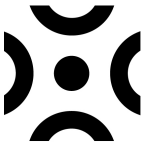
Perfectionnement à la programmation en C

Fiche de TP 2

L2 Informatique - 2024–2025

Partir sur de bonnes bases



Dans ce TP en **une séance**, nous révisons l'utilisation de la bibliothèque graphique en mode texte ncurses, dont l'introduction a été faite lors du TP précédent. Nous révisons quelques notions vues au semestre précédent et posons aussi quelques bonnes habitudes à prendre en programmation pour le reste de ces TP. Nous considérons ici en particulier les notions de

- prise en main de la documentation `man` ;
- écriture d'applications interactives sous ncurses ;
- gestion des arguments d'un programme ;
- utilisation et approfondissement de l'opérateur modulo .

Exercice 1 (Documentation `man`)

La *section 3* du manuel de l'utilisateur de Linux recense la documentation de la plupart des fonctions proposées par la bibliothèque standard. Il faut donc la consulter systématiquement pour connaître précisément le rôle d'une fonction. Par exemple, pour afficher la documentation de la fonction `putchar`, on saisit la commande

```
man 3 putchar
```

sur le terminal. Beaucoup d'informations sont alors affichées, comprenant la description des paramètres de la fonction et de sa valeur de retour.

1. Rechercher la documentation de la fonction `strcmp` et expliquer, en trois lignes maximum, ce qu'elle fait.

2. Écrire un programme qui illustre de manière exhaustive le rôle de `strcmp`. Ce programme ne doit pas réécrire la fonction. Il doit seulement l'utiliser de plusieurs manières différentes de sorte à illustrer le comportement de la fonction sur divers exemples parlants.

3. Rechercher la documentation des fonctions `strchr` et `strrchr` et expliquer, en trois lignes maximum, ce qu'elles font.

4. De la même façon que ce qui a été fait dans la question 2, écrire un programme qui illustre de manière exhaustive les rôles de `strchr` et de `strrchr`.

5. Rechercher la documentation de la fonction `atoi` et expliquer, en deux lignes maximum, ce qu'elle fait.

6. De la même façon que ce qui a été fait dans les questions 2 et 4, écrire un programme qui illustre de manière exhaustive le rôle de `atoi`.

Exercice 2 (Compléments sur ncurses)

1. Créer un programme `Rectangle.c` qui affiche un rectangle rouge de hauteur 3 et de largeur 15 dans une fenêtre. Le rectangle doit être au centre de la fenêtre.

   Indication : garder la page de documentation de la bibliothèque en permanence ouverte au cours les séances de TP est une bonne idée.

2. Créer un programme `CarreClic.c` qui affiche un carré rouge de côté 7 centré dans une fenêtre. Lorsque l'utilisateur clique sur le rectangle, il change de couleur et devient bleu. Lorsque il est bleu et que le l'utilisateur clique dessus, il redevient rouge et ainsi de suite.

3. Créer un programme `Rebond.c` qui affiche un petit carré bleu au centre d'un grand rectangle blanc. Lorsque l'utilisateur appuie sur la touche `Entrée`, le petit carré se déplace progressivement en direction nord-est. Lorsqu'il atteint le bord défini par le grand rectangle, celui-ci rebondit. Le petit carré continue à bouger tant que l'utilisateur n'appuie à nouveau sur `Entrée`, provoquant la fin de l'exécution du programme. Le petit carré se déplace à une vitesse de 2 cases par seconde.

4. Créer un programme `Menu.c` qui affiche un menu de la forme

   ```
   1 - Demarrer
   2 - Options
   3 - Credits
   4 - Quitter
   ```

   Initialement, la première ligne du menu doit s'afficher en surbrillance. Les flèches directionnelles servent à faire déplacer la ligne en surbrillance. L'utilisateur doit pouvoir choisir une partie soit en appuyant sur Entrée (dans ce cas la ligne en surbrillance est celle choisie), soit en saisissant en appuyant sur les touches `1`, `2`, `3` ou `4` (dans ce cas la ligne correspondant au numéro est choisie).

   Le comportement du programme est le suivant : le choix `1` lance ce qui a été fait dans la question 3 ; le choix `2` permet de spécifier la vitesse de déplacement du petit carré ; le choix `3` affiche les auteurs du programme, leur affiliation et la date d'édition du programme ; le choix `4` provoque la fin d'exécution du programme.

Notes importantes : pour compiler un programme `Prog.c` utilisant la bibliothèque ncurses, on utilise la commande

```
gcc -Wall -std=c17 -o Prgm Prgm.c -lncurses
```

Compiler pour le moment uniquement en utilisant cette commande.

Nous utiliserons des fichiers `Makefile`, adaptés dans le cadre de la conception de projets à plusieurs fichiers, pour les prochains TP.

