



TD 1

Programmation sécurisée en C++ moderne	Enseignant	Chiheb Ameer ABID
--	------------	-------------------

Exercice 1.

Trouver les erreurs dans le programme suivant :

```
#include <iostream>
namespace A
{
    int n;
    void f();
    class C {
    public:
        int k;
        int get();
    };
}
namespace A
{
    int C::get()
    {
        cin >> k;
        return k;
    }
}
void f() {
    n++;
}
int main() {
    C c;
    using A::n;
    int n;
    n = c.get();
    return 0;
}
```

Exercice 2.

Définir un littéral `_rev` qui permet d'inverser une chaîne de caractères de type `std::string`.

```
int main() {
    std::cout << R("xyz" reversed is ")<< "xyz"_rev << R(") " << '\n';
    return 0;
}
```

Ce programme doit afficher :
"xyz" reversed is "zyx"

Exercice 3.

Écrire un littéral défini par l'utilisateur pour les degrés Celsius et un autre pour les degrés Fahrenheit, qui permettent de convertir une température d'une unité à l'autre. Par exemple, on devrait pouvoir écrire :

```
double t1{100.0_C}; //t1 is a Temperature object with value 100.0 in Celsius
double t2{212.0_F}; //t2 is a Temperature object with value 212.0 in Fahrenheit
cout<<t1.value<<" C = "<<t1.toFahrenheit().value<< " F\n"; //should print
100.0 C = 212.0 F
cout << t2.value << " F = " << t2.toCelsius().value << " C\n"; // should print
212.0 F = 100.0 C
```

Indice : vous pouvez utiliser la formule suivante pour convertir les degrés Celsius en degrés Fahrenheit :

$$F = C * 1.8 + 32$$

et la formule inverse pour convertir les degrés Fahrenheit en degrés Celsius :

$$C = (F - 32) / 1.8$$

Exercice 4.

Compléter ou re-écrire le code suivant selon ce qui est demandé dans les commentaires :

```
#include <iostream>
int main() {
    //Créer une chaîne s de type std::string initialisé avec la chaîne "abc"
    // Déclarer un objet de même type que s
    std::string
    /* Re-écrire le code suivant en C++ Moderne */
    const int N=10;
    int tab[N];
    for (int j=0;j<N;++j) {
        tab[j]=0;
    }
    tab[0]= 1;
    tab[1]= 2;
    tab[2]= 5;

    for (int j=0;j<N;++j) {
        std::cout<<tab[j]<<"\n";
    }
    /* ----- */
    //
    struct maStructure {
        int x;
        int y;
        int z;
    };

    // Créer un objet o de type maStructure, puis initialiser les champs comme
    // suit : o.y=2 et o.z=3
    // Récupérer les valeurs o.x et o.y dans deux variables a et b
}
```

Exercice 5.

On désire faire des statistiques sur des éleveurs de moutons. Pour chacun des N éleveurs on dispose : son numéro de carte d'identité (CIN), son nombre de brebis, son nombre de béliers, son nombre d'agneaux.

On demande d'écrire un programme en C++17 qui réalise :

1. La saisie de données relatives aux N éleveurs
2. On affiche pour chaque éleveur :
 - Le nombre total de ses animaux
 - Le pourcentage d'agneaux par rapport à l'ensemble d'animaux
3. Afficher le nombre d'animaux que possède en moyenne un éleveur
4. Le numéro de CIN et le nombre d'animaux de l'éleveur ayant le plus grand troupeau

Exercice 6.

1) Pour chacun des appels des fonctions, indiquer quelle fonction est appelée, ou bien erreur si l'appel est incorrect :

```
void foo1(const std::string& lr);
void foo1(std::string&& rv);
```

```
...
std::string s{"hello"};
...
foo1(s); //
foo1(std::move(s)); //
foo1(std::string{"Ok"}); //
```

```
void foo2(const std::string& lr);
...
```

```
std::string s{"hello"};
...
foo2(s); //
foo2(std::move(s)); //
foo2(std::string{"Ok"}); //
```

```
void foo3(std::string&);
...
std::string s{"hello"};
...
foo3(s); //
foo3(std::move(s)); //
foo3(std::string{"Ok"}); //
```

2) Expliquer la différence entre les deux appels :

```
void fooByVal(std::string str);
void fooByRRef(std::string&& str);
...
std::string s1{"hello"}, s2{"hello"};
...
fooByVal(std::move(s1)); //
fooByRRef(std::move(s2)); //
```

Exercice 7.

Utiliser les énumérations typés dans le programme ci-après.

```
#include <iostream>
#include <string>

enum Apple {
    RedDelicious,
    GoldenDelicious,
    Gala,
    Fuji,
    GrannySmith
};

enum Orange {
    Navel,
    Valencia,
    Hamlin,
    ParsonBrown
};

void printOrange(int orange)
{
    std::cout << "printOrange => " << orange << "\n";
}

int main(int argc, char**argv)
{
    std::cout << "\n\n----- Exercice 4 ----- \n\n";

    Apple apple = RedDelicious;
    Orange orange{Hamlin};

    std::cout << "apple = " << apple << "\n";
    std::cout << "orange = " << orange << "\n";

    apple = Gala;
    std::cout << "apple = " << apple << "\n";

    if (apple == orange)
    {
        std::cout << "apple == orange" << "\n";
        orange = Valencia;
    }
    else
    {
        std::cout << "apple != orange" << "\n";
    }

    printOrange(orange);
    printOrange(apple);

    std::cout << "Complete.\n";
    return 0;
}
```

Exercice 7.

L'un des jeux de hasard les plus populaires est un jeu de dés connu sous le nom de "craps", qui se joue dans les casinos et les ruelles du monde entier. Les règles du jeu sont simples :

Un joueur lance deux dés. Chaque dé a six faces contenant 1, 2, 3, 4, 5 et 6 points.

Après que les dés se soient immobilisés, la somme des points sur les deux faces supérieures est calculée. Si la somme est 7 ou 11 au premier lancer, le joueur gagne. Si la somme est 2, 3 ou 12 au premier lancer (appelé "craps"), le joueur perd (c'est-à-dire que la "maison" gagne). Si la somme est 4, 5, 6, 8, 9 ou 10 au premier lancer, alors cette somme devient le "point" du joueur. Pour gagner, vous devez continuer à lancer les dés jusqu'à ce que vous "réalisiez votre point". Le joueur perd en lançant un 7 avant de réaliser le point.

Dans les règles, notez que le joueur doit lancer deux dés au premier lancer et aux lancers suivants. Nous définirons une fonction rollDice pour lancer les dés et calculer et afficher leur somme. La fonction peut être appelée plusieurs fois : une fois pour le premier lancer du jeu et peut-être de nombreuses autres fois si le joueur ne gagne ni ne perd au premier lancer.

Implementer les jeux en définissant :

- la fonction rollDice() qui simule le lancement de deux dés, et donc renvoie la somme des points de deux dés lancés.
- la fonction firstRoll() qui simule un premier coup en renvoyant sous forme d'une structure le nombre de points et l'état du jeu : keepRolling, won, lost.
- la fonction contRoll() qui simule les autres coups.

Indication : On donne le code suivant pour générer aléatoirement une valeur de 1 à 6 :

```
static random_device rd; // used to seed the default_random_engine
static default_random_engine engine{rd()}; // rd() produces a seed
static uniform_int_distribution randomDie{1, 6};
const int dice{randomDie(engine)};
```