Langage C

Rémy Bulot

remy.bulot@univ-amu.fr
pageperso.lis-lab.fr/remy.bulot/gii

Progression :

I. Premiers pas en C
II. Type entier, réel, constantes et variables
III. Les entrées-sorties simplifiées
IV. Expressions arithmétiques et logiques
V. Syntaxe des instructions

VI. Les entrées-sorties
VII. Quelques compléments commodes
VIII. Variables indexées (tableaux)
IX. Adresse et pointeur
X. Les Fonctions
XI. Les objets structurés
XII. Les fichiers
XIII. Piles, files, listes chaînées

I.
Premiers pas en C

Le C a été mis au point par **D.Ritchie** et **B.W.Kernighan**

- Au début des années 70
- Pour écrire un système d'exploitation portable : **UNIX**
- Le succès d'UNIX a entraîné celui de C :
- un des langages le plus utilisé (gestion des réseaux, …).

La première définition de ce langage a été donnée dans leur livre « The C programming language ».

Ce langage a été normalisé en 1989 : C-ANSI (*American National Standards Institute*)

Quelques caractéristiques du langage :

- adapté aux programmes de taille moyenne
- programmation de bas niveau (proche assembleur)
- comme de haut niveau (**programmation structurée**)
- indépendant de la machine (**portabilité**)
- C original et C-ANSI (1989) plus fiable au niveau syntaxique

Ce cours : un sous-ensemble du C-ANSI

Premières remarques sur la constitution d'un programme C :

- Programme C : fichier texte *instructions* + *commentaires*
- Commentaires :

  /* *coucou* */

  // *commentaire sur une ligne*

1. Développement d'un programme

1 : L'algorithme : « recette » décrite en pseudo-français

2 : Code source : transcription de l'algorithme dans le langage

fichier construit sous un **éditeur de texte**

3 : Compilation : traduction du code source en code machine exécutable

4 : Edition de lien : (link) compléter le code avec des fonctions prédéfinies déjà compilées (*sin , sqrt,* …)

➔ **Fichier exécutable** : résultat de cette séquence d'opérations

Développer un programme :

- réaliser l'enchainement des 4 étapes
- tester l'exécution avec différents jeux de données
- recommencer la séquence jusqu'à obtention d'un fonctionnement satisfaisant

2. Code source

Un fichier source est une succession d'éléments indépendants :

- ***directives*** *pour le préprocesseur (lignes qui débutent par #)*
- ***constructions de types***
- ***déclarations*** *de variables et de fcts externes*
- **définitions** de variables (réservation d'espace mémoire)
- **définitions de fcts** dont la fct principale *main( )*

3. Compiler/exécuter

Fichier source *salut.c*

```
/* Auteur : Melanie Zettofret
   Objectif : afficher coucou ! */

#include <stdio.h>

void main(void)
{
  printf("coucou !\n");
}
```

Dans une console UNIX :

Compiler avec la commande

gcc salut.c -o salut

Lancer l'exécution du code en tapant

./salut

II.
Constantes, types et variables

1. Codage de l'information

a) le bit

- Plus petit élément physique constitutif de la mémoire
- Deux états possibles : 0 ou 1
- Un ensemble de n bits permet de coder 2^n informations différentes
- Un octet : assemblage de 8 bits ➔ $2^8 = 256$ combinaisons possibles
- Coder une information sur n bits : convention pour représenter une "information" parmi 2^n informations possibles d'un type donné
- Pour un type d'information donné, le nombre d'octets est préalablement fixé par le langage en fct du nombre de valeurs à représenter.

b) Codage d'un entier

Exemple sur 3 bits pour les entiers dans [0, 8 [et dans [-4, 4 [

nombre	codage
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

nombre	codage
-4	100
-3	101
-2	110
-1	111
0	000
1	001
2	010
3	011

- Sur 1 octet, **{1001 1010}** représente suivant le cas
  - -102 pour les entiers relatifs
  - +154 pour les entiers naturels
  - '€' pour les caractères alphanumériques (code ASCII)
- Sur 2 octets (16 bits), on peut coder tous les entiers dans [-32768, 32767]
- Sur 4 octets (32 bits), on peut coder les entiers relatifs sur 9 chiffres

c) Codage d'un réel

Deux informations à coder :

- la précision (nombre de chiffres) 3.14 3.1415926
- la dimension (l'exposant) $1. \times 10^{-2}$ $1. \times 10^{6}$

Les réels sont souvent codés sur 32 ou 64 bits, dont 6 pour l'exposant (de -32 à +31)

d) Codage d'un caractère alphanumérique

Les caractères sont codés sur un octet.

