

# ACTIVITY NO.

1

REVIEW OF C++ PROGRAMMING											
Course Code: CPE010	Program: Computer Engineering										
Course Title: Data Structures and Algorithms	Date Performed: 7/29/25										
Section: CPE21S4	Date Submitted: 7/29/25										
Name: ALCANTARA, JASON P.	Instructor: Engr, Jimlord Quejado										
1. Objective(s)											
<input type="checkbox"/> Implement basic programming and OOP in C++											
2. Intended Learning Outcomes (ILOs)											
<p>After this module, the student should be able to:</p> <ol style="list-style-type: none"> <li>Create code that follows the basic C++ code structure;</li> <li>Implement appropriate class definition and instances based on given requirements;</li> <li>Solve different problems using the C++ programming language.</li> </ol>											
3. Discussion											
<p><u>Part A: Introduction to C++ Code</u></p> <p>Structure of C++ Code</p> <table border="1"> <thead> <tr> <th>Sections</th> <th>Sample Code</th> </tr> </thead> <tbody> <tr> <td>Header File Declaration Section</td> <td><code>#include&lt;iostream&gt; using namespace std;</code></td> </tr> <tr> <td>Global Declaration Section</td> <td><code>int count = 0;</code></td> </tr> <tr> <td>Class Declaration and Method Definition Section</td> <td> <pre>class rectangle{ private:     double recLength, recWidth; public:     rectangle(double L, double W); void     setLength(double L); void setWidth(double W);     double getPerimeter(); };</pre> </td> </tr> <tr> <td>Main Function</td> <td> <pre>int main(){     rectangle shape1(2, 5);    std::cout &lt;&lt; "The perimeter of the rectangle is " &lt;&lt;     shape1.getPerimeter() &lt;&lt; ".\n";     std::cout &lt;&lt; count &lt;&lt; " number of objects created."; return 0; }</pre> </td> </tr> </tbody> </table>		Sections	Sample Code	Header File Declaration Section	<code>#include&lt;iostream&gt; using namespace std;</code>	Global Declaration Section	<code>int count = 0;</code>	Class Declaration and Method Definition Section	<pre>class rectangle{ private:     double recLength, recWidth; public:     rectangle(double L, double W); void     setLength(double L); void setWidth(double W);     double getPerimeter(); };</pre>	Main Function	<pre>int main(){     rectangle shape1(2, 5);    std::cout &lt;&lt; "The perimeter of the rectangle is " &lt;&lt;     shape1.getPerimeter() &lt;&lt; ".\n";     std::cout &lt;&lt; count &lt;&lt; " number of objects created."; return 0; }</pre>
Sections	Sample Code										
Header File Declaration Section	<code>#include&lt;iostream&gt; using namespace std;</code>										
Global Declaration Section	<code>int count = 0;</code>										
Class Declaration and Method Definition Section	<pre>class rectangle{ private:     double recLength, recWidth; public:     rectangle(double L, double W); void     setLength(double L); void setWidth(double W);     double getPerimeter(); };</pre>										
Main Function	<pre>int main(){     rectangle shape1(2, 5);    std::cout &lt;&lt; "The perimeter of the rectangle is " &lt;&lt;     shape1.getPerimeter() &lt;&lt; ".\n";     std::cout &lt;&lt; count &lt;&lt; " number of objects created."; return 0; }</pre>										

Method Definition	<pre>rectangle::rectangle(double L, double W) {    recLength = L; recWidth = W;    count++; }</pre>
-------------------	---

```

void rectangle::setLength(double L) {   recLength = L;
}
void rectangle::setWidth(double W) {   recWidth = W;
} double rectangle::getPerimeter() {   return
(2*recLength) + (2*recWidth);
}

```

It is not required for all sections to have code for every use-case. However, for best practices you would prefer to have an overall structure to follow to increase code readability and reusability.

## Data Types

- d. Primary Data Type: int, float, char and void
- e. User defined data type: structure, union, class, enumeration
- f. Derived data type: array, function, pointer, reference

## Local & Global Variables

```

#include <iostream> using
namespace std;
int globalVal = 0; //Global Variable

int main(){   int localVal = 5; //Local Variable
    std::cout << "Global Variable has value " << globalVal << ".\n";   std::cout << "Local Variable
has value " << localVal << ".\n";

    return 0;
}

```

## Operators

Arithmetic	Relational	Logical
Addition +	Greater than >	AND
Subtraction –	Less than <	&&
Multiplication *	Greater than or equal >=	OR
Division /	Less than or equal <=	NOT !
Modulo %	Equal ==	
Increment ++	Not equal !=	
Decrement --		

## Bitwise Operators

Let A = 60 and B = 13. Binary values are as follows:

```

A = 0011 1100
B = 0000 1101

```

Bitwise AND -> &	A & B	0000 1100
Bitwise OR ->	A   B	0011 1101
Bitwise XOR -> ^	A ^ B	0011 0001
Bitwise Complement -> ~	~A	1100 0011

## Assignment Operator

Assign a value to a variable. Example:

Assign the value 20 to a variable A.  

```
int A = 20;
```

The assignment operator is a basic component denoted as “=”.

