Université de Nice-Sophia Antipolis
Polytech Elec3

Lundi 1er octobre 2018
Durée: 2h

Types élémentaires, variables, opérateurs et expressions
Travaux Dirigés – Séance n. 1

---

1 Objectifs

Cette séance sera divisée en deux parties. Tout d'abord, nous nous familiariserons avec l'environnement de travail que vous utiliserez tout au long de l'année pour éditer et compiler vos programmes C. Dans un second temps, nous étudierons les premières bases du langage C : les types simples, des variables et des expressions.

2 Première prise en main

Le langage C est un langage d'écriture de système apparu au cours de l'année 1972. Développé par Dennis Ritchie et Ken Thompson, il a été spécialement conçu pour la réécriture du système Unix jusque là écrit en langage machine.

Il fait suite aux langages d'écriture de système BCPL et surtout B, d'où son nom C. La syntaxe du langage C au départ est très libre et laisse place à beaucoup de variantes. À la fin des années 70, le livre de Dennis Ritchie et Brian Kernighan popularisera le langage et en fixera la forme pendant près d'une dizaine d'années.

En 1989 l'institut national de normalisation Américain (ANSI) aboutit à une norme pour le langage.

En 1990, l'ISO (International Organization for Standardization) a également adopté la norme ANSI, c'est la norme ISO C90. En 1999 et 2011, deux nouvelles normes C99 et C11 ont été produites par l'ISO, proposant des évolutions du langage C.

Support de cours Pour cette série de TD, une partie cours sera présentée sur les feuilles de TD. Si vous cherchez des informations sur une fonction particulière, la commande `man` pourra vous donner une bonne documentation.

Vous aurez également besoin du support de cours de C accessible sur le site `http://elec.polytech.unice.fr/~vg/` dans la section enseignement. Sur ce site, vous devrez récupérer et imprimer le document « cours-C » et l'avoir avec vous en séance de TD.

L'environnement Pour développer vos programmes,vous utiliserez l'éditeur de texte Emacs. Lorsque vous éditerez un programme C (un fichier avec comme extension `.c`), vous le visualiserez et le compilerez à l'aide de cet éditeur. Les commandes abrégées à connaître parfaitement sont les suivantes :

- `Ctrl-x Ctrl-f` Ouvrir un fichier ;
- `Ctrl-x Ctrl-s` Sauvegarder un fichier ;
- `Ctrl-x Ctrl-c` Quitter Emacs ;
- `M-x compile` Compiler un fichier C.

!!! Attention quand vous éditez un fichier C n'oubliez pas son extension !!!

Compilateur Pour créer les fichiers exécutables, à partir des fichiers sources écrits en C, vous devez utiliser un compilateur.

Dans cet enseignement, nous utiliserons le compilateur de GNU : `gcc`. Une ligne de commandes typique pour compiler un fichier C sera de la forme suivante :

```
gcc -Wall -o nomexecutable nomfichier.c
```

L'option `Wall` indique à `gcc` d'afficher *tous* les messages d'avertissement. Ces 3 premières options doivent être mises *systématiquement*. L'option `-o nomexecutable` sert à définir le nom du fichier qui sera produit par `gcc` et pourra être exécuté par la suite. Le fichier à compiler est toujours le dernier paramètre et doit comporter **une extension** `.c`.

2.1 Un programme : Bonjour à tous

Le premier programme que nous allons écrire est le classique « Bonjour à tous » qui écrit un message de bienvenue sur la sortie standard (*i.e.* l'écran). Le texte source de ce programme est :

```
1  /*
2   * Ce programme affiche le message 'Bonjour à tous'.
3   * @author vg@unice.fr 23/09/2010
4   */
5  #include <stdlib.h>
6  #include <stdio.h>
7
8  int main (void) {
9     printf("\tBonjour à tous\n");
10    printf("\t***************\n");
11    return EXIT_SUCCESS;
12 }
```

exercice 1) Avec Emacs, éditez le programme précédent dans le fichier `bonjour.c`. Ce programme est composé de 3 parties :

- les 3 premières lignes sont des commentaires. Tous vos programmes devront être correctement commentés, avec des *commentaires de documentation* et des *assertions* qui permettent de prouver la *validité* de votre programme ;
- la ligne 4, est une directive au *préprocesseur* (dont nous parlerons ultérieurement) nécessaire pour (entre autres) l'affichage sur l'écran ;
- les lignes 5 à 9 définissent la fonction `main` (tout programme C doit comporter une (et une seule) fonction `main` par laquelle débute l'exécution du programme). Ici, le corps de cette fonction exécute la fonction `printf` qui écrit la valeur de son paramètre sur la sortie standard (l'écran). Le corps du programme peux être changé pour effectuer des actions différentes. Chacune des instructions du corps d'un programme doit être suivie du caractère point-virgule (`;`).

