Université Paris-Nord — L1
Institut Galilée — Année 2011-2012
Licence 1 - 1er semestre

Travaux dirigés 4 : la structure de contrôle `for`

L'objectif de ce TD est de vous familiariser avec la notion d'itération en programmation. On parle communément de "boucle". Cette notion sera illustrée sur des problèmes de comptage et de répétition d'actions.

Correction. Note aux chargés de TD.

- En cours, ils ont vu des notions de compilation (analyse lexicale, syntaxique, sémantique) et la structure de contrôle itérative `for`. Sa sémantique a été donnée en langage machine.
- Nous poursuivons les exercices type. Ils doivent savoir résoudre/reproduire les exos marqués exercices type et **faire leur trace sur un exemple quelconque**.
- Le code leur est toujours donné avec les commentaires, en suivant scrupuleusement l'indentation choisie. Ils doivent bien comprendre que le code et les commentaires sont indissociables. N'hésitez pas à ajouter des commentaires en fonction des difficultés rencontrées dans votre groupe.
- Insister sur la pertinence des noms des variables. Une déclaration par ligne, pour pouvoir commenter la variable.

1 Itération : l'instruction `for`

1.1 Rappel

Soit le programme suivant :

```
1  /* déclaration de fonctionalités supplémentaires */
2  #include <stdlib.h> /* EXIT_SUCCESS */
3  #include <stdio.h> /* printf */
4
5  /* déclaration constantes et types utilisateurs */
6
7  /* déclaration de fonctions utilisateurs */
8
9  /* fonction principale */
10 int main()
11 {
12     /* déclaration et initialisation variables */
13     int i; /* var. de boucle */
14
15     for(i = 0;i < 5;i = i + 1)
16     {
17         printf("i = %d\n",i);
18     }
19     /* i >= 5 */
20
21     printf("i vaut %d après l'exécution de la boucle.\n",i);
22
23     return EXIT_SUCCESS;
24 }
```

```
25
26  /* implantation de fonctions utilisateurs */
```

1. Quelle est la signification de chaque argument du `for` ? Quelles instructions composent le corps de la boucle ?

   Correction. Vu en cours. Tout est expression en C (avec effet de bord) mais on ment ici et on considère que ce sont des instructions. Ne sert a rien de les embrouiller et en plus c une mauvaise pratique de les utiliser comme expressions (source d'erreurs, de confusion).

   ```
   for(instruction1; expression_booléenne; instruction2)
   {
       corps de la boucle : le bloc défini par la séquence d'instructions entre {}
   }
   ```

   – instruction1 : sert à initialiser la variable de boucle
   – expression_booléenne : (s'évalue à vrai ou faux) si vrai exécute le corps de boucle, sinon passe a l'instruction suivant la boucle. On parle parfois de la garde de la boucle.
   – instruction2 : prépare l'itération suivante ; il est important de comprendre que cette instruction doit forcément modifier la valeur de (au moins une variable, on ne leur parle que de la var de boucle) la variable intervenant dans l'expression booléenne SINON la val de expression_booléenne est constante et on ne sort jamais de la boucle

2. Faire la trace du programme. Qu'affiche le programme ?

   Correction. L'exécution de $i = i + 1$ est mise à la ligne 18 pour montrer que c à la fin du corps de la boucle, mm si c pas très clair.

   ```
   ligne          | i | affichage (sortie/écriture à l'écran)
   -------------------------------------------------
   initialisation | ? |
   15             | 0 |
   17             |   | i = 0
   18             | 1 |
   17             |   | i = 1
   18             | 2 |
   17             |   | i = 2
   18             | 3 |
   17             |   | i = 3
   18             | 4 |
   17             |   | i = 4
   18             | 5 |
   21             |   | i vaut 5 après l'exécution de la boucle.
   ```

   Il affiche :

   ```
   i = 0
   i = 1
   i = 2
   i = 3
   i = 4
   i vaut 5 après l'exécution de la boucle.
   ```

3. Modifiez le programme afin que la séquence affichée soit exactement (on passera à la ligne juste avant la fin du programme) :
   – `0 1 2 3 4`
   – `1 2 3 4`
   – `1 2 3 4 5`
   – `1 3 5`
   – `(0,0) (1,1) (2,2).`

Correction. Ici, les corrections peuvent se faire plus ou moins rapidement, en fonction de la compréhension des étudiants. Chaque séquence demande la modif d'un ou de plusieurs arguments afin d'insister sur leur rôle.

