Université de Nice-Sophia Antipolis — mardi 9 janvier 2018
Polytech ELEC3 — Durée: 2h

Boucle for et tableaux
Travaux Pratiques – Séance n° 7

---

1 Boucle for

D'un point de vue algorithmique, mais aussi dans beaucoup de langages de programmation, l'énoncé **for** (`pour` en français) est un énoncé *itératif* qu'on utilise lorsque le nombre d'itérations est connu à l'avance, c'est-à-dire que l'énoncé ne comporte pas de condition d'arrêt de la boucle.

Malheureusement, comme nous allons le voir, l'énoncé **for** du langage C n'est pas un véritable énoncé `pour`, car la condition d'arrêt doit apparaître explicitement. La syntaxe de l'énoncé **for** est la suivante :

```
for (exp1; exp2; exp3) E
```

où *exp1*, *exp2* et *exp3* sont des expressions quelconques, éventuellement vides !

Sous sa forme la plus habituelle, *exp1* correspond à l'initialisation de la boucle d'incrémentation, *exp2* correspond au test de fin de boucle et *exp3* à l'incrémentation de la boucle.

Par exemple :

```
for (int i=0; i<6; i++) printf("%d,", i);
```

Cette boucle affiche 6 fois la valeur de i, c'est-à-dire 0, 1, 2, 3, 4, 5. Elle commence à i=0, vérifie que i est bien inférieur à 6, *etc.*. jusqu'à atteindre la valeur i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours.

Un énoncé **for** est en fait équivalent à l'énoncé **while** suivant :

```
exp1;
while (exp2) {
    E;
    exp3;
}
```

Comme pour chaque énoncé itératif que vous écrivez, il faudra dans la mesure du possible chercher d'abord l'invariant de boucle, et comme l'énoncé **for** de C ne garantit pas la finitude de la boucle, vous devrez vous assurer que la condition de sortie est bien vérifiée.

Toutefois, la syntaxe de l'énoncé **for** est plus compacte et on l'utilisera lorsqu'un, d'un point de vue algorithmique, le nombre d'itérations est déjà connu avant d'entrer dans la boucle. Ainsi, la boucle :

```
int nb_iterations=10;
for (int i=0; i<nb_iterations; i++)  printf("%d", i);
```

et équivalente à la boucle suivante :

```
int nb_iterations = 10, i=0;
while (i<nb_iterations) {
     printf("%d", i);
     i++;
}
```


2 Les tableaux (à 1 dimension)

On appelle *tableau* un objet formé de composants (des données) de même type, rangé de manière contiguë en mémoire (les unes à la suite des autres). Chacun des composants est accessible de façon individuelle et directe grâce à un indice.

Un tableau est donc une suite de cases (espace mémoire) de même taille. La taille de chacune des cases est conditionnée par le type de donnée que le tableau contient.

2.1 Déclaration

En langage C, la syntaxe de la déclaration d'une variable de type tableau unidimensionnel est la suivante :

```
type nom_du_tableau [nombre_d_éléments]
```

- `type` définit le type des éléments du tableau, c'est-à-dire qu'il définit la taille d'une case du tableau en mémoire. Le type des éléments est quelconque.
- `nom_du_tableau` est le nom de la variable de type tableau. Le nom du tableau suit les mêmes règles qu'un nom de variable.
- `nombre_d_éléments` est un nombre entier positif > 1 qui détermine le nombre de d'éléments du tableau.

Voici par exemple la déclaration d'un tableau `t` à 8 éléments de type **char** :

```
char t[8];
```

Jusqu'à la norme C99, le nombre d'éléments spécifié dans une déclaration de tableau devait être constant, et donc connu à la compilation. La norme C99 permet de déclarer des tableaux de taille variable comme dans l'exemple suivant :

```
int n;
char t[n];
scanf("%d", &n);
....
```

2.2 Accéder aux éléments

L'accès à un élément d'un tableau se fait par l'intermédiaire du nom du tableau et d'un *indice*. La notation `t[i]` désigne l'élément d'indice `i` du tableau `t`.

L'indice `i` est une *expression entière* à valeurs prises sur l'intervalle $0..n-1$ où n est le nombre d'éléments du tableau `i`. Ainsi pour le tableau `t` précédent à 8 éléments de type **char**, le 1er élément est à l'indice 0, le second à l'indice 1, ... et le dernier à l'indice 7.

- l'indice du premier élément du tableau est 0 ;
- in indice est toujours positif ;
- l'indice du dernier élément du tableau est égal au nombre d'éléments - 1.

Ainsi, on accédera au 5ème élément du tableau en écrivant `t[4]`.

Attention : vérifiez que 4 est compris entre 0 et 7 est assez simple, mais souvent, vérifiez que l'indice est valide est bien plus difficile. Puisque, l'indice peut être une expression quelconque, on pourrait très bien écrire `t[x*x*+2-y]`. Comme le langage C ne fait aucune vérification sur la validité de l'indice, *i.e.*aucun message d'erreur n'apparaîtra ni à la compilation ni à l'exécution, pour signaler que l'indice est hors des bornes, c'est à vous de vous assurer de façon analytique que l'indice est valide.

