TD 3 : Structures de contrôles : instructions conditionnelles, boucles simples

Notes pour les enseignants :
- Ce TD/TP est essentiel : les conditionnelles et boucles sont des notions fondamentales, et difficiles pour certains.
- Leur faire ouvrir leur poly à la page if, puis while.
- Ne pas hésiter à demander d'abord l'algorithme en *français* avant de l'écrire en C++.
- De façon générale, quand on écrit du code au tableau, il est profitable de le prononcer en français en même temps (beaucoup d'élèves ne sont pas bons en anglais ne font pas l'association immédiate if -> si, while -> tant que, etc.).

Dans les exercices suivants, vous préciserez le nom, l'entrée et la sortie de vos programmes. Par exemple, pour un programme calculant le maximum de deux nombres, vous écrirez :

```
// Programme maximum
// Entrée: deux entiers a et b de type int
// Sortie: le maximum de a et de b, stocké dans la variable max
int max;
if ( a >= b ) {
    max = a;
} else {
    max = b;
}
```

Si vous vous sentez à l'aise, mettez votre programme sous la forme d'une fonction :

```
int maximum(int a, int b) {
    if ( a >= b ) {
        return a;
    } else {
        return b;
    }
}
```

Instructions conditionnelles : `if`

Exercice 1.

(1) Écrire un programme qui prend en entrée l'âge d'une personne (un entier donc) et dont la sortie est `true` si la personne est majeure et `false` sinon.

(2) Modifier votre programme pour que la sortie soit une chaîne de caractères (type `string`) contenant le tarif SNCF appliqué à cette personne : `"enfant"` (moins de 11 ans), `"jeune"` (12-27 ans), `"senior"` (60 ans ou plus) ou `"plein tarif"` sinon.

Correction : (1) Solution naïve :

```
// Entrée: un entier age
// Sortie: un booléen stocké dans la variable majeur, qui vaut
//         true si on est majeur, false sinon
bool majeur;
if ( age >= 18 ) {
    majeur = true;
} else {
```

```
    majeur = false;
}
```

Meilleure solution :

```
bool majeur = age >= 18;
```

(2)

```
// Entrée : un entier age
// Sortie : une chaîne de caractère stockée dans la variable tarif,
// qui indique la catégorie de voyageur.

if (age < 11) {
        tarif = "enfant";
} else if (age >= 12 and age <= 27) {
        tarif = "jeune";
} else if (age > 60){
        tarif = "senior";
} else {
        tarif = "plein tarif";
}
```

Exercice 2 (À faire chez vous)**.**

(1) Le magasin Honeydukes est ouvert de 10h à 12h et de 14h à 19h. Écrire un programme qui prend en entrée l'heure sous forme d'un entier et dont la sortie est `true` si le magasin est ouvert et `false` sinon.

(2) Si ce n'est pas déjà le cas, modifier votre programme pour qu'il y ait au plus un `if`.

(3) Si ce n'est pas déjà le cas, modifier votre programme pour qu'il n'y ait pas de `if`. Indication : remarquer que l'expression booléenne qui sert de condition au `if` a déjà la valeur recherchée (voir aussi le cours : « Erreurs classiques avec les conditionnelles »).

Correction :

```
bool ouvert(int heure) {
    if ((heure >= 10 and heure <= 12) or (heure >= 14 and heure <= 19)) {
        return true;
    } else {
        return false;
    }
}
```

Ce qui s'écrit, de manière beaucoup plus rapide et plus simple :

```
bool est_ouvert(int heure) {
    return (heure >= 10 and heure <= 12) or (heure >= 14 and heure <= 19);
}
```

Instructions itératives : boucle `while` et `for`

Exercice 3 (Programme mystère)**.**

(1) Exécuter pas à pas le fragment de programme suivant pour $a = 3$ et $b = 11$:

```
      while ( b >= a ) {
        b = b - a;
      }
```

(2) Pour a et b entiers positifs quelconques, quelle est la valeur de b après la boucle ?

