



# Programmation Structurée 2

---

Pr. EL OUKKAL Sanae

# Contenu du Cours: Programmation Structurée

- Fonctions
- Structure
- Les Notions des Pointeurs (& et \*)
- Les Listes Chaînées
- Les Fichiers
- ...



---

# RAPPEL DES NOTIONS ACQUISES EN PROGRAMMATION STRUCTURÉE 1:

# Les Composants élémentaires du C

## Identificateur

- Les identificateurs,
  - Le rôle d'un identificateur est de donner un nom à une entité du programme. Plus précisément, un identificateur peut désigner
- Ils peuvent être:
  - un nom de variable ou de fonction,
  - un type défini par typedef, struct, union ou enum,
  - une étiquette.

# Les Composants élémentaires du C

## Mots-Clés

- Les Mots-Clés:

- Un certain nombre de mots, appelés mots-clefs, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. L'ANSI C compte 32 mots clefs:

**int   break   case   char   const   continue   default   do**  
**if   float   for   while   else   switch   struct   Void**  
**type   return   long   double**

- Ces types peuvent être classifiés sous :

- Spécificateur de Stockage,
- Spécificateur de Type,
- Instruction de Contrôle.



# Les Composants élémentaires du C

---

- Les identificateurs,
- les mots-clefs,
- les constantes,
- les chaînes de caractères,
- les opérateurs,
- les signes de ponctuation.



# Structure d'un Programme C

---

- Plusieurs règles à suivre pour écrire un programme C:
  - Expression?
  - Les Instructions? Une Ligne? Bloc d'Instructions?
  - Variables Utilisées?
  - Programme Principal? Son Rôle?



# Types Prédéfinis

---

- Le C est un langage typé.
  - Toute variable, constante ou fonction doit avoir un type précis.
- Le type d'un objet définit la façon dont il est représenté en mémoire.
- On trouve:
  - Char,int, float,double, long..



# Déclaration des Variables et Constantes

- Variable?

**Type** nom\_variable;

**Type** nom\_variable = valeur;

- Constante?

**Const** nom\_constant = valeur;

**#define** nom\_constant valeur

# Opérateurs

- Opérateurs:
  - Affectation;
  - Arithmétiques;
  - Relationnels;
  - Logiques Booléens;
  - Affectation Composée,
  - Incrémentation et Décrémentation
  - ....

# Instructions

## Alternatives/Itératives

---

- Instructions Alternatives?

If (condition1) instruction1;

Switch;(multi-choix)

- Instructions Itératives?

while;

do while;

for.

# Fonctions

## Entrée/ Sortie

- Écriture:
  - Printf,
  - Contient du texte à afficher,
  - Et aussi des fois les spécifications du format d'impression,
  - Format d'impression: %d, %f, %c, %s.
  
- Lecture:
  - Scanf,
  - Contient la spécification et l'adresse de la variable,



---

# EXERCICES PARTIE 1



---

# TABLEAUX

# Tableau en C

- À partir des types prédéfinis, on peut créer de nouveaux types appelés types composés, représentant un ensemble de données organisées.
- On le déclare suivant la syntaxe:  
`Type nom_tableau [nbr-elts];`

# Tableau en C

- Le tableau correspond à un pointeur vers le premier élément du tableau.
- L'écriture de `tab1 = tab2` est incorrecte.
  - Il faut la faire cellule par cellule.
- On peut aussi lors de la déclaration d'un tableau, l'initialiser.



# Tableau en C:

## Exemple 1

Qu'affiche ce petit programme?

```
#define N 4  
  
int tab[N] = { 1, 2, 3, 4};  
  
main(){  
    int i;  
    for (i = 0; i < N; i++)  
        printf("tab[%d] = %d\n",i,tab[i]);  
}
```

# Tableau en C:

## Exemple 2

Qu'affiche ce petit programme?

```
#define N 8
char tab[N] = "exemple";
main(){
    for (i = 0; i < N; i++)
        printf("tab[%d] = %c\n",tab[i]);
}
```

# Tableau en C: Exemple 2

```
8  #define N 8
9  char tab[N] = "exemple";
10 main()
11 {
12  int i;
13  for (i = 0; i < N; i++)
14  printf("tab[%d] = %c\n", i, tab[i]);
15  |
16  }
```

```
tab[0] = e
tab[1] = x
tab[2] = e
tab[3] = m
tab[4] = p
tab[5] = l
tab[6] = e
tab[7] =
```