Chaque caractère est associé à un entier dans [0, 256[(8 bits) : le code ASCII

'a' : 97
'b' : 98
'A' : 65
'0' : 48
'?' : …

2. Les constantes

Ce sont les valeurs définies **explicitement** dans le code source.

On distingue :

- *les caractères alphanumérique-étendus*

  `'a'  'A'  '1'  ' '  '@'  '\n'`

- *les entiers*

  `0  -1  32767`

- *les réels*

  `0.  3.14  0.314e1  0.314E1`

ATTENTION : **1** est différent de **'1'**

3. Identificateur

- Pourquoi faire ? donner des noms aux variables et aux actions !
- 31 caractères max, majuscule ≠ minuscule

  identificateur : *lettre*[*lettre* ou *chiffre* ou _]
- Exemple : **`x  x1  delta  Delta  Bond_007`**

- Mots **réservés** du langage (voir fiche résumé) :
  - une trentaine
  - tous en minuscule

4. Variable

- associée à un emplacement mémoire (1, 2, 4 octets ou plus)
- on en crée (presque) autant qu'on veut
- en donnant des noms qui suggèrent bien leurs rôles
  - `compteur, somme, valeurMin, ...`
- le contenu peut changer dans le temps
- contient toujours une valeur !!!

Une variable sera toujours définie avant son utilisation par :

- un **type** (sa nature qui détermine sa taille et le codage)
- et un **identificateur** (son nom)

4.1 Type d'une variable

- convention de codage de l'information
- dans un emplacement de taille préfixée (ex : compter sur 8 bits)

7 types de base en C :

- ✓ **char** : petit entier codé sur 1 octet (entre –128 et 127)
- ✓ **short** : entier de taille moyenne sur 2 octets (de –32768 à 32767)
- ✓ **long** : grand entier sur 8 octets (jusqu'à 19 chiffres)
- ✓ **int** : entier dont la taille est liée au processeur (mot machine)
- ✓ **float :** réel avec 7 chiffres significatifs
- ✓ **double** : réel de grande précision
- ✓ **long double** : réel de très grande précision

Le qualificatif **unsigned** : redéfinit l'intervalle de validité à partir de 0

Un ***unsigned short*** est dans [0, 65535]

Un ***unsigned char*** est dans [0, 255]

4.2 Définition d'une variable :

Objectifs :

- réserver un emplacement mémoire
- qui soit adapté au codage de l'information
- associer un nom (identificateur) à cet emplacement

Syntaxe pour la définition de variable :

-> *type variableInit* [, *variableInit*] ;

où *variableInit* :

-> *identificateur*

-> *identificateur* = *expression constante*

Exemple :

```
float x ;
long i, j=1, k ;
char c1='A', c2 = 66 ;
```

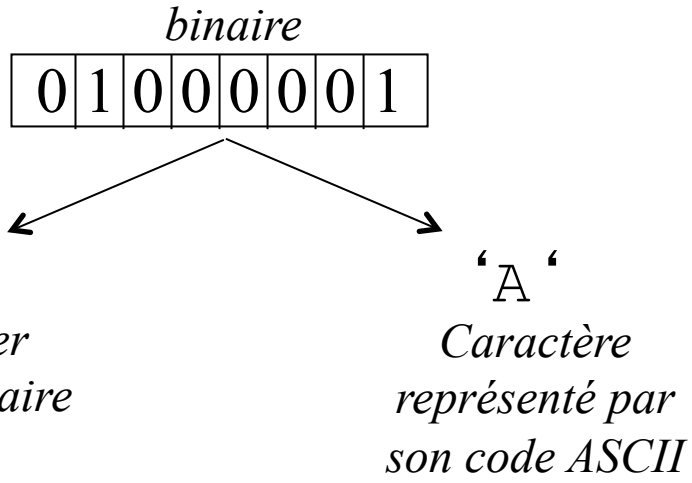
Exemples d'erreur :

```
int i=0, j=i+1 ;
char a=300 ;
```

Remarques :

- Le C ne fait pas de différence entre les petits entiers et les caractères qui sont déclarés indifféremment comme des *char*

```
char c ;
```



- Le code ASCII des lettres et des chiffres respecte la relation d'ordre partiel intuitif des caractères alphanumériques

  'a' (97), 'b' (98), …

  'A' (65), 'B' (66), …

  '0' (48), '1' (49), '2' (50), …

5. Affectation d'une variable :

Syntaxe provisoire de l'instruction d'affectation :

variable = *expression* ;

Exemple :

```
/* définition d'une variable */
short debut, cpt;

/* affectation d'une variable */
debut = 0 ;     /* initialisation */
cpt = debut+1; /* initialisation */

cpt = cpt+1 ;  /* modification */
```