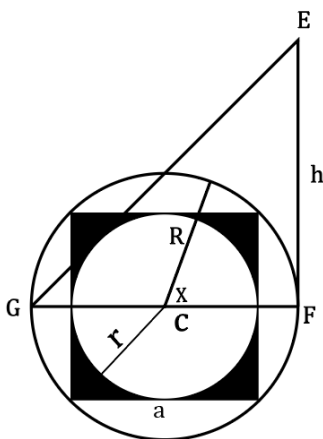
Figure e.

✍ Calcule de la surface de EFG :

On sait dans le schéma que le triangle rectangle alors la surface est égale à le produit de longueur de ces cotes adjacentes divisé par 2, C.-à-d. :

$$S = \frac{FE \times FG}{2}$$

Avec FG égale a le diamètre du grand cercle qui égale à la diagonale du carré ABCD qui égale de plus :



$$S = c \cdot h / 2$$

$$\begin{cases} a = 2 \times r \\ a = \frac{2 \times R}{\sqrt{2}} \end{cases} \Rightarrow 2 \times R = \frac{2 \times R}{\sqrt{2}} \Rightarrow R = \sqrt{2} \times r$$

Et on a le diamètre du carrée est :

$$c = a \times \sqrt{2} = 2 \times \sqrt{2} \times r$$

$$\text{D'où : } FG = 2 \times \sqrt{2} \times r$$



Et FE si juste calcule de distance car le point E est donné et le point F par l'analyse précédent est égale a :

$$F = (x, y) = (x_E, y_X)$$

D'où on calcule la distance entre E et F :


Si on utilise la fonction distance déjà crée dans le programme.

Ou on utilise la formule :

$$FE = \sqrt{(x_E - x_F)^2 + (y_E - y_F)^2}$$

D'où :

$$S = \frac{FE \times FG}{2} = \frac{\sqrt{(x_E - x_F)^2 + (y_E - y_F)^2} \times 2\sqrt{2} \times r}{2}$$

 Le code de fonction :

```
float SurfaceGEF (spt Z, spt Q, float r){  
    float S;  
    spt F;  
  
    F.Nom = 'F';  
    F.x = Q.x;  
    F.y = Z.y;  
  
    S = (distance(F,Q)*(2*sqrt(2)*r)) / 2;  
  
    return S;  
}
```

 L'exécution sera comme ça :

NB : toujours Le même, comme les fonctions précédent. Avant un saisie et après un printf





```

Calculution Du surface de Triangle GEF :
    Entrer Les cordonner du point Z :
Donnez le nom de point : E
Donnez X : 1
Donnez Y : 3

Entrer Les cordonner du point Q :
Donnez le nom de point : X
Donnez X : 2.3
Donnez Y : -4.6

Entrer Le Rayon R : 5

La surface du triangle GEF est : S = 53.74
-----
Process exited after 25.27 seconds with return value 0
Press any key to continue . . .

```

### Trier un tableau de 100 points :

Pour faire ça on suit les étapes suivant :

- ✖ Saisie des points et stocker dans un tableau de structure spt.
- ✖ Puis prendre ce tableau et calcule la distance entre chaque point (trouve dans chaque case) et l'origine 0 du repéré. Puis stocker cette distance avec le nom du point dans un tableau de structure sdis.
- ✖ Puis donne ce dernier tableau de type sdis a une fonction de trie par sélection avec ordre croissant.

### Voici le code de chaque étape :

NB : le structure sdis n'es pas donné dans l'énoncer mais c'mieux d'utilisé pour faciliter les choses.



```

struct point {char Nom ; float x ; float y ;} ;
struct Distance {char Nom ; float dis ;} T1,tab;
typedef struct point spt ; //Déclaration de structure utilisée
typedef struct Distance sdis ;
spt O ={'O',0,0}; // Preinialisation de point O L'origine du repère

```

```

void tableau_dis(sdis T1[]){

```

```

    spt T[Max_Pts+1]; // Déclaration d'un tableau de 100 points
    int i=0,j;

