# Chapitre 4 : Fonctions

## *Exercice 4.1 Passage par valeur*

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

```
#include <cstdlib>
#include <iostream>
using namespace std;

void f(int n) {
   n *= n + 1;
   cout << "B : n = " << n << endl;
}

int main() {

   int n = 2;

   cout << "A : n = " << n << endl;
   f(n);
   cout << "C : n = " << 2 * n << endl;

   return EXIT_SUCCESS;
}
```

## *Solution exercice 4.1*

```
A : n = 2
B : n = 6
C : n = 4
```

## *Exercice 4.2 Appels de fonctions*

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

```
#include <cmath>
#include <cstdlib>
#include <iostream>
using namespace std;

double k(double x) {return 2 * (x + 1);}
double h(double x) {return x * x + k(x) - 1;}
double g(double x) {return 4 * h(x);}
double f(double x) {return g(x) + sqrt(h(x));}

int main() {

   double x1 = f(2),
          x2 = g(h(2)),
          x3 = k(g(2) + h(2)),
          x4 = f(0) + f(1) + f(2),
          x5 = f(-1) + g(-1) + h(-1) + k(-1);

   cout << "x1 = " << x1 << endl;
   cout << "x2 = " << x2 << endl;
   cout << "x3 = " << x3 << endl;
   cout << "x4 = " << x4 << endl;
   cout << "x5 = " << x5 << endl;

   return EXIT_SUCCESS;
}
```

## *Solution exercice 4.2*

```
x1 = 39
x2 = 400
x3 = 92
x4 = 62
x5 = 0
```

## *Exercice 4.3 Volume d'une pyramide à base rectangulaire*

Ecrire un programme C++ mettant à disposition une fonction permettant de calculer le volume d'une pyramide à base rectangulaire.

Appliquer la fonction pour calculer le volume des 2 pyramides suivantes :

1) Pyramide 1 : base de longueur 10 et de largeur 3.5; hauteur = 12
2) Pyramide 2 : base de longueur 3.6 et de largeur 2.4; hauteur = 2.7

Afficher les résultats avec un chiffre après la virgule.

## *Solution exercice 4.3*

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <string>
using namespace std;

/**
 * @brief Retourne le volume d'une pyramide à base rectangulaire
 * @param longueur   la longueur de la pyramide
 * @param largeur    la largeur de la pyramide
 * @param hauteur    la hauteur de la pyramide
 * @return le volume de la pyramide à base rectangulaire
 */
double volumePyramide(double longueur, double largeur, double hauteur);

/**
 * @brief Affiche à l'écran <code>texte</code> suivi de <code>valeur</code>.
 * <p>
 * <code>valeur</code> est affichée en mode fixed avec une précision
 * (nombre de chiffres après la virgule) fixée par le paramètre
 * <code>precision</code>.
 * @param texte      le texte à afficher
 * @param valeur     la valeur à afficher
 * @param precision  la précision souhaitée
 */
void afficher(const string& texte, double valeur, int precision);

int main() {

   afficher("volume pyramide 1 = ", volumePyramide(10, 3.5, 12), 1);   // 140.0
   afficher("volume pyramide 2 = ", volumePyramide(3.6, 2.4, 2.7), 1); // 7.8

   return EXIT_SUCCESS;
}

double volumePyramide(double longueur, double largeur, double hauteur) {
   return longueur * largeur * hauteur / 3;
}

void afficher(const string& texte, double valeur, int precision) {
   cout << fixed << setprecision(precision) << texte << valeur << endl;
}
```

## *Exercice 4.4 Année bissextile*

Ecrire un programme C++ affichant à l'écran si les années 1900, 2000, 2010 et 2020 sont bissextiles ou non.

Pour ce faire, écrire une fonction booléenne `estBissextile` prenant en paramètre l'année à tester.

**Rappel** Une année est bissextile si elle est divisible par 400 ou alors par 4 mais pas par 100.

## *Solution exercice 4.4*

