Examen final - LO01

A2014

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

N'oubliez pas de commenter vos programmes.

Vous aurez besoin de 3 copies : une par exercice

1. Sudoku 2ième version (7 points)

Le Sudoku est un jeu bien connu que vous avez eu l'occasion de découvrir au médian. Vous allez écrire un programme permettant à un utilisateur de conserver l'historique des grilles de Sudoku qu'il a réalisées. Il n'est pas demandé de tester si les grilles sont correctes.
Un jeu est caractérisé par une date, un niveau et une grille.
La date comporte 3 champs le jour, le mois et l'année, exemple : 9 janvier 2016 (le mois étant une chaîne de caractères).
Le niveau est un entier de 1 à 5 caractérisant la difficulté.
La grille est un tableau d'entiers à deux dimensions 9x9, chaque entier aura une valeur de 1 à 9.

1 – Ecrire la structure **`uneDate`** et la structure **`unJeu`** qui utilisera la structure **`uneDate`**.

2 – Ecrire la fonction **`compareDate(date1, date2)`** qui comparera les deux paramètres **`date1`** et **`date2`** en renvoyant 1 si les dates sont identiques et 0 sinon.

3 – Ecrire la fonction **`initUnJeu(jeu)`** qui remplira les différents champs du paramètre **`jeu`** de type **`unJeu`** par un dialogue avec l'utilisateur.

4 – Ecrire la fonction **`insereJeuDansFichier()`** sans paramètre qui ajoutera à la fin du fichier existant **`"Sudoku.dat",`** un nouveau **`jeu`** demandé à un utilisateur.

5 – Ecrire la fonction **`afficheUneGrille(date)`** qui aura comme paramètre une date, qui recherchera dans le fichier **`"Sudoku.dat"`** la date correspondant à une grille de Sudoku et qui affichera la grille associée. La fonction renverra -1 si la date correspondante n'existe pas et 1 sinon.

```
#define MAXJ 15
#define MAXI 7


typedef struct {
        int jour;
        char mois[MAXJ];
        int annee;
        }uneDate;

typedef struct {
        uneDate date;
        int diff;
        int grille[MAXI][MAXI];
        }unJeu;

int compareDate(uneDate d1, uneDate d2)
{
   if((d1.jour == d2.jour) && strcmp(d1.mois,d2.mois) == 0 && (d1.annee == d2.annee) )
      return 1;
   else
      return 0;
}

void initUnJeu(unJeu *unJ)
{ int i,j;
   printf("Entrez la date jj mois AAAA");
   scanf("%d %s %d",&unJ->date.jour, unJ->date.mois, &unJ->date.annee);
   printf("Entrez la difficulté 1 à 5");
   scanf("%d",&unJ->diff);
   for (i=0; i<MAXI; i++)
      for(j=0; j<MAXI; j++) {
         printf("grille[%d][%d] = ", i,j);
```

```
            scanf("%d",&unJ->grille[i][j]);
    }
}

void insereJeuDansFichier()
{
    FILE *fp;
    unJeu unJ;
    fp = fopen("sudoku.dat","a+");
    initUnJeu(&unJ);
    fwrite(&unJ,sizeof(unJeu),1,fp);
    fclose(fp);
}

int afficheUnJeu(uneDate date)
{
    FILE *fp;
    unJeu unJ;
    int i,j,result;
    fp = fopen("sudoku.dat","r");
    do{
        if((result=fread(&unJ,sizeof(unJeu),1,fp))==1)
            if(compareDate(date,unJ.date)){
                for (i=0; i<MAXI; i++){
                    for(j=0; j<MAXI; j++)
                        printf("%d ", unJ.grille[i][j]);
                    printf("\n");
                    }
                fclose(fp);
                return 1;
                }
    }
    while(result == 1);
    fclose(fp);
    return 0;
}

int main()
{
    uneDate date;
    insereJeuDansFichier();
    insereJeuDansFichier();
    printf("entrez une date de recherche");
    scanf("%d %s %d",&date.jour, date.mois, &date.annee);
    if(!afficheUnJeu(date))
        printf("Date inexistante !\n");
    return 0;
}
```

