Introduction au langage C

thibaut.lust@lip6.fr

Polytech Sorbonne

2020

`https://moodle-sciences.upmc.fr` (cours Informatique Générale EPU-R5-IGE)

Cours basé sur les diapositives créées par Julien Brajard

Plan du cours

1. Le langage C
2. Les commentaires
3. Les variables
4. Opérateurs
5. Structures de contrôle et répétitive
6. Conversion automatique et "cast"

1 Le langage C

2 Les commentaires

3 Les variables

4 Opérateurs

5 Structures de contrôle et répétitive

6 Conversion automatique et "cast"

"There are only two kinds of programming languages : those people always bitch about and those nobody uses."

Bjarne Stroustrup,
auteur du language de programmation C++.

Welcome to the machine

Qu'y-a-t'il à l'intérieur de l'ordinateur ?

Des communications avec "l'extérieur"

Une unité de traitement

De la mémoire

Que fait l'ordinateur ?

- Un ordinateur traite de façon automatique de l'information
  - Sous forme magnétique ou électrique,
  - Représentation binaire (i.e. 0 et 1).
- Besoin d'abstraction pour représentaiton l'information utilement.
  - Entiers,
  - Réels,
  - Caractères,
  - Mots.
- Besoin d'abstraction pour représenter les traitements
  - Affectations,
  - Boucles,
  - Opération arithmétiques,
  - Tests.

Historique

- 1969 : Création du système UNIX (Lab. Bell, K. Thompson et D. Ritchie). UNIX est intialement écrit en assembleur.
- 1970 : K. Thompson définit le langage B et réécrit UNIX dans ce langage.
- 1972 : D. Ritchie définit le langage C.
- 1973 : UNIX (version 5) est réécrit en C à 90%
- 1989 : l'ANSI (American National Standard Institute) définit une norme du langage C : C ANSI.

Caractéristiques du C

- Langage de programmation impérative.
- Programmation modulaire.
- Langage de bas-niveau (proche du sytème).
- Simplicité et minimalité du langage.
- Bibliothèques standards assez riches.
- Exécutable généré rapide et de petite taille.
- Langage permissif : faiblement typé (le compilateur ne vérifie pas tout).

Un programme en C

- Le `Code C` est un fichier texte (ASCII) lisible par des éditeurs de texte (emacs, gedit, vi, ...) et donc par vous.
- L'`Exécutable` est un fichier binaire directement interprétable par le sytème (mais illisble par vous).

Création d'un programme

1. Écriture d'un programme dans un éditeur de texte (emacs, gedit,...)
   - Sauvegarde dans un fichier : `fichier_source.c`
2. Compilation sous Linux avec gcc (dans un terminal)
   - `gcc -Wall -o fichier_executable fichier_source.c`
3. Exécution comme une commande (dans un terminal)
   - `./fichier_executable` *[paramètres]*

Exemple simple : "Hello World"

- Pré en-tête (inclusion de bibliothèques,...) commençant par #
- Commentaire
- Définition du `main` (fonction principale du programme)
- Accolade ouvrante {
- Bloc d'instruction : suite d'instructions entre { et } séparées par des ;
- Accolade fermante }

Fichier hello.c

```c
#include <stdio.h>

/* Programme principal */
int main()
{
  printf("Hello world");
  return 0;
}
```

En C, une instruction se termine par un point-virgule ;

Exemple simple : "Hello World"

Compilation

```
>>gcc -Wall hello.c -o hello
```

Exécution

```
>>./hello
```

Le programme affiche à l'écran

```
Hello world
```

Indentation

- La façon de présenter un programme n'a pas d'impact sur ses performances.
- Mais indenter un programme le rend plus lisible et donc plus facile à corriger et donc à faire fonctionner.
- Indenter : décaler de quelques espaces les parties du code dans un bloc.

Avec indentation

```
#include <stdio.h>

/*Programme principal*/
int main()
{
  printf("Hello world");
  return 0;
}
```

Sans indentation

```
#include <stdio.h> /*Pro
gramme principal*/int
main(){printf("Hello
world");return 0;}
```

1 Le langage C

2 Les commentaires

3 Les variables

4 Opérateurs

5 Structures de contrôle et répétitive

6 Conversion automatique et "cast"

Eagleson's Law : "Any code of your own that you haven't looked at for six or more months might as well have been written by someone else."

