

# La programmation structurée en C

Loubna EL FAQUIH

TDI 1A

ISMONTIC 2016/2017



# Introduction

# Programmation

- Programmation structurée (langage structuré)
  - Ex. Pascal, C
- Programmation orientée objet (langage orienté objet)
  - Ex. Delphi basé sur Pascal ,C++ Builder et Visual C++ basés sur C, java, C#, ...
- Langage C :
  - langage de programmation de base
  - très proche de la machine
  - très puissant
- Plusieurs versions : C (Unix/Linux) ,Turbo C ou Turbo C++ (Dos), Visual C++ (Windows), ....

# Comment programmer en C

- Choix de la plateforme et de la version C
  - Choisir la plateforme Unix/Linux ou Windows
- Les instructions du programme source sont enregistrées dans un fichier texte
  - Extension : « .c » si programme C  
ou « .cpp » si programme C++
- Dans nos TPs, nous utiliserons **Dev C++**:  
compilateur + éditeur intégré
- Il y a d'autres IDE : Visual C++ 2008 Express, Eclipse CDT, NetBeans I.D.E...

# Exemple de programme

```
#include <stdio.h>
void main ()
{
    printf("Ceci est mon premier programme C. \n") ;
}
```

**<stdio.h>** pour "Standard Input/Output Header" ou "En-tête Standard d'Entrée/Sortie", est l'en-tête de la bibliothèque standard du C déclarant les macros, les constantes et les définitions de fonctions utilisées dans les opérations d'entrée/sortie



# Ecriture d'un programme C

Un programme C est formé de :

- Directives de pré-compilations (lignes commençant par #)
- Commentaires (limités par /\* et \*/)
- Déclarations ou définitions globales de variables /fonctions (visibles par toutes les parties du programme)
- Une fonction principale main (qui lance l'exécution d'un programme)
- Un bloc principal lié à main et délimité par { et }

# ■ Ecriture d'un programme C

## ■ à l'intérieur du bloc principal

- des déclarations ou définitions de variables/ fonctions (locales au programme principal)
- des instructions ou des blocs d'instructions délimités par { et }

## ■ Rq.

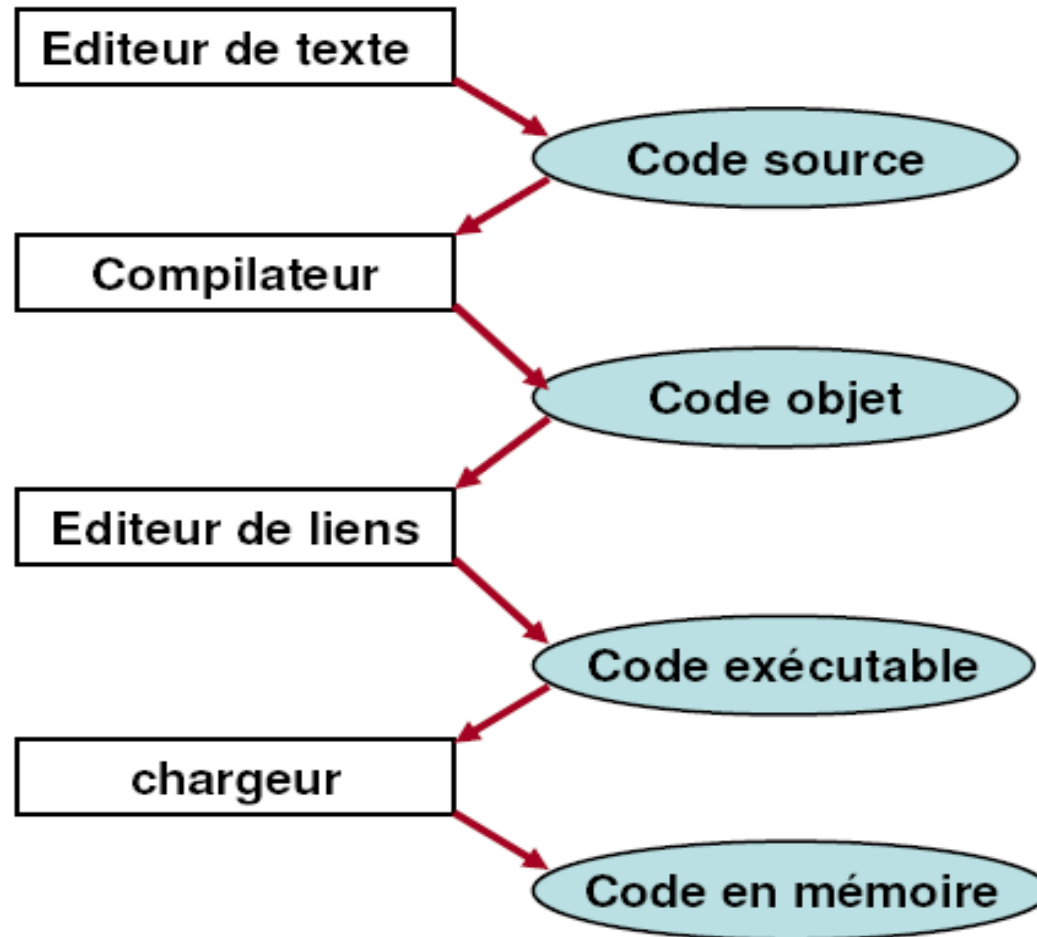
- Les instructions, les déclarations ou définitions de variables/fonctions sont terminées par un point virgule (";")

# Structure générale d'un programme C

```
[#include <Fichiers ou bibliothèques externes>]
[#define <Constantes ou macros fonctions>]
[const <Définition de constantes externes> ;]
[typedef <Définition de types de données externes> ;]
[<Définition de variables externes> ;]
[<Prototypes de fonctions externes> ;]
[<Définition de fonctions> ;]
void main ( )
{
    [<Définition de variables> ;]
    [<Prototypes de fonctions > ;]
    <Corps du Programme Principal> ;
}
[<Définition de fonctions> ;]
```



# Exécution d'un programme C





# Concepts de Base

# 1) Notion d 'environnement (Déclarations)

- Environnement : ensemble d 'objets nécessaires à la résolution d 'un problème (déclaration des objets)
- *Ex. modèle Surface-Disque*
  - *Paramètres :  $R, S$*
  - *Relation :  $S = pR_2$*
  - *Environnement :  $R, S$*
- En conclusion
  - Pour programmer un problème sur ordinateur:
    - Il faut commencer par définir son environnement (déclarer ses objets)

# 1) Notion d 'environnement (Déclarations)

- Deux sortes d 'objets de base :

- ☐ Variables
- ☐ Constantes

- Ex. Calcul de S à partir de la formule :  $PI * R * R$

- ☐ Variables : R , S
- ☐ Constante : PI (3.14)

# 1) Notion d 'environnement (Déclarations)

## ■ Représentation physique en mémoire :

### □ Variable

- **nom ou identificateur** (pour référencer la variable en mémoire)
- **Type** (pour définir sa forme ou sa taille en MC)
- **Valeur** (pour définir sa situation à un moment donné)

Le nom et le type sont fixes, La valeur est variable.

### □ Constante

- **nom**
- **Type**
- **Valeur**

Le nom, le type et la valeur sont fixes.

# 1) Notion d 'environnement (Déclarations)

## ■ Les types d 'objets :

Numérique entier (**int**)

réel (**float**)

double précision (**double**)

■ Caractère ou chaîne de caractères (**char**)

■ Logique (ou booléen) (**bool**)

■ Pointeur (\*)

# 1) Notion d 'environnement (Déclarations)

## ■ Déclaration des variables

- En C, les variables sont déclarées au début d'un bloc
- Un bloc est limité par { et }
- Un bloc est composé éventuellement de déclarations et/ou d'instructions.

```
int a,b;  
int c=0;  
float x,y,z;  
char m;  
bool n;
```

# 1) Notion d 'environnement (Déclarations)

## ■ Déclaration des Constantes

- ☐ Déclarer la constante dans la section « const »
- ☐ Utiliser la directive `#define` qui permet de remplacer constamment un symbole par la chaîne de caractères qui définit sa valeur

Rq. On peut utiliser « `#define` » aussi pour remplacer un appel de fonction ou de procédure



# 1) Notion d 'environnement (Déclarations)

Algorithmique	C
Constante C1 = 'a'	const char C1 = 'a' ; ou #define C1 'a'
C2 = '#13' (RETURN)	const char C2 = '\13' ; ou #define C2 '\13'
C3 = 'abc'	const char C3 = "abc" ; ou #define C3 "abc" C3[]
Nmax = 100	const int Nmax = 100 ; ou #define Nmax 100
	#define fin printf(" Fin de programme ")

## 2) Les types d'objets simples

### ■ Type Entier

C	Valeur sur des mots 16 bits
char	-128 ... 127
unsigned char	0 ... 255
int / short int / short	-32768 ... 32767
unsigned int / unsigned	0 ... 65535
long int ou long	-2147483648 ... 2147483647
unsigned long int	0 ... 4294967295

