Université Paris 13 — Éléments d'informatique
Institut Galilée — PC-SPI
Licence 1 - 1er semestre

Travaux dirigés 4 : structure de contrôle *if* et `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.

- Ils doivent savoir résoudre/reproduire les exos marqués exercices types et **faire leur trace sur un exemple quelconque**.
- On garde la procédure des TD précédents :
  - on se donne des exemples
  - on trouve un algorithme en francais
  - on traduit l'algorithme en C, en s'aidant de commentaires
  - on teste sur les exemples
- 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 Execution conditionnelle d'instructions : *if*

1.1 ✎✎ Exercice type : Le minimum de 3 valeurs

Soient 3 variables `a`, `b`, `c`, initialisées à des valeurs quelconques. Écrire un programme qui calcule et affiche à l'écran le minimum des 3 valeurs.

Correction. Nous donnons une version "standard" pour la recherche d'un extremum : une valeur par défaut. On parcourt les autres valeurs et on modifie en conséquence. Servira plus tard quand calcul extremum d'une série, bien qu'en général il soit plus difficile d'écrire le code initialisant l'extremum avec la première valeur de la série et on préfère initialiser avec une valeur limite (i.e. min = +infini).

```
algorithme :
soit min = a /* valeur par defaut */
si b < min /* b plus petit que min courant */
  /* b est le min courant */
  min = b
/* min contient min(a,b) */
si c < min /* c plus petit que min courant */
  /* c est le min courant */
  min = c
/* min contient min(min(a,b),c) = min(a,b,c) */
affiche min
```

2 Itération : l'instruction `for`

2.1 ✎✎ Exercice type : calcul de $\Sigma_{i=1}^{n} i$

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

Correction. Note aux chargés de TD.

— Faire remarquer l'utilisation de `#define` pour la valeur limite de la somme, Cela rend plus facile de changer cette limite. Insister sur le fait que cela est plus propre que mettre 4 directement dans la boucle. Aussi, le programme est plus lisible.

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

#define N 4 /* limite de la somme */

/* 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 > N */

    /* somme vaut 1 + 2 + 3 + ... + N */
    printf("somme = %d\n",somme);

    return EXIT_SUCCESS;
}

/* definitions des fonctions utilisateurs */
```

2.2 ✎✎✎ La suite de Fibonacci

La *suite de Fibonacci* est une suite d'entiers telle que :

1. les deux premiers termes de la suite sont 1 et 1 ;
2. chaque terme suivant est la somme des deux termes qui le précèdent.

Tout nombre de cette suite est dit un *nombre de Fibonacci*. Le premier nombre de Fibonacci est donc 1, le deuxième est 1, le troisième est $1 + 1 = 2$, le quatrième est $2 + 1 = 3$, etc. Voici ci après un tableau avec les premiers dix nombres de Fibonacci.

ordre	1	2	3	4	5	6	7	8	9	10
somme	1	1	$1+1$	$2+1$	$3+2$	$5+3$	$8+5$	$13+8$	$21+13$	$44+21$
nombre Fib.	1	1	2	3	5	8	13	21	44	65

Écrire un programme qui affiche les 20 premiers nombres de Fibonacci.

Correction. Note aux chargés de TD.

- Faire remarquer que à chaque itération il faut garder et mettre à jour deux valeurs.
- Faire remarquer que comme dans le cas de l'échange de deux valeurs, il faut faire gaffe quand on met à jour |avant_dern| et |dern| : il faut calcule le suivant avant d'écraser l'avant dernier.
- Attention aussi à l'utilisation de`#define`. Même remarques que pour l'exercice de la somme.
- Attention aussi au contrôle des itération. La valeur i correspond aux nombres de Fibonacci déjà calculés : on démarre donc à 2 et on termine à `N`, quand `i` devient égal à `N`, donc la boucle est itérée tant que `i < N`.

```
#include <stdio.h> /* printf */

/* déclaration constantes et types utilisateurs */
#define N 20 /* nombres de Fibonacci à afficher */

/* déclaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
    /* déclaration et initialisation variables */
    int dern = 1 , avant_dern = 1; /* dernier et avant dernier
                                      nombre de Fibonacci,
                                      on les initialise aus deux
                                      premières valeurs
                                    */
    int suiv; /* var auxiliare pour le nombre suivant */
    int i; /* var. de boucle */

    printf("%d %d ", avant_dern, dern);

    for(i = 2; i < N; i = i + 1)
    {
        suiv = avant_dern + dern;
        printf("%d ", suiv);
        avant_dern = dern;
        dern = suiv;
    }
    /* i >= N */

    return EXIT_SUCCESS;
}
```

Correction. Note aux chargés de TD.

- En cours, ils ont vu l'évaluation d'expressions booléennes. Les constantes symboliques TRUE et FALSE ont été introduites. Les expressions booléennes ont été présentées en utilisant les connecteurs logiques `&&`, ||.

3 Évaluation d'expressions booléennes

Soit le programme suivant :

