TD 4 Programmation – DUT 1 – Boucles imbriquées

P.Courtieu

Septembre 2017

Résumé

On écrit différents algorithme en C nécessitant des boucles imbriquées, d'abords sans tableau puis avec des tableaux à simple et double entrées.

1 Début du td

La bibliothèque d'entrée-sortie : inout.h et inout.c et mettez les dans un répertoire `td2`. Pour cela vous pouvez faire les commandes suivantes (une seule fois) dans un terminal (menu principal : `terminal/Konsole`) :

```
mkdir tdc
cd tdc
wget "http://deptinfo.cnam.fr/~courtiep/inout/inout.h"
wget "http://deptinfo.cnam.fr/~courtiep/inout/inout.c"
```

Gardez la documentation de cette librairie dans un onglet de votre navigateur, ainsi que celle du module `math.h`.
Squelette initial :

```c
//#include<stdio.h> // déjà fait par inout.h
#include"inout.h"
#include<stdlib.h>
#include <string.h>
#include <math.h>
// Des alias pratique pour "cacher" que int et bool sont la même chose
// en C...
#define BOOL int
#define FALSE 0
// les tests positifs ne retournent pas toujours 1, bien utiliser
// "if(test)" et surtout pas "if(test==TRUE)"
#define TRUE 1


// f et g sont des exemples de fonction qui ne retourne rien (void).
// Ce sont des "procédures". NE PAS CONFONDRE AFFICHER ET RETOURNER.
void f () {
  ecrireString("Hello World\n");
}

void g() {
  int n = lireInt(); // attend que l'utilisateur tape un entier
  ecrireString("Voici la valeur incrémentée: ");
  ecrireInt(n + 1);
  ecrireString("\n");
}
```

```
// h est un exemple de fonction qui RETOURNE une valeur de type int:
int h(int n){
  return 2 * n;
}

long fermat (int n) {
  return 0L;
}

void affichenx10(int t[][10], int nblignes) {
  // Syntaxe des bouvle "for" en C:
  // for (int i = 0 ; <test pour continuer>; <mise à jour de i>) {
  // exemple:
  for (int k = 0 ; k < 100; k = k + 1) {
    ecrireString("k = ");
    ecrireInt(k);
    ecrireString(", ");
  }
}

// Fonction principale (c'est elle qui est appelée quand on lance le programme)
int main(char **args,int nargs) {

  // f et g peuvent être appelées mais elle ne retourne pas de
  // résultat (malgré l'affichage. On ne peut donc pas les utiliser
  // dans un calcul. Le seul intérêt est leur "effet" immédiat.

  f(); // Appel à la procédure f
  g(); // Appel à la procédure g

  // h retourne une valeur, on peut l'utiliser dans une expression (un calcul)
  ecrireInt(h(12));

  // Tests

  ecrireChar('*');
  ecrireSautDeLigne();
  ecrireInt(M_PI); // oooops le nombre à virgule (double) est "traduit" en entier (int)!
  ecrireDouble(M_PI); // là c'est mieux
  ecrireSautDeLigne();


  /*
  ecrireString("nombre de fermat no ?");
  int n = lireInt();
  ecrireString("\n");
  ecrireLong(fermat(n));
*/
/*
  int t[][10] =
    {
      { 0,0,0,0,0,0,0,0,0,0,},
      { 0,0,0,0,0,0,0,0,0,0,},
      { 0,0,0,0,0,0,0,0,0,0,},
      { 0,0,0,0,0,0,0,0,0,0,},
```

```
        { 0,0,0,0,0,0,0,0,0,0,},
        { 0,0,0,0,0,0,0,0,0,0,},
        { 0,0,0,0,0,0,0,0,0,0,},
        { 0,0,0,0,0,0,0,0,0,0,},
        { 0,0,0,0,0,0,0,0,0,0,},
        { 0,0,0,0,0,0,0,0,0,0,} };

  affichenx10(t, 10);
*/


  // DERNIERE INSTRUCTION
  return 0; // 0 = tout va bien, sinon code d'erreur

}
```

Programmez et tester les fonctions et procédures *une par une* (testez dès que vous pensez que ça marche).

