

Langage C

Fonctions



`#include<stdio.h>`

Programmation modulaire

- Il y a deux types de modules en C : la fonction et le fichier source.
- Le module de base s'appelle une fonction.
- Un programme en C exécute uniquement ce qu'il y a dans la fonction principale `main()` .
- La simple lecture du `main()` doit permettre de comprendre ce que fait le programme.
- La fonction `main()` doit être réduite au maximum et ne contenir dans l'idéal que des appels à des fonctions qui sont écrites ailleurs.

Intérêts des fonctions :
réutilisabilité, lisibilité, évolutivité

Bibliothèques de fonctions

- Il existe des bibliothèques de fonctions déjà programmées.

Exemples :

- *math.h : fonctions mathématiques (sin, sqrt ...)*
- *stdio.h : fonctions d'entrées-sorties (printf, scanf ...)*
- *stdlib.h : allocation mémoire, tirages aléatoires, conversions*
- *time.h : fonctions temporelles (clock, time ...)*

- Vous créez vos propres bibliothèques de fonctions.

La fonction main

```
#include <stdio.h>  
#include <stdlib.h>
```

} directives au préprocesseur

```
int main()  
{  
    printf("Hello world !\n");  
    return 0;  
}
```

} instructions } fonction

Deux sortes de fonctions

- Des fonctions qui s'exécutent et **qui ne renvoient rien** :

Appelées procédures dans d'autres langages

Ce sont en général des fonctions d'affichage

Elles sont typées **void**.

exemple :

```
void affiche_bonjour()  
{ printf("bonjour !"); }
```

Deux sortes de fonctions

- Des fonctions qui s'exécutent et **qui ne renvoient rien** :

Appelées procédures dans d'autres langages

Ce sont en général des fonctions d'affichage

Elles sont typées **void**.

exemple :

```
void affiche_bonjour()  
{ printf("bonjour !"); }
```

- Des fonctions qui s'exécutent et **qui renvoient une valeur** :
elles ont le type de la valeur retournée.

exemple :

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

*La fonction triple retourne une
valeur qui vaut 3 fois la valeur
de l'argument x*

Construction d'une fonction

- Sur papier, on réfléchit à ce que doit faire la fonction :

Une fonction = UNE action ou UN calcul

On choisit un nom explicite. On détermine le ou les paramètres nécessaires à la réalisation et le type retourné.

- On code ensuite et pour l'instant de la façon suivante :

```
#include <stdio.h>
#include <stdlib.h>
```

2. la **déclaration** de la fonction **au-dessus** du main

```
int triple(int x);
```

```
int main()
{
```

```
    int x;
    x=triple(5);
    printf("%d",triple(10));
    return 0;
```

```
}
```

3. un ou des **appels** à la fonction **dans** le main

1. la **définition** de la fonction **sous** le main

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

```
}
```

De façon plus formelle ...

```
#include <stdio.h>
#include <stdlib.h>
```

permet au compilateur de vérifier l'adéquation des types. C'est le minimum nécessaire pour compiler.

2. la **déclaration** de la fonction **au-dessus** du main

```
type_retourné nom_fonction(type1 param1, type2 param2 ...);
```

3. un ou des **appels** à la fonction **dans** le main

```
int main()
{
    déclaration de variables ;
    nom_fonction(x,y, ...);
    return 0;
}
```

Les types n'apparaissent plus

param1, param2 ... sont aussi des variables **locales** à la fonction

1. la **définition** de la fonction **sous** le main

return :

- 1) retourne la valeur de la fonction et stoppe l'exécution de la fonction.
- 2) une valeur au plus peut être retournée.

```
type_retourné nom_fonction(type1 param1, type2 param2 ...)
{
    déclaration des autres variables de la fonction (dites LOCALES);
    code de la fonction;
    return (valeur) ou rien si type void ;
}
```

Fonctions et tableaux

Un tableau peut être un paramètre d'une fonction, mais PAS un élément retourné.

Pour donner un caractère générique à la fonction, la dimension du tableau est systématiquement un paramètre de la fonction.

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 5

void init_tab(int tab[], int dim, int val);

int main()
{
    int tab1[DIM];
    init_tab(tab1, DIM, 10);
    return 0;
}

void init_tab(int tab[], int dim, int val)
{
    int i ;
    for (i=0; i<dim; i++) tab[i]=val;
}
```

En-tête imposé et obligatoire pour toutes les fonctions

A écrire pour chaque fonction
et à placer au-dessus de la
déclaration de la fonction.

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 5
void init_tab(int tab[], int dim, int val);

/* init_tab : fonction initialisant un tableau de
dim entiers à une valeur val.
ENTREES :
tab : tableau d'entiers
dim : dimension du tableau
val : valeur d'initialisation
SORTIES :
tab : tableau initialisé
Auteur/date/version
*/

int main()
{...
    return 0;
}

void init_tab(int tab[], int dim, int val)
{
int i ;
for (i=0;i<dim;i++) tab[i]=val;
}
```

Quelques remarques importantes

- Lorsque vous écrivez une fonction : **testez-la** et assurez-vous de son bon fonctionnement **avant** de passer à l'écriture de la suivante !!
- Ce que l'on ne doit **jamais** faire : écrire plusieurs 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 pour debugger vos fonctions mais retirez-les dès que la fonction marche correctement.
- 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)