Contents

1

Langage C : une courte introduction

1.1 Introduction et bases

Nous allons discuter dans cette première partie les idées de base à savoir

- comment créer et écrire un fichier C
- comment le compiler
- comment l'exécuter

1.1.1 Comment créer et écrire un fichier C

Commençons par créer un dossier dans le Desktop de l'ordinateur. Pour cela ouvrons le Terminal. Puis tapons

```
cd Desktop
mkdir Atelier_CPI-1
```

Créons le fichier code0.c qui sera le fichier dans lequel nous écrirons le programme à proprement parlé. Dans le Terminal, on commence par aller dans le dossier Atelier_CPI-1 puis on crée le fichier

```
cd Atelier_CPI-1
gedit code0.c
```

A ce stade nous avons le fichier dans lequel on va maintenant écrire le programme C le plus simple possible (code0.c) :

```c
//code0.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main(){

/*printf est la fonction contenu dans la librairie stdio.h qui permet d'écrire à l'écran*/
printf("ceci est le code C le plus simple que l'on puisse écrire\n");

}
```

Remarques :

- Les commentaires : on peut écrire des choses qui seront ignorées par le compilateur et qui ne sont que des indications pour le lecteur ou l'auteur du programme. C'est ce que l'on appelle un commentaire. En C les commentaires s'écrivent des deux façons suivantes
  //ceci est un commentaire
  /*ceci est un commentaire*/

- Les librairies : les fichiers ".h" sont des librairies, c'est-à-dire, des codes C dans lesquels des fonctions sont préalablement définies et auxquelles on peut faire appelle en les incluants dans notre code grâce à la ligne #include

- stdlib.h : cette librairie contient les fonctions permetant de gérer les entrées/sorties d'un programme C (par exemple l'impression à l'écran avec "printf", l'écriture dans un fichier extérieur avec "fprintf" ou la lecture dans un fichier extérieur avec "fscanf")

- math.h : cette librairie contient les fonctions mathématiques comme exp, log, pow, cos, sin et autres

- stdlib.h : je ne commenterai pas ici de l'intérêt de cette librairie

- main : fonction principale. Toute instruction se trouvant dans le main sera exécutée après compilation lors de l'exécution

- Toute instruction en C se termine toujours par ";"

1.1.2 Comment compiler un code

L'étape de compilation est absolument capital. Il permet de traduire le code que nous avons écrit, dans une langue compréhensible par l'ordinateur. Cette traduction se fait par l'intermédiaire d'un "compilateur". Chaque langage a son propre compilateur et il existe en général plusieurs compilateurs. Sur LINUX, le compilateur C le plus communément utilisé est "gcc".

Pour compiler un code C, il suffit d'ouvrir un Terminal, de se déplacer avec le Terminal, en utilisant les commandes cd, jusqu'au dossier où se trouve le fichier C que l'on souhaite compiler. Puis la compilation se fait par l'intermédiaire de la ligne de commande suivante

```
gcc -o nom_de_executable nom_du_fichier.c -lm
```

exemple :
si mon programme s'appelle "code0.c" et que je souhaite que mon exécutable s'appelle prog_executable alors la ligne de commande est

```
gcc -o prog_executable code0.c -lm
```

cette ligne de commande transforme le code contenu dans le fichier code0.c en un fichier exécutable dont le nom sera prog_executable.

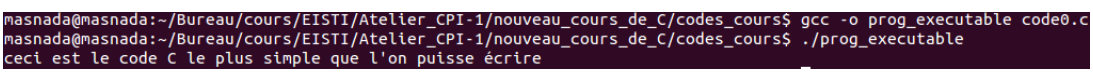
A ce stade, nous avons donc été capable de créer un code C et de le compiler. Nous devons maintenant l'exécuter.

1.1.3 Comment exécuter un programme C

A l'aide du Terminal, juste après avoir compiler le code C grâce à la ligne de commande précédente, nous pouvons maintenant exécuter le code précédent en écrivant

```
./prog_executable
```

Vous obtiendrez alors le résultat suivant dans votre Terminal



1.1.4 Conclusion

Maintenant que l'on est en mesure de créer un code C, que l'on est capable de le compiler et de l'exécuter, on va dans la suite pouvoir apprendre réellement à coder des choses un peu plus compliquées. C'est le but du reste de ce chapitre.