Les commentaires

- Pour mettre le reste d'une ligne en commentaire, on le précède de // :

```
int a ; // a est un entier
// fin de la déclaration des variables
```

- Pour mettre tout un bloc en commentaire, on l'encadre entre /* et */ :

```
#include <stdio.h>

/* Ce qui suit est la fonction principale de hello.c
Elle affiche : hello world*/

int main()
{...
```

Remarques sur les commentaires

Pourquoi utiliser les commentaires ?

- Maintenir le code (le modifier après une semaine, un mois, un an, ...)
- Travailler en équipe
- Expliquer une partie d'un algorithme
- Peut permettre le déboggage (voir cours n°6)
- Générer une documentation automatique (doxygen, javadoc, ...)

Attention

Commenter un programme fait partie du travail du programmeur (c'est-à-dire : **vous**).

1 Le langage C

2 Les commentaires

3 Les variables

4 Opérateurs

5 Structures de contrôle et répétitive

6 Conversion automatique et "cast"

Qu'est-ce qu'une variable ?

Emplacement destiné à recevoir des données.

Zone située dans la mémoire de l'ordinateur (repérée par une adresse)

Une variable :

- peut recevoir une valeur,
- peut être modifiée,
- doit être déclarée.

Déclaration de variables

Syntaxe

```
type_variable nom_variable ;
```

- Une variable est un "objet" manipulé par le programme, on peut également l'appeler *opérande*.
- Toute variable doit être typée.
- Règles de nommage d'une variable :
  - Composée de lettre, de chiffres et du caractère _ uniquement.
  - Ne doit pas être un mot réservé du C.
- Le type de la variable doit être séparé du nom de la variable par un espace.
- Une ligne de déclaration se termine par un ; (comme toute instruction C).

Exemples de déclaration

```
float x;
short n = 10;
int j,k;
unsigned int p;
```

Recommandations

- Choisir des noms de variables significatifs (qui ont un sens) ni trop longs, ni trop courts.
- Attention à bien distinguer majuscules et minuscules.

Mots réservés en C

Certains mots sont réservés au langage C, on ne peut les utiliser que dans un but bien défini (impossible de les utiliser pour nommer une variable par exemple).

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

Types du langage C

`char`	caractère	$\geq$8 bits
`short`	entier court	$\geq$16 bits
`int`	entier	$\geq$16 bits (souvent 32)
`long`	entier long	$\geq$32 bits
`float`	réel	32 bits
`double`	réel double précision	64 bits
`long double`	réel de longueur max	128 bits

Les données peuvent aussi être déclarées signées (par défaut) ou non signées (`unsigned`).

Plus de détails dans le cours 9.

Quelques utilisations simples

- Déclaration d'un entier

```
int n;
```

- Affectation d'une variable

```
n = 2;
```

En pseudo-langage, on écrirait : $n \leftarrow 2$.

- Un calcul simple :

```
int p;
p = 2*n + 1;
```

Affichage d'une variable

```
int n = 2 ;
printf("n est égal à %d\n",n);
```

- %d indique qu'on affiche un entier
- \n indique qu'on veut "afficher" un retour à la ligne.
- ,n affiche le contenu de la variable n à la place de %d.

Résultat à l'écran

```
n est égal à 2
```

Lecture d'une variable

```
int n = 2 ;
scanf("%d",&n);
```

- `%d` indique qu'on lit un entier,
- `&n` indique qu'on affecte à la variable `n` la valeur entrée au clavier.

Ne pas oublier le **&**

Plus de détails dans le cours 4.

1 Le langage C

2 Les commentaires

3 Les variables

4 Opérateurs

5 Structures de contrôle et répétitive

6 Conversion automatique et "cast"

Rappel : l'opérateur d'affectaction

L'opérateur d'affectation est le signe =

```
a = 1;
x = 2 * 3 + 5;
a = a + x;
```

a reçoit la valeur 1 ;
x reçoit la valeur 11 ;
a reçoit la valeur 12 ;

Les opérateurs arithmétiques

Les expressions arithmétiques sont évaluées de la gauche vers la droite en prenant compte de la préséance (priorité des opérateurs)

*	multiplication
/	division
%	reste de la division entière (modulo)
+	addition
-	soustraction

```
a+b*c   //   a + (b*c)
-c%d   //   (-c)%d
a*b + c%d   //   (a*b)+(c%d)
-a/-b+c   //   ((-a)/(-b))+c
a/b/c   //   (a/b)/c
```

Les opérateurs de comparaison

Ils permettent de comparer entre elles des expressions ou des variables

>	strictement supérieur à
>=	supérieur ou égal à
<	strictement inférieur à
<=	inférieur ou égal à
==	égal à
!=	différent de

```
x = 5;
y = 4;
x > y;  //=1 (vrai)
x == y;  //=0 (faux)
```

Le résultat de la comparaison est un entier de valeur :

- `non nulle` si la comparaison est vraie.
- 0 si la comparaison est fausse.