---

2. Promotions (7 points) **Nouvelle Copie**

Dans une administration, le grade d'un employé est caractérisé par :

- son rang : B ou A
- sa classe : 2e classe, 1ère classe ou classe exceptionnelle (classe 0)
- son échelon : varie de 1 à 5

Le niveau le plus bas dans la hiérarchie est le rang B, 2e classe au 1er échelon.
Le niveau suivant est rang B, 2e classe, 2e échelon, puis rang B, 2e classe, 3e échelon, etc. Après avoir atteint le 5e échelon, il passera au rang B, 1ère classe, 1er échelon, puis rang B, 1ère classe, 2e échelon et ainsi de suite.
La promotion d'un employé au rang B, classe 0 au 5e échelon le fait passer au rang A, 2e classe, 1e échelon.
Le niveau le plus élevé est le rang A, classe 0 à l'échelon 5.

Un employé est défini par son nom (40 caractères max), son numéro de service, et son grade.

1 – Définir les types **`unGrade`** et **`unEmploye`** basés sur une structure permettant de décrire un grade et un employé d'administration.

2 – Ecrire la fonction **`promotion(G)`** qui fait évoluer le grade **`G`** d'un employé au niveau suivant. La fonction retournera 1 si la promotion a pu être appliquée et 0 si l'employé ne peut plus être promu car ayant atteint le niveau le plus élevé.

3 – Ecrire une fonction **`compare(G1, G2)`** qui compare les 2 grades **`G1`** et **`G2`** et retourne 1 si le grade **`G1`** est plus élevé que **`G2`**, -1 si le grade **`G2`** est plus élevé que **`G1`** et 0 si les 2 grades sont équivalents.

4 – On suppose que l'on dispose d'un tableau contenant des employés. Ecrire une fonction **`void promotionEmploye(unEmploye T[], int nb, char nomE[])`** qui applique une promotion à l'employé dont le nom est donné en paramètre (on suppose que l'employé est bien dans le tableau **`T`** contenant **`nb`** employés).

Corrigé

1)
```
typedef struct {
       char rang ;
       int classe ;
       int echelon ;
       } unGrade ;

typedef struct {
       char nom[40] ;
       int service ;
       unGrade grade ;
       } unEmploye ;
```

2)
```
int promotion(unGrade *G)
{ int R =1 ;

  if (G->echelon != 5) G->echelon++ ;
  else if (G->classe != 0) {
             G->classe-- ;
             G->echelon=5 ;
             }
        else if (G->rang != 'A') {
                   G->rang='A' ;
                   G->classe=2 ;
                   G->echelon=5 ;
                   }
               else R=0 ;
  return R ;
}
```

3)
```
int compare(unGrade G1, unGrade G2)
{
  if (G1.rang < G2.rang) return -1 ;
  else if (G1.rang > G2.rang) return 1 ;
       else if (G1.classe > G2.classe) return -1 ; // attention c'est inversé
            else if (G1.classe < G2.classe) return 1 ;
                 else if (G1.echelon < G2.echelon) return -1 ;
                    else   if (G1.echelon > G2.echelon) return 1 ;
                           else return 0 ;
}
```

4)
```
void promotionEmploye(unEmploye T[], int nb, char nomE[])
{ int i ;

 for (i=0; i<nb; i++)
       if (strcmp(nomE, T[i].nom) == 0) promotion(&T[i].grade) ;
}
```

---

3. Statistiques (7 points) **Nouvelle Copie**

On se propose d'étudier sur un échantillon de 100 personnes, le nombre d'enfants dans les familles.

1 - Ecrire une fonction **`void initTab (...)`** qui permet à un utilisateur de saisir dans un tableau, passé en paramètre, le nombre d'enfants (0 à 5) de 100 employé(e)s (familles) d'une entreprise.

