Algorithmique et structures de données

Chapitre 3 – Structures conditionnelles et répétitives

Samar MOUCHAWRAB, PhD Eng

2A Cycle Préparatoire – Semestre 1
2017/2018

Les structures conditionnelles

La forme la plus simple de structure conditionnelle est d'exécuter quelque chose dans le cas où une condition est vraie.

Pour cela, on utilisera la structure de contrôle if, dont la syntaxe est la suivante :

```
if ( condition )
    Instruction à exécuter si la condition est vraie
else
    Instruction à exécuter si la condition est fausse
```

Exercice 1

Ecrire un algorithme qui reçoit en entrée les coordonnées de 3 points et affiche le type du triangle formé (équilatéral, isocèle, rectangle ou autre).

Pour rappel:

La distance entre les points $A(x_1, y_1)$ et $B(x_2, y_2)$ est donnée par la formule suivante:
$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Exercice 1

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

int main() {

        // Définition des variables
        float xA, yA, xB, yB, xC, yC; // coordonnées des trois
points
        double AB, AC, BC; //Les distances des côtés

        //Lecture des coordonnées
        printf("Entrer l'abscisse du point A : ");
        scanf("%d", &xA);
        printf("Entrer l'ordonnee du point A : ");
        scanf("%d", &yA);
        printf("Entrer l'abscisse du point B : ");
        scanf("%d", &xB);
        printf("Entrer l'ordonnee du point B : ");
        scanf("%d", &yB);
        printf("Entrer l'abscisse du point C : ");
        scanf("%d", &xC);
        printf("Entrer l'ordonnee du point C : ");
        scanf("%d", &yC);
```

Exercice 1

```
//Calcul des distances
AB = sqrt(pow((xB - xA),2) + pow((yB - yA),2));
AC = sqrt(pow((xC - xA),2) + pow((yC - yA),2));
BC = sqrt(pow((xC - xB),2) + pow((yC - yB),2));

//Tests pour identification de type de triangle
if ((AB + AC == BC) || (AC + BC == AB) || (BC + AB == AC)) {
        printf("ABC n\'est pas un triangle.");
} else {
        if ((AB == AC) && (AC == BC)) {
                printf("Le triangle ABC est equilateral");
        } else if (AB == AC) {
                if ((pow(BC,2) == (pow(AC,2) + pow(AB,2))))
                        printf("Le triangle ABC est rectangle
                        et isocele en A");
                else
                        printf("Le triangle ABC est isocele en A");
        } else if (AC == BC) {
                if ((pow(AB,2)) == (pow(AC,2) + pow(BC,2)))
                        printf("Le triangle ABC est rectangle et
                        isocele en C");
                else
                        printf("Le triangle ABC est isocele en C");
```

Exercice 1

```
                } else if (AB == BC) {
                        if ((pow(AC,2)) == (pow(AB,2) + pow(BC,2)))
                                printf("Le triangle ABC est rectangle
                                et isocele en B");
                        else
                                printf("Le triangle ABC est isocele en
                                B");
                } else if ((pow(AB,2)) == (pow(AC,2) + pow(BC,2))) {
                        printf("Le triangle ABC est rectangle en C");
                } else if ((pow(AC,2)) == (pow(AB,2) + pow(BC,2))) {
                        printf("Le triangle ABC est rectangle en B");
                } else if ((pow(BC,2)) == (pow(AC,2) + pow(AB,2))) {
                        printf("Le triangle ABC est rectangle en A");
                } else {
                        printf("Le triangle ABC n\'est ni isocele ni
                        equilateral ni rectangle.");
                }
        }
        return 0;
}
```

Exercice 2

Ecrire un algorithme qui permet de calculer le salaire d'un employé payé à l'heure à partir de son salaire horaire et du nombre d'heures de travail.

Les règles de calcul sont les suivantes :

le taux horaire est majoré, pour les heures supplémentaires,

- de 25% au-delà de 160 heures,

- de 50% au-delà de 200 heures.

Exercice 2

