Algo/JavaScript – USAL34

TP3 Les boucles et les tableaux en C

Objectifs du TP

- Apprendre à utiliser les structures de contrôles itératives (boucles `for`)
- Apprendre à manipuler les tableaux à une dimension
- Apprendre à manipuler les tableaux à deux dimensions

1 Rappels : bibliothèque et compilation

La bibliothèque d'entrée-sortie : inout.h et inout.c et mettez les dans un répertoire `tp3`. Pour cela vous pouvez faire les commandes suivantes (une seule fois) dans un terminal (menu pricipal : `terminal/Konsole`) :

```
mkdir tp3
cd tp3
wget "https://cedric.cnam.fr/~lamberta/enseignements/DSP/C/sources/inout.h"
wget "https://cedric.cnam.fr/~lamberta/enseignements/DSP/C/sources/inout.c"
```

Programmez et tester ces fonctions *une par une* (testez une fonction dès que vous pensez qu'elle est finie). Pour tester une fonction `f` on programme un ou plusieurs appels à cette procédure dans une procédure `test_f` : on teste le résultat obtenu par rapport au résultat attendu et on affiche un message d'erreur si ils ne sont pas égaux.

2 Les boucles et les tableaux à 1 dimension

Exercice 1 — *Les fonctions itératives*

1. Écrivez une procédure `void carre(int n)` qui affiche `n` lignes de largeur `n`, remplies de caractères `'*'`.

   Par exemple, `carre(4);` doit afficher :

   ```
   ****
   ****
   ****
   ****
   ```

   Testez cette fonction dans une procédure `void~testCarre()` qui demande à l'utilisateur la largeur qu'il désire (et qui passe la réponse en paramètre à la procédure `carre`.
2. Écrivez une procédure `void triangle(int n)` qui affiche un triangle (pointe vers le haut) composé de caractère `'*'`. Testez la de la même manière que `carre`.

   Par exemple, `triangle(4);` doit afficher :

   ```
   *
   **
   ***
   ****
   ```
3. Écrivez une procédure `void triangleInverse(int n)` qui affiche un triangle pointe vers le bas composé de caractère `'*'`. Testez.

   Par exemple, `triangleInverse(4);` doit afficher :

```
****
***
**
*
```

4. Écrivez une procédure `void triangleInverse2(int n)` qui affiche un triangle pointe vers le bas composé de caractère `'*'` mais dont l'angle droit est à droite. Testez.
   Par exemple, `triangleInverse(4);` doit afficher :

```
****
 ***
  **
   *
```

► **Correction**

```
#include "inout.h"

void carre(int n){
  int i,j;
  for (i=0;i<n;i++)
    {
      for (j=0;j<n;j++)
        ecrireString("*");
      ecrireString("\n");
    }
}

void triangle(int n){
  int i,j;
  for (i=0;i<n;i++)
    {
      for (j=0;j<=i;j++)
        ecrireString("*");
      ecrireString("\n");
    }
}

void triangleInverse(int n){
  int i,j;
  for (i=0;i<n;i++)
    {
      for (j=0;j<n-i;j++)
        ecrireString("*");
      ecrireString("\n");
    }
}

void triangleInverse2(int n){
  int i,j;
  for (i=0;i<n;i++)
    {
      for (j=0;j<n;j++)
        if (j<i)
          ecrireString(" ");
        else
          ecrireString("*");
      ecrireString("\n");
    }
}

void main ( void ) {
```

```
  ecrireString("Entrez la largeur du carrÃ©\n");
  int n= lireInt();
  carre(n);
  ecrireString("Entrez la largeur du triangle\n");
  n= lireInt();
  triangle(n);
  ecrireString("Entrez la largeur du triangle\n");
  n= lireInt();
  triangleInverse(n);
  ecrireString("Entrez la largeur du triangle\n");
  n= lireInt();
  triangleInverse2(n);


}
```

Exercice 2 — *Fonction manipulant un tableau à une dimension*

Programmez les fonctions et procédures ci-dessous. Elles nécessitent des boucles pour parcourir les tableaux.

