Programmation 3

L2 Informatique 2012-2013

Fiche de TD 2

Notions abordées : macro instructions ; effets de bord ; structures conditionnelles et de boucle.

Exercice 1. (Débogage conditionnel)

1. La macro `assert`, définie dans l'en-tête `assert.h`, sert à vérifier si les arguments d'une fonction respectent certains pré-requis. Par exemple,

```
1 double division(double x, double y){
2     assert(y != 0);
3     return x / y;
4 }
```

   permet de stopper l'exécution du programme si l'argument `y` vaut `0`. L'intérêt d'un tel procédé réside dans le fait que lorsque la condition testée par le `assert` est fausse, l'endroit du code qui a provoqué l'erreur peut être connu et le débogage du programme est ainsi facilité.

   Il existe cependant un problème : dans certains cas, les `assert` testent des conditions gourmandes en temps. Il est ainsi souvent préférable de pouvoir les désactiver pour gagner en efficacité.

   Définir une macro `ASSERT` qui, lorsqu'elle est utilisée à la place de `assert`, peut être désactivée. Pour cela, définir une constante `DEBOGUER`. En fonction de sa valeur, `1` ou `0`, la macro `ASSERT` délègue son argument à `assert` ou ne fait rien.

   Indication : utiliser les commandes `#if`, `#else` et `#endif` du préprocesseur.

2. Pour déboguer efficacement un programme, on souhaite afficher, pour chaque structure conditionnelle `if`, le nom du fichier `.c` qui le contient, sa ligne dans le code et si la condition testée est vérifiée ou non.

   Pour cela, définir une macro `IF` qui prend trois paramètres : une condition `cond`, un bloc d'instructions `bloc_then` et un bloc d'instructions `bloc_else`. Cette macro teste la condition `cond` et exécute `bloc_then` si `cond` est vraie, et `bloc_else` sinon. Elle affiche également que l'on exécute une structure `if`, le fichier qui la contient, sa ligne dans le code et si c'est `bloc_then` qui est exécuté ou bien `bloc_else`.

   Indication : utiliser les commandes `__LINE__` et `__FILE__` du préprocesseur qui permettent respectivement de connaître la ligne courante et le fichier courant.

Exercice 2. (Effets de bord)

1. Rappeler ce qu'est une *expression à effet de bord*.
2. Dire si les expressions suivantes sont à effet de bord :

(a)

```
2 + (8 * 2);
```

(b)

```
printf("Bonjour\n");
```

(c)

```
int a;
```

(d)

```
a = 2 + (8 * 2);
```

(e)

```
if (a == 17) {a;}
```

(f)

```
if (--a == 16) {a;}
```

3. Rappeler ce qu'est une *fonction à effet de bord*.
4. Dire si les fonctions suivantes sont à effet de bord :

(a)

```
int somme(int *tab, int n) {
    int i, res;

    res = 0;
    for (i = 0; i < n; i++)
        res += tab[i];
    return res
}
```

(b)

```
void afficher(int *tab, int n) {
    int i;

    for (i = 0; i < n; i++)
        printf("%d ", tab[i]);
}
```

(c)

```
void echanger(int *x, int *y) {
    int tmp;

    tmp = *x;
    *x = *y;
    *y = tmp;
}
```

(d)

```
1       int nb_appels = 0;
2
3       int fct_1(int n) {
4           nb_appels++;
5           return n + 1;
6       }
```

(e)

```
1       int nb_appels = 0;
2
3       int fct_2(int n) {
4           if (nb_appels == 0)
5               return 0;
6           else
7               return n + 1;
8       }
```

5. Expliquer ce qu'affiche le programme suivant et en quoi il n'est pas recommandable.

```
1  #include <stdio.h>
2
3  int main() {
4      int n;
5
6      n = 0;
7      n = etrange(&n, etrange(&n, 10));
8      printf("%d\n", n);
9
10     return 0;
11 }
12
13 int etrange(int *n, int m) {
14     *n += m;
15     return *n + 1;
16 }
```

Exercice 3. (Structures conditionnelles)

