



# PROGRAMMATION EN LANGUAGE C

---

Pr.: B. CHERKAOU  
b.cherkaoui@ucd.ac.ma

# WHAT IS C?

“C est un langage de programmation polyvalent qui se caractérise par une économie d’expression, un flux de contrôle et des structures de données modernes, et un riche ensemble d’opérateurs. Le C n’est pas un langage de « très haut niveau », ni un « grand » langage, et n’est pas spécialisé à un domaine d’application particulier. “

Kernighan & Richie, 1978

# PLAN

- I. Généralités
- II. Éléments de base de C
- III. Entrées / Sorties
- IV. Structures de Contrôle

# HISTORIQUE RAPIDE...

- Le Langage C a été conçu en 1972 par Dennis Richie et Ken Thompson
  - Deux chercheurs au Bell Labs USA
  - Objectif : développer un système d'exploitation UNIX
- En 1978, Brian Kernighan et Dennis Richie publient la définition classique du C
  - Première édition du livre "The C programming language"
- En 1983, l'ANSI décida de normaliser le langage
  - 1989: la définition de la norme ANSI C (ou C89)
  - Deuxième édition "The C programming language"
- L'ISO a repris la même norme en 1990 (ou C90)

# CARACTÉRISTIQUES DU C

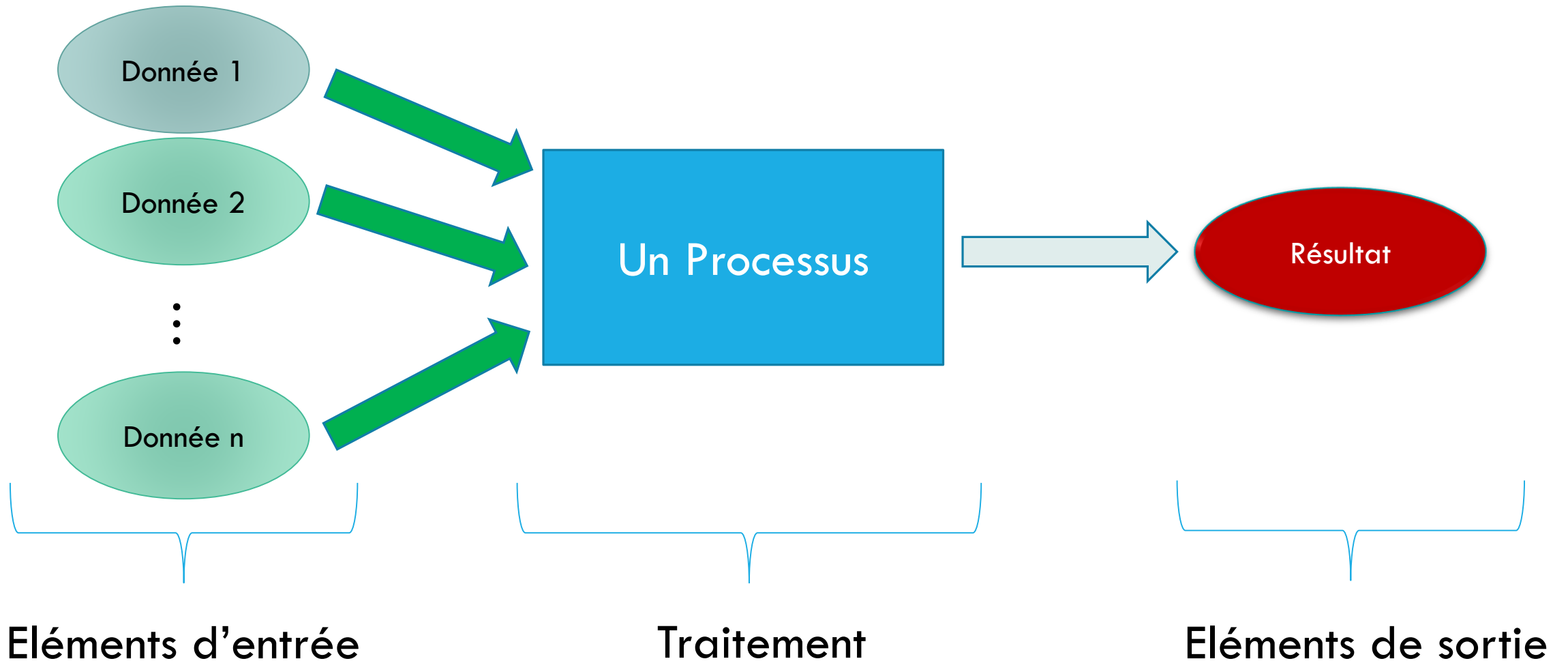
- **Universel** : n'est pas orienté vers un domaine d'application particulier (applications scientifiques, de gestion, ...)
- **Près de la machine** : offre des opérateurs qui sont proches de ceux du langage machine (manipulations de bits, d'adresses, ...) ➡ efficace
- **Modulaire** : peut être découpé en modules qui peuvent être compilés séparément
- **Portable** : en respectant le standard ANSI-C, il est possible d'utiliser le même programme sur plusieurs systèmes (hardware, système d'exploitation )

# LA COMPILATION

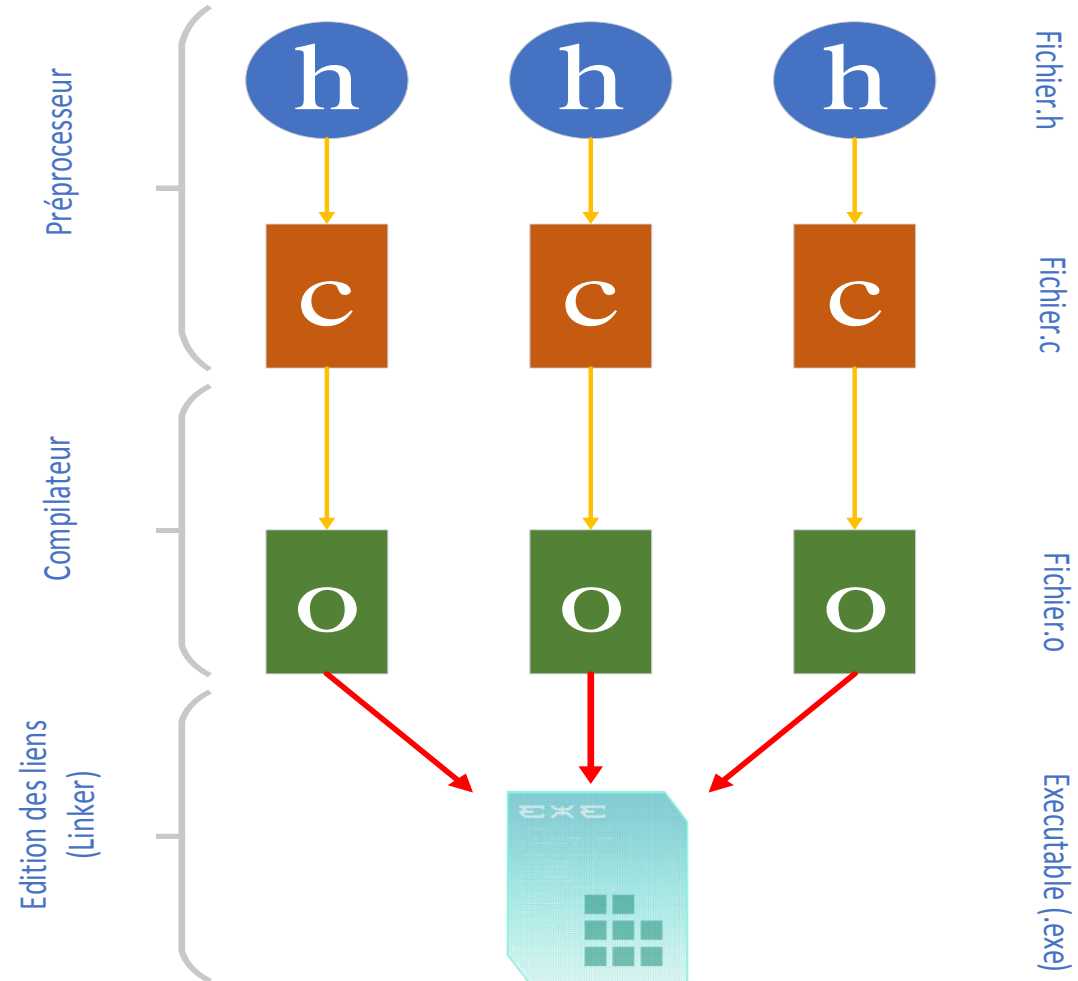
« Le fait de transformer un code source lisible par un humain en un fichier binaire exécutable par une machine »

- C est un langage compilé
  - Par opposition au langages interprétés (Python, Matlab, ...)
- Un programme C, écrit dans un fichier source, est traduit en totalité en langage machine avant exécution.
- Quatre phases :
  - Traitement par le préprocesseur
  - Compilation
  - Assemblage
  - Édition de liens

# LA COMPILATION



# LA COMPILATION





# ELÉMENTS DE BASE: PREMIER PROGRAMME

Pour réussir son programme, 3 étapes sont essentielles:

- A éditer via un éditeur de texte → Fichier source: programme.c
- Compiler le programme
- Exécuter 😊

```
#include <stdio.h>

int main()
{
    printf("Hello World");

    return 0;
}
```

# ELÉMENTS DE BASE: STRUCTURE D'UN PROGRAMME

## IMPORTANT:

Chaque instructions termine impérativement par un point-virgule ;

### ○ Partie 1: Les déclarations

Elle comporte la déclaration des fonctions des bibliothèques (bibliothèque standard ou autre) par inclusion de fichiers fournis avec le langage et peut comprendre des déclarations des variables « globales ».

### ○ Partie 2 : Le corps du programme

Tout programme C doit comporter une fonction principale main. Cette fonction est celle utilisée par le système pour exécuter le programme.

```
#include <stdio.h>

int main()
{
    instruction 1;
    instruction 2;
    .
    .
    .
    instruction n;
}
```

Partie 1:  
Déclaration

Partie 2:  
Corps du  
programme

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

- Six catégories de composants élémentaires :
  1. Les identificateurs
  2. Les mots clés
  3. Les types prédéfinis
  4. Les opérateurs
  5. Les chaînes de caractères
- Les commentaires sont enlevés par le préprocesseur

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## 1. Les identificateurs:

Le rôle d'un identificateur est de donner un nom à une entité du programme. Plus précisément, un identificateur peut désigner :

- Un nom de variable ou de fonction
- Un type défini par *typedef*, *struct*, *union* ou *enum*
- Une étiquette

Un identificateur est une suite de caractères parmi :

- Les lettres (minuscules ou majuscules, mais non accentuées)
- Les chiffres
- le ``blanc souligné" ( \_ )

Ne doit pas être un des mots clés du langage.

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## 2. Les mots clés:

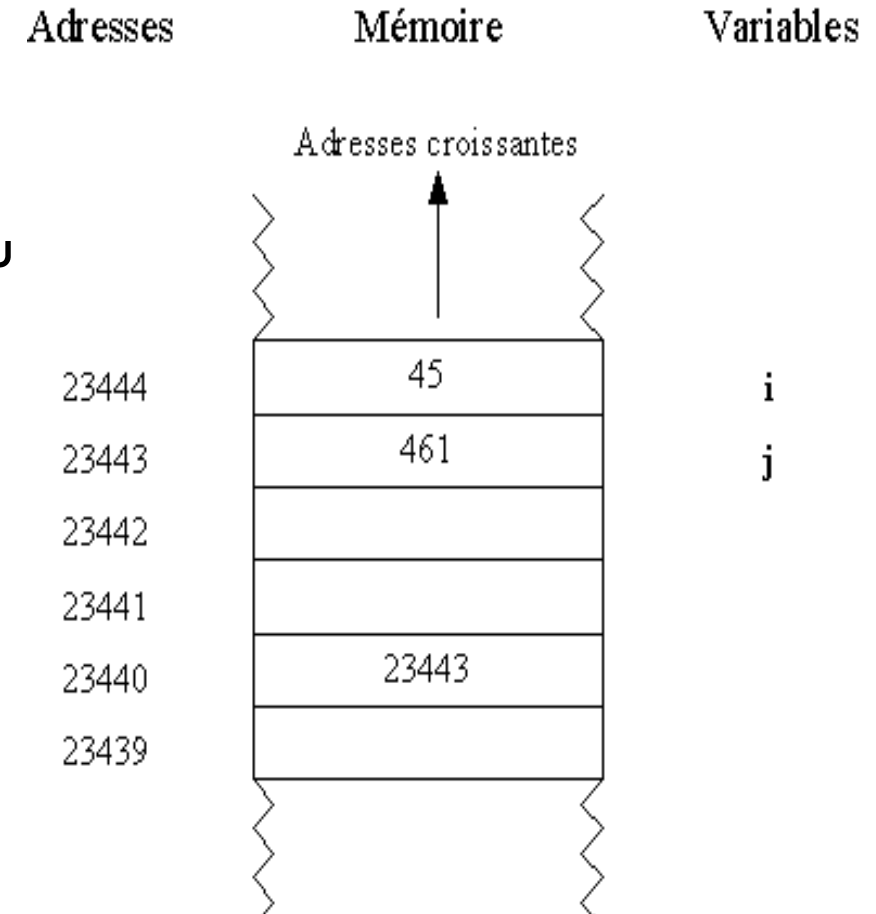
Un certain nombre de mots, appelés mots-clefs, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. L'ANSI C compte 32 mots clefs :

auto const double float int short struct unsigned  
break continue else for long signed switch void  
case default enum goto register sizeof typedef volatile  
char do extern if return static union while

