TP n° 01 - Module 1207

1. Introduction

a) Démarrer la machine sous LINUX et ouvrir un terminal.
b) Créer ensuite sur le bureau, le répertoire `M1207-TP-01-XXX` (où il faut remplacer XXX par votre nom).
c) Lancer un éditeur de texte (**geany** par exemple) et taper le code suivant :

```
#include <stdio.h>
int main()
{
  printf("Bonjour\n");
  return 0;
}
```

d) Enregistrer ce fichier dans le répertoire précédent en lui donnant le nom `bonjour.c`.
e) Changer le répertoire courant du terminal pour qu'il corresponde à votre répertoire de travail.
f) Lancer alors la compilation du fichier source avec la commande : `gcc -Wall bonjour.c -o bonjour`
g) Exécuter enfin le programme exécutable obtenu : `./bonjour`

2. Les différents formats d'affichage de printf()

La fonction `printf()` permet d'afficher du texte et des variables selon plusieurs formats (d'où le f à la fin) à l'aide du caractère % qui à une signification particulière. Voici les principaux formats :

%d **%i**	Affiche un entier décimal (int)	**%g** **%G**	Affiche un réel en notation courte (double)	**%c**	Affiche un caractère (char)
%u	Affiche un entier non signé (unsigned int)	**%f** **%F**	Affiche un réel en notation « normale » (double)	**%s**	Affiche une chaîne de caractères (char *)
%o	Affiche un entier en base 8 – octal (int)	**%e** **%E**	Affiche un réel en notation exponentielle (double)	**%p**	Affiche une adresse mémoire ou un pointeur (void *)
%x **%X**	Affiche un entier en base 16 – hexadécimal (int)				

a) Copier le fichier `bonjour.c` dans le fichier `format1.c` et remplacer l'instruction `printf("Bonjour\n");` par : `printf("Valeur : %d\n",10);`. Compiler ce nouveau programme, exécuter le et remarquer que le texte qui s'affiche ne contient plus le `%d` mais qu'il a été remplacé par la valeur spécifiée après la virgule.
b) Évidement l'exemple précédent n'est pas très parlant, car on a utilisé la valeur fixe 10 (On aurait pu écrire `printf("Valeur : 10\n");`) ! Mais en général, on se sert de variables qui peuvent servir à faire des calculs... Pour déclarer, initialiser et afficher une variable entière, on peut alors écrire :

```
int x=48;
printf("Valeur : %d\n",x);
```

Tester ce code (dans `format1.c`) puis ajouter des instructions pour afficher à nouveau la variable mais en hexadécimal.
c) Créer un nouveau programme (`format2.c`) pour tester l'affichage de réels (`double x=3.14159265,y=-654.321e+3;`) avec les différents formats `%g`, `%f` et `%e`, sachant que dans le format d'une instruction `printf()`, on peut ajouter autant de % qu'il y a de valeurs correspondantes entre les parenthèses : `printf("X=%g et Y=%g\n",x,y);` Noter bien l'ordre des variables `,x,y` et non pas `,y,x` !
d) Créer ensuite un nouveau programme (`format3.c`) pour tester l'affichage des caractères (`%c`) et des chaînes de caractères (`%s`) :

```
char x='a',y=36; // Caractères
char* s="Texte a afficher"; // Chaîne de caractères
```

e) Qu'obtient-on s'il on utilise le format `%p` sur la variable `s` ?

3. Demande de valeurs à l'utilisateur

Il est souvent utile, pendant l'exécution du programme, de pouvoir demander des valeurs à l'utilisateur afin de ne pas avoir à modifier le code source pour changer les données et le recompiler à chaque fois.
Plusieurs fonctions permettent de réaliser cela (principalement : `scanf()`, `gets()` et `fgets()`). Dans un premier temps, nous utiliserons, pour simplifier les programmes, la fonction **`scanf()`**, mais nous verrons un peu plus tard qu'il est en fait déconseillé d'utiliser `scanf()` et `gets()` pour ne pas introduire de « buffer overflow » dans un programme.

