Hands-On Activity No. 1.2	
Basic C++ Programming	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** September 9, 2024
Section: CPE21S4	**Date Submitted:** September 9, 2024
Name(s): Gob, Mark Jeonel Kenn E.	**Instructor:** Ma'am Maria Rizette H. Sayo

6. Output

Sections	**Answer**
Header File Declaration Section	#include <iostream> using namespace std;
Global Declaration Section	/* A global variable could be used to track something globally. However, in this example, it is simply a placeholder/invalid. */

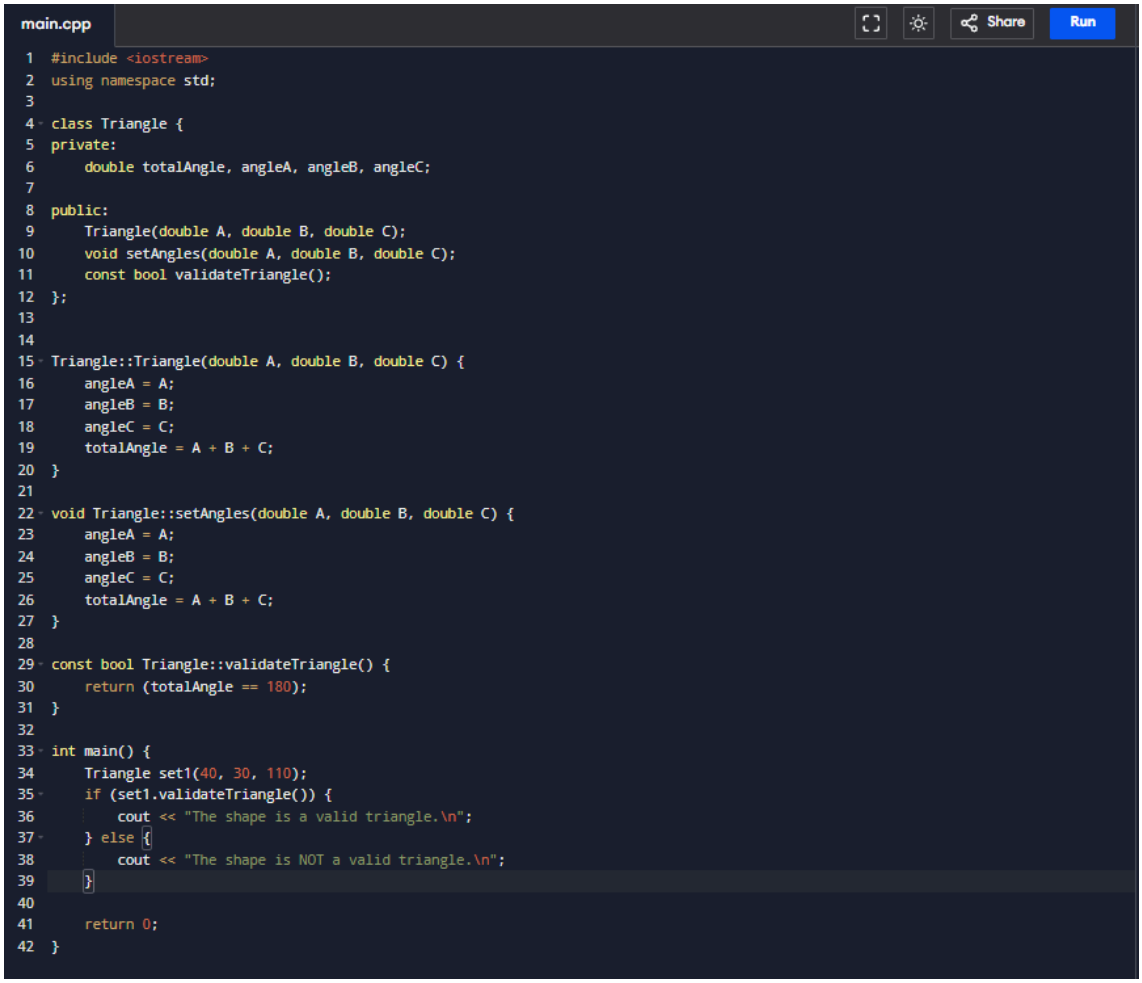
Class Declaration and
Method Definition Section
	class Triangle {
	private:
	  double totalAngle, angleA, angleB, angleC;
	
	public:
	  Triangle(double A, double B, double C);
	  void setAngