

# Programmation I:

## **Chapitre 1**

**Variables, types,  
opérateurs et expressions**

# Introduction

## Langages informatiques

- Un langage informatique est un outil permettant de donner des ordres (instructions) à la machine

A chaque instruction correspond une action du processeur
- Intérêt : écrire des programmes (suite consécutive d'instructions) destinés à effectuer une tache donnée

Exemple: un programme de gestion de comptes bancaires
- Contrainte: être compréhensible par la machine

## Langage machine

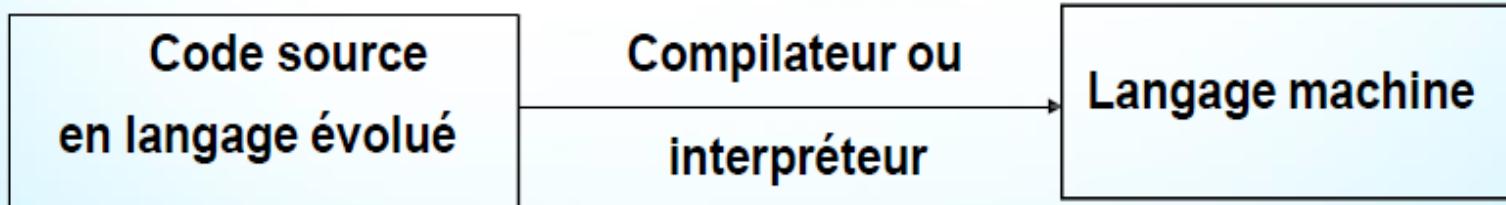
- Langage binaire: l'information est exprimée et manipulée sous forme d'une suite de bits
- Un bit (binary digit) = 0 ou 1 (2 états électriques)
- Une combinaison de 8 bits= 1 Octet possibilités qui permettent de coder tous les caractères alphabétiques, numériques, et symboles tels que ?, \*, &, ...

Le code ASCII (American Standard Code for Information Interchange) donne les correspondances entre les caractères alphanumériques et leurs représentation binaire,  
Ex. A= 01000001, ?=00111111

- Les opérations logiques et arithmétiques de base (addition, multiplication, ...) sont effectuées en binaire.

# Langages haut niveau

- Intérêts multiples pour le haut niveau:
  - proche du langage humain «anglais» (compréhensible)
  - permet une plus grande portabilité (indépendant du matériel)
  - Manipulation de données et d'expressions complexes (réels, objets,  $a^*b/c$ , ...)
- Nécessité d'un traducteur (compilateur/interpréteur), exécution plus ou moins lente selon le traducteur



# Compilateur/interpréteur

- Compilateur: traduire le programme entier une fois pour toutes



+plus rapide à l'exécution  
+sécurité du code source  
- il faut recompiler à chaque modification

- Interpréteur: traduire au fur et à mesure les instructions du programme à chaque exécution



+ exécution instantanée appréciable pour les débutants  
- exécution lente par rapport à la compilation

## Langages de programmation

- Deux types de langages:
  - Langages procéduraux: le processus de programmation est défini comme le développement d'une séquence de commandes qui manipulent des données pour produire le résultat souhaité.
  - Langages orientés objets: Le paradigme orienté objets (OOP) voit les unités de données comme objets "actifs", contrairement aux unités passives vues par le paradigme procédural.
- Exemples de langages:
  - **Fortran, Cobol, Pascal, C, ...**
  - **C++, Java, ...**

# Composantes d'un programme C

- Directives du préprocesseur
  - inclusion des fichiers d'en-tête (fichiers avec extension .h)
  - définitions des constantes avec **#define**
- déclaration des variables globales
- définition des fonctions (En C, le programme principal et les sous-programmes sont définis comme fonctions )
- Les commentaires : texte ignoré par le compilateur, destiné à améliorer la compréhension du code

exemple : **#include<stdio.h>**  
**main(){**  
                 **printf( "notre premier programme C \n");**  
                 **/\*ceci est un commentaire\*/**  
**}**