# Tableau en C: Exemple 2'

```
8 #define N 8
9 char tab[N] = "exemple";
10 main()
11 {
12     int i;
13     for (i = 0; i < N; i++)
14         printf("tab[%d] = %c\n", i, tab[i]);
15
16     printf("Nombre de caracteres du tableau = %d\n", N);
17 }
18
```

tab[0] = e  
tab[1] = x  
tab[2] = e  
tab[3] = m  
tab[4] = p  
tab[5] = l  
tab[6] = e  
tab[7] =  
Nombre de caracteres du tableau = 8



---

# EXERCICES PARTIE 2

# SÉANCE 2

# Introduction

- Pour l'instant, un programme est une séquence d'instruction.
- Seul problème: Pas de partage des parties importantes ni réutilisation.

- Exemple:

```
do {  
    printf ( “Entrez le nombre de points du joueur: “);  
    scanf ( “ %d “, &nb);  
} while ((nb<0) or (nb>100));
```

# Introduction

- Si je veux réutiliser cette tâche dans plusieurs parties du programme, la solution est de copier-coller cette partie dans les différents endroits où je veux l'utiliser.
- Sauf qu'il est déconseillé de copier-coller dans un programme;
- D'où la nécessité d'utiliser des **FONCTIONS**.



# FONCTIONS

# Fonctions

- Il est souhaitable, pour diverses raisons, de décomposer un problème en plusieurs sous-tâches, et de programmer ces sous-tâches comme des blocs indépendants.
- Une fonction est un sous-programme permettant de calculer et de "retourner" un seul résultat de type simple à partir d'une liste de paramètres .
- Le terme utilisé pour retourner le résultat est :  
*return*.

# Fonctions

- Une fonction est objet logiciel qui est caractérisée par:
  - *Un corps*: portion du programme à réutiliser ou mettre en évidence, qui a justifié la création de la fonction.
  - *Un nom*: par lequel on désignera cette fonction,
  - *Des paramètres*: appelés aussi arguments ou entrées, représentant un ensemble de variables extérieures à la fonction dont le corps dépend pour fonctionner,
  - *Un type et une valeur de retour*: ce que la fonction renvoie au reste du programme.



# Fonctions

---

- Pour utiliser la fonction en C, on doit avoir trois phases:
  - Définition de la fonction,
  - Déclaration de la fonction,
  - Appel de la fonction,

# Fonctions

- La définition d'une fonction est la définition de son type de retour.
- La fonction se termine par l'instruction *return*:
  - *return (expression);* : *expression du type de la fonction.*
  - *return;* / *return();* : *fonction sans type; dans ce cas, le type de la fonction est : void*

# Fonctions: Syntaxe

## ■ Déclaration:

```
type_de_retour  nom_fonction(liste de paramètre(s)) {  
    déclarations locales si nécessaire  
    calcul et retour (avec return) du résultat  
}
```

## ■ Appel:

- Comme en algorithmique, l'appel dépend de son utilisation:

```
Nom_variable = nom_fonction (liste des paramètres);
```

```
printf("message ou/et format d'impression", nom_fonction(liste  
    des paramètres));
```

```
nom_fonction(liste des paramètres); //fonction type void
```

# Fonctions: Remarque

- Toute fonction ne peut appeler que des fonctions déclarées avant elle ou elle-même.

```
... f1 (..) {
```

```
...}
```

```
... f2 (...) {
```

```
...}
```

```
void main (...) { ... }
```

- La fonction main peut appeler f1,f2
- La fonction f2 peut appeler f1, f2
- La fonction f1 peut appeler f1

# Fonctions: Remarque

- Toute fonction ne peut appeler que des fonctions déclarées avant elle ou elle-même.

```
... f1 (..) {  
...}  
... f2 (...) {  
...}  
void main (...) { ... }
```

Lorsqu'une fonction s'appelle elle-même, on dit qu'elle est "réursive".

- La fonction main peut appeler f1, f2
- La fonction f2 peut appeler f1, f2
- La fonction f1 peut appeler f1



# Prototype

- La règle précédente est très contraignante.
- *Prototype:*
  - En début de programme on donne le type de chaque fonction , son nom, le nombre et les types des arguments,
  - Information suffisante pour le compilateur.