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## 3. Les types prédéfinis

- Le C est un langage typé: Toute variable, constante ou fonction est d'un type précis.
- Le type définit la représentation mémoire d'un objet.
- Les types de base en C concernent:
  - Les entiers
  - Les flottants (nombres réels)
  - Les caractères



# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## Les types entiers

Le type entier représente l'ensemble des entiers relatifs (positifs et négatifs) avec plusieurs sous-types:

Type	Taille	Valeurs	Intervalle
char	8 bits	caractères	$[-128, 127]$
short	16 bits	Entiers courts	$[-32768, 32767]$
int	32 bits	Entiers	$[-2^{31}, 2^{31} - 1]$
long	64 bits	Entiers long	$[-2^{63}, 2^{63} - 1]$
unsigned char	8 bits	caractères	$[0, 255]$
unsigned short	16 bits	Entiers courts non signés	$[0, 65536]$
unsigned int	32 bits	Entiers non signés	$[0, 2^{32} - 1]$
unsigned long	64 bits	Entiers long non signés	$[0, 2^{64} - 1]$

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## Les types flottants

3 types correspondant à différentes précisions :

Type	Taille	Valeurs
float	32 bits	Flottants simple précision
double	64 bits	Flottants double précision
long double	128 bits	Flottants précision étendue



# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## Les types caractères

- Le type « **char** » représente le jeu de caractères de la machine.
- Un char en C est codé sur un octet (8 bits).
- Un caractère est encodé en utilisant un entier avant sa représentation binaire en mémoire.
- Plusieurs encodages sont utilisés:
  - ASCII (American Standard Code for Information Interchange)
    - Version originale (1960) représente 128 caractères avec les nombres de 0 à 127 sur 7 bits.
    - Version étendue (1980) représente 256 sur 8bits
  - Unicode (1990) représente 65 536 sur 16 bits

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

- L'ensemble des caractères ASCII (Version originale)

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	\$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	DEL		

## Comment lire:

R sur ligne 8 colonne 2 est encodé par 82.

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

- Les caractères non imprimables :

NOTATION EN C	CODE ASCII (hexadécimal)	ABRÉVIATION USUELLE	SIGNIFICATION
<code>\a</code>	07	BEL	cloche ou bip (alert ou audible bell)
<code>\b</code>	08	BS	Retour arrière (Backspace)
<code>\f</code>	0C	FF	Saut de page (Form Feed)
<code>\n</code>	0A	LF	Saut de ligne (Line Feed)
<code>\r</code>	0D	CR	Retour chariot (Carriage Return)
<code>\t</code>	09	HT	Tabulation horizontale (Horizontal Tab)
<code>\v</code>	0B	VT	Tabulation verticale (Vertical Tab)
<code>\\</code>	5C	<code>\</code>	
<code>\'</code>	2C	<code>'</code>	
<code>\"</code>	22	<code>"</code>	
<code>\?</code>	3F	<code>?</code>	

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## Les constantes en C

- Une constante est une valeur qui apparaît littéralement dans le code source d'un programme. Exemple: 123, 'A', "Hello", 1.5, ...
- La manière avec laquelle on écrit une constante détermine implicitement son type
- Définition des constantes symboliques à l'aide de la directive :

`#define NOM Valeur`

- Demande au préprocesseur de remplacer NOM par Valeur dans la suite du fichier source. Exemples:

`#define PI 3.14`

`#define N 100`

`#define MIN 0`

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## Variables

- Les variables servent à stocker les valeurs des données utilisées pendant l'exécution d'un programme.
- Les variables doivent être déclarées avant d'être utilisées, elles doivent être caractérisées par :



# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## Déclaration des variables

- Les déclarations introduisent les variables qui seront utilisées, fixent leur type et parfois aussi leur valeur de départ (initialisation)
- Syntaxe de déclaration en C:

`<Type> <NomVar1>,<NomVar2>,...,<NomVarN>;`

- Exemples:

`int i, j,k;`

`float x, y ;`

`double z=1.5;`

`short compteur;`

`char c=`A`;`

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

Le choix d'un identificateur (nom d'une variable ou d'une fonction) est soumis à quelques règles :

- doit être constitué uniquement de lettres, de chiffres et du caractère souligné \_ (Eviter les caractères de ponctuation et les espaces)

Correcte: PRIX\_HT, prixHT

Incorrecte: PRIX-HT, prix HT, prix.HT

- doit commencer par une lettre (y compris le caractère souligné)

Correcte : A1, A1

Incorrecte: 1A

- doit être différent des mots clés réservés du langage.

**Remarque:** C distingue entre les majuscules et les minuscules. **NOMBRE** et **nombre** sont deux identificateurs différents.

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

## 4. Les opérateurs:

Le langage C est riche en opérateurs. Outre les opérateurs standards, il comporte des opérateurs originaux d'affectation, d'incrémentation et de manipulation de bits.

Catégorie	Opérateurs	Syntaxe et remarques
Affectation	=	<code>variable = expression;</code> • Ne pas confondre avec ==
arithmétiques	+ - * / %	• Division entière et réelle : si les deux opérandes sont entières, / produira une division entière (quotient de la division). Exemple : float x; x = 3/2; → x = 1.0 x = 3.0/2; → x = 1.5 • Pas d'opérateur de puissance en C. on utilise la fonction pow(x,y) de math.h
Comparaison	< <= > >= == !=	<code>expression1 op expression2</code> • Le résultat est de type int (pas de type booléen en C): 1 si vrai, et 0 sinon.



# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

Catégorie	Opérateurs	Syntaxe et remarques
Logiques booléens	<code>&amp;&amp;</code> (le ET) <code>  </code> (le OU) <code>!</code> (le NON)	<ul style="list-style-type: none"><li>• Le résultat est de type <code>int</code>: 1 si vrai, et 0 sinon.</li><li>• L'évaluation d'une expression se fait de gauche à droite et s'arrête dès que le résultat final est déterminé.</li></ul> <p>Exemple:</p> <pre>int i, j, n; if (i!=j &amp;&amp; i&lt;n &amp;&amp; j&lt;n) i&lt;n ne sera évaluée que si i!=j est vraie j&lt;n ne sera évaluée que si i!=j et i&lt;n vraies</pre>

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

Catégorie	Opérateurs	Syntaxe et remarques
Logiques bit à bit  (bitwise operators)	&   ^  - << >>	Exemple et signification: unsigned char a = 103, b = 41;   //sur 8 bits  expr      binaire      déc      signification ----- a            0110 0111      103      valeur de a b            0010 1001      41       valeur de b a & b       0010 0001      33       et bit à bit a   b       0110 1111      111      ou bit à bit a ^ b       0100 1110      78       ou exclusif ~a           1001 1000      152      complément à 1 a >>2       0001 1001      25       décalage à droite a <<3       0011 1000      56       décalage à gauche

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

Catégorie	Opérateurs	Syntaxe et remarques
Incrémentation Décrémentation	++ --	<ul style="list-style-type: none"><li>• ++ ajoute 1 à son opérande</li><li>• -- soustrait 1 à son opérande</li><li>• S'utilisent en suffixe (<code>var++</code> et <code>var--</code>) et en préfixe (<code>++var</code> et <code>--var</code>).</li></ul> Exemple: <pre>int x, n; n = 5;</pre> <ul style="list-style-type: none"><li>• <code>x = n++</code>; incrémentation après affectation → <code>x = 5</code> puis <code>n = 6</code></li><li>• <code>x = ++n</code>; incrémentation avant affectation → <code>n = 6</code> puis <code>x = 6</code></li></ul>
Affectation composée	<code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code>	<code>variable op= expression;</code> Équivalent à: <code>variable = variable op expression;</code>

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

Catégorie	Opérateurs	Syntaxe et remarques
Opérateur conditionnel ternaire	? :	<p><i>Condition ? Expression1 : Expression2</i></p> <ul style="list-style-type: none"><li>• Le résultat est Expression1 si la condition est vraie et Expression2 sinon.</li><li>• C'est l'équivalent d'un if – else.</li></ul> <p>Exemple :</p> <pre>int a, b, max, min; min = (a&lt;=b) ? a : b; max = (a&gt;=b) ? a : b;  float x; x = (x&gt;=0) ? x : (-x);</pre>

# ELÉMENTS DE BASE: COMPOSANTS ÉLÉMENTAIRES DE C

- Vous pouvez télécharger Dev-C++ librement, par exemple sur le site:

[www.bloodshed.net](http://www.bloodshed.net)



# ENTRÉES / SORTIES:

- Il 'agit des instructions permettant à la machine de communiquer avec l'utilisateur:
- Il s'agit des fonctions de la librairie standard **stdio.h** utilisées avec les unités classiques d'entrées / sorties: Le clavier et l'écran
- La librairie standard **stdio.h** contient un ensemble de fonctions qui assurent la lecture et l'écriture des données:
  - **printf()** écriture formatée de données
  - **scanf()** lecture formatée de données

# ENTRÉES / SORTIES: FONCTION D’AFFICHAGE

- La fonction **printf**:
  - C'est une fonction d'impression formatée, les données sont converties selon le format choisi avant impression.

- Syntaxe:

`printf("Format", expression1, ..., expressionN);`

- `expression1, ...` : sont les variables et les expressions dont les valeurs sont à représenter.
- `Format` : est une chaîne de caractères qui peut contenir:
  - du texte
  - des séquences d'échappement (`'\n'`, `'\t'`, ...) ➡ Les caractères non imprimables
  - des spécificateurs de format : un ou deux caractères précédés du symbole `%`, indiquant le format d'affichage

**Remarque:** Le nombre de spécificateurs de format doit être égale au nombre d'expressions!

# ENTRÉES / SORTIES: FONCTION D’AFFICHAGE

- Spécificateurs de format pour printf:

format	conversion en	écriture
%d	int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	unsigned int	octale non signée
%lo	unsigned long int	octale non signée
%x	unsigned int	hexadécimale non signée
%lx	unsigned long int	hexadécimale non signée
%f	double	décimale virgule fixe
%lf	long double	décimale virgule fixe
%e	double	décimale notation exponentielle
%le	long double	décimale notation exponentielle
%g	double	décimale, représentation la plus courte parmi %f et %e
%lg	long double	décimale, représentation la plus courte parmi %lf et %le
%c	unsigned char	caractère
%s	char*	chaîne de caractères



# ENTRÉES / SORTIES: FONCTION D’AFFICHAGE

○ 'affichage du texte peut être contrôlé à l'aide des séquences d'échappement(caractères non imprimables) :

- `\n` : nouvelle ligne
- `\t` : tabulation horizontale
- `\a` : signal sonore
- `\b` : retour arrière
- `\r` : retour chariot
- `\v` : tabulation verticale
- `\f` : saut de page
- `\\` : back slash ( `\` )
- `\'` : apostrophe
- `\"` : guillemet

# ENTRÉES / SORTIES: FONCTION D’AFFICHAGE

## ○ Exemple 1:

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

```
main()
```

```
{
```

```
    int i=1 , j=2, N=15;
```

```
    printf("la somme de %d et %d est %d \n", i, j, i+j);
```

```
    printf(" N= %0x \n" , N);
```

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

```
    printf(" le code Ascii de %c est %d \n", c, c);
```

```
}
```

```
la somme de 1 et 2 est 3
N= f
le code Ascii de A est 65
Appuyez sur une touche pour continuer...
```

# ENTRÉES / SORTIES: FONCTION D’AFFICHAGE

## ○ Exemple 2:

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

```
main()
```

```
{
```

```
    float x=10.5, y=2.5, result;
```

```
    result = x/y;
```

```
    printf("%f divisé par %f égal à %f \n", x, y, result);
```






```
    printf("%f divisé par %f égal à %f \n", x, y, x/y);
```

```
}
```

```
10.500000 divisé par 2.500000 égal à 4.200000
10.500000 divisé par 2.500000 égal à 4.200000
```







# ENTRÉES / SORTIES: FONCTION D'AFFICHAGE

- Par défaut, les entiers sont affichés sans espaces avant ou après.
- Pour agir sur l'affichage, un nombre est placé après % afin de préciser le nombre de caractères **minimum à utiliser**
- Exemple: `printf("%4d", n);`

<code>n = 25;</code>		<code>~~20</code> (~: espace)
<code>n = 75636</code>		<code>75636</code>
<code>printf("%4X", 123);</code>		<code>~~ 7B</code>
<code>printf("%4x", 123);</code>		<code>~~ 7b</code>
<code>printf("%4o", 123);</code>		<code>~173</code>

# ENTRÉES / SORTIES: FONCTION D’AFFICHAGE

- Pour les réels, on peut préciser la largeur minimale de la valeur à afficher et le nombre de chiffres après le point décimal.
- La précision par défaut est fixée à six décimales. Les positions décimales sont arrondies à la valeur la plus proche. Exemples :

<code>printf("%f", 100.123);</code>		100.123000
<code>printf("%12f", 100.123);</code>		~~100.123000
<code>printf("%.2f", 100.123);</code>		100.12
<code>printf("%5.0f", 100.123);</code>		~~100
<code>printf("%10.3f", 100.123);</code>		~~~100.123
<code>printf("%.4f", 1.23456);</code>		1.2346

# ENTRÉES / SORTIES: FONCTION DE LECTURE

- La fonction **scanf** permet de saisir des données au clavier les convertir selon les formats spécifiés puis les stocker en mémoire.