## Part B: Classes and Objects using C++

To create a class use the class keyword. Syntax is:

```
class myClass { public:
    int myNum; string
    myString;
};
```

public here is an access specifier. It indicates that the attributes and methods listed under it are accessible outside the class. A simple table is provided below to summarize the access specifiers used in c++.

Specifiers	Within same class	In derived class	Outside the class
private	Yes	No	No
protected	Yes	Yes	No
public	Yes	Yes	Yes

We can then create an object from this class:

```
int main(){
    //this creates the object myClass object1;

    //this accesses the public attributes object1.myNum = 5;
    object1.myString = "Sample";

    return 0;
}
```

## 4. Materials and Equipment

Personal Computer with C++ IDE

Recommended IDE:

- CLion (must use TIP email to download)
- DevC++ (use the embarcadero fork or configure to C++17)

5. Procedure

## ILO A: Create Code That Follows the Basic C++ Code Structure

For this activity, you have to demonstrate the use of a function prototype. The section on class declaration and method definition will be used for the function prototype and the function will be defined in the follow method definition section after the main function.

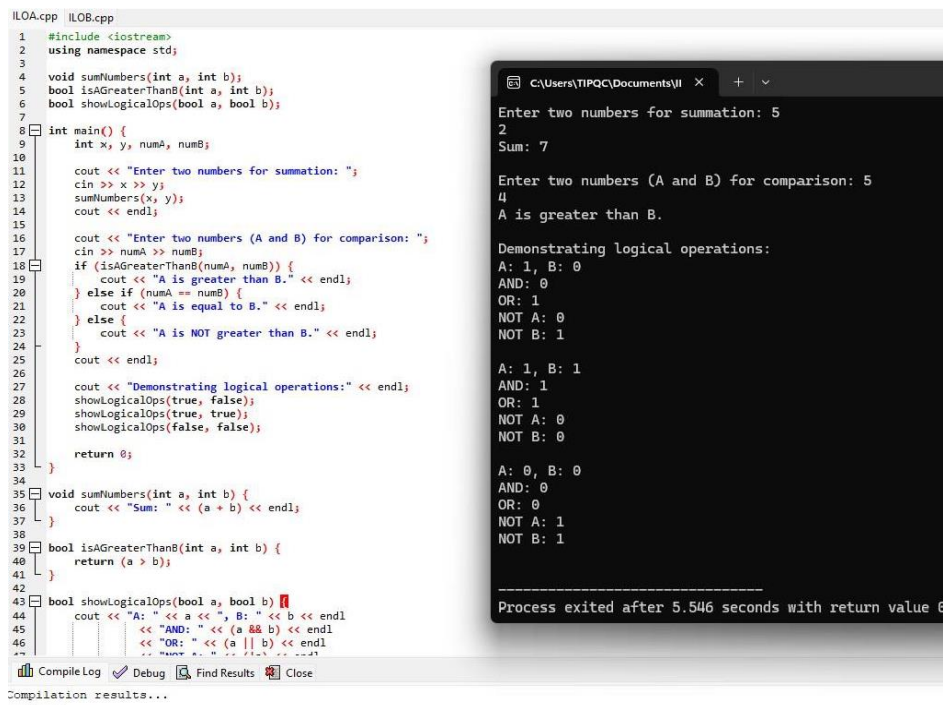
A function prototype in c++ is a declaration of the name, parameters and return type of the function before its definition. Write a C++ code the satisfies the following:

- Create a function that will take two numbers and display the sum.
- Create a function that will return whether variable A is greater than variable B.
- Create a function that will take two Boolean values and display the result of all logical operations then return true if it was a success.

Note:

- The driver program must call each function.
- The definitions must be after the main function.