Cette fonction **`scanf()`**utilise également, un peu à la manière de `printf()`, le caractère %. Seulement, pour modifier une variable, il faut en fait indiquer son **adresse mémoire** (ce qui se fait avec le caractère **&**). Voici un petit exemple pour demander un entier :

```
int x;
printf("Entrer un nombre entier : ");
scanf("%d",&x);
printf("Valeur : %d\n",x);
```

a) Écrire le programme `scanf1.c` qui permette de tester ces 4 lignes de code. Tester avec différentes valeurs : 1234, 012, 1.23, 12abc, 0x12abc et zerty.

b) Écrire le programme `scanf2.c` qui permette de demander une valeur réelle de type `double` (`%lf`) : long float.
c) Écrire le programme `scanf3.c` qui permette de demander un caractère de type `char` (`%c`).
d) Écrire le programme `scanf4.c` qui permette de demander une valeur chaîne de caractères (`%s`). Dans ce cas là, il faut créer une zone mémoire dans laquelle on pourra écrire des caractères. Le plus simple pour l'instant est de définir un tableau de caractères avec une taille prédéfinie (par exemple 20 caractères) : `char str[20];`. Mais attention, dans le `scanf()`, il ne faut pas utiliser le caractère & comme auparavant car en fait `str` représente déjà l'adresse mémoire du tableau de caractères !
e) Tester le programme avec les chaînes « salut » puis « salut a tous ». Que faut-il en conclure ?

4. Les branchements conditionnels

Le mot clé `if` permet de réaliser un bloc d'instructions si l'expression fournie est vraie et sinon le programme exécute la partie `else` :

```
if (x == 5)
{
  // Bloc exécuté si x est égal à 5
}
else
{
  // Bloc exécuté sinon
}
```

a) Écrire un programme `age.c` qui demande l'age de l'utilisateur et qui affiche « Bonjour » s'il a plus de 30 ans et « Salut » dans les autres cas.
b) Écrire un programme `somme.c` qui demande deux nombres entiers, les additionne, affiche le résultat et indique s'il est pair ou impair.
c) Écrire un programme `ordre.c` qui demande trois nombres entiers puis les affiche dans l'ordre croissant.

5. Les boucles

Les boucles permettent de répéter plusieurs fois une suite d'instructions donnée. Il y a 3 façons de réaliser des boucles en langage C avec les mots clé : `while`, `do/while` et `for` :

```
int x=0;
while (x < 5) {
  printf("%d\n",x);
  x=x+1;
}
```

```
int x=0;
do {
  printf("%d\n",x);
  x=x+1;
} while (x < 5);
```

```
int x;
for (x=0 ; x < 5 ; x=x+1){
  printf("%d\n",x);
}
```

a) Tester ces trois exemples dans le programme `boucle1.c`.
b) Écrire le programme `boucle2.c` qui affiche les nombres de 0 à 20 en avançant de 2 en 2 (avec 1 boucle).
c) Écrire le programme `boucle3.c` qui affiche les nombres de 30 à 12 en reculant de 3 en 3 (avec 1 boucle).
d) Écrire le programme `boucle4.c` qui affiche (avec 2 boucles) :

```
0 0
0 1
1 0
1 1
2 0
2 1
```

e) Écrire le programme `boucle5.c` qui affiche (avec 2 boucles) :

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
```

f) Écrire le programme `boucle6.c` qui affiche le menu suivant en permanence (tant que l'utilisateur n'a pas tapé sur la lettre 'q'). On pourra utiliser l'instruction `switch/case` (voir cours) et se servir de la fonction `getchar()` pour « escamoter » le caractère retour chariot lors de la saisie :

```
********** Menu **********
a : Fixer la valeur de a
b : Afficher la valeur de a
q : Quitter le programme
**************************
Option choisie ?
```

6. Exercices optionnels :

a) Copier le programme `scanf1.c` dans le fichier `scanf12.c` et modifier le `%d` du `scanf()` par `%i`. Re-tester les mêmes valeurs qu'à la question 3a et en déduire les principales différences entre `%d` et `%i` pour la fonction scanf().

b) Quelle(s) différence(s) il y a-t-il entre %x et %X dans un printf ?

c) On peut dans un printf, suivant les formats, spécifier des options supplémentaires entre le % et le type. Tester par exemple les formats %8d puis %08d puis %+8d et enfin %-8d.

d) Dans le cas des nombres réels également, on peut spécifier des options. Tester par exemple, les formats %14g, %014g, %+14g et enfin %-14g.

e) Tester le format %.2f sur la variable x pour indiquer le nombre de chiffres après la virgule.