Compilez ce programme à l'aide de la commande abrégée `M-x compile`. Remplacez la ligne contenant `make -k` par la commande : `gcc -Wall -o bonjour bonjour.c`. Cette commande produit un exécutable `bonjour`. Pour l'exécuter, utilisez la commande `./bonjour` dans un terminal.

exercice 2) Enlevez un des symboles '\t'. Que se passe-t-il ? Pourquoi ?

exercice 3) Enlevez le premier symbole '\n'. Que se passe-t-il ? Pourquoi ?

exercice 4) Enlevez le second symbole '\n'. Que se passe-t-il ? Pourquoi ?

exercice 5) Écrivez sur la sortie standard : votre nom, votre âge et votre groupe, chacune sur des lignes différentes.

2.2 Utilisation de printf

La fonction `printf` permet d'écrire autre chose que des chaînes de caractères, comme par exemple, des nombres entiers ou à virgule. Pour se faire, vous devez utiliser des *spécificateurs de conversion* qui seront remplacés par des valeurs passées en paramètre. Pour le moment, les spécificateurs de conversion à connaître sont :

- `%d` qui sera remplacé par un entier,
- `%f` qui sera remplacé par un nombre à virgule,
- `%c` qui sera remplacé par un caractère, et
- `%s` qui sera remplacé par une chaîne de caractères.

exercice 6) Dans le fichier `trois.c`, écrivez un programme avec une fonction `main` dont le corps contient l'appel à la fonction `printf` suivant :

```
printf("le chiffre trois : %d,\nà virgule : %f\n",3, 3.0);
```

exercice 7) Compilez et exécutez.

exercice 8) Reprenez l'exercice 5 en utilisant des spécificateurs de conversion.

3 Types, variables et expressions

3.1 Types et constantes

Nous avons vu dans la partie précédente des moyens d'afficher des valeurs constantes. Mais à quoi correspondent-elles réellement en C ?

En C, il existe de nombreux types simples qui définissent des ensembles de valeurs possédant des caractéristiques communes.

- Les *entiers* (positifs et négatifs) sont représentés par les types **short**, **int**, **long** et**long long**. Ce qui les distinguent, c'est leur cardinalité. Les constantes entières sont des suites de chiffres pouvant être négatifs (précédés du signe -). Par exemple : `1234`, `-324`...
  Les constantes précédentes sont en notation décimales. C propose deux notations octale et hexadécimale en faisant précéder respectivement le nombre par un `0` ou `0X`.
  Il est possible de définir un ensemble d'entiers strictement positifs en faisant précéder le nom du type par **unsigned**. Par exemple **unsigned long**.
- Les *réels* sont représentés par les types **float** et **double**. Le type **double** offre une plus grande précision dans la représentation de réels. Les constantes réelles s'écrivent comme habituellement : `0.23451`, `123134.3`, `345.89e-3`.
- Les caractères sont définis par le type **char**. C'est en fait un sous-type du type entier. Une constante de type **char** est représentée par un caractère entre deux apostrophes. Par exemple : `'a'`, `','`...

exercice 9) Indiquez les notations de constantes valides et le type des nombres suivants :

```
  0.31   +273.3   0.005e+3     0x10
   010     .389         15   0x5e-4
 33.75    1.5+2      3,250      .E1
  1234      3E5         08    10e-4
 0X1a2     0037     1e2768    0x1A2
```

exercice 10) Déterminez le type des constantes suivantes :


```
  100     033      'a'     2
 0x10     .23    "nom"    '2'
 "a"     2311    '\n'    "2"
```

exercice 11) Écrivez un programme qui affiche les nombres :

- 230 en utilisant une notation hexadécimale ;
- 123 en utilisant une notation octale
- 156003 en utilisant un notation exponentielle
- 12423215 en utilisant une constante **long**

3.2 Variables

Les variables sont des noms qui servent à désigner des données dans la mémoire centrale de l'ordinateur. Dans le langage C, elles doivent toujours être déclarées avant leur utilisation. Une déclaration de variable se compose de deux ou trois parties : le type de la variable, son nom et une initialisation facultative. Donc par exemple, pour déclarer une variable `i` de type **int**, cela peut s'écrire de la manière suivante :

int `i = 0;` ou **int** `i;`

Plusieurs variables de même type peuvent être déclarées à la suite, en les séparant par une virgule. Par exemple :

int `i, j;` ou **int** `i = 1, j = 0;`

Tout au long du programme on peut affecter une nouvelle valeur à une variable à l'aide de l'opérateur d'affectation `=`.

exercice 12) Écrivez un programme dans lequel vous déclarez deux variables entières, la première prenant comme valeur 1, la seconde n'étant pas initialisée. Affichez leurs valeurs. Expliquez le résultat.

exercice 13) À l'aide d'une variable entière, affichez la valeur du caractère `'a'`.Expliquez le résultat.

3.3 Expressions

Les expressions sont notées sous forme infixe et sont composées d'opérandes et d'opérateurs.