Les opérateurs logiques

Ils effectuent des opérations logiques (et, ou, ...) entre expressions ou variables.

`&&`	**et** logique
`||`	**ou** logique
`!`	**non** logique

`x`	0 (faux)	$\neq$ 0 (vrai)
`!x`	$\neq$ 0 (vrai)	0 (faux)

`x`	`y`	`(x && y)`	`(x || y)`
0 (faux)	0 (faux)	0 (faux)	0 (faux)
0 (faux)	$\neq$ 0 (vrai)	0 (faux)	$\neq$ 0 (vrai)
$\neq$ 0 (vrai)	0 (faux)	0 (faux)	$\neq$ 0 (vrai)
$\neq$ 0 (vrai)	$\neq$ 0 (vrai)	$\neq$ 0 (vrai)	$\neq$ 0 (vrai)

Traitement de bits

- Ils permettent des manipulations au niveau des bits dans la représentation binaire des nombres
- On ne peut les appliquer que sur des opérandes entiers (`short`, `int`, `long`), signé ou non.

&	ET bit à bit
|	OU inclusif bit à bit
^	OU exclusif bit à bit
<<	décalage à gauche
>>	décalage à droite
~	complément à un

```
int a = 5;//a=0101(binaire)
int b = 3;//b= 0011(binaire)
int c ;
c = a & b ; // c = ...
c = a | b ; // c = ...
c = a ^ b ; // c = ...
c = a << 1 ; // c = ...
c = b >> 1 ; // c = ...
c = ~b ; // c = 1100 = ...
```

Incrémentation/Décrémentation

Incrémenter(++) ou décrémenter (--) de 1 une variable.

```
x++ ;  // x = x+1
--y ;  // y = y-1
```

- Si l'opérateur est placé avant la variable (préfixé), celle-ci est incrémentée (ou décrémentée) avant son utilisation.
- Si l'opérateur est placé après la variable (postfixé), celle-ci est incrémentée (ou décrémentée) après son utilisation.

```
a = 2;
b = a++;// b=2, a=3
```

```
a = 2;
b = ++a; // a=3, b=3
```

Utilisez l'opérateur seul pour éviter les erreurs.

Opérateurs d'affectation élargie

Ils permettent d'alléger l'écriture des opérations de mise à jour de variable.

```
x += b;  // x = x + b
x -= b;  // x = x - b
x /= b;  // x = x / b
x *= b;  // x = x * b
x %= b;  // x = x % b
x |= b;  // x = x | b
x &= b;  // x = x & b
x <<= b;  // x = x << b
x >>= b;  // x = x >> b
```

Attention de ne pas confondre avec l'opérateur de comparaison <=.

L'opérateur conditionnel

expr1 ? expr2 : expr3

Cet opérateur (ternaire) permet d'agir en fonction du résultat d'évaluation d'une expression.

On évalue *expr1* :

- Si elle est vraie (donc non nulle), on évalue *expr2*
- Sinon, on évalue *expr3*

```
int x = 5, y = 4, z;
z = (x>y) ? x : y;
```

Si x>y, z prend la valeur de x, sinon celle de y.

Récapitulatif : les opérateurs du C

Unaires	`() [] -> .`
Unaires	`! + - ++ -- * & sizeof()`
Arithmétiques	`* / %`
Décalages	`<< >>`
Comparaison	`< <= > >=`
Egalités	`== !=`
Manip. bits	`&`
Manip. bits	`^`
Manip. bits	`|`
Logiques	`&&`
Logiques	`||`
Conditionnel	`? :`
Affectations	`= += *= ^= %= &= |= <<= >>=`

- Les opérateurs sur une même ligne ont même priorité.
- Les lignes sont classées dans l'ordre décroissant de priorité.

