IN 261 - Pré-formation C 1

ENSTA ParisTech

1 Un premier programme en C

aloha.c

```
#include <stdio.h> /* Pour acceder a printf. */

int main ()
{
  printf ("Aloha cousins...\n") ;
  return (0) ;
}
```

Un programme en C a toujours 1 et 1 seule ≪fonction≫ principale : **main**. C'est par ce point que commence l'exécution d'un programme.

Cette fonction commence par l'entête : **int main** (). Nous verrons plus tard qu'il existe une petite variante.

Son ≪*corps*≫ est constitué de tout ce qu'il y a entre les **{}**.

Elle doit toujours se terminer par le ≪retour≫ d'une valeur entière : **return** (0).

Par convention, pour le **main**, 0 signifie ≪*pas d'erreur*≫, $\neq 0$ signifie ≪*erreur survenue*≫.

Chaque instruction (≪*ordre*≫) est terminé par un **;** (sauf pour les blocs dont il est question plus loin, ou des formes avancées d'utilisation des instructions).

Un programme s'écrit sous forme d'un texte ≪*source*≫. On utilise donc un éditeur de texte.

Une fois écrit, il faut transformer ce texte en exécutable. On utilise un compilateur C. Dans un terminal :

```
gcc aloha.c
```

Le fichier exécutable créé par défaut se nomme `a.out`.

Vous pouvez alors exécuter votre programme en tapant dans un terminal :

```
./a.out
```

2 Constructions élémentaires de C

2.1 Types et variables

Les informations que l'on souhaite traiter peuvent être de natures variées : entiers naturels, rationnels, valeurs de vérité (*vrai* / *faux*), lettres, texte ....

Les langages de programmation fournissent des types de données variés à cet effet.

En C, 3 types scalaires de base :

- Les entiers (**int**).
- Les réels (**float** et **double**).
- Les caractères (**char**).

On appelle booléen le type des valeurs de vérité (≪vrai≫ et ≪faux≫).C ne fournit pas de vrais booléens. En C, ce sont des entiers avec pour convention 0 ≡ ≪faux≫ et tout sauf 0 ≡ ≪faux≫.

L'utilisation de **#include** <stdbool.h> permet de bénéficier de la pseudo-définition de booléens :

— Type `bool` (alias de **int**).
— Constante `true` (=1) et `false` (= 0).

Un programme manipule des données stockées en mémoire. Il faut donc pouvoir manipuler des « *réceptacles* » d'information : des variables.

La première chose à faire pour pouvoir manipuler une variable est de la déclarer puis de l'initialiser (i.e. lui donner une valeur).

On déclare une variable en donnant son type suivi de son nom :

```
int main ()
{
  int my_var ;   ◄
  return (0) ;
}
```

On initialise une variable directement au moment de sa déclaration en lui affectant une valeur par la construction = :

```
int main ()
{
  int my_var = 42 ;   ◄
  return (0) ;
}
```

2.2 Expressions et instructions

En simplifiant, on différencie 2 types de constructions de C :
— Les expressions, qui « *ont une valeur* » (elles « *valent* ») et n'ont pas d'effet.
— Les instructions, qui « *ont un effet* » (elles « *font* ») sans forcément valoir quelque chose.

La différence réelle entre instruction et expression est plus complexe et plus floue en C.

2.2.1 Expressions

Expressions de base :
— Variables : toute variable déclarée et de portée accessible.
— Entiers : 344578 −5
— (∼ Booléens : true false)
— Caractères : 'U' '\n'
— Flottants : −4.6 5e-12
— Chaînes : "plop" "\\tGlop\\n"

On combine les expressions pour en exprimer de plus complexes à l'aide d'opérateurs :
— Arithmétiques entre entiers et/ou flottants :
+, −, /, ∗, % (modulo)
— Relationnels (« tests ») entre expressions de même type :
== (égalité), != (inégalité), < (inférieur à), >, <= (inférieur ou égal à), >=
— Logiques entre booléens :
&& (et), || (ou), ! (négation)
— Binaire (« bit-à-bit ») entre entiers/caractères :
~ (négation), ^ (ou exclusif), & (et), | (ou)
— Appels de fonctions : détaillé ultérieurement même si déjà superficiellement aperçu.