1.2 Déclaration de variables

1.2.1 Règles de base sur les déclarations

- toute variable utilisée doit être déclarée. En C, il est formellement interdit de déclarer implicitement. Donc à chaque fois que vous avez besoin d'une variable, il faut que cette variable soit préalablement déclarée.

- Quelque soit la nature de la variable déclarée, son nom peut contenir des chiffres et/ou des lettres ainsi que "_" mais aucun espace, "*", "-" ou autres symboles spéciaux. En C, on a par exemple le droit de nommer une variable kjhfkjt0564g_k3d

- dans une même fonction (on définira plus précisément ce que cela signifie) un nom de variable ne peut être utilisé qu'une fois. On ne peut pas déclarer deux fois le même nom de variable.

1.2.2 Déclaration d'un scalaire

1. int i; //déclaration d'un entier i sans lui attribuer une valeur
   int i,j; //déclaration de deux entiers i et j sans leur attribuer de valeur
   int i=2; //déclaration d'un entier i auquel on attribue la valeur 2

2. double a;//déclaration d'un réel a sans lui attribuer une valeur
   double a,b; //déclaration de deux réels a et b sans attribuer de valeur
   double a=2.4;//déclaration d'un réel a auquel on attribue la valeur 2.4

3. déclaration d'une variable en en-tête du code. On peut également définir une variable en en-tête de programme en écrivant en début de programme

   #define nom_de_la_variable valeur_de_la_variable

par exemple

#define aa 2.4

Signifie que lorsque l'on voudra utiliser la variable aa, on aura pas besoin de la définir et elle vaudra toujours 2.4. On ne peut donc pas changer la valeur d'une variable définie comme ceci. Cette déclaration est très souvent utilisée pour définir des constantes mathématiques ou physiques par exemple

#define pi 3.1415
#define G 6.6742e-11 // 6.6742×10^{-11}

Le code1.c donne des exemples de déclaration d'entier et d'opérations simples sur les entiers.

```c
//code1.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main(){

int j,k,l; //on définie les trois variables j, k et l qui sont des entiers.

j=3; //on attribue à j la valeur 3
k=4; //on attribue à k la valeur 4

l=j+k; //on attribue à l la somme des valeurs de j+k=7

printf("j=%i\tk=%i\tl=%i\n",j,k,l);//%i permet de dire à la fonction printf
                                  //qu'elle doit imprimer un entier.
                                  //le premier %i correspond à la première
                                  //variable après le " de droite.
                                  //le deuxième %i correspond à la deuxième
                                  //variable et ainsi de suite.

l=l+1; //on augmente l de 1 donc l=8.
printf("l est augmenté de 1\t %i\n",l);

//l=l+1 peut aussi s'écrire
l+=1; //donc ici l=9
printf("l est augmenté de 1\t %i\n",l);

//ou alors
l++; //donc ici l=10
printf("l est augmenté de 1\t %i\n",l);

l=j*k;//multiplication 4*3=12
printf("l=%i*%i=%i\n",j,k,l);

}
```

Le code2.c donne des exemples de déclaration de nombre à virgule et d'opérations. Il donne également des exemples d'écriture scientifique et non scientique de nombres à virgule

```c
//code2.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define pi 3.1415   //on définie une constante nommé pi qui est égale à 3.1415

main(){

double a,b,c; //on définie les trois variables a, b et c qui sont des nb à virgules.

a=3.5; //on attribue à j la valeur 3
b=4.5; //on attribue à k la valeur 4

c=a*a-b*b+a/b; //on attribue à c la valeur de l'opération suivante

printf("a=%le\tb=%le\tc=%le\n",a,b,c);//%le permet de dire à la fonction printf
                                      //qu'elle doit imprimer un nombre à virgule
                                      //avec la notation scientifique
                                      //le premier %le correspond à la première
                                      //variable après le " de droite.
                                      //le deuxième %le correspond à la deuxième
                                      //variable et ainsi de suite.
//écriture scientifique
printf("\nécriture scientifique des nombres à virgule\n");
a=1.34e3;//a=1.34*10^3
b=2.51e-2;//a=2.51*10^-2
printf("a=%le\tb=%le\tpi=%le\n",a,b,pi);

//écriture non scientifique
printf("\nécriture non scientifique\n");
printf("a=%f\tb=%f\tpi=%f\n",a,b,pi);  //%f est une écriture non scientifique d'un nb
                                       //à virgule

}
```

