

Cours 6

Programmation impérative

Emna Chebbi, Revekka Kyriakoglou

Plan du cours

1 Structure

2 Enumeration

3 Chaîne de caractères

4 Fonctions predefinies

Structure



Structure

Une **structure** est un type qui permet de stocker plusieurs données dans une même variable de type structure. Une structure est composée de plusieurs champs, chaque champ correspond à une donnée.



Les données pourraient être de même type ou de types différents !



Les champs se déclarent comme des variables mais on ne peut pas initialiser les valeurs.

Exemple (1/2)

```
//declaration de la structure
struct point_3d{
    float x, y, z; //trois champs
}
```

```
//declaration d'une variable P
struct point P;
```

Pour accéder aux données x, y, z du point_3d par un point :
P.x, P.y, P.z



Question 1

■ Les données P.x, P.y, P.z sont de quel type ?

Exemple (2/2)

```
int main(void){
    struct point_3d P;
    printf("Entrer les coordonn es d'un point_3D");
    scanf("%f%f%f", &P.x, &P.y, &P.z);
    printf("Entrer les coordonn es d'un point_3D");
    float d;
    d = sqrt((P.x * P.x) + (P.y * P.y) + (P.z - P.z));
    printf("distance ((%f,%f,%f), (0,0,0)) = %f", P.x, P.y, P.z, d);

    return 0;
}
```



Le mot clé typedef peut-être utilisé pour donner un nouveau nom à un type.

```
//definition d'un type Point_3d
```

```
typedef struct point{
    float x, y, z;
}Point_3d;
```

```
Point_3d P; //Declaration d'une variable P de type Point_3d
```

Structures imbriquées



Structure

Les champs d'une **structure** peuvent être de tout type, y compris de type **structure**. Voici par exemple un type structure `T_date` qui est utilisé pour définir le champ naissance d'un autre type structure `T_pers`.

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

```
typedef struct personne {  
    char nom[30];  
    char prenom[40];  
    int age;  
    char sexe;  
    date dateNaissance;  
} personne;
```

Structures imbriquées

```
int main () {  
    personne toto;  
    strcpy (toto.nom, "Sarradin");  
    strcpy (toto.prenom, "Aline");  
    toto.age = 11;  
    toto.sexe = 'F';  
    toto.dateNaissance.jour = 10;  
    toto.dateNaissance.mois = 9;  
    toto.dateNaissance.annee = 2010;  
    printf("_Nom_: _%s_\n_Prenom_: _%s_\n_Age_: _%d_\n"  
        "Sexe_: _%c_\n_date_de_naissance_: _%d/%d/%d_\n",  
        toto.nom, toto.prenom, toto.age, toto.sexe,  
        toto.dateNaissance.jour, toto.dateNaissance.mois, t  
    return 0;  
}
```

Résultats :

Output :

```
Nom : Sarradin  
Prenom : Aline  
Age : 11  
Sexe : F  
date de naissance : 10/9/2010
```


Enumération (enum)



Enumération

L'énumération `enum` est un type de données défini par l'utilisateur. Elle est principalement utilisée pour attribuer des noms aux constantes intégrales, les noms rendent un programme facile à lire et à maintenir.

\\Declaration

```
enum jours{Lu, Ma, Me, Je, Ve, Sa, Di};
```

```
int main(void){  
    enum jours jour;  
    jour = Me;  
    printf("%d\\n", jour);  
    return 0;  
}
```

Output : 2

boucle



Nous pouvons faire une boucle en utilisant les valeurs de l'enum :

\\Declaration

```
enum jours{Lu, Ma, Me, Je, Ve, Sa, Di};
```

```
int main(void){  
    int i;  
    for (i = Lu; i <= Di; i++)  
        printf("%d_", i);  
  
    return 0;  
}
```

Output : 0, 1, 2, 3, 4, 5, 6



Deux noms d'enum peuvent avoir la même valeur.

```
enum State {Working = 1, Failed = 0, Freezed = 0};
```



Nous pouvons attribuer des valeurs à un nom dans n'importe quel ordre.



Nous ne pouvons pas utiliser les mêmes noms pour différents enum.



Question 2



Créer une variable check de type enum boolean.

Chaîne de caractères



Chaîne de caractères

Les chaînes de caractères sont définies comme un **tableau de caractères** qui se termine par un caractère spécial "`\0`".

```
//Premiere facon
```

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
//Deuxieme facon
```

```
char greeting[] = "Hello";
```

Index	0	1	2	3	4	5
-------	---	---	---	---	---	---

Variable	H	e	l	l	o	\0
----------	---	---	---	---	---	----

H	e	l	l	o	\0
---	---	---	---	---	----

Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456
---------	---------	---------	---------	---------	---------	---------

0x23451	0x23452	0x23453	0x23454	0x23455	0x23456
---------	---------	---------	---------	---------	---------

Exemple (scanf)

```
#include<stdio.h>

int main(void){
    // declaration;
    char str[50];

    printf("Quel_est_ce_cours_?\n");
    scanf("%s",str);

