Université Mohamed Premier
Ecole Supérieure de Technologie
Oujda

Module : Programmation C++

Filière Mécatronique (Deuxième Année)
Année Académique : 2017/2018

Solutions des TD/TP N° 2

Exercice 1 (TD)

Trouvons les erreurs dans les fonctions suivantes :

(A)
```
void total(int value1, value2, value3)
{
   return value1 + value2 + value3;
}
```

value2 : paramètre sans type

value3 : paramètre sans type

Instruction 'return' avec une valeur dans une fonction sans retour (void)

(B)
```
double average(int value1, int value2, int value3)
{
   double average;
   average = value1 + value2 + value3 / 3;
}
```

Le calcul de average est faux. Il faut écrire :

```
average = (value1 + value2 + value3) / 3.0;
```

Absence d'une instruction 'return' avec une valeur, dans une fonction retournant 'double'.

Par défaut, cette fonction retournera une valeur indéfinie égale à NaN (Not a Number)

(C)
```
void area(int length = 30, int width)
{
   return length * width;
}
```

L'argument par défaut est mal placé :

```
void area(int width, int length = 30)
```

Instruction 'return' avec une valeur, dans une fonction retournant 'void'

(D)
```
void getValue(int value&)
{
   cout << "Enter a value: ";
   cin >> value&;
}
```

Erreur dans la déclaration de l'argument : `(int & value)`

Erreur dans l'utilisation de l'argument passé par référence : `cin >> value;`

```
 // Fonctions surcharges
(E) int getValue()
    {
       int inputValue;
       cout << "Enter an integer: ";
       cin >> inputValue;
       return inputValue;
    }
(E) double getValue()
    {
       double inputValue;
       cout << "Enter a floating-point number: ";
       cin >> inputValue;
       return inputValue;
    }
```

Définitions ambiguës des deux fonctions "`getValue()`" : ce sont les arguments qui permettent de distinguer entre deux fonctions de même nom (par leur nombre et/ou leurs types).

```
(F) int f1()
    {
       double s;
       // ...
       return s;
    }
```

Fonction "`f1()`" : retournant un double alors qu'il doit retourner un int.

```
(G) f2(int i)
    {
        /* ... */
    }
```

Fonction "`f2()`" : n'a pas de type de retour.

```
(H) double square(double x) return x * x;
```

Oubli des accolades { }

```
double square(double x)
{
   return x * x;
}
```

Exercice 2 (TD)

1. Ecrire une fonction ordinaire C++ qui teste si ses deux arguments entiers sont de même parité. Tester votre fonction dans un programme « main ».

```
bool meme_parite(int x, int y)
{
   return x % 2 == y % 2;
}
```

2. Transformer cette fonction en une fonction en ligne.

```
inline bool meme_parite(int x, int y)
{
   return x % 2 == y % 2;
}
```

Exercice 3

Ecrire une fonction C++ ayant deux arguments entiers x et y, et qui divise x par y (qui est par défaut égal à 10).

1. en utilisant le passage des arguments par adresses. Tester votre fonction.

```
void diviser(int * x, int y = 10)
{
   *x = (*x) / y;
}
```

2. en utilisant le passage des arguments par référence. Tester votre fonction.

```
void diviser(int & x, int y = 10)
{
   x = x / y;
}
```

Exercice 4

1. Ecrire une fonction C++ ayant deux arguments : x (double) et y (entier), et qui calcule x^y en utilisant la formule $x^y = x * x * \dots * x$ (y fois). Tester votre fonction.

```
double puissance(double x, int y)
{
   double res = 1.0;

   for(int i = 1; i <= y; i++)
      res = res * x;
   return res;
}
```

2. Surcharger cette fonction en une fonction ayant deux arguments x et y de type double et en calculant x^y par l'utilisation de la formule $x^y = e^{y\log(x)}$.

```
double puissance(double x, double y)
{
   return exp(y * log(x));
}
```