1.2.3 Déclaration d'un vecteur

Nous n'étudierons pas l'allocation dynamique de mémoire. Il s'agit d'un choix de ma part qui a pour objectif de ne pas trop complexifié le langage. Ainsi nous n'étudierons pas les pointeurs ou pointeurs de pointeurs.

On peut définir un vecteur de la façon suivante

- int a[5];
  signifie que a est un vecteur dont chaque composante est un entier. Ce vecteur contient cinq composantes :
  (a[0],a[1],a[2],a[3],a[4]). Pour affecter une valeur à la composante i de ce vecteur vous devez écrire
  a[i]=3; par exemple

- double a[5];
  signifie que a est un vecteur dont chaque composante est un "double".

Ce vecteur contient cinq composantes : (a[0],a[1],a[2],a[3],a[4]). Pour affecter une valeur à la composante i de ce vecteur vous devez écrire a[i]=21.08; par exemple

- segmentation fault : il s'agit du plus commun des bugs lorsque l'on exécute un code C qui n'a eu aucun problème lors de la compilation. Ce bug apparait notamment (mais pas seulement) lorsque l'on essaie d'affecter une valeur à une coordonnée qui n'appartient pas à la définition du vecteur. Par exemple si vous avez défini un vecteur avec dix composantes de type double (double a[10];) mais que vous attribuez ou utilisez la 15ième composante c'est-à-dire a[14].

```c
//code3.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define pi 3.1415   //on définie une constante nommé pi qui est égale à 3.1415

main(){

double vecteur1[3]; //vecteur contenant 3 composantes de type double.
                    //(vecteur1[0];vecteur1[1];vecteur1[2])
double vecteur2[3]; //vecteur contenant 3 composantes de type double.
                    //(vecteur2[0];vecteur2[1];vecteur2[2])



vecteur1[0]=1.0;vecteur1[1]=0.0;vecteur1[2]=0.0;//vecteur1=(1;0;0)
vecteur2[0]=0.0;vecteur2[1]=1.0;vecteur2[2]=0.0;//vecteur2=(0;1;0)

double produit_scalaire;

produit_scalaire=vecteur1[0]*vecteur2[0]+vecteur1[1]*vecteur2[1]+vecteur1[2]*vecteur2[2];

printf("u.v=%le\n",produit_scalaire);
printf("avec\n");
printf("u=(%le ; %le ; %le)\n",vecteur1[0],vecteur1[1],vecteur1[2]);
printf("v=(%le ; %le ; %le)\n",vecteur2[0],vecteur2[1],vecteur2[2]);

}
```

1.2.4 Déclaration d'un tableau (d'une matrice)

On peut définir une matrice de la façon suivante

- int a[2][3]; //signifie que a est une matrice de 2 lignes et 3 colonnes dont chaque élément est un entier. L'élément a[i][j] est l'élément correspondant à la ligne i et la colonne j. Pour affecter une valeur à

l'élément a[0][1] par exemple il suffit d'écrire a[0][1]=12;

$$\begin{pmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][1] & a[1][1] & a[1][2] \end{pmatrix} \tag{1.1}$$

- double a[2][3]; //signifie que a est une matrice de 2 lignes et 3 colonnes dont chaque élément est un "double". L'élément a[i][j] est l'élément correspondant à la ligne i et la colonne j. Pour affecter une valeur à l'élément a[1][2] par exemple il suffit d'écrire a[1][2]=1.232;

$$\begin{pmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][1] & a[1][1] & a[1][2] \end{pmatrix} \tag{1.2}$$

- segmentation fault : un dépassement du premier ou du deuxième indice par rapport à la déclaration de la variable entrainerai encore un bug lors de l'exécution du code signalé par "segmentation fault"

