Automne 2019 **Examen final- LO01**

2 H - Sans documents (sauf diagrammes de C) et une feuille resto-verso avec notes manuscrites

La clarté de vos réponses sera prise en compte. Ne pas écrire au crayon.

Les types des paramètres dans les entêtes des fonctions sont volontairement omis. A vous de les compléter.

N'oubliez pas de commenter vos programmes

Vous aurez besoin de 4 copies : une par exercice

1. Tirage du loto (5 points) -> copie n°1

Dans un fichier structuré « loto2019.dat » sont stockés tous les résultats des tirages du loto de l'année 2019. Chaque enregistrement est composé d'une date, d'un tirage et de gains.

1.1 – Écrire la structure *uneDate* composée de 3 entiers représentant le jour, le mois et l'année. Écrire la structure *unTirage* composé de la date du tirage, d'un tableau de 5 entiers représentant les 5 numéros tirés et d'un tableau de 4 entiers contenant les gains pour 2, 3, 4 ou 5 bons numéros.
Exemple :

uneDate : 25 12 2019
tabTirage : 2 9 48 30 21
tabGains : 12 234 5937 359790

1.2 - Écrire la fonction *initUnTirage()* ayant un paramètre de type *unTirage* qui demande à un utilisateur de remplir les différents champs du tirage. On considère que l'utilisateur entrera des valeurs correctes, il n'est pas utile de les vérifier.

1.3 – Écrire la fonction *ajouteUnTirage()* sans paramètre qui initialise un tirage à l'aide de la fonction précédente et l'ajoute à la fin du fichier « loto2019.dat ».

1.4 – Écrire une fonction *calculeGainMoyen()* sans paramètre qui retourne le gain moyen de l'ensemble des tirages contenus dans le fichier « loto2019.dat ». Un gain étant la somme des 4 gains pour un tirage.

1.5 – Écrire le programme principal, qui ajoute un tirage dans le fichier « loto2019.dat », puis affiche le gain moyen pour l'année 2019.

2. Gestion d'un parc automobiles (5 points) -> copie n°2

On souhaite gérer le parc de voitures d'une agence de location. Une voiture est caractérisée par les éléments suivants :

modele : le modèle de la voiture (chaîne de 30 caractères),
immat : son n° d'immatriculation (chaîne de 10 caractères),
km : son kilométrage (entier),
etat : son état (disponible ou en cours de location entier 1 ou 0).

L'ensemble des voitures sera stocké dans un tableau de voiture (parc).

2.1 - Écrire une fonction *int immatValide (char *immat)* permettant de vérifier si une immatriculation *immat* est valide, de la forme AB-159-XZ (2 lettres suivies d'un tiret suivi d'un nombre suivi d'un tiret suivi de 2 lettres) . La fonction retourne 1 si elle est valide 0 sinon.

2.2 - Définir la structure *uneVoiture* et créer une fonction *void initVoiture()* qui aura comme paramètre une variable de type *uneVoiture* permettant d'initialiser les différents champs de la variable par un dialogue avec l'utilisateur (on devra vérifier la validité de l'immatriculation).

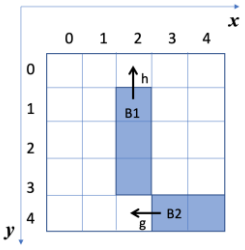
2.3 - Écrire une fonction *int changeEtatVoiture(int etat, char * matricule, voiture *parc, int nbV)* qui aura comme paramètre un *etat* à modifier pour une voiture d'immatriculation *matricule* dans le parc de voiture *parc* permettant de changer l'état d'une voiture par la valeur *etat* et retourne 1 si la voiture n'existe pas 0 sinon, *nbV* étant le nombre de voitures dans le parc.

2.4 - Écrire le programme principal qui demande à l'utilisateur le nombre de voitures *nbV* à rentrer dans le parc, créer le tableau *parc* (tableau dynamique), initialise les *nbV* voitures dans le tableau *parc* puis demande à l'utilisateur l'immatriculation d'une voiture et la loue.

