



Université Chouaïb Doukkali
Faculté des Sciences El Jadida



PROGRAMMATION II (en langage C)

Programmation en langage C

Plan

1

Rappels (opérateurs, structures de contrôles, tableaux, ...)

2

Les fonctions

3

Les pointeurs

4

L'allocation dynamique

5

Les chaînes de caractères

6

Les types composés (structures & énumérations)

7

Les fichiers

8

Le préprocesseur

Programmation en langage C

Plan

1 Rappels (opérateurs, structures de contrôles, tableaux, ...)

2 Les fonctions

3 Les pointeurs

4 L'allocation dynamique

5 Les chaînes de caractères

6 Les types composés (structures & énumérations)

7 Les fichiers

8 Le préprocesseur

Programme = structures de données + algorithmes

Les **données** d'un problème à résoudre via la programmation peuvent être **simples** ou **complexes**, la programmation doit disposer de **structures** pour pouvoir les représenter. On parle alors de **structures de données**.

- Une structure de données est une organisation logique des informations destinées à simplifier leur traitement.
- Le choix d'utiliser une structure de données appropriée à un traitement informatique peut faire baisser de manière significative la complexité algorithmique.

1. **Données simples** : sont de type simple : entier, réel, caractère.
2. **Données complexes** : composées de types simples. On distingue deux types essentiels de ces structures de données :
 - **Les tableaux** : Les données sont de même type et directement accessibles à tout moment.
 - **Les enregistrements** : Si la donnée est formée de plusieurs composantes pas nécessairement du même type, mais restent accessibles directement.

Les structures de données peuvent être :

■ **Linéaires :**

- Tableaux
- Listes chaînées
- Piles
- Files

■ **Non linéaires :**

- Arborescentes: Arbres
- Relationnelles: Graphes

- Une structure est un assemblage de variables qui peuvent avoir différents types.
- Une structure est généralement composée d'au moins deux « sous-variables », sinon elle n'a pas trop d'intérêt.
- Il n'y a pas de limite au nombre de variables dans une structure.
- Les objets contenus dans la structure sont appelés champs de la structure.
- Les structures sont généralement définies dans les fichiers .h

Syntaxe :

```
struct NomDeVotreStructure  
{  
    Type 1 membre1;  
    Type 2 membre2;  
    .....  
    Type N membreN;  
};
```

membre1, membre2 ...: les champs constituant la structure.

Exemple : une structure qui permet de stocker à la fois la valeur de l'abscisse (x) et celle de l'ordonnée (y) d'un point

```
struct Coordonnees  
{  
    int x; // Abscisses  
    int y; // Ordonnées  
};
```

Exemple :

Une structure qui permet de stocker diverses informations sur une personne :

```
struct Personne
{
    char nom[100];
    char prenom[100];
    char adresse[1000];
    int age;
};
```


4

Les Structures

Utilisation d'une structure

```
#include "StructPersonne.h" // Inclusion du .h qui contient les structures
```

```
struct Personne
```

```
{  
    char nom[100];  
    char prenom[100];  
    char adresse[1000];  
    int age;  
};
```

```
int main()
```

```
{  
    struct Personne P1, P2.
```

```
    .....
```

```
}
```

- Utilisation de **typedef** : pour ne plus écrire le mot « struct » à chaque définition d'une variable de type structure.
- Dans le fichier .h qui contient la définition de notre structure, on ajoute l'instruction typedef avant la définition de la structure.
- Elle sert à créer un alias de structure

Syntaxe :

typedef struct NomDeLaStructure NomEquivalent;

Exemple :

Fichier.h

```
typedef struct Coordonnees LesCoordonnees;  
struct Coordonnees  
{  
    int x;  
    int y;  
};
```

Fichier.c

```
int main()  
{  
    LesCoordonnees point; // L'ordinateur comprend qu'il s'agit de "struct  
                           Coordonnees" grâce au typedef  
    return 0;  
}
```

4

Les Structures

Utilisation d'une structure

```
typedef struct Personne Personne;
```

```
struct Personne  
{  
    char nom[100];  
    char prenom[100];  
    char adresse[1000];  
    int age;  
};
```

```
int main()  
{  
    Personne P1, P2.
```

```
.....
```

```
}
```


4

Les Structures

Utilisation d'une structure

```
typedef struct Personne
{
    char nom[100];
    char prenom[100];
    char adresse[1000];
    int age;
} Personne;

int main()
{
    Personne P1, P2.

    .....
}
```

Modifier les composantes de la structure

Pour accéder à chaque composante d'une structure, on utilise la syntaxe :

variable.nomDeLaComposante

➤ Le point fait la séparation entre la variable et la composante.