### **OUTPUT:**



```
ILOA.cpp ILOA.cpp
1 #include <iostream>
2 using namespace std;
3
4 void sumNumbers(int a, int b);
5 bool isAGreaterThanB(int a, int b);
6 bool showLogicalOps(bool a, bool b);
7
8 int main() {
9     int x, y, numA, numB;
10
11     cout << "Enter two numbers for summation: ";
12     cin >> x >> y;
13     sumNumbers(x, y);
14     cout << endl;
15
16     cout << "Enter two numbers (A and B) for comparison: ";
17     cin >> numA >> numB;
18     if (isAGreaterThanB(numA, numB)) {
19         cout << "A is greater than B." << endl;
20     } else if (numA == numB) {
21         cout << "A is equal to B." << endl;
22     } else {
23         cout << "A is NOT greater than B." << endl;
24     }
25     cout << endl;
26
27     cout << "Demonstrating logical operations:" << endl;
28     showLogicalOps(true, false);
29     showLogicalOps(true, true);
30     showLogicalOps(false, false);
31
32     return 0;
33 }
34
35 void sumNumbers(int a, int b) {
36     cout << "Sum: " << (a + b) << endl;
37 }
38
39 bool isAGreaterThanB(int a, int b) {
40     return (a > b);
41 }
42
43 bool showLogicalOps(bool a, bool b) {
44     cout << "A: " << a << ", B: " << b << endl;
45     cout << "AND: " << (a && b) << endl;
46     cout << "OR: " << (a || b) << endl;
47     cout << "NOT A: " << !a << endl;
48     cout << "NOT B: " << !b << endl;
49 }
50
51 Compile Log Debug Find Results Close
Compilation results...
```

```
C:\Users\TIPQC\Documents\ILOA
Enter two numbers for summation: 5
2
Sum: 7

Enter two numbers (A and B) for comparison: 5
4
A is greater than B.

Demonstrating logical operations:
A: 1, B: 0
AND: 0
OR: 1
NOT A: 0
NOT B: 1

A: 1, B: 1
AND: 1
OR: 1
NOT A: 0
NOT B: 0

A: 0, B: 0
AND: 0
OR: 0
NOT A: 1
NOT B: 1

Process exited after 5.546 seconds with return value 0
```

## ILO B: Implement Appropriate Class Definition and Instances Based on Given Requirements

In this section, the initial implementation for a class triangle will be implemented. The step-by-step procedure is shown below:

Step 1. Include the necessary header files. For this one, we only need `#include <iostream>`

Step 2. Create the triangle class. Assign it with private variables: totalAngle, angleA, angleB, and angleC.

```
class Triangle{ private:  
    double totalAngle, angleA, angleB, angleC;
```

Step 3. We then create public methods. The constructor must allow for creation of the object with 3 initial angles to be stored in our previously defined variables angleA, angleB and angleC. Another method has to be made if the user wants to change the initial values, this will also accept 3 arguments to change the values in angleA, angleB and angleC. Lastly, a function to validate whether the given values make our shape an actual triangle.

```
public:  
    Triangle(double A, double B, double C); void setAngles(double A,  
double B, double C); const bool validateTriangle(); };
```

Step 4. Define the methods.

```
Triangle::Triangle(double A, double B, double C) { angleA = A;
```

```

    angleB = B;   angleC = C;
    totalAngle = A+B+C;
}
void Triangle::setAngles(double A, double B, double C) {   angleA = A;   angleB
= B;   angleC = C;
    totalAngle = A+B+C;
}
const bool Triangle::validateTriangle() {   return
(totalAngle <= 180); }

```

Step 5. Create the driver code.

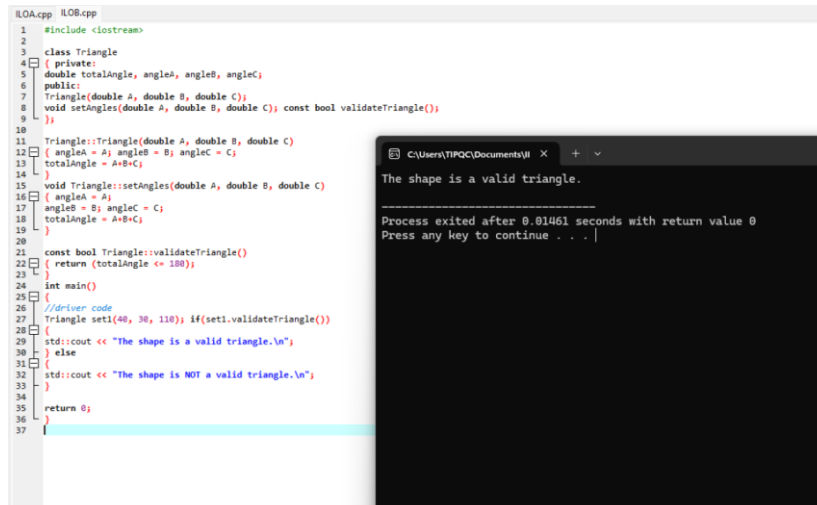
```

int main(){ //driver code
    Triangle set1(40, 30, 110);   if(set1.validateTriangle()){
        std::cout << "The shape is a valid triangle.\n";
    } else {
        std::cout << "The shape is NOT a valid triangle.\n";   }   return 0;
}