```
#include <cstdlib>
#include <iostream>
using namespace std;

bool estBissextile(int annee); // prototypes
void test(int annee);

int main() {

   test(1900);
   test(2000);
   test(2010);
   test(2020);

   return EXIT_SUCCESS;
}

bool estBissextile(int annee) {
   return (annee % 400 == 0) or (annee % 4 == 0 and annee % 100 != 0);
// Autres variantes possibles :
//    return (annee % 400 == 0) || (annee % 4 == 0 && annee % 100 != 0);
//    return annee % 400 == 0 or annee % 4 == 0 and annee % 100 != 0;
//    return annee % 400 == 0 || annee % 4 == 0 && annee % 100 != 0;
//    return !(annee % 400) || !(annee % 4) && annee % 100;
}

void test(int annee) {
   cout << annee << " : " << boolalpha << estBissextile(annee) << endl;
}

// 1900 : false
// 2000 : true
// 2010 : false
// 2020 : true
```

## *Exercice 4.5 Affichage de caractères compris entre 2 bornes*

Ecrire une fonction qui a 2 paramètres de type `unsigned char` et qui affiche, sur une même ligne, tous les caractères compris entre le premier et le deuxième paramètre (bornes comprises), pour autant que le premier représente un caractère qui précède le deuxième.

La fonction retourne une valeur booléenne indiquant si oui ou non il y a eu affichage de caractères.

## *Solution exercice 4.5*

```
#include <cstdlib>
#include <iostream>
#include <limits>
using namespace std;

bool listerCaracteres(unsigned char c1, unsigned char c2);

int main() {

   cout << "Test 1" << endl;
   listerCaracteres('A', 'A');

   cout << "Test 2" << endl;
   listerCaracteres('A', 'B');

   cout << "Test 3" << endl;
   listerCaracteres('B', 'A');

   cout << "Test 4" << endl;
   listerCaracteres('0', '9');

   cout << "Test 5" << endl;
   listerCaracteres(65, 66);

   cout << "Test 6" << endl;
   unsigned char c1 = numeric_limits<unsigned char>::max() - 1,
                 c2 = numeric_limits<unsigned char>::max();
   listerCaracteres(c1, c2);

   cout << "Test 7" << endl;
   listerCaracteres('\x93', '\x9B'); // du caract 147 au caract 155

   cout << "Fin des tests" << endl;

   return EXIT_SUCCESS;
}

bool listerCaracteres(unsigned char c1, unsigned char c2) {
   if (c1 >= c2) {
      return false;
   } else {
      const unsigned NB_CHAR = c2 - c1 + 1;
      for (unsigned i = 0; i < NB_CHAR; ++i)
         cout << (unsigned char)(c1 + i);
      cout << endl;
      return true;
   }
}
```

```
// Autre variante possible (mais moins bonne)
// bool listerCaracteres(unsigned char c1, unsigned char c2) {
//     if (c1 >= c2) {
//         return false;
//     } else {
//         for (unsigned char c = c1; c <= c2; ++c) {
//             cout << c;
//             if (c == numeric_limits<unsigned char>::max()) break;
//         }
//         cout << endl;
//         return true;
//     }
// }

// Test 1
// Test 2
// AB
// Test 3
// Test 4
// 0123456789
// Test 5
// AB
// Test 6
// ■
// Test 7
// ôöòûùÿÖÜø
// Fin des tests
```

## *Exercice 4.6 Correction d'erreurs*

Quelles modifications faut-il apporter au programme ci-dessous pour le rendre correct ?

**NB** Aucune ligne de code ne doit être ajoutée !

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {
   f;
   g(1, 2.0);
   h(64, 65);
   return EXIT_SUCCESS;
}

void f {
   cout << "Appel de f\n";
}

void g(int i, j) {
   cout << "Appel de g avec i = " << i << " et j = " << j << "\n";
}

void h(int i; char c) {
   cout << "Appel de h avec i = " << i << " et c = '" << c << "'\n";
}
```