# Les fichiers Header

- Une autre solution existe pour éviter le problème de compilation des programmes, est la création de votre propre « bibliothèque ».
- Pour se faire il faut avoir deux fichiers en plus de votre fichier programme principal:
  - Un fichier ".h" où je déclare les prototypes de toutes les fonctions,
  - Un fichier ".c", où j'insère les directives de mes fonctions,
  - Sans oublier de mentionner #include "nomfichier.h" au début du fichier '.c' et le fichier du programme principal.

# Attention

- `int a; :`
  - Déclaration de variable non initialisée,
- `int a(); :`
  - Prototype de fonction sans paramètre,
- `int a = 2;`
  - Déclaration/initialisation de variable,
- `a(7);`
  - Appel de fonction à un argument.

# Transmission des Paramètres

- Les paramètres sont associés aux arguments suivant l'ordre de déclaration.
- En C, cette association se fait par **COPIE** de la valeur du paramètre dans l'argument. Chaque argument est en fait une variable locale de la fonction. La fonction travaille sur l'argument.
- On parle du passage par valeur.

# Transmission des Paramètres

- Exemple:
  - Quel est le résultat de ce programme?

```
void f(int a){  
    a=a+1;}  
void main(){  
    int b;  
    b=0;  
    f(b);  
    printf("%d\n",b);}
```

# Transmission des Paramètres

- Si l'on veut qu'une fonction modifie un paramètre, on ne passe pas la variable mais l'adresse de la variable.
- Il y a copie de l'adresse de la variable. Dans la fonction on va chercher la variable par son adresse.

# Transmission des Paramètres

- Rappels :
  - opérateur & : &variable → adresse de la variable
  - opérateur \* : \* variable → valeur qui se trouve à l'adresse de cette variable.

## Exemple

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

```
void change (int v);
```

```
void main ( ) {
```

```
    int var = 5;
```

```
    change (var);
```

```
    printf( "main: var = %d\n", var); }
```

change: v = ?

main: var = ?

```
void change (int v){
```

```
    v *= 100;
```

```
    printf("change: v = %d\n", v);}
```



# Passage des Paramètres

## Exemple

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

```
void change (int v);
```

```
void main ( ){
```

```
    int var = 5;
```

```
    change (var);
```

```
    printf( "main: var = %d\n", var); }
```

**change: v = 500**

**main: var = 5**

```
void change (int v){
```

```
    v *= 100;
```

```
    printf("change: v = %d\n", v);}
```

## Exemple

```
#include <stdio.h>
int change (int v);
void main (void) {
    int var = 5;
    int valeur;
    valeur = change (var);
    printf("main: var = %d\n", var);
    printf("main: valeur= %d\n", valeur);}

int change (int v) {
    v *= 100;
    printf ("change: v = %d\n", v);
    return (v+1);}
```

change: v = ?  
main: var = ?  
main: valeur= ?

# Fonctions: Exemple 1

- Écrire un programme qui contient une fonction permettant de retourner la plus grande valeur parmi deux réels saisis au clavier.

# Fonctions:

## Exemple 2

- Ecrire une fonction qui retourne vrai ou faux selon la parité d'un entier saisi au clavier

```
1 #include <stdio.h>
2
3 int c;
4
5 char* parite(int a)
6     if(a%2 == 0)
7         return "vrai";
8     else
9         return "faux";
10 }
11
12 void main(){
13     printf("Entrez une valeur d'un entier:\n");
14     scanf("%d",&c);
15     printf("Le resultat est:%s\n",parite(c));
16 }
```

```
Entrez une valeur d'un entier:
6
Le resultat est:vrai
```



---

# FONCTIONS RÉCURSIVES



# La Récursivité

---

- Il est parfois difficile de définir un objet,
- Et il est parfois plus simple de le définir en fonction de lui-même.
- Ce procédé s'appelle la récursivité.
- On peut utiliser la récursivité pour définir des ensembles, des suites, des fonctions.

# Type de Récursivité

- Récursivité Simple:
  - Une fonction simplement récursive, c'est une fonction qui s'appelle elle-même une seule fois,
  
- Récursivité Multiple:
  - Une fonction peut exécuter plusieurs appels récursifs
    - Typiquement deux, parfois plus!!



---

# BON COURAGE