TP 4 : Langage C. Découverte et manipulation

Exercice 1 – Programmation en C

Rappelons le premier exemple de programme du cours :

```c
// mon premier programme C
#include <stdio.h>
int main(void) {
  printf("Hello World!\n");
  return 0;
}
```

▶ **Question 1**
Recopier ce programme dans un fichier d'extension `choisir_un_nom.c`, puis lancer la commande suivante depuis le même dossier :

```
$ gcc votre_fichier.c -o votre_executable
```

Le fichier `votre_fichier.c` contient votre code : la commande ci-dessus l'a utilisé pour produire un fichier `votre_fichier` exécutable par l'ordinateur avec la commande `./votre_fichier` :

▶ **Question 2** Exécuter `votre_fichier` et admirez le résultat (10 secondes maximum).

▶ **Question 3** Enlevez le point virgule ligne 5, et réessayez de compiler. Qu'observe-t-on ? Même chose en commentant la ligne 2.

▶ **Question 4** Avant la fonction `main`, mais après la directive `#include`, écrire une fonction de prototype `int absolue(int x)` qui renvoie la valeur absolue d'un entier.
Pour commencer, on testera les fonctions que l'on écrit en imprimant son résultat avec `printf`.

▶ **Question 5** Écrire des fonctions permettant d'afficher les figures suivantes :

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

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

▶ **Question 6** Écrire la fonction d'exponentiation rapide de prototype `double expo_rapide (double x, int n)`.

▶ **Question 7** Écrire une fonction de prototype `bool est_premier(int n)` qui renvoie `true` si l'entrée est un nombre premier, `false` sinon. *Pour utiliser les booléens en C, on utilisera la directive* `#include <stdbool.h>`.

▶ **Question 8** Écrire une fonction de prototype `void affiche_diviseurs(int n)` qui affiche tous les diviseurs entiers strictement positifs de n.

▶ **Question 9** Écrire une fonction `void separateur_exercice(int exo_suivant)` qui sépare l'affichage de tests de deux exercices. *Par exemple, sauter une ligne, faire une ligne de la forme* `=====` *avec le numéro de l'exercice à l'intérieur, puis re-sauter une ligne.*

Exercice 2 – Manipulation de tableaux

Dans cet exercice, on utilisera la bibliothèque `stdlib.h`. Juste après les directives, on va définir un tableau global :

```c
#define TAILLE_TABLEAU 25
int tab[TAILLE_TABLEAU]; // tableau d'entiers non initialisés
```

Ce tableau est donc utilisable dans toutes les fonctions suivantes, sans avoir besoin de le mettre en argument. Pour accéder à une case de tableau ou la modifier, on utilise la syntaxe `tab[k]`.

▶ **Question 1** Écrire une fonction de prototype `void initialiser(void)` qui remplit le tableau `tab` de zéros.

▶ **Question 2** Écrire une fonction `void affiche(void)` qui affiche le contenu du tableau `tab`.
On utilisera cette fonction dans les questions suivantes pour tester sommairement que vos réponses sont correctes.

▶ **Question 3** En utilisant la fonction `int rand(void)` (définie dans la bibliothèque `stdlib`, elle renvoie un entier choisi "uniformément" entre `0` et `MAX_RAND`), écrire une fonction initialisant un tableau avec des valeurs aléatoires.

▶ **Question 4** Écrire une fonction `int indice_min(int debut, int fin)` renvoyant le premier indice où l'on trouve un minimum de `tab` dans la tranche $tab[debut : fin]$.

▶ **Question 5** En déduire une implémentation du *tri par sélection*, dont la stratégie est la suivante :
- chercher le minimum du tableau entier,
- échanger sa valeur avec celle de la case numéro 0,
- trier le tableau à partir à partir de la case 1

On utilisera une boucle plutôt que des appels récursifs. On pourra écrire une fonction auxiliaire `void swap(int x, int y)` *qui échange les valeurs aux indices x et y dans le tableau* `tab`.

▶ **Question 6** Montrer la terminaison et la correction de votre algorithme. Combien de comparaisons sont exécutées par le tri par sélection, dans le pire et dans le meilleur des cas, pour un tableau de taille n ?
Les tableaux que l'on manipule dans cet exercice sont des tableaux *statiques* : leur taille est une constante littérale. On s'attend quand même à pouvoir faire plus : par exemple, déterminer dynamiquement la taille du tableau à allouer, renvoyer un tableau en sortie d'une fonction, etc. On verra comment manipuler ces tableaux dits *de taille dynamique* la semaine prochaine.

Exercice 3 – Arguments de la ligne de commandes

Pour l'instant, votre utilisation de la ligne de commandes se limite à exécuter alternativement les commandes suivantes :

Shell
```
$ gcc votre_fichier.c -o votre_executable
$ ./votre_executable
```

On pourra utiliser un nouveau fichier pour cet exercice. De manière générale, puisqu'on peut avoir envie de plusieurs exécutables pour tester plusieurs exercices / tâches différents, il ne faut pas hésiter à prendre dix secondes pour écrire un nouveau fichier `.c` si le contexte s'y prête.
La première commande utilise des *arguments*, comme le nom du fichier ou l'option `-o` et son exécution dépend de ces arguments. Notamment, on utilisera beaucoup les différentes options de `gcc` comme `-Wall` ou `-fsanitize=...`. Il est possible de récupérer ces valeurs en adaptant la signature de notre fonction `main` :

C
```c
int main(int argc, char* argv[]) { ... }
```

`argc` est le nombre d'arguments donné dans la commande, et `argv` est la valeur de ces arguments, donnés dans un tableau de chaînes de caractères. En OCaml, le même résultat peut être obtenu par les valeurs `Sys.argc` et `Sys.argv`.

▶ **Question 1** Afficher la valeur de `argv[0]`.

▶ **Question 2** En adaptant le code de la fonction `affiche`, écrire une fonction `void affiche_arguments(int argc, char* argv[])` qui affiche tous les arguments donnés à la fonction.
Pour les questions suivantes, on pourra utiliser les fonctions de la bibliothèque `stdlib.h` suivantes : `int atoi(const char* theString)` qui lit une chaîne de caractère représentant un entier et renvoie l'entier, et `double atof(const char* theString)` qui fait la même chose pour un flottant.

▶ **Question 3** Écrire un programme qui prend en arguments un flottant x, un entier n, et affiche la valeur x^n. *S'il n'y a pas assez d'arguments, on affichera un message précisant l'usage de votre programme. En guise d'exemple, vous pourrez lancer la commande* `tldr` *sans arguments et voir ce qu'il propose.*

▶ **Question 4** Que se passe-t-il si l'on utilise la fonction `atoi` ou `atof` avec des arguments qui ne représentent pas un entier / un flottant ?
Il est aussi possible d'utiliser des fonctions `strtoi`, `strtod` qui lèvent des exceptions si leur entrée n'est pas un entier / flottant correctement écrit.