## *Solution exercice 4.6*

```
#include <cstdlib>
#include <iostream>
using namespace std;

// Erreur 1 : f, g et h doivent figurer avant main
void f() { // Erreur 2 : paire de parenthèses vides obligatoire
   cout << "Appel de f\n";
}

void g(int i, int j) { // Erreur 3 : chaque paramètre doit faire l'objet
                       // d'une déclaration de type
   cout << "Appel de g avec i = " << i << " et j = " << j << "\n";
}

void h(int i, char c) { // Erreur 4 : on liste les arguments en les
                        // séparant par une virgule et non un point-
                        // virgule
   cout << "Appel de h avec i = " << i << " et c = '" << c << "'\n";
}

int main() {
   f(); // Erreur 5 : paire de parenthèses vides obligatoire
   g(1, 2.0); // OK. Conversion implicite (dégradante)
              // Possible warning du compilateur
   h(64, 65);
   return EXIT_SUCCESS;
}
```

## *Exercice 4.7 Mystère, mystère !*

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

```
#include <cstdlib>
#include <iostream>
using namespace std;

void mystere(int& a, int& b);

int main() {

   int x = 3,
       y = 4;

   mystere(x, y);
   cout << "x = " << x << " " << "y = " << y << endl;

   return EXIT_SUCCESS;
}

void mystere(int& a, int& b) {
   a = a - b;
   b = b + a;
   a = b - a;
}
```

## *Solution exercice 4.7*

```
x = 4 y = 3      => mystere permet d'échanger les valeurs a et b
```

## *Exercice 4.8 Permutation circulaire droite de 3 réels*

Ecrire un programme C++ qui met à disposition une fonction sans valeur de retour et prenant 3 paramètres de type `double` permettant d'effectuer une permutation circulaire droite de ces 3 valeurs.

## *Solution exercice 4.8*

```
#include <cstdlib>
#include <iostream>
using namespace std;

void permutationCirculaireDroite(double& x, double& y, double& z);
void afficher(double x, double y, double z);

int main() {

   double x = 1,
          y = 2,
          z = 3;

   for (int i = 1; i <= 3; ++i) {
      permutationCirculaireDroite(x, y, z);
      afficher(x, y, z);
   }

   return EXIT_SUCCESS;
}

void permutationCirculaireDroite(double& x, double& y, double& z) {
   double tmp = z;
   z = y;
   y = x;
   x = tmp;
}

void afficher(double x, double y, double z) {
   cout << "x = " << x << "  y = " << y << "  z = " << z << endl;
}

// x = 3  y = 1  z = 2
// x = 2  y = 3  z = 1
// x = 1  y = 2  z = 3
```

## *Exercice 4.9 Retrait d'argent*

Ecrire un programme C++ mettant à disposition une fonction permettant de retirer un certain montant sur un compte bancaire.

La fonction :

- doit prendre 2 paramètres : le solde courant du compte et le montant du retrait (que l'on supposera tous deux ≥ 0)
- doit vérifier les contraintes métier suivantes :
  - le solde courant ne peut jamais être négatif
  - si le solde courant est insuffisant pour retirer l'intégralité du montant souhaité, seul le montant maximal possible est retiré
- doit retourner le montant du retrait effectif réalisé

Vérifier le bon fonctionnement de la fonction en appliquant le scénario de test suivant :

- Solde initial du compte = 500
- Premier retrait = 300
- Afficher le montant du retrait effectif et le solde courant du compte
- Deuxième retrait = 300
- Afficher le montant du retrait effectif et le solde courant du compte

## *Solution exercice 4.9*

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <string>
using namespace std;

double retrait(double montantRetrait, double& soldeCourant);
void afficher(const string& texte, double valeur, int precision);

int main() {

   double soldeCourant = 500,
          retraitEffectif;

   retraitEffectif = retrait(300, soldeCourant);
   afficher("Montant retrait effectif = ", retraitEffectif, 1);
   afficher("Solde courant            = ", soldeCourant, 1);

   retraitEffectif = retrait(300, soldeCourant);
   afficher("Montant retrait effectif = ", retraitEffectif, 1);
   afficher("Solde courant            = ", soldeCourant, 1);

   return EXIT_SUCCESS;
}

double retrait(double montantRetrait, double& soldeCourant) {
   double retraitEffectif;
   if (soldeCourant > montantRetrait) {
      retraitEffectif = montantRetrait;
      soldeCourant = soldeCourant - montantRetrait;
   } else { // soldeCourant <= montantRetrait
      retraitEffectif = soldeCourant;
      soldeCourant = 0;
   }
   return retraitEffectif;
}

void afficher(const string& texte, double valeur, int precision) {
   cout << texte << fixed << setprecision(precision) << valeur << endl;
}

