[Math-L312] TP 2 : Structures de contrôle en C

Adrien Semin
adrien.semin@math.u-psud.fr

Attention : cette feuille de TP servira également pour le TP3

1 Rappel de la syntaxe des structures de contrôle

1.1 Structures conditionnelles

La structure `if` permet d'effectuer une instruction si une condition est réalisée : on parlera de test. Elle peut prendre une clause `else` facultative qui sera exécutée si la condition n'est pas réalisée. La condition doit être une expression de type entier (ou convertible en entier), lorsque sa valeur est nulle la condition sera considérée comme non réalisée et toute valeur non nulle sera considérée comme une condition réalisée.

```
if (Expression) Instruction; /* Instruction sera exécutée si Expression ≠ 0
    */
if (Expression) Instruction1 else Instruction2; /* Instruction1 sera
    exécutée si Expression ≠ 0, sinon on exécute Instruction2 */
```

La structure `switch` permet de réaliser simplement une série de tests imbriqués. Si $Expression = Valeur1$ alors le programme exécutera `Instruction1`, si $Expression = Valeur2$ alors il exécutera `Instruction2`, etc... Si *Expression* n'est égale à aucune des valeurs de la liste des `case` alors le programme exécutera `InstructionDefault` (si toutefois la clause `default` facultative a été spécifiée, sinon il passe directement à la suite).

```
switch (Expression){
  case Valeur1 : Instruction1;
                 break; /* L'instruction break est obligatoire ici sinon le
                     programme exécutera aussi Instruction2 */
  case Valeur2 : Instruction2;
                 break;
  case Valeur3 : Instruction3;
                 break;
  /* ... */
  default : InstructionDefault;  /* Cette ligne est facultative */
}
```

1.2 Boucles

La boucle `for` est une boucle multi-usages. L'instruction `Initialisation` est d'abord exécutée, puis :

(a) si l'expression *Condition* est nulle alors le programme quitte la boucle, sinon on passe à l'étape (b) ;

(b) l'instruction `Instruction` est exécutée ;

(c) l'instruction `Transition` est exécutée ;

(d) retour à l'étape (a).

```
for (Initialisation;Condition;Transition){
  Instruction;
}
```

Exemple pour afficher les entiers de 0 à 99 (les accolades sont ici inutiles puisqu'il n'y a qu'une seule instruction dans le corps de la boucle) :

```
for (i=0;i<100;i++)
  printf("%i\n",i);
```

La boucle `while` effectue une instruction tant qu'une condition est vérifiée. Elle admet aussi une variante avec `do`, qui elle effectue l'instruction au moins une fois même si la condition est fausse dès le début.

```
while (Condition){
  Instruction; /* Instruction exécutée tant que Condition≠0 */
}

do { /* Variante */
  Instruction; /* Instruction exécutée tant que Condition≠0, mais le test a
       lieu à la fin de la boucle donc Instruction sera toujours exécutée au
       moins une fois */
} while (Condition);
```

Toute boucle peut être interrompue avec l'instruction `break` (auquel cas le programme sort de la boucle et exécute le code qui suit), et on peut forcer le retour au début de la boucle grâce à l'instruction `continue`. Toutefois on préférera, dans la mesure du possible, d'éviter de les utiliser car elles ont tendance à "déstructurer" le programme et à le rendre moins lisible (traduction : un programme utilisant cette instruction sera considéré comme "faux").

2 Exercices

1. `carre.c` ***À Rendre!*** Ecrire un programme qui demande un entier n strictement positif, et qui affiche un carré $n \times n$ comme suit (exemple avec $n = 5$) :

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

2. `triangle.c` ***À Rendre!*** Ecrire un programme qui demande un entier n strictement positif, et qui affiche un triangle supérieur $n \times n$ comme suit (exemple avec $n = 5$) :

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

3. `trace_v.c` ***À Rendre!*** Ecrire un programme qui demande un entier n stricement positif, et qui affiche un "V" sur n lignes comme suit (exemple avec $n = 5$) :

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

4. `isprime.c` ***À Rendre!*** Ecrire un programme qui demande un nombre entier strictement positif p et qui affiche si p est premier (on vérifiera que p n'est divisible par aucun k, avec k entre 2 et $\sqrt{p}$)

5. `fibonacci.c` ***À Rendre!*** Ecrire un programme qui demande un entier n positif et qui calcule le $n^{\text{ème}}$ terme de la suite de Fibonacci, défini par

$$u_0 = 1, \quad u_1 = 1, \quad u_{n+2} = u_{n+1} + u_n \tag{1}$$

6. `affprime.c` ***À Rendre!*** Ecrire un programme qui demande un entier strictement positif n et qui affiche les n premiers nombres premiers.

7. `pgcd.c` **∗ À Rendre! ∗** Ecrire un programme qui demande a et b entiers strictement positifs, et qui calcule le *pgcd* de a et de b par l'algorithme suivant :

$$pgcd(a,b) = \left\{ \begin{array}{lll} a & \text{si} & a = b \\ pgcd(a, b-a) & \text{si} & a < b \\ pgcd(a-b, b) & \text{si} & a > b \end{array} \right\} \quad (2)$$

8. `euclide.c` **∗ À Rendre! ∗** Ecrire un programme qui demande a et b entiers strictement positifs, et qui calcule le *pgcd* de a et de b par l'algorithme suivant :

$$pgcd(a,b) = \left\{ \begin{array}{lll} a & \text{si} & b = 0 \\ pgcd(b, a \mod b) & \text{sinon} & \end{array} \right\} \quad (3)$$

   Comparer cet algorithme par rapport à l'algorithme du programme `pgcd.c`. Que peut-on en déduire ?

9. `guess.c` **∗ À Rendre! ∗** Ecrire un programme qui choisit un nombre entre 1 et 1024, que l'utilisateur doit deviner avec des indications de type "plus grand" ou "plus petit". Retourner le nombre de coups utilisés par le joueur pour trouver. Quel est le nombre maximum de ces coups si l'utilisateur est futé ?

10. `guess2.c` **∗ À Rendre! ∗** On reprend l'exercice précédent, mais ici, c'est le programme lui-même qui montre les étapes de résolution (i.e. le programme choisit un nombre, puit "demande" si il est plus grand ou plus petit,...)

11. `tableau.c` **∗ À Rendre! ∗** Ecrire un programme qui déclare un tableau de taille 32 et qui :
    (a) le remplit avec des 1,
    (b) le remplit avec les carrés successifs (i.e. le $i^{\text{ème}}$ élément du tableau est i^2),
    (c) demande un nombre entier x à l'utilisateur, regarde si x est edans le tableau, retourne sa position si x est dans le tableau et -1 sinon,
    (d) échange la valeur de la première case avec la dix-septième case, de la deuxième case avec la dix-huitième...,
    (e) retourne le tableau
    (f) remplit le tableau avec les 32 premiers nombres premiers dans l'ordre décroissant.

    Pour chaque point, on affichera le tableau obtenu, et on indiquera en commentaires l'ordre de grandeur du nombre d'opérations effectuées si le tableau était de taille N

12. `dicetest.c` **∗ À Rendre! ∗** Ecrire un programme qui déclare un tableau de taille 10000 et qui
    (a) le remplit avec des nombres aléatoires entre 1 et 6,
    (b) compte le nombre d'apparitions de chaque chiffre,
    (c) compare, pour chaque chiffre, la probabilité que ce chiffre sorte par rapport à 1/6 (qui correspond à la probabilité que ce chiffre sorte sur un dé à 6 faces parfaitement équilibré). Le générateur de nombres aléatoires est-il un bon générateur ?