Exercice 3 (Arguments d'un programme)

Un programme C peut être exécuté comme une commande habituelle et peut donc recevoir des arguments. Un argument est une chaîne de caractères. Pour profiter de cette fonctionnalité, la fonction `main` doit être écrite en respectant le prototype suivant :

```c
int main(int argc, char *argv[]);
```

La variable `argc` contient le nombre d'arguments avec lesquels le programme vient d'être lancé et la variable `argv` est un tableau de chaînes de caractères, indicé de 0 à `argc - 1`, qui contient les arguments dans l'ordre de leur saisie. Il est a noter que le nom du programme est lui-même considéré comme un argument et occupe de ce fait invariablement la première position du tableau `argv`.

Par exemple, supposons que l'on souhaite lancer un programme nommé `Prog` avec les arguments `arg1` et `autreArg`. Pour cela, nous saisirons la commande `./Prog arg1 autreArg`, qui a pour effet d'affecter à la variable `argc` la valeur 3 et le tableau `argv` va vérifier `argv[0] = "Prog"`, `argv[1] = "arg1"` et `argv[2] = "autreArg"`.

1. Écrire un programme `LectureArg.c` qui affiche sur la sortie standard le nombre d'arguments avec lesquels il est lancé.

2. Écrire un programme `Calc.c` qui prend trois arguments : un opérateur parmi `+`, `-`, `x` et `/` et deux entiers. Le programme affiche ensuite le résultat de l'opération spécifiée par le 1 argument entre ses 2 et 3 arguments. Il est important de capturer ici tous les cas d'erreur et de renseigner l'utilisateur lorsque ceux-ci surviennent.

3. Écrire un programme `Triangle.c` qui prend deux arguments : un entier positif `n` et un caractère `c`. Le programme affiche dans une fenêtre ncurses un triangle fait de caractères `c`, de base `2n + 1`, de hauteur `n + 1` et pointant vers le bas. Par exemple, pour `n = 3` et `c = '*'`, la fenêtre doit contenir

```
*******
 *****
  ***
   *
```

Exercice 4 (Cryptage de César)

Le but de cet exercice est de programmer un système de cryptage élémentaire basé sur le cryptage dit de *César*. Il consiste, étant donné un texte *en clair* (c.-à-d. non crypté), et une constante $k \in \mathbb{Z}$ appelée *clé de cryptage*, à décaler circulairement chacune des lettres du texte de k positions par rapport à sa place dans l'alphabet. Par exemple, pour le texte en clair

`Les TP de programmation sont super.`

et la clé $k = 3$, on obtient le texte *crypté* suivant :

`Ohv WS gh surjudppdwlrq vrqw vxshu.`

Les caractères non alphabétiques ne sont pas modifiés (comme le point d'exclamation dans l'exemple donné). On ne considère que des textes contenant des caractères ASCII (et donc pas de lettres accentuées).

La formule qui permet de calculer, étant donné une lettre alphabétique en clair l, la lettre cryptée c qui lui correspond est

$$c := (l + k) \pmod{26}$$

Réciproquement, pour décrypter un texte crypté, il suffit de décrypter chacune des lettres alphabétiques qu'il contient. Étant donné une lettre cryptée c, le calcul de la lettre en clair ℓ qui lui correspond s'effectue par la formule

$$l := (c - k) \pmod{26}$$

1. Déterminer ce en quoi est évaluée l'expression `-9 % 5` et en quoi cela est gênant pour la suite de l'exercice.

   Indication : regarder la seconde formule et s'interroger sur quand ℓ définit bien un caractère.

2. Programmer une fonction `int bon_modulo(int a, int b)` qui calcule le reste de la division euclidienne de `a` par `b` (la valeur de retour doit être toujours positive). On appelle cette opération le *bon modulo*. L'algorithme de calcul du bon modulo est simple : quand `a % b` est positif, cette valeur doit être renvoyée. Sinon, `a % b` est strictement négatif et il faut renvoyer cette valeur incrémentée de `b`.

3. Écrire un programme `Crypt.c`, qui accepte comme arguments une valeur numérique (la clé) et une suite de mots (la phrase à crypter). Ce programme affiche sur la sortie standard la suite de mots cryptée selon la clé en suivant la première formule.

4. Écrire un autre programme `Decrypt.c` qui réalise le décryptage. Il prend comme arguments une valeur numérique (la clé) et une suite de mots (la phrase à décrypter). Ce programme affiche sur la sortie standard la suite de mots décryptée selon la clé en suivant la seconde formule.

5. Tester les deux programmes précédents, en particulier en vérifiant que la phrase obtenue en cryptant puis en décryptant successivement une phrase avec une même clé est bien égale à celle de départ.