Les fonctions en C

- Dès qu'un groupe de lignes revient plusieurs fois on les regroupe dans une fonction
- Une fonction se reconnaît à ses ()
- Une fonction en C est assez proche de la notion mathématique de fonction:

  Exemples :

  y = sqrt(x) ;

  Z = pgcd(A,B) ;

Intérêt des fonctions

- Lisibilité du code
- Réutilisation de la fonction
- Tests facilités
- Évolutivité du code

- *Plus tard* : les fonctions dans des fichiers séparés du main.c

- *Nb : une fonction peut faire appel à d'autres fonctions*
  - *dans son code*
  - *dans ses arguments*

Bibliothèques de fonctions

- Il existe des bibliothèques de fonctions déjà programmées.

  Exemples :
  - math.h : *fonctions math.*
  - stdio.h: *standard input-output*
  - stdlib.h : *bibli. standard*
  - time.h : *fonctions temporelles*

Nous pouvons aussi créer les notre

2 types de fonctions

- Des fonctions qui s'exécutent sans retourner de valeurs

  → *nommées procédures dans certains langages*

  → Seront typées **void**

  Ex : une fonction qui affiche « bonjour »

```
void affiche_bonjour()
{printf(« bonjour »);}
```

- Des fonctions qui s'exécutent et retournent une valeur

  Exemples : `sin(x) ; z = sqrt(x) ;`

  → Auront le type de la valeur à retourner

Paramètres réels – paramètres formels

- Un paramètre ou argument **réel**, est une valeur ou une variable qui est mis entre parenthèses lors de l'appel de la fonction.

  - Il existe vraiment en mémoire.

- Un paramètre ou argument **formel** est un nom de variable utilisé lors de la déclaration de la fonction.

  - Le nom peut être omis (pas conseillé)
  - Ne correspond pas à un emplacement mémoire

Déclaration d'une fonction

- Permet au compilateur de vérifier l'adéquation des types et de réserver l'espace mémoire pour la valeur de retour
- A l'aide d'un **prototype** de fonction utilisant des *paramètres formels* typés de la forme :

Type-retourné **NOM-FONCTION (type1 paramètre1, type2 paramètre2, …) ;**

```
double CalculePrixNet(double prix, double tauxTVA) ;
```

NB : on peut définir une fonction avec autant de paramètres formels qu'on veut. Dans l'exemple, il y a deux paramètres formels.

Définition d'une fonction

- C'est le **code** de la fonction, de la forme :

Type-retourné NOM-FONCTION (type1 paramètre1, type2 paramètre2, …)

{

Déclaration des autres variables de la fonction;

Code de la fonction;

return (valeur-de-la-fonction) ;

}

- En C, une fonction ne peut retourner qu'une valeur (au plus) grâce à la commande *return*
- Le type de la fonction doit être le même que celui de la valeur retournée
- Le programme appelant doit stocker ce résultat dans une variable de même type (ou bien ne rien stocker)
- Quand une fonction ne retourne pas de valeur elle est typée *void*
  - *Exemples* : void main() ; void AfficheBonjour();

Le return

- Retourne la valeur au programme appelant
- Et interrompt immédiatement l'exécution de la fonction
  - On peut avoir plusieurs *return*
  - Mais un seul *return* pris en compte à chaque exécution

Variables locales – variables globales

- Définition : Un bloc est la partie de code compris entre {}
- Une variable créee dans un bloc n'existe que dans ce bloc
  - C'est une **variable locale** au bloc
  - Elle ne sera pas connue en dehors
  - Sa valeur est perdue à la sortie du bloc
  - *« Sa durée de vie est celle du bloc »*

Variables locales – variables globales

- Une variable **globale** existe en dehors de tout bloc
  - Elle a sa mémoire réservée pour toute l'exécution du programme
  - « Sa durée de vie est celle du programme »
  - *Exemple :*

```
int i ;
main()
{ i=2;
printf(''%d'',i);
}
```

- ***Conseil : Soyez le plus local possible***

Déclaration de variables dans les fonctions

- De 2 manières :

```
int triple (int x)
{
int y;
y = 3 * x ;
return (y) ;
}
```

  - y est **locale** à la fonction
  - Sa valeur sera perdue à la sortie de la fonction

  - x est **locale** à la fonction
  - Elle est initialisée lors de l'appel à la valeur fournie par le programme appelant
  - Sa valeur sera perdue à la sortie de la fonction
  - *Nb : on parle de passage par valeur des arguments :* ***leurs valeurs sont recopiées dans des variables locales à la fonction***

Appel d'une fonction

Appel de la fonction à partir d'une autre fonction (par ex le main)

Création des variables locales et copie des valeurs des arguments de l'appel dans les variables locales de la fonction

Exécution de la fonction

La fonction retourne une valeur. Fin de la fonction.

Les variables locales à la fonction cessent d'exister

La fonction appelante peut récupérer la valeur de retour

z=produit(a,b)

int produit(x,y)

z prend le produit de a et b

```c
#include<stdio.h>

int produit(int X,int Y);

int main()

{
int a,b,Z;

printf("Calcul du produit de deux entiers:\n");
printf("merci de donner deux entiers écris sous la forme xx,yy \n");
scanf("%d,%d",&a,&b);
Z=produit(a,b);
printf("le produit de %d et %d est %d \n",a,b,Z );
return 0;
}

int produit(int X,int Y)
{
int k;

k=X*Y;
return k;

}
```

Codage d'une fonction : exemple

```
#include <stdio.h>

int triple(int x) ; //prototype

int main()
{
int a=2 ;
int b ;
triple(2) ; //appels
triple(a) ;
b = triple(a) ;
a = triple(a) ;
return 0;
}

int triple(int x) //définition
{
return (3*x) ;
}
```

Les bonnes pratiques de programmation

- Une fonction ne fait en général qu'une chose
- Le nom de la fonction décrit cette chose ;
- Prendre le temps de bien choisir les fonctions, leur nom, leurs paramètres

  → Bien choisir un nom explicite … et l'utiliser par copier-coller avec son jeu de paramètres

- Une fonction reçoit un nombre limité de paramètre (2-3 dans la plupart des cas) ;
- Une fonction ne compte pas trop de lignes
- Tester chaque fonction avant de passer à l'écriture de la suivante

Les erreurs courantes avec les fonctions

- Une fonction est déclarée mais non définie
- Une fonction est appelée et n'existe pas
- Le type de la fonction ne correspond pas au type de la valeur retournée
- La valeur retournée n'est pas stockée dans une variable du bon type
- Entre la déclaration, la définition et l'appel, le nombre de paramètres n'est pas le même
- Au moins un paramètre n'a pas le bon type
- Ne confondez pas *valeur retournée* par la fonction (qui peut être stockée dans une variable en mémoire) et *affichage à l'écran d'un résultat* (qui n'est pas automatiquement stocké en mémoire)

Conseils

- ✓ Si vous utilisez beaucoup de fonctions,tenez leur liste à jour (Tableur, texte, …)

- ✓ Lorsque vous écrivez une fonction : testez-la et assurez-vous de son bon fonctionnement avant de passer à l'écriture de la suivante !!

- ✓ Ce qu'on ne doit **jamais** faire : écrire toutes les fonctions et tester ensuite tout d'un bloc.

- ✓ Evitez les printf dans une fonction qui n'est pas dédiée à l'affichage. Vous pouvez utiliser des affichages avec printf dans vos fonctions pour les débugger, mais retirez-les dès que la fonction marche correctement.