CSCI 335 **Summer 2023**

First programming assignment : Big Five
100 points
Homework due: 11:59pm June 16th, 2023

Please follow the blackboard instructions on writing and submitting programming assignments. It should compile to receive any credit.
Submit only the files requested in the deliverables at the bottom of this description to blackboard by the deadline.

Programming: The big five

Create and test a class called **Points2D**. This class describes a sequence of 2D points. For example (1, 3), (4, 5) is a sequence of two points, where each coordinate is an integer. (1.2, 3.4), (5.6, 10.1), (11.1, 12.0) is a sequence of three points where each coordinate is a double. A sequence can have any size. An empty sequence has size 0.

The purpose of this assignment is to have you create a Points2D class from scratch with limited help from the STL. Since Points2D can have arbitrary size, you should use pointers. The private data members should be:

```
size_t size;  std::array<Object, 2> *sequence_;
```

Object is the template type parameter (i.e. int, double, etc.). An initial piece of code with the structure of the class is provided. Do not change the data representation (for instance do not use a vector or list to represent the sequence_).

Pay special attention to Weiss's **"big five",** the destructor, copy constructor, copy assignment operator, move constructor and move assignment operator.

Included are the two files (`points2d.h`, `test_points2d.cc`) you will need, as well as the Makefile. Do not modify the Makefile or the file names. Do not modify the `test_points2d.cc` file except by changing or adding include files if needed. You can comment in the main file the parts you didn't complete. The `points2d.h` file is not complete. In the file it is explained where to provide changes.

At the end of this document, we are also providing you with a sample input file `test_input_file.txt` with explanations on how to use it.

Make sure to think about the inputs - valid and invalid - that could be provided. Use the "`Testing Document.pdf`" document as a guide. As such, any invalid input should be rejected with an error message (cerr) and abort the program. The output for the invalid output should be "`ERROR`".

This assignment will help you revisit constructors, destructors, overloading of operators, and templates. Follow a consistent C++ coding style, for instance
https://google.github.io/styleguide/cppguide.html

PART 1 [70 points]

Implement the "big five". Add the stream insertion (<<) and extraction (>>) operators.

Demonstrate that you are able to read and write correctly by including the following code in the main file. The code is already provided for you in the main file. You can comment parts of it as you are testing your implementation. For full credit all functions should work.

```
void TestPart1() {

  Points2D<int> a, b;  // Two empty Points2D are created.

  cout << a.size() << " " << b.size() << endl; // yields 0 0.
  const array<int, 2> a_point2{{7, 10}};
  Points2D<int> d{a_point2}; // Sequence (7, 10) should be  created.
  cout << d;  // Should just print (7, 10)
  cout << "Enter a sequence of points (integer)" << endl;
  cin >> a;         // User enters a set of points in the form:
                    // 3 7 4 3 2 1 10
                    // 3 specifies number of points. Points are the pairs
                    // (7, 4) (3, 2) and (1, 10)
  cout << "Output1: " << endl;
  cout << a;  // Output should be what user entered.
  cout << "Enter a sequence of points (integer)" << endl;
  cin >> b;   // Enter another sequence.
  cout << "Output2: " << endl;
  cout << b;
  Points2D<int> c{a};  // Calls copy constructor for c.
  cout << "After copy constructor1 c{a}: " << endl;
  cout << c;
  cout << a;
  a = b;  // Should call the copy assignment operator for a.
  cout << "After assignment a = b" << endl;
  cout << a;
  Points2D<int> e = move(c);  // Move constructor for d.
  cout << "After e = move(c) " << endl;
  cout << e;
  cout << c;
  cout << "After a = move(e) " << endl;
  a = move(e);  // Move assignment operator for a.
  cout << a;
  cout << e;
}
```

PART 2 [30 points]

Overload the + and [] operators for your Points2D class. Test with the following code. The code is already provided for you in the main file. You can comment parts of it as you are testing your implementation. For full credit all functions should work.

```
void TestPart2() {
Points2D<double> a, b;
    cout << "Enter a sequence of points (double)" << endl;
    cin >> a;  // User provides input for Points2D a.
    cout << a;
    cout << "Enter a sequence of points (double)" << endl;
    cin >> b;  // User provides input for Points2D b.
    cout << b << endl;
    cout << "Result of a + b" << endl;
    cout << a + b << endl;
    Points2D<double> d = a + b;
    cout << "Result of d = a + b" << endl;
    cout << d;
    cout << "Second element in a: " << endl;
    cout << a[1][0] << ", " << a[1][1] << endl;  // Should print the 2nd
element.
}  // End of TestPart2
```

Do not send to the standard output anything else other than what is asked. Do not pause for input, or interact with the user.

You can run the program as follows:

```
./test_points2d
```

In that case you should enter the required data in the standard input.

You can also run it as follows

```
./test_points2d < test_input_file.txt
```

In that case the output should be the one contained in `expected_output.txt`

Note that the code for test_points2 is the same in both cases. In the second call the shell redirects the contents of test_input_file.txt to the standard input.

SUBMISSION

On or before the due date you should submit the following to **blackboard**:

- Submit your modified `points2d.h` file.
- Submit a single README.md file, for which you can find more specific details in the Programming Rules. The README file is an opportunity to explain your solution to the problem, how you approached it, what was simple and why (if anything) and what was difficult and why (if anything) Note about rubric: As mentioned in class there will be visible testing and invisible testing (test cases unknown to the students ahead of time). Up to 60 points will be assigned for the visible testing and this will be visible immediately upon submission. 12 points will be assigned for the invisible tests.

The final 28 points (code quality) will be assigned after manual grading and the rubric is follows:

- Style Structure: 8 points
- Documentation : 8 points
- Code Efficiency: 8 points
- README: 4 points