// Montant retrait effectif = 300.0
// Solde courant            = 200.0
// Montant retrait effectif = 200.0
// Solde courant            = 0.0
```

**NB** En utilisant <*cmath*>, on peut écrire la fonction *retrait* de manière plus compacte :

```
double retrait(double montantRetrait, double& soldeCourant) {
   double retraitEffectif = fmin(montantRetrait, soldeCourant);
   soldeCourant -= retraitEffectif;
   return retraitEffectif;
}
```

## *Exercice 4.10 Fonction "Opérations arithmétiques"*

Ecrire une fonction qui reçoit en paramètres d'entrée 2 nombres réels (de type `double`) et un caractère, et qui fournit en paramètre de sortie le résultat correspondant à l'une des 4 opérations appliquées à ses 2 premiers paramètres, en fonction de la valeur du troisième paramètre, à savoir : addition pour le caractère '+', soustraction pour '-', multiplication pour '*' et division pour '/'. Si l'un de ces 4 caractères est passé en argument, la fonction doit renvoyer `true` comme valeur (statut) de retour; dans le cas contraire, elle doit renvoyer `false`. Ne pas tenir compte des risques de division par zéro.

Ecrire aussi un petit programme (`main`) utilisant cette fonction pour effectuer les 4 opérations sur les 2 nombres fournis en entrée par l'utilisateur. Ne pas tenir compte des éventuelles erreurs de saisie utilisateur.

## *Solution exercice 4.10*

```
#include <cstdlib>
#include <iostream>
using namespace std;

bool operation(double operandeGauche,
               double operandeDroite,
               char operateur,
               double& resultat);

void test(double operandeGauche,
          double operandeDroite,
          char operateur);

int main() {

   double operandeGauche, operandeDroite;

   cout << "Donnez 2 nombres reels : ";
   cin >> operandeGauche >> operandeDroite; // saisie non contrôlée ici

   test(operandeGauche, operandeDroite, '+');
   test(operandeGauche, operandeDroite, '-');
   test(operandeGauche, operandeDroite, '*');
   test(operandeGauche, operandeDroite, '/');
   test(operandeGauche, operandeDroite, '?');

   return EXIT_SUCCESS;
}

bool operation(double operandeGauche,
               double operandeDroite,
               char operateur,
               double& resultat) {
   bool OK = true; // statut de retour de la fonction
   switch (operateur) {
      case '+':
         resultat = operandeGauche + operandeDroite;
         break;
      case '-':
         resultat = operandeGauche - operandeDroite;
         break;
      case '*':
         resultat = operandeGauche * operandeDroite;
         break;
      case '/':
         resultat = operandeGauche / operandeDroite; // pas d'erreur si division
                                                     // par 0 sur les réels
         break;
      default:
         OK = false;
         break; // pas strictement nécess mais mieux de le mettre
   }
   return OK;
}
```

```
void test(double operandeGauche,
          double operandeDroite,
          char operateur) {
   double resultat;
   if (operation(operandeGauche, operandeDroite, operateur, resultat)) {
      cout << operandeGauche << " " << operateur << " " << operandeDroite
           << " = " << resultat << endl;
   } else {
      cout << "L'operation " << "'" << operateur << "'" << " est illicite" << endl;
   }
}

// Donnez 2 nombres reels : 3 5
// 3 + 5 = 8
// 3 - 5 = -2
// 3 * 5 = 15
// 3 / 5 = 0.6
// L'operation '?' est illicite
```

## *Exercice 4.11 Passage par valeur vs par référence*

Le programme ci-dessous contient plusieurs erreurs.

1) Pour chaque ligne fautive, indiquez son numéro et précisez la nature de l'erreur
2) En admettant maintenant que les lignes fautives ont été supprimées du programme ci-dessous, déterminer "à la main" ce qu'il va afficher à l'exécution ?