1. `void affiche(int t[], int n)` qui affiche les éléments du tableau `T` de taille `n`
2. `BOOL estPresent(int t[], int n, int x)` qui teste si un entier `x` est dans le tableau `T` de taille `n`
3. `int nombreOccurence(int t[], int n, int x)` qui retourne le nombre d'occurrences de l'entier `x` dans le tableau `T` de taille `n`
4. `int minimum(int t[], int n)` qui retourne la plus petite valeur du tableau `T` de taille `n`
5. `int maximum(int t[], int n)` qui retourne la plus grande valeur du tableau `T` de taille `n`
6. `double moyenne(int t[], int n)` qui retourne la moyenne des valeurs du tableau `T` de taille `n`
7. `BOOL estTrie(int t[], int n)` qui teste si le tableau `T` de taille `n` est trié.
8. `void decalage(int t[], int n)` qui décale les valeurs du tableau `T` de taille `n`. Si le tableau `T = {1,2,3,4}`, la procédure le modifie en `T = {2,3,4,1}`.

► **Correction**

```
#include "inout.h"
#define BOOL int
#define TRUE 1
#define FALSE 0

void affiche(int t[],int n){
  int i;
  for (i=0;i<n;i++)
      ecrireInt(t[i]);
}

BOOL estPresent (int t[],int n, int x){
  int i;
  for (i=0;i<n;i++)
    {
      if(t[i] ==x)
        return TRUE;
    }
  return FALSE;
}
```

```
int nbOccurrence(int t[],int n, int x){
  int i;
  int cpt=0;
  for (i=0;i<n;i++)
    {
      if(t[i] ==x)
        cpt++;
    }
  return cpt;
}

int minimum(int t[],int n){
  int i;
  int min =t[0];
  for (i=1;i<n;i++)
    {
      if(t[i] < min)
        min = t[i];
    }
  return min;
}

int maximum(int t[],int n){
  int i;
  int max =t[0];
  for (i=1;i<n;i++)
    {
      if(t[i] > max)
        max = t[i];
    }
  return max;
}

int moyenne(int t[],int n){
  int i;
  int moyenne =0;
  for (i=0;i<n;i++)
    {
      moyenne+= t[i];
    }
  return moyenne/n;
}

BOOL estTrie(int t[],int n){
  int i;
  for (i=0;i<n-1;i++)
    {
      if(t[i] >t[i+1])
        return FALSE;
    }
  return TRUE;
}
void decalage(int t[],int n){
  int temp=t[0];
  int i;
  for (i=0;i<n-1;i++)
    {
      t[i]=t[i+1];
    }
  t[n-1]=temp;
}
```

```
void main ( void ) {
  int t[3]={1,1,3};

  affiche(t,3);
  BOOL res = estPresent (t,3,1);
  ecrireInt(res);

  int occ =nbOccurrence(t,3,1);
  ecrireInt(occ);
  int min = minimum(t,3);
  ecrireInt(min);
  int max = maximum(t,3);
  ecrireInt(max);
  int moy = moyenne(t,3);
  ecrireInt(moy);
  res = estTrie(t,3);
  ecrireInt(res);
  decalage(t,3);
  affiche(t,3);


}
```

3 Fonctions manipulant un tableau à double entrées

En C (contrairement à Java par exemple), les tableaux à entrées multiples sont en fait des tableaux simples en mémoire. Par exemple le tableau `t1` déclaré comme ceci :

```
int t1 [n][m];
```

sera en fait alloué en mémoire exactement comme le tableau `t2` déclaré comme cela :

```
int t2 [n*m];
```

En revanche on n'accède pas aux cases de la même manière. Pour accéder aux cases de `t1` il faut donner deux indices. Par exemple pour accéder à la treizième case on écrit :

```
t2[1][2]
```

qui est transformé par le compilateur en :

```
t2[1*m+2]
```

c'est-à-dire `t2[12]` qui correspond bien à la 13^e case de `t`.

Le tableau est donc « découpé » en tranche de m cases. En général, on décide que le tableau à double entrée représente donc une matrice : chaque « tranche » représente une « ligne ». Pour accéder à la j^e case de la i^e ligne on écrit `t2[i][j]`, autrement dit le premier indice représente le numéro de la ligne et le deuxième le numéro de la colonne.

Attention en C, lorsqu'on transmet un tableau à plusieurs dimensions en paramètre d'une fonction (ou procédure)on fait comme pour les tableaux à une dimension, sauf que *seule la dimension la plus à gauche peut être omise dans le prototype de la fonction.*

Exercice 3 — *Fonctions manipulant un tableau à 2 dimensions*
Programmez et testez les fonctions et procédures ci-dessous.