Correction : (1) Exécution pas à pas : on indique les valeurs des variables à chaque étape.

instruction	a	b	b >= a
while (b >=a)	3	11	true
b = b -a	3	8	true
while (b >=a)	3	8	true
b = b -a	3	5	true
while (b >=a)	3	5	true
b = b -a	3	2	false
while (b >=a)	3	2	false
fin du programme			

(2) La valeur de b après la boucle pour a et b quelconques est b % a (le reste de la division euclidienne de b par a).

Exercice 4 (logarithme entier en base 2)**.**

(1) Écrire un programme dont l'entrée est un entier positif `n` et qui calcule le plus petit i de la forme 2^k tel que $n \leq i$. Rappel : il n'y a pas d'opérateur puissance en `C++`. Indication : partir de `i=1` et le multiplier par 2 tant que nécessaire.

(2) Modifier votre programme pour que la sortie soit `k` tel que $i = 2^k$ (où i est l'entier défini à la question précédente).

Correction : (1)

```
// Entrée : un entier n
// Sortie : un entier stocké dans la variable i
//  de la forme 2^k tel que n <= i

int i;
i = 1;
while (n > i){
        i = i * 2;
}
```

(2)

```
// Entrée : un entier n
// Sortie : un entier stocké dans la variable k
//  tel que i = 2^k où n <= i

int i, k;
i = 1;
k = 0;

while (n > i){
        i = i * 2;
        k++;
```

```
}
/// END logarithme-exposant
```

Exercice 5 (Instructions itératives avec compteur)**.**
On rappelle les exemples de programmes vus en cours pour afficher les nombres de 1 à 10 avec respectivement une boucle `while` et une boucle `for` :

```
int i = 1;
while ( i <= 10 ) {
    cout << i << endl;   // Affiche la valeur de i
    i = i + 1;
}
```

```
for ( int i = 1; i <= 10; i = i + 1 ) {
    cout << i << endl;   // Affiche la valeur de i
}
```

(1) Dans chacun de ces deux programmes, entourer la déclaration du compteur, l'initialisation, la condition et l'incrémentation.

(2) Adapter le premier exemple pour afficher les nombres pairs inférieurs ou égaux à 10, en commençant par 0. De même avec le deuxième exemple.

(3) Même chose pour afficher les nombres de 10 à 1 ;

(4) Même chose pour afficher les nombres entiers de carré inférieur à 10 ;

(5) Même chose pour calculer la valeur de la somme $1^2 + 2^2 + \cdots + 10^2$.

Notes pour les enseignants :
- Normalement, les étudiants ont davantage vu la boucle `while` que `for` jusqu'à cet exercice. Passer le temps nécessaire sur cet exercice car il leur est utile pour assimiler la syntaxe de la boucle `for` et la différence avec le `while`.
- Si vous corrigez au tableau, s'arranger pour avoir les deux versions (while et for) au tableau simultanément, pour que les points communs et les différences apparaîssent bien.
- Insister sur le fait que lors de l'éxécution d'une boucle for, l'incrémentation se fait à la fin du corps de la boucle, même si au niveau de l'écriture du code elle apparaît avant.

Correction : (1)

(2)

```
int i = 0;
while ( i <= 10 ) {
    cout << i << endl;
    i = i + 2;
}
```

```
for ( int i = 0; i <= 10; i = i + 2 ) {
    cout << i << endl;
}
```

(3)

```
int i = 10;
while ( i >= 0 ) {
    cout << i << endl;
    i = i - 1;
}
```

```
for ( int i = 10; i >= 0; i = i-1 ) {
    cout << i << endl;
}
```

(4)

```
int i = 0;
while ( i*i <= 10 ) {
    cout << i << endl;
    i++;
}
```

```
for ( int i = 0; i*i <= 10; i++ ) {
    cout << i << endl;
}
```

(5)

```
    int accumulateur = 0;
    int compteur = 1;
    while ( compteur <= n ) {
        accumulateur = accumulateur + compteur * compteur;
        compteur++;
    }
```

```
    int accumulateur = 0;
    for (int compteur = 1; compteur <= n ; compteur++ ) {
        accumulateur = accumulateur + compteur * compteur;
    }
```

À votre avis, dans chacun des cas ci-dessus, laquelle des deux formes est la plus naturelle ?

Correction : Pour les exemples de cet exercice, c'est la boucle for qui est plus naturelle (présence naturelle d'un compteur simple). Sauf pour la question 4 où la boucle while est plus naturelle (d'ailleurs en Python ce serait compliqué de faire la question 4 avec une boucle for).