```
#include <stdio.h>

int main() {
        float taux, salaire;
        int nombre_heures;

        printf("Entrer le taux horaire: ");
        scanf("%f", &taux);
        printf("Entrer le nombre d\'heures travailles: ");
        scanf("%d", &nombre_heures);

        salaire = nombre_heures * taux;

        if (nombre_heures > 200) {
                salaire = salaire + 40*0.25*taux + ((nombre_heures -
                        200)*taux* 0.5);
        } else if (nombre_heures > 160) {
                salaire = salaire +((nombre_heures - 160)*taux* 0.25);
        }
        printf("Le salaire a verser est : %.2f", salaire);

        return 0;
}
```

Structures répétitives

Les structures répétitives ou boucles permettent de répéter une série d'instructions tant qu'une certaine condition est vérifiée.

Boucle while

La syntaxe de `while` est la suivante :

```
while ( expression ) instruction(s)
```

Tant que `expression` est vérifiée, `instruction` est exécutée. Si expression est non vérifiée au départ, `instruction` ne sera jamais exécutée. `instruction` peut évidemment être une instruction composée. Par exemple, le programme suivant imprime les entiers de 1 à 9.

```
int i = 1;
while  (i < 10) {
       printf("i = %d\n",i); i++;
}
```

Structures répétitives

Boucle do -- while

Il peut arriver que l'on ne veuille effectuer le test de continuation qu'après avoir exécuté l'instruction. Dans ce cas, on utilise la boucle do while. Sa syntaxe est:

```
do

instruction(s)

while ( expression );
```

Ici, `instruction` sera exécutée tant que `expression` est vérifiée. Cela signifie donc que `instruction` est toujours exécutée au moins une fois. Par exemple, pour saisir au clavier un entier entre 1 et 10 :

```
int a;
do {
        printf("\n Entrez un entier entre 1 et 10 : ");
        scanf("%d",&a);
} while ((a <= 0) || (a >  10));
```

Structures répétitives

Boucle for

La syntaxe de for est :

```
for ( expr 1 ; expr 2 ; expr 3) instruction(s)
```

Une version équivalente plus intuitive est :

```
expr  1;
while ( expr 2 ) {
    instruction(s)
    expr 3;
}
```

Par exemple, pour imprimer tous les entiers de 0 à 9, on écrit :

```
for (i = 0; i < 10; i++)
       printf("i =%d\n ",i);
```

A la fin de cette boucle, i vaudra 10.

Structures répétitives

Boucle for

Les trois expressions utilisées dans une boucle for peuvent être constituées de plusieurs expressions séparées par des virgules. Cela permet par exemple de faire plusieurs initialisations à la fois. Par exemple, pour calculer la factorielle d'un entier, on peut écrire :

```
int i, fact; //n entier défini avant ce code
for (i = 1, fact = 1; i <= n; i++)
       fact *= i;
printf("%d ! = %d \n",n,fact);
```

On peut également insérer l'instruction `fact *= i;` dans la boucle for ce qui donne :

```
int i, fact;
for (i = 1,fact = 1;i <= n;fact *= i, i++);
printf("%d!= %d\n",n,fact);
```

A éviter toutefois ce type d'acrobaties qui n'apportent rien et rendent le programme difficilement lisible.

Exercice 3

Calcul du PGCD de deux entiers

On considère que pgcd(a,0) = a et que pour b ≠ 0 pgcd(a,b) = pgcd(b, a mod b). On progresse dans l'algorithme en diminuant à chaque étape les nombres considérés par calcul du modulo.

Si a < b, la première itération de la boucle a pour effet de « permuter a et b ».

Calculons, par exemple, le PGCD de 1071 et de 1029 à l'aide de l'algorithme d'Euclide :

$1071 = 1029 \times 1 + 42$

$1029 = 42 \times 24 + 21$

$42 = 21 \times 2 + 0$

Il faut prendre le dernier reste avant le zéro, donc PGCD(1071 ; 1029) = 21

Exercice 3

```
#include <stdio.h>
int main() {
        int a, b, r;
        printf("Entrer deux nombres entiers :\n");
        scanf("%d", &a);
        scanf("%d", &b);
        if (b==0) {
                if (a==0) {
                        printf ("Erreur!!");
                        return 0;
                } else {
                        printf("Le PDCD est %d", a);
                        return 0;
                }
        }
        while (b!=0) {
                r = a % b;
                a = b;
                b = r;
        }
        printf("Le PDCD est %d ", a);
        return 0;
}
```

Exercice 4

Écrire un algorithme utilisant la boucle do while permettant de calculer et d'afficher la somme suivante :

somme = 10 + 15 + 20 + 25 + ... + 50