1 Le langage C

2 Les commentaires

3 Les variables

4 Opérateurs

5 Structures de contrôle et répétitive

6 Conversion automatique et "cast"

Introduction

Pour un comportement "intelligent" du programme

- Possibilité d'effectuer des choix, de se comporter différemment suivant les "circonstances" (test) :
  Instructions `if...else` et `switch`
- Possibilité de répéter plusieurs fois un ensemble d'instructions (boucle) :
  Instructions `do...while`, `while` et `for`

L'instruction `if...else`

Permet d'exprimer une prise de décision entre 2 choix.

```
if (expression)
   bloc_instructions1;
```

```
if (expression)
   bloc_instructions1;
else
   bloc_instructions2;
```

- *expression* : expression à évaluer (vraie ou fausse)
- *bloc_instructions1* : bloc d'instructions, ou instruction simple à effectuer si *expression* est vraie.
- *bloc_instructions2* : bloc d'instructions, ou instruction simple à effectuer si *expression* est fausse. Ce bloc est optionnel.

Possibilité d'imbriquer des instructions `if..else` dans d'autres expressions `if..else`. Le `else` se rapporte au dernier `if` rencontré.

L'instruction `if...else`

Exemple

if somme introduite = prix de la boisson **then**
  Delivrer boisson
  nbre_boissons ← nbre_boissons - 1
end if

```
#include <stdio.h>

int main()
{
  ...
  float somme = 0 ;
  ...
  if (somme == prix_boisson)
    {
      printf("Voici votre boisson\n");
      nb_boissons--;
    }
  ...
  return 0;
}
```

L'instruction if...else

Exemple

Max de 2 entiers

```
#include <stdio.h>

int main()
{
  int max, a, b;
  printf("Entrez 2 nombres :");
  scanf("%d %d",&a,&b);
  if (a > b) {
    max = a ;
    b = 0;
  }
  else {
    max = b ;
    a = 0 ;
  }
  printf("maximum : %d\n",max);
  return 0;
}
```

valeur absolue

```
#include <stdio.h>

int main()
{
  int val_abs, n;
  printf("Entrez 1 nombre :");
  scanf("%d",&n);
  if (n > 0) {
    val_abs = n ;
  }
  else {
    val_abs = -n ;
  }
  printf("v. a. de %d : %d\n",
      n,val_abs);
  return 0;
}
```

L'instruction switch

Permet d'exprimer une prise de décision à choix multiple.

```
switch (expression) {
   case constante_1 :
      bloc_instructions1
   ......
   case constante_N :
      bloc_instructionsN
   default:
      bloc_instructions
}
```

- Les arguments des `case` et *expression* doivent être des entiers.
- Si la valeur de *expression* correspond à l'un des arguments de `case`, alors le bloc d'instruction correspond est exécuté
- L'alternative `default` est optionnelle

Attention, à partir du moment où on est entré dans un `case`, on execute toutes les instructions des `case` en-dessous jusqu'à rencontrer un `break` qui nous fait sortir du `switch`.

L'instruction switch

exemple 1

détermination du prix d'une boisson

```
...
int selection ;
...
switch (selection)
{
  case 0 : prix_boisson = prix_jus_orange ;
           break ;
  case 1 : prix_boisson = prix_eau_plate ;
           break ;
  case 2 : prix_boisson = prix_eau_gazeuse ;
           break ;
  default : printf("Erreur de selection\n");
}
...
```

L'instruction switch

exemple 2

```
#include <stdio.h>

int main()
{
 int a ;
 printf("\nEntrez un nombre :");
 scanf("%d",&a);
 switch (a)
 {
  case 0 : printf("\nNul");
           break;
  case 1 :
  case 2 : printf("\nPetit");
  case 3 :
  case 4 : printf("\nMoyen");
           break;
  default : printf("\Grand");
 }
 return 0;
}
```

Test d'exécution

```
Entrez un nombre :1
Petit
Moyen

Entrez un nombre :3
Moyen

Entrez un nombre :852
Grand

Entrez un nombre :-1
Grand
```

La boucle `for`

Permet de répéter une même action un certain nombre de fois (généralement connu à l'avance).

```
for (expr1 ; expr2 ; expr3)
  bloc_instructions
```

- *expr1* : condition initiale fixée.
- *expr2* : Test de continuation de boucle. Le *bloc_instructions* est exécuté tant que cette valeur est vraie (et s'arrête quand elle est fausse).
- *expr3* : Opération effectuée à chaque tour de boucle (ex : incrémentation).

Toutes ces expressions sont facultatives. `for (;;)` est une boucle infinie.

La boucle for

Exemple

```
#include <stdio.h>

int main()
{
  int i;
  for (i=1 ; i <= 5 ; i++)
  {
    printf("\nBonjour %d fois",i);
  }
  return 0;
}
```

Test d'exécution

```
Bonjour 1 fois
Bonjour 2 fois
Bonjour 3 fois
Bonjour 4 fois
Bonjour 5 fois
```

Ne pas oublier de déclarer la variable de boucle `i`.