Exercice ♣ 6 (Nombres premiers)**.**

(1) Écrire une fonction qui prend en argument (entrée) un entier n et *teste* si n est un nombre premier (c'est-à-dire renvoie `true` si n est premier et `false` sinon).

(2) Écrire une fonction qui prend en argument un entier n et affiche tous les nombres premiers entre 1 et n.

(3) Écrire une fonction qui affiche les n premiers nombres premiers.

Correction :

```
bool premier(int n) {
    if (n == 1) {
         return false ;
    }
    for (int i = 2; i < n; i++){
        if (n % i == 0 ) {
                return false;
        }
    }
    return true ;
}
```

```
void nombres_premiers(int n) {
    for (int i = 2 ; i <= n ; i++) {
        if (premier(i)) {
                cout << i << " " ;
        }
    }

}

void premiers_nombres_premiers(int n) {
    int compteur = 0 ;
    int i = 2 ;

    while (compteur < n) {
        if (premier(i)) {
            cout << i << " " ;
            compteur++ ;
        }
        i++ ;
    }
}
```

Exercice ♣ 7 (Dates)**.**

(1) Écrire une fonction qui prend en entrée une date sous la forme de trois entiers jour / mois / année, et teste si c'est une date valide. Pour l'instant, on ignore les années bissextiles. Par exemple :
— `date_valide(28, 5, 1973)` renvoie `true`
— `date_valide(31, 2, 2015)` renvoie `false`
Correction :

```
// Entrées: trois entiers j, m et a contenant le jour, le mois et l'année
// Sortie : un booléen indiquant si la date est valide.

bool valide(int j, int m, int a) {
    if (m < 12) {
        if ((m == 1) or (m == 3) or (m == 5) or (m == 7) or
                (m == 8) or (m == 10) or (m == 12)) {
            return (j <= 31);
        } else if (m != 2) {
            return (j <= 30);
        } else {
            return (j <= 28);
        }
    } else {
        return false;
    }
}
```

(2) Écrire une fonction qui prend en argument une date valide sous la forme de trois entiers et affiche la date du lendemain. Par exemple :
— `jour_suivant(18, 12, 2017)` affiche `Le jour suivant est le 19 12 2018.`
Correction :

```
// Entrées: trois entiers j, m et a contenant le jour, le mois et l'année
// Sortie : Aucune mais affichage du jour suivant.

void jour_suivant (int j, int m, int a) {
    if (j < 28 or
        (j < 31 and ((m == 3) or
        (m == 5) or (m == 7) or (m == 8)
        or (m == 10) or (m == 12)))
        or (j < 30 and ((m == 1) or (m == 4) or
        (m == 6) or (m == 9) or (m == 11)))) {

        cout << "Le jour suivant est le " << jour +
             1 << " / " << m << " / " << a << endl;
    } else if (m < 12) {
        cout << "Le jour suivant est le " << 1 << " / " << m +
             1 << " / " << a;
    } else {
        cout << "Le jour suivant est le 1  / 1  / " << a + 1;
    }
}
```

(3) Reprendre la question 1 en considérant les années bissextiles (une année est bissextile si elle est divisible par 4, mais pas 100 sauf si elle est divisible par 400).

Correction :

```
// Entrées: trois entiers j, m et a contenant le jour, le mois et l'année
// Sortie : un booléen indiquant si la date est valide,
// y compris pour les années bissextiles.

int valide_bissextile(int j, int m, int a) {
    if (m < 12){
        if ((m == 1) or (m == 3) or (m == 5) or (m == 7) or
                 (m == 8) or (m == 10) or (m == 12)) {
            return (j <= 31);
        } else if (m != 2){
            return (j <= 30);
        } else if (a % 400 == 0 or (a % 4 == 0 and a % 100 != 0)){
            return (j <= 29);
        } else {
            return (j <= 28);
        }
    } else {
        return false;
    }
}
```

(4) Écrire une fonction qui prend en argument une date valide et renvoie le jour de la semaine de cette date.

(5) Projet Euler- 19 : Combien de 1er du mois ont été des dimanches au XXe siècle ?