1. Rappeler ce qu'est un *bloc*.
2. Rappeler quelles sont les structures conditionnelles et expliquer leur utilité.
3. On suppose définies trois fonctions

```
1  int f();
2  int g();
3  int h();
```

qui retournent chacune `0` ou `1` en fonction du résultat d'un test qu'elles réalisent. On souhaite afficher un caractère en fonction des résultats obtenus, conformément au tableau suivant :

f	g	h	Caractère
0	0	0	`A`
0	0	1	`B`
0	1	0	`C`
0	1	1	`D`
1	0	0	`E`
1	0	1	`F`
1	1	0	`G`
1	1	1	`H`

Écrire un programme qui répond à ce problème avec les contraintes suivantes :
- utiliser des structures conditionnelles `if` ;
- il est interdit d'utiliser des variables ;
- toute exécution du programme appelle exactement une fois chacune des fonctions `f`, `g` et `h`.

4. Reprendre la question précédente avec les contraintes suivantes :
   - utiliser une structure conditionnelle `switch` ;
   - utiliser au plus une variable de type `int` ;
   - toute exécution du programme appelle exactement une fois chacune des fonctions `f`, `g` et `h`.

   Indication : utiliser des masques de bits.
5. Donner et expliquer la sortie produite par les programmes suivants :

   (a)

```c
#include <stdio.h>

int main() {
    int i;

    i = 2;
    switch (i) {
        case 1 : printf("1");
        case 2 : printf("2");
        case 3 : printf("3");
        case 4 : printf("4");
        default : printf("d");
    }

    return 0;
}
```

   (b)

```c
#include <stdio.h>
```

```
2
3  int main() {
4      int i;
5
6      i = 32;
7      switch (i) {
8          case 1 : printf("1");
9          case 2 : printf("2");
10         case 3 : printf("3");
11         case 4 : printf("4");
12         default : printf("d");
13         case 5 : printf("5"); break;
14         case 6 : printf("6");
15     }
16
17     return 0;
18 }
```

(c)

```
1  #include <stdio.h>
2
3  int main() {
4      int i;
5
6      i = 32;
7      switch (i) {
8          case 1 : printf("1");
9          case 2 : printf("2");
10         case 3 : printf("3");
11         case 4 : printf("4");
12         default : printf("d");
13         case 5 : printf("5"); break;
14         case 6 : printf("6");
15         case 32 : printf("32");
16     }
17
18     return 0;
19 }
```

6. Rappeler quelles sont les structures de boucle et expliquer leur utilité.
7. Expliquer dans quelles circonstances il est préférable d'utiliser une boucle `for` plutôt qu'une boucle `while` et réciproquement.
8. Écrire, en utilisant judicieusement des boucles `for`, `while` ou encore `do while`, les fonctions :

   (a)

```
1  void afficher_triangle_1(int n);
```

qui produit la sortie suivante (donnée ici dans le cas `n` = 4) :

```
*
* *
* * *
* * * *
```

(b)

```
1 void afficher_triangle_2(int n);
```

qui produit la sortie suivante (donnée ici dans le cas `n` = 4) :

```
* _ _ _
* * _ _
* * * _
* * * *
```

(c)

```
1 void afficher_damier(int n);
```

qui produit la sortie suivante (donnée ici dans le cas `n` = 4) :

```
* _ * _
_ * _ *
* _ * _
_ * _ *
```

(d)

```
1 void afficher_triangle_3(int n);
```

qui produit la sortie suivante (donnée ici dans le cas `n` = 6) :

```
*
* *
* * * *
* * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * * * * *
```

9. La *suite de Syracuse* est une suite $(S_i)_{i\geq 0}$ d'entiers dépendant d'un paramètre n définie de la manière suivante :

$$S_i := \begin{cases} n & \text{si } i = 0, \\ \dfrac{S_{i-1}}{2} & \text{si } S_{i-1} \text{ est pair,} \\ 3S_{i-1} + 1 & \text{sinon.} \end{cases}$$

Une conjecture énonce que pour tout entier $n \geq 1$, il existe un entier $i \geq 0$ tel que $S_i = 1$. Écrire un programme qui demande à l'utilisateur d'entrer au clavier un entier `n` et qui affiche les éléments de la suite $(S_i)_{i\geq 0}$ correspondante et s'arrête dès qu'un terme est égal à 1.