```
 1 #include <cstdlib>
 2 #include <iostream>
 3 using namespace std;
 4
 5 void f(double x);
 6 void g(double& x);
 7 void h(const double& x);
 8
 9 int main() {
10
11    const int C = 1;
12    int n = C;
13    double x = C;
14
15    f(C);
16    f(n);
17    f(x);
18    g(C);
19    g(n);
20    g(x);
21    h(C);
22    h(n);
23    h(x);
24
25    return EXIT_SUCCESS;
26 }
27
28 void f (double x) {
29    x = x + 1;
30    cout << "x = " << x << endl;
31 }
32
33 void g (double& x) {
34    x = x + 1;
35    cout << "x = " << x << endl;
36 }
37
38 void h (const double& x) {
39    x = x + 1;
40    cout << "x = " << x << endl;
41 }
```

## *Solution exercice 4.11*

1) Ligne 18 : Le paramètre effectif de g ne peut être qu'une variable
   Ligne 19 : Le paramètre effectif de g doit être du même type que le paramètre formel
   (pas de conversion implicite possible dans ce cas !)
   Ligne 39 : Le mot clé `const` précise que le paramètre ne peut pas être modifié dans le corps de la fonction

2) x = 2
   x = 2
   x = 2
   x = 2
   x = 1
   x = 1
   x = 2 (2 car le paramètre effectif x a été préalablement modifié lors de l'appel g(x))

## *Exercice 4.12 Portée des variables*

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

```
#include <cstdlib>
#include <iostream>
using namespace std;

int a, b;

int f(int c) {
   int n = 0;
   a = c;
   if (n < c) {
      n = a + b;
   }
   return n;
}

int g(int c) {
   int n = 0;
   int a = c;
   if (n < f(c)) {
      n = a + b;
   }
   return n;
}

int main() {
   int i = 1;
   int b = g(i);
   cout << a + b + i << endl;
   return EXIT_SUCCESS;
}
```

## *Solution exercice 4.12*

3

## *Exercice 4.13 Fonction "compteur"*

Ecrire un programme C++ qui appelle 3 fois la même fonction `afficher`, cette dernière se contentant d'afficher, à chaque appel, le nombre total de fois où elle a été appelée, sous la forme :

```
Appel numéro n
```

On supposera que la fonction `afficher` est une fonction sans argument ni valeur de retour.

## *Solution exercice 4.13*

Première solution (déconseillée toutefois !) : utiliser une variable globale statique (ici déclarée explicitement static de manière à ce qu'elle ne soit visible que dans notre fichier)

```
#include <cstdlib>
#include <iostream>
using namespace std;

static int compteur; // initialisation à 0 inutile car faite par défaut

void afficher();

int main() {
   for (int i = 1; i <= 3; ++i)
      afficher();

   return EXIT_SUCCESS;
}

void afficher() {
   cout << "Appel no " << ++compteur << endl;
}
```

Seconde solution : utiliser une variable locale statique

```
#include <cstdlib>
#include <iostream>
using namespace std;

void afficher();

int main() {
   for (int i = 1; i <= 3; ++i)
      afficher();

   return EXIT_SUCCESS;
}

void afficher() {
   static int compteur; // initialisation à 0 inutile car faite par défaut
   cout << "Appel no " << ++compteur << endl;
}
```

## *Exercice 4.14 Surcharge (1)*

Soient les déclarations suivantes :