```

Include the output of running this code in section 6. Note your observations and comments.

## OUTPUT:



The screenshot shows a C++ IDE with two windows. The left window displays the source code for `ILOA.cpp`, which includes the `Triangle` class definition and a `main` function. The right window shows the program's output, which is "The shape is a valid triangle." followed by a message indicating the process exited after 0.01461 seconds with a return value of 0.

```

ILOA.cpp: ILOA.cpp
1  #include <iostream>
2
3  class Triangle
4  { private:
5      double totalAngle, angleA, angleB, angleC;
6  public:
7      Triangle(double A, double B, double C);
8      void setAngles(double A, double B, double C); const bool validateTriangle();
9  };
10
11  Triangle::Triangle(double A, double B, double C)
12  {   angleA = A;   angleB = B;   angleC = C;
13      totalAngle = A+B+C;
14  }
15  void Triangle::setAngles(double A, double B, double C)
16  {   angleA = A;
17      angleB = B;   angleC = C;
18      totalAngle = A+B+C;
19  }
20
21  const bool Triangle::validateTriangle()
22  {   return (totalAngle <= 180);
23  }
24
25  int main()
26  { //driver code
27      Triangle set1(40, 30, 110);   if(set1.validateTriangle())
28      {
29          std::cout << "The shape is a valid triangle.\n";
30      } else
31      {
32          std::cout << "The shape is NOT a valid triangle.\n";
33      }
34
35      return 0;
36  }
37

```

```

C:\Users\TIPOC\Documents\ILOA
The shape is a valid triangle.

-----
Process exited after 0.01461 seconds with return value 0
Press any key to continue . . .

```

## 6. Output



Sections	Answer
Header File Declaration Section	
Global Declaration Section	
Class Declaration and Method Definition Section	
Main Function	
Method Definition	

Table 1-1. C++ Structure Code for Answer

--

Table 1-2. ILO B output observations and comments.

## 7. Supplementary Activity

### ILO C: Solve Different Problems using the C++ Programming Language

The supplementary activities are meant to gauge your ability in using C++. The problems below range from easy to intermediate to advanced problems. Note your difficulties after answering the problems below.

1. Create a C++ program to swap the two numbers in different variables.

```

ILOA.cpp ILOB.cpp
1  #include <iostream>
2
3  class Triangle
4  { private:
5      double totalAngle, angleA, angleB, angleC;
6  public:
7      Triangle(double A, double B, double C);
8      void setAngles(double A, double B, double C); const bool validateTriangle();
9  }
10
11 Triangle::Triangle(double A, double B, double C)
12 { angleA = A; angleB = B; angleC = C;
13   totalAngle = A+B+C;
14 }
15 void Triangle::setAngles(double A, double B, double C)
16 { angleA = A;
17   angleB = B; angleC = C;
18   totalAngle = A+B+C;
19 }
20
21 const bool Triangle::validateTriangle()
22 { return (totalAngle <= 180);
23 }
24
25 int main()
26 {
27   //Driver code
28   Triangle set1(40, 30, 110); if(set1.validateTriangle())
29   {
30     std::cout << "The shape is a valid triangle.\n";
31   }
32   else
33   {
34     std::cout << "The shape is NOT a valid triangle.\n";
35   }
36   return 0;
37 }

```

The shape is a valid triangle.

Process exited after 0.01461 seconds with return value 0  
Press any key to continue . . .