.

```c
//code4.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define pi 3.1415   //on définie une constante nommé pi qui est égale à 3.1415

main(){

double matrice1[2][2]; //matrice de dimension 2*2
                       //     [0][0]    [0][1]
                       //     [1][0]    [1][1]
double matrice2[3][3]; //matrice de dimension 2*2
                       //     [0][0]    [0][1]
                       //     [1][0]    [1][1]


matrice1[0][0]=1.0;matrice1[0][1]=0.0;matrice1[1][0]=0.0;matrice1[1][1]=1.0;
matrice2[0][0]=2.0;matrice2[0][1]=3.0;matrice2[1][0]=4.0;matrice2[1][1]=5.0;

double matrice3[3][3];
//calcul du produit matriciel
matrice3[0][0]=matrice1[0][0]*matrice2[0][0]+matrice1[0][1]*matrice2[1][0];
matrice3[0][1]=matrice1[0][0]*matrice2[0][1]+matrice1[0][1]*matrice2[1][1];
matrice3[1][0]=matrice1[1][0]*matrice2[0][0]+matrice1[1][1]*matrice2[1][0];
matrice3[1][1]=matrice1[1][0]*matrice2[0][1]+matrice1[1][1]*matrice2[1][1];

//on imprime le résultat sous une forme matricielle
printf("%le\t%le\n",matrice3[0][0],matrice3[0][1]);
printf("%le\t%le\n",matrice3[1][0],matrice3[1][1]);

}
```

1.3 Les instructions for, while, if

1.3.1 Instruction for

L'objectif de l'instruction for est de créer une boucle. La syntaxe de la boucle for est

$for(i = i_0; i < i_1; i++)\{$
liste d'instructions dépendant de i
$\}$

ce qui signifie que pour $i = i_0$, on effectue l'ensemble des instructions dans les accolades. Une fois que toutes les opérations sont effectuées i est incrémenté (augmenté) de 1 (c'est le sens de $i++$) puis on effectue à nouveau l'ensemble de ces opérations et ainsi de suite jusqu'à ce que i devienne plus grand que i_1 et alors on ne peut plus rentrer dans la boucle for (fin de la boucle for). Notons qu'il n'y a aucune obligation de faire varier i de 1 en 1. On peut par exemple le faire varier de deux en deux en remplaçant $i++$ par $i = i + 2$. Dans tous les cas i doit être un entier. Une boucle "for" peut être stoppé grâce à la commande

break;

Dans le code5.c on donne deux exemples de l'utilisation des boucles "for". La première consiste à faire le produit scalaire de deux vecteurs. Soient $\vec{u} = (u_0, u_1, ..., u_n)$ et $\vec{v} = (v_0, v_1, ..., v_n)$. Alors le produit scalaire est donné par

$$\vec{u} \cdot \vec{v} = \sum_{i=0}^{n} u_i v_i \tag{1.3}$$

La seconde à faire le calcul des entiers de 0 à N

$$\sum_{i=0}^{N} i = \frac{N(N+1)}{2} \tag{1.4}$$

```c
//code5.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define pi 3.1415   //on définie une constante nommé pi qui est égale à 3.1415

main(){
//calcul du produit scalaire
int N=10;
double vecteur1[N];
double vecteur2[N];
int i;
for(i=0;i<N;i++){
vecteur1[i]=1.0*(i+1.0);
vecteur2[i]=-1.0*(i+1.0);
}

double produit_scalaire=0.0;

for(i=0;i<N;i++){
produit_scalaire=produit_scalaire+vecteur1[i]*vecteur2[i];
}
printf("produit scalaire=%le\n",produit_scalaire);

//sommation de Gauss
int NN=100;
int somme=0;

for(i=0;i<=NN;i++){
somme=somme+i;
}
printf("\nla somme de 1 à %i est\n",NN);
printf("valeur numérique=%i\tvaleur formule de Gauss=%i\n\n",somme,NN*(NN+1)/2);


}
```