## 2) Les types d'objets simples

### ■ Type Réel

C	Valeur sur des mots 16 bits
Float	$3.4 \text{ E } -38 \dots 3.4 \text{ E } 38$
long float / double	$1.7 \text{ E } -308 \dots 1.7 \text{ E } 308$
long double	$3.4 \text{ E } -4932 \dots 1.1 \text{ E } 4932$

## 2) Les types d'objets simples

### ■ Type Caractère

Algorithmique	C
Caractère	char

Exemple:

Algorithmique	C
<b>c : Caractère</b> <b>c ← 'b'</b>	<b>char c ;</b> <b>c = 'b' ;</b>

Le type « char » représente en même temps le caractère et son code ASCII

## 2) Les types d'objets simples

### ■ Type Booléen

`bool x;`

`x=true;`

Les opérateurs logiques:

Algorithmique	C
Vrai	1
Faux	0
Non   Et   Ou	!   &&

### 3) Les instructions de base

#### ■ Affectation

Algorithmique	C
$\langle \text{Variable} \rangle \leftarrow \langle \text{Expression} \rangle$	$\langle \text{variable} \rangle = \langle \text{expression} \rangle ;$

#### Exemple

Algorithmique	C
$i \leftarrow i + 1$	$i = i + 1 ;$

### 3) Les instructions de base

- La spécification de format permet de préciser le format des variables à lire dans l'ordre
- Le format de chaque variable est décrit par un code commençant par le caractère « % » suivi d'un symbole qui dépend de la variable décrite

Type	Format du type
int	%d %X /* affiché en décimal ou en hexadécimal */
unsigned int	%u %o %X /* affiché en décimal, en octal ou en hexadécimal */
long	%ld
unsigned long	%lu
float	%f %e %E /* notations décimale, exponentielle avec e ou E */
double	%lf %le %IE /*notations décimale, exponentielle avec e ou E*/
long double	%Lf %le %IE
char	%c /* caractère */
char *	%s /* chaîne de caractères */

### 3) Les instructions de base

#### ■ Lecture

Algorithmique	C
<b>Lire</b> (<Variable1>, <Variable2>, ...)	<b>scanf</b> (<spécification-format>, <adresse-variable1>, <adresse-variable2>, ...) ; <u>ou</u> <Variable-Caractère> = <b>getchar</b> (); <u>ou</u> <b>gets</b> (<Variable-Chaîne>) ;



### 3) Les instructions de base

- L'adresse de la variable à lire est notée par le caractère « & » suivi du nom de la variable
- Des séparateurs peuvent être utilisés
- Si on place l'un des séparateurs suivants entre deux valeurs consécutives, on a:

Séparateur	signification
<espace>	Caractère « espace » entre les deux valeurs consécutives.
\t	Caractère « tabulation ».
\b	Caractère « curseur arrière ».
\r	Caractère « retour début de ligne ».
\n	Caractère « fin de ligne ».
\"	Caractère « " ».
\\	Caractère « \ ».
\0	Caractère « NULL » ou vide.
\a	Caractère « Bip sonore ».

### 3) Les instructions de base

#### ■ Exemples

- Lecture d'un entier a: `scanf("%d", &a)`
- Lecture d'un réel b: `scanf("%f", &b)`
- Lecture d'un caractère r : `scanf("%c", &r)`
- Lecture avec séparateur:  
`scanf("%d %f %c", &a,&b,&r)`

### 3) Les instructions de base

#### ■ Ecriture

Algorithmique	C
<b>Écrire</b> (<Expression1>[:<format>], <Expression2>[:<format>], ...)	<b>printf</b> (< spécification-format>, <expression1>, <expression2>, ...) ; <u>ou</u> <b>putchar</b> (<VariableCaractère>); <u>ou</u> <b>puts</b> (<Variable Chaîne> ) ;

### 3) Les instructions de base

#### ■ Exemples

- Afficher un message: `printf(" ***** Bonjour à tous ***** ")`  
Ou `puts(" ***** Bonjour à tous ***** ")`
- Afficher une variable de type entier: `printf("%d", a)`
- Afficher une variable de type réel: `printf("%.2f", b)`
- Afficher une variable de type caractère: `printf("%c", r)`
- Afficher message et variables:  
`printf("Les valeurs saisies : %d %f %c \n", a,b,r)`

### 3) Les instructions de base

#### ■ Exemples: Type Caractère

Le code ASCII de 'A' est 97

Posons :

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

```
void main ()
```

```
{
```

```
    char c ='A' ;
```

```
    c=c+1 ;
```

```
    printf("%c \n", c) ; /* Le résultat est 'B' */
```

```
    printf("%d \n", c) ; /* Le résultat est le code ASCII 98*/
```

```
}
```

### 3) Les instructions de base

#### ■ Exercices

Ex1. Calcul de la surface d'un disque de rayon  $R$ .

Ex2.

Connaissant le dividende  $X$  et le diviseur  $Y$ ,  
écrire un programme  $C$  qui permet de calculer le quotient  $Q$   
et le reste de la division Euclidienne de  $X$  par  $Y$



# OPERATEURS ET EXPRESSIONS

# 1) Expressions

Une expression est formée à partir:

- D'opérandes
- D'opérateurs

Une expression intervient dans :

- Les instructions D'affectation
- D'affichage
- .....

Une expression peut être:

- Une constante
- Une variable
- Un appel de fonction
- Une expression arithmétique, logique ou chaîne
- D'opérateurs

Une expression possède une valeur et un type

Les variables dans une expression peuvent être:

- Simples (entiers, réels, caractères, pointeurs)
- Structurées (tableaux, structures, unions)



## 2) Opérateurs arithmétiques

+
-
*
/
/ (si les 2 opérandes sont des entiers)
% (reste de division entière)

### ■ Exemple

C
x = 9 % 4 ; /* x= 1 */

### 3) Opérateurs Relationnels

Algorithmique	C
<	<
<=	<=
>	>
>=	>=
=	==
<>	!=

## 4) Exemples de fonctions (math.h)

Algorithmique	C
$ x $	<code>abs (x)</code>
$x * x$	<code>pow (x, 2)</code>
$\sqrt{x}$	<code>sqrt (x)</code>
$\text{Sin } (x)$	<code>sin (x)</code>
$\text{Cos } (x)$	<code>cos (x)</code>

## 5) Opérateurs Booléens

Algorithmique	C
et	&&
ou	
non	!
xou	non défini

## 6) Opérateurs Spécifiques

### ■ Les affectations simples

- Une affectation simple s'écrit:

```
<variable> = <expression> ;
```

- **L'opérateur ++ (incrément)**

L'opération d'incrément  $i = i+1$  peut être réalisée par un opérateur unaire  $++i$  ou  $i++$

- **L'opérateur -- (décrément)**

L'opération de décrément  $i = i-1$  peut être réalisée par un opérateur unaire  $--i$  ou  $i--$

## 6) Opérateurs Spécifiques

### ■ Tableau descriptif des opérateurs ++ et --

Opérateur	Notation	Notation équivalente	Signification	Exemple
++	++i (préfixé)	i=i+1	n=++i $\Leftrightarrow$ i=i+1 puis n=i	i=2; n=++i-2; $\Rightarrow$ i=3 et n=1
	i++ (postfixé)	i=i+1	n=i++ $\Leftrightarrow$ n=i puis i=i+1	i=2; n=i++-2; $\Rightarrow$ n=0 et i=3
--	--i (préfixé)	i=i-1	n--i $\Leftrightarrow$ i=i-1 puis n=i	i=2; n--i-2; $\Rightarrow$ i=1 et n=-1
	i-- (postfixé)	i=i-1	n=i-- $\Leftrightarrow$ n=i puis i=i-1	i=2; n=i-- -2; $\Rightarrow$ n=0 et i=1

## 6) Opérateurs Spécifiques

### ■ Les opérateurs d'affectation élargie

Une affectation peut être « élargie »

**Ex:  $n = n + i$**

En utilisant des opérateurs unaires comme  **$n += i$**

opérateur	Notation	Notation équivalente
$+=$	$n += i$	$n = n + i$
$*=$	$n *= i$	$n = n * i$
$/=$	$n /= i$	$n = n / i$
$\%=$	$n \% = i$	$n = n \% i$
$<<=$	$n << = i$	$n = n << i$
$>>=$	$n >> = i$	$n = n >> i$
$\&=$	$n \& = i$	$n = n \& i$
$ =$	$n  = i$	$n = n   i$
$\wedge=$	$n \wedge = i$	$n = n \wedge i$

## 6) Opérateurs Spécifiques

- **Les opérateurs d'affectation élargie**

Exemples



## 6) Opérateurs Spécifiques

### ■ L'opérateur conditionnel « ? »

Une condition peut être traitée dans une affectation comme suit:

```
<variable> = <condition> ? <expression1> : <expression2> ;
```

Equivalent à:

<u>Si</u>	<condition>	<u>Alors</u>	<variable> ← <expression1>
		<u>Sinon</u>	<variable> ← <expression2>
<u>FinSi</u>			



