
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 **src/Entrées-Sorties/**.

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

(b) Créez le fichier **~/workspace/src/Entrées-Sorties/exercice\arabic{exercicenum}.c** et ouvrez-le.

(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 formatage* qui sert à mettre en forme les données qui sont contenues dans les autres arguments, dans l'ordre de lecture. Pour cela, la chaîne de formatage contient des *directives d'affichage*. La première directive sera utilisée pour afficher la première donnée après la chaîne de formatage, la seconde directive sera utilisée pour afficher la seconde donnée après la chaîne de formatage, 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 d'un entier ;
- % 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 lettre de 0 à 9 suivi de sa valeur en tant qu'entier. 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 type (lettres minuscules, chiffres, etc.) se suivent, on va écrire un programme qui va lire un caractère, et va afficher son type (lorsqu'on le connaît).

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 traiter 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" ;
printf ( "%s.\n" , hello ) ;
```

(a) Écrivez dans le fichier **exercice4.c** un programme qui affiche les 3 premiers éléments du tableau **argv**. Compilez et exécutez ce programme avec *et sans* arguments. Que remarque-t'on ?

(b) Dans le terminal, tapez la commande :

```
echo $LANGUAGE_PID_FILE
```

Que remarquez vous ?

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