# Composantes d'un programme C

- **#include<stdio.h>** fichier d'entête contenant la déclaration des fonctions d'entrées-sorties dont la fonction printf
- La fonction **main** est la fonction principale des programmes en C: Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel de la fonction **main**.
- L'appel de **printf** avec l'argument "notre premier programme C\n" permet d'afficher : notre premier programme C et \n ordonne le passage à la ligne suivante
- En C, toute instruction simple est terminée par un point-virgule ;
- Un commentaire en C est compris entre // et la fin de la ligne ou bien entre /\* et \*/

# **Variables, types, opérateurs et expressions**

## Les 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 :
  - un nom (**Identificateur**)
  - un **type** (entier, réel, ...)

(Les types de variables en C seront discutés par la suite)

## Les identificateurs

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



Remarque : C distingue les majuscules et les minuscules. NOMBRE et nombre sont des identificateurs différents

## Les types de base

- Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre et le nombre d'octets à lui réservé en mémoire
- En langage C, il n'y a que deux types de base *les entiers* et *les réels* avec différentes variantes pour chaque type

### Remarques:

- Un type de base est un type pour lequel une variable peut prendre une seule valeur à un instant donné contrairement aux types agrégés
- Le type caractère apparaît en C comme cas particulier du type entier (un caractère est un nombre entier, il s'identifie à son code ASCII)
- En C il n'existe pas de type spécial pour chaînes de caractères. Les moyens de traiter les chaînes de caractères seront présentés aux chapitres suivants
- Le type booléen n'existe pas. Un booléen est représenté par un entier (un entier non nul équivaut à vrai et la valeur zero équivaut à faux)

## Types Entier

4 variantes d'entiers :

- **char** : caractères (entier sur 1 octet : - 128 à 127)
- **short ou short int** : entier court (entier sur 2 octets : - 32768 à 32767)
- **int** : entier standard (entier sur 2 ou 4 octets )
- **long ou long int** : entier long (4 octets : - 2147483648 à 2147483648)

Si on ajoute le préfixe **unsigned** à la définition d'un type de variables entières, alors la plage des valeurs change:

- **unsigned char** : 0 à 255
- **unsigned short** : 0 à 65535
- **unsigned int** : dépend du codage (sur 2 ou 4 octets)
- **unsigned long** : 0 à 4294967295

## Types Réel

3 variantes de réels :

- **float** : réel simple précision codé sur 4 octets de  
 $-3.4 \times 10^{38}$  à  $3.4 \times 10^{38}$
- **double** : réel double précision codé sur 8 octets de  
 $-1.7 \times 10^{308}$  à  $1.7 \times 10^{308}$
- **long double** : réel très grande précision codé sur 10 octets de  $-3.4 \times 10^{4932}$  à  $3.4 \times 10^{4932}$

## 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>;**

- Exemple:

```
int i, j,k;  
float x, y ;  
double z=1.5; // déclaration et initialisation  
short compteur;  
char c='A';
```

# Déclaration des constantes

- Une constante conserve sa valeur pendant toute l'exécution d'un programme
- En C, on associe une valeur à une constante en utilisant :
  - la directive `#define` :  
`#define nom_constante valeur`  
Ici la constante ne possède pas de type.  
exemple: `#define Pi 3.141592`
  - le mot clé `const` :  
`const type nom = expression;`  
Dans cette instruction la constante est typée  
exemple : `const float Pi =3.141592`

(Rq: L'intérêt des constantes est de donner un nom parlant à une valeur, par exemple `NB_LIGNES`, aussi ça facilite la modification du code)

# Expressions et opérateurs

- Une *expression* peut être une valeur, une variable ou une opération constituée par des valeurs, des constantes et des variables reliées entre eux par des *opérateurs*  
**exemples:** 1, b, a<sup>2</sup>, a+ 3\*b-c, ...
- Un *opérateur* est un symbole qui permet de manipuler une ou plusieurs variables pour produire un résultat. On distingue :
  - les *opérateurs binaires* qui nécessitent deux opérandes (ex : a + b)
  - les *opérateurs unaires* qui nécessitent un seul opérande ( ex:a++)
  - *l'opérateur conditionnel ?:* , le seul qui nécessite trois opérandes
- Une expression fournit une seule valeur, elle est évaluée en respectant des règles de priorité et d'associativité