## Syntaxe:

`scanf("formats", adresse1, adresse2, ... , adresseN);`

- **Formats:** le format de lecture de données, est le même que pour printf
- **Adresse1, Adresse2,..., AdresseN:** adresses des variables auxquelles les données seront attribuées. L'adresse d'une variable est indiquée par le **nom** de la variable précédé du signe **&**

# ENTRÉES / SORTIES: FONCTION DE LECTURE

## ○ Exemple:

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

```
main()
```

```
{
```

```
    int i , j;
```

```
    scanf("%d%d", &i, &j);
```

```
    printf("i=%d et j=%d", i, j);
```

```
}
```

Ce programme permet de lire deux entiers du clavier et les afficher à l'écran

# ENTRÉES / SORTIES:

## Exercices d'application:

- 1) Ecrire un programme qui demande deux nombres entiers à l'utilisateur, puis calcule et affiche la somme, la différence et le produit de ces nombres.
- 2) Ecrire un programme qui lit une valeur et calcule l'inverse de cette valeur, puis affiche le résultat.
- 3) Ecrire un programme qui permute les valeurs de deux entiers saisis par l'utilisateur. Affichez les entiers avant et après l'échange.
- 4) Ecrire un programme qui permet de déterminer si un entier saisi est pair ou impair.
- 5) Ecrire un programme qui lit en entrée trois entiers et affiche leur moyenne avec une précision de deux chiffres après la virgule.
- 6) Ecrire un programme qui lit en entrée un caractère alphabétique entre a et y, qui peut être soit une majuscule ou une minuscule. Et affiche la lettre qui vient juste après lui dans l'ordre alphabétique.



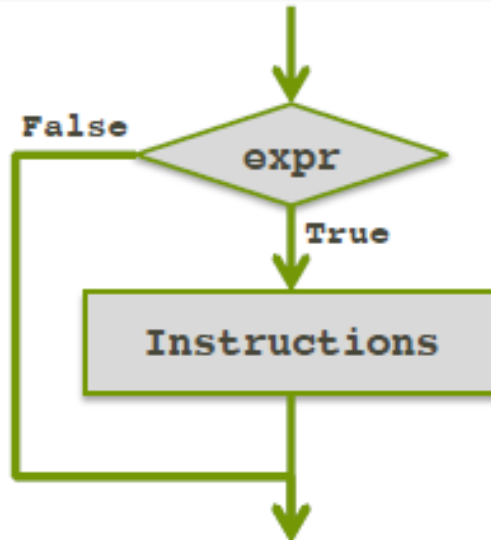
# STRUCTURES DE CONTRÔLE

- Les structures de contrôle définissent la façon avec laquelle les instructions sont effectuées. Elles conditionnent l'exécution d'instructions à la valeur d'une expression.
- Il existe deux types de structures:
  - I. Les structures alternatives (de test ou de sélection): permettre de faire des sélections, c'est-à-dire de se comporter différemment selon les circonstances. En langage C on dispose de ***if ... else*** et ***switch***
  - II. Les structures répétitives (les boucles): permettre de répéter plusieurs fois un bloc d'instructions en utilisant: ***while***, ***do ... while*** et ***for***.

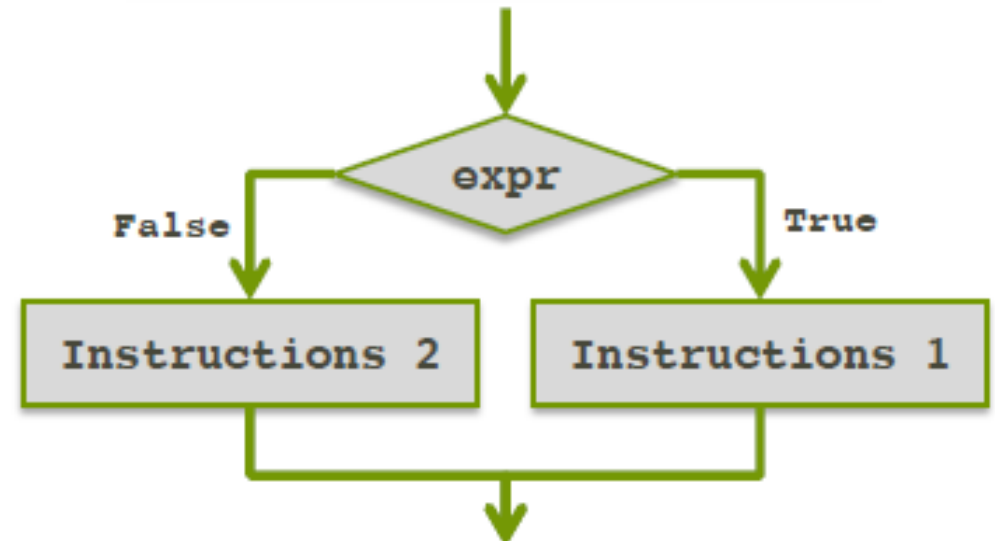
# STRUCTURES DE CONTRÔLE: SÉLECTION SIMPLE

- Utilisée pour l'exécution conditionnelle:
  - Syntaxe:

```
if (expression){  
    Bloc d'instructions  
}
```



```
if (expression) {  
    Bloc d'instructions 1  
}  
else{  
    Bloc d'instructions 2  
}
```



# STRUCTURES DE CONTRÔLE: SÉLECTION SIMPLE

- **bloc-instructions:** peut être une seule instruction terminée par un point-virgule ou une suite d'instructions délimitées par des accolades **{ }**
- **expression:** est évaluée, si elle est vraie (valeur différente de 0), alors *bloc-instructions1* est exécuté. Si elle est fausse (valeur 0) alors *bloc-instructions2* est exécuté

# STRUCTURES DE CONTRÔLE: SÉLECTION SIMPLE

## ○ Exemples:

```
#include <stdio.h>

int main()
{
    int a, b;
    printf("Donnez un entier a:");
    scanf("%d", &a);
    printf("Donnez un entier b:");
    scanf("%d", &b);
    if (a < b)
        printf("%d est inférieur à %d \n", a, b);
    else
        printf("%d est supérieur à %d \n", a, b);

    return 0;
}
```

```
Donnez un entier a: 89
Donnez un entier b: 15
89 est supérieur à 15
```

```
Donner un entier a: 2
Donner un entier b: 7
2 est inférieur à 7
```

# STRUCTURES DE CONTRÔLE: SÉLECTION SIMPLE

## ○ Exemples:

```
#include <stdio.h>

int main()
{
    int a;
    printf("Saisir un entier: ");
    scanf("%d", &a);
    if ((a%2)==0)
        printf("%d est pair", a);
    else
        printf("%d est impair", a);

    return 0;
}
```

```
Saisir un entier: 15
15 est impair
```

```
Saisir un entier: 80
80 est pair
```

# STRUCTURES DE CONTRÔLE: SÉLECTION SIMPLE

## ○ Imbrication des instructions *if*:

```
#include <stdio.h>

int main()
{
    int a, b;
    printf("Donner un entier a: ");
    scanf("%d", &a);
    if(a<=0)
    { if(a==0)
      printf("a est nul ");
      else
        printf("a est strictement négatif ");}
    else
      printf("a est strictement positif ");

    return 0;
}
```

Donner un entier a: -6  
a est strictement négatif

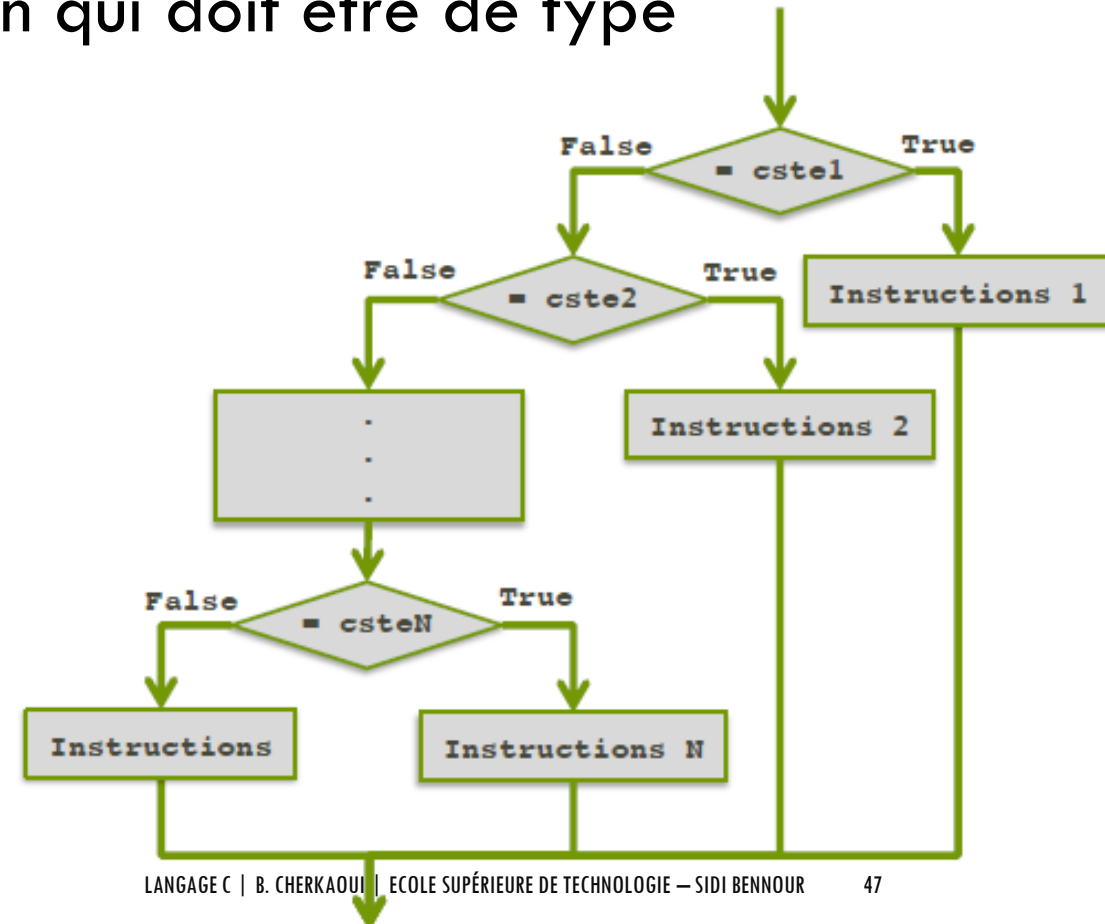
Donner un entier a: 38  
a est strictement positif

# STRUCTURES DE CONTRÔLE: SÉLECTION MULTIPLE

- Sélection multiple switch: Permet de choisir des instructions à exécuter selon la valeur d'une expression qui doit être de type entier

- Syntaxe:

```
switch (expression){  
  case constant1 : {  
    instructions 1  
    break; }  
  case constante2 : {  
    instructions 2  
    break; }  
  ...  
  case constanteN : {  
    instructions N  
    break; }  
  default : {  
    instructions  
    break; }  
}
```



# STRUCTURES DE CONTRÔLE: SÉLECTION MULTIPLE

- **constante *i*** doit être une expression constante entière
- **Instructions *i*** peut être une instruction simple ou composée
- **break** : permet de sortir de de switch
- **break** et **default** sont optionnels et peuvent ne pas figurer
- Si la valeur choisi est égale à une **constante *i***, on se branche à ce cas et on exécute les **instructions *i*** qui lui correspondent
  - On exécute aussi les instructions des cas suivants jusqu'à la fin du bloc ou jusqu'à une instruction break.
- Si la valeur de l'expression n'est égale à aucune des expressions constantes
  - Si **default** existe, alors on exécute les instructions qui le suivent
  - Sinon aucune instruction n'est exécutée



# STRUCTURES DE CONTRÔLE: SÉLECTION MULTIPLE

```
int main()
{
    int a, b, choix;
    printf("Donner un entier a: ");
    scanf("%d", &a);
    printf("Donner un entier b: ");
    scanf("%d", &b);
    printf("Choisir l'opération arithmétique que vous souhaitez\n");
    printf("'1' pour la somme \n'2' pour la multiplication \n'3' pour la soustraction \n");
    printf("Saisissez votre choix: ");
    scanf("%d", &choix);
    switch (choix)
    {
        case 1:
            printf("La somme de %d et %d est: %d", a,b, a+b);
            break;
        case 2:
            printf("Le produit de %d et %d est: %d", a,b, a*b);
            break;
        case 3:
            printf("la différence de %d et %d est: %d", a,b, a-b);
            break;
        default:
            printf("Le choix que vous avez introduit n'est pas répertorié");
    }

    return 0;
}
```