- `i=1` : On entre dans la boucle avec `i` initialisé à `1`.
- `i<=5` : On repasse dans la boucle tant que i reste inférieur ou égal à 5.
- `i++` : A chaque passage dans la boucle, `i` est incrémenté de `1`.
- A chaque passage dans la boucle, on affiche `Bonjour` puis la valeur de `i` puis le mot `fois`.

La boucle `while`

Permet de répéter une même action tant qu'une condition est vraie (généralement on ne connaît pas à l'avance le nombre d'itérations effectuées par la boucle).

```
while (condition)
  bloc_instructions
```

- *condition* est évaluée, puis, si elle est vraie *bloc_instructions* s'exécute, puis on réévalue *condition*, si elle est vraie *bloc_instructions* s'exécute, etc.
- *condition* est donc évaluée avant chaque passage dans le corps de la boucle. On n'est pas sûr d'exécuter au moins une fois le bloc d'instructions.
- Si des variables doivent changer de valeur à chaque passage de boucle, il est nécessaire de les mettre à jour dans *bloc_instructions*.

La boucle while

Exemple

```
...
while(nb_boissons != 0)
{
 printf("Voulez-vous prendre une boisson ? ") ;
 scanf("%d",&choix);
 if (choix==1)
 //Le code 1 correspond à la réponse "oui"
 {
  nb_boissons--;
 }
}
```

La boucle do...while

Permet de répéter une même action tant qu'une condition est vraie (généralement on ne connaît pas à l'avance le nombre d'itérations effectuées par la boucle).

```
do
{
  bloc_instructions
} while (condition)
```

- *bloc_instructions* s'exécute, puis *condition* est évaluée, puis, si elle est vraie *bloc_instructions* s'exécute, puis on réévalue *condition*, etc.
- *condition* est donc évaluée après chaque passage dans le corps de la boucle. On est sûr d'exécuter au moins une fois le bloc d'instructions.
- Si des variables doivent changer de valeur à chaque passage de boucle, il est nécessaire de les mettre à jour dans *bloc_instructions*.

La boucle do...while

2 exemples

```
int main()
{
  int i=1, x=20;
  do
  {
    i = i*2;
  }while (i<x);
  return 0;
}
```

A la fin de l'exécution

- i = ...
- x = ...

La boucle do...while

2 exemples

```
int main()
{
  int i=1, x=20;
  do
  {
    i = i*2;
  }while (i<x);
  return 0;
}
```

A la fin de l'exécution

- i = ...
- x = ...

```
int main()
{
  int i=1, x=20;
  do
  {
    i = i*2 ;
    x = x/2 ;
 }while ((i<x) && (x>0));
 return 0;
}
```

A la fin de l'exécution

- i = ...
- x = ...

Quel type de boucle choisir ?

1 Le langage C

2 Les commentaires

3 Les variables

4 Opérateurs

5 Structures de contrôle et répétitive

6 Conversion automatique et "cast"

La conversion de Saint Paul (vers 1690)
Luca Giordano (1634-1705)
Musée des Beaux-Arts de Nancy

Conversions implicites

Si une expression contient des opérandes de types différents (expressions mixtes), des conversions se font automatiquement.

Hiérarchie :

`int` → `long` → `float` → `double`

```
int a ; float x;
a*x ; // sera de type float
a*a + a*x // sera de type float
```

Les conversions se font au fur et à mesure de l'évaluation, opération par opération.

Conversions explicites (cast)

Il est possible de forcer une conversion d'une expression quelconque dans un type de son choix par la syntaxe :

(type) operande ;

(type) expression ;

```
float x = 5.2 ;
int a = 5 ;
(int) x ; // = 5
(float) a ; // = 5.0
(int) (x+a) ; // =10
```

Remarques

- Une division entre entiers est par défaut une division entière (euclidienne).

```
float x ;
x = 3/2 ; //x = 1.0
x = ((float) a) / ((float) b) ; //x=1.5
```

Remarques

- Une division entre entiers est par défaut une division entière (euclidienne).

```
float x ;
x = 3/2 ; //x = 1.0
x = ((float) a) / ((float) b) ; //x=1.5
```

- La conversion se fait avant l'affectation

```
int a = 3 , b = 2 ; float x ;
x = a / b ; // x = 1.0
x = (float) (a/b) ; // x = 1.0
x=  ((float) a) / ((float) b) ; //x=1.5
```