1.3.2 Instruction while

La syntaxe de cette instruction est

while(condition){
liste d'instructions
}

1.3.3 Instruction if

L'instruction if est de la forme

if(conditions){
liste d'instructions
}elseinstructions

Les conditions peuvent être

- element1>element2
- element1<element2
- element1==element2 : signifie que l'élément 1 est égal à l'élément 2
- element1!=element2 : signifie que l'élément 1 est différent de element2

On peut également combiner plusieurs conditions grâce aux opérateurs || (ou), && (et). Le "else" n'est pas obligatoire et peut être complétement supprimer.

```
//code6.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main(){
//calcul du produit scalaire
int N=3; int i;
double vecteur1[N]; double vecteur2[N];
vecteur1[0]=1.0;vecteur1[1]=0.0;vecteur1[2]=0.0;
vecteur2[0]=0.0;vecteur2[1]=2.0;vecteur2[2]=0.0;
double produit_scalaire=0.0;
for(i=0;i<N;i++){
produit_scalaire=produit_scalaire+vecteur1[i]*vecteur2[i];
}

//calcul de la norme des deux vecteurs
double norme_vecteur1=0.0; double norme_vecteur2=0.0;
for(i=0;i<N;i++){
norme_vecteur1=norme_vecteur1+vecteur1[i]*vecteur1[i];
norme_vecteur2=norme_vecteur2+vecteur2[i]*vecteur2[i];
}
norme_vecteur1=sqrt(norme_vecteur1); norme_vecteur2=sqrt(norme_vecteur2);

//Si le produit scalaire=0 et que la norme du vecteur1=1 et que la norme du vecteur2=1
if(produit_scalaire==0 && norme_vecteur1==1 && norme_vecteur2==1){
printf("les deux vecteurs sont orthonormés\n");
}

//Si le produit scalaire n'est pas égal à 0 ou que la norme
//du vecteur1 n'est pas égal à 1 ou que la norme du vecteur2 n'est pas 1
if(produit_scalaire!=0 || norme_vecteur1!=1 || norme_vecteur2!=1){
printf("les deux vecteurs ne sont pas orthonormé\n");
}

}
```

1.4 Fonctions

On va maintenant apprendre à créer des fonctions, c'est-à-dire un ensemble d'instructions défini hors du "main". L'exécution d'une fonction se fait

soit par appelle de la fonction dans le main() soit par appelle dans d'autres fonctions qui seront, elles, appelées dans le main().

Créons par exemple une fonction permettant de calculer le produit scalaire de deux vecteurs quelconques. Pour faire le calcul du produit scalaire entre deux vecteurs, on a besoin

- de la dimension des deux vecteurs: N
- du vecteur 1
- du vecteur 2
- d'une méthode pour communiquer le résultat du calcul de la fonction

Illustons la structure d'un tel programme pour comprendre exactement la raison du dernier point. Pour cela, imaginons que l'on écrive la fonction sous la forme suivante.

```
fonction_calcul_produit_scalaire(int N,double vec1[],double vec2[]){
/*on a déclaré par l'intermédiaire des arguments de la fonction, l'entier N
qui représente la dimension des vecteurs et les deux vecteurs vec1 et vec2.
On peut donc maintenant faire le calcul du produit scalaire*/
        int i;
        double produit_scalaire;
            for(i=0;i<N;i++){
                produit_scalaire=produit_scalaire+vec1[i]*vec2[i];
            }
}
```

Pour exécuter cette fonction, il faudrait alors y faire appelle dans le main en écrivant

```
fonction_calcul_produit_scalaire(N,vec1,vec2);
```

où N, vec1 et vec2 auront été défini préalablement dans le main.

