EE3093 Tutorial: C++ week 2

Instructions to users

This tutorial can be completed up to 4 degrees of complexity (one for each section); make sure you have understood and completed a section (i.e. code implemented and tested successfully) before moving on to the next one.

Part 1: Basic - Create an empty CPP project

Create a New Project using the IDE of your choice

Step-by-step guidance provided for Visual Studio and for Dev-C in separate Instructions files (PDF).

Test your project

In your main, call the following function (having included the relevant header files). You should be able to compile and run the program. Check that the on-screen output is correct:

The following test routine **test_part_1()**, called by **main**, allocates one instance for each of: **rectangle**, **right_triangle**, **rectangleWcolor** and **right_triangleWcolor;** the routine fills each object with random values and prints the content to screen.

```
// Which header files are necessary?
// include all relevant files (#include “filename.h”) and system libraries.
void test_part1()
{
    rectangle obj1;
    right_triangle obj2;
    rectangleWcolor obj3;
    right_triangleWcolor obj4;
    cout << "object 1:" << endl;
    obj1.inputRandomSides();
    obj1.printRectangleInfo();
    cout << "object 1 (again):" << endl;
    //obj1.reset();
    obj1.inputRandomSides();
    obj1.printRectangleInfo();
    cout << endl << endl;
    cout << "object 2:" << endl;
    obj2.inputRandomSides();
    obj2.printTriangleInfo();
    cout << endl << endl;
    cout << "object 3:" << endl;
    obj3.inputRandomValues();
    obj3.printInfo();
    cout << endl << endl;
    cout << "object 4:" << endl;
    obj4.inputRandomValues();
    obj4.printInfo();
    cout << "object 4 (again):" << endl;
    //obj4.reset();
    obj4.inputRandomValues();
    obj4.printInfo();
}
```

Next: Modify all the classes mentioned above so that each supports a public function **void reset()** that allows resetting a previously initialized object; after being reset, the object can accept new input values for the sides and/or color.

In the function test_part1() above, uncomment the lines where **reset()** is called; compile, run the code and verify the on-screen output is correct.

Verify that the constructor is called for each object (and at what point in the program the call happens). Enable DEBUGGING your project, as shown in class and add a breakpoint in the opening line of each class constructor. To simplify the process, you may add an opening line to the constructor of Class X similar to: *cout << "Class X constructor invoked" << endl;* where there isn't a constructor (why?) you can write one.

Which constructors are called for an instance of right_triangleWcolor? In which order? Why?

Part 2: Lower Intermediate

Implement a class **square**, similar to the class rectangle (you may implement square by including a rectangle object). Write and include in your project the header file (optionally also a cpp file) implementing class Square with (public) member functions described below:

square();// constructor
double **getArea**(); // as in rectangle
bool **isInitialized**(); // as in rectangle
void **printInfo**(); // as in rectangle
double **getSide**(); // as in rectangle, but there is only one side here
void **inputSide**(double in_side); // see above
void **inputSideFromKeyboard**(); // see above
void **inputRandomSide**(double max_val=100); // see above
void **reset**(); // as described in Part 1

Then implement a class **squareWcolor** similar to rectangleWcolor.

Modify test_part_1() to test your implementation of these two classes.

Part 3: Higher Intermediate

Write and include in your project the header file (optionally also a cpp file) implementing class **squareWcolor;** this is similar to the "Rectangle and Triangle with colors" implemented previously. Public member functions to implement:

// a constructor if neded;
void **inputFromKeyboard**(); // allows the user to input data for the triangle with colors from keyboard.
void **printInfo**(); // print to screen data for the square with colors.
void **inputRandomValues**(double max_val=100); //assign random values to the square with colors.

Implement routine **test_part_3()** called by main to test your squareWcolor implementation.

In **main**, allocate **array_rc[]**, an array that holds MAX_ELEMENTS of **rectangleWcolor** objects (begin by setting MAX_ELEMENTS = 100); also allocate **array_sq[]**, an array that holds max_elements of **squareWcolor** objects. Implement routine void **test_part_3b(**rectangleWcolor array_rc[], int MAX_ELEMENTS**)**, which is called by main to fill each entry in array_rc[] with random values. Then implement the following routine (called by main):
void **test_part_3c(**rectangleWcolor array_rc[], squareWcolor array_sq[], int MAX_ELEMENTS**)**
to initialize each entry array_sq[i] as a square that has the same area and color as the rectangle array_rc[i].

Part 4: Advanced

In **main**, allocate **array_sq_sorted[]**, of the same type and size of array_sq[] then write a routine:
void **test_part_4(**squareWcolor array_sq[], squareWcolor array_sq_sorted[], int MAX_ELEMENTS**)** that is called by main to fill array_sq_sorted[] with the squares having the same value (side & color) as in array_sq[], with the difference that elements of array_sq_sorted[] are sorted (ascendingly) according to their color (i.e. ascending values of the enumerator representing the color).

If your implementation of test_part_4(…) does not support the following functionality, implement an improved version, called **test_part_4b(…)**, where items in array_sq_sorted[] are sorted according to their color (ascendingly) and those with the same color are sorted according to their area (ascendingly).

Print to screen the content of array_sq[] and of array_sq_sorted[] obtained by your implementation of the above.

Note on possible implementation:

For the implementation of test_part_4(…) and test_part_4b(…) consider the option of filling array_sq[] via a "sorted insertion"; this is explained with the following **pseudocode** example:

In the following example, an array can accommodate 10 letters; the 7 opening elements are initialized and sorted; The array can accommodate 3 additional elements, those that are "empty" (i.e. have not yet been initialized with a valid input) and are denoted as "*":
Arrayexample[10] = {b, e, g, k, k, o, p,*,*,*};

(Ascending alphabetically) sorted insertion of element "h" is performed as follows:

(**1**) The insertion point is found:

Arrayexample = {b, e, g, k, k, o, p,*,*,*};

(**2**) A gap is created: all elements past the insertion point are shifted to the right; an "empty" element (*) is placed at the insertion point (there are still 3 empty elements in total):
Arrayexample = {b, e, g,*****, k, k, o, p,*,*}.

(**3**) The new element is inserted (occupying one previously "empty" element) and the array remains sorted:
Arrayexample = {b, e, g, **h**, k, k, o, p,*,*}.