# Algorithmique et Programmation, IMA

## Cours 2 : C Premier Niveau

Laure Gonnord
http://laure.gonnord.org/pro/teaching/
Laure.Gonnord@polytech-lille.fr

Université Lille 1 - Polytech Lille

















## Notations

Notations **algorithmiques** :

**Faire**
| action
**Tantque** *condition* ;

(Règles de) **Grammaire** du C :

```
instruction  ::=  expression ;
                |  instruction_composee
```

**Exemples** en C :

```
char c;
```

## Identificateurs

Mot désignant des variables, fonctions, types.

- Suite de charactères, chiffres et '_' (underscore) ;
- Commence par une lettre
- Distinction majuscule/minuscule

Convention : les **variables** sont en minuscule.











## Variables

Une **variable** est une place en mémoire qui a un **nom** (convention : en minuscules) :

- Une variable a un **type** qui définit quelles opérations sont valides (entier, booléen, réel, caractère, . . . )
- Elle doit être déclarée AVANT d'être utilisée.





## Type entier

Caractéristiques :

- Codé sur 2 (ou 4 octets, ou 8) : range = $[-2^{15}, 2^{15} - 1]$
- `sizeof(int)` rend 2 ou 4 ou 8
- Opérateurs : +, *, /, %(reste modulo)
- Comparaison : !=, ==, <=

```
int x;      // declaration simple
int z=10;   // declaration avec valeur initiale
```

## Type booléen

Caractéristiques :

- N'existe pas en C : `int`,
- Représentation : deux valeurs entières, 0 pour faux, 1 pour vrai (en fait toute valeur différente de 0) : `stdbool`
- Opérateurs et (`&&`), ou (`||`) : paresseux de gauche à droite

```
#include <stdbool.h>
bool a;
bool b=false;
```





## Type réel

Caractéristiques :

- Float 4 octets et double 8.
- Notation décimale ou exponentielle (12.3, -.38, .5e-11)
- Opérateurs : mêmes que int sauf %. / est la division réelle.

```
float x;        // declaration simple
float r=0.34; // declaration avec valeur initiale
```





## Type caractère

Caractéristiques :

- 1 octet : 256 valeurs de l'ASCII étendu
- Notation `'a'`
- Mais en fait, c'est un entier !
- Caractères spéciaux `\n` saut de ligne, `\t` tabulation, ...
- La relation d'ordre est induite par le code ASCII

```
char c;        // declaration simple
char c='a'; // declaration avec valeur initiale
int i='a'     // fonctionne aussi !
c = 80;       // ascii code 80 == P
char d;
d= c+1;       // d vaut ? Q!
```

## Expression numérique, expression booléenne

Expression **numérique** :

```
1+x+y+41
```

Expression **booléenne** :

```
(x<7 && y==2) || b /* avec b booléen */
```

En C on peut mixer mais on ne DOIT pas !





## Syntaxe générale des expressions en C

Grammaire simplifiée (voir poly p188) :

```
expression    ::=    identificateur
                   | constante
                   | chaine_litterale
                   | expression_num
                   | expression_bool
                   | expression_affectation
```

en C, l'affectation est une expression !





## Expression-affectation

```
expression_affectation   ::= lvalue = expression
```

Une **lvalue** est une expression qui doit délivrer une variable (par opp. à constante) : une variable simple, ou un élément de tableau, par exemple : `x = 7`.

**Sémantique** :

- Effet de bord : la valeur de droite est calculée et affectée à la variable de gauche.
- La valeur de l'expression entière est cette valeur calculée : `x = (y=8) +1` est une expression dont la valeur vaut .....

## Syntaxe générale d'un programme

(cf poly p 28)

```
programme   ::=   liste_declarations (option)
                  liste_defs_fonctions (option)
                  definition_main
```

**Exemple** simple :

```
#include <stdio.h>

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





## Syntaxe générale du main

(cf poly p 22, le main est un cas particulier de **fonction**)
**Syntaxe** simplifiée :

```
definition_main ::=
          int main ()
         {
            liste_declarations  (option)
            liste_instructions

            return 0;           (option mais on le met)
         }
```

Dans la partie suivante, nous allons voir quelques exemples d'instructions possibles.









## Syntaxe générale des instructions C

(cf poly p193) **Syntaxe** simplifiée (il n'y a pas tout !) :

```
instruction  ::=  expression ;
               |  instruction_composee
               |  instruction_cond
               |  instruction_iteration
               |   ...
```

## Instruction simple

Instruction **simple** : expression suivie d'un ;

```
x=4  ;                  //affectation
z=42+x;
printf("Hello!")  ;     //impression
scanf(%d,&x);           //demande d'un entier
toto(x);                // appel de procedure
w=f(z,x);               // appel de fonction
```

(ce sont des expressions, cf poly p 188)
Attention `2+4;` est donc bien une instruction simple !





## Instruction composée

On l'appelle aussi **bloc**.