```

```

    for(i=0;i<Max_Pts;i++){
        printf("Entrer les cordonner de point %d : \n",i+1);
        printf("Entrer Le nom : ");
        scanf("%s",&T[i].Nom);
        printf("Entrer X : ");
        scanf("%f",&T[i].x);
        printf("Entrer Y : ");
        scanf("%f",&T[i].y);
    }

```

```

    i=0;
    for(i=0;i<Max_Pts;i++){
        T1[i].dis = distance(T[i],O);
        T1[i].Nom = T[i].Nom;
    }

```

```

    i=0;
    for(i=0;i<Max_Pts;i++){
        T1[i].dis = distance(T[i],O);
        T1[i].Nom = T[i].Nom;
    }

```

```

    i=0;
    printf("L'ordre initiale du points : \n");
    printf("=====\\n");
    for(i=0;i<Max_Pts;i++){
        printf("%c\\t%.2f\\n",T1[i].Nom,T1[i].dis);
    }

```

```

//_____ fonction qui permet de trier ce tableau selon la distance des points à L'origine O _____

```

```

void tri_pts(sdis tab[]){

```

```

    int i, j, index;
    float tmp;
    char tmp1;

```

```

    for (i=0; i < (Max_Pts-1); i++){
        index = i;
        for (j=i + 1; j < Max_Pts; j++){
            if (tab[index].dis > tab[j].dis)
                index = j;
        }
    }

```

📄 Suite de code dans la page suivante !



```

        index = j;
    }

    if (index != i){
        tmp = tab[i].dis;
        tmp1 = tab[i].Nom;
        tab[i].dis = tab[index].dis;
        tab[i].Nom = tab[index].Nom;
        tab[index].dis = tmp;
        tab[index].Nom = tmp1;
    }

    printf("\n\n");
    printf("L'ordre du points par distance au l'origine 0 du repere :\n");
    for (i=0; i < Max_Pts; i++)
        printf("%c\t%.2f\n", tab[i].Nom, tab[i].dis);
}

```

🔗 L'exécution sera comme ça :

NB : j'ai utilisé juste 3 point pour l'exemple il reste le même pour 100 points et pour n'importe quel nombre de points.

```

Entrer les cordonner de point 1 :
Entrer Le nom : A
Entrer X : -5
Entrer Y : 3
Entrer les cordonner de point 2 :
Entrer Le nom : B
Entrer X : 0
Entrer Y : 1
Entrer les cordonner de point 3 :
Entrer Le nom : C
Entrer X : 0
Entrer Y : 0.2
L'ordre initiale du points :
=====
A      5.83
B      1.00
C      0.20

L'ordre du points par distance au l'origine 0 du repere :
C      0.20
B      1.00
A      5.83

-----
Process exited after 27.04 seconds with return value 0
Press any key to continue . . .

```

### Trace de schema :

Franchement je n'ai aucune idée comment peut tracer un schéma avec C.  
Si vous avez m'aider à comprendre comment faire ça, je vous serai très reconnaissant

Mon email est dans la premier page.

### ❖ Main code :

Comme j'ai dit dans le premier exercice, le main Code ou fonction main est une fonction permet de tester les fonctions précédent ainsi que Simplifier les choses et l'exécution de chaque fonction à part.

### Main code de ce programme :

```
44 // _____ main code _____
45
46 int main(){
47
48     spt ptr,ptr1,ptr2;
49     svect e1,e2;
50     int choix,freq;
51     float dxx,dyy;
52
53     menu2(); //Procédure pour afficher le menu (prédéfini dans "menu h"
54     scanf("%d",&choix);
55     system("cls");
56
57     switch(choix){
58         case 1: // case 1 pour saisie et afficher un point
59             printf("Saisir et affichage d'un point : \n");
60             ptr = saisie();
61             printf("\n");
62             afficher(ptr);
63             break;
64         case 2: // case 2 pour saisie et afficher un vecteur
65             printf("Saisie et affichage d'un vecteur : \n");
66             e1 = ss();
67             printf("\n");
68             aff(e1);
69             break;
70         case 3: // case 3 pour calcul la distance entre deux points
71             printf("Calcul des Distance entre deux points : \n");
72             ptr1 = saisie();
73             afficher(ptr1);
74             printf("\n\n");
75             ptr2 = saisie();
76             afficher(ptr2);
77             printf("\n\n");
78             printf("La distance Entre %c et %c est : %.2f",ptr1.Nom,ptr2.Nom,distance(ptr1,ptr2));
79             break;
```

