


*Éléments d'informatique – Cours 3*  
*La programmation structurée en langage C*  
*Les instructions de contrôle `if` et `for`*

Pierre Fouilhoux et Lucas Létocart



- Éléments d'architecture des ordinateurs (+mini-assembleur) 
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, enregistrements
  - Fonctions d'entrées/sorties (scanf/printf)
  - Écriture et appel de fonctions
  - Débogage
- Notions de compilation
  - Analyse lexicale, analyse syntaxique, analyse sémantique
  - préprocesseur du compilateur C (include, define)
  - Édition de lien
- Algorithmes élémentaires
- Méthodologie de résolution, manipulation sous linux

## *Liens utiles*

- Un livre de la BU : *Le livre du C, premier langage (pour les vrais débutants en programmation)*, Claude Delannoy.
- <http://www.siteduzero.com/> (chercher langage C)
- <http://www.developpez.com/> (chercher langage C)
- codeblocks : <http://www.codeblocks.org/>
- ubuntu : <http://www.ubuntu-fr.org/>
- virtualbox : <http://www.virtualbox.org/>

## *Programmation*

Structure d'un programme C

Variables impératives et traduction de l'affectation

L'instruction de contrôle if

L'instruction de contrôle for

## *Trace*

## *Expressions booléennes*

Syntaxe

Constantes

## *printf/scanf (1)*

## *La compilation en pratique (gcc - présentation simplifiée)*

## *Demos*

## La programmation structurée

### Definition (Programmation structurée)

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :

1. exécuter les blocs les uns à la suite des autres (*séquence*)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (*sélection*)
3. recommencer l'exécution d'un bloc tant qu'une certaine condition est vraie (*répétition*).

Un bloc peut lui-même contenir une combinaison de blocs.

Cette idée simple conduisit à l'introduction de langages dits de haut niveau tel le langage C.

Aujourd'hui nous allons voir la sélection en langage C, le **if else**, et une première forme de répétition en C, le **for**

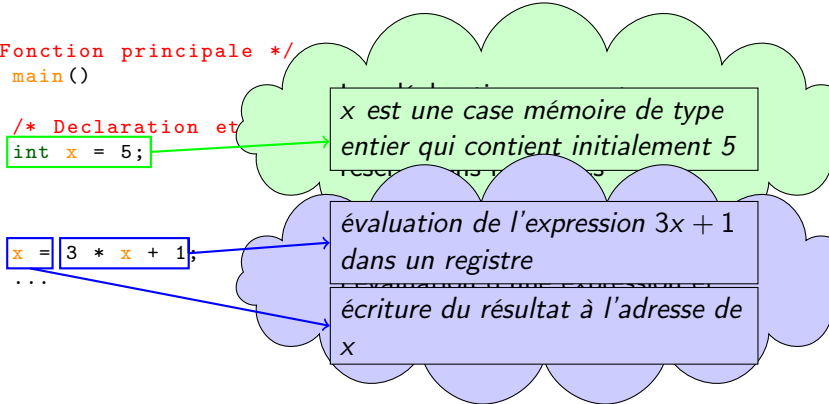
Les commentaires sont ignorés lors de la traduction en langage machine.

# Traduction de l'affectation

## Code source

```
...
/* Fonction principale */
int main()
{
    /* Declaration et
    int x = 5;
```

## Schéma de traduction



## *Expressions arithmétiques*

Les expressions arithmétiques permettent d'effectuer les opérations de calcul usuel.