```
void f(char);  // fonction I
void f(int);   // fonction II
void f(float); // fonction III

bool b = true;
char c = 'A';
short s = 1;
int i = 2;
float x = 3.f;
double y = 4.0;
```

Les appels suivants sont-ils corrects et, si oui, quelles seront les fonctions effectivement appelées et les conversions éventuelles mises en œuvre ?

1) `f(b);`
2) `f(c);`
3) `f(s);`
4) `f(i);`
5) `f(x);`
6) `f(y);`

## *Solution exercice 4.14*

1) fonction II (après conversion de la valeur de b en int)
2) fonction I
3) fonction II (après conversion de la valeur de s en int)
4) fonction II
5) fonction III
6) Erreur à la compilation !
   Appel ambigu car toutes les fonctions sont potentiellement candidates

***Règles (cf Delannoy p. 677 à 680)***

*1) Correspondance exactes (on ne tient pas compte d'un éventuel const sauf dans le cas d'un passage par pointeur (const t * vs t *) ou par référence (const t & vs t &))*

*2) Promotions numériques (bool, char, short -> int; float -> double)*

*3) Conversions standards (y compris dégradantes)*

## *Exercice 4.15 Surcharge (2)*

Soient les déclarations suivantes :

```
void f(int, float); // fonction I
void f(float, int); // fonction II

int i = 1;
char c = 'A';
float x = 2.f;
double y = 3.0;
```

Les appels suivants sont-ils corrects et, si oui, quelles seront les fonctions effectivement appelées et les conversions éventuelles mises en œuvre ?

1) `f(i, c);`
2) `f(c, i);`
3) `f(i, i);`
4) `f(x, c);`
5) `f(c, x);`
6) `f(x, x);`
7) `f(i, y);`
8) `f(y, i);`
9) `f(y, y);`

## *Solution exercice 4.15*

1) Appel ambigu (`I, II → ∅`)
2) Appel ambigu (`I, II → ∅`)
3) Appel ambigu (`I, II → ∅`)
4) Fonction II (après conversion de la valeur de c en int) (`II, II → II`)
5) Fonction I (après conversion de la valeur de c en int) (`I, I → I`)
6) Appel ambigu (`II, I → ∅`)
7) Fonction I (après conversion dégradante de la valeur de y en float) (`I, {I, II} → I`)
8) Fonction II (après conversion dégradante de la valeur de y en float) (`{I, II}, II → II`)
9) Appel ambigu (`{I, II}, {I, II} → {I, II}`)

***Astuce** : examiner séparément chacun des paramètres et faire l'intersection des ensembles solutions*

## *Exercice 4.16 Suite de Syracuse*

***Partie A***

La suite de Syracuse repose sur un principe simple. Prenez un nombre entier positif quelconque :

- s'il est pair, divisez-le par 2;
- s'il est impair, multipliez-le par 3 et ajoutez 1.

Renouvellez cette opération plusieurs fois. Après suffisamment d'itérations, vous devriez finir par tomber sur la valeur 1.
Par exemple, à partir de 17, on trouve la suite de valeurs : 52 26 13 40 20 10 5 16 8 4 2 1.

On demande ici d'écrire un programme C++ (dans un seul fichier) qui demande à l'utilisateur d'entrer un nombre entier $n$ ($1 \leq n \leq 10000$) et qui affiche à l'écran 1°) les valeurs successives de la suite de Syracuse relative à ce nombre (en s'arrêtant bien sûr à 1), 2°) combien d'itérations ont été nécessaires pour parvenir à 1.

***Partie B***

Modifiez le programme précédent pour qu'il affiche le nombre d'itérations nécessaires pour parvenir à 1, non plus pour le nombre entré par l'utilisateur, mais pour les $n$ premiers entiers positifs ($n$ fixé par l'utilisateur, $1 \leq n \leq 1000$)

**Prescriptions (valables pour les parties A et B) :**
- Structurez votre code en sous-programmes
- Il n'est pas demandé de vérifier les saisies utilisateurs

## Exemple d'exécution (partie A)

```
Entrez un entier dans l'intervalle [1, 10000] : 29
Suite de Syracuse pour n = 29 :
88
44
22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
1
Suite terminee en 18 iterations.
```

## Exemple d'exécution (partie B)