    // print string
    printf("%s",str);

    return 0;
}
```



Question 3



Pourquoi n'utilisons-nous pas le & dans scanf ?

Exemple

```
void affichageStr(char str[]){
    printf("Texte_:_%s",str);
}

int main(void){
    // declation
    char str[] = "Programmation";

    affichageStr(str);

    return 0;
}
```

Output :

Texte : Programmation

Initialisation de tableaux de caractères

Il est souvent proposé à placer des chaînes dans des tableaux de caractères. Mais, si la déclaration est comme ça :

```
char ch [20] ;
```

Il ne sera pas possible de transférer une chaîne constante dans ch, en écrivant une affectation du genre :

```
ch = "bonjour" ;
```

Il est possible à initialiser votre tableau de caractères à l'aide d'une chaîne constante :

```
char ch [20] = "bonjour" ;
```

Cela est équivalent à une initialisation de ch réalisée par une énumération de caractères :

```
char ch [20] = { 'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0' } ;
```

Fonctions manipulant les chaînes de caractères - 1

Il est souvent utile d'utiliser les **fonctions prédefinies** permettant de manipuler les chaînes de caractères :

- **strcpy**(chaîne1, chaîne2) : Pour copier le contenu de la chaîne2 dans l'emplacement pointé par chaîne1



Évitez d'utiliser **chaîne1=chaîne2**

- **int strlen**(const char *chaîne) : Renvoie le nombre de caractères utiles de chaîne.
- **char * strchr**(const char *chaîne, char c) : Recherche la 1ère occurrence du caractère c dans chaîne. Renvoie l'adresse sur la case de chaîne contenant le caractère s'il existe, NULL sinon.
- **int strcmp**(const char *first, const char *second) : Compare deux chaînes selon l'ordre lexicograph..
Soit first == second \Rightarrow 0.
Soit first < second \Rightarrow valeur négative.
Soit first > second \Rightarrow valeur positive.

Fonctions manipulant les chaines de caractères - 2

- `char * strcat(char *destination, const char *source)` : Copie source à la suite de destination et renvoie destination.
- `int sprintf(char *destination, const char *format)` : Même mise en forme que `printf`, sauf qu'elle stocke le résultat dans destination.
- `int gets,puts` : saisie, écriture d'une chaîne au clavier.



Remarque



Dans les exemples precedents **const char *source** signifie que la chaîne source ne peut être modifiée par la fonction.

Exemple sprintf

```
#include<stdio.h>
int main(void){
    char buffer[50];
    int a = 10, b = 20, c;
    c = a + b;
    sprintf(buffer, "Somme : %d + %d = %d", a, b, c);
    // La chaine de caract res
    //"Somme : 10 + 20 = 30"
    //est stock e dans buffer.

    return 0;
}
```

scanf VS sscanf

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

//%d ou %c ou %f dépend du type de données de la variable à lire et "x" étant la variable que nous voulons entrer.

- int **sscanf**(char *source, const char *format) : Même analyse lexicale que scanf, mais à partir de source.

```
sscanf(s1, "%s", s2);
```

//s1 et s2 étant les deux chaînes où s1 = chaîne à copier et s2 = chaîne dans laquelle nous voulons copier la première chaîne.

Exemple : print, scanf, strcpy, strcat, sprintf

```
#include <stdio.h>
#include <string.h>
int main()
{
    char nom[100], chemin[100], fichier[100], extension[100];
    printf("donner_un_chemin_\n"); scanf("%s", chemin);
    printf("donner_un_fichier_\n"); scanf("%s", fichier);
    printf("donner_une_extension_\n"); scanf("%s", extension);
    strcpy(nom, chemin);
    strcat(nom, "/");
    strcat(nom, fichier);
    strcat(nom, ".");
    strcat(nom, extension);
    printf("%s\n", nom);
    strcpy(nom, "\0"); printf("%s\n", nom);
    sprintf(nom, "%s/%s.%s", chemin, fichier, extension);
    printf("Après_concatenation_dans_nom_:_\n%s\n", nom);
    return 0;
}
```

Résultats :

Output :

```
donner un chemin
D:/local
donner un fichier

exercice_chaine
donner une extension
c
D:/local/exercice_chaine.c

le resultat apres concatenation dans la variable nom :
D:/local/exercice_chaine.c
```

Fonctions manipulant des chaines de caractères - 3

- `int atoi(const char *theString)` : Cette fonction permet de transformer une chaîne de caractères, représentant une valeur entière, en une valeur numérique de type `int`. Le terme d'`atoi` est un acronyme signifiant : ASCII to integer.
- `double atof(const char *theString)` : Cette fonction permet de transformer une chaîne de caractères, représentant une valeur flottante, en une valeur numérique de type `double`. Le terme d'`atof` est un acronyme signifiant : ASCII to float.
- `long atoll(const char *theString)` : Cette fonction, signifiant ASCII to long long, permet de transformer une chaîne de caractères, contenant la représentation textuelle d'un entier, en un entier très long (type `long long` ou `long long int` : ce sont des synonymes).