Développement C **L2 CUPGE 2018–2019**

Travaux pratiques EntréesSorties

Dans ce thème, nous verrons comment un programme écrit en C interagit avec son environnement, que ce soit un terminal (la fenêtre dans laquelle il est exécuté) ou des fichiers.

I Exercice : *Premier programme*

On va travailler sur les types, donc dans le répertoire ~/workspace/programmation-en-C-CUPGE/src/Entrées-Sorties/.

(a) Vérifiez que vous êtes bien sur la branche `master`, et si ce n'est pas le cas, allez sur cette branche.

(b) Créez le fichier exercice1.c dans le répertoire ~/workspace/programmation-en-C-CUPGE/src/Entrées-Sorties/ et ouvrez-le.

ATTENTION !!! Il ne faut jamais faire du copier-coller d'un fichier pdf vers un fichier texte : même si les caractères affichés se ressemblent, ils sont différents.

(c) Tapez le code suivant dans le fichier :

```
#include <stdio.h>

int main ( int argc , char * argv[] )
{
  return 0;
}
```

(d) Sauvegardez le fichier (avec Ctrl-S).

On va maintenant compiler et exécuter ce programme.

(e) Changer de répertoire pour aller dans ~/workspace/src/Entrées-Sorties.

(f) Compiler le programme avec la commande :

(g) Exécutez le programme. Que se passe-t'il ?

(h) Ajoutez la ligne suivante juste au dessus de `return 0` :

```
printf ( "Bonjour." ) ;
```

et compilez et exécutez le programme.

`printf` est une *fonction* qui permet à un programme d'envoyer des données au terminal dans lequel il est exécuté.

(i) Ajoutez `\n` après Bonjour, compilez et exécutez le programme.

`\n` est un caractère qui demande au terminal de passer à la ligne suivante.

II Exercice : *Affichage d'un entier.*

Le 'f' de `printf` signifie *mettre en forme*. Cette fonction est un peu spéciale car contrairement aux fonctions normales en C, elle peut prendre un nombre variable d'arguments, mais il en faut au moins 1.

Le premier argument est une *chaîne de formattage* qui sert à mettre en forme les données qui sont contenues dans les autres arguments, dans l'ordre de lecture. Pour celà, la chaîne de formattage contient des *directives d'affichage*. La première directive sera utilisée pour afficher la première donnée après la chaîne de formattage, la seconde directive sera utilisée pour afficher la seconde donnée après la chaîne de formattage, etc.

Directives d'affichage. Une directive commence toujours par %. Il y en a des dizaines, donc on va juste en regarder quelques unes :

- `%d` : permet d'afficher un entier en base `dix` ;
- `%c` : permet d'afficher une lettre (un caractère) à partir de son encodage en machine ;
- `%f` : permet d'afficher un nombre à virgule ;
- `%s` : permet d'afficher une chaîne de caractères ;

Pour l'exercice 2, on commence par copier le contenu du programme `exercice1.c` dans un nouveau fichier `exercice2.c`.

(a) Modifiez le premier programme pour afficher sur 10 lignes chaque caractère de '0' à '9' suivie de la valeur entière de l'encodage machine du caractère. Que remarque-t'on ?

(b) Faites de même pour les lettres `'a'` et `'z'`, et affichez leur différence `'z' - 'a'`. Idem pour les majuscules.

III Exercice : *Classes de caractères*

Pré-requis : Thème Variables, Constantes, et Types en C, exercice 1.

Comme les caractères de même nature (lettres minuscules, chiffres, etc.) se suivent, on va écrire un programme qui va lire un caractère, et va afficher sa nature (lorsqu'on la connaît). Par exemple, l'encodage du caractère '5' est bien compris entre l'encodage du caractère '0' et l'encodage du caractère '9'.

IV Exercice : *Chaînes de caractères*

Pré-requis : Thème Variables, Constantes, et Types en C, exercice 3.

Contrairement à Python, C fait la différence entre un caractère seul et une suite de caractères. Les valeurs de type caractère (`char`) s'écrivent entre des apostrophes, comme `'a'`, `'3'`, `'\n'`. Les *chaînes de caractères* sont juste des tableaux de caractères. Pour indiquer la fin de la chaîne, on met dans le tableau l'entier 0, qu'on peut aussi écrire `'\0'`. Il s'agit d'une convention qui est utilisée pour pouvoir traîter plus facilement les chaînes de caractères :

- Il n'y a pas de type spécifique : une chaîne de caractères est définie par l'adresse du premier caractère, donc on utilise `char *` ;
- Pour afficher une chaîne de caractères avec `printf`, on utilise la directive d'affichage `%s` :

Exemple 1 : Les deux morceaux suivants de programme font la même chose :

```
printf ( "Bonjour.\n" ) ;

char * hello = "Bonjour" ; /* Création de la chaîne de caractères
printf ( "%s.\n" , hello ) ;
```

(a) Écrivez dans le fichier `exercice4.c` un programme qui affiche les 3 premiers éléments du tableau de chaînes de caractères passé en paramètre du main `argv`. Compilez et exécutez ce programme avec *et sans* arguments sur la ligne de commande. Exemple ci-dessous avec 4 arguments (nombre récupéré grâce à argc) qui sont 4 chaînes de caractères qui sont le nom de l'exécutable suivi de 3 arguments sans signification particulière :

```
exercice4 toto 12 argument3
```

Que remarque-t'on ?

(b) Dans le terminal, tapez la commande :

```
echo $LANG
```

Que remarquez vous ?

(c) On va maintenant tenter de voir ce qu'il y a plus loin. Affichez la case 80 du tableau `argv`.

V Exercice : *Lecture d'entrées*

Même s'il est possible de lire tout ce que tape l'utilisateur avec la fonction `getchar` pour ensuite traduire ce qui est marqué, on utilise en général la fonction `scanf` qui permet de lire des entrées structurées en utilisant des directives.

(a) Écrivez le programme suivant :

```
#include <stdio.h>

int
main ( int argc , char * argv[] )
{
  printf ( "Écrivez : Bonjour , monde ! \n" ) ;
  scanf ( "Bonjour , monde !" ) ;
  printf ( "Je vous ai compris ! \n" ) ;
  return 0 ;
}
```

Et essayez de vérifier que `scanf` a bien lu ce que vous tapez.

(b) Pour faire la différence entre une lecture réussie et une lecture qui a échoué, lisez et affichez tous les caractères que `scanf` n'a pas réussi à lire jusqu'au caractère de fin de ligne (`'\n'`) en utilisant la fonction `getchar`.

(c) L'intérêt de `scanf` est de pouvoir lire des données avec les types existants. Au contraire de `printf`, il ne faut pas donner en argument une **valeur** mais l'adresse à laquelle il faudra stocker la valeur qui a été lue. Modifiez le programme précédent pour lire une chaîne de la forme

"Bonjour, je suis XXX", où XXX est un entier. Le programme devra répondre par "Bonjour, XXX".