Lab 10: Operator Overloading

CPSC 1021 – Spring 2020

CLEMSON School of COMPUTING

Introduction

During this lab you will:
1. Work with a class that contains a 2d vector representing pixels.
2. Read files and store the content in the 2d vector of your class objects.
3. Practice overloading operators.
4. Create a tarball.

In this lab, you will to process several ascii files, combine them together using simple arithmetic operation, and output the result to the terminal.

Part I – Reading from the file

The file reading process is no different than what you have done in the previous labs. Today, you need to read file line by line, then convert each char of the string into integer and store it into a 2d vector. The conversion can be done this way:

```
1.  char a = '4';
2.  int ia = a - '0';
```

Part II – 2d vector, vector<vector<int>>

In this lab all files contain integers for your class, and a 2d vector of type vector<vector<int>> should be included.

Vectors are known as dynamic arrays with the ability to resize themselves automatically when an element is inserted or deleted, with their storage being handled automatically by the container.

Vector of Vectors is a two-dimensional vector with a variable number of rows where each row is also a vector. Each index of vector stores a vector which can be traversed and accessed using iterators. It is similar to an Array of Vectors but with dynamic properties.

Below is the example to demonstrate insertion into a vector of vectors.

```
1.  // C++ program to demonstrate insertion
2.  // into a vector of vectors
3.
4.  #include <iostream>
5.  #include <vector>
6.  using namespace std;
7.
8.  // Defining the rows and columns of
9.  // vector of vectors
10. #define ROW 4
11. #define COL 5
12.
13. int main()
```

```
{
    // Initializing the vector of vectors
    vector<vector<int> > vec;

    // Elements to insert in column
    int num = 10;

    // Inserting elements into vector
    for (int i = 0; i < ROW; i++) {
        // Vector to store column elements
        vector<int> v1;

        for (int j = 0; j < COL; j++) {
            v1.push_back(num);
            num += 5;
        }

        // Pushing back above 1D vector
        // to create the 2D vector
        vec.push_back(v1);
    }

    // Displaying the 2D vector
    for (int i = 0; i < vec.size(); i++) {
        for (int j = 0; j < vec[i].size(); j++)
            cout << vec[i][j] << " ";
        cout << endl;   }
    return 0;
}
```

Part III – Operator Overloading

In this lab, you need to overload 3 operators, +, - , and << .

Overloading operator+

The meaning of an operator is always the same for variables of basic types, such as *int, float*, *double,* etc. For example: to add two integers, + operator is used.

However, for user-defined types, like objects, you can redefine the way operator works. For example, if there are two objects that contains string as its data member, you can redefine the meaning of + operator and use it to concatenate these strings.

This feature in C++ programming that allows programmer to redefine the meaning of an operator (when they operate on class objects) is known as operator overloading.

Below is an example of two forms of overloading operator+ . Overloading operator- will be similar.

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

class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }
```

```
9.        void setLength( double len ) {
10.          length = len;
11.       }
12.       void setBreadth( double bre ) {
13.          breadth = bre;
14.       }
15.       void setHeight( double hei ) {
16.          height = hei;
17.       }
18.
19.       // Overload + operator to add two Box objects.
20.       Box operator+(const Box& b) {
21.          Box box;
22.          box.length = length + b.length;
23.          box.breadth = breadth + b.breadth;
24.          box.height = height + b.height;
25.          return box;
26.       }
27.
28.       // Overload + operator to add a integer Box objects.
29.       Box operator+(int num) {
30.          Box box;
31.          box.length = length + num;
32.          box.breadth = breadth + num;
33.          box.height = height + num;
34.          return box;
35.       }
36.
37.
38.    private:
39.       double length;      // Length of a box
40.       double breadth;     // Breadth of a box
41.       double height;      // Height of a box
42. };
43.
44. // Main function for the program
45. int main() {
46.    Box Box1;                // Declare Box1 of type Box
47.    Box Box2;                // Declare Box2 of type Box
48.    Box Box3;                // Declare Box3 of type Box
49.    double volume = 0.0;     // Store the volume of a box here
50.
51.    // box 1 specification
52.    Box1.setLength(6.0);
53.    Box1.setBreadth(7.0);
54.    Box1.setHeight(5.0);
55.
56.    // box 2 specification
57.    Box2.setLength(12.0);
58.    Box2.setBreadth(13.0);
59.    Box2.setHeight(10.0);
60.
61.    // volume of box 1
62.    volume = Box1.getVolume();
63.    cout << "Volume of Box1 : " << volume <<endl;
64.
65.    // volume of box 2
66.    volume = Box2.getVolume();
67.    cout << "Volume of Box2 : " << volume <<endl;
68.
69.    // Add two object as follows:
70.    Box3 = Box1 + Box2;
71.
72.    // volume of box 3
73.    volume = Box3.getVolume();
```

```
74.    cout << "Volume of Box3 : " << volume <<endl;
75.
76.    return 0;
77. }
```

Important:
The Box class in the above example contains only doubles, however, in your class the data member is vector<vector<int>>, if you do not allocate memory for your 2d vector, you cannot use [ ][ ] to assign new value, instead, you need manually call the push_back() function to put the new value into your 2d array.

Overloading operator<<

Overloading operator<< is similar to overloading operator+ (they are both binary operators), except that the parameter types are different.

Consider the expression std::cout << box. If the operator is <<, what are the operands? The left operand is the std::cout object, and the right operand is your Point class object. std::cout is actually an object of type std::ostream. Therefore, our overloaded function will look like this:

```
1.  // std::ostream is the type for object std::cout
2.  friend std::ostream& operator<< (std::ostream &out, const Box &box);
```

Implementation of operator<< for Box class is fairly straightforward, because C++ already knows how to output doubles using operator<<, and our members are all doubles, we can simply use operator<< to output the member variables of box. Here is the above Box class with the overloaded operator<<.

```
1.  // std::ostream is the type for object std::cout
2.  friend std::ostream& operator<< (std::ostream &out, const Box &box)
3.  {
4.      out<<box.length<<endl;
5.      out<<box.breadth<<endl;
6.      out<<box.height<<endl;
7.      return os;
8.  }
```

Part IV- Lab assignment.

In this lab you will do the following

1. Download the starter kit archive lab10.zip. The files *pixelArt.h* and *driver.cpp* are provided, and you do not need to modify them. File *pixelArt.cpp* is provided, but the methods are missing their bodies.

2. Implement the missing methods and overload the operators for the pixelArt class.

3. Run your driver and send the output to the terminal. The file named *result* is provided for reference in the starter kit as well. Your output to the screen should look exactly the same.

What/How to submit

When done, please remove all object files, create a tarball containing all .cpp and .h files and submit to canvas.

NOTE: If you submit the archive that cannot be open, or is corrupt, you will not be given the opportunity to redo the work and resubmit. You have one shot to get it right.

That's it, you are done!