MOOC Init Prog C++
Corrigés semaine 4

Les corrigés proposés correspondent à l'ordre des apprentissages : chaque corrigé correspond à la solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante.

Exercice 11 : Prototypes (niveau 1, puis 2 pour le point 5)

> Cet exercice correspond à l'exercice n°12 (pages 31 et 210) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Solution de 1,2,3 :

La fonction `demander_nombre()` ne prend aucun paramètre puisqu'elle se contente de demander un entier à l'utilisateur (elle n'a donc besoin de rien !). Par contre elle devra *retourner* la valeur saisie par l'utilisateur. Le type de la valeur retournée par la fonction devant être spécifié dans le prototype, celui-ci est donc `int demander_nombre();`.

L'écriture du corps de la fonction ne présente rien de nouveau : on déclare une variable entière `a_retourner`, on l'affecte à la valeur entrée par l'utilisateur, puis on retourne cette valeur à l'aide de l'instruction `return`.

Afin de pouvoir être utilisée dans la fonction `main()`, la fonction doit avoir été prototypée avant. Sa définition peut ensuite être placée n'importe où après le prototype. La valeur retournée par la `demander_nombre()` peut être assignée à la variable `num` de la fonction `main()` avec l'instruction `num = demander_nombre();`.

Voici le code complet de la première partie de l'exercice :

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

int demander_nombre();

int main()
{
  int num;
  num = demander_nombre();
  cout << "Le nombre est : " << num << endl;
  return 0;
}

int demander_nombre()
{
  int res;
  cout << "Entrez un nombre entier : ";
  cin >> res;
  return res;
}
```

Solution de 4 :

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

int demander_nombre(int min, int max);

int main()
{
  int num;
  num = demander_nombre(1, 100);
  cout << "Le nombre est : " << num << endl;
```

```
  return 0;
}

int demander_nombre(int a, int b)
{
  // échange a et b si ils ne sont pas dans le bon ordre
  // ceci est NÉCESSAIRE si on ne veut pas de boucle infinie
  // dans le cas ou on appelle la fonction avec a>b !!
  if (a > b) { int tmp(b); b=a; a=tmp; }

  int res;
  do {
    cout << "Entrez un nombre entier compris entre "
         << a << " et " << b <<" : ";
    cin >> res;
  } while ((res < a) or (res > b));
  return res;
}
```

Solution de 5 :

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

int demander_nombre(int min, int max);

int main()
{
    int num;
    num = demander_nombre (1, 100);
    cout << "Le nombre est : " << num << endl;
    num = demander_nombre (100, 1);
    cout << "Le nombre est : " << num << endl;
    return 0;
}

int demander_nombre(int a, int b)
{
    int res;

    do {
      cout << "Entrez un nombre entier ";
      if (a >= b)
        cout << "supérieur ou égal à " << a;
      else
        cout << "compris entre " << a << " et " << b;
      cout << " : ";
      cin >> res;
    } while ((res < a) or ((a < b) and (res > b)));

   return res;
}
```

Exercice 12 : Passage des paramètres (niveau 1)

> Cet exercice correspond à l'exercice n°13 (pages 31 et 212) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Le code de la fonction `main()` nous donne des indications sur le fonctionnement de la fonction `echange` :

- La valeur de retour de la fonction n'est assignée à aucune variable. Le type de la valeur de retour de la fonction est donc `void`.
- Les variables `i` et `j` passées en paramètre à la fonction sont de type `int`.
- Les valeurs des paramètres sont échangées bien que les variables soient locales à la fonction `main()`, i.e. hors de portée de la fonction `echange`. Les paramètres doivent donc être passés par référence.

Nous pouvons donc maintenant écrire le prototype de la fonction :

```
void echange(int& a, int& b);
```

La définition de la fonction peut maintenant échanger les valeurs en utilisant une variable intermédiaire du même type pour stocker temporairement l'une des valeurs :

```
void echange(int& a, int& b)
{
  int copie(a);
  a=b;
  b=copie;
}
```

Exercice 13 : La fonction cosinus (définition et appel de fonction, niveau 2)

Cet exercice correspond à l'exercice n°14 (pages 32 et 213) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

