Week 6: Operator Overloading, Friend Function

Learning Materials: Chapter 8

1. **[You can use your solution code of previous lab task, Read the following instruction VERY CAREFULLY as you must refactor the code for today's task]**
2. **[Make the member function const and make the parameter const where it is necessary.]**
3. **[While passing an object, use the Reference variable in the parameter, Do not use the pass by value method.]**

Task 1

Create a class named "**Counter**". An object of the Counter class keeps track of the count. The object also stores the value of the **increment steps**. For example, if the increment step is 5, the count will be incremented by 5 each time an increment is done. To do these, declare the **necessary private member** data.

Note that there should not be any public methods to assign a value to count.

Implement the following public member functions (the task of the function is written after a hyphen):

- void setIncrementStep(int step_val): it sets the value of increment step in the appropriate member data. It will only set when the count is 0; otherwise, it will print in the console that it cannot set the increment step. The default value is 1. Restrict assigning a negative value. Keep the previous assigned value if a negative value is passed as an argument. However, the constructor assigns a default value if a negative value is passed.
- int getCount(): it returns the current count value.
- void increment(): it increments the count by increment step. For example: if the current count is 4 and the increment step value is 2 then executing the function will update the count to 6.
- void resetCount(int step = 1): it resets the value of count to 0 and incrementStep to step parameter.

Write the necessary member or non-member functions to achieve the following functionalities.

- Assume c1, c2, c3 are Counter objects. c1 = c2 + c3; After this statement, if the increment step of c2 and c3 is same, then the count value of c1 will be the summation of the count of c2 and c3. The increment step of c1 will be equal to c2 or c3. Otherwise, show a message in the console mentioning the increment step is not the same. No changes to c2 and c3.
- Implement >, <, >=, <=, ==, and != operator where they only compare the count value of the object and they follow their usual meaning. Example: c1==c2 will evaluate to true if the value of count variable of c1 and c2 are equal. Hints: Use bool as return type.
- c1+=c2; No changes to c2. After this statement, the count value of c1 will be the summation of the count of c1 and c2. No changes to c2. The increment step value will be the maximum of c1's and c2's.
- c1 = c2++; c1= ++c2; functions similar to increment() function.

```
The return policy should follow conventional rules.
void testFunction(const Counter& c)
{
     cout<< c.getCount();
}
```

Task 2

Create a class "**Coordinate**". An object of the **Coordinate** class stores the abscissa and ordinate (float type).
Implement the following **public** member functions (task of the function is written after a hyphen):

- **Define constructor (argumented and non-argumented), destructor, and display functions.**
- **float operator - (Coordinate c):** Distance from object c
  [Use the Euclidean distance formula, $d = \sqrt{[(x2 - x1)^2 + (y2 - y1)^2]}$.
- **float getDistance():** Distance from origin (0,0) coordinate
- **void move_x(float val)** - val will be added to member data abscissa
- **void move_y(float val)** - val will be added to member data ordinate
- **void move(float x_val, float y_val) -** x_val will be added to abscissa and y_val will be added to ordinate.

Write the necessary member or **non-member functions (use friend function if necessary)** to achieve the following functionalities.

- Assume c1, c2, c3 are Coordinate objects. Overload the following comparison operators >,<,>=,<=,==,!= where distance from the origin of each operand will be compared. Example c1 == c2 returns true when c1 contains (abscissa = 1, ordinate = 1) and c2 contains (abscissa = -1, ordinate = -1)

In the main function, create an array of 10 Coordinate objects. Write a function that will randomly assign abscissa and ordinate value to these objects. Write a sort function to sort the list of coordinate objects (small to large) based on distance from the origin.

```
void randomAssignment(Coordinate c[], int size){
    ///Write code to assign random abscissa and ordinate to elements of c
}
void sort(Coordinate c[], int size){
}
int main(){
    Coordinate coord[10];
    randomAssignment(coord,10);
    sort(coord,10);
    for(int i=0;i<10;i++)
    {
        coord[i].display();
    }

}
```

Task 3

Temperature is a measure of the thermal energy of a system. It can be expressed in Celsius $(°C)$, Fahrenheit $(°F)$, or Kelvin (k) scale. Absolute zero (0 (k)) is the lowest limit of the thermodynamic temperature scale.

Create three classes Celsius, Fahrenheit, and Kelvin – that store the temperature value. The temperature value can be a fractional value but will not store a value lower than absolute zero.

The conversion formula among the units are

$$(°F) = (°C) * 9/5 + 32;$$

$$(k) = (°C) + 273.15$$

Implement the following member functions for each class and select appropriate data types for the parameters that satisfy the provided functionalities:

- **Add constructor to initialize the temperature. (zero or argumented)**
- **assign** - this function sets the value for the data members to a particular temperature.
- **display**() - this member function will display the temperature in its current form. For example: "**The temperature is 100 Celsius.**"
- **Overload type operation so that conversion between these classes can take place.**

```
/// Some example code
#include <string>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()

// Function to generate random string
std::string generateRandomString(int length) {
    std::string randomString;
    const char alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int alphabetSize = sizeof(alphabet) - 1; // Exclude null character

       srand(static_cast<unsigned  int>(time(0)));  //  Seed  the  random  number
generator

    for (int i = 0; i < length; ++i) {
        randomString += alphabet[rand() % alphabetSize];
    }

    return randomString;
}

int randomInRange(int min, int max) {
    // Ensure min is less than or equal to max
    if (min > max) {
        std::swap(min, max); // Swap if min is greater than max
    }

    return rand() % (max - min + 1) + min;
}

// Function to generate a random double within a given range [min, max]
double randomInRange(double min, double max) {
    // Ensure min is less than or equal to max
    if (min > max) {
        std::swap(min, max); // Swap if min is greater than max
    }

    // Generate a random double between 0 and 1
    double randomFraction = static_cast<double>(rand()) / RAND_MAX;

    // Scale and shift the random value to the desired range
    return min + randomFraction * (max - min);
}
```