Examen final - LO01

A2015

2 H - Sans documents (sauf diagrammes de C) ni calculette

N'oubliez pas de commenter vos programmes.

Vous aurez besoin de 4 copies : une par exercice

1 . Fonctions et structures de données en C (5 points)

On voudrait utiliser dans un programme la notion d'horaire comme déjà vu en TD. Un horaire comprend 3 informations : les heures, les minutes et les secondes. 3 étudiants ont proposé une solution différente en langage C pour représenter un horaire :

Etudiant 1 :
```
int h;  /*heures*/
int m;  /*minutes*/
int s;  /*secondes*/
```

Etudiant 2 :
```
int S[3] ;
    /*S[0] heures*/
    /*S[1] minutes*/
    /*S[2] secondes*/
```

Etudiant 3 :
```
typedef struct {
  int h; /*heures*/
  int m; /*minutes*/
  int s; /*secondes*/
  } unHoraire;

unHoraire H;
```

1.1 - Ecrire **pour les 3 étudiants** la fonction *initUnHoraire()* qui initialise un horaire à partir d'un dialogue utilisateur. On vérifiera que l'horaire est correctement entré (les secondes et minutes comprises entre 0 et 59 et les heures compris entre 0 et 23).
1.2 - Ecrire **pour l'étudiant 2** la fonction *secToHoraire()* qui transforme en horaire un nombre de secondes fourni en argument. Par exemple 3728 (secondes) sera transformé en 1h 2m 8s. Cet horaire sera un résultat de la fonction.
1.3 - Ecrire **pour l'étudiant 3** la fonction *horaireToSeconde()* qui retourne le nombre de secondes correspondant à un horaire donné. Par exemple, pour l'horaire 2h 48m 10s la fonction retourne 10090 (secondes).
1.4 - Ecrire **pour l'étudiant 3** la fonction *ajouteUneSeconde()* qui modifie l'horaire passé en paramètre en lui ajoutant une seconde.

2. La COP 21 (5 points) Nouvelle copie

On voudrait écrire un programme permettant de gérer les personnes des différentes délégations participant à la COP 21 à Paris.

Pour cela, vous devez :

2.1 - Créer une structure *unePersonne*, comprenant les informations suivantes :
- nom de la personne (40 caractères max),
- prénom de la personne (40 caractères max),
- nom de l'hôtel (40 caractères max),

Créer une structure *uneDelegation*, comprenant les informations suivantes :
- nom du pays de la personne (40 caractères max),
- nombre de personnes de la délégation,
- tableau comprenant toutes les personnes de la délégation (200 personnes max).

2.2 – Ecrire une fonction *initUneDelegation()* permettant de remplir les informations concernant une délégation, qui sera passée en paramètre, à partir d'un dialogue avec l'utilisateur.

2.3 – Ecrire une fonction *ajouteUneDelegation()* qui ajoutera dans le fichier "delegations.dat", une délégation supplémentaire en fin de fichier. Cette fonction n'aura pas de paramètre et fera appel à la fonction précédente.

2.4 – Ecrire une fonction *recherchePersonne()* qui aura comme paramètre un nom de personne, recherchera dans le fichier *"delegation.dat"* si cette personne appartient à une délégation. Cette fonction retournera 1 si la personne fait partie d'une délégation et 0 sinon.

3. Jeu de la vie (5 points) Nouvelle copie

Le jeu de la vie fut inventé par John Horton Conway en 1970, décrivant le développement, le déclin et les altérations d'une colonie de micro-organismes. Le but de cet exercice est d'en programmer une version simplifiée.
Nous disposons d'un échiquier, dans lequel les cases peuvent prendre les valeurs 0 (cellule morte) ou 1 (cellule vivante). À chaque étape, on calcule la nouvelle valeur de chacune des cases en fonction des valeurs des cases voisines. Les cases voisines d'une case étant celles se trouvant de part et d'autre de cette case sur la même ligne ou sur la même colonne (pas en diagonale).

- La valeur d'une case passe à 0 si elle possède 0 ou 4 voisins à 1(elle meurt isolée ou étouffée).