2.3 Manipuler les éléments

Un élément du tableau (repéré par le nom du tableau et son indice) peut être manipulé exactement comme une variable. On peut donc effectuer des opérations avec des éléments de tableau comme avec n'importe quelles variables simples. Par exemple, définissons un tableau de 10 entiers :

```
int t[10];
```

Pour affecter la valeur 6 au huitième élément on écrira :

```
t[7] = 6;
```

Pour affecter au 10ème élément le résultat de l'addition des éléments 1 et 2, on écrira :

```
t[9] = t[0] + t[1];
```

2.4 Initialiser les éléments

Lorsque l'on définit un tableau, les valeurs des éléments qu'il contient ne sont pas définies, il faut donc les initialiser, c'est-à-dire leur affecter une valeur. Une méthode simple consiste à affecter des valeurs aux éléments un par un :

```
t[0]=t[1]=t[2]=1;
```

L'intérêt de l'utilisation d'un tableau est alors bien maigre. Une manière plus élégante consiste à utiliser le fait que pour passer d'un élément du tableau à l'élément suivant il suffit d'incrémenter son indice. Il est donc possible d'utiliser une boucle qui va permettre d'initialiser successivement chacun des éléments grâce à un indice de boucle, souvent appelé conventionnellement `i`.

```
int t[10];
for (int i=0; i<10; i++)  t[i]=1;
```

Pour initialiser un tableau avec des valeurs spécifiques, il est également possible d'initialiser le tableau à la définition en plaçant entre accolades les valeurs, séparées par des virgules :

```
int t[10] = {1, 2, 6, 5, 2, 1, 9, 8, 1, 5};
```

- le nombre de valeurs entre les accolades ne doit pas être supérieur au nombre d'éléments du tableau.
- si le nombre de valeurs entre accolades est inférieur au nombre d'éléments du tableau, les derniers éléments sont initialisés à 0.

Notez que la déclaration suivante permet d'initialiser tous les éléments du tableau `t` à zéro :

```
int t[10] = {};
```

Enfin si le nombre d'éléments n'est pas spécifié, il sera ajusté automatiquement par le compliateur au nombre de valeurs initiales spécifiées dans la déclaration :

```
int t[] = {-1, 6, -10}; // tableau de 3 entiers
```

Il est fortement conseillé d'employer le plus possible des *identificateurs de constante* dans vos programmes, notamment pour la taille des tableaux. Le code ci-dessus peut s'écrire ainsi :

```
#define NB_ELEMENTS 10
int t[NB_ELEMENTS];

for (int i=0; i<NB_ELEMENTS; i++) t[i]=0;
```

Voici les avantages liés à l'utilisation de constantes :

- pour modifier la taille du tableau il suffit de changer le `define` en début du code source.
- le code possède une lisibilité accrue.

3 Paramètres tableaux d'une fonction

L'en-tête de la fonction `f` suivant comporte un paramètre de type tableau de réels.

```
void f(double t[])
```

Notez qu'entre les crochets le nombre d'éléments n'est pas précisé. Imaginons que l'on veuille écrire une fonction qui initialise des tableaux d'entiers de telle façon que chaque t_i soit égal à i. On écrira :

```
/* Rôle : initialise les éléments de t tels que t[i] = i */
void initTab(int t[], int n) {
   for (int i; i<n; i++) t[i]=i;
}
```

Notez que puisqu'on ne connaît pas le nombre d'éléments de `t`, il est nécessaire d'avoir un second paramètre qui le précise. En fait, le nom `t` ne désigne pas tous les éléments du tableau, mais uniquement le premier élément. *Nous reviendrons sur cet aspect ultérieurement.*

Le code suivant utilise la fonction `initTab` précédente pour l'initialisation de deux tableaux `t1` et `t2`.

```
#define N1 9
#define N2 150
int main(void) {
     int t1[N1];
     int t2[N2];

     initTab(t1,N1);
     initTab(t2,N2);
     return EXIT_SUCCESS;
}
```

Pour les exercices suivants, vous devrez écrire les invariants de boucle et la preuve de la finitude.

1) Écrivez un programme qui calcule la somme des N premiers termes de la série suivante : $1 + 1/2 + 1/3 + ... + 1/N$.

2) Écrivez un programme qui affiche la liste des valeurs de la fonction $f(x) = x^3/2$ sur un intervalle d'entiers qui sera lu sur l'entrée standard par l'utilisateur.

3) Écrivez une fonction qui initialise un tableau de N entiers de façon aléatoire à l'aide de la fonction `rand`. Le tableau est passé en paramètre de la fonction.

4) Complétez le programme précédent avec la fonction `max` qui renvoie la valeur maximale du tableau. Vous utiliserez cette fonction dans la fonction `main` pour écrire sur la sortie standard la valeur maximale du tableau.

5) Écrivez une fonction qui lit N notes (des réels doubles) sur l'entrée standard et qui les mémorise dans un tableau. Écrivez une seconde fonction qui calcule la moyenne des notes contenues dans le tableau. Le tableau est passé en paramètre à chaque fois. Testez vos fonctions.

6) Écrivez le programme *tac* qui écrit les lignes de texte lues sur l'entrée standard en inversant l'ordre des caractères de chaque ligne sur la sortie standard. Exemple :

en entrée :	*en sortie :*
`123456 xyz`	`zyx 654321`
`abcdefghifk`	`kfihgfedcba`