# STRUCTURES DE CONTRÔLE: SÉLECTION MULTIPLE

```
Donner un entier a: 7
Donner un entier b: 1
Choisir l'opération arithmétique que vous souhaitez
'1' pour la somme
'2' pour la multiplication
'3' pour la soustraction
Saisissez votre choix: 1
La somme de 7 et 1 est: 8
```

```
Donner un entier a: 7
Donner un entier b: 1
Choisir l'opération arithmétique que vous souhaitez
'1' pour la somme
'2' pour la multiplication
'3' pour la soustraction
Saisissez votre choix: 2
Le produit de 7 et 1 est: 7
```

```
Donner un entier a: 7
Donner un entier b: 1
Choisir l'opération arithmétique que vous souhaitez
'1' pour la somme
'2' pour la multiplication
'3' pour la soustraction
Saisissez votre choix: 3
la différence de 7 et 1 est: 6
```

```
Donner un entier a: 7
Donner un entier b: 1
Choisir l'opération arithmétique que vous souhaitez
'1' pour la somme
'2' pour la multiplication
'3' pour la soustraction
Saisissez votre choix: 9
Le choix que vous avez introduit n'est pas répertorié
```

# STRUCTURES DE CONTRÔLE: SÉLECTION

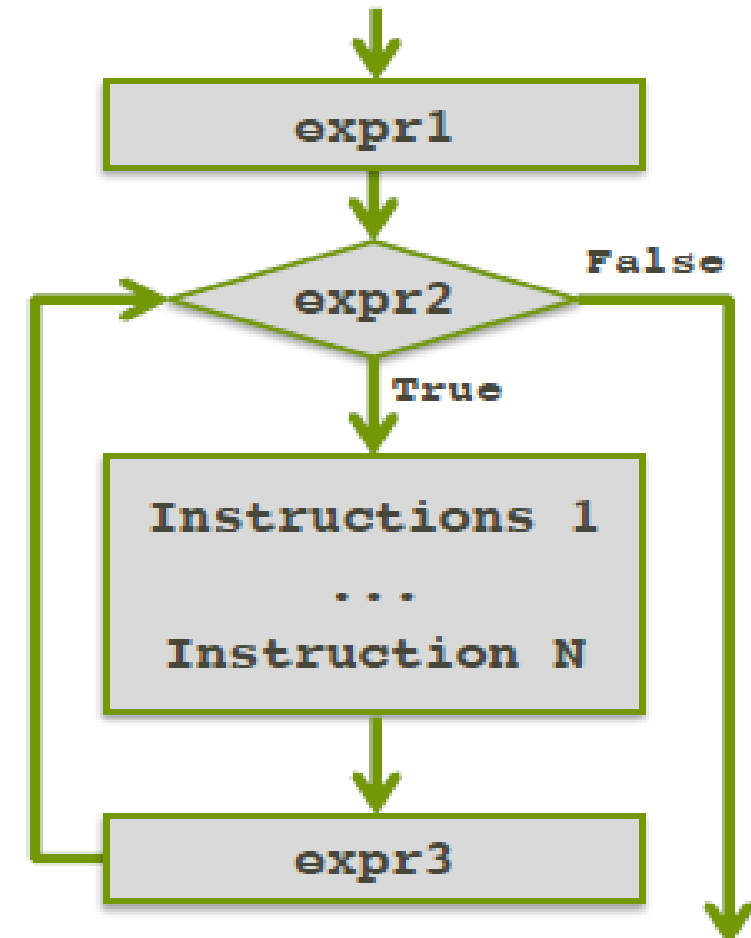
## Exercices d'application:

- 1) Ecrire un programme qui détecte le minimum de trois réels initiés par l'utilisateur.
- 2) Ecrire un programme qui lit un indice destiner pour un mois donné, puis il affiche son nom approprié.
- 3) Ecrire un programme qui lit un caractère et détermine s'il fait partie des alphabets ou non. Et s'il l'est, dire en plus s'il est une minuscule ou une majuscule.

# STRUCTURES DE CONTRÔLE: RÉPÉTITION

- La boucle **for** permet de répéter un bloc d'instructions un nombre prédéfini de fois
- Syntaxe:

```
for (expr1; expr2; expr3){  
    Instruction 1  
    ...  
    Instruction N  
}
```



# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

- *expr1* est évaluée une seule fois au début de l'exécution de la boucle. Elle effectue l'initialisation des données de la boucle.
- *expr2* est évaluée et testée avant chaque passage dans la boucle. Elle constitue le test de continuation de la boucle.
- *expr3* est évaluée après chaque passage. Elle est utilisée pour réinitialiser les données de la boucle

## Exemple:

Ecrire un programme qui lit un nombre entier et affiche son factoriel.

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

```
#include <stdio.h>

int main()
{
    int n,i;
    double fact;
    printf("Saisir un nombre entier pour calculer son factoriel: ");
    scanf("%d",&n);
    fact = 1;
    for(i=1;i<=n;i++)
    {
        fact = fact*i;
    }
    printf("le factoriel de %d est: %.0f", n, fact);
}
```

```
Saisir un nombre entier pour calculer son factoriel: 6
le factoriel de 6 est: 720
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i;

    for(i=0; i<5; i++)
        printf("Salut les GI \n");
    return 0;
}
```

```
Salut les GI
Salut les GI
Salut les GI
Salut les GI
Salut les GI
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i;

    for(i=0;i<5;i++)
        printf("Voici la ligne %d \n", i+1);
    return 0;
}
```

```
Voici la ligne 1
Voici la ligne 2
Voici la ligne 3
Voici la ligne 4
Voici la ligne 5
```



# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

```
int main()
{

    int i,j;

    for(i=1;i<=2;i++)
    {printf("Voici la liste %d \n", i);
      for(j=1;j<=3;j++)
      {printf("\tVoici l'élément %d de la liste %d \n", j,i);
      }
    }
    return 0;
}
```

```
Voici la liste 1
    Voici l'élément 1 de la liste 1
    Voici l'élément 2 de la liste 1
    Voici l'élément 3 de la liste 1
Voici la liste 2
    Voici l'élément 1 de la liste 2
    Voici l'élément 2 de la liste 2
    Voici l'élément 3 de la liste 2
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{

    int i, j, liste, element;
    printf("Entrez le nombre de liste que vous voulez initier: ");
    scanf("%d", &liste);
    printf("Entrez le nombre d'élément par liste: ");
    scanf("%d", &element);
    for(i=1;i<=liste;i++)
    {printf("Voici la liste %d \n", i);
      for(j=1;j<=element;j++)
      printf("\tVoici l'élément %d de la liste %d \n", j,i);
    }
    return 0;
}
```

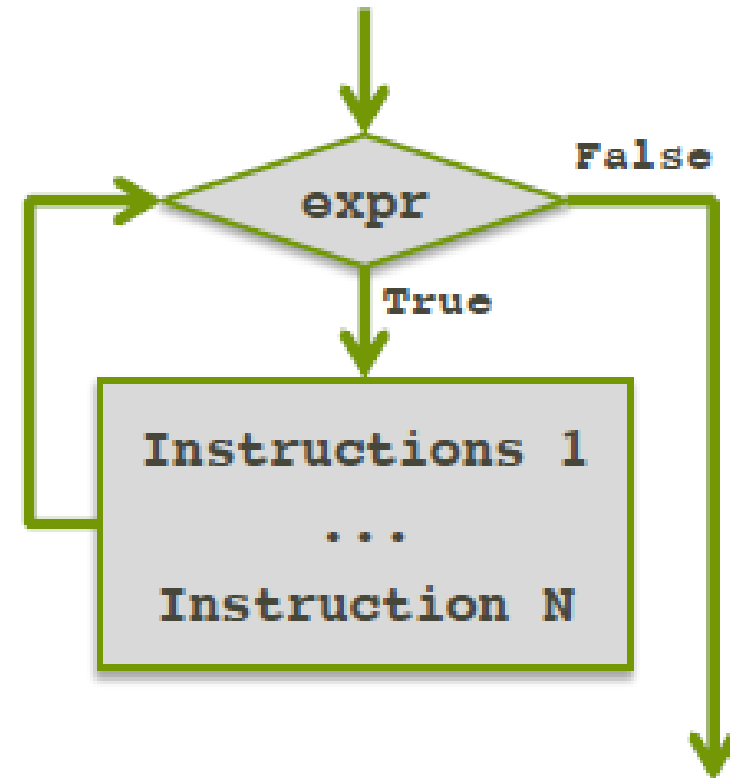
# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

```
Entrez le nombre de liste que vous voulez initier: 3
Entrez le nombre d'élément par liste: 2
Voici la liste 1
    Voici l'élément 1 de la liste 1
    Voici l'élément 2 de la liste 1
Voici la liste 2
    Voici l'élément 1 de la liste 2
    Voici l'élément 2 de la liste 2
Voici la liste 3
    Voici l'élément 1 de la liste 3
    Voici l'élément 2 de la liste 3
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

- La boucle **while** permet de répéter un bloc d'instructions tant qu'une condition est vraie. On teste puis on exécute:
- Syntaxe:

```
while (expression){  
    Instruction 1  
    ...  
    Instruction N  
}
```



# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

- la condition (condition de contrôle de la boucle) est évaluée à chaque itération.
- Les instructions (corps de la boucle) sont exécutés tant que la condition est vraie, on sort de la boucle dès que la condition devient fausse.
- Le test de continuation s'effectue avant d'intégrer le corps de la boucle qui, de ce fait, peut ne jamais s'exécuter.

## Exercice d'application:

Ecrire un programme qui lit un nombre entier et affiche son factoriel en utilisant la boucle while.

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

```
int main()
{
    int n,i;
    double fact;
    printf("Saisir un nombre entier pour calculer son factoriel: ");
    scanf("%d",&n);
    fact = 1;
    i = 1;
    while(i<=n)
    {
        fact = fact * i;
        i++;
    }
    printf("le factoriel de %d est: %.0f", n, fact);
    return 0;
}
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i, j;

    i=1;
    while(i<=5)
    {
        printf("Salut les GI \n");
        i++;
    }
    return 0;
}
```

```
Salut les GI
Salut les GI
Salut les GI
Salut les GI
Salut les GI
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i, j;

    i=1;
    while(i<=5)
    {
        printf("Voici la ligne %d \n", i);
        i++;
    }
    return 0;
}
```

```
Voici la ligne 1
Voici la ligne 2
Voici la ligne 3
Voici la ligne 4
Voici la ligne 5
```



# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i, j;
    i=1;

    while(i<=3)
    {
        printf("Voici la liste %d: \n", i);
        j=1;
        while(j<=2)
        {
            printf("\tVoici l'élément %d de la liste %d \n", j,i);
            j++;
        }
        i++;
    }
    return 0;
}
```

```
Voici la liste 1:
    Voici l'élément 1 de la liste 1
    Voici l'élément 2 de la liste 1
Voici la liste 2:
    Voici l'élément 1 de la liste 2
    Voici l'élément 2 de la liste 2
Voici la liste 3:
    Voici l'élément 1 de la liste 3
    Voici l'élément 2 de la liste 3
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i, j, liste, element;
    printf("Entrez le nombre de liste que vous voulez initier: ");
    scanf("%d", &liste);
    printf("Entrez le nombre d'élément par liste: ");
    scanf("%d", &element);
    i=1;
    while(i<=liste)
    {
        printf("Voici la liste %d: \n", i);
        j=1;
        while(j<=element)
        {
            printf("\tVoici l'élément %d de la liste %d \n", j,i);
            j++;
        }
        i++;
    }
    return 0;
}
```

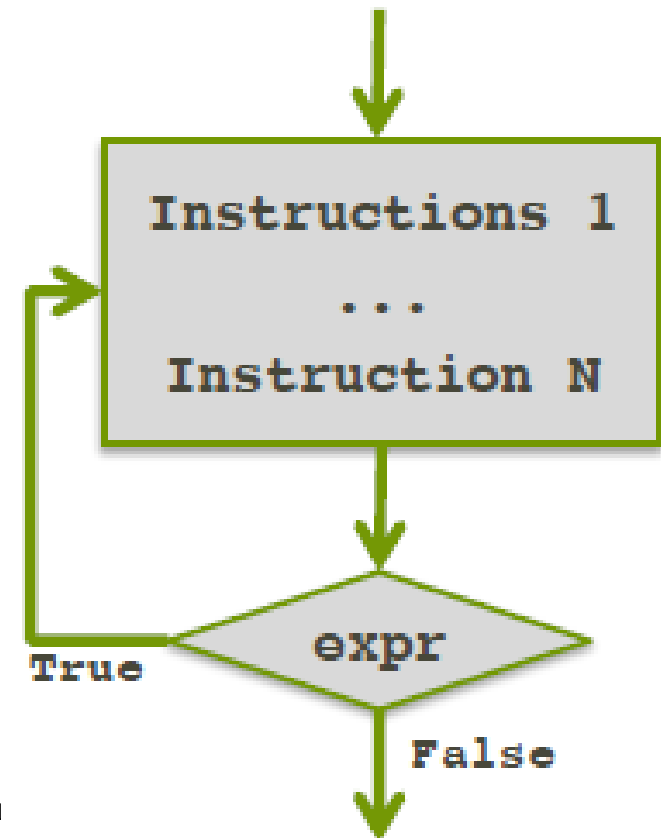
```
Entrez le nombre de liste que vous voulez initier: 2
Entrez le nombre d'élément par liste: 5
Voici la liste 1:
    Voici l'élément 1 de la liste 1
    Voici l'élément 2 de la liste 1
    Voici l'élément 3 de la liste 1
    Voici l'élément 4 de la liste 1
    Voici l'élément 5 de la liste 1
Voici la liste 2:
    Voici l'élément 1 de la liste 2
    Voici l'élément 2 de la liste 2
    Voici l'élément 3 de la liste 2
    Voici l'élément 4 de la liste 2
    Voici l'élément 5 de la liste 2
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

- La boucle **do...while** permet de répéter un bloc d'instructions tant qu'une condition est vraie, contrairement à la boucle **while**, on exécute puis on teste:

- Syntaxe:

