TP Soutien IPI

Dimitri Watel

Faites les exercices des sections suivantes dans l'ordre. Pour chaque exercice, créez un nouveau fichier. Votre fichier doit compiler sans erreur ni warning avec la commande
`gcc -Wall -Wextra -ansi -lm`.
Testez chaque exercice dans le `main`. Si vous réussissez sans difficulté à faire 2 exos d'une section, vous pouvez, si vous le souhaitez, passer à la section suivante.

1 Bases

Quelques exercices simples.

Exercice 1 — *Affichage basique*

Créer un programme qui affiche `Hello world!` en console.

Exercice 2 — *Inverse d'un entier*

Créer un programme qui affiche un entier $\texttt{a} > 1$ de votre choix et le flottant $\texttt{a}^{-1}$.

2 Boucles

Quelques exercices d'utilisation des boucles. N'utilisez une boucle `while` si la boucle `for` est plus adaptée.

Exercice 3 — *Somme des entiers de 1 à n*

Créer un programme qui affiche un entier $n > 10$ de votre choix et la somme des entiers de 1 à n.

Exercice 4 — *Puissance de 3*

Créer un programme qui affiche l'entier i tel que $i^3 \leq 1548 < (i+1)^3$.

Exercice 5 — *`int` et `long`*

Créer un programme qui affiche

- l'entier 2^i tel que $2^i \leq MI < 2^{i+1}$ où MI est la plus grande valeur qu'une variable de type `int` puisse avoir.
- l'entier 2^j tel que $2^j \leq ML < 2^{j+1}$ où ML est la plus grande valeur qu'une variable de type `long` puisse avoir.

Exercice 6 — *Rectangle*

Créer un programme qui, pour un entier $n > 10$ de votre choix, affiche en console des caractères `*` organisés sous forme d'un carré de largeur et longueur n. Par exemple pour $n = 4$:

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

3 Fonctions

Pour les fonctions mathématiques, n'oubliez pas `#include <math.h>` et l'option `-lm` pour la compilation.

Exercice 7 — *Formule simple*

Créer une fonction qui, connaissant un entier n, renvoie $n\log(n)$.

Exercice 8 — *Triangle*

Créer une fonction qui, connaissant un entier n, affiche en console des caractères `*` organisés sous forme d'un triangle isocèle de hauteur n, de base de taille $2n-1$ et pointant vers le bas. Par exemple pour $n=4$, le programme dervait afficher

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

Exercice 9 — *Gravité*

Créer une fonction qui, connaissant trois **entiers** $m1$, $m2$ et d, affiche la force de gravité entre deux corps de masses $m1$ et $m2$ éloignés d'une distance d égale à $G{\cdot}m1{\cdot}m2/d^2$, avec $G = 6.67{\cdot}10^{-11}$. Définissez G avec le mots-clef `#define`.

Exercice 10 — *Triangle*

Créer une fonction qui, connaissant un entier n, affiche en console des caractères `*` organisés sous forme d'un cerle de rayon n (affiche toutes les `*` dont la distance en norme 2 au centre est inférieure ou égale à 4). Par exemple pour $n=4$, le programme dervait afficher

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

4 Structure

Exercice 11 — *Complexe*

Créer une structure `struct complexe` avec 2 entiers `p` et `q` représentant le complexe `p + i · q`.

- Créer une fonction qui affiche un nombre complexe.
- Créer une fonction qui additionne deux nombres complexes.
- Créer une fonction qui calcule l'opposé d'un nombre complexe.
- Créer une fonction qui calcule le conjugué d'un nombre complexe.
- Créer une fonction qui calcule le produit de deux nombres complexes.
- Créer une fonction qui calcule le module d'un nombre complexe.
- Créer une fonction qui calcule l'argument d'un nombre complexe.
- Créer une fonction qui, connaissant le module et l'argument, renvoie le nombre complexe associé.

5 Pointeur

Manipulation basique des pointeurs.

Exercice 12

Créer une fonction qui prend en entrée 3 entiers a, b et c. À l'issue de la fonction,

- b a la valeur que a avait au début de la fonction
- c a la valeur que b avait au début de la fonction
- a a la valeur que b avait au début de la fonction

Exercice 13

Créer une fonction qui crée et renvoie un pointeur sur un entier égal à 0, puis une autre fonction qui prend un pointeur en entrée et libère la mémoire associée à ce pointeur.