# Opérateurs en C

- 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
- On distingue les opérateurs suivants en C :
  - **les opérateurs arithmétiques** : +, -, \*, /, %
  - **les opérateurs d'affectation** : =, +=, -=, \*=, /=, ...
  - **les opérateurs logiques** : &&, ||, !
  - **les opérateurs de comparaison** : ==, !=, <, >, <=, >=
  - **les opérateurs d'incrémentation et de décrémentation** : ++, --
  - **les opérateurs sur les bits** : <<, >>, &, |, ~, ^
  - **d'autres opérateurs particuliers** : ?:, sizeof, cast

## Opérateurs arithmétiques

- binaires : + - \* / et % (modulo) et unary : -
- Les opérandes peuvent être des entiers ou des réels sauf pour % qui agit uniquement sur des entiers
- Lorsque les types des deux opérandes sont différents il y'a conversion implicite dans le type le plus fort
- L'opérateur / retourne un quotient entier si les deux opérandes sont des entiers ( $5 / 2 \rightarrow 2$ ). Il retourne un quotient réel si l'un au moins des opérandes est un réel ( $5.0 / 2 \rightarrow 2.5$ )

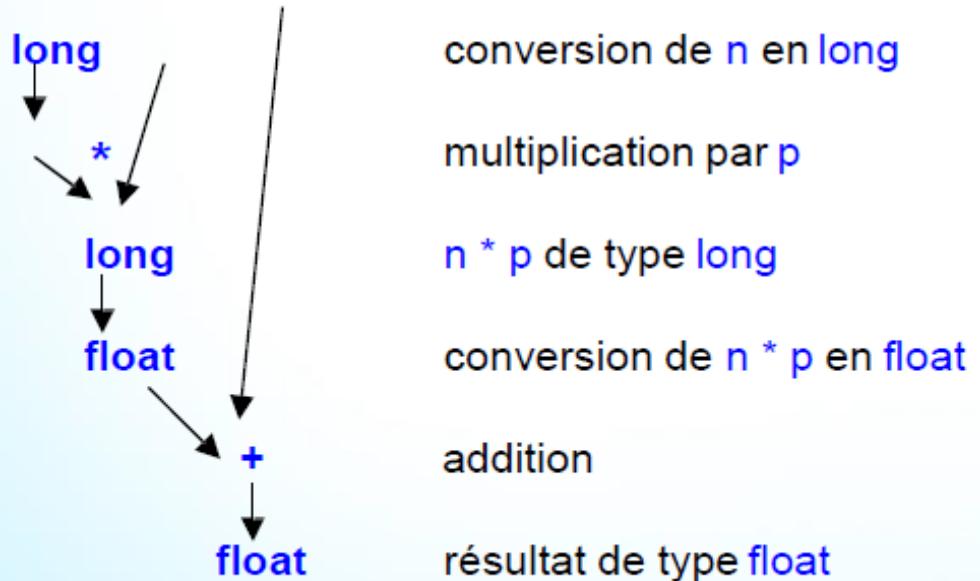
## Conversions implicites

- Les types `short` et `char` sont systématiquement convertis en `int` indépendamment des autres opérandes
- La conversion se fait en général selon une hiérarchie qui n'altère pas les valeurs `int` → `long` → `float` → `double` → `long double`
- **Exemple1 :** `n * x + p`    (`int n,p; float x`)
  - exécution prioritaire de `n * x` : conversion de `n` en `float`
  - exécution de l'addition : conversion de `p` en `float`
- **Exemple2 :** `p1 * p2 + p3 * x`    (`char p1, short p2, p3 ; float x`)
  - `p1, p2` et `p3` d'abord convertis en `int`
  - `p3` converti en `float` avant multiplication