2 Fonctions sans tableaux

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 :

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

3 Fonction manipulant un tableau à une entrée

En C les tableaux on ne peut pas « demander » à un tableau sa taille. Il faut la connaître à l'avance. Dans cette exercice la taille du tableau est connue.

Programmez les fonctions et procédures ci-dessous. Elles nécessitent des boucles imbriquées bien que le tableau manipulé soit à une seule entrée. Dans ce qui suit on se donne un tableau de taille 50, qu'on décide d'afficher comme une grille de 5 lignes de 10 entiers.

1. `void affichenx10(int t[])` Affichage du tableau `t` en 5 lignes de 10 entiers. Par exemple :

   ```
   1 2 3 4 5 6 7 8 9 0
   1 2 3 4 5 6 7 8 9 0
   1 2 3 4 5 6 7 8 9 0
   1 2 3 4 5 6 7 8 9 0
   1 2 3 4 5 6 7 8 9 0
   ```

   Astuce : pour afficher un entier `n` avec un nombre minimal de caractères de 3 à utiliser, utiliser `printf("%3d",n)` (nécessite `#include<stdio.h>`)
2. `void affichenx10Transpose(int t[])` Affichage du tableau `t` en affichant lesd lignes en colonnes et les colonnes en lignes.
3. `void afficheDiagonale(int t[][10])` Affichage de la diagonale (haut gauche vers bas droit) du tableau `t`.
4. `void afficheDiagonale2(int t[][10], int nblignes)` Affichage de la diagonale (haut droit vers bas gauche) du tableau `t`.

4 Fonction manipulant un tableau à double entrées

4.1 Rappel sur les tableaux à entrées multiples

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 [5][10];
```

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

```
int t2 [50];
```

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*10+2]
```

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

Le tableau est donc « saucissonné » en tranches de 10 cases. En général (mais ce n'est pas toujours le cas) on décide que le tableau à double entrée représente donc une matrice. En général on considère que chaque « tranches » représente une « ligne » et que. Dans ce cas on accède à la i^e case de la j^e ligne en écrivant `t2[j][i]`, autrement dit le premier indice représente l'ordonnée et le deuxième l'abscisse (attention donc c'est un peu contre-intuitif).

N'oubliez cependant pas que c'est un choix virtuel : il n'y a pas de ligne ou de colonne dans la mémoire, il n'y a que des cases mémoire consécutives. Libre à vous de considérer que les « tranches » sont en fait des colonnes. L'important est qu'une fois le choix de considérer les tranches comme l'un ou l'autre, tout le reste du programme (les procédures d'affichage, de saisie de case, de calcul en colonne et en ligne etc) doit être cohérent avec cette convention.

4.2 Exercices

Programmez et testez les fonctions et procédures ci-dessous.

1. **void** affichenx10(**int** t[][10], **int** nblignes) Affichage du tableau t de taille $10 \times$nblignes. Par exemple :

```
1 2 3 4 5 6 7 8 9 0
1 2 3 4 5 6 7 8 9 0
1 2 3 4 5 6 7 8 9 0
1 2 3 4 5 6 7 8 9 0
1 2 3 4 5 6 7 8 9 0
```

   Astuce : pour afficher un entier n avec un nombre minimal de caractères de 3 à utiliser, utiliser printf("%3d",n) (nécessite **#include**<stdio.h>)

2. **void** affichenx10Transpose(**int** t[][10], **int** nblignes) Affichage du tableau t de taille $10 \times$nblignes en renversant les lignes et les colonnes.
3. **void** afficheDiagonale(**int** t[][10], **int** nblignes) Affichage de la diagonale (haut gauche vers bas droit) du tableau t de taille $10 \times$nblignes.
4. **void** afficheDiagonale2(**int** t[][10], **int** nblignes) Affichage de la diagonale (haut droit vers bas gauche) du tableau t de taille $10 \times$nblignes.
5. **void** affichecolonne(**int** t[][10], **int** nblignes, **int** ncol) Affichage de la colonne col du tableau t de taille $10 \times$nblignes.
6. **void** cross(**int** t[][10], **int** nblignes) qui affiche les deux diagonales et les deux colonnes du milieu du tableau de taille $10 \times$nblignes.
7. BOOL testColonneZeros(**int** t[][10], **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[][10], **int** nblignes, **int** ncol) qui retourne la somme des éléments de la colonne ncol du tableau t de taille $10 \times$nblignes.