```
do {  
    Instruction 1  
    ...  
    Instruction N  
}while (expression);
```



# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

- La boucle `do...while`, le test est effectué après le corps de boucle, ce dernier sera alors exécuté au moins une fois.

## Exercice d'application:

Ecrire un programme qui lit un nombre entier et affiche son factoriel en utilisant la boucle `do...while`.

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

```
int main()
{
    int i;
    i=1;
    do {
        printf("Salut les GI \n");
        i++;
    }while(i<=5);

    return 0;
}
```

```
Salut les GI
Salut les GI
Salut les GI
Salut les GI
Salut les GI
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i;
    i=1;
    do {
        printf("Voici la ligne %d \n", i);
        i++;
    }while(i<=5);

    return 0;
}
```

```
Voici la ligne 1
Voici la ligne 2
Voici la ligne 3
Voici la ligne 4
Voici la ligne 5
```

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i, j;
    i=1;
    do {
        printf("Voici la liste %d: \n", i);
        j=1;
        do {
            printf("\tVoici l'élément %d de la liste %d \n", j,i);
            j++;
        }while(j<=3);
        i++;
    }while(i<=2);

    return 0;
}
```

Voici la liste 1:

Voici l'élément 1 de la liste 1

Voici l'élément 2 de la liste 1

Voici l'élément 3 de la liste 1

Voici la liste 2:

Voici l'élément 1 de la liste 2

Voici l'élément 2 de la liste 2

Voici l'élément 3 de la liste 2

# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

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

int main()
{
    int i, j, liste, element;
    printf("Entrez le nombre de liste que vous voulez initier: ");
    scanf("%d", &liste);
    printf("Entrez le nombre d'élément par liste: ");
    scanf("%d", &element);
    i=1;
    do {
        printf("Voici la liste %d: \n", i);
        j=1;
        do {
            printf("\tVoici l'élément %d de la liste %d \n", j,i);
            j++;
        }while(j<=element);
        i++;
    }while(i<=liste);

    return 0;
}
```

```
Entrez le nombre de liste que vous voulez initier: 2
Entrez le nombre d'élément par liste: 5
Voici la liste 1:
    Voici l'élément 1 de la liste 1
    Voici l'élément 2 de la liste 1
    Voici l'élément 3 de la liste 1
    Voici l'élément 4 de la liste 1
    Voici l'élément 5 de la liste 1
Voici la liste 2:
    Voici l'élément 1 de la liste 2
    Voici l'élément 2 de la liste 2
    Voici l'élément 3 de la liste 2
    Voici l'élément 4 de la liste 2
    Voici l'élément 5 de la liste 2
```



# STRUCTURES DE CONTRÔLE: LA RÉPÉTITION

```
int main()
{
    int n,i;
    double fact;
    printf("Saisir un nombre entier pour calculer son factoriel: ");
    scanf("%d",&n);
    fact = 1;
    i = 1;
    do
    {
        fact = fact * i;
        i++;
    }
    while(i<=n);

    printf("le factoriel de %d est: %.0f", n, fact);
    return 0;
}
```

# STRUCTURES DE CONTRÔLE: RÉPÉTITION

## Exercices d'application:

- 1) Ecrire un programme qui affiche la table de multiplication d'un entier.
- 2) Ecrire un programme qui affiche tous les nombres entiers positifs inférieurs à un entier donné  $n$  saisi par l'utilisateur.
- 3) Ecrire un programme qui calcule la suite suivante:

$$U_0 = 1$$

$$U_n = 2n + 1$$

# STRUCTURES DE CONTRÔLE: RÉPÉTITION

## Solution Exercise1:

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

int main()
{
    int i,n;
    printf("Entrez un nombre entier pour afficher sa table de multiplication: ");
    scanf("%d",&n);
    printf("La table de multiplication de %d est: \n", n);
    for(i=1;i<=10;i++)
    {
        printf("\t %d * %d = %d \n", n, i, n*i);
    }
    return 0;
}
```

# STRUCTURES DE CONTRÔLE: RÉPÉTITION

## Solution Exercise 2:

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

int main()
{
    unsigned int i,n;
    printf("Entrez un nombre entier positif: ");
    scanf("%d",&n);

    i=n;
    while(i>0)
    {
        printf("%d \n",i);
        i--;
    }
    return 0;
}
```

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

int main()
{
    unsigned int i,n;
    printf("Entrez un nombre entier positif: ");
    scanf("%d",&n);

    for(i=n;i>0;i--)
        printf("%d \n",i);

    return 0;
}
```

# STRUCTURES DE CONTRÔLE: RÉPÉTITION

## Solution Exercise 3:

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

int main()
{
    int i,n;
    printf("Ce programme permet de calculer la suite  $U_n = 2n + 1$  avec  $U_0 = 1$  \n");
    printf("Entrez n: ");
    scanf("%d",&n);
    printf("\tU0 = 1 \n");
    for(i=1;i<=n;i++)
        printf("\tU%d = %d \n",i,(2*i+1));
    return 0;
}
```

# STRUCTURES DE CONTRÔLE: RÉPÉTITION

## Solution Exercise 3:

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

int main()
{
    int i,n;
    printf("Ce programme permet de calculer la suite  $U_n = 2n + 1$  avec  $U_0 = 1$  \n");
    printf("Entrez n: ");
    scanf("%d",&n);

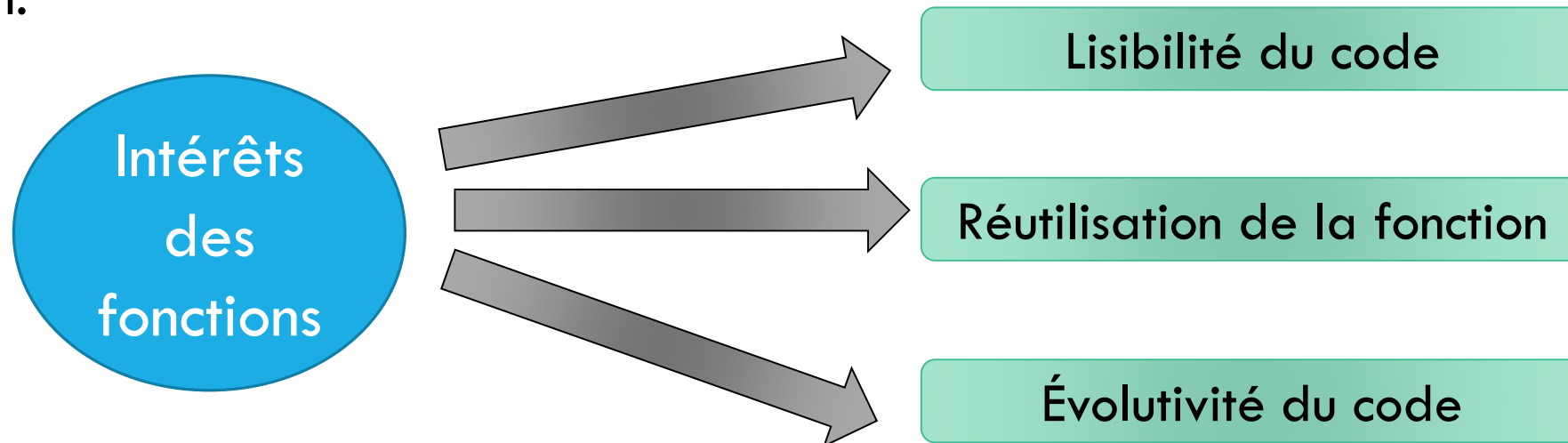
    i=0;
    while(i<=n)
    {
        printf("\tU%d = %d \n",i,(2*i+1));
        i++;
    }
    return 0;
}
```



# LES FONCTIONS EN C

# LES FONCTIONS

- Une fonction est un sous-programme qui permet d'effectuer un ensemble d'instructions par simple appel dans le corps du programme principal.
- Le besoin d'intégrer des fonctions dans un programme devient primordiale dès qu'un groupe de lignes revient plusieurs fois on les regroupe dans une fonction.



Remarque : une fonction peut faire appel à d'autres fonctions



# LES FONCTIONS

- Il existe **deux** types de fonctions:
  - Des fonctions qui s'exécutent sans retourner de valeurs, nommées procédures dans certains langages. Se sont des fonctions de type **void**

```
#include <stdio.h>

void affiche_bonjour()
{
    printf("Hello World de la fonction \n");
}

int main()
{
    printf("Hello World du programme \n");
    affiche_bonjour();

    return 0;
}
```

```
Hello World du programme
Hello World de la fonction
```

# LES FONCTIONS

- Des fonctions qui s'exécutent et retournent une valeur.

```
#include <stdio.h>

int addition(int a, int b)
{
    return a+b;
}

int main()
{
    int x,y,somme;
    printf("Saisir deux entier:");
    scanf("%d%d",&x,&y);
    somme = addition(x,y);
    printf("La somme de %d et %d est: %d", x,y, somme);
    return 0;
}
```

```
Saisir deux entier: 15 6
La somme de 15 et 6 est: 21
```

# LES FONCTIONS

○ Dans un programme, on rencontre le nom des fonctions dans 3 cas:

1. Déclaration : le type de la fonction et de ses arguments

➡ 1 seule fois

2. Définition : codage de la fonction

➡ 1 seule fois

3. Appels (= utilisations) de la fonction

➡ n fois

# LES FONCTIONS

- Déclaration d'une fonction:

**Type\_fonction** **Nom\_De\_La\_Fonction**(type argument1, type argument2, ...);

- Définition d'une fonction:

**Type\_fonction** **Nom\_De\_La\_Fonction**(type argument1, type argument2, ...)

{

instructions;

return (valeur-de-la-fonction);

}

# LES FONCTIONS

- En C, une fonction ne peut retourner qu'une valeur (**au plus**) grâce à la commande **return**
- Le type de la fonction doit être le même que celui de la valeur retournée.
- Le programme appelant(soit une autre fonction ou le programme principale) doit stocker ce résultat dans une variable de même type (ou bien ne rien stocker).
- Quand une fonction ne retourne pas de valeur elle est typée **void**

# LES FONCTIONS: LA PORTÉE DES VARIABLES

- Une variable créée dans un bloc **{ }** est appelée **locale** :
  - C'est une variable qui n'existe qu'au sein du bloc
  - Elle ne sera pas connue en dehors de ce bloc
  - Sa valeur est perdue à la sortie du bloc, alors sa durée de vie est celle du **bloc**
- Une variable globale existe en dehors de tout bloc
  - Elle a sa mémoire réservée pour toute l'exécution du programme
  - Sa durée de vie est celle du **programme**

Conseil: Soyez le plus local possible

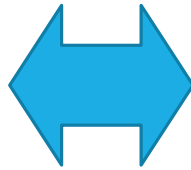
# LES FONCTIONS: EXEMPLE 1

```
#include <stdio.h>

int maximum(int n, int m)
{
    if (n>m)
        return n;
    else
        return m;
}

int main()
{
    int a, max, b;

    printf("Saisir deux entiers: ");
    scanf("%d%d", &a, &b);
    max=maximum(a,b);
    printf("Le maximum est: %d \n", max);
    return 0;
}
```



```
#include <stdio.h>

int main()
{
    int a, max, b;

    printf("Saisir deux entiers: ");
    scanf("%d%d", &a, &b);
    if(a<b)
        max=b;
    else
        max=a;
    printf("Le maximum est: %d \n", max);
    return 0;
}
```

```
#include <stdio.h>

int maximum(int n, int m)
{
    if (n>m)
        return n;
    else
        return m;
}

int main()
{
    int a, b;

    printf("Saisir deux entiers: ");
    scanf("%d%d", &a, &b);

    printf("Le maximum est:%d", maximum(a,b))
    return 0;
}
```