## Exemple de conversion

Exemple :  $n * p + x$  (int n ; long p ; float x)

n \* p + x



# Opérateurs relationnels

- **Opérateurs**
  - $<$  : inférieur à                       $\leq$  : inférieur ou égal à
  - $>$  : supérieur à                       $\geq$  : supérieur ou égal à
  - $==$  : égal à                               $\neq$  : différent de
- Le résultat de la comparaison n'est pas une valeur booléenne, mais 0 si le résultat est faux et 1 si le résultat est vrai
- Les expressions relationnelles peuvent donc intervenir dans des expressions arithmétiques
- Exemple:  $a=2$ ,  $b=7$ ,  $c=4$ 
  - $b==3 \rightarrow 0$  (faux)
  - $a!=b \rightarrow 1$  (vrai)
  - $4*(a < b) + 2*(c \geq b) \rightarrow 4$

## Opérateurs logiques

- `&&` : ET logique    `||` : OU logique    `!` : négation logique
- `&&` retourne vrai si les deux opérandes sont vrais (valent 1) et 0 sinon
- `||` retourne vrai si l'une des opérandes est vrai (vaut 1) et 0 sinon
- Les valeurs numériques sont acceptées : toute valeur non nulle correspond à vraie et 0 correspond à faux
  - Exemple :  $5 \&\& 11 \rightarrow 1$   
 $!13.7 \rightarrow 0$

# Évaluation de && et ||

- Le 2<sup>ème</sup> opérande est évalué uniquement en cas de nécessité
  - **a && b** : b évalué uniquement si a vaut vrai (si a vaut faux, évaluation de b inutile car **a && b** vaut faux)
  - **a || b** : b évalué uniquement si a vaut faux (si a vaut vrai, évaluation de b inutile car **a || b** vaut vrai)
- **Exemples**
  - **if ((d != 0) && (n / d == 2))** : pas de division si d vaut 0
  - **if ((n >= 0) && (sqrt(n) < p))** : racine non calculée si n < 0
- L'intérêt est d'accélérer l'évaluation et d'éviter les traitements inappropriés

# Incrém entation et décrém entation

# Opérateurs de manipulations de bits

- **opérateurs arithmétiques bit à bit :**  
 $\&$  : ET logique    $|$  : OU logique    $^$  : OU exclusif    $\sim$  : négation
- Les opérandes sont de type entier. Les opérations s'effectuent bit à bit suivant la logique binaire

b1	b2	$\sim b1$	b1&b2	b1   b2	b1^b2
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0

- Ex :  $14 = 1110$  ,  $9 = 1001 \rightarrow 14 \& 9 = 1000 = 8$ ,  $14 | 9 = 1111 = 15$

## Opérateurs de décalage de bits

- Il existe deux opérateurs de décalage :  
 $\gg$  : décalage à droite               $\ll$  : décalage à gauche
- L'opérande gauche constitue l'objet à décaler et l'opérande droit le nombre de bits de décalage
- Dans le cas d'un décalage à gauche les bits les plus à gauche sont perdus. Les positions binaires rendues vacantes sont remplies par des 0
  - Ex : char x=14; ( $14=00001110$ )  $\rightarrow 14\ll 2 = 001110\textcolor{green}{00} = 56$   
char y=-7; ( $-7=11111001$ )  $\rightarrow -7\ll 2 = 111001\textcolor{green}{00} = -28$
- Rq : un décalage à gauche de  $k$  bits correspond (sauf débordement) à la multiplication par  $2^k$

## Opérateurs de décalage de bits

- Lors d'un décalage à droite les bits les plus à droite sont perdus.
  - si l'entier à décaler est non signé, les positions binaires rendues vacantes sont remplies par des 0
  - s'il est signé le remplissage dépend de l'implémentation (en général le remplissage se fait par le bit du signe)
- Ex : char x=14; (14=00001110)  $\rightarrow$  14>>2 = **00000011** = 3