Il y a trois types d'opérateurs :

- Opérateurs binaires (deux opérands) : +, -, \*, /, %
- Opérateurs d'affectation composée : +=, -=, \*=, /=, %=
- Opérateurs d'incrément et de décrémentation : ++, --

Remarques :

- Les expressions arithmétiques peuvent être combinées entre elles

$x = x * y + z / 2$

- Les opérateurs suivent des règles de priorité  
+ et - sont moins prioritaires que \*, /, %
- On peut contrôler les priorités à l'aide de parenthèses

$x = x * ((y + z) / 2)$



## *Opérateurs arithmétiques binaires*

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$

Toujours deux opérandes

Attention :

- $/$  effectue la division entière quand les opérandes sont des entiers (quotient de la division)
- $\%$  calcule le reste de la division entière des opérandes (obligatoirement des entiers)

```

1  ...
2  x = y + z;
3  y = z * 2;
4  q = 4 / 5;      q2 = z / y;
5  r = 4 % 5;      r3 = x % z;
```

## Opérateurs d'affectation composée

$+=, -=, *=, /=, \%=$

- L'opérande de gauche doit être une variable
- Association de l'affectation et d'un opérateur arithmétique  
*L'opérande de gauche reçoit le résultat du calcul entre l'opérande de gauche et l'opérande de droite*

**x** += 5;

- Les expressions de gauche sont équivalentes aux expressions de droite

1 **x** += 5;

2 **y** -= 10;

3 **z** \*= 2;

4 **t** /= 4;

5 **u** %= 5;

**x** = **x** + 5;

**y** = **y** - 10;

**z** = **z** \* 2;

**t** = **t** / 4;

**u** = **u** % 5;

## Opérateurs d'incrémentation et de décrémentation

++, --

- Une seule opérande placée à droite ou à gauche de l'opérateur  
*L'opérande reçoit la somme/soustraction de l'opérande et de 1*

```
1  /* suffixe      prefixe */
2  i++;           ++t;
3  j--;           --u;
```

- Attention : L'opérande est obligatoirement une variable
- La place de l'opérateur (notation suffixe ou préfixe) a une incidence sur la valeur de retour :
  - notation suffixe (i++), l'ancienne valeur est retournée
  - notation préfixe (++i), la nouvelle valeur est retournée
- Les expressions de gauche sont équivalentes aux expressions de droite

```
1  printf("%d", i++);           printf("%d", i); i = i + 1;
2  printf("%d", ++i);          i = i + 1; printf("%d", i);
```

## *L'instruction de contrôle if*

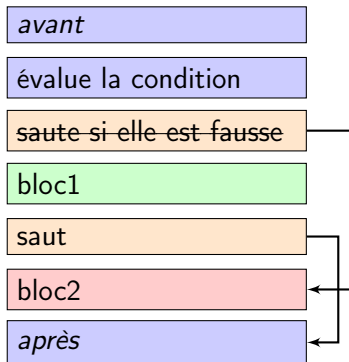
*Si condition alors instructions, sinon instructions*

*Syntaxe :* `if (condition) { bloc1 } else { bloc2 }.`

*Code source*

```
/* avant */
if (age < 18)
{
    permis = 0;
}
else
{
    permis = 1;
}
/* après */
```

*Schéma de traduction*



## L'instruction de contrôle for

Pour toutes les valeurs de  $n$  à  $m$ , faire instructions

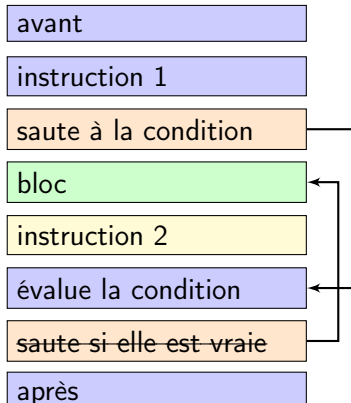
*Syntaxe :*

for (*instruct1*; *condition*; *instruct2*) { *bloc* }.

*Code source*

```
/* avant */
for (i = 0; i < 5; i = i + 1)
{
    printf("%d\n", i);
    ...
}
/* après */
```

*Schéma de traduction*



La variable  $i$  est appelée **variable de boucle**, elle doit être préalablement déclarée comme toute autre variable.

