Examen final - LO01

P2013

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

N'oubliez pas de commenter vos programmes.
Remarque 2: Les types des paramètres et des fonctions sont parfois volontairement omis dans l'énoncé. A vous de les définir.

Vous aurez besoin de 4 copies : une par exercice

1. Tableaux de nombres complexes (5 points) NOUVELLE COPIE

Un nombre complexe est un nombre de la forme $z = a+i.b$ où a et b sont des réels quelconques représentant la partie réelle et la partie imaginaire de z respectivement. i, l'unité imaginaire, est un nombre particulier tel que $i^2 = -1$.

1) Définir le type Complexe basé sur une structure.
2) Soit les complexes $z_1=a_1+ i.b_1$ et $z_2=a_2+ i.b_2$ et soit la relation d'ordre suivante : z_1 est inférieur à z_2 si ($a_1 < a_2$) ou si ($a_1 = a_2$ et $b_1 < b_2$). Ecrire une fonction *ordre(z_1,z_2)* qui retourne -1 si z_1 est inférieur à z_2, 0 si z_1 est égal à z_2 et 1 z_2 est inférieur à z_1.
3) Ecrire une fonction *permuteC()* qui permute le contenu de deux complexes.
4) Écrire une fonction *initTabC()* permettant de remplir un tableau de complexes demandés à l'utilisateur. Cette fonction retourne le nombre de complexes effectivement rentrés (maximum 60).
5) Ecrire une fonction *triTabC()* qui trie un tableau de N complexes en ordre croissant en utilisant les fonctions *ordre()* et *permute().*
6) Ecrire le programme qui initialise, trie et affiche un tableau de complexes. La taille maximum du tableau sera égale à 60.

2. Filtrage de tableau (5 points) NOUVELLE COPIE

1) Ecrire une fonction qui permet le remplissage d'une matrice carrée de dimensions 30x30 avec des nombres entiers saisis par l'utilisateur.
2) Ecrire une fonction qui remplit un second tableau de la manière suivante : chaque élément du tableau est le maximum des éléments d'une ligne de la matrice saisie en (1) : le ième élément du tableau sera le maximum des a_{ij} (élément de la ligne i et de la colonne j de la matrice).
3) Ecrire une fonction qui remplit un troisième tableau de la manière suivante : chaque élément du tableau est la moyenne des éléments impairs ou pairs des lignes de la matrice saisie en (1) : t_i (ième élément du tableau) sera la moyenne des a_{ij} (élément de la ligne i et de la colonne j de la matrice) pour tout j pair si i est impair, ou pour tout j impair si i est pair.
4) Ecrire une fonction qui modifie la matrice saisie en (1) de la manière suivante : pour chaque ligne i de la matrice saisie en (1), elle enlève tous les éléments a_{ij} tels que $a_{ij} < t_i$ (ième élément du tableau généré en (3)) en décalant vers la gauche les éléments restants. La ligne sera complétée avec le maximum de la ligne stocké dans le tableau de la question (2).

Exemple pour une ligne i (i impair) de 10 entiers

matrice

	1	2	3	4	5	6	7	8	9	10	max	moyenne
ligne i	1	85	-59	6	18	-34	-4	106	10	-13	106	-5.67
ligne i modifiée	1	85	6	18	-4	106	10	106	106	106		

Remarque : pour passer un tableau 2D en paramètre il faut donner la taille de la 2e dimension.
Ex : void fonction(int T[ ][MAX]). Lors de l'appel on transmet en argument le nom du tableau (comme en 1D)

3. Fonctions et structures de données (5 points) NOUVELLE COPIE

1ère partie: Les souris.

Reprenons l'exemple de la souris (vu en cours) se promenant dans un labyrinthe représenté par une grille 2D (non utilisée ici). Cette fois-ci nous proposons le codage suivant : la position de la souris est donnée par le numéro de ligne et le numéro de colonne de la case du labyrinthe et sa direction est codée par un entier : 0→nord, 1→ouest, 2→sud, 3→ est. Par exemple la souris (1,4,2) est en ligne d'indice 1, en colonne d'indice 4 et se dirige vers le sud.

3 étudiants proposent une solution différente pour représenter une souris:

Etudiant 1 :
```
int nlS ;/*ligne*/
int ncS ;/*colonne*/
int dS ;  /*direction*/
```

Etudiant 2 :
```
int S[3] ;
    /*S[0] ligne*/
    /*S[1]colonne*/
    /*S[2]direction*/
```

Etudiant 3 :
```
typedef struct {
  int nl ;/*ligne*/
  int nc ;/*colonne*/
  int d ;  /*direction*/
  }uneSouris ;

uneSouris S ;
```

1) Quelle est selon vous la meilleure proposition ? Justifier en quelques mots.
2) Ecrire **pour chaque étudiant** la fonction *initUneSouris()* qui initialise une souris à partir d'un dialogue utilisateur. On suppose que l'utilisateur rentre des informations correctes et conformes à la taille et la configuration du labyrinthe.
3) Ecrire **pour chaque étudiant** la fonction *avanceUneCase()* qui fait avancer la souris d'une case dans sa direction. On supposera que l'on aura préalablement testé que cela est possible.
4) Ecrire **pour chaque étudiant** la fonction *tourneADroite()* qui fait faire un quart de tour à droite à la souris. Cette fonction ne modifie pas la position de la souris mais uniquement sa direction.

2e partie : Récursivité.

On cherche à calculer la valeur du polynôme $P_n(x) = 1 + x + x^2 + x^3 + \ldots + x^n$ $\quad n >= 1$

Afin d'éviter de calculer les puissances successives de x, on va utiliser le schéma suivant (dit de Horner) :

$$P_n(x) = 1 + x\,(1 + x\,(1 + x\,(1 + \ldots\, x\,(1 + x\,(1 + x))\ldots)))$$

Ecrire la fonction récursive *PnHorner*() qui calcule selon le schéma de Horner, P_n pour x et n donnés.

4. Tirages du loto (5 points) NOUVELLE COPIE

Chaque tirage du loto correspond à 6 nombres compris entre 1 et 49 pour une date donnée (exemple 20 janvier 2006). On propose de conserver dans un fichier "loto.dat" l'historique tous les tirages depuis sa création. Chaque enregistrement du fichier aura la structure suivante :

date nombre1 nombre2 nombre3 nombre4 nombre5 nombre6

Exemple : 12 juin 2007 2 47 26 14 43 3

Pour des raisons de facilité, les nombres seront stockés dans un tableau contenant les 6 nombres.

1) Ecrire la structure *uneDate*. Ecrire la structure *unTirage* qui utilisera la structure *uneDate*.
2) Ecrire la fonction *initUnTirage(tirage)* qui remplira la variable tirage de type *unTirage* par un dialogue avec l'utilisateur.
3) Ecrire la fonction *creeFichierLoto()* sans paramètre qui créera le fichier "loto.dat" et qui remplira chaque enregistrement par l'intermédiaire de la fonction *initUnTirage()*.
4) Ecrire la fonction *statistique(int an)* qui pour une année **an** (donnée en paramètre), calculera le nombre de fois qu'a été tiré un nombre. La fonction affichera le résultat sous la forme :

annee : 2008
Nombre 1 : 56
Nombre 2 : 34
…
…
Nombre 49 : 88