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

Travaux dirigés 6 : type booléen en C ; structures de contrôle *for* et *while*

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. Le "while" a été présenté et utilisé avec des expressions booléennes utilisant les connecteurs logiques `&&`, ||.
- La fin du TD (1h, 1h30) doit être consacré à la préparation du TP, qui est un peu long.

1 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
```

2 Boucles *for* ou *while* ?

Ces boucles ont exactement la même sémantique et on peut facilement réécrire l'une en l'autre. Par convention, on préfère utiliser la boucle *for* lorsque l'on connaît le nombre d'ité-

rations à l'avance ; on utilise *while* dans le cas contraire, lorsque le nombre d'itérations n'est pas connu à l'avance. Par exemple, pour faire la somme des éléments d'un tableau on utilisera *for* et pour trouver un élément dans un tableau, ne sachant pas où il se trouve, on choisira *while*. L'intérêt de respecter cette convention est que la lecture d'un programme est facilité en indiquant à quoi sert la boucle.

Résoudre les problèmes suivants en utilisant soit *for*, soit *while*.

2.1 Test d'égalité entre tableaux

Écrire un programme qui teste si deux tableaux de même `TAILLE` (constante symbolique) ont les mêmes données.

Correction.

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

#define FALSE 0
#define TRUE 1

#define TAILLE 5 /* taille des tableaux à comparer */

/* declaration de fonctions utilisateurs */

int main()
{
    int tab1[TAILLE] = {-2,10,-10,3,4};
    int tab2[TAILLE] = {-2,10,-10,3,3};
    int egaux = TRUE; /* TRUE si les tableaux sont égaux */
    int i; /* var. boucle */

    /* premiere case */
    i = 0;

    while(i < TAILLE && egaux) /* pas fin tableau et égaux */
    {
        if(tab1[i] != tab2[i]) /* pas égaux */
        {
            egaux = FALSE;
        }

        /* préparation itération suivante */
        i = i + 1; /* case suivante */
    }
    /* fin tableau ou pas égaux */

    if(egaux)
    {
        printf("Ils sont égaux.\n");
    }
    else
    {
        printf("Ils diffèrent.\n");
    }

    return EXIT_SUCCESS;
}
```

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

2.2 Nombre d'occurrences dans un tableau

Écrire un programme qui fait initialiser un tableau de taille `TAILLE` par l'utilisateur, demande à l'utilisateur un entier, et affiche le nombre d'occurrences de l'entier dans le tableau. Des exemples de sorties sont les suivants :

```
Saisissez 4 entiers : -2 -2 -2 3
Compte le nombre d'occurrences de quel entier ?
-2
Il y a 3 occurrences de -2 dans le tableau.

Saisissez 4 entiers : -2 -2 -2 3
Compte le nombre d'occurrences de quel entier ?
3
Il y a 1 occurrences de 3 dans le tableau.

Saisissez 4 entiers : -2 -2 -2 3
Compte le nombre d'occurrences de quel entier ?
0
Il y a 0 occurrences de 0 dans le tableau.
```

Correction.

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

#define TAILLE 4 /* taille du tableau utilisateur */

/* declaration de fonctions utilisateurs */

int main()
{
    int tab[TAILLE]; /* tableau a initialiser par l'utilisateur */
    int elt; /* l'elt a chercher */
    int nb_occ = 0; /* le nombre d'occurrences de elt dans tab */
    int i; /* var. de boucle */

    /* saisie de TAILLE entiers*/
    printf("Saisissez %d entiers : ",TAILLE);

    for(i = 0;i < TAILLE;i = i + 1) /* chaque case du tableau */
    {
        /* saisir sa valeur */
        scanf("%d",&tab[i]);
    }
    /* i >= TAILLE */

    /* demande a l'utilisateur de saisir l'entier à chercher */
    printf("Compte le nombre d'occurrences de quel entier ?\n");
    scanf("%d",&elt);

    /* compte le nombre d'occurrences */
    for(i = 0;i < TAILLE;i = i + 1) /* chaque case du tableau */
```

```
    {
        if(tab[i] == elt) /* trouvé */
        {
            /* un de plus */
            nb_occ = nb_occ + 1;
        }
    }
    /* i >= TAILLE */

    /* affiche résultats */
    printf("Il y a %d occurrences de %d dans le tableau.\n",nb_occ,elt);

    return EXIT_SUCCESS;
}

/* Definition de fonctions utilisateurs */
```

2.3 Élévation à la puissance

Écrire un programme qui demande à l'utilisateur d'entrer deux nombres entiers x et $n \geq 0$ puis calcule x^n et affiche le résultat.

Correction. Trivial (for).

2.4 Test de primalité

Écrire un programme qui demande à l'utilisateur d'entrer un nombre entier positif n, teste si n est premier puis affiche le résultat.

Correction. Le faire avec un while plutôt qu'un for.

```
#define TRUE 1
#define FALSE 0
...
int main()
{
  int premier = TRUE;
  int n;
  int d = 2;

  /* saisie */
  printf("n?");
  scanf("%d",&n);

  /* test de primalite */
  while (premier && (d < n))
  {
     if (n % d == 0)
     {
        premier = FALSE;
     }
     d = d + 1;
  }

  /* affichage */
  if (premier)
```

```
  {
    printf("%d est premier\n", n);
  }
  else
  {
    printf("%d n'est pas premier, car divisible par %d\n", n, d - 1);
  }


  return EXIT_SUCCESS;
}
```

3 Carré d'étoiles

Écrire un programme qui demande à l'utilisateur d'entrer un nombre entier positif n et affiche un carré creux d'étoiles de côté n. Exemple (l'utilisateur entre 4) :

```
n? 4
****
*  *
*  *
****
```

Correction.

```
  double for et un if comme ça :
 if ( (i == 0) || (j == 0) || (i == n - 1) || (j == n - 1) )
 {
   printf("*");
 }
 else
 {
   printf(" ");
 }
```