2 - Ecrire une fonction **`void moments(...)`** qui calcule la moyenne et la variance des valeurs comprises dans le tableau. La fonction prendra en entrée le tableau passé en paramètre, et sortira la moyenne et la variance. Les définitions des moyennes et variances sont les suivantes :

Soit un échantillon de taille *n* d'une variable aléatoire *X* (un ensemble contenant *n* valeurs numériques x_i de *X*), la moyenne empirique et la variance empirique sont respectivement calculées par :

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$s^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x})^2$$

3 - On supposera connue la fonction **`void triTab(...)`** qui trie le tableau passé en paramètre par ordre croissant.
Ecrire une fonction **`void freqRel(...),`** qui après voir trié le tableau, remplit une matrice (tableau de 2 dimensions) passée en paramètre, à partir du tableau aussi passé en paramètre et trié, de manière suivante :
- 1ère colonne : les nombres d'éléments n_k pour chaque type de famille dans le tableau (où *k* est le nombre d'enfants : 0 à 5);
- 2ième colonne : la fréquence relative f_k (nombre de fois que la valeur x_k apparait dans le tableau divisé par le nombre total d'éléments);
- 3ième colonne : la fréquence accumulée F_k (accumulation des fréquences relatives jusqu'à f_k, y comprise)
La fonction affichera la matrice suivante (pour un exemple numérique donné de 100 familles) :

Nb enfants	n_k	f_k	F_k
0	22	0,22	0,22
1	26	0,26	0,48
2	30	0,30	0,78
3	11	0,11	0,89
4	9	0,09	0,98
5	2	0,02	1

Sous la forme :

Nb enfants	nk	fk	Fk
0	22	0.22	0.22
1	26	0.26	0.48
2	30	0.3	0.78
3	11	0.11	0.89
4	9	0.09	0.98
5	2	0.02	1

```
1)
void permute(int* x, int* y)
{
	int temp;
	temp = *x;
	*x = *y;
	*y = temp;
}

void Init_Tab_Inverse(int* tab)
{
	int i, j;
	for(i = 0; i < MAX_TAB; i++){
		printf("tab[%d] = ", i);
		scanf("%d",&tab[i]);
	}

	for(i = 0; i < MAX_TAB / 2; i++)
		permute(&tab[i], &tab[MAX_TAB-1 - i]);
}

2)
void moments(int* tab, float* moy, float* var)
{
   int i;

   *moy = 0.0;
   for(i = 0; i< MAX_TAB; i++)
	*moy += tab[i];
   *moy /=MAX_TAB;
```

```
   *var = 0.0;
   for(i = 0; i<MAX_TAB;i++)
          *var +=(tab[i]- *moy)*(tab[i]- *moy);
   *var /= MAX_TAB;
}

3)

void Tab_Freq(int* tab, float Mat[][3])
{
    int i = 0, type_famille, nb_type_famille;

       // met tab en ordre croissant : FONCTION CONNUE
       //Tri_Tab(tab);

       // calcul le nombre de cas pour chaque type de famille
       type_famille = 0;
       i = 0;
       while( i < MAX_TAB && type_famille < 6){
              nb_type_famille = 0;
              while( tab[i] == type_famille){
                     nb_type_famille++;
                     i++;
              }
              Mat[type_famille][0] = nb_type_famille;
              type_famille++;
       }

       // calcul des fréquences relative et accumulée pour chaque cas
       Mat[0][1] = Mat[0][0]/MAX_TAB;
       Mat[0][2] = Mat[0][1];

       for(i = 1; i < 6; i++){
              Mat[i][1] = Mat[i][0]/MAX_TAB;
              Mat[i][2] = Mat[i][1] + Mat[i-1][2];
       }

       // affichage du tableau de fréquence
       for(i = 0; i< 6; i++)
              printf("\n %d %f %f %f",i,Mat[i][0], Mat[i][1], Mat[i][2]);

}


void main()
{
       int tab[MAX_TAB],i;
       float Mat[6][3], moy, var;

       Init_Tab_Inverse(tab);
       moments(tab,&moy,&var);
       Tab_Freq(tab,Mat);

       printf("\n");
}
```