Exercice 14

Créer une fonction qui prend en entrée un entier n et crée et renvoie un pointeur p sur n flottant et initialise `p[i]` avec $\log(i)$.

Exercice 15

Créer une fonction qui prend en entrée un entier n et un pointeur l sur n entiers. Renvoyez en sortie un pointeur sur pointeur d'entiers p représentant deux listes d'entiers. La première liste contient le nombre d'entiers pairs pointés par l, suivi de la liste de ces entiers. La seconde liste contient le nombre d'entiers impairs pointés par l, suivi de la liste de ces entiers.

6 Tableaux

Dans ces exercices, utilisez des tableaux et non pas des pointeurs.

Exercice 16 — *Appartenance*

Créer une fonction qui, connaissant un entier n, un tableaux de n entiers et un entier m, renvoie 1 si m appartient au tableau n.

Exercice 17 — *Dichotomie*

Créer une fonction qui, connaissant un entier n, un tableaux de n entiers triés et un entier m, effectue une recherche dichotomique de m dans le tableau.

Exercice 18 — *L'école, version IPI*

Les membres de l'école peuvent être soit des étudiants, soit des administratifs, soit des enseignants-chercheurs.

Un étudiant est caractérisé par son nom et son numéro d'inscription.

Un administratif est caractérisé par son nom et sa catégorie administrative (A, B ou C).

Un enseignant-chercheur est caractérisé par son nom et le nom du laboratoire de recherche auquel il est rattaché.

- En définissant éventuellement des types intermédiaires, définissez un type `personne` pour représenter un membre de l'école.
- Écrivez une fonction `listing` qui prend un tableau de personnes en argument et sa taille et rend la liste des noms de tous les membres de l'école.
- Écrivez une fonction `separe` qui prend un tableau de personnes et sa taille en argument et qui affiche d'une part la liste des noms des étudiants, et d'autre part la liste des noms des administratifs et des enseignants-chercheurs.
- Écrivez une fonction `effectifs` qui prend un tableau de personnes en argument et qui calcule le nombre d'administratifs pour chacune des trois catégories.

7 Un peu de tout

Exercice 19 — *Jeu de mémo*

Le jeu de mémo consiste à retrouver les paires de tuiles identiques dans une grille carrée 2D.

Initialement, les tuiles sont retournées faces cachées. À son tour, le joueur choisit deux tuiles faces cachées, consulte leurs valeurs et:

- soit il s'agit d'une paire et le joueur marque un point, replace les tuiles faces visibles et rejoue;
- soit il ne s'agit pas d'une paire, auquel cas les tuiles sont replacées faces cachées et on passe au joueur suivant.

Le jeu s'arrête lorsque toutes les paires ont été découvertes, et le joueur ayant le plus de points remporte la partie.

Ici, les valeurs des tuiles seront représentées par des chaînes de caractères. Pour modéliser le jeu, nous allons utiliser deux grilles:

- une grille `tuiles` contenant les valeurs de chaque tuile
- une grille `jeu` où chaque aura pour valeur `"*"` si la tuile correspondante est face cachée, et la valeur de la tuile si celle-ci est face visible.

Ces grilles seront de taille `SIZE` $\times$ `SIZE` où `SIZE` est une constante (variable globale).

- Écrire une fonction d'initialisation pour la grille `jeu`.
- Écrire une fonction d'initialisation pour la grille `tuiles`.

  On demandera à l'utilisateur de fournir $\texttt{SIZE}^2/2$ chaînes de caractères. Pour chaque chaîne s, on placera aléatoirement deux tuiles de valeur s dans deux emplacements vides.
- Écrire une fonction qui demande deux cases à l'utilisateur et affiche (si possible) les valeurs des tuiles correspondantes.

  Si les entrées sont valides et correspondent à des tuiles faces cachées, la fonction retournera 0. Sinon, la valeur de retour sera 1.
- Écrire une fonction simulant un tour de jeu. Cette fonction renverra 0 si le joueur courant marque un point et doit rejouer, et 1 sinon.
- Écrire une fonction permettant de simuler une partie entière à deux joueurs.