# Structures alternatives

# 1) Schéma if ... else

Algorithmique	C
<b>Si</b> <Condition> <b>alors</b> <action1> [ <b>sinon</b> <action2>] <b>FinSi</b>	<b>If</b> ( <condition> ) <action1> ; [ <b>else</b> <action2> ;]

Algorithmique	C
<b>Si</b> taxe <1000 <b>alors</b> taux ← 0 <b>sinon</b> <b>si</b> taxe <2000 <b>alors</b> taux ← 1 <b>sinon</b> taux ← 2 <b>FinSi</b> <b>FinSi</b>	<b>if</b> (taxe <1000.) taux = 0 ; <b>else if</b> (taxe <2000.) taux = 1 ; <b>else</b> taux = 2 ;

## 2) Schéma Case

C
<pre>switch (&lt;variable&gt;) {   case &lt;Valeur1&gt;: &lt;actions1&gt;;                   [break] ;   case &lt;Valeur2&gt;: &lt;actions2&gt;;                   [break] ;   ...   case &lt;Valeurn&gt;: &lt;actionsn&gt;;                   [break] ;   [default : &lt;actions&gt;]; }</pre>

## 2) Schéma Case (Exemple)

```
int j;  
printf("Donner un numero entre 1 et 7: ");  
scanf("%d",&j);  
switch(j)  
{  
    case 1: { printf("Lundi \n");break; }  
    case 2: printf("Mardi \n");break;  
    case 3: printf("Mecredi \n");break;  
    case 4: printf("Jeudi \n");break;  
    case 5: printf("Vendredi \n");break;  
    case 6: printf("Samedi \n");break;  
    case 7: printf("Dimanche \n");break;  
    default: printf("Incorrect \n");break;  
}
```



# Structures itératives

## 1) Boucle While (Tantque )

Algorithmique	C
<b>Tant que</b> <condition> <b>faire :</b> <action> <b>Fin tant que</b>	<b>while</b> (<condition>) <action> ;

<action> peut être une action simple ou composée  
Dans ce dernier cas, elle sera délimitée par: { et }

# 1) Boucle While (Tantque )

## Exemple

Algorithmique	C
<pre>i ← 0 s ← 0 Tant que i &lt;= 10 faire :     i ← i + 1     s ← s + i Fin tant que</pre>	<pre>i = 0 ; s = 0 ; while (i &lt;=10) {     i = i + 1 ;     s= s + i ; }</pre>



## 2) Boucle do ... while (Faire .. Tantque)

Algorithmique	C
<b>Répéter :</b> <Action> <b>Jusqu'à :</b> <condition>	<b>do</b> <Action> ; <b>while</b> (non <condition>) ;

### Exemple

Algorithmique	C
<b>i</b> ← 0 <b>Répéter :</b> <b>i</b> ← <b>i</b> + 1 <b>Jusqu'à :</b> <b>i</b> = 10	<b>i</b> = 0 ; <b>do</b> <b>i</b> = <b>i</b> + 1 ; <b>while</b> ( <b>i</b> != 10) ;

### 3) Boucle for (pour)

Algorithmique	C
<b>Pour</b> <var> allant de <val initiale> à <val finale> <b>faire</b> <action> <b>Fin Pour</b>	<b>for</b> ([<var>=<val initiale>] ; [<var> <= <val finale>] ; [<var>=<var> + <pas-variation>] ) <action> ;

#### Exemple

Algorithmique	C
<b>S</b> ← 0 <b>Pour</b> i allant de 1 à 10 <b>faire</b> <b>s</b> ← <b>s</b> + i <b>Écrire</b> ('i = ', i, ' et ' , 's = ', s) <b>FinPour</b>	<b>s</b> = 0 ; <b>for</b> (i = 1 ; i <= 10 ; i = i + 1) { <b>s</b> = <b>s</b> + i ; <b>printf</b> ("i = %d et s = %d \n", i, s) ; }



# Variables Structurées

# 1) Tableaux

## ■ Notion de tableau

- Un tableau : collection de variables de même type
- Déclaré par: son nom, son type, ses dimensions, ses bornes

Chaque élément du tableau est représenté par:

- Le nom du tableau
- Un ou plusieurs indices placés chacun entre crochets

# 1) Tableaux

## ■ Tableau à une dimension (Vecteur)

C
<pre>#define Nmax &lt;valeur&gt;     ou const int Nmax = &lt;valeur&gt;; &lt;Type-données&gt; Table [Nmax];</pre>

C
<pre>#define Nmax 10     ou const int Nmax = 10;</pre>
<pre>float T[Nmax] ; /*premier indice : 0 */     ou /* en cas d'initialisation */ float T[Nmax]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10} ;     ou /* en cas de qq initialisations */ float T[Nmax]={1, , , ,0, , , 0, ,1} ;</pre>

## Exemple

`float X[10];` /\* premier indice : 0 , dernier indice 9 \*/

# 1) Tableaux

## ■ Tableau à deux dimensions (Matrice)

C
<code>#define Nmax &lt;valeur&gt;</code> <b>ou</b> <code>const int Nmax = &lt;valeur&gt;;</code>
<code>&lt;Type-données&gt; Table [Nmax] [Nmax];</code>

### Exemple

```
float M[10][20];    /* premiers indices : 0, 0 */  
                    /* derniers indices  : 9, 19 */
```

# 1) Tableaux

## ■ Tableau à deux dimensions (Matrice)

C
<pre>#define Nmax 10</pre>
<pre>    <u>ou</u></pre>
<pre>const int Nmax = 10;</pre>
<pre>float M[Nmax][Nmax]; /*premiers indices : 0, 0*/</pre>
<pre>    <u>ou</u> /* en cas d'initialisation */</pre>
<pre>float M[Nmax][Nmax]={1,2,3,4,5,6,7,8,9,10},</pre>
<pre>                    {1,2,1,4,1,6,1,8,1,10},</pre>
<pre>                    ...};</pre>
<pre>    <u>ou</u> /* en cas de qq initialisations */</pre>
<pre>float M[Nmax][Nmax]={1,0,1,,,,,,0}, {0,,,1,,,,, 0},</pre>
<pre>                    ,,,,...};</pre>

# 1) Tableaux

## ■ Tableau à deux dimensions (Matrice)

Écrire un programme  
qui remplit une  
matrice de la façon  
suivante :

0	1	1	1	1
-1	0	1	1	1
-1	-1	0	1	1
-1	-1	-1	0	1
-1	-1	-1	-1	0



## 2) Enregistrement

### ■ Définition

- Autres noms: variable composée, structure
- Un ensemble d'informations de types différents
- Chaque information est représentée par un champ
- Chaque **champ** est désigné par : **nom** et **type**

## 2) Enregistrement

### ■ Type Enregistrement ou (Structure)

Algorithmique	C
<p><u>Type</u> &lt;Nom de type&gt; = <u>Enregistrement</u>     &lt;Nom de champ1&gt; :&lt;Type de données&gt;     &lt;Nom de champ2&gt; :&lt;Type de données&gt;     ...      <u>FinEnregistrement</u></p>	<pre>typedef struct {     &lt;Type de données&gt; &lt;Nom de champ1&gt; ;     &lt;Type de données&gt; &lt;Nom de champ2&gt; ;     ... } &lt;Nom de type&gt; ;</pre> <pre>struct &lt;Nom de type&gt; {     &lt;Type de données&gt; &lt;Nom de champ1&gt; ;     &lt;Type de données&gt; &lt;Nom de champ2&gt; ;     ... } ;</pre>

## 2) Enregistrement

### ■ Type Enregistrement ou (Structure)

#### Ex.

Un abonné au téléphone :

- code (entier)
- nom (20c)
- prénom (20c)
- adresse (40c)
- téléphone (9c)

Abonné est déclaré par :

```
typedef struct {  
    int code;  
    char *nom;  
    char *prenom;  
    char *adresse;  
    char *telephone;  
} abonne;
```

## 2) Enregistrement

### ■ Type Enregistrement ou (Structure)

**Ex.**

Calculer la somme de deux nombres complexes représentés en coordonnées cartésiennes.