La suite dans les pages suivantes !



```

80 case 4: // case 4 pour déplacer un point
81     printf("Déplacement des point : \n");
82     printf("Entrer Les cordonner de point : \n");
83     ptr = saisie();
84     printf("\n");
85     afficher(ptr);
86     printf("\n");
87     printf("\tentrer la valeur que vous allez deplacez Dans l'axe Ox : ");
88     scanf("%f",&dxx);
89     printf("\tentrer la valeur que vous allez deplacez Dans l'axe Oy : ");
90     scanf("%f",&dyy);
91     printf("\n\n\t");
92     Deplacer(&ptr,dxx,dyy);
93     break;
94 case 5: // case 5 pour calcul le produite scalaire de deux vecteurs
95     printf("Calcul de produit scalaire des vecteur : \n");
96     printf("Entrer Les cordonner du 1er Vecteur : \n");
97     e1 = ss();
98     printf("\n");
99     aff(e1);
100     printf("\nEntrer Les cordonner du 1eme Vecteur : \n");
101     e2 = ss();
102     printf("\n");
103     aff(e2);
104     printf("\n\t");
105     printf("Le produit scalair de deux vecteur est : %.2f",ProdScal(e1, e2));
106     break;
107 case 6: // case 6 pour vérifier si trois points est colénaire
108     printf("Entrer Les cordonner du 1er point : \n");
109     ptr = saisie();
110     printf("\n");
111     afficher(ptr);
112     printf("\n");
113     printf("Entrer Les cordonner du 2eme point : \n");
114     ptr1 = saisie();
115     printf("\n");
116     afficher(ptr1);
117     printf("\n");
118     printf("Entrer Les cordonner du 3eme point : \n");
119     ptr2 = saisie();
120     printf("\n");
121     afficher(ptr2);
122     printf("\n");
123     freq = Colineaires(ptr,ptr1,ptr2);
124     if(freq == 1){
125         printf("Les Trois points est colineaires!");
126     }else if(freq == 0){
127         printf("Les Trois points n'est pas colineaires!");
128     }
129     break;
130 case 7: // case 7 pour donne l'équation cartésienne d'un droit
131     printf("Entrer Les cordonner du 1er point : \n");
132     ptr1 = saisie();
133     printf("\n");
134     afficher(ptr1);
135     printf("\n");
136     printf("Entrer Les cordonner du 2eme point : \n");
137     ptr2 = saisie();
138     printf("\n");
139     afficher(ptr2);
140     printf("\n");
141     AffichEquatCartesienne (ptr1, ptr2);
142     break;