4.2.1 Aire d'une couronne

Cet exercice a pour but de calculer l'aire d'une couronne, c'est-à-dire l'aire comprise entre deux disques de même centre mais de rayons différents, et de travailler sur la notion d'hypothèses.

— Donner une définition ainsi qu'un jeu de tests de la fonction aire_disque qui calcule l'aire πr^2 d'un disque de rayon r. Remarque : En C, la constante π est déjà définie dans le module math. Pour l'utiliser, il faut déclarer l'utilisation de ce module en tête du programme avec l'instruction suivante (déjà présente dans le squelette) :

```
#include<math.h>
```

   La constante π peut alors être utilisée :

```
ecrireDouble(M_PI);
```

— Donner une définition ainsi qu'un jeu de tests de la fonction aire_couronne qui, étant donnes deux nombres r1 et r2, calcule l'aire de la couronne de rayon intérieur r_1 et de rayon extérieur r_2. Par hypothèse, on considère que le rayon intérieur est inférieur ou égal au rayon extérieur.

4.2.2 Calcul de fonctions polynomiales

Cet exercice a pour but de définir des fonctions de calcul de polynômes. On souhaite mettre l'accent sur l'efficacité des algorithmes utilisés (minimisation du nombre de multiplications).

— Après avoir spécifié le problème, écrire un jeu de tests et donner une définition de la fonction polynomiale telle que `polynomiale(a,b,c,d,x)` rend la valeur de $ax^3 + bx^2 + cx + d$. Quel est le nombre de multiplications effectuées par votre définition ? Peut-on faire mieux ? Si oui, proposer une version plus efficace, sinon justifiez.

— Après avoir spécifié le problème, écrire un jeu de tests et donner une définition de la fonction `polynomial_carre` qui rend la valeur de $ax^4 + bx^2 + c$. Quel est le nombre de multiplications effectuées par votre définition ? Peut-on faire mieux ? Si oui, proposer une version plus efficace, sinon justifiez.

4.2.3 Calcul du nième nombre de Fermat

Les nombres de Fermat sont des nombres qui s'écrivent sous la forme $F_n = 2^{2^n} + 1$. Ces nombres sont premiers jusqu'à F_4, non premiers de F_5 à F_{32}, on connaît peu la primalité de ces nombres au delà. Le but de cet exercice est d'écrire un programme permettant de calculer, étant donné une valeur n quelconque, le $n$ième nombre de Fermat.

— Donner une définition et un jeu de tests de la fonction `fermat` qui calcule F_n.

— Testez votre fonction sur 5 et 7. Si les résultat sont étrange discutée en avec le prof !

— Proposez dans la fonction `main` un test (**`if`** `(...)`) qui affiche `"est divisible par 642"` si F_5 est divisible par 641, et `"n'est pas divisible par 642"` sinon.

— Proposez une même expression pour la divisibilité par 641.

— La même méthode fonctionne-t-elle pour F_7 ou F_{11} ?

4.2.4 Coût d'une excursion

Le but de cet exercice est de réfléchir à la division entière en écrivant un programme implémentant une formule qui fait usage de cette notion de manière non triviale.

Une association propose des excursions à la journée. Ses frais incluent :

— le transport en autocars de 60 places facturé 1200 euros la journée par autocar.

— le salaire de guides touristiques, à raison d'un guide pour 18 personnes maximum, facturé 300 euros la journée par guide.

Donner la définition et un jeu de tests de la fonction `excursion` qui étant donné un entier `nb_pers` calcule le coût (minimum) pour l'association d'une excursion de `nb_pers` personnes.

Remarque : Tout autocar contenant au moins une personne (de même, tout guide affecté à au moins une personne) doit être payé en totalité.