### **Solution**

{ Données :  $z_1, z_2$  (complexes)  
Résultat :  $z$  (complexe)

## 2) Enregistrement

### ■ Type Union

L'union en C, permet d'interpréter l'espace mémoire d'une variable de plusieurs façons différentes (économie d'espace mémoire) :

réserver un seul espace pour toutes les interprétations  
au lieu de  
un espace pour chacune des interprétations

Le type « union » est défini par :

```
typedef union
{
    <Type de données> <Nom de champ1> ;
    <Type de données> <Nom de champ2> ;
    ...
} <Nom de type> ;
```

## 2) Enregistrement

### ■ Type Union

ou

```
union <Nom de type>
{
    <Type de données> <Nom de champ1> ;
    <Type de données>: <Nom de champ2> ;
    ...
} ;
```

## 2) Enregistrement

### ■ Type Union

Ex.

```
typedef union
```

```
{  
    long entier ;  
    float reel ;  
} EntierOuReel;
```

```
void main()
```

```
{ long n;  
  EntierOuReel nombre;  
  printf ("taper un nombre entier : ") ;  
  scanf ("%d", &n);  
  if (n>32767)  
      {nombre.reel= (float) n;  
        printf ("Reel correspondant : %11.4e\n", nombre.reel) ;  
      }  
  else {nombre.entier= n;  
        printf ("Entier correspondant : %d\n", nombre.entier) ;  
      }  
}
```

## 2) Enregistrement

- Exercice : Écrire un programme qui permet de lire N noms, prénoms d'élèves et leurs moyennes puis affiche les meilleurs entre eux
- Exercice : Écrire un programme qui permet de saisir N produits et permet d'afficher la liste des produits périmés





# Chaines de caractères

# 1) Définition

- Une chaîne de caractères est un tableau de char contenant un caractère nul. Le caractère nul a 0 pour code ASCII et s'écrit '\0'.
- Les valeurs significatives de la chaîne de caractères sont toutes celles placées avant le caractère nul.
- On remarque donc que si le caractère nul est en première position, on a une chaîne de caractères vide.

# 1) Déclaration

Comme une chaîne de caractères est un tableau de char, on le déclare :

**char** <nom\_chaine>[<taille\_chaine >] ;

Par exemple, on déclare une chaîne c de 200 caractères de la sorte : `char c[200] ;`

Le nombre maximal de lettres qu'il sera possible de placer dans c ne sera certainement pas 200 mais 199, car il faut placer après le dernier caractère de la chaîne un caractère nul

## 2) initialisation

On initialise une chaîne a la déclaration, et seulement a la déclaration de la sorte :

**char** <nom\_chaine>[<taille\_chaine>] = <valeur\_initialisation>;

Ou la valeur d'initialisation contient la juxtaposition de caractères formant la chaîne entourée de guillemets (" ").

Exemple:

char c[] = "bonjour"  $\leftrightarrow$

char c[] = {'b','o','n','j','o','u','r','\0'} (taille calculée par le compilateur)

### 3. Accès aux éléments

- Du fait qu'une chaîne de caractère est un tableau, il est aisé d'en isoler un élément. On teste donc si le premier caractère de c est une majuscule de la sorte :

```
if (c[0] >= 'A' && c[0] <= 'Z')  
    printf("Cette phrase commence par une majuscule \n");  
else  
    printf ("Cette phrase ne commence pas par une majuscule \n") ;
```

- Cette propriété permet aussi d'afficher une chaîne de caractère par caractère :

```
int i = 0 ;  
while (c[i] != 0)  
    printf ("%c" , c [ i++]);
```

### 3. Accès aux éléments

```
char D[27];
```

```
int i;
```

```
for(i=0;i<26;i++)
```

```
D[i]='a'+i;
```

```
D[26]=0;
```

```
printf("*****Affichage avec while ***** \n");
```

```
i=0;
```

```
while(D[i]!=0)
```

```
printf(" \n %c ",D[i++]);
```

```
printf("***** Affichage avec putchar ***** \n");
```

```
i=0;
```

```
while(D[i]!=0)
```

```
putchar(D[i++]);
```

```
printf("***** Affichage avec %s ***** \n");
```

```
printf("Vous avez saisi: \n %s \n",D);
```

## 4. Affichage avec « puts »

- **puts** écrit la chaîne de caractères désignée par <Chaîne> sur stdout et provoque un retour à la ligne.

En pratique, puts(TXT); est équivalent à  
printf("%s\n",TXT);

Exemple:

```
char TEXTE[] = "Voici une première ligne.";
puts(TEXTE);
puts("Voici une deuxième ligne.");
```

# Exemples

- `char a[] = "un\ndeux\ntrois\n";` Déclaration correcte

- `char b[12] = "un deux trois";`

Déclaration incorrecte: la chaîne d'initialisation dépasse le bloc de mémoire réservé.

Correction: `char b[14] = "un deux trois";`

ou mieux: `char b[] = "un deux trois";`

- `char c[] = 'abcdefg';`

Déclaration incorrecte: Les symboles " encadrent des caractères; pour initialiser avec une chaîne de caractères, il faut utiliser les guillemets (ou indiquer une liste de caractères).

Correction: `char c[] = "abcdefg";`

- `char d[10] = 'x';`

Déclaration incorrecte: Il faut utiliser une liste de caractères ou une chaîne pour l'initialisation

Correction: `char d[10] = {'x', '\0'}` ou mieux: `char d[10] = "x";`



# Suite des exemples

- `char e[5] = "cinq";`

Déclaration correcte

- `char f[] = "Cette " "phrase" "est coupée";`

Déclaration correcte

- `char g[2] = {'a', '\0'};`

Déclaration correcte

- `char h[4] = {'a', 'b', 'c'};`

Déclaration incorrecte: Dans une liste de caractères, il faut aussi indiquer le symbole de fin de chaîne.

Correction: `char h[4] = {'a', 'b', 'c', '\0'};`

## 5. Saisie avec « gets »

Tout débordement d'indice et/ou absence de caractère nul peut donner lieu a des bugs difficiles a trouver :

```
#define N 20
char chaine[N] ;
int i ;
printf ("Saisissez une phrase : \ n") ;
gets (chaine) ;
for (i = 0 ;chaine[i] != 0 ; i++)
printf ("chaine[%d] = %c (code ASCII : %d )\n" , i , chaine[i],
    chaine[i]);
```

## 6. « fgets » au lieu de « gets »

**fgets(<chaine>, <taille>, stdin ) ;**

La taille de la chaîne saisie est limitée par <taille>, caractère nul compris. Le résultat est placé dans <chaine>.

Tous les caractères supplémentaires saisis par l'utilisateur ne sont pas placés dans <chaine>, seuls les (<taille>-1) premiers caractères sont récupérés par fgets. Nous saisissons donc la phrase de notre programme de la sorte : fgets (c , 200 , stdin ) ;

## 7. Saisie avec scanf

- scanf avec le spécificateur %s permet de lire un mot isolé à l'intérieur d'une suite de données du même ou d'un autre type.
- scanf avec le spécificateur %s lit un mot du fichier d'entrée standard stdin et le mémorise à l'adresse qui est associée à %s

### Exemple:

```
char LIEU[25];  
int JOUR, MOIS, ANNEE;  
printf("Entrez lieu et date de naissance : \n");  
scanf("%s %d %d %d", LIEU, &JOUR, &MOIS, &ANNEE);
```

## 7. Saisie avec scanf

La fonction scanf a besoin des adresses de ses arguments:

- Les noms des variables numériques (int, char, long, float, ...) doivent être marqués par le symbole '&'
- Comme le nom d'une chaîne de caractères est le représentant de l'adresse du premier caractère de la chaîne, il ne doit pas être précédé de l'opérateur adresse '&'
- L'utilisation de scanf pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.

## 8. Précédence alphabétique et lexicographique

- Pour le code ASCII, nous pouvons constater l'ordre suivant:

. . . ,0,1,2, ... ,9, . . . ,A,B,C, ... ,Z, . . . ,a,b,c, ... ,z, . . .

- Les symboles spéciaux (' ,+ ,- ,/ ,{ ,] , ...) et les lettres accentuées (é ,è ,à ,û , ...) se trouvent répartis autour des trois grands groupes de caractères (chiffres, majuscules, minuscules).
- Leur précédence ne correspond à aucune règle d'ordre spécifique.
- '0' est inférieur à 'Z' et noter '0' < 'Z'

# Exemple

- `if (C>='0' && C<='9') printf("Chiffre\n");`
- `if (C>='A' && C<='Z') printf("Majuscule\n");`
- `if (C>='a' && C<='z') printf("Minuscule\n");`

Il est facile, de convertir des lettres majuscules dans des minuscules:

- `if (C>='A' && C<='Z') C = C-'A'+'a';`

ou vice-versa:

- `if (C>='a' && C<='z') C = C-'a'+'A';`

## 9. La bibliothèque « string.h »

Cette bibliothèque propose des fonctions de maniement de chaînes de caractères, à savoir :

**strlen**(<s>) : fournit la longueur de la chaîne sans compter le '\0'  
Final

**strcpy**(<s>, <t>) : copie <t> vers <s>

**strcat**(<s>, <t>) : ajoute <t> à la fin de <s>

**strcmp**(<s>, <t>) : compare <s> et <t> lexico-graphiquement et retourne:

- 1 : si <s> précède <t>
- 0 : si <s> est égal à <t>
- 1 : si <s> suit <t>

**strncpy**(<s>, <t>, <n>) : copie au plus <n> caractères de <t> vers <s>

**strncat**(<s>, <t>, <n>) : ajoute au plus <n> caractères de <t> à la fin de <s>



## 9. La bibliothèque « string.h »

**strncmp** (<s>, <t>, <n>): comme strcmp mais limite la comparaison en n caractères

**strcmpli**(<s>, <t>) : comme strcmp mais ne distingue pas les majuscules et les minuscules

## 9. La bibliothèque « string.h »

### ■ Les fonctions de recherche

#### **strchr(chaine, caractère)**

Retourne NULL si le caractère ne se trouve pas dans la chaîne sinon un pointeur vers le premier caractère trouvé