2. Create a C++ program that has a function to convert temperature in Kelvin to Fahrenheit.

```
#include <iostream>

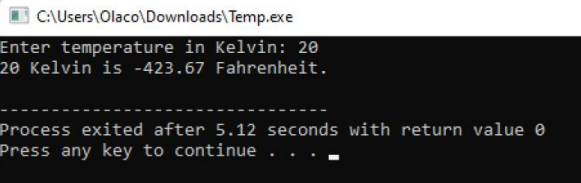
double kelvinToFahrenheit(double kelvin) {
    return (kelvin - 273.15) * 1.8 + 32;
}

int main() {
    double kelvinTemp;

    std::cout << "Enter temperature in Kelvin: ";
    std::cin >> kelvinTemp;

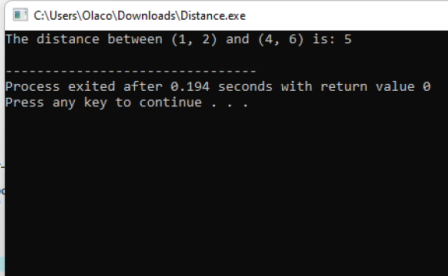
    if (kelvinTemp < 0) {
        std::cout << "Error: Temperature cannot be below 0 Kelvin.\n";
    } else {
        double fahrenheitTemp = kelvinToFahrenheit(kelvinTemp);
        std::cout << kelvinTemp << " Kelvin is " << fahrenheitTemp << " Fahrenheit.\n";
    }

    return 0;
}
```



3. Create a C++ program that has a function that will calculate the distance between two points.

```
Distance.cpp
1 #include <iostream>
2 #include <cmath>
3
4 double calculateDistance(double x1, double y1, double x2, double y2) {
5     double dx = x2 - x1;
6     double dy = y2 - y1;
7     return sqrt(dx*dx + dy * dy);
8 }
9
10 int main() {
11     double point1_x = 1.0;
12     double point1_y = 2.0;
13
14     double point2_x = 4.0;
15     double point2_y = 6.0;
16
17     double distance = calculateDistance(point1_x, point1_y, point2_x, point2_y);
18
19     std::cout << "The distance between (" << point1_x << ", " << point1_y << ") and (" << point2_x << ", " << point2_y << ") is: " << distance << std::endl;
20
21     return 0;
22 }
```



4. Modify the code given in ILO B and add the following functions:

- A function to compute for the area of a triangle
- A function to compute for the perimeter of a triangle
- A function that determines whether the triangle is acute-angled, obtuse-angled or 'others.'

```
#include <iostream>
#include <cmath>

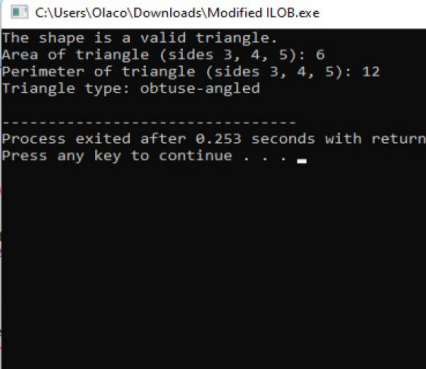
class Triangle {
private:
    double totalAngle, angleA, angleB, angleC;
public:
    Triangle(double A, double B, double C) {
        angleA = A;
        angleB = B;
        angleC = C;
        totalAngle = A+B+C;
    }

    void setAngles(double A, double B, double C) {
        angleA = A;
        angleB = B;
        angleC = C;
        totalAngle = A+B+C;
    }

    const bool validateTriangle() {
        return (totalAngle <= 180);
    }

    double computeArea(double a, double b, double c) {
        double s = (a + b + c) / 2;
        double area = std::sqrt(s*(s-a)*(s-b)*(s-c));
        return area;
    }

    double computePerimeter(double a, double b, double c) {
        double perimeter = a + b + c;
        return perimeter;
    }
};
```



## 8. Conclusion

Provide the following:

- Summary of lessons learned: I was reminded of the fundamentals of C++ by this exercise. Using header files, global declarations, and the main function in particular made it difficult for me to keep my code organized. However, I came to see how crucial these are to making the code understandable and reusable.
- Analysis of the procedure: The additional tasks were particularly difficult and made things worse for me. Even though tasks like converting temperatures and swapping numbers appeared straightforward, I still had to remember and apply the proper syntax and reasoning.
- Analysis of the supplementary activity: The additional tasks were fairly difficult and increased my difficulties. Tasks such as exchanging numbers and transforming temperatures, while they appeared straightforward initially, demanded that I attentively recall and implement the appropriate syntax and reasoning
  - Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement? To be honest, I considered this task quite challenging. I faced difficulties not just with recalling the concepts but also due to my insufficient basic skills and knowledge required to finish it with confidence. I have much to enhance, particularly in grasping the syntax, logic, and structuring my code correctly.

## 9. Assessment Rubric