Modifier les composantes de la structure

Exemple 1: accéder aux valeurs des coordonnées x et y de la variable point et les modifier

```
int main()
{
    LesCoordonnees point;

    point.x = 10;
    point.y = 20;

    return 0;
}
```

4

Les Structures

Utilisation d'une structure

```
typedef struct Personne
{
    char nom[100];
    char prenom[100];
    char adresse[1000];
    int age;
} Personne;

int main()
{
    Personne etudiant;
    printf("Quel est votre nom ? ");
    scanf("%s", etudiant.nom);
    printf("Quel est votre age ? ");
    scanf("%d", &etudiant.age);
    printf("L'etudiant %s a %d ans", etudiant.nom, etudiant.age);
    .....
}
```

Modifier les composantes de la structure**Remarque :**

utilisateur.nom : représente l'adresse du tableau de char nom

utilisateur.nom[i] : le caractère numéro « i » du champ nom de la structure Personne

Initialiser une structure

Pour les structures, l'initialisation ressembler à celle d'un tableau.

Exemple 1 :

LesCoordonnees point = {0, 0};

Cela définira, dans l'ordre, **point.x = 0** et **point.y = 0**

Exemple 2 :

```
int main()  
{  
    Personne etudiant = {"", "", "", 0};  
    .....  
}
```

Utilisation globale d'une structure

Il est possible d'affecter à une structure le contenu d'une autre structure **définie à partir du même modèle.**

Exemple 1:

struct Coordonnees point1, point2;

On peut écrire : **point1 = point2**

Cette affectation implique que :

point1.x = point2.x
point1.y = point2.y

Utilisation globale d'une structureExemple 2:

```
int main()
{
    Personne etudiant_1, etudiant_2;
    etudiant_1 = etudiant_2;
    .....
}
```

Cette affectation implique que :

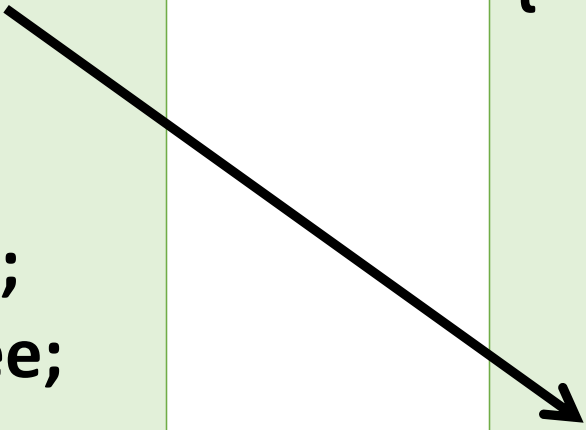
```
etudiant_1 .nom = etudiant_2 .nom
etudiant_1 .prenom = etudiant_2 . prenom
etudiant_1 .adresse = etudiant_2 .adresse
etudiant_1 .age = etudiant_2 .age
```


Structures comportant d'autres structures

Supposant que pour la structure `Personne` on souhaite ajouter un champ « date de naissance » qui est elle-même une structure avec trois champs : « jour », « mois » et « année » :

```
struct date  
{  
    int jour;  
    int mois;  
    int annee;  
}
```

```
struct Personne  
{  
    char nom[100];  
    char prenom[100];  
    char adresse[1000];  
    int age;  
    struct date dateNaissance;  
};
```

A black arrow originates from the 'struct date' definition in the left box and points to the 'struct date dateNaissance;' line in the 'struct Personne' definition in the right box, illustrating the forward reference.

Structures comportant d'autres structures

Exemple :

`Personne etudiant_1, etudiant_2;`
`etudiant_1.dateNaissance.annee` // représente l'année de naissance
correspondant à la structure `etudiant_1`. Il s'agit d'une valeur de type
« int »

`etudiant_1.dateNaissance` // représente la date de naissance
correspondant à la structure `etudiant_1`. Il s'agit d'une structure de type
« date »

On peut donc faire :

`etudiant_1.dateNaissance = etudiant_2.dateNaissance`

Tableau de structures

Possibilité de créer juste une **variable nom** et une autre **prenom** sans utiliser les structures, mais l'intérêt ici est de pouvoir créer une autre variable de type **Personne** qui aura aussi son propre nom, son propre prénom, etc.

Exemple : stocker les informations sur trois joueurs. Chaque joueur a son propre nom, son propre prénom, etc.