- Ex: `printf("%s \n", strchr("Bonjour ! ceci est un cours de C", 'c'));`

- Le résultat est : " ceci est un cours de C"

#### **strrchr(chaine, caractère)**

Retourne NULL si le caractère ne se trouve pas dans la chaîne sinon un pointeur vers le dernier caractère trouvé

- Ex: `printf("%s\n", strrchr("Bonjour ! ceci est un cours de C", 'c'));`

- Le résultat est : " cours de C"

## 9. La bibliothèque « string.h »

### ■ Les fonctions de recherche

#### **strpbrk**(chaîne1, char\* lettresARechercher)

Retourne NULL si aucun caractère n'est trouvé ou un pointeur sur le premier caractère trouvé

□ Ex: printf("%s \n ", strpbrk("ceci est un cours de c", "tuo"));

■ Résultat : "t un cours de C "

#### **strstr**(chaîne, chaîneARechercher)

Retourne NULL si aucun caractère n'est trouvé ou un pointeur sur la première chaîne trouvée

□ Ex: printf("%s \n", strstr("ceci est un cours de c pas un cours de pascal", "course"));

■ Résultat : <null>

## 10. La bibliothèque « `stdlib.h` »

La bibliothèque `<stdlib>` contient des déclarations de fonctions pour la conversion de nombres en chaînes de caractères et vice-versa

### Conversion de chaînes de caractères en nombres :

**atoi**(`<s>`) : retourne la valeur numérique représentée par `<s>` comme `int`

**atol**(`<s>`) : retourne la valeur numérique représentée par `<s>` comme `long`

**atof**(`<s>`) : retourne la valeur numérique représentée par `<s>` comme `double`

### Conversion de nombres en chaînes de caractères:

**itoa** (`<n_int>`, `<s>`, `<b>`)

**ltoa** (`<n_long>`, `<s>`, `<b>`)

**ultoa** (`<n_uns_long>`, `<s>`, `<b>`)

Les espaces au début d'une chaîne sont ignorées

# 11. La bibliothèque « ctype.h »

**islower**(char c): retourne une valeur différente de 0 si c est minuscule

**isupper**(char c): retourne une valeur différente de 0 si c est majuscule

**isdigit**(char c): retourne une valeur différente de 0 si c est un chiffre décimal ('0'..'9')

**isalpha**(char c): retourne une valeur différente de 0 si c est une lettre ('a'..'z', 'A'..'Z')

**isspace**(char c): retourne une valeur différente de 0 si c est un signe d'espacement

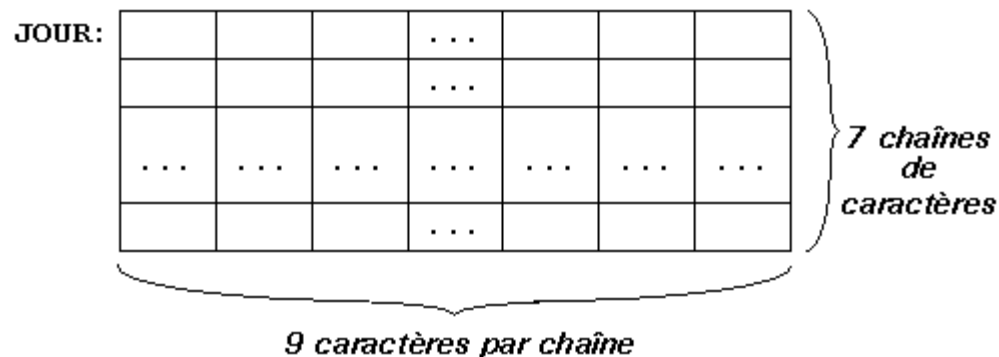
**tolower**(char c): Convertit le caractère c en minuscule

**toupper**(char c): Convertit le caractère c en majuscule

## 12. Tableaux de CDC

- Un tableau de chaînes de caractères correspond à un tableau à deux dimensions du type `char`, où chaque ligne contient une chaîne de caractères.
- Exemple : `char JOUR[7][9];`

Réserve l'espace en mémoire pour 7 mots contenant 9 caractères (dont 8 caractères significatifs).



## 12. Tableaux de CDC

- Lors de la déclaration il est possible d'initialiser toutes les composantes du tableau par des chaînes de caractères constantes:

```
char JOUR[7][9]= {"lundi", "mardi", "mercredi", "jeudi",  
"vendredi", "samedi", "dimanche"};
```

JOUR:

'l'	'u'	'n'	'd'	'i'	'\0'			
'm'	'a'	'r'	'd'	'i'	'\0'			
'm'	'e'	'r'	'c'	'r'	'e'	'd'	'i'	'\0'
...	...	...	...	...	...	...	...	...
'd'	'i'	'm'	'a'	'n'	'c'	'h'	'e'	'\0'

## 12. Tableaux de CDC

### Affichage

```
int l = 2;  
printf("Aujourd'hui, c'est %s !\n", JOUR[l]);  
=> Aujourd'hui, c'est mercredi !
```

```
for(l=0; l<7; l++) printf("%c ", JOUR[l][0]);  
va afficher les premières lettres des jours de la semaine: l m m j v s d
```

### Affectation

```
strcpy(JOUR[4], "Friday"); changera le contenu de la 5e composante du  
tableau JOUR de "vendredi" en "Friday".
```



# 13) Opérateurs sur chaînes

## ■ Exemple d'opérations sur les chaînes

Concaténation et opérateurs de comparaison

Algorithmique	C
ch1 + ch2    (+ : opérateur)	strcat (ch1, ch2)    (strcat : fonction)
= <> < <= > >=	== != < <= > >=

Longueur, copie et recherche

Algorithmique	C
Longueur (ch)	strlen (ch)
Ch2=Copie(ch1,position,longueur)	Strncpy(ch2,ch1,longueur)
P=PositionChaine(Sch,ch)	strstr(ch, sch)



# Les sous-programmes

## « Procédures et fonctions »

# 1) Définition

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

} Directives de préprocesseur

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

} Instructions } Fonction

# 1) Définition de sous programme

- C'est une partie du programme qui:
  - Possède un nom
  - Peut être appelée (par ce nom) pour exécuter une tâche bien déterminée
- C'est une partie du programme qui:
  - Procédure
  - Fonction

## 2) Procédure

Algorithmique	C
<b>Procédure</b> <nom procédure> ([ <b>variable</b> ] <paramètre1> [,<paramètre2>] [, ...] : <type> ; ...) [déclarations locales] <b>Début</b> : <actions> <b>Fin</b>	<b>void</b> <nom-procédure>([<type> <paramètre1> [, <type> <paramètre2> [, ...])  { [déclarations locales] ; <actions> ; }

Un tableau peut être déclaré par un identificateur dans la liste des arguments ou suivi de crochets [ et ]

## 2) Procédure

```
#include<stdio.h>
void afficheBonjour ( )
{
    printf ( "Bonjour \n" ) ;
}
int main ( )
{
    afficheBonjour ( ) ;
    return 0 ;
}
```

## 2. Procédure

```
#include <stdio.h>
#include <stdlib.h>
void procedure1()
{
printf("début procédure \n");
printf("fin procedure 1 \n");
}
void procedure2 ()
{
printf("début procédure 2 \n");
procedure1() ;
printf("fin procedure 2 \n");
}
```

```
void procedure3()
{
printf("début procédure 3 \n") ;
procedure1();
procedure2();
printf("fin procedure 3 \n") ;
}
main()
{
printf("debut main \n");
procedure2();
procedure3();
printf("fin main \n") ;
system("pause"); }
```

### 3) Fonction

Algorithmique	C
<b>Fonction</b> <nom fonction> ([ <b>variable</b> ] <paramètre1> [, <paramètre2>] [, ...] : <type> ; ...) : <type valeur de la fonction> [déclarations locales] <b>Début :</b> <actions> <nom fonction>← <expression> <b>Fin</b>	<b>[&lt;type valeur de fonction&gt;]</b> <nom fonction> ([<type> <paramètre1>] [, <type> <paramètre2>] [, ...])  { [déclarations locales]; <actions> ; <b>return &lt;expression&gt; ;</b> }



### 3) Fonction

Ex.