```



```

143 case 8: // case 8 pour vérifier si un triangle est Rectangle
144     printf("Entrer Les cordonner du 1er point : \n");
145     ptr = saisie();
146     printf("\n");
147     afficher(ptr);
148     printf("\n");
149     printf("Entrer Les cordonner du 2eme point : \n");
150     ptr1 = saisie();
151     printf("\n");
152     afficher(ptr1);
153     printf("\n");
154     printf("Entrer Les cordonner du 3eme point : \n");
155     ptr2 = saisie();
156     printf("\n");
157     afficher(ptr2);
158     printf("\n");
159     freq = triangleRectangle(ptr,ptr1,ptr2);
160     if(freq == 1){
161         printf("Le Triangle est Rectangle!");
162     }else if(freq == 0){
163         printf("Le Triangle n'est pas Rectangle!");
164     }
165     break;
166 case 9: // case 9 pour vérifier si un triangle est Isocèle
167     printf("Entrer Les cordonner du 1er point : \n");
168     ptr = saisie();
169     printf("\n");
170     afficher(ptr);
171     printf("\n");
172     printf("Entrer Les cordonner du 2eme point : \n");
173     ptr1 = saisie();
174     printf("\n");
175     afficher(ptr1);
176     printf("\n");
177     printf("Entrer Les cordonner du 3eme point : \n");
178     ptr2 = saisie();
179     printf("\n");
180     afficher(ptr2);
181     printf("\n");
182     freq = triangleIsocеле(ptr,ptr1,ptr2);
183     if(freq == 1){
184         printf("Le Triangle est Isocele!");
185     }else if(freq == 0){
186         printf("Le Triangle n'est pas isocele!");
187     }
188     break;
189 case 10: // case 10 pour accéder au sous-programme ( Q9 à Q12 )
190     printf("Votre demande sera transfere sur quel'que instant!\n\n");
191     delay(1000000000); // fonction pour faire Le retard avec La efface d'écran (prédéfini dans "menu.h")
192     system("cls");
193     printf("votre demande est maintenant disponible!\n\t\tApuyez any touch pour continue!");
194     getch(); // pour lire une clique au clavier
195     system("cls"); // pour vider l'écran
196     sous_prog();
197     break;
198 case 11: // case 11 pour Developpeur information
199     printf("\n\n\n\t\t\tHAMZA MESRAR");
200     break;
201 case 0: // case 0 pour sortir et fermer Le programme
202     printf("\n\n\t\t\tMESRAR HAMZA vous remercie de votre visite! _____\n\n");
203     getch();
204     break;
205 default : // Juste pour Incorrect choix
206     system("cls");
207     main(); // Exécuté La main à nouveau
208 }
209
210
211 return 0;
212 }
213

```

✦ Explication de ce Code :

Ce code consiste de quatre choses : déclaration de variable, fonction menu, écrire et lire à l'écran, et finalement une boucle **switch () case**.

### 💧 Déclaration des variables :

J'ai utilisé trois variables de type structure `spt` et deux de type `svect` pour stocker les informations successives des points et des vecteurs, et une variable `choix` pour la boucle `switch ()` `case`, est une variable `freq` pour stocker la `return` des fonctions qui `return 0` et `1` (booléenne) et deux variables `float` `dxx` et `dyy` pour la question de déplacement d'un point.

### 🔹 Fonction menu () :

 Code de fonction :

```
void menu2(void){  
    blue();  
    printf("\n\n\t\t");printf("*****\n");  
    printf("\t\t");printf("*\t\tMesure Section\t\t*\n");  
    printf("\t\t");printf("*****\n\n\n");  
    printf("\t\t\t");printf(" 1)- saisie et afficher un point.\n");  
    printf("\t\t\t");printf(" 2)- saisie et afficher un vecteur.\n");  
    printf("\t\t\t");printf(" 3)- calculer La distance entre deux point.\n");  
    printf("\t\t\t");printf(" 4)- deplacer un point.\n");  
    printf("\t\t\t");printf(" 5)- Calculer le produit scalaire de deux vecteur.\n");  
    printf("\t\t\t");printf(" 6)- verifie si trois points est colénair.\n");  
    printf("\t\t\t");printf(" 7)- L'Equation cartesienne d'une droit.\n");  
    printf("\t\t\t");printf(" 8)- verifie si un triangle est Rectangle.\n");  
    printf("\t\t\t");printf(" 9)- verifie si un triangle est Isocele.\n");  
    printf("\t\t\t");printf("10)- Accédez a un sous programme complementaire.\n");  
    printf("\t\t\t");printf("11)- Devloper Information.\n");  
    printf("\t\t\t");printf("00)- Exit.");  
    printf("\n\n\n\t\t");printf("____Entrer votre choix : ");  
    white();  
}
```

🖱 Le menu s'affiche comme ça :



```

*****
*                               *
*           Mesure Section      *
*                               *
*****

1)- saisie et afficher un point.
2)- saisie et afficher un vecteur.
3)- calculer La distance entre deux point.
4)- deplacer un point.
5)- Calculer le produit scalaire de deux vecteur.
6)- verifie si trois points est colÚnair.
7)- L'Equation cartesienne d'une droit.
8)- verifie si un triangle est Rectangle.
9)- verifie si un triangle est Isocele.
10)- Accedez a un sous programme complementaire.
11)- Devloper Information.
00)- Exit.

_____Entrer votre choix : _

```

#### ◆ Ecrire et lire le choix entrer :

Comme l'exercice précèdent.

```

Printf ("_____Entrer votre choix : ");
Scanf ("%d", &choix);

```

#### ◆ Une fonction sous prog :

Est une fonction pour accéder à un sous-programme qui trait les questions de 9 à 12.