Personne joueur1, joueur2, joueur3;

on a 3 joueur => Créer un tableau de structure

Exemple : **Personne joueurs[3];**

Tableau de structures

Pour accéder au nom du joueur n° 0 : **joueurs[0].nom**

On peut initialiser par :

```
Personne joueurs[3] = {"Nom1", "prenom1", 25},  
{"Nom2", "prenom2", 23}, {"Nom3", "prenom3", 22}
```

L'avantage d'utiliser un tableau, c'est de **pouvoir faire une boucle pour demander les infos du joueur 1, joueur 2, ..., joueur N**, sans avoir à répéter N fois le même code. Il suffit de **parcourir le tableau joueurs** et de demander à chaque fois nom, prénom, adresse...

Exercice 1 :

Créez ce **tableau de N joueurs de type « Personne »** et demandez les infos (nom et prénom) de chacun grâce à une boucle (qui se répète tant qu'il y a des joueurs).

Affichez à la fin du programme les infos que vous avez recueillies sur chacun des joueurs.

```
typedef struct Personne
{
char nom[100];
char prenom[100];
char adresse[1000];
int age;
} Personne;

int main()
{
    int N,i;
    printf("Quel est le nombre des joueurs ? ");
    scanf("%d", &N);

    Personne joueurs[N];

    for(i=0; i<N; i++)
    {
        printf("Quel est le nom du joueur %d ? : ",i+1);
        scanf("%s", joueurs[i].nom);

        printf("Quel est le prenom du joueur %d ? : ",i+1);
        scanf("%s", joueurs[i].prenom);
    }
    for(i=0; i<N; i++)
        printf("Le nom et le prenom du joueur %d est : %s %s\n", i+1, joueurs[i].nom, joueurs[i].prenom);
    return 0;
}
```

Un **pointeur de structure** se crée de la même manière qu'un pointeur de int, de double ou de n'importe quelle autre type de base.

Exemple :

```
int main()
{
```

```
    Personne* etudiant_1 = NULL;
```

```
    Personne *etudiant_2 = NULL;
```

```
    // Pour définir plusieurs pointeurs sur la même ligne, il faut placer l'étoile devant  
    chaque nom de pointeur
```

```
    Personne *etudiant_1= NULL, *etudiant_2 = NULL;
```

```
    .....
```

```
}
```

Envoyer une structure à une fonction

Il s'agit **d'envoyer un pointeur de structure à une fonction** pour que celle-ci puisse modifier le contenu de la variable.

Exemple :

Création d'une variable de type « Personne » dans le main et **envoyer son adresse à la fonction** « initialiserPersonne ». Cette fonction aura pour rôle de mettre tous les éléments de la structure à 0.

Envoyer une structure à une fonction

```
int main()
{
    Personne etudiant;
    initialiserPersonne(&etudiant);
    return 0;    .....
}
```

```
void initialiserPersonne(Personne *P)
{
    // Initialisation de chacun des membres de la structure
    (*P).nom = "";
    (*P).prenom = "";
    (*P).adresse = "";
    (*P).age = 0;
}
```

Envoyer une structure à une fonction

-> : un raccourci pratique et très utilisé

P -> nom est équivalent à (*P).nom

```
void initialiserPersonne(Personne *P)
{
    // Initialisation de chacun des membres de la structure
    P -> nom = "";
    P -> prenom = "";
    P -> adresse = "";
    P -> age = 0;
}
```

Ne pas confondre la flèche avec le «.»

La flèche est réservée aux pointeurs, le « point » est réservé aux variables.

On considère un tableau de N joueurs, chacun est identifié par son nom, prénom et sa date de naissance selon les structures suivantes :

```
struct date  
{  
    int jour;  
    int mois;  
    int annee;  
}
```

```
struct Personne  
{  
    char nom[100];  
    char prenom[100];  
    struct date dateNaissance;  
};
```

En utilisant la structure « Personne » et « Date », écrire :

- 1. Une fonction qui permet de saisir les informations sur les joueurs.**
- 2. Une fonction qui permet d'afficher la liste des joueurs avec leurs informations.**
- 3. Une fonction qui permet de faire la recherche des informations sur un joueur selon son nom donné en paramètre.**
- 4. La fonction main qui fait appel à toutes ces fonctions.**

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

typedef struct date
{
    int jour;
    int mois;
    int annee;
}Date;

typedef struct Personne
{
    char nom[100];
    char prenom[100];
    Date dateNaissance;
}Personne;
```

1. Une fonction qui permet de saisir les informations sur les joueurs.

