Examen final - LO01 A2018

2 H - Sans documents (sauf diagrammes de C) ni calculette
N'oubliez pas de commenter vos programmes. Les types des paramètres et des fonctions sont parfois volontairement omis dans l'énoncé. À vous de les définir.

Vous aurez besoin de 4 copies : une par exercice

1. Gestion d'un stock (5 points) -> copie n°1

Le propriétaire d'un magasin de vêtements a stocké dans un fichier "magasin.dat" l'ensemble des articles qu'il propose à la vente.
Un article est défini par :

- Un nom (chaîne de caractères de 30 caractères maximum)
- Un prix (montant réel en euros)
- Un nombre d'articles restants (entier)
- Un fournisseur

Un fournisseur est défini par :

- Un nom (chaîne de caractères de 30 caractères maximum)
- Un numéro siret (entier)

1.1 / Définir le type *unFournisseur* et *unArticle* basé sur les informations ci-dessus.

1.2 / Écrire la fonction *rechercheUnArticle()* qui a comme paramètre une chaîne de caractères correspondant au nom d'un article et qui renvoie la position (en octets) de l'article dans le fichier "magasin.dat" et -1 s'il n'existe pas. Un article est inexistant si le nom de l'article ne se trouve pas dans le fichier ou que nombre d'articles correspondant est nul.

1.3 / Écrire la fonction *decrementeUnArticle()* qui a comme paramètre une chaîne de caractères correspondant au nom d'un article et décrémente de 1 le nombre d'articles correspondant dans le fichier "magasin.dat". La fonction utilisera *rechercheUnArticle()*.

1.4 / Écrire le programme principal qui suite à une vente, demande au propriétaire le nom d'un article et décrémente le nombre correspondant dans le fichier.

Remarque : on considérera que le fichier "magasin.dat" est déjà pré remplit.

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

Soit à développer un programme permettant la gestion d'un parc d'ordinateurs d'une société de vente de matériels informatiques.
On souhaite représenter sous forme d'un tableau, nommé TO (taille maximum 100), l'ensemble des ordinateurs.
Un ordinateur est identifié par : sa référence (un entier), sa marque (une chaîne de caractères), sa capacité mémoire (un entier), sa capacité disque (un entier), son système d'exploitation (une chaîne de caractères).

2.1/ Définir la structure unOrdi.

2.2/ Écrire une fonction *ajouter()* qui permet d'ajouter un ordinateur au tableau TO. Le tableau sera passé en argument de la fonction.

2.3/ Écrire une fonction *afficher()* qui affiche tous les ordinateurs de TO. Le tableau sera passé en argument de la fonction.

2.4/ Écrire une fonction *supprimer()* qui permet de supprimer un ordinateur dont la référence est passée en paramètre, le tableau TO étant également passé en argument. Le tableau sera compacté afin de récupérer l'espace libéré par l'ordinateur effacé.

2.5/ Écrire le programme principal qui propose par un dialogue avec l'utilisateur de :

1- Ajouter un ordinateur (les caractéristiques seront demandées à l'utilisateur),
2- Afficher les ordinateurs,
3- Supprimer un ordinateur (la référence sera demandée à l'utilisateur)
4- Quitter le programme.

3. Géométrie (5 points) -> copie n°3

Soit 2 points A et B dans $\mathbb{R}^3$ définis chacun par ses coordonnées (x,y,z). On cherche à écrire une fonction *calculeMilieu(A,B,M)* qui calcule le milieu du segment [AB] dans M et retourne la valeur suivante :

2 : si le milieu est sur un des axes Ox, Oy, Oz,
1 : si le milieu est dans un des 3 plans Oxy, Oxz, Ozy (mais par sur un axe),
0 : sinon.

3.1 / Définir le type *unPoint* et écrire la fonction la fonction *calculeMilieu(A,B,M).*

3.2 / Supposons connue la fonction *void saisitUnPoint(unPoint *P)* qui initialise un point à partir d'un dialogue avec l'utilisateur, écrire la fonction *remplitTab(T, n, naxe, nplan)* qui remplit un tableau de *n* points (*n* est un entier pair) de la manière suivante : les $n/2$ premiers points (P_1, P_2 et $P_{n/2}$) seront saisis par l'utilisateur (utiliser la fonction *saisitUnPoint()),* et les $n/2$ derniers seront les milieux des segments [P_1P_2] [P_2P_3] … [$P_{n/2}P_1$] (utiliser la fonction *calculeMilieu()).*

3.3 / Écrire un programme qui remplit un tableau de *N* points (*N* constante paire) à l'aide de la fonction *remplitTab()*, affiche à l'écran le nombre de points milieux appartenant à un axe et le nombre de points milieux appartenant à un plan (mais par sur un axe) et enfin sauve les points du tableau dans un fichier texte *"lesPoints.txt"* (un point par ligne).

4. Fusion (5 points) -> copie n°4

On définit le type structuré suivant qui permet de représenter des ensembles d'au maximum MAX nombres entiers :

```
const int MAX=100;

typedef struct {
        int tab[MAX];
        int taille;
} unEnsemble;
```

Les entiers de l'ensemble sont stockés toujours dans **l'ordre croissant** dans le tableau tab. Le champ taille représente le nombre d'éléments de l'ensemble. Les seuls éléments à considérer sont alors ceux stockés dans tab entre les indices 0 et taille-1 de tab (la place restante du tableau peut être utilisée pour ajouter de nouveaux éléments). Le tableau ne doit contenir **aucun doublon**.

4.1 / Écrire la fonction d'entête "*void unionEnsemble(unEnsemble E1, unEnsemble E2, unEnsemble* res);* " qui permet de faire l'union de deux ensembles *E1* et *E2*. Le résultat sera stocké dans l'ensemble pointé par la variable *res* en respectant les contraintes décrites ci-dessus. La fonction ne sera pas récursive et n'utilisera pas la question suivante.

4.2 / On décide d'écrire la fonction *unionEnsemble* de la manière suivante :

```
void unionEnsemble(unEnsemble E1, unEnsemble E2, unEnsemble* res)
{
    res->taille = fusionTableau(E1.tab, E1.taille, 0, E2.tab, E2.taille, 0, res->tab, 0) ;
}
```

Écrire la fonction récursive " *int fusionTableau(int tab1[], int n, int i, int tab2[], int m, int j, int res[], int k);* " qui permet de fusionner deux tableaux d'entiers triés dans l'ordre croissant (sans doublon) représentés par les variables *tab1* et *tab2* et de stocker le résultat dans le tableau représenté par la variable *res*. Les paramètres n et m représentent les tailles des tableaux *tab1* et *tab2*. Le tableau *res* doit stocker à partir de l'indice *k,* dans l'ordre croissant et sans doublon, tous les éléments du tableau tab1 stockés à partir de l'indice i et tous les éléments du tableau *tab2* stockés à partir de l'indice j. La fonction renvoie le nombre d'éléments stockés dans *res* à partir de l'indice k.