#### 📄 Code de cette fonction :

NB : ce code est comme une fonction main de sous-programme les fonctions utilisé dans ce code sont déjà expliqué dans les parties précèdent.





```

446 // _____ fonction main de sous programme ( espace d'exécution des fonctions de sous programme ) _____
447
448 void sous_prog(void){
449
450     sdis M[Max_Pts+1];
451     spt Z1,Q1;
452     float rayon,surface;
453     int chx;
454
455     again :    // fixe un point pour Le revenir après
456     menu3();    //Procédure pour afficher Le menu (prédéfini dans "menu.h")
457     scanf("%d",&chx);
458     switch(chx){
459         case 1:    // case 1 Pour calculer la surface du triangle GEF dans Le schéma
460             system("cls"); // pour vider L'écran (fonction prédéfini dans <Windows.h>)
461             printf("Calculution Du surface de Triangle GEF : \n\t");
462             printf("Entrer Les cordonner du point Z : \n");
463             Z1 = saisie();
464             printf("\n");
465             printf("Entrer Les cordonner du point Q : \n");
466             Q1 = saisie();
467             printf("\n");
468             printf("Entrer Le Rayon R : ");
469             scanf("%f",&rayon);
470             printf("\n");
471             surface = SurfaceGEF (Z1, Q1, rayon);
472             printf("\n\t\tla surface du triangle GEF est : S = %.2f",surface);
473             break;
474         case 2:    // case 2 pour remplir est trie Le tableau de 100 points
475             system("cls");
476             tableau_dis(M);
477             tri_pts(M);
478             break;
479         case 3:    // case 3 pour tracer Le schéma
480             system("cls");
481             printf("\t\t-----\n\n");
482             break;
483         case 0:    // case 0 pour Developpeur information
484             system("cls");
485             printf("\n\n\t\t_____ MESRAR HAMZA vous remercie de votre visite! _____\n\n");
486             getch();
487             break;
488         default :    // Juste pour incorrect choix
489             printf("\n\t\t\terreur!");
490             system("cls");
491             goto again;    // pour routeur à un point déjà fixé
492     }
493
494 }
495

```

### ◆ Menu de ce sous-programme :

 Code :

```

void menu3(void){

    blue();
    printf("\n\n\t\t\t");printf("*****\n");
    printf("\t\t\t");printf("*\t\t\t sous programme\t\t\t*\n");
    printf("\t\t\t");printf("*****\n\n\n");
    printf("\t\t\t");printf(" 1)- Calculer La surface du triangle EGF.\n");
    printf("\t\t\t");printf(" 2)- Trier un tableau de 100 pts par distance a l'origine.\n");
    printf("\t\t\t");printf(" 3)- Tracer Le shema de figure.\n");
    printf("\t\t\t");printf(" 0)- exit.");
    printf("\n\n\n\t\t\t");printf("_____ Entrer votre choix : ");
    white();

}

```



## L'exécution :

```
*****
*                               *
*      sous programme          *
*                               *
*****

1)- Calculer La surface du triangle EGF.
2)- Trier un tableau de 100 pts par distance a l'origine.
3)- Tracer Le shema de figure.
0)- exit.

_____Entrer votre choix :
```

## V. Conclusion :

Cette expérience de travail sur ce projet fut très constructive et m'a permis de répondre aux questionnements que j'avais en ce qui concerne les structures, et comment va utiliser cette dernière dans les programmes.

Pour conclure, j'ai décidé de donner un bref aperçu de ce qui traite ce projet.

Premièrement le projet est divisé par deux exercices le premier et un programme qui traite les polynômes, et fait les calculs polynomiaux, et la deuxième qui traite les points et les vecteurs dans le plan. Et le but est comment utiliser les structures pour manipuler ces calculs.

Merci d'avoir lu tout ça, j'ai vraiment très reconnaissant 😊