```
Entrez un entier dans l'intervalle [1, 1000] : 10
Nbre d'iter. pour n = 1    :   0
Nbre d'iter. pour n = 2    :   1
Nbre d'iter. pour n = 3    :   7
Nbre d'iter. pour n = 4    :   2
Nbre d'iter. pour n = 5    :   5
Nbre d'iter. pour n = 6    :   8
Nbre d'iter. pour n = 7    :  16
Nbre d'iter. pour n = 8    :   3
Nbre d'iter. pour n = 9    :  19
Nbre d'iter. pour n = 10   :   6
```

# *Solution exercice 4.16*

## *Partie A*

```
#include <cstdlib>
#include <iostream>
using namespace std;

unsigned lireEntier(unsigned a, unsigned b);

// Calcule la suite de Syracuse du nombre n
// Renvoie en valeur de retour le nombre d'itérations nécessaires
// Affiche les diverses valeurs de la suite si avecAffichage vaut true;
// ne les affiche pas, sinon
unsigned suiteDeSyracuse(unsigned n, bool avecAffichage);

int main() {

   const unsigned MIN = 1,
                  MAX = 10000,
                  N = lireEntier(MIN, MAX);

   cout << "Suite de Syracuse pour n = " << N << " :\n";
   const unsigned NB_ITERATIONS = suiteDeSyracuse(N, true);
   cout << "Suite terminee en " << NB_ITERATIONS << " iteration"
        << (NB_ITERATIONS > 1 ? "s" : "") << "." << endl;

   return EXIT_SUCCESS;
}

unsigned lireEntier(unsigned a, unsigned b) {
   int n;
   cout << "Entrez un entier dans l'intervalle"
        << " [" << a << ", " << b << "] : ";
   cin >> n;
   return (unsigned) n;
}

// NB La fonction ci-dessous n'affiche rien qui soit dépendant d'une langue
unsigned suiteDeSyracuse(unsigned n, bool avecAffichage) {
   unsigned m = n,
            nbIterations = 0;
   while (m > 1) {
      if (m % 2 == 0) {
         m = m / 2;
      } else {
         m = m * 3 + 1;
      }
      if (avecAffichage) {
         cout << m << endl;
      }
      nbIterations++;
   }
   return nbIterations;
}
```

***Partie B (si l'on s'y prend bien dans la partie A… très peu de changements sont nécessaires)***

```
#include <cmath>
#include <cstdlib>
#include <iomanip>
#include <iostream>
using namespace std;

unsigned lireEntier(unsigned a, unsigned b);

// Calcule la suite de Syracuse du nombre n
// Renvoie en valeur de retour le nombre d'itérations nécessaires
// Affiche les diverses valeurs de la suite si avecAffichage vaut true;
// ne les affiche pas, sinon
unsigned suiteDeSyracuse(unsigned n, bool avecAffichage);

int main() {

   const unsigned MIN = 1,
                  MAX = 1000,
                  N = lireEntier(MIN, MAX);

   const int W = (int) log10(MAX) + 1; // pour l'affichage (setw(W))

   for (unsigned n = MIN; n <= N; ++n) {
      cout << "Nbre d'iter. pour n = "
           << setw(W) << left << n << " :"
           << setw(W) << right << suiteDeSyracuse(n, false) << endl;
   }

   return EXIT_SUCCESS;
}

unsigned lireEntier(unsigned a, unsigned b) {
   int n;
   cout << "Entrez un entier dans l'intervalle"
        << " [" << a << ", " << b << "] : ";
   cin >> n;
   return (unsigned) n;
}

// NB La fonction ci-dessous n'affiche rien qui soit dépendant d'une langue
unsigned suiteDeSyracuse(unsigned n, bool avecAffichage) {
   unsigned m = n,
            nbIterations = 0;
   while (m > 1) {
      if (m % 2 == 0) {
         m = m / 2;
      } else {
         m = m * 3 + 1;
      }
      if (avecAffichage) {
         cout << m << endl;
      }
      nbIterations++;
   }
   return nbIterations;
}
```