```
#include <stdio.h>
int main() {
        const int BORNE1 = 10,
                  BORNE2 = 50,
                  LEPAS  = 5;
        int terme, somme;

        somme = 0;
        terme = BORNE1;
        do {
                somme += terme;
                terme += LEPAS;
        } while (terme <= BORNE2);

        printf("La somme calculee est : %d\n", somme);

        return 0;
}
```

Exercice 5

Le jeu du lièvre et de la tortue:

Règle du jeu.

À chaque tour, on lance un dé. Si le 6 sort, alors le lièvre gagne la partie, sinon la tortue avance d'une case.

La tortue gagne quand elle a avancé 6 fois.

Question : le jeu est-il à l'avantage du lièvre ou de la tortue ?

La fonction int rand(void) de la librairie stdlib retourne un nombre pseudo-random dans l'intervalle 0 à RAND_MAX qui est une constante définie dans stdlib.h.

La valeur de RAND_MAX peut varier suivant les compilateurs, mais elle est forcément d'au moins 32767.

Exercice 5

Les nombres aléatoires dans C:

L’ordinateur ne sait pas générer des nombres aléatoires, toutes les informations qu'il nous fournit sont calculées. Le hasard n'existe que dans la nature et n'a pas de sens en informatique. Alors, pour nous fournir des données aléatoires, l'ordinateur doit simuler le hasard. On parlera alors de données pseudo-aléatoires.

Les suites de nombres pseudo-aléatoires que peut nous fournir la fonction rand sont calculées à partir d'une donnée seed (graine). Si cette dernière n'est pas modifiée, la suite de nombres sera toujours la même. L'idée est donc d'initialiser cette donnée avec une valeur toujours différente, à chaque démarrage du programme, à l'aide de la fonction srand.

Exercice 5

Les nombres aléatoires dans C:

On peut initialiser la donnée seed avec la date actuelle. Il existe la fonction time qui renvoie le nombre de secondes entre l'instant où elle est appelée et le 01/01/1970. Pour l'appeler, il faut inclure le fichier d'en-tête time.h.

Donc au final, avant d’utiliser la fonction rand dans vos programmes pour générer des nombres pseudo-aléatoires, pensez à appeler la fonction srand(time(NULL));

Exercice 5

Le jeu du lièvre et de la tortue:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
        int de;
        int total_tortue = 0;
        srand(time(NULL));
        do {
                de = (rand()%6) + 1;
                printf("%d\n", de);
                if (de == 6) {
                        printf("Lievre gagne !");
                        return 0;
                } else {
                        total_tortue++;
                }
        } while (total_tortue <6);

        printf("Tortue gagne!");

        return 0;
}
```

Exercice 6

Décomposition d'un montant en euros

Écrire un algorithme permettant de décomposer un montant entré au clavier en billets de 20, 10, 5 euros et pièces de 2, 1 euros, de façon à minimiser le nombre de billets et de pièces.

L'idée consiste ici à déterminer dans un premier temps le nombre de billets de 20 euros nécessaires (qui correspond au quotient de la division du montant par 20) puis, pour la somme restante (à calculer…), le nombre de billets de 10 euros nécessaires et ainsi de suite.

Exercice 6

Exercice 6 : Décomposition d'un montant en euros => Solution

```
#include <stdio.h>
void main () {
        int montant, reste;
        int billets20, billets10, billets5, pieces2, pieces1;
        printf ("Entrer le montant :");
        scanf ("%d", &montant);
        billets20 = montant / 20;
        reste = montant % 20;
        billets10 = reste / 10;
        reste = reste % 10;
        billets5 = reste / 5;
        reste = reste % 5;
        pieces2 = reste / 2;
        reste = reste % 2;
        pieces1 =  reste;
        // affichage résultat;
        printf ( "Billets de 20 : %d\n",billets20);
        printf ( "Billets de 10 : %d\n",billets10);
        printf ( "Billets de 5 : %d\n",billets5);
        printf ( "Pieces de 2 : %d\n",pieces2);
        printf ( "Pieces de 1 : %d\n",pieces1);
}
```

Exercice 7

Somme de deux fractions

Écrire un algorithme permettant de calculer le numérateur et le dénominateur d'une somme de deux fractions entières (on ne demande pas de trouver la fraction résultat sous forme irréductible).