Mais comment récuperer la valeur de produit_scalaire calculée dans la fonction ?
Il existe plusieurs façons de faire :

1. utiliser un return (code7.c) :

Il faut alors dire que la fonction fonction_calcul_produit_scalaire renvoie un scalaire de type double. Pour cela, on écrit

```
double fonction_calcul_produit_scalaire(int N,double vec1[],double vec2[]){
/*on a déclaré par l'intermédiaire des arguments de la fonction, l'entier
N qui représente la dimension des vecteurs et les deux vecteurs vec1
et vec2. On peut donc maintenant faire le calcul du produit scalaire*/
        int i;
        double produit_scalaire;
            for(i=0;i<N;i++){
                produit_scalaire=produit_scalaire+vec1[i]*vec2[i];
            }
        return(produit_scalaire);
}
```

Pour exécuter cette fonction et récuperer la valeur, il faudrait alors y faire appelle dans le main en écrivant

```
double produit;
produit=fonction_calcul_produit_scalaire(N,vec1,vec2);
```

Tous les éléments changés ou ajoutés ont été soulignés. En résultat, la variable "produit_scalaire" sera écrite dans "produit".

2. utiliser un vecteur (code7bis.c):

```
void fonction_calcul_produit_scalaire(int N,double vec1[],double vec2[],double
produit_scalaire[]){
/*on a déclaré par l'intermédiaire des arguments de la fonction, l'entier
N qui représente la dimension des vecteurs et les deux vecteurs vec1
et vec2. On peut donc maintenant faire le calcul du produit scalaire*/
        int i;
        produit_scalaire[0]=0.0;
            for(i=0;i<N;i++){
                produit_scalaire[0]=produit_scalaire[0]+vec1[i]*vec2[i];
            }
}
```

Pour exécuter cette fonction et récuperer la valeur, il faudrait alors y faire appelle dans le main en écrivant

double produit[1];
fonction_calcul_produit_scalaire(N,vec1,vec2,produit);

alors le résultat du produit scalaire est donné dans produit[0].

3. utiliser une structure : cette méthode un peu plus complexe mais très efficace est pour l'instant laissée de côté. J'espère avoir le temps de vous en parler d'ici la fin du semestre.

On donne dans les deux figures suivantes les codes associés aux deux méthodes discutées précédemment. Le premier code (code7.c) correspond à la methode utilisant return, le deuxième code (code8.c) correspond à la méthode du vecteur.

```c
//code7.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double calcul_produit_scalaire(int N,double vec1[],double vec2[]){
//N est donc la taille des vecteurs. vecteur1 et vecteur2 sont les
//deux vecteurs dont on veut calculer les produits scalaires
//le resultat sera écrit dans produit_scalaire. Ces variables étant déclarées dans les arguments
//de la fonction, elles n'ont pas à être déclarer dans la fonction
   int i;double produit_scalaire;
   produit_scalaire=0.0;
        for(i=0;i<N;i++){
          produit_scalaire=produit_scalaire+vec1[i]*vec2[i];
        }
   return(produit_scalaire);
}

main(){
int N=3;double produit_scalaire1;double produit_scalaire2;double produit_scalaire3;
double vecteur1[N]; double vecteur2[N]; double vecteur3[N];
vecteur1[0]=1.0;vecteur1[1]=0.0;vecteur1[2]=0.0;
vecteur2[0]=1.0;vecteur2[1]=1.0;vecteur2[2]=0.0;
vecteur3[0]=0.0;vecteur3[1]=3.0;vecteur3[2]=1.0;

    produit_scalaire1=calcul_produit_scalaire(N,vecteur1,vecteur2);
    produit_scalaire2=calcul_produit_scalaire(N,vecteur1,vecteur3);
    produit_scalaire3=calcul_produit_scalaire(N,vecteur3,vecteur2);

printf("\n%le\t%le\t%le\n",produit_scalaire1,produit_scalaire2,produit_scalaire3);
if(fabs(produit_scalaire1)==0.0 && fabs(produit_scalaire2)<1.e-10 && fabs(produit_scalaire3)<1.e-10){
printf("\nla base est orthogonale puisque les produits scalaires sont deux à deux nuls\n\n");
}else{printf("\nla base n'est pas orthonormale puisque les vecteurs ne sont pas orthogonaux\n\n");}

}
```

```c
//code7bis.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double calcul_produit_scalaire(int N,double vec1[],double vec2[],double produit_scalaire[]){
//N est donc la taille des vecteurs. vecteur1 et vecteur2 sont les
//deux vecteurs dont on veut calculer les produits scalaires
//le resultat sera écrit dans produit_scalaire. Ces variables étant déclarées dans les arguments
//de la fonction, elles n'ont pas à être déclarer dans la fonction
   int i;
   produit_scalaire[0]=0.0;
        for(i=0;i<N;i++){
          produit_scalaire[0]=produit_scalaire[0]+vec1[i]*vec2[i];
        }
}

main(){
int N=3;double produit_scalaire1[1];double produit_scalaire2[1];double produit_scalaire3[1];
double vecteur1[N]; double vecteur2[N]; double vecteur3[N];
vecteur1[0]=1.0;vecteur1[1]=1.0;vecteur1[2]=0.1;
vecteur2[0]=1.0;vecteur2[1]=1.0;vecteur2[2]=10.0;
vecteur3[0]=0.0;vecteur3[1]=3.0;vecteur3[2]=1.0;

    calcul_produit_scalaire(N,vecteur1,vecteur2,produit_scalaire1);
    calcul_produit_scalaire(N,vecteur1,vecteur3,produit_scalaire2);
    calcul_produit_scalaire(N,vecteur3,vecteur2,produit_scalaire3);

printf("\n%le\t%le\t%le\n",produit_scalaire1[0],produit_scalaire2[0],produit_scalaire3[0]);
if(fabs(produit_scalaire1[0])==0.0 && fabs(produit_scalaire2[0])<1.e-10 && fabs(produit_scalaire3[0])<1.e-10){
printf("\nla base est orthogonale puisque les produits scalaires sont deux à deux nuls\n\n");
}else{printf("\nla base n'est pas orthonormale puisque les vecteurs ne sont pas orthogonaux\n\n");}

}
```