## Trace

```

1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int i; /* var. de boucle */
5
6      for (i = 0; i < 3; i = i + 1) /* pour chacune des 3 etapes */
7      {
8          printf("etape_\%d\n", i);
9      }
10     printf("i_vaut_\%d\n", i);
11
12     return EXIT_SUCCESS;
13 }

```

ligne	i	sortie écran
initialisation	?	
6	0	
8		etape 0
9	1	
8		etape 1
9	2	
8		etape 2
9	3	
10		i vaut 3
12		Renvoie EXIT_SUCCESS

## Expressions booléennes

Les *conditions* employées dans les structures de contrôle (if, for ou while) sont des **expressions booléennes**, pouvant être *Vrai*, *Faux* ou :

- des inégalités entre expressions arithmétiques

$$\begin{aligned} \textit{inégalité} := e_1 < e_2 \mid e_1 > e_2 \mid e_1 \neq e_2 \\ \mid e_1 \leq e_2 \mid e_1 \geq e_2 \mid e_1 == e_2 \end{aligned}$$

- ou des combinaisons logiques d'expressions booléennes :

$$\begin{aligned} \textit{condition} := (\textit{condition}) \ \&\& \ (\textit{condition}) && \text{(et)} \\ \mid (\textit{condition}) \ \mid \mid \ (\textit{condition}) && \text{(ou)} \\ \mid \text{!}(\textit{condition}) && \text{(non)} \\ \mid \textit{Vrai} \mid \textit{Faux} \mid \textit{inégalité} && \text{(cas de base)} \end{aligned}$$

## Constantes booléennes

- Certains langages possèdent un type booléen (admettant deux valeurs *true* et *false*) pour les expressions booléennes.
- En langage C, les expressions booléennes sont de type entier (*int*), l'entier *zéro* joue le rôle du Faux, l'entier *un* joue le rôle du Vrai et tout entier différent de zéro est évalué a vrai.
- On se donne deux constantes symboliques :

```
/* Declaration des constantes et types utilisateur */
#define TRUE 1
#define FALSE 0

int main()
{
    int continuer = TRUE; /* faut-il continuer ?*/

    if (continuer)
    {
        ...
    }
}
```



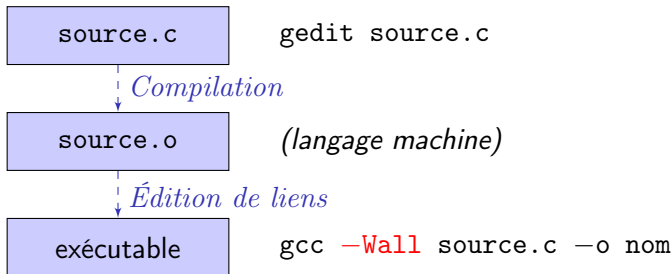
## *printf/scanf (1)* ~~✗~~

- Pour afficher un texte à l'écran, nous utilisons la fonction **printf** (*print formatted*).
- Chaque % dans le texte à afficher est substitué par la valeur formatée d'un **paramètre supplémentaire** de la fonction. Le caractère suivant le symbole % détaille la conversion à utiliser. La conversion %d met une valeur au format **entier décimal**.
- Exemples :
  - `printf("Bonjour\n")` affiche Bonjour et un saut de ligne
  - `printf("i vaut %d\n", i)` affiche i vaut suivi de la valeur décimale de i (et d'un saut de ligne)
  - `printf("(%d, %d)\n", 31, -4)` affiche (31, -4) et un saut de ligne.
- Réciproquement pour faire entrer dans le programme une donnée saisie par l'utilisateur, nous utiliserons **scanf**.
- Exemple : `scanf("%d", &x)`

## *La compilation en pratique (gcc)*

### *présentation simplifiée*

Les langages dits de haut-niveau nécessitent une **traduction en langage machine**, par un compilateur ou bien un interprète.



# Démos