Algorithmique	C
<b>Fonction</b> factorielle (n :entier) : entier Variable F,i : entiers Début : F ← 1 For i allant de 1 à n faire F ← F * i Fin pour Factorielle ← F Fin	<b>long</b> factorielle (int n) { int i ; long F = 1 ; for (i = 1 ; i <= n ; i++) F = F * i ; return F ; }
<b>Fonction</b> fact (n :entier) : entier Début : Si n=0 alors Fact ← 1 Sinon Fact ← n * Fact(n-1) FinSi Fin	<b>long</b> fact (int n) { if (n==0) return 1; else return n * Fact(n-1); }

### 3) Fonction

#### Ex1 (Fonctions qui renvoient ou affichent un résultat)

Ecrire un programme qui définit et utilise :

- une fonction qui calcule l'aire d'un triangle
- une fonction `fact(n)` qui renvoie la factorielle du nombre `n`.
- une fonction `affiche fact(n)` qui ne renvoie rien et affiche la factorielle du nombre `n`.
- une fonction `estDivisible( a, b)` qui renvoie 1 si `a` est divisible par `b`

Rappel :

- $\text{factorielle}(n) = n ! = n (n-1) (n-2) \dots 1.$

# 3) Fonction

## Ex2 (Calculatrice)

Il s'agit de reprendre l'exercice de la calculatrice en le découpant en fonctions.

Ecrire les fonctions suivantes :

- **saisir\_operande** : elle demande de saisir un opérande (un flottant), elle effectue la saisie et elle retourne la valeur saisie
- **saisir\_operateur** : elle demande de saisir un operateur (un caractère), elle effectue la saisie et elle retourne l'opérateur saisi
- **afficher\_resultat** : elle effectue le calcul et elle affiche le résultat; elle a en paramètres les deux opérandes et l'opérateur; elle affiche un message d'erreur dans le cas de la division par zéro ou si l'opérateur est inconnu
- **continuer** : elle demande si on veut faire une nouvelle opération, elle saisit la réponse et elle retourne 1 si oui et 0 si non.

Ecrire un programme principal qui réalise la calculatrice en utilisant ces fonctions.

## 4) Programmation modulaire

### a) Prototype :

- Permet d'annoncer les fonctions à l'ordinateur
- Avec les prototypes (se terminent avec un ';'), on peut placer nos fonctions après le main.

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

```
// La ligne suivante est le prototype de la fonction aireRectangle :
double aireRectangle(double largeur, double hauteur);
```

```
int main(int argc, char *argv[])
{
    printf("Rectangle de largeur 5 et hauteur 10. Aire = %f\n", aireRectangle(5, 10));
    printf("Rectangle de largeur 2.5 et hauteur 3.5. Aire = %f\n", aireRectangle(2.5, 3.5));
    printf("Rectangle de largeur 4.2 et hauteur 9.7. Aire = %f\n", aireRectangle(4.2, 9.7));

    return 0;
}

// Notre fonction aireRectangle peut maintenant être mise n'importe où dans le code source :
double aireRectangle(double largeur, double hauteur)
{
    return largeur * hauteur;
}
```

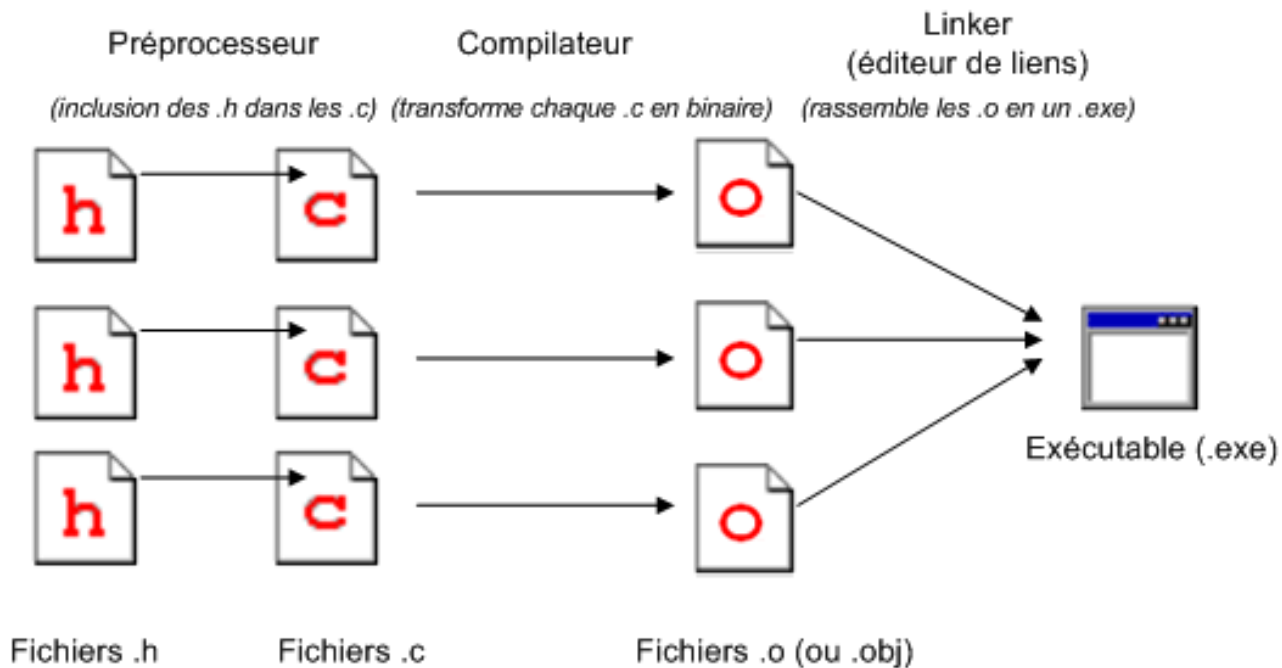
## 4) Programmation modulaire

### b) Header :

- les headers sont des fichiers d'extension **'h'** dans les quelles on peut définir les constantes, les fonctions et procédures.
- Ils facilitent la programmation modulaire
- Ils sont annoncés dans les fichiers **'c'** comme suit:
  - `include "header1.h"` avec des guillemets et pas des chevrons `<>` comme on faisait avec les fichiers de librairies standards `stdio.h`, `stdlib.h`, `string.h` ...

## 4) Programmation modulaire

### c) Compilation



## 5) Portée des variables

### a) Les variables propres aux fonctions :

```
int triple(int nombre)
{
    int resultat = 0; // La variable resultat est créée en mémoire

    resultat = 3 * nombre;
    return resultat;
} // La fonction est terminée, la variable resultat est supprimée de la mémoire
```

*Testez par vous-même : dans le *main*, affichez la valeur de la variable *résultat* déclarée dans la fonction *triple**

## 5) Portée des variables

a) Les variables propres aux fonctions :

Résultat de la compilation ➔

```
int triple(int nombre);

int main(int argc, char *argv[])
{
    printf("Le triple de 15 est %d\n", triple(15));
    printf("Le triple de 15 est %d", resultat); // Cette ligne plantera à la compilation
    return 0;
}

int triple(int nombre)
{
    int resultat = 0;

    resultat = 3 * nombre;
    return resultat;
}
```

Les variables déclarées dans une fonction ne sont accessibles que dans cette fonction



# 5) Portée des variables

## b) Les variables globales

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

int resultat = 0; // Déclaration de variable globale

void triple(int nombre); // Prototype de fonction

int main(int argc, char *argv[])
{
    triple(15); // On appelle la fonction triple, qui modifie la variable globale resultat
    printf("Le triple de 15 est %d\n", resultat); // On a accès à resultat
    return 0;
}

void triple(int nombre)
{
    resultat = 3 * nombre;
}
```

Les variables globales sont accessibles partout

## 5) Portée des variables

### c) Les Variables statiques à une fonction:

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

int incremente();

int main(int argc, char *argv[])
{
    printf("%d\n", incremente());
    printf("%d\n", incremente());
    printf("%d\n", incremente());
    printf("%d\n", incremente());

    return 0;
}

int incremente()
{
    static int nombre = 0;

    nombre++;
    return nombre;
}
```

Les variables déclarées avec le mot clé « **static** » sont accessibles uniquement dans le fichier où elles sont déclarées.

## 5) Portée des variables

### ***Conclusion:***

- 1) Une variable déclarée dans une fonction n'est accessible que dans cette fonction.
- 2) Une variable déclarée dans une fonction avec le mot-clé ***static*** devant n'est pas supprimée à la fin de la fonction, elle conserve sa valeur au fur et à mesure de l'exécution du programme
- 3) Une variable déclarée en-dehors des fonctions est une variable globale, accessible depuis toutes les fonctions de tous les fichiers source du projet
- 4) Une variable globale avec le mot-clé ***static*** devant est globale uniquement dans le fichier où elle se trouve, elle n'est pas accessible depuis les fonctions des autres fichiers.

## 6) Passage par valeur & par adresse

### a ) Passage par valeur :

Quand on crée une copie de notre variable lors de son passage en argument d'une fonction

Ex :

```
void increment(int);

int main(int argc, char *argv[])
{
    int nombre= 0;
    increment(nombre);
    printf("%d",nombre);
    getch();
    return 0;
}

void increment(int n){
    n++;
}
```

le résultat est : 0 !!  
nombre n'a pas changé !!