## Opérateurs d'affectation combinés

- Soit un opérateur de calcul **op** et deux expressions *exp1* et *exp2*. L'expression *exp1= exp1 op exp2* peut s'écrire en général de façon équivalente sous la forme *exp1 op= exp2*
- Opérateurs utilisables :
  - $+=$        $-=$        $*=$        $/=$        $\%=$
  - $>>=$        $\&=$        $\wedge=$        $|=$
  - $<<=$
- Exemples :
  - $a=a+b$  s'écrit :  $a+=b$
  - $n=n\%2$  s'écrit :  $n\%=2$
  - $x=x*i$  s'écrit :  $x*=i$
  - $p=p>>3$  s'écrit :  $p>>=3$

## Opérateur de forçage de type (cast)

- Il est possible d'effectuer des conversions explicites ou de forcer le type d'une expression
  - Syntaxe : (*<type>*) *<expression>*
  - Exemple : int n, p ;  
*(double) (n / p);* convertit l'entier n / p en double
- Remarque : la conversion (ou casting) se fait après calcul  
*(double) (n/p) ≠ (double) n / p ≠ (double) (n) / (double) (p)*
  - float n = 4.6, p = 1.5 ;
  - (int) n / (int) p = 4 / 1 = 4
  - (int) n / p = 4 / 1.5 = 2.66
  - n / (int) p = 4.6 / 1 = 4.6
  - n / p = 4.6 / 1.5 = 3.06

# Opérateur conditionnel ?

- **Syntaxe:** `exp1 ? exp2 : exp3`

`exp1` est évaluée, si sa valeur est non nulle c'est `exp2` qui est exécutée, sinon `exp3`

- **Exemple1 :** `max = a > b ? a : b`

Si `a>b` alors on affecte à `max` le contenu de `exp2` càd `a` sinon on lui affecte `b`

- **Exemple2 :** `a>b ? i++ : i--;`

Si `a>b` on incrémente `i` sinon on décrémente `i`

## Opérateur séquentiel ,

- Utilité : regrouper plusieurs sous-expressions ou calculs en une seule expression
- Les calculs sont évalués en séquence de gauche à droite
- La valeur de l'expression est celle de la dernière sous-expression
- Exemples
  - `i++ , i + j;` // on évalue `i++` ensuite `i+j` (on utilise la valeur de `i` incrémentée)
  - `i++ , j = i + k , a + b;` // la valeur de l'expression est celle de `a+b`

# Opérateur sizeof

- **Syntaxe : sizeof (<type>) ou sizeof (<variable>)**  
fournit la taille en octets d'un type ou d'une variable
- **Exemples**
  - `int n;`
  - `printf ("%d \n",sizeof(int)); // affiche 4`
  - `printf ("%d \n",sizeof(n)); // affiche 4`

## Priorité et associativité des opérateurs

- Une expression est évaluée en respectant des règles de priorité et d'associativité des opérateurs
  - Ex: \* est plus prioritaire que +, ainsi  $2 + 3 * 7$  vaut 23 et non 35
- Le tableau de la page suivante donne la priorité de tous les opérateurs. La priorité est décroissante de haut en bas dans le tableau.
- Les opérateurs dans une même ligne ont le même niveau de priorité. Dans ce cas on applique les règles d'associativité selon le sens de la flèche. Par exemple:  $13\%3*4$  vaut 4 et non 1
- Remarque: en cas de doute il vaut mieux utiliser les parenthèses pour indiquer les opérations à effectuer en priorité. Ex:  $(2 + 3) * 7$  vaut 35

# Priorités de tous les opérateurs

- La priorité est décroissante de haut en bas dans le tableau.
- La règle d'associativité s'applique pour tous les opérateurs d'un même niveau de priorité. (-> pour une associativité de gauche à droite et <- pour une associativité de droite à gauche). Les parenthèses forcent la priorité

Priorité 1 (la plus forte):	<b>0</b>	→
Priorité 2:	<b>! ++ --</b>	←
Priorité 3:	<b>* / %</b>	→
Priorité 4:	<b>+ -</b>	→
Priorité 5:	<b>&lt; &lt;= &gt; &gt;=</b>	→
Priorité 6:	<b>== !=</b>	→
Priorité 7:	<b>&amp;&amp;</b>	→
Priorité 9 (la plus faible):	<b>= += -= *= /= %=</b>	←