```
#include <iostream>
#include <cmath> // pour fmod et abs ; optionnel (pas demandé)
using namespace std;

double factorielle(int k);
double somme_partielle(double x, int N);
int demander_nombre(int a = 2, int b = 85);

int main()
{
  int N(demander_nombre());

  double x;
  cout.precision(16);
  do {
    cout << "entrez une valeur x pour le calcul de cos(x) : ";
    cin >> x;
    x=fmod(abs(x), 2.0*M_PI); // optionnel : garantit que x est dans [0; 2pi]
    cout << "cos(" << x << ") ~ " << somme_partielle(x, N) << endl;
  } while (x != 0.0);
  return 0;
}

double factorielle(int k)
{
  double fact(1.0);
  // k! = 1*2*..*k
  for (int i(2); i <= k; ++i) {
    fact *= i;
  }
  return fact;
}

double somme_partielle(double x, int N)
{
  double current_approx(0.0);  // approximation courante
  double powerx(1.0);          // puissance de x (initialisé à x^0=1)

  for (int i(0); i <= N; ++i) {
    if (i%2 != 0) {
      current_approx -= powerx / factorielle(i*2);
    } else {
      current_approx += powerx / factorielle(i*2);
    }
    powerx *= x*x; // powerx représente x^0, x^2, x^4, x^6, ..., x^2n
  }
  return current_approx;
}

int demander_nombre(int a, int b)
{
  int res;

  if (a > b) { res=b; b=a; a=res; }

  do {
    cout << "Entrez le degré d'approximation (entre "
         << a << " et " << b <<") : ";
    cin >> res;
  } while ((res < a) or (res > b));
  return res;
}
```

Exercice 14 : nombres de Fibonacci

Cet exercice correspond à l'exercice n°33 (pages 82 et 254) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

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

int demander_nombre(int min, int max);
int Fibonacci(int n);
int FibonacciIteratif(int n);

// ----------------------------------------------------------------------
int main()
{
  char rep;

  do {
    int n(demander_nombre(0, 40));
    cout << "Méthode itérative :" << endl;
    cout << "    F(" << n << ") = " << FibonacciIteratif(n)
         << endl;
    cout << "Méthode récursive :" << endl;
    cout << "    F(" << n << ") = " << Fibonacci(n) << endl;

    do {
      cout << "Voulez-vous recommencer [o/n] ? ";
      cin >> rep;
    } while ((rep != 'o') and (rep != 'n'));
  } while (rep == 'o');

  return 0;
}

/* ----------------------------------------------------------------------
 * Calcule de façon itérative le n-ieme nombre de Fibonacci.
 * Entrée : le nombre n
 * Sortie : F(n)
 * ---------------------------------------------------------------------- */
int FibonacciIteratif(int n)
{
  int Fn(0);    // stocke F(i)  , initialisé par F(0)
  int Fn_1(Fn); // stocke F(i-1), initialisé par F(0)
  int Fn_2(1);  // stocke F(i-2), initialisé par F(-1)

  for (int i(1); i <= n; ++i) {
    Fn   = Fn_1 + Fn_2;    // pour n>=1 on calcule F(n)=F(n-1)+F(n-2)
    Fn_2 = Fn_1;           // et on decale...
    Fn_1 = Fn;
  }
  return Fn;
}

/* ----------------------------------------------------------------------
 * Calcule de façon récursive le n-ieme nombre de Fibonacci.
 * Entrée : le nombre n
 * Sortie : F(n)
 * ---------------------------------------------------------------------- */
int Fibonacci(int n)
{
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return Fibonacci(n-1) + Fibonacci(n-2);
}

/* ----------------------------------------------------------------
```

```
 * fonction demandant à l'utilisateur un nombre compris
 * dans un intervalle [a, b], ou supérieur ou égal à a
 * si b < a.
 * Entrée : les deux nombres a et b définissant l'intervalle
 * Sortie : le nombre saisi par l'utilisateur
 * ---------------------------------------------------------- */
int demander_nombre(int a, int b)
{
    int res;

    do {
      cout << "Entrez un nombre int ";
      if (a >= b)
        cout << "supérieur ou égal à " << a;
      else
        cout << "compris entre " << a << " et " << b;
      cout << " : ";
      cin >> res;
    } while ((res < a) or ((a < b) and (res > b)));

   return res;
}
```