Auteur : Mohamed Messabihi
Matière : Programmation et structures de données
Date 11 juillet 2019
Durée 1h30

Université Abou Bakr Belkaïd - Tlemcen
Faculté des Sciences
1ère Année MI
Semestre 2

Correction du Contrôle Continu

Aucun document n'est autorisé
Les solutions doivent être rédigées en **C**
Les appareils portables doivent être éteints et posés sur le bureau du surveillant

1 Affichage — 8 pts. 30'

Qu'affichent les deux programmes suivants :

```c
#include<stdio.h>
void Toto(int *a, int *b)
{   int *c;
    c=a;
    a=b;
    b=c;
}
void main()
{   int i=1, j=2;
    int *p=&i, *q=&j;
    printf("a=%d\n",i);
    printf("b=%d\n",j);
    Toto(&i,&j);
    printf("a=%d\n",i);
    printf("b=%d\n",j);
}
```

```c
#include<string.h>
void main()
{
 int T[8]={0, 1, 2, 3, 4, 5, 6};
 int M[3][4]={{1,2,3}, {4,5}, {6}};
 int i, *p = T, *q=M;
 printf("*p = %d \n", *p);
 for(i=0;i<7;i++)
   {
     (*p)= *(p+1);
     p++;
   }
 printf("T[1]=%d\n",T[1]);
 printf("**M = %d \n", **M);
 printf("*(*(M+1)+1) = %d \n", *(*(M+1)+1));
}
```

Solution

Affichage

```
a=1
b=2
a=1
b=2
```

Affichage

```
*p = 0
T[1]=2
**M = 1
*(*(M+1)+1) = 5
```

2 Carrés magiques — 12 pts. 60'

Un carré magique est une matrice carrée de taille $n \times n$ telle que la somme de chaque ligne, de chaque colonne et de chaque diagonale principale soient égales. Un carré magique est dit normal s'il contient chaque entier compris entre 1 et n^2 exactement une fois, où n est l'ordre du carré.

Exemples. Le tableau suivant est un carré magique normal d'ordre 3 :

6	7	2
1	5	9
8	3	4

1. Écrire deux fonctions Somme_Ligne et Somme_Colonne qui prennent en entrée un tableau et un numéro de ligne (respectivement de colonne) et qui renvoient la somme des éléments de cette dernière.

Solution

```
int somme_ligne(int n, int carre[n][n], int i){     //1pt
    int j, s = 0;
    for (j=0 ; j<n ; ++j)
        s += carre[i][j];
    return s;
}
int somme_colonne(int n, int carre[n][n], int j){   //1pt
    int i, s = 0;
    for (i=0 ; i<n ; ++i)
        s += carre[i][j];
    return s;
}
```

2. Écrire deux fonctions Somme_Diagonale et Somme_AntiDiagonale qui retournent la somme de la diagonale (respectivement de l'antidiagonale) du tableau passé en paramètre.

Solution

```
int somme_diagonale(int n, int carre[n][n]){        //1pt
    int i, s = 0;
    for (i=0 ; i<n ; ++i)
        s += carre[i][i];
    return s;
}
int somme_antidiagonale(int n, int carre[n][n]){    //2pt
    int i, s = 0;
    for (i=0 ; i<n ; ++i)
        s += carre[i][n-1-i];
    return s;
}
```

3. Écrire une fonction Carre_Magique qui retourne 1 si le tableau passé en paramètre est un carré magique normal et 0 sinon

Solution

```
int Carre_Magique(int n, int t[n][n]){              //3pt
    int i, somme=somme_diagonale(n,t);
    if (somme!=somme_antidiagonale(n,t))
        return 0;
    for (i=0 ; i<n ; ++i)
        if (somme != somme_ligne(n,t,i)
            || somme != somme_colonne(n,t,i))
            return 0;
    return 1;
}
```

4. Écrire un programme qui demande à l'utilisateur de saisir un tableau, et affiche s'il s'agit d'un carré magique normal.

Solution

```c
int main(void){                                                    //2pt
    printf("Ordre du carre ? ");
    int i, j, n;
    scanf("%d",&n);
    int carre[n][n];
    printf("\n\n");
        for(i=0; i<n; i++)
                for(j=0; j<n; j++)
                        scanf("%d", &carre[i][j]);
    //generer_carre_magique(n,carre);
    printf("Ce carre %s magique\n", Carre_Magique(n,carre)? "est" : "n'est pas");
}
```

5. Écrire une fonction Generer_Carre_Magique qui permet de construire un carré magique d'ordre n impair (le tableau et l'ordre *n* sont passés comme paramètres).

Solution

— On commence par initialiser tous les éléments du tableau à zéro. L'initialisation à zéro de tous les éléments du carré permettra ultérieurement de déduire si la case pointée est libre ou non.
— Le chiffre 1 est placé dans la case au milieu de la dernière ligne du carré.
— Une fois qu'une case a été remplie, on en choisit une autre en effectuant, par rapport à la case qui vient d'être remplie précédemment, deux mouvements successifs :
  (a) l'un horizontal vers la case de droite
  (b) l'autre vertical d'une case vers le haut
— Si i est l'indice de ligne et j l'indice de colonne, ce déplacement s'obtient en effectuant i = i+1 et j = j+1. Deux cas peuvent alors se présenter :
  (a) La case atteinte est libre, on y place alors le nombre suivant.
  (b) La case atteinte est déjà remplie, on en choisit une autre en effectuant un déplacement vertical vers le haut à partir de la case de départ.
— La méthode mentionnée ci-dessus entraînera tôt ou tard une sortie hors des limites du carré défini. Dès qu'un déplacement oblige à sortir du carré, on se place dans la case correspondante du bord opposé et on continue à effectuer la suite des déplacements.
On peut interpréter cette règle d'une autre manière, en considérant simplement que le carré est, pour chaque dimension, placé sur un cylindre de telle sorte que la ligne du bas est située immédiatement au-dessus de celle du haut, que la colonne de droite est située à gauche de la première colonne, et réciproquement pour la colonne de gauche.

```c
void initialiser_carre(int n, int t[n][n]){
    int i, j;
        for(i=0; i<n; i++)
                for(j=0; j<n; j++)
                        t[i][j] = 0;
}
void generer_carre_magique(int n, int t[n][n]){        //2pt
    int i, j, k;
    initialiser_carre(n,t);
    for (k=1, i=n-1, j=n/2 ; k<=n*n ; ++k){
        t[i][j] = k;
        i = (i+1)%n;
        j = (j+1)%n;
        if (t[i][j]){
            i = (i-2+n)%n;
            j = (j-1+n)%n;
        }
    }
}
```

« Bon courage »