Par exemple, le prédicat $x \in]-1;2[\cup [10;10+3[$
$\Longrightarrow$ ((x > −1)&& (x < 2))|| ((x >= 10)&& (x < (10 + 3)))

2.2.2 Instructions

L'instruction d'affection permet de stocker une valeur dans une variable. Elle est notée par = (**ne pas confondre** avec == le test d'égalité) et attend :

- **À gauche** : une variable (plus généralement, une case mémoire).
- **À droite** : une expression dont la valeur sera stockée.

```
int main ()
{
  int my_var = 42 ;
  my_var = (my_var * 90) / 100 ;  ◂
  return (0) ;
}
```

Note : il existe des instructions d'affectation-arithmétique abrégées qui mixent affectation et opérateurs.

Note : Les instructions d'affectation peuvent être utilisées dans certains contextes d'expression.

ATTENTION !!! On ne lit jamais la valeur contenue dans une variable si cette dernière n'a pas été initialisée ou préalablement affectée. Dans le cas contraire, la valeur obtenue sera quelconque (donc vide de sens).

L'instruction conditionnelle permet effectuer un traitement si une condition est vraie :

if (*expression***) { *instruction*(*s*) ; }**

... ou (optionnellement) un autre traitement sinon :

if (*expression***) { *instruction*$_1$(*s*) ; } else { *instruction*$_2$(*s*) ; }**

La construction **{ ... }** permet de grouper plusieurs instructions : c'est un « *bloc* ».

```
int main ()
{
  int my_var = 5 ;
  if (my_var < 0) {  ◂
    printf ("Negatif.\n") ;
  }  ◂
  else {  ◂
    printf ("Positif.\n") ;
  }  ◂
  return (0) ;
}
```

ATTENTION !!! Il n'y a pas de **;** après l'accolade fermant un bloc.

L'instruction while permet effectuer une boucle tant qu'une condition est vraie :

while (*expression***) { *instruction*(*s*) ; }**

```
int main ()
{
  int my_var = 5 ;
  while (my_var > 0) {  ◂
    printf ("Pas nulle.\n") ;
    my_var = my_var - 1 ;
  }  ◂
  return (0) ;
}
```

3 Affichage à l'écran

La fonction `printf` permet d'afficher une chaîne de caractères dans laquelle on peut « *insérer l'affichage* » de valeurs.

```
#include <stdio.h>

int main ()
{
```

```
    printf ("Une chaine...\n") ;
    printf ("Valeur de 3 + 5 = %d\n", 3 + 5) ;
}
```

Première chaîne (≪ *format* ≫) toujours obligatoire et décrit la ≪ *forme* ≫ de l'affichage.

À chaque "%♪" doit correspondre un argument passé à `printf`.

Nombreux "%♪" dont : %d pour afficher une *valeur entière signée*, %f pour une valeur flottante.

Nécessite l'utilisation de `stdio.h` : **#include** <stdio.h>

```
printf ("I understand\n") ;
⟶ I understand
printf ("I understand %s.\n", "very well") ;
⟶ I understand very well.
printf ("Hi user%d. Did you know that Pi is %f?\n", 2 + 3, 4.0 * atan (1.0)) ;
⟶ Hi user5. Did you know that Pi is 3.141593?
```

Rappel des formats :

%d	un **int**
%ld	un **long int** en décimal
%u	un **unsigned int** en décimal
%x	un **int** en hexadécimal
%f	un **float**
%lf	un **double**
%e	un **double** en notation scientifique
%.7lf	un **double** avec 7 chiffres après la virgule
%07d	un **int** en décimal sur 7 digits (padding frontal avec des 0)
% 7d	un **int** en décimal sur 7 digits (padding frontal avec des ' ')
%c	un **char** (comme caractère ASCII, pas comme entier)