```
#include <stdlib.h> /* EXIT_SUCCESS */
```

```
#include <stdio.h> /* printf */

#define FALSE 0
#define TRUE 1

/* Declaration de fonctions utilisateurs */

int main()
{
    int beau_temps = TRUE;
    int pas_de_vent = FALSE;

    printf("%d\n",beau_temps && pas_de_vent);
    printf("%d\n",beau_temps || pas_de_vent);
    printf("%d\n",! beau_temps || pas_de_vent);
    printf("%d\n",! (! beau_temps || pas_de_vent) == (beau_temps && ! pas_de_vent));

    return EXIT_SUCCESS;
}

/* Definition de fonctions utilisateurs */
```

1. Qu'affiche le programme ?

   Correction.

   ```
   0
   1
   0
   1 /* toujours vrai : théorème de De Morgan : NON (a OU b) = NON a ET NON b */
   ```

2. Modifiez le programme pour qu'il demande la valeur des booléens à l'utilisateur (0 pour FALSE, sinon TRUE).

   Correction.

   ```
   #include <stdlib.h> /* EXIT_SUCCESS */
   #include <stdio.h> /* printf, scanf */

   /* declaration de fonctions utilisateurs */

   int main()
   {
       int beau_temps; /* booléen */
       int pas_de_vent; /* booléen */

       /* avec saisie utilisateur */
       printf("Entrer la valeur des 2 booléens (0 pour FALSE et 1 pour TRUE).\n");
       scanf("%d",&beau_temps);
       scanf("%d",&pas_de_vent);

       printf("%d\n",beau_temps && pas_de_vent);
       printf("%d\n",beau_temps || pas_de_vent);
       printf("%d\n",! beau_temps || pas_de_vent);
       printf("%d\n",! (! beau_temps || pas_de_vent) == (beau_temps && ! pas_de_vent));

       return EXIT_SUCCESS;
   }
   ```

```
/* Definition de fonctions utilisateurs */
```

Exemples de sortie :

```
1038$ ./a.out
1 1
1
1
1
1
1039$ ./a.out
1 0
0
1
0
1
1039$ ./a.out
0 1
0
1
1
1
1039$ ./a.out
0 0
0
0
1
1
```

1. Que fait le programme suivant ?

```
1  /* declaration de fonctionnalites supplementaires */
2  #include <stdlib.h> /* EXIT_SUCCESS */
3  #include <stdio.h> /* printf, scanf */
4
5  /* declarations des constantes et types utilisateurs */
6
7  /* declarations des fonctions utilisateurs */
8
9  /* fonction principale */
10 int main()
11 {
12     int a;
13     double b;
14     char c;
15
16     printf("Entrez un nombre entier puis un nombre réel puis un caractère : ");
17
18     scanf("%d",&a);
19     scanf("%lg",&b);
20     scanf(" %c",&c);
21
22     printf("Vous avez saisi %d puis %g puis %c.\n",a,b,c);
23
24     return EXIT_SUCCESS;
25 }
26
27 /* definitions des fonctions utilisateurs */
```

Correction. Le programme :

- déclare 3 variables a,b et c, respectivement de type entier, réel (rationnel) et caractère ;
- demande à l'utilisateur de saisir 3 valeurs, respectivement de type entier, réel (rationnel) et caractère ;
- affecte la valeur entière saisie à la variable entière a (même type sinon quelle est la signification ? le compilateur détecte cette erreur sémantique lors de l'analyse sémantique mais il acceptera de compiler en faisant une conversion automatique de type => source de bugs difficiles à détecter)
- affecte la valeur réelle saisie à la variable réelle/rationnelle b ;
- affecte le caractère saisie à la variable caractère c ;
- affiche les valeurs pour montrer les affectations réalisées (vous pouvez utiliser un tel programme pour vérifier que les représentations sont bornées, cf. cours et TP)

2. Faire la trace du programme en considérant que l'utilisateur saisit au clavier : 1 puis "entrée", 12.2 puis "entrée" et 'c' puis "entrée" .

Correction.

```
ligne          | a      | b      | c      | affichage (sortie/écriture à l'écran)
-------------------------------------------------------------------------------
initialisation | ?      | ?      | ?      |
16             |        |        |        | Entrez un nombre entier puis un nombre réel
18             | 1      |        |        |
19             |        | 12.2   |        |
20             |        |        | 'c'    |
22             |        |        |        | Vous avez saisi 1 puis 12.2 puis c.
```

Ils l'ont vu en cours : vous pouvez leur faire remarquer que si la lecture du caractère s'était faite avec `"%c"`, la var c contiendrait `'\n'`, caractère qui suit la chaîne "12.2" (due à la mémoire tampon + scanf).