1. `void affiche(int t[][3], int nblignes)` Affichage du tableau `t` de taille 3×`nblignes`. Par exemple :

   ```
   1 2 3
   4 5 6
   7 8 9
   ```

2. `void affichenTranspose(int t[][3], int nblignes)` Affichage du tableau `t` de taille 3×`nblignes` en renversant les lignes et les colonnes.
3. `void afficheDiagonale(int t[][3], int nblignes)` Affichage de la diagonale (haut gauche vers bas droit) du tableau `t` de taille 3×`nblignes`.
4. `void afficheDiagonale2(int t[][3], int nblignes)` Affichage de la diagonale (haut droit vers bas gauche) du tableau `t` de taille 3×`nblignes`.
5. `void affichecolonne(int t[][3], int nblignes, int ncol)` Affichage de la colonne `col` du tableau `t` de taille 3×`nblignes`.
6. `BOOL estPresent(int t[][3],int nblignes, int x)` qui teste si un entier `x` est dans le tableau `T` de taille 3×`nblignes`. les deux diagonales et les deux colonnes du milieu du tableau de taille 3×`nblignes`.
7. `BOOL testColonneZeros(int t[][3], int nblignes, int ncol)` qui retourne `TRUE` si la colonne `ncol` du tableau `t` est composée uniquement de zéros.Testez cette fonction dans un `if` de la forme :

   ```
   if (testColonneZeros(t,5,2)) { ... ] else { ... }
   ```

   Remarque : la fonction `testColonneZeros` doit *retourner* `BOOL`, pas l'afficher.
8. `int sommeColonne(int t[][3], int nblignes, int ncol)` qui retourne la somme des éléments de la colonne `ncol` du tableau `t` de taille 3×`nblignes`.

► **Correction**

```
#include "inout.h"
#define BOOL int
#define TRUE 1
#define FALSE 0

void affiche(int t[][3], int nblignes){
  int i,j;
  for (i=0;i<nblignes;i++)
    {
      for (j=0;j<3;j++)
        ecrireInt(t[i][j]);
      ecrireString("\n");
    }
  ecrireString("\n");
}

void afficheTranspose(int t[][3], int nblignes){
  int i,j;
  for (j=0;j<3;j++)
    {
      for (i=0;i<nblignes;i++)
        ecrireInt(t[i][j]);
      ecrireString("\n");
    }
  ecrireString("\n");
}



void afficheDiag(int t[][3], int nblignes){
  int i;
```

```
  for (i=0;i<nblignes;i++)
    ecrireInt(t[i][i]);
  ecrireString("\n");
}

void afficheDiag2(int t[][3], int nblignes){
  int i;
  for (i=0;i<nblignes;i++)
    ecrireInt(t[i][2-i]);
  ecrireString("\n");
}

void afficheColonne(int t[][3], int nblignes, int ncol){
  int i;
  for (i=0;i<nblignes;i++)
    {
      ecrireInt(t[i][ncol]);
      ecrireString("\n");
    }
  ecrireString("\n");
}

BOOL estPresent(int t[][3],int nblignes, int x){
  int i,j;
  for (i=0;i<nblignes;i++)
    {
      for (j=0;j<3;j++)
         if(t[i][j] ==x)
        return TRUE;
    }
  return FALSE;
}

BOOL testColonneZeros(int t[][3], int nblignes, int ncol){
  int i;
  for (i=0;i<nblignes;i++)
    {
      if(t[i][ncol] != 0)
        return FALSE;
    }
  return TRUE;
}

int sommeColonne(int t[][3], int nblignes, int ncol){
  int i;
  int somme = t[0][ncol];
  for (i=1;i<nblignes;i++)
    {
       somme += t[i][ncol];

    }
  return somme;
}




void main ( void ) {
  int t[3][3]={{1,2,3},{4,5,6},{7,8,9}};

  affiche(t,3);
```

```
  afficheTranspose(t,3);
  afficheDiag(t,3);
  afficheDiag2(t,3);
  afficheColonne(t,3,1);
  BOOL res = estPresent (t,3,1);
  ecrireInt(res);
  ecrireString("\n");
  res = testColonneZeros(t,3,1);
  ecrireInt(res);
  ecrireString("\n");
  int somme = sommeColonne(t,3,1);
  ecrireInt(somme);
  ecrireString("\n");
}
```