## 6) Passage par valeur & par argument

### a ) Passage par valeur

Un autre exemple qui montre le problème du passage par valeur quand on doit modifier la valeur de deux variables entrées en paramètres

```
void exchange(int,int);

main(){
    int a, b;
    a = 5;
    b = 10;
    printf("a = %d\nb = %d\n",a,b); // a = 5      b = 10
    exchange(a, b);
    printf("a = %d\nb = %d\n",a,b); // a = 5      b = 10
}

void exchange(int x, int y)
{
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}
```

## 6) Passage par valeur & par argument

### b ) Passage par adresse

La solution au problème de la fonction exchange est résolu par passage par adresse. càd qu'au lieu de passer une copie des variables a et b en argument de la fonction exchange , on passe leurs adresses. &a et &b

```
void exchange(int*,int*);

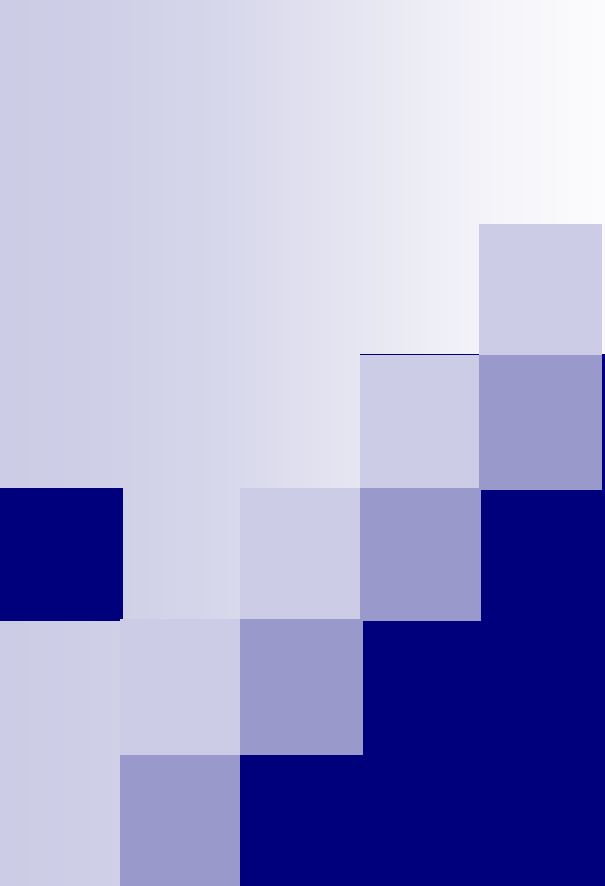
main() {
    int a, b;
    a = 5;
    b = 10;
    printf("a = %d\nb = %d\n",a,b); // a = 5      b = 10
    exchange(&a, &b);
    printf("a = %d\nb = %d\n",a,b); // a = 5      b = 10
}

void exchange(int* x, int* y)
{
    int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}
```

## 6) Passage par valeur & par argument

"Quand est-ce que je dois passer l'adresse d'une variable à une fonction ?"

- Quand on veut modifier la valeur de la variable
- Quand la variable est un tableau (obligatoire, mais implicite)
- Quand la variable est une structure (recommandé et explicite).



# STRUCTURES DE DONNÉES DYNAMIQUES



# 1) Les pointeurs (initiation)

- Définition: Un pointeur est une variable qui contient l'adresse mémoire d'une autre variable
- Déclaration : **type \*nom\_du\_pointeur ;**

Algorithmique	C
↑<Type de données>	* <Nom du pointeur>

- Ex : `int *pt_int;`
- Règle IMPORTANTE : Un pointeur doit toujours être initialisé avant utilisation.

# 1) Les pointeurs (initiation)

## ■ Deux usages possibles

- La valeur d'une variable est stockée en mémoire à une adresse donnée. Pour y accéder on utilise un pointeur.
- Un pointeur permet de gérer une variable dynamique (c'est-à-dire créer et détruire cette variable pendant l'exécution du programme chaque fois que le programmeur le désire)
- Le pointeur indique toujours l'adresse de la variable

# 1) Les pointeurs (initiation)

## ■ Utilisation

```
int a =14;
int *pt_a = &a;
/* Utilisation de la variable a ou du pointeur sur a
- Sur une variable, comme la variable a:
    ==> "a" signifie : la valeur de la variable a
    ==> "&a" signifie: l'adresse où se trouve la variable a
- Sur un pointeur, comme pt_a:
    ==> "pt_a" signifie : la valeur de la variable pt_a (qui est une adresse)
    ==> "*pt_a" signifie: la valeur de la variable qui se trouve à l'adresse
contenue dans pt_a
*/
printf("la valeur de a = %d \n",a);
printf("L'adresse de a = %p \n",&a);
printf("La valeur de la variable pointeur pt_a (adresse)= %p \n",pt_a);
printf("La valeur de la variable sur laquelle pointe pt_a = %d \n",*pt_a); // =1
```

# 1) Les pointeurs (initiation)

## ■ Initialisation

### □ Avec la valeur NULL

- `int * ptnint = NULL;`

### □ Avec l'adresse d'une de nos variables déclarées

- `int a = 10;`

- `int *pt_a = &a;`

### □ En faisant une allocation dynamique

- `int *ptnint = (int *) malloc(10);`

# Exemple

```
float *pf1,*pf2, f1, f2;  
pf1=&f1;  
pf2=&f2;  
printf("Saisir deux valeurs : ");  
scanf("%f %f",pf1, pf2);  
printf("Reel 1 = %f , son adresse = %p \n",*pf1, pf1);  
printf("Reel 2 = %f , son adresse = %p \n",*pf2, pf2);
```

## 2) Opérations sur les pointeurs

On peut déplacer un pointeur dans un plan mémoire à l'aide des opérateurs d'addition, soustraction, incrémentation et décrémentation. Les opérations possibles sur un pointeur sont les suivantes:

- Affectation d'une adresse au pointeur: `pt=&a`
- Accès à l'objet dont l'adresse est contenue dans le pointeur : `var = *pt`
- Addition d'un entier `n` à un pointeur, la nouvelle adresse est celle du nième objet à partir de l'adresse initiale
- Soustraction de deux pointeurs retourne le nombre d'objets contenus entre les deux pointeurs. Ces derniers doivent être du même type, et doivent contenir des adresses d'objets appartenant au même tableau.

### 3) les pointeurs et les tableaux

En C, l'adresse d'un tableau est l'adresse de son premier élément

```
int T[10]; <--> int *pt=(int*) malloc(sizeof(int)*10);
```

T est un pointeur constant sur la première case du tableau, on a:

```
int T[5]= {3,5,4,23,1};
```

```
int *pt;
```

```
pt = T;
```

```
printf("pt = %p , &T[0]=%p \n",pt,&T[0]);//pt = &T[0]
```

```
printf("*pt = %d , T[0]=%d \n",*pt,T[0]);//*pt = T[0]
```

```
printf("(pt+2) = %p , &T[2]=%p \n",(pt+2),&T[2]);//(pt + i ) = &T[i]
```

```
printf("*(T+3) = %d , T[3]=%d \n",*(T+3),T[3]);/*(T + i) == T[i]
```

### 3) les pointeurs et les tableaux

```
int T[5]= {3,5,4,23,1};  
int *p, i;  
for(i=0, p=T; i<5;i++, p++)  
    printf("%d ",*p);
```

Tableau à deux dimensions: pointeur de pointeur

int T[3][4]; => T est un pointeur de 3 tableaux de 4 éléments  
ou bien de trois lignes à 4 éléments.

```
T[0]   = &T[0][0];  
T[1]   = &T[1][0];  
T[i]   = &T[i][0];  
T[i]+1 = &(T[i][0])+1;
```



### 3) les pointeurs et l'allocation dynamique

Les données statiques en C occupent un emplacement défini lors de la compilation.

Les données dynamiques n'ont pas de taille définie à priori, leur création ou leur libération dépend des demandes explicites faites lors de l'exécution du programme.

Exemple:

```
char *pc=NULL; char v ='a';  
pc=&v; //Code ASCII rangé dans la case pointée par pc  
*(pc+1) = 'b'; //Code ASCII rangé dans la case plus loin (place non réservée)  
*(pc+2) = 'c';  
*(pc+3) = 'd';
```

### 3) les pointeurs et l'allocation dynamique

La fonction **malloc**: void \*malloc (t)

Alloue un espace mémoire et fournit son adresse en retour.

**Exemple:** char \*pc = (char \*) malloc(20);

Pour le convertir vers le type du pointeur déclaré, il faut utiliser un cast explicite.

**Exemple:**

```
int X, *pi;
```

```
printf(«Donner le nombre de valeurs»);
```

```
scanf(«%d», &X);
```

```
pi = (int *) malloc(X*sizeof(int));
```

### 3) les pointeurs et l'allocation dynamique

La fonction **calloc**: void \*calloc(int nb\_bloc, int tl)

Alloue un espace mémoire à nb\_bloc consécutifs, ayant chacun tl octets, et les remets à zéro.

La fonction **realloc**: void \*realloc(void \*pt, size\_t n)

Permet de modifier la taille d'une zone préalablement allouée. Le pointeur pt doit être l'adresse de la zone à modifier et la taille n représente la nouvelle taille souhaitée.