On va maintenant créer un programme qui permet de calculer la fonction $f(x) = x^2$ sur $I = [-1; 1]$ à l'aide d'une fonction extérieure au "main". Le résultat de la fonction sera donné dans deux vecteurs déclarés dans les arguments : un vecteur contenant les x et un vecteur contenant les $f(x)$

```
//code8.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void calcul_de_fonction(int N,double delta_x,double xmin,double x[],double fonction[])
{     int i;
          for(i=0;i<N;i++){
             x[i]=xmin+i*delta_x;
             fonction[i]=x[i]*x[i];
          }
}

main(){
//définition de l'intervalle I=[xmin;xmax]
double xmin=-1.0;
double xmax=1.0;

int N=10000;//nombre de points de discrétisation sur le segment I

double delta_x;//intervalle entre deux points de discrétisation
delta_x=(xmax-xmin)/(1.0*N);

double x[N];double f[N];

                calcul_de_fonction(N,delta_x,xmin,x,f);

//création du fichier ''courbe''
FILE *file_out1;file_out1=fopen("courbe","w");
int i;
for(i=0;i<N;i++){
        fprintf(file_out1,"%le\t%le\n",x[i],f[i]);
}

}
```

Ce code crée, grâce à "fprintf", un fichier extérieur nommé "courbe" et placé dans le même dossier. Ce fichier contient deux colonnes séparés par une tabulation. La première colonne est la valeur de x, la deuxième colonne la valeur de $f(x)$.

On peut tracer la courbe ainsi obtenu en utilisant gnuplot. Dans le Terminal

gnuplot
plot "courbe" using 1:2 with line lw 2 lt 1

ce qui signifie que gnuplot doit tracer à l'aide du fichier "courbe" en utilisant

les colonnes 1 et 2 (using 1:2). Cette ligne de commande trace la colonne 2 en fonction de la colonne 1.

1.5 Conclusion

L'introduction donnée ici de la programmation C est évidemment succinte nous n'avons ainsi pas parlé

- des structures
- des pointeurs
- des pointeurs de pointeurs
- de l'allocation dynamique de mémoire
- de la lecture de fichiers extérieurs
- des modulos
- etc...

Toutefois, cette introduction en langage C est suffisante pour pouvoir développer les premières méthodes numériques afin de résoudre de manière simple certains problèmes mathématiques.

Pour les élèves intéressés par un cours plus complet vous pouvez consulter le cours de Bernard Cassagne disponible en ligne

ftp://ftp-developpez.com/c/cours/bernard-cassagne//Introduction_ANSI_C.pdf