3. Bateaux (5 points) -> copie n°3

Reprenons les bateaux de la bataille navale (vue en TD) mais cette fois-ci on veut positionner dans la grille des bateaux de taille variable comprise en 1 et 4 cases. La grille ne sera pas utilisée dans cet exercice. Un bateau est caractérisé par sa position de l'avant du bateau (ligne, colonne), la taille et la direction de navigation. La direction « gauche » ou « droite » implique un bateau placé horizontalement , « haut » ou « bas » implique un bateau vertical. La structure de donnée associée est la suivante ;

```
typedef struct {
        int nl ; /*ligne*/
        int nc ; /*colonne*/
        int lg ; /*taille*/
        char d ;  /*direction : g->gauche, d->droite, h->haut, b->bas */
}unBateau;
```



3.1 - Écrire la fonction *saisieUnBateau(B)* qui initialise un bateau à partir d'un dialogue utilisateur. On suppose que l'utilisateur rentre des informations correctes et conformes à la taille et la configuration de la grille.

3.2 - Soit la structure suivante :

```
typedef struct {
        int minX, maxX, minY, maxY ;/* coord de la boite : min max sur les axes X et Y */
} uneBoite;
```

Écrire la fonction *boiteEnglobante(B, box) qui* calcule dans *box* la boite englobante d'un bateau B.

Par exemple le bateau B1 en ligne 1 et colonne 2 de taille 3 et de direction 'h' a une boite englobante minX= 2, maxX =2, minY = 1, maxY = 3, le bateau B2 en ligne 4 et en colonne 3 de taille 2 et de direction 'g' a une boite englobante minX= 3, maxX =4, minY = 4, maxY = 4.

3.3 - Écrire la fonction *disjoints(B1,B2)* qui retourne 1 si les 2 bateaux B1 et B2 passés en paramètres ne se recouvrent pas du tout et 0 sinon. On utilisera leurs boites englobantes pour détecter facilement le recouvrement.

3.4 - Soit *jeu* un tableau de *N* bateaux placés pour un jeu (*N* étant une constante). À l'aide des fonctions précédentes, écrire la fonction *initUnJeu(jeu)* qui remplit le tableau. Pour chaque nouveau bateau saisi par l'utilisateur, on vérifiera qu'il ne recouvre pas les autres bateaux déjà rentrés. Si c'est le cas, on redemande un autre bateau.

4. Jeu du taquin (5 points) -> copie n°4

Le taquin est un jeu composé de 15 pièces mobiles numérotées de 1 à 15 pouvant être déplacées dans un cadre prévu pour 16 et de taille 4x4 (tableau à deux dimensions NBxNB, NB étant égale à 4). A chaque tour, le joueur a la possibilité de déplacer une pièce adjacente à la case vide. Le but du jeu est de remettre les 15 pièces dans l'ordre, la case vide figurant alors en dernière position.

13	3	7	15
4	12		8
9	6	1	14
2	5	10	11

Déplacements possibles dans cette configuration:
- 12 vers la droite;
- 7 vers le bas ;
- 8 vers la gauche;
- 1 vers le haut.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Situation finale (jeu résolu)

4.1 - Écrire la fonction *afficheGrille(int grille[][NB])* qui affichera le contenu de la grille à l'écran.

4.2 - Écrire la fonction *deplacePiece(grille, caseVideX, caseVideY)* qui demandera à l'utilisateur un déplacement à effectuer (sous forme de caractère):

- H : déplace la pièce située en haut de la case vide ;
- B : déplace la pièce située en bas ;
- D : déplace la pièce située à droite ;
- G : déplace la pièce située à gauche.

La position actuelle de la case vide est indiquée par les paramètres *caseVideX* et *caseVideY* et sera mise à jour par cette fonction.
La fonction retournera 1 si le déplacement est possible, 0 sinon (par exemple si l'utilisateur entre 'D' alors que la case vide est située sur le bord droit de la grille).

4.3 - On supposera connues les fonctions *void initGrille(int grille[][NB], int *caseVideX, int *caseVideY)* qui initialisera aléatoirement la grille et renverra les coordonnées initiales de la case vide dans *caseVideX* et *caseVideY* et *int verifGrille(int grille[][NB])* qui retournera 1 si toutes les pièces de la grille sont rangées correctement et 0 sinon.
Ecrire le programme principal qui initialisera une grille aléatoirement et demandera à chaque tour un déplacement à réaliser après avoir affiché l'état actuel de la grille.
Un coup joué ne sera comptabilisé que si le déplacement proposé par l'utilisateur est valide (*verifGrille*).
Le programme s'arrête lorsque les pièces ont été remises en ordre, le nombre de coups joués sera alors affiché.
Votre programme devra être conçu de manière à pouvoir changer facilement la taille de la grille et donc le nombre de pièces (par ex. carré de 5x5 avec 24 pièces).