```
instruction_composee ::=
 {
  liste_declaration (option)
  liste_instructions (option)
 }
```

Elle permet :

- grouper l'ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction (voir le IF)
- de déclarer des variables accessibles uniquement à l'intérieur du bloc.





## Conditionnelle - 1

Le test/la **conditionnelle** en pseudo code :

**Si** *condition* **alors**
| action_alors
**Fsi**

**Si** *condition* **alors**
| action_alors
**Sinon**
| action_sinon
**Fsi**

*condition* est une expression booléenne.





## Conditionnelle - 2

En C cela donne :

```
instruction_cond ::=
  if (expression) instruction
| if (expression) instruction_1 else instruction_2
```

```
if (a>b) max=a; else max=b;

if (a>b) if c<d u=v; else i=j;
// le else est associe au if le plus proche

if (a)    // teste si a!=0
{
...       // groupement d'instructions (bloc)
}
```

## Conditionnelle - 2

Important ! : l'instruction

```
if (x==4) t=3;
```

est différente de :

```
if (x=4) t=3;
```

Cette dernière est fortement déconseillée !





## Instruction d'itération : WHILE

La boucle **tant que** en pseudo code :

**Tq** *condition* **faire**
| action
**Ftq**

En C cela donne :

```
instruction_iteration ::=
  while (expression) instruction | ...
```

```
while (x>0) x=x-1;
```





## Instruction d'itération : WHILE - exemple

Longueur d'une ligne :

```
...
l=0;c=getchar();
while(c != '\n')
  {
    l=l+1;        //augmentation du compteur
    c=getchar()  //on avance !
   }
```





## Instruction d'itération : FOR - 1

La boucle **for** en pseudo code : forme très générale

**Pour** *(exp1 ;exp2 ;exp3)* **Faire**
| corps
**Fpour**

Avec :

- `exp1` : expression d'initialisation
- `exp2` : test de fin
- `exp3` : progression

En C cela donne :

```
instruction_iteration ::=
  for (expression1;expression2;expression3) instruction
```

## Instruction d'itération : FOR - 2

Utilisation classique avec **compteur**

**Pour** $i$ **de** *inf* **à** *sup* **Faire**
| corps
**Fpour**

(augmentation implicite de $1$ à chaque tour).

En C cela donne :

```
for (i=inf;i<=sup;i++)
    corps
```

À utiliser en priorité lorsqu'on connaît le nombre d'itérations





## Instruction d'itération : DO WHILE

**Faire**
| action
**Tantque** *condition* ;

En C cela donne :

```
instruction_iteration ::=
  do intruction while(expression)| ...
```

**Exemple**
Passer les lignes terminées par `\n` :

```
do
   c = getchar();
while (c != '\n');
```







## Conditionnelle

Écrire les suites d'**instructions** pour

- Afficher le maximum de deux entiers $x$ et $y$
- Afficher la valeur absolue de l'entier $z$
- Afficher `pair` ou `impair` selon la parité de l'entier $x$.
- Afficher le maximum de 3 entiers

## Instructions d'itération - 1

Écrire les suites d'**instructions** pour

- Afficher les entiers de $1$ à $10$ séparés par des espaces.
- Afficher les entiers de $10$ à $1$ séparés par des espaces.
- Ajouter les entiers de $1$ à $100$, puis afficher le résultat.
- Ajouter les entiers pairs de $6$ à $2048$, puis afficher le résultat.
- Afficher la liste des multiples de $3$ et des multiples de $5$ (dans l'ordre croissant) inférieurs à $60$ ; puis un point.





## Instructions d'itération - 2

Écrire un **programme** qui :

- Lit (au clavier) une suite de caractères qui finit par # et qui affiche le nombre de caractères lus différents de #
- Lit au clavier une suite de notes entre 0 et 20 et qui s'arrête lorsque l'utilisateur tape -1, puis affiche la moyenne des notes.





## Programme

Écrire un **programme** qui :

- lit 50 entiers rentrés au clavier ;
- calcule la somme de tous ces entiers en affichant la somme partielle à chaque nouveau nombre lu ;
- affiche à la fin la somme et la moyenne de ces entiers ;
- modifier le programme pour qu'il affiche la moyenne des entiers strictements positifs
- modifier ... entiers pairs

▶ On a besoin d'une fonction de sélection