- La valeur d'une case passe à 1 si elle possède 2 ou 3 voisins à 1 (elle renaît).
- Elle ne change pas dans les autres cas.

Pour éviter les cas particuliers des cases se trouvant en périphérie de l'échiquier, nous considérerons que ce dernier est stocké dans un tableau comprenant une bordure remplie de zéros (0) qui n'évoluent pas.

0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	0	1	1	0
0	0	1	1	0	0
0	0	0	0	0	0

Exemple : La figure ci-contre représente un échiquier 4x4 avec sa bordure.
La case à l'intersection de la 3ème ligne et de la 3ème colonne (grisée foncé) à deux voisins à 1 et deux voisins à 0

Le programme principal vous est donné et est le suivant :

```
#define TAILLE 50
void main()
{
       int Tab[TAILLE][TAILLE]; /* on déclare un tableau carré */
       int i, nbEtapes;

       configEchiquier(Tab);

       printf("Entrez le nombre d'etapes : ");
       scanf("%d", &nbEtapes);

       for (i=1; i<=nbEtapes; i++) {
         changeEtat(Tab);
         imprimeEchiquier(Tab);
       }
}
```

3.1- En utilisant une fonction *int binRandom()* qui retourne au hasard la valeur 0 ou la valeur 1. Nous souhaitons initialiser l'échiquier avec une valeur au hasard pour chacune des cases. Ecrire une fonction *configEchiquier(int Tab[][TAILLE])* qui prend en paramètre un tableau stockant un échiquier et l'initialise, c'est à dire non seulement initialiser l'échiquier mais aussi la bordure.

3.2- Nous souhaitons visualiser l'évolution du jeu. Ecrire une fonction *imprimeEchiquier(int Tab[][TAILLE])* qui prend en paramètre un tableau stockant un échiquier et affiche ce dernier à l'écran (sans la bordure et uniquement les données sans cadre).

3.3- Ecrire une fonction *nbVoisins1 (int Tab[][TAILLE], int ligne, int colonne)* qui permet d'obtenir le nombre de voisins à 1 pour une case de l'échiquier.

3.4- En utilisant la fonction précédente, écrire la fonction *evolutionCellule(int Tab[][TAILLE], int ligne, int colonne)* qui retourne la valeur que devra prendre une case à l'étape suivante.

3.5- Ecrire la fonction *changeEtat(int Tab[][TAILLE])* qui permet de faire évoluer l'ensemble de l'échiquier d'une étape vers une autre. Attention, on devra utiliser un deuxième tableau dans la fonction qui permettra de stocker temporairement le nouvel état.

---

4. Dans quelle base ? Objets (5 points) Nouvelle copie

En C, une littérale entière est un nombre en base décimale (10), octale (8) ou base hexadécimale (16). Au niveau de la syntaxe :

- Une littérale en base 10 ne commence pas par un 0 (hormis si le 0 est seul) et ne contient que des caractères parmi 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.
- Une littérale en base 8 commence par 0 contient au moins deux chiffres et ne comprend que des caractères parmi 0, 1, 2, 3, 4, 5, 6, 7.
- Une littérale en base 16 commence par 0x ou 0X et ne contient que des caractères parmi 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, A, B, C, D, E, F.

4.1 - Ecrire une fonction *base()* qui prendra en argument une chaîne de caractères contenant des caractères (chiffres, lettres ou autres). La fonction renverra 10, 8 ou 16 suivant que la chaîne saisie représente une littérale entière en base 10, 8 ou 16. La fonction renverra 0 s'il s'agit d'une donnée d'une autre nature. On pourra utiliser le fait que, dans la table ASCII, les chiffres se suivent dans l'ordre croissant, les minuscules (respectivement les majuscules) se suivent dans l'ordre lexicographique.

Soit la classe suivante :

```
class Litterale {
private :
       char nombre[10]; // chaîne de caractères représentant une littérale
};
```

4.2 - Quel est le terme utilisé en programmation orienté objet pour désigner la donnée membre `nombre` ?

4.3 - En utilisant la fonction *base()*, définir une méthode publique *getBase()*dans la classe *Litterale* qui permet de connaître la base d'un objet *Litterale*.