# LES FONCTIONS: EXEMPLE 2

```
#include <stdio.h>

void affichageMenu()
{
    printf("---Menu---\n\n");
    printf("1: Addition\n");
    printf("2: Multiplication\n");
    printf("3: Soustraction\n");
    printf("4: Divition\n");
    printf("\nVotre choix?: ");
}

int saisi()
{
    int x;
    printf("Saisir un entier: ");
    scanf("%d",&x);
    return x;
}
```

```
int main ()
{
    int choix, a, b;
    affichageMenu ();
    scanf ("%d", &choix);
    while (choix < 1 || choix > 4)
    {
        printf ("Choix inexistant!! Faites un choix parmi le menu suivant: \n");
        affichageMenu ();
        scanf ("%d", &choix);
    }

    switch (choix)
    {
        case 1:
            a = saisi ();
            b = saisi ();
            printf ("La somme de %d et %d est: %d", a, b, a + b);
            break;
        case 2:
            a = saisi ();
            b = saisi ();
            printf ("Le produit de %d et %d est: %d", a, b, a * b);
            break;
        case 3:
            a = saisi ();
            b = saisi ();
            printf ("La diffC)rence de %d et %d est: %d", a, b, a - b);
            break;
        case 4:
            a = saisi ();
            b = saisi ();
            if (b != 0)
                printf ("La division de %d sur %d est: %d", a, b, a / b);
            else
                printf ("Division impossible! \n");
            break;
    }
    return 0;
}
```



# LES FONCTIONS: EXEMPLE 2

```
---Menu---  
1: Addition  
2: Multiplication  
3: Soustraction  
4: Divition  
  
Votre choix?: 7  
Choix inexistant!! Faites un choix parmi le menu suivant:  
---Menu---  
1: Addition  
2: Multiplication  
3: Soustraction  
4: Divition  
  
Votre choix?:
```

```
Votre choix?: 7  
Choix inexistant!! Faites un choix parmi le menu suivant:  
---Menu---  
1: Addition  
2: Multiplication  
3: Soustraction  
4: Divition  
  
Votre choix?: 2  
Saisir un entier: 12  
Saisir un entier: 5  
Le produit de 12 et 5 est: 60
```



# LES TABLEAUX EN C

# LES TABLEAUX:

- Un Tableau est un ensemble *fini* d'éléments de *même type* stockés en mémoire d'une manière *contiguë* (Les éléments sont successifs en mémoire, l'un après l'autre).
- Le type des éléments, du tableau, peut être :
  - Simple: int, float, double, char...
  - Pointeur ou Structure.
- On peut définir des tableaux :
  - à une dimension (tableau unidimensionnel ou vecteur)
  - à plusieurs dimensions (tableau multidimensionnel)

# LES TABLEAUX:

- Déclaration d'un tableau:

**type** Nom\_Tableau[**taille**];

- Déclaration et initialisation:

**type** Nom\_Tableau[**taille**] = {val1, val2, val3,...,valn};

N.B.: La taille est le nombre de **cases réservés** en mémoire.

- Exemples:

- `int T[5] = {15, 6, 85, 0, -6};`
- `float X[10] = {1.2, 7.1, 9.6, 0.4, 10, 7.6};`

array				
0	1	2	3	4
int	int	int	int	int
15	6	85	0	-6

X	array									
	0	1	2	3	4	5	6	7	8	9
	float	float	float	float	float	float	float	float	float	float
	1.2	7.1	9.6	0.4	10	7.6	0	0	0	0

# LES TABLEAUX:

- Les cases d'un tableau sont indexées de **0** à *taille - 1*
- L'accès à un élément se fait à l'aide de son indice.

`Nom_tableau[indice]`

- Exemple:

`int T[5] = {4, 11, -1};`

`T[4] = 20;`

`T[3] = T[1] * 2;`

T	array				
	0	1	2	3	4
	int	int	int	int	int
	4	11	-1	0	0
T	array				
	0	1	2	3	4
	int	int	int	int	int
	4	11	-1	0	20
T	array				
	0	1	2	3	4
	int	int	int	int	int
	4	11	-1	22	20

# LES TABLEAUX:

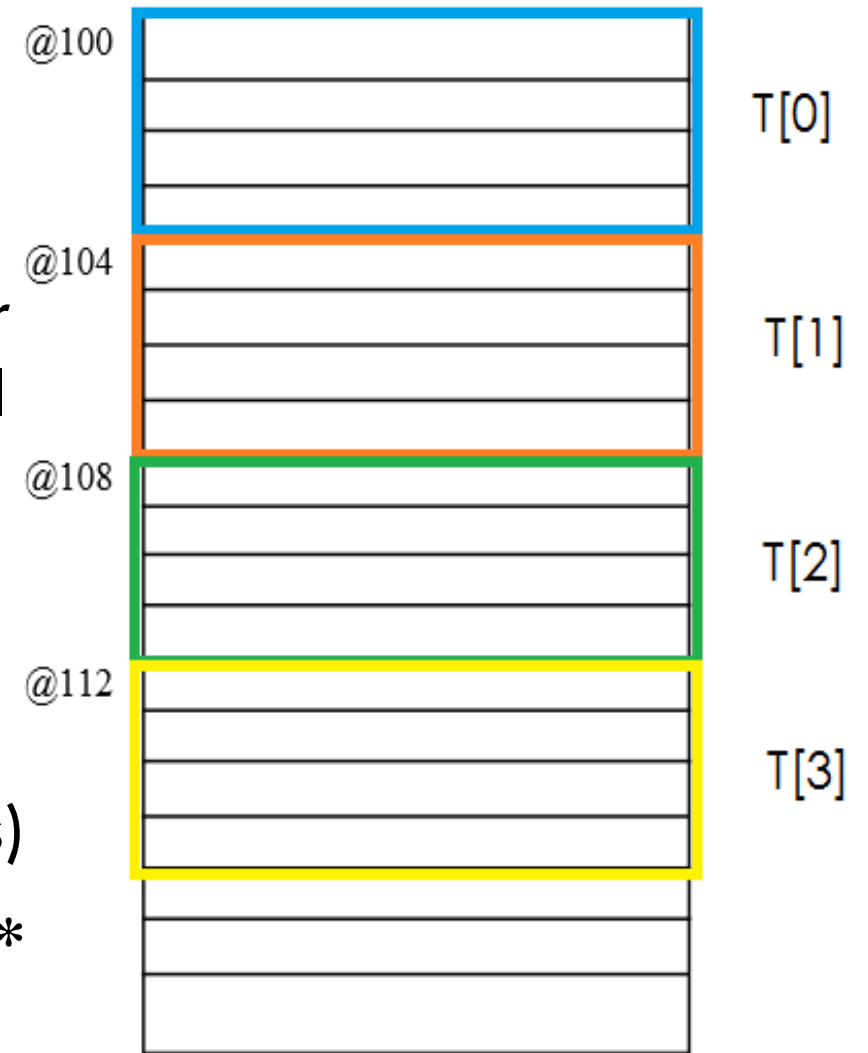
- La déclaration d'un tableau permet de lui réserver un espace mémoire dont la taille (en octets) est égal à :

Taille du tableau \* Taille du type

- Exemple:

`int T[4];` // on réserve 16 octets (4 \* 4 octets)

`char prenom[10];` // on réserve 10 octets (10 \* 1 octet)



# LES TABLEAUX:

- Affichage du premier élément d'un tableau d'entiers T de taille n:

```
printf("Voici le premier élément du tableau: %d", T[0]);
```

- Affichage de tous les éléments d'un tableau d'entiers T de taille n:

```
for(i=0; i<n;i++)  
    printf(" %d \n", T[i]);
```

- Saisie des éléments de tous les éléments d'un tableau d'entiers T de taille n:

```
for(i=0; i<n;i++)  
    scanf(" %d \n", &T[i]);
```

# LES TABLEAUX:

## ○ Exemple:

```
Entrez la valeur de l'élément 1 du tableau: 12
Entrez la valeur de l'élément 2 du tableau: 7
Entrez la valeur de l'élément 3 du tableau: -9
Entrez la valeur de l'élément 4 du tableau: 5
La somme des éléments du tableau est: 15
```

```
#include <stdio.h>

int main()
{
    int T[4];
    int i, somme;
    // Saisie des éléments du tableau
    for(i=0;i<4;i++)
    {
        printf("Entrez la valeur de l'élément %d du tableau: ", i+1);
        scanf("%d", &T[i]);
    }

    // Calcul de la somme des éléments du tableau
    somme = 0;
    for(i=0;i<4;i++)
        somme = somme + T[i];

    printf("La somme des éléments du tableau est: %d", somme);
    return 0;
}
```



# LES TABLEAUX: TABLEAUX MULTIDIMENSIONNELS

- Un tableau multidimensionnel se définit de la manière suivante :

`type Nom_du_tableau[Taille1][Taille2][Taille3] ... [TailleN];`

où `Taille_i` est le nombre d'éléments dans la dimension i.

- Exemple 1: La déclaration d'un tableau d'entiers tridimensionnel (5 x 10 x 4):

```
Int Tab[5][10][4];
```

- Exemple 2: pour stocker les notes de 25 étudiants en 5 modules dans deux examens, on peut déclarer un tableau :

```
Float notes[25][5][2];
```

(notes[i][j][k] est la note de l'examen k dans le module j pour l'étudiant i)

# LES TABLEAUX: TABLEAUX À DEUX DIMENSIONS (MATRICES)

- Déclaration : `Type Nom_du_Tableau[nombre ligne][nombre colonne];`
- Exemple: `int Tab[3][4];`

Tab[0][0]	Tab[0][1]	Tab[0][2]	Tab[0][3]
Tab[1][0]	Tab[1][1]	Tab[1][2]	Tab[1][3]
Tab[2][0]	Tab[2][1]	Tab[2][2]	Tab[2][3]

- Déclaration et Initialisation:

```
int Tab[3][4] = {{15, -6, 78, 1}, {70, 10, 96, 30}, {-75, 22, 98, 44}};
```

# LES TABLEAUX: TABLEAUX À DEUX DIMENSIONS (MATRICES)

- Saisie et affichage des éléments d'un tableau à deux dimensions:

```
int Tab[n][m];
int i,j;
for (i=0;i<n;i++)
    { for (j=0;j<m;j++)
        printf("Saisir la valeur de l'élément Tab[%d][%d]: ", i,j);
        scanf("%d", &Tab[i][j]);
    }
```

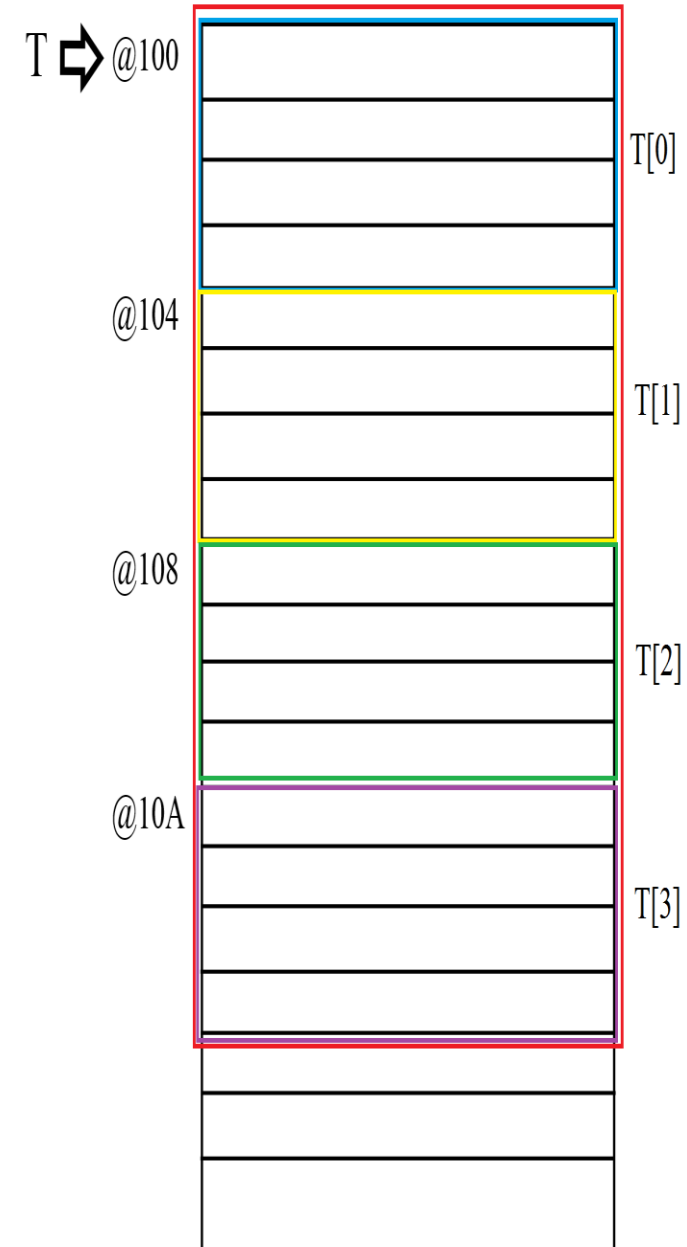
```
for (i=0;i<n;i++)
    { for (j=0;j<m;j++)
        printf("Tab[%d][%d] = %d", i, j, Tab[i][j]);
    }
```

# LES TABLEAUX: REPRÉSENTATION D'UN TABLEAU EN MÉMOIRE

- En C, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau (pour un tableau T:  $T = \&T[0]$  )
- Les composantes du tableau étant stockées en mémoire à des emplacements contigus, les adresses des autres composantes sont calculées (automatiquement) relativement à cette adresse :

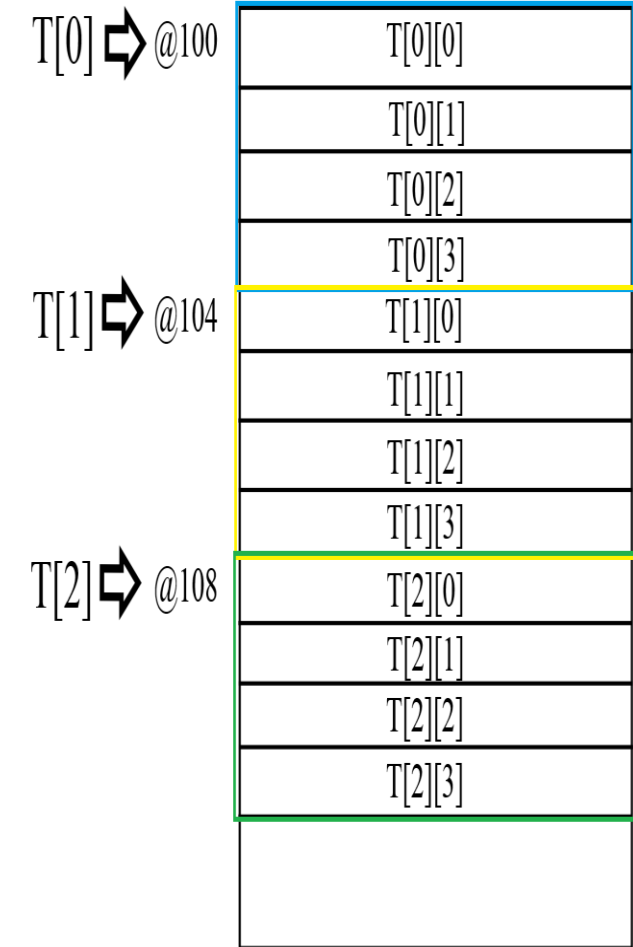
$$\&T[i] = \&T[0] + \text{sizeof}(\text{type}) * i;$$

La fonction « sizeof » renvoie le nombre d'octets nécessaire pour coder un type dans la mémoire.