Pour `1 2 3 4 5`, c'est soit i < 6 ou i <=5. Ca dépend du problème.

```
/* correction (0,0) (1,1) (2,2)*/
/* déclaration de fonctionalités supplémentaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* déclaration constantes et types utilisateurs */

/* déclaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
    /* déclaration et initialisation variables */
    int i; /* var. de boucle */

    for(i = 0; i < 3; i = i + 1)
    {
        printf("(%d,%d) ",i,i);
    }
    /* i >= 3 */

    printf("\n");

    return EXIT_SUCCESS;
}

/* implantation de fonctions utilisateurs */
```

4. Modifiez le programme afin que la séquence affichée soit :
   – `0 1 2 0 1 2`.
   – `0 1 2 0 1 2 3`.
   De combien de boucles avez-vous besoin ? De combien de variables de boucles avez-vous besoin ?

   Correction. 2 boucles à la suite, 1 seule variable (on la réutilise)

5. Modifiez le programme afin que la séquence affichée soit :
   – `(0,0) (0,1) (0,2) (1,0) (1,1) (1,2) (2,0) (2,1) (2,2)`.
   De combien de boucles avez-vous besoin ? De combien de variables de boucles avez-vous besoin ? Quelle est la différence de structuration des boucles entre le point 4 et le point 5 ?

   Correction. C'est un produit cartésien : {0,1,2} x {0,1,2}. 2 boucles imbriquées, 2 variables. La différence, c'est 1) 2 à la suite ; 2) 2 imbriquées

```
/* déclaration de fonctionalités supplémentaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* déclaration constantes et types utilisateurs */

/* déclaration de fonctions utilisateurs */
```

```
/* fonction principale */
int main()
{
    /* déclaration et initialisation variables */
    int i; /* var. de boucle */
    int j; /* var. de boucle */

    for(i = 0;i < 3;i = i + 1)
    {
        for(j = 0;j < 3;j = j + 1)
        {
            printf("(%d,%d) ",i,j);
        }
        /* j >= 3 */
    }
    /* i >= 3 */

    /* passe a la ligne pour faire joli */
    printf("\n");

    return EXIT_SUCCESS;
}

/* implantation de fonctions utilisateurs */
```

1.2 Exercice type : calcul de $\Sigma_1^n i$

Écrire un programme qui calcule et affiche la somme des entiers de 1 à n : $\Sigma_1^n i$. n est un entier quelconque.

Correction.

```
/* déclaration de fonctionalités supplémentaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* déclaration constantes et types utilisateurs */

/* déclaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
    /* déclaration et initialisation variables */
    int somme = 0; /* élément neutre pour l'addition */
    int i; /* var. de boucle */

    for(i = 1;i <= 4;i = i + 1) /* i allant de 1 à 4 */
    {
        /* ajoute i à la somme partielle */
        somme = somme + i;
    }
    /* i > 4 */

    /* somme vaut 0 + 1 + 2 + 3 + 4 */
    printf("somme = %d\n",somme);
```

```
    return EXIT_SUCCESS;
}

/* implantation de fonctions utilisateurs */
```

1.3 Affichage de n fois "Coucou"

Écrire un programme qui affiche n fois la chaîne de caractères `"Coucou\n"`.

Correction. On n'introduit pas de var pour n car c'est une constante. Ils vont voir les constantes symboliques en C.

```
/* déclaration de fonctionalités supplémentaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* déclaration constantes et types utilisateurs */

/* déclaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
    /* déclaration et initialisation variables */
    int i; /* var. de boucle */

    for(i = 0;i < 5;i = i + 1)
    {
        printf("Coucou\n");
    }
    /* i >= 5 */

    return EXIT_SUCCESS;
}

/* implantation de fonctions utilisateurs */
```

1.4 Calcul de la somme d'une série d'entiers saisie par l'utilisateur

Écrire un programme qui demande à l'utilisateur combien d'entiers composent sa série, lit la série d'entiers et affiche la somme des valeurs de la série.

Rappel : l'instruction `scanf("%d", &a)` permet de réaliser une saisie utilisateur d'un entier dont la valeur sera affectée à la variable `a` (comme toute variable, `a` doit être préalablement déclarée).

Correction.

```
/* declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */

/* declaration constantes et types utilisateurs */

/* declaration de fonctions utilisateurs */

/* fonction principale */
```

```
int main()
{
    int n; /* taille de la serie a saisir par l'utilisateur*/
    int elt; /* un element de la serie a saisir par l'utilsateur */
    int somme = 0; /* somme de la serie a calculer */

    int i; /* var. de boucle */

    /* demande la taille de la serie a l'utilisateur */
    printf("Combien d'elements dans la série ? ");
    scanf("%d",&n);

    /* saisie serie (n entiers) et calcul incremental de la somme */
    for(i = 0; i < n; i = i + 1) /* chaque entier de la serie */
    {
        /* saisir sa valeur */
        scanf("%d",&elt);

        /* l'ajoute a la somme partielle */
        somme = somme + elt;
    }
    /* i >= n */

    /* somme contient la somme des elements de la serie. */
    printf("La somme des valeurs de cette serie est : %d\n",somme);

    return EXIT_SUCCESS;
}

/* implantation de fonctions utilisateurs */
```