Cinq types d'opérateurs différents peuvent être utilisés : des opérateurs arithmétiques, de comparaison, logiques, de bits et d'affectation. Chacune des opérations est typée et le type du résultat est fonction de ses opérandes. Par exemple, l'opération addition (`+`) calcule un entier si ses deux paramètres sont des entiers, ou un réel si ses paramètres sont de type double.

3.3.1 Opérateurs arithmétiques

En C, vous pouvez trouver les opérateurs arithmétiques habituels.

`+`	addition	`x + 1`
`-`	soustraction	`x - y`
`*`	multiplication	`x * y`
`/`	division	`x / y`
`%`	modulo	`x % y`

Opérateurs unaires

`-`	négation	`-x`

3.3.2 Autres opérateurs

Opérateurs logiques (booléens) :
(l'entier 0 représente la valeur booléenne « faux », tout autre entier représente la valeur « vrai »)

`&&`	et	`x && y`
`||`	ou	`x || y`
`!`	non	`!x`

Opérateurs binaires :
(les opérations binaires sont effectuées bits à bits)

`&`	et	`x & y`
`|`	ou	`x | y`
`^`	ou exclusif	`x ^ y`
`~`	non	`~ x`
`<<`	décalage logique à gauche	`x << 1`
`>>`	décalage logique à droite	`x >> 1`

Opérateurs de comparaison :
(ils renvoient comme résultat 0 ou 1)

`>`	supérieur	`x > y`
`<`	inférieur	`x < y`
`>=`	supérieur ou égal	`x >= y`
`<=`	inférieur ou égal	`x <= y`
`==`	égal	`x == y`
`!=`	différent	`x != y`

exercice 14) Donnez le résultat des opérations suivantes : (sur papier)

```
  033 & 2    033 | 02   0xff & 0x0         6 ^ 3
033 && 2   02 == 0x2   2.0 == 2.0   ~0xffffffff
```

exercice 15) Vérifiez les résultats de l'exercice précédent sur l'ordinateur.

3.3.3 Affectation

La valeur désignée par une variable peut être modifiée à l'aide de l'opérateur d'affectation `=`. En particulier, il permet d'affecter le résultat de l'évaluation d'une expression. Par exemple, si `y == 3`, `x = (y = y + 1)` affectera la valeur `4` à `x`.

Il existe une manière abrégée pour écrire les affections d'une variable à elle même. L'expression `x op = y` correspond à l'opération `x = x op y`. Par exemple pour l'addition `x += y` correspond à la même action que `x = x + y`.

En C une variable peut être incrémentée avec l'opérateur `++` et décrémentée avec l'opérateur `--`. Si l'opérateur est placé avant la variable, la valeur renvoyée par l'expression est la valeur de la variable après l'incrémentation. Si elle est placée après la variable, la variable est incrémentée après l'évaluation de l'expression.

exercice 16) Écrivez de la manière la plus compacte possible les séries d'expression suivantes :

1. `y = x + y; x = x + 1; y = y + 3;`
2. `b1 = b1 & b2; b2 = b1 == b3; b2 = b2 + 1;`

exercice 17) Écrivez un programme qui transforme une lettre minuscule en lettre majuscule.

3.3.4 Types des expressions

Le résultat de l'évaluation d'une expression est typé. Ce type dépend du type des opérandes utilisés. Si les opérandes sont de même type, par exemple l'addition de deux entiers, le résultat est de type **int**.

Si les opérandes sont de type différents, des conversions de types *implicites* ont lieu. En C, ces conversions sont nombreuses et les règles de conversion complexes. Nous reviendrons dessus plus tard.

Il est possible d'expliciter la conversion d'un type en un autre (quand la conversion de type a un sens). Cette opération de conversion explicite s'appelle en C un *cast*. Pour cela, il suffit de mettre le type souhaité devant la valeur à convertir..

exercice 18) Écrivez le programme qui affiche la partie entière des expressions suivantes, si a = 2, b = 3, c = 4, d = 5, e = 6 et f = 7 :

$$a^2 - c + \frac{a}{bc + \frac{c}{d + \frac{e}{f}}} \qquad \frac{-b + b^2 - 4ac}{2a} \qquad \frac{\frac{1}{a} + \frac{1}{b}}{c + d}$$

3.3.5 Priorités

En notation infixe des expressions, il est nécessaire de fixer des règles de priorité pour déterminer l'ordre d'évaluation des opérateurs. En C, les règles de priorité sont nombreuses, et un tableau les décrivant est disponible à la page 37 du support de cours mentionné précédemment.

exercice 19) Déterminer le résultat des expressions suivantes, avec x = 1 et y = 2 :

```
2 + 3 * 4 + 12 % 3
3  << 4 + 23 - 2 == 5 != 4
1 != (x++ == --y)
~ 4 + 22 - 3 && 6
```