# LES TABLEAUX: REPRÉSENTATION D'UN TABLEAU EN MÉMOIRE

- Les éléments d'un tableau sont stockés en mémoire à des emplacements contigus ligne après ligne
- Comme pour les tableaux unidimensionnels, le nom d'un tableau  $A$  à deux dimensions est le représentant de l'adresse du premier élément :  $T = \&T[0][0]$
- Rappelons qu'une matrice  $A[n][m]$  est à interpréter comme un tableau de dimension  $n$  dont chaque composante  $T[i]$  est un tableau de dimension  $m$ .
- Exemple : `char T[3][4]; T=&T[0][0] = 100`



# LES TABLEAUX:

## Exercices:

1. Ecrire un programme en C qui calcule le nombre d'entiers pairs et le nombre d'entiers impairs d'un tableau d'entiers de taille 20.
2. Ecrire un programme qui permet de lire et stocker 15 valeurs réels et de déterminer le nombre de celles qui sont supérieures à la moyenne.
3. Soit T un tableau de N réels. Ecrire le programme qui permet de calculer le nombre des occurrences d'un nombre X (c'est-à-dire combien de fois ce nombre X figure dans le tableau T).
4. Soit deux matrices  $M[4][3]$  et  $N[4][3]$ :
  - a) Ecrire un programme qui calcule la somme des deux matrices puis affiche le résultat.
  - b) Ecrire un programme qui calcule et affiche l'inverse de la matrice  $M[3][3]$ .



# LES CHAINES DE CARACTÈRES

# LES CHAINES DE CARACTÈRES:

- Une **chaîne de caractères** (string) est un tableau à une dimension d'éléments de type **char** qui se termine par le caractère de terminaison **'\0'**
- Syntaxe de déclaration:

**char Nom\_Chaine[taille];**

- Déclaration et initialisation:

**char Nom\_Chaine[taille] = {C1,C2,...,Cn,'\0'};**

**char Nom\_Chaine[taille] = "Une chaîne";**

**char Nom\_Chaine[] = "Une chaîne";**

N.B.: pour un  
texte de **n**  
caractères, nous  
devons prévoir  
**n+1** octets



# LES CHAINES DE CARACTÈRES:

- Exemples:

```
char S[10] = "Bonjour";
```

```
char Ch[] = "Bonjour";
```

S	B	o	n	j	o	u	r	'\0'		
Ch	B	o	n	j	o	u	r	'\0'		

- Les éléments d'une chaîne sont indexés de **0** à **taille - 1**

- L'accès à un char se fait à l'aide de son indice:

**nom\_chaine[indice]**

- Pour afficher une chaîne de caractère à l'aide de la fonction **printf** en utilisant spécificateur de format **%s**

# LES CHAINES DE CARACTÈRES:

- Exemple:

```
#include <stdio.h>

int main()
{
    char ch[] = "hello world";
    printf("Message avant modification: %s\n", ch);
    ch[0] = 'H';
    ch[6] = 'W';
    printf("Message après modification: %s\n", ch);

    return 0;
}
```

```
Message avant modification: hello world
Message après modification: Hello World
```

# LES CHAINES DE CARACTÈRES:

- La fonction `scanf` permet de lire une chaîne de caractère en utilisant le spécificateur de format `%s`

Problème: espace, tabulation et retour à la ligne(`\n`) sont des séparateurs

```
#include <stdio.h>

int main()
{
    char Nom[40];
    printf("Votre Nom?: ");
    scanf("%s", Nom);

    printf("Votre nom est: %s\n", Nom);

    return 0;
}
```



# LES CHAINES DE CARACTÈRES:

- La fonction **gets(nom\_chaine)** (get string):
  - Permet de lire au clavier une chaînes de caractères qui se termine par un retour à la ligne(\n) et de l'affecter à la chaîne en question.
  - Exemple:

```
char Nom[40];  
gets(Nom);
```

- La fonction **printf** permet d'afficher une chaîne en utilisant le format **%s**
- La fonction **puts(nom\_chaine)** permet d'afficher sur l'écran une chaîne de caractère avec un retour à la ligne(\n).
  - Exemple:

```
char Nom[40] = "Amine Allali";  
puts(Nom);
```



```
printf("%s\n", Nom);
```

# LES CHAINES DE CARACTÈRES:

- La fonction `getchar()` permet de lire un caractère au clavier
- la fonction `putchar(C)` permet d'afficher le caractère `C` sur l'écran.
- Exemple:

```
char x;
```

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

```
printf("%c", x);
```



```
x = getchar();
```

```
putchar(x);
```

# LES CHAINES DE CARACTÈRES:

- La bibliothèque **string.h** contient un ensemble de fonctions prédéfinies sur les chaines de caractères:
  - strlen : retourne la longueur d'une chaine
  - strchr : Pour chercher un caractère dans une chaine
  - strstr : Pour chercher une sous chaine dans une chaine
  - strcmp : Pour comparer deux chaines de caractères
  - strcpy : Pour copier une chaine dans une autre
  - strcat : Pour concaténer deux chaines
  - toupper: Pour convertir une chaine en majuscule
  - tolower : pour convertir une chaine en miniscule
  - ...

# LES CHAINES DE CARACTÈRES:

- Exercices d'application: (sans utiliser les fonctions de la bibliothèque `string.h`)
  1. Ecrire un programme qui permet de lire une chaîne de caractère (qui ne dépasse pas 20 caractères) puis affiche sa longueur, le nombre d'occurrence du caractère 'a'.
  2. Ecrire un programme qui permet de lire une chaîne de caractère, puis il l'affiche d'une manière inversée.
  3. Ecrire un programme qui permet de lire deux chaînes de caractères et détermine s'elles sont identiques ou non.

```

#include <stdio.h>

int main()
{
    char S[20];
    int i, L = 0;
    printf("Saisir une chaine de caractères: ");
    gets(S);

    for(i=0; i<=19; i++)
    {
        if(S[i]!='\0')
            L++;
        else
            break;
    }
    printf("Voici votre chaine inversée: ");
    for(i=L; i>=0; i--)
    {
        printf("%c", S[i]);
    }
    printf("\n");

    return 0;
}

```

```

#include <stdio.h>
int main()
{
    char S1[20];
    char S2[15];
    int test = 1, i;

    printf("Saisir la première chaine de caractères: ");
    gets(S1);
    printf("Saisir la deuxième chaine de caractères: ");
    gets(S2);

    if(strlen(S1)!=strlen(S2))
        printf("Les chaines ne sont pas identiques \n");
    else
    {
        for(i=0; i<=strlen(S1)-1; i++)
        {
            if(S1[i]!=S2[i])
            {
                test = 0;
                break;
            }
        }
    }
    if(test == 1)
        printf("les chaines sont identiques \n");
    else
        printf("Les chaines ne sont pas identiques \n");
    return 0;
}

```

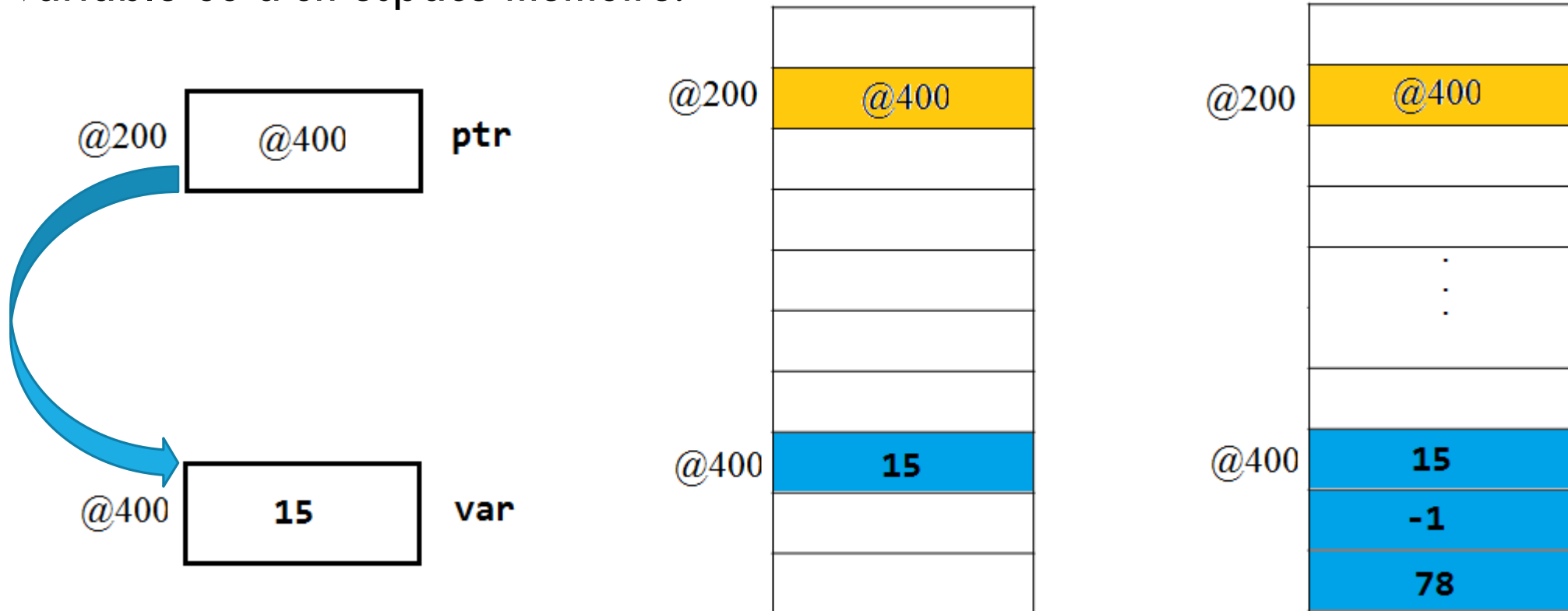




# LES POINTEURS

# LES POINTEURS:

- Un **pointeur** est une **variable** destinée à contenir l'adresse mémoire d'une autre variable ou d'un espace mémoire.



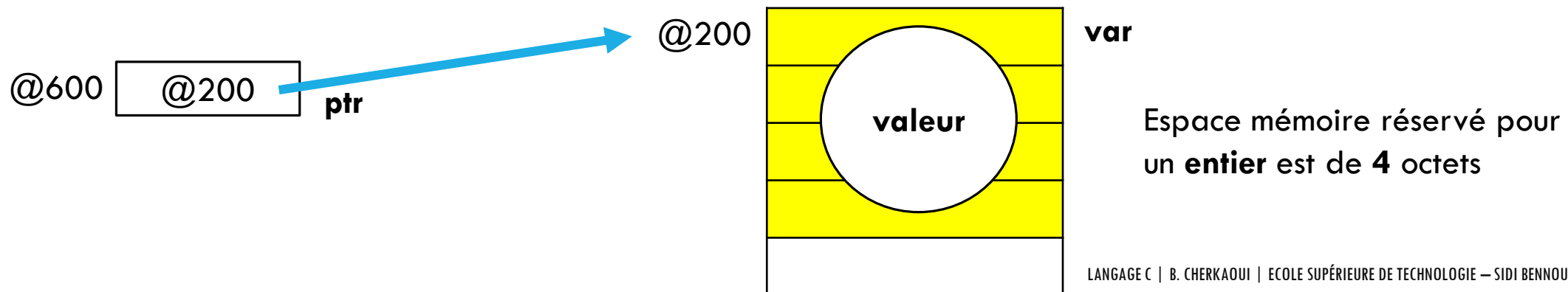
# LES POINTEURS:

**N.B.:** la valeur d'un pointeur donne l'adresse du premier octet parmi les n octets où la variable est stockée

- Déclaration d'un pointeur:

**Type** \***Nom\_pointeur**;

- **Type** est le type de l'espace mémoire pointé.
- **Nom\_pointeur** est l'identificateur du pointeur(la variable pointeur).
- **\*** est l'opérateur qui indiquera au compilateur que c'est un pointeur.