```
/*LECTURE DES INFORMATIONS A STOKER */
void read(Personne *joueurs[], int N)
{
    int i=0;
    for(i=0; i<N; i++)
    {
        printf("\n joueurs Num  %d\n",i+1);
        joueurs[i]=(Personne *) malloc(sizeof(Personne));
        printf("\n Nom  :");
        scanf("%s",joueurs[i]->nom);
        printf("\n Prenom :");
        scanf("%s",joueurs[i]->prenom);
        printf("\n Date de naissance :");
        printf("\n Jour :");
        scanf("%d",&joueurs[i]->dateNaissance.jour);
        printf("\n Mois :");
        scanf("%d",&joueurs[i]->dateNaissance.mois);
        printf("\n Annee :");
        scanf("%d",&joueurs[i]->dateNaissance.annee);
    }
}
```

2. Une fonction qui permet d'afficher la liste des joueurs avec leurs informations.

```
****AFFICHAGE DES INFORMATIONS ****/  
void affiche(Personne *joueurs[], int n)  
{  
    int i;  
    printf("\nN° | Nom      | Prenom    | Date de naissance    |\n");  
  
    for(i=0; i<n; i++)  
        printf("\n%d |%12s  |%12s  |%d/%d/%d  |\n",i+1,joueurs[i]->nom, joueurs[i]->prenom,  
            joueurs[i]->dateNaissance.jour,joueurs[i]->dateNaissance.mois,joueurs[i]->dateNaissance.annee);  
}
```

3. Une fonction qui permet de faire la recherche des informations sur un joueur selon son nom donné en paramètre.

```
/****** Rechercher un joueur par son nom *****/
void recherche(Personne *joueur[],int n, char *name)
{   int i;
    for (i=0;i<n;i++)
    {
        if (strcmp(joueur[i]->nom,name) == 0)
        {
            printf("\n|%d  |%12s  |%12s  |%d/%d/%d  |\n",i+1,joueur[i]->nom, joueur[i]->prenom,
                joueur[i]->dateNaissance.jour,joueur[i]->dateNaissance.mois,joueur[i]->dateNaissance.annee);
        }
    }
}
```



```
/******Programme principal*****/
```

```
main()
```

```
{
```

```
    int N;
```

```
    char Rname[20];
```

```
    printf("Quel est le nombre des joueurs ? ");
```

```
    scanf("%d", &N);
```

```
    Personne *joueurs[N];
```

```
    printf("LECTURE DES INFORMATIONS SUR LES JOUEURS :");
```

```
    read(joueurs, N);
```

```
    printf("AFFICHAGE DE LA LISTE DES JOUEURS :");
```

```
    affiche(joueurs, N);
```

```
    printf("RECHERCHE D'UN JOUEUR PAR SON NOM :");
```

```
    printf(" \nDonner le nom du joueur que vous cherchez ");
```

```
    scanf("%s", Rname);
```

```
    recherche(joueurs, N, Rname);
```

```
}
```

- Les énumérations sont des entités qui permettent de **définir un type par la liste des valeurs qu'il peut prendre**.
- Une énumération **ne contient pas de « sous-variables »** comme les structures. C'est une liste de « valeurs possibles » pour une variable.
- Une énumération **ne prend qu'une case** en mémoire et cette case peut prendre une des valeurs définies (et une seule à la fois).

Syntaxe de déclaration :

```
enum nom_enum {valeur_1, valeur_2, ..., valeur_n};
```

Exemple :

```
typedef enum Volume Volume;  
enum Volume  
{  
    FAIBLE, MOYEN, FORT  
};
```

L'énumération s'appelle ici **Volume**. C'est un type de variable personnalisé qui peut prendre une des trois valeurs qu'on a indiquées : soit **FAIBLE**, soit **MOYEN**, soit **FORT**.

Exemple :

Dans notre programme, on peut créer une variable de type **Volume**, par exemple **musique**, qui stockera le volume actuel de la musique. On peut donc **initialiser** la musique au volume **MOYEN** :

```
Volume musique = MOYEN;
```

On peut ensuite modifier la valeur de musique en lui affectant soit FAIBLE, soit FORT.

Association de nombres aux valeurs

Le compilateur associe automatiquement un nombre à chacune des valeurs possibles de l'énumération.

Les valeurs possibles {valeur_1, valeur_2, .., valeur_n}

sont codées par les entiers {0, 1, .., n-1}

Exemple :

Dans le cas de l'énumération Volume, FAIBLE vaut 0, MOYEN vaut 1 et FORT vaut 2. L'association est automatique et commence à 0.

Associer une valeur précise

Il est possible d'associer une valeur précise à chaque élément de l'énumération.

Exemple : supposons qu'on veut gérer le volume entre 0 et 100 (0 = pas de son, 100 = 100 % du son). Il est alors pratique d'associer une valeur précise à chaque élément :

```
typedef enum Volume Volume;  
enum Volume  
{  
FAIBLE = 10, MOYEN = 50, FORT = 100  
};
```

Ici, le volume FAIBLE correspondra à 10 % de volume, le volume MOYEN à 50 %, et FORT à 100%.