CCI - Principes des langages de programmation
TD 5 : Structures itératives

François Yvon[*], Thomas Tang[†]

3 octobre 2007

Première partie
Pour bien assimiler ...

1 Tableaux

Un tableau permet de représenter une collection d'objets de même type, quel qu'il soit. On peut les utiliser sans avoir à en déclarer la forme comme avec les structures.

1.1 Tableaux à une dimension

1.1.1 Déclaration

Pour déclarer qu'une variable est un tableau d'éléments de type *type*, on utilise la syntaxe suivante :

```
type nom_tableau[dim1]}
```

où *dim1* représente le nombre de cases dans le tableau.

```
int table[5] : tableau de 5 entiers (type int)
```

1.1.2 Accès

Pour accéder aux valeurs d'un tableau, on utilise toujours les crochets de la façon suivante : x[i] où i est une position dans le tableau. **Attention ! La première case d'un tableau est à la position 0, et la dernière case se trouve à la position n - 1.**

```
int main() {
    int table[5] ;
    table[0] = 2 ; // 1ère case, position 0
    table[1] = 4 ; // 2ème case, position 1
    table[2] = 6 ; // 3ème case, position 2
    table[3] = 8 ; // 4ème case, position 3
    table[4] = 10 ; // 5ème case, position 4
}
```

La variable *table* est un tableau à 5 cases de la forme :

Case	1	2	3	4	5
Position	0	1	2	3	4
Accès	table[0]	table[1]	table[2]	table[3]	table[4]
Valeur	2	4	6	8	10

[*] yvon@limsi.fr
[†] tang@cgm.cnrs-gif.fr

1.2 Tableaux à plusieurs dimensions

On peut faire des tableaux à plusieurs dimensions.

```
int main() {
    int table[3][2] ;
    table[0][0] = 1 ;
    table[0][1] = 2 ;
    table[1][0] = 3 ;
    table[1][1] = 4 ;
    table[2][0] = 5 ;
    table[2][1] = 6 ;
}
```

La variable *table* est un tableau à deux dimensions : une dimension de taille 3, une dimension de taille 2. En notant *table[i][j]* les accès aux valeurs de *table*, avec i indice entier qui varie de 0 à 2 et j de 0 à 1, le tableau suivant récapitule les déclarations.

i \ j	0	1
0	1	2
1	3	4
2	5	6

2 Structure itérative FOR

En métalangage, la structure for s'écrit de la façon suivante :

```
POUR <variable contrôle de boucle> VARIANT DE < init> A <fin>
    TRAITEMENT
FIN POUR
```

En langage C, la syntaxe est donc la suivante :

```
for (instruction1;expression;instruction2) {
    TRAITEMENT
}
```

- *instruction1* précise l'initialisation de la boucle.
- *instruction2* est exécutée à la fin de chaque boucle.
- *traitement* est répété tant que *expression* reste vraie.

Voici un exemple :

```
int main() {
    int i,j=0;
    /* calcule la somme des 100 premiers entiers */
    for (i = 0; i <100; i++) {
        j = j + i ;
    }
    printf("somme de 0 à 100 : %i",j) ;
}
```

Exercice 1

Définir un tableau d'entiers une dimension de taille 20 et le remplir des entiers de 0 à 19 à l'aide d'une première boucle for. A l'aide d'une seconde boucle for et du tableau précédent, calculer le produit des entiers de 0 à 19 et l'afficher. A l'aide d'une troisième boucle for, afficher le tableau précédent.

3 Structure itérative WHILE

3.1 Structure TANT QUE

En métalangage, la structure while s'écrit de la façon suivante :

```
TANT QUE <expression> FAIRE
    TRAITEMENT
FIN TANT QUE
```

En langage C, la syntaxe est donc la suivante :

```
while ( expression ) {
    TRAITEMENT
}
```

Tant que *expression* est vraie (c'est-à-dire tant que *expression* est différent de zéro), on répète le traitement, sinon on sort de la boucle. Dans ce cas, l'expression est testée **avant** le traitement.

Exercice 2

Reprendre l'exercice 1 en utilisant des boucles while.

Exercice 3

*Définissez un tableau d'entiers à 2 dimensions de taille 5*5 et remplissez-le avec les entiers de 0 à 24 de façon à avoir :*

```
tab[0][0]=0;
tab[0][1]=1;
tab[0][2]=2;
...
tab[1][0]=5;
```

Exercice 4

Ecrire un programme qui lit un caractère en entrée et :
- Affiche "initiale" si la touche 'i' a été pressée
- Affiche "essentielle" si la touche 'e' a été pressée
- Affiche "plenitude" si la touche 'p' a été pressée
- Quitte le programme si la touche 'q' a été pressée
- Reboucle tant qu'aucune des 4 autres touches n'ait été frappée.

3.2 Structure REPETER... TANT QUE

En métalangage, la structure do... while s'écrit de la façon suivante :

```
REPETER
    TRAITEMENT
TANT QUE <expression> est vraie
```

En langage C, la syntaxe est donc la suivante :

```
do {
    TRAITEMENT
} while ( expression ) ;
```

Dans cette structure "do... while", l'expression est testée **après** le traitement. Ce qui veut dire que dans ce type de boucle, **le traitement est exécuté au moins une fois**, même si dès le départ, l'expression est fausse.

Exercice 5

Utilisez une boucle do while pour calculer factorielle de 10. Rappel : factorielle de n, notée !n, vaut n!(n-1) si n différent de 0 et 1 sinon.*

Deuxième partie
Pour approfondir ...

Exercice 6

Créer un tableau de 20 caractères nommé *chaine*. Demandez à l'utilisateur de la remplir en utilisant *scanf* :

```
scanf("\&s",chaine); ATTENTION A LA SYNTAXE !
```

- Affichez *chaine* en partant de la fin pour remonter au début.
- Affichez *chaine* en ne conservant que les consonnes et y.

Exercice 7

On se propose de transformer un tableau à 2 dimensions en tableau à 1 dimension, pour ce faire, on va utiliser les règles suivantes :

- Le tableau initial ayant pour taille x et y, le tableau à une dimension aura pour dimension x * y.
- La case [i][j] se retrouve dans la case [y * i + j]
- A l'aide de deux boucles while imbriquées, transformer le tableau de l'exercice 3 en un tableau à 1 dimension.

Exercice 8

Ecrivez un programme demandant à l'utilisateur de deviner un nombre entier (!) entre 0 et 100 choisi aléatoirement par l'ordinateur. A chaque tentative, il faudra dire si le nombre proposé est plus petit ou plus grand que le nombre cherché. On indiquera le nombre d'essais après avoir trouvé la solution. Utilisez la fonction rand() qui rend un entier aléatoire (déclarée dans stdlib.h).

```
srand(time(NULL)); //initialiser le générateur à partir du compteur
de temps, pour qu'il soit plus aléatoire
int solution=(int)((double)rand() / ((double)RAND_MAX + 1) * 100)
```

Exercice 9

En utilisant des boucles while,

- convertissez un nombre décimal en hexadécimal.
- calculez le PGCD de deux nombres.