# LES POINTEURS:

## ○ Remarques:

- Le nom d'une variable permet d'accéder directement à sa valeur (adressage direct).
- Un pointeur qui contient l'adresse de la variable, permet d'accéder indirectement à sa valeur (adressage indirect).
- Le nom d'une variable est lié à la même adresse, alors qu'un pointeur peut pointer sur différentes adresses au sein du même programme.

## ○ Intérêts des pointeurs:

- Ils sont indispensables pour permettre le passage par référence pour les paramètres des fonctions.
- Ils permettent de créer des structures de données (listes et arbres) dont le nombre d'éléments peut évoluer dynamiquement. Ces structures sont très utilisées en programmation.
- Ils permettent d'écrire des programmes plus compacts et efficaces.

# LES POINTEURS:

- Afin de manipuler les pointeur, deux opérateur sont utilisés:

Un opérateur '**adresse de**': **&** pour obtenir l'adresse d'une variable.

Un opérateur '**contenu de**': **\*** pour accéder au contenu d'une adresse.

- Exemple 1:

**int \* p;** //on déclare un pointeur vers une variable de type int

**int i=10, j=30;** // Déclaration, initialisation de deux variables de type int

**p=&i;** // on met dans p, l'adresse de la variable i (**p** pointe sur **i**)

**printf("\*p = %d \n",\*p);** //affiche : \*p = 10

**\*p=20;** // met la valeur 20 dans la case mémoire pointée par p (i vaut 20 après cette instruction)

**p=&j;** // p pointe sur j

**i=\*p;** // on affecte le contenu de p à i (i vaut 30 après cette instruction)

# LES POINTEURS:

## ○Exemple 2:

```
#include <stdio.h>

int main()
{
    float a, *p;
    p=&a;
    printf("Saisir un réel: ");
    scanf("%f", p);
    printf("L'adresse de a est: %x, la valeur de a est: %.2f \n", p,*p);
    *p+=1.5;
    printf("a=%.2f \n", a);

    return 0;
}
```

```
Saisir un réel: 1.2
L'adresse de a est: cc089314, la valeur de a est: 1.20
a=2.90
```

# LES POINTEURS:

- **Remarque :** si un pointeur P pointe sur une variable X, alors \*P peut être utilisé partout où on peut écrire X:

**X+=2** équivaut à **\*P+=2**

**++X** équivaut à **++ (\*P)**

**X++** équivaut à **(\*P)++** // les parenthèses ici sont obligatoires car l'associativité des opérateurs unaires \* et ++ est de droite à gauche

- A la déclaration d'un pointeur p, on ne sait pas sur quel zone mémoire il pointe. Ceci peut générer des problèmes :

**int \*p;**

**\*p = 10;** //provoque un problème mémoire car le pointeur p n'a pas été initialisé

**N.B.:** Toute utilisation d'un pointeur doit être précédée par une initialisation.

# LES POINTEURS:

- Exercice: Donnez les valeurs de A, B,C,P1 et P2 après chaque instruction

```
#include <stdio.h>

int main()
{
    int A = 1, B = 2, C = 3, *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;
}
```



# LES POINTEURS:

Les opérations arithmétiques sur les pointeurs:

- La valeur d'un pointeur étant un entier, certaines opérations arithmétiques sont possibles : ajouter ou soustraire un entier à un pointeur ou faire la différence de deux pointeurs.
- Pour un entier  $i$  et des pointeurs  $p$ ,  $p1$  et  $p2$  sur une variable de type  $T$ :
  - $p+i$  (ou  $p-i$ ) : désigne un pointeur sur une variable de type  $T$ . Sa valeur est égale à celle de  $p$  incrémentée (ou décrémentée) de  $i*\text{sizeof}(T)$ .
  - $p1-p2$  : Le résultat est un entier dont la valeur est égale à (différence des adresses)/ $\text{sizeof}(T)$ .
- Remarque:
  - on peut également utiliser les opérateurs  $++$  et  $--$  avec les pointeurs
  - La somme de deux pointeurs n'est pas autorisée

# LES POINTEURS:

- Exemple:

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

```
int main()
```

```
{
```

```
    float *p1, *p2;
```

```
    float z = 1.5;
```

```
    p1 = &z;
```

```
    printf("Adresse p1 = %x \n",p1);
```

```
    p1++;
```

```
    p2 = p1+1;
```

```
    printf("Adresse p1 = %x \t Adresse p2 = %x\n",p1,p2);
```

```
    printf("p2-p1 = %d \n",p2-p1);
```

```
}
```

```
Adresse p1 = b13ac2ec
```

```
Adresse p1 = b13ac2f0      Adresse p2 = b13ac2f4
```

```
p2-p1 = 1
```

# LES POINTEURS:

## Pointeurs et Tableaux:

- Comme on l'a déjà vu au, le nom d'un tableau **T** représente l'adresse de son premier élément (**T=&T[0]**). Avec le formalisme pointeur, on peut dire que **T** est un pointeur constant sur le premier élément du tableau.
- En déclarant un tableau **T** et un pointeur **P** du même type, l'instruction **P=T** fait pointer **P** sur le premier élément de **T** (**P=&T[0]**) et crée une liaison entre **P** et le tableau **T**.
- A partir de là, on peut manipuler le tableau **T** en utilisant **P**, en effet:
  - P pointe sur T[0] et \*P désigne T[0]
  - P+1 pointe sur T[1] et \*(P+1) désigne T[1]
  - P+i pointe sur T[i] et \*(P+i) désigne T[i]

# LES POINTEURS:

- Exemple:

```
short x, A[7]={5,0,9,2,1,3,8};
```

```
short *P;
```

```
P=A;
```

```
x=*(P+5);
```

- Le compilateur obtient l'adresse  $P+5$  en ajoutant  $5 * \text{sizeof}(\text{short}) = 10$  octets à l'adresse dans P.
- D'autre part, les composantes du tableau sont stockées à des emplacements contigus et  $\&A[5] = \&A[0] + \text{sizeof}(\text{short}) * 5 = A + 10$
- Ainsi, x est égale à la valeur de A[5] ( $x = A[5]$ )

# LES POINTEURS:

## ○ Exemple 2: Saisie et affichage d'un tableau

Avec  
l'indice i

```
#include <stdio.h>

int main()
{
    float T[100] , *pt;
    int i,n;
    do {
        printf("Entrez la taille du tableau: \n " );
        scanf("%d" ,&n);
    }while(n<0 || n>100);
    pt=T;
    for(i=0;i<n;i++)
    {
        printf ("Entrez T[%d]: \n ",i );
        scanf("%f", pt+i);
    }
    for(i=0;i<n;i++)
        printf ("T[%d] = %.2f \t",i,*(pt+i));

    return 0;
}
```

```
#include <stdio.h>

int main()
{
    float T[100] , *pt;
    int n;
    do {
        printf("Entrez la taille du tableau: \n " );
        scanf("%d" ,&n);
    }while(n<0 || n>100);

    for(pt=T;pt<T+n;pt++)
    {
        printf ("Entrez T[%d]: \n",pt-T);
        scanf("%f",pt);
    }
    for(pt=T;pt<T+n;pt++)
        printf ("T[%d] = %.2f \t",pt-T,*pt);

    return 0;
}
```

## Pointeurs et tableaux à deux dimensions:

- Le nom d'un tableau A à deux dimensions est un pointeur constant sur le premier élément du tableau A[0][0].
- En déclarant un tableau A[N][M] et un pointeur P du même type, on peut manipuler le tableau A en utilisant le pointeur P en faisant pointer P sur le premier élément de A ( $P = \&A[0][0]$ ), Ainsi :
  - P            pointe sur A[0][0]                      \*P          désigne A[0][0]
  - P+1        pointe sur A[0][1]                     \*(P+1) désigne A[0][1]
  - P+M        pointe sur A[1][0]                     \*(P+M) désigne A[1][0]
  - P+i\*M      pointe sur A[i][0]                    \*(P+i\*M) désigne A[i][0]
  - P+i\*M+j pointe sur A[i][j]                    \*(P+i\*M+j) désigne A[i][j]

# LES POINTEURS:

- Exemple: Saisie et affichage d'une matrice

```
#include <stdio.h>
#define N 2
#define M 3
int main()
{
    int i, j, A[N][M], *pt;
    pt=&A[0][0];
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
        {
            printf ("Entrez A[%d][%d]: ",i,j );
            scanf("%d", pt+i*M+j);
        }
    for(i=0;i<N;i++)
    {
        for(j=0;j<M;j++)
            printf ("A[%d][%d] = %d \t",i,j,*(pt+i*M+j));
        printf ("\n");
    }

    return 0;
}
```

```
Entrez A[0][0]: 12
Entrez A[0][1]: 85
Entrez A[0][2]: 63
Entrez A[1][0]: 77
Entrez A[1][1]: 10
Entrez A[1][2]: 3
A[0][0] = 12      A[0][1] = 85      A[0][2] = 63
A[1][0] = 77      A[1][1] = 10      A[1][2] = 3
```

# LES POINTEURS:

En C on peut définir:

- Un tableau de pointeurs :

`int *T[10];` //déclaration d'un tableau de 10 pointeurs d'entiers

- Un pointeur de tableaux :

`int (*pt)[20];` //déclaration d'un pointeur sur des tableaux de 20 éléments

- Un pointeur de pointeurs :

`int **pt;` //déclaration d'un pointeur pt qui pointe sur des pointeurs d'entiers



# LES POINTEURS:

## La fonction malloc:

- La fonction `void *malloc(size_t size);` de la bibliothèque `<stdlib>` permet de localiser et de réserver de la mémoire, sa syntaxe est : `malloc(N)`
- Permet de réserver un espace mémoire avec une taille donnée en **octets**
- Cette fonction retourne un pointeur de type `char *` pointant vers le premier octet d'une zone mémoire libre de **N** octets.
- En cas d'échec de réservation, la fonction retourne un pointeur nul (**NULL**)
- L'espace mémoire réservé n'est pas initialisé.
- **Exemple:** Si on veut réserver la mémoire pour un texte de 1000 caractères, on peut déclarer un pointeur `pt` sur `char (char *pt)`.
  - L'instruction: `T = malloc(1000);` fournit l'adresse d'un bloc de 1000 octets libres et l'affecte à **T**. S'il n'y a pas assez de mémoire, **T** obtient la valeur zéro (NULL).
- **Remarque:** Il existe d'autres fonctions d'allocation dynamique de mémoire dans la bibliothèque `<stdlib>`

# LES POINTEURS:

## La fonction malloc:

- Exemple:

```
float *t = malloc(10*sizeof(float));
```

// Réservation d'un espace mémoire pour stocker 10 réels (tableau)

## La fonction calloc:

- `void *calloc(size_t nmemb, size_t size);`
  - Réserve l'espace mémoire pour un tableau avec un nombre d'élément et la taille de chaque élément donnée, et **initialise l'espace mémoire par 0**.
  - Retourne un pointeur sur l'espace mémoire en cas de succès.
  - En cas d'échec de réservation, la fonction retourne un pointeur nul (NULL).

# LES POINTEURS:

## La fonction calloc:

- Exemple:

```
float *t = calloc(10, sizeof(float));
```

// Réservation d'un espace mémoire pour stocker 10 réels (tableau)  
avec des valeurs initiales de 0

- Si on n'a plus besoin d'un bloc de mémoire réservé par malloc/calloc, alors on peut le libérer à l'aide de la fonction free , dont la syntaxe est: free(pointeur);
- Si on ne libère pas explicitement la mémoire à l'aide de free, alors elle est libérée automatiquement à la fin du programme.

# LES POINTEURS:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char * pointeur = malloc(20 * sizeof(char)); //Allocation de 20 octets (Un char est égal à 1 octet)
    if(pointeur == NULL)
    {
        printf("L'allocation n'a pu être réalisée\n");
    }
    else
    {
        printf("L'allocation a été un succès\n");
        free(pointeur); //Libération des 20 octets précédemment alloués
        pointeur = NULL; // Invalidation du pointeur
    }
    return 0;
}
```

# LES POINTEURS: EXEMPLE SAISIE ET AFFICHAGE D'UN TABLEAU

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n;
    printf("Entrez la taille du tableau \n" );
    scanf("%d" ,&n);
    float *pt = malloc(n*sizeof(float));

    if (pt == NULL)
    {
        printf( " pas assez de mémoire \n" );
        system(" pause " );
    }
    printf(" Saisie du tableau \n " );
    for(i=0;i<n;i++)
    {
        printf ("Élément %d ? \n ",i+1);
        scanf(" %f" , pt+i);
    }
    printf(" Affichage du tableau \n " );
    for(i=0;i<n;i++)
        printf (" %f \t",*(pt+i));
    free(pt);

    return 0;
}
```