La fonction **free**: void free(void \*pointeur)

Libère un emplacement mémoire alloué

### 3) les pointeurs et l'allocation dynamique

#### **Exemple:**

```
int *pi;  
pi=(int *)calloc(n, sizeof(int));
```

Est équivalent à:

```
pi=(int *) malloc(n* sizeof(int));  
for(i =0; i< n; i++)  
*(pi+i) = 0;
```

## 4) Les pointeurs (Exemples)

### Exemple:

```
int a =1, b =2, c =3;
```

```
int *p1, *p2;
```

```
p1= &a;
```

```
p2= &c;
```

```
*p1 =(*p2)++;
```

```
p1 = p2;
```

```
p2 = &b;
```

```
*p1 -= *p2;
```

```
++*p2;
```

```
*p1 *= *p2;
```

```
a = ++*p2 * *p1;
```

```
p1 = &a;
```

```
*p2 = *p1 /= *p2;
```

	A	B	C	p1	P2
initialisation	1	2	3		
p1= &a	1	2	3	&a	
p2= &c					
*p1 =(*p2)++					
p1 = p2					
p2 = &b					
*p1 -= *p2					
++*p2					
*p1 *= *p2					
a = ++*p2 * *p1					
p1 = &a					
*p2 = *p1 /= *p2					

## 4) Les pointeurs (Exemples)

### Exemple:

```
int a =1, b =2, c =3;
int *p1, *p2;
p1= &a;
p2= &c;
*p1 =(*p2)++;
p1 = p2;
p2 = &b;
*p1 -= *p2;
++*p2;
*p1 *= *p2;
a = ++*p2 * *p1;
p1 = &a;
*p2 = *p1 /= *p2;
```

	A	B	C	p1	P2
initialisation	1	2	3		
p1= &a	1	2	3	&a	
p2= &c	1	2	3	&a	&c
*p1 =(*p2)++	3	2	4	&a	&c
p1 = p2	3	2	4	&c	&c
p2 = &b	3	2	4	&c	&b
*p1 -= *p2	3	2	2	&c	&b
++*p2	3	3	2	&c	&b
*p1 *= *p2	3	3	6	&c	&b
a = ++*p2 * *p1	24	4	6	&c	&b
p1 = &a	24	4	6	&a	&b
*p2 = *p1 /= *p2	6	6	6	&a	&b



# Gestion des fichiers

# 1) Généralités

- Un fichier est un ensemble d'informations stockées sur une mémoire de masse (disque dur, ,,,).
- En C, un fichier est une suite d'octets. Les informations contenues dans le fichier ne sont pas forcément de même type (un char, un int, une structure ...)



## 2) Manipulation des fichiers

Ensemble d'opérations possibles avec les fichiers:

- ☐ Création
- ☐ Ouverture
- ☐ Fermeture
- ☐ Lecture
- ☐ Écriture
- ☐ Destruction
- ☐ Renommage
- ☐ Positionnement

Rq: La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard `stdio.h`

## 2) Manipulation des fichiers

### a - Déclaration

```
FILE* fichier; /* majuscules obligatoires pour  
FILE */
```

## 2) Manipulation des fichiers

### b - Ouverture

`FILE* fopen(char* nom_fichier, char* mode_ouverture);`

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

main() {

    FILE* fichier1 = NULL;

    fichier = fopen("texte.txt", "r");

    getch();
}
```

## 2) Manipulation des fichiers

### b - Ouverture

FILE \***fopen**(char \*nom\_fichier, char \*mode\_ouverture);

On distingue différents modes d'ouverture :

- ☐ **"r"** Lecture seule ( suppose que le fichier existe déjà)
- ☐ **"w"** Écriture seule (si le fichier n'existe pas il sera créé)
- ☐ **"w+"** Lecture/Écriture (destruction ancienne version si elle existe!!)
- ☐ **"a"** Ajout. Vous écrirez dans le fichier, en partant de la fin du fichier( si le fichier n'existe pas il sera créé)
- ☐ **"r+"** Lecture/Écriture d'un fichier existant ( le fichier doit avoir été créé au préalable)
- ☐ **"a+"** Lecture/Écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

## 2) Manipulation des fichiers

### b - Ouverture

```
FILE *fopen(char *nom_fichier, char  
*mode_ouverture);
```

Rq : Le fichier créé ou utilisé doit être dans le même répertoire que l'exécutable ou bien il va falloir préciser son chemin

**Chemin relatif** : fichier = fopen("dossier/test.txt", "r+");

**Chemin absolu**: fichier = fopen("C:\\Program Files\\Notepad++\\readme.txt", "r+");

## 2) Manipulation des fichiers

### b - Ouverture

`FILE *fopen(char *nom_fichier, char *mode_ouverture);`

Rq : Il faut toujours tester si l'ouverture s'est bien déroulée

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

main() {

    FILE* fichier1 = NULL;

    fichier = fopen("texte.txt", "r");

    if ( fichier != NULL ) {
        // instructions
    } else {
        printf("impossible d'ouvrir le fichier texte.txt");
    }

    getch();
}
```

## 2) Manipulation des fichiers

### c - Fermeture

int **fclose**(FILE \*fichier);

Retourne 0 si la fermeture s'est bien passée, EOF en cas d'erreur.

Rq : Il faut toujours fermer un fichier à la fin d'une session

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

main() {

    FILE* fichier1 = NULL;

    fichier = fopen("texte.txt","r");

    if ( fichier != NULL ) {
        // instructions
    } else {
        printf("impossible d'ouvrir le fichier texte.txt");
    }

    fclose(fichier);
    getch();
}
```

## 2) Manipulation des fichiers

### d - Destruction

int **remove** (char \*nom);

Retourne 0 si la destruction s'est bien passée.

Att: il faut fermer le fichier avant de le détruire

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

main() {

    FILE* fichier1 = NULL;

    fichier = fopen("texte.txt", "r");

    remove("texte.txt");

    getch();
}
```



## 2) Manipulation des fichiers

### e - Ecriture

**fputc** : écrit un caractère dans le fichier (UN SEUL caractère à la fois).

**fputs** : écrit une chaîne dans le fichier

**fprintf** : écrit une chaîne "formatée" dans le fichier, fonctionnement quasi-identique à printf

## 2) Manipulation des fichiers

### e - Ecriture

#### fputc

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

int main(int argc, char *argv[])
{
    FILE* fichier = NULL;

    fichier = fopen("test.txt", "w");

    if (fichier != NULL)
    {
        fputc('A', fichier); // Ecriture du caractère A
        fclose(fichier);
    }

    getch();
    return 0;
}
```

## 2) Manipulation des fichiers

### e - Ecriture

#### fputs

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

int main(int argc, char *argv[])
{
    FILE* fichier = NULL;

    fichier = fopen("test.txt", "w");

    if (fichier != NULL)
    {
        fputs("Hello World !!", fichier);
        fclose(fichier);
    }

    getch();
    return 0;
}
```

## 2) Manipulation des fichiers

### e - Ecriture

### fprintf

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

int main(int argc, char *argv[])
{
    FILE* fichier = NULL;
    int nombre = 0;

    fichier = fopen("test.txt", "w");

    if (fichier != NULL)
    {
        // Nombre d'utilisateurs à créer
        printf("Combien d'utilisateurs ? ");
        scanf("%d", &nombre);

        // On l'écrit dans le fichier
        fprintf(fichier, "Le nombre d'utilisateur a cree est %d", nombre);
        fclose(fichier);
    }

    return 0;
}
```

## 2) Manipulation des fichiers

### f - Lecture

**fgetc** : lit un caractère

**fgets** : lit une chaîne

**fscanf** : lit une chaîne formatée

## 2) Manipulation des fichiers

### f - Lecture

#### **fgetc**

```
//Lire le premier caractère du fichier  
char c = fgetc(f);  
printf("%c**\n",c);
```

```
//Lire le fichier entier en utilisant fgetc  
while( (car = fgetc(f)) != EOF) //End Of File  
{  
    printf("%c",car);  
}
```

## 2) Manipulation des fichiers

### f - Lecture

#### **fgets**

```
char ch[100];
```

```
// lire une ligne a partir du fichier f et la mettre dans ch  
fgets(ch,sizeof(ch),f);
```

```
//Lire le fichier entier en utilisant fgets  
while(!feof(f))  
{  
    fgets(ch,sizeof(ch),f);  
    printf("%s",ch);  
}
```

## 2) Manipulation des fichiers

### f - Lecture

### fscanf

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

int main(int argc, char *argv[])
{
    FILE* fichier = NULL;
    int chiffres[3] = {0}; // Tableau de 3 chiffres

    fichier = fopen("donnee.txt", "r");

    if (fichier != NULL)
    {
        fscanf(fichier, "%d %d %d", &chiffres[0], &chiffres[1], &chiffres[2]);
        printf("Les chiffres stockés dans le fichier sont : %d, %d et %d", chiffres[0], chiffres[1], chiffres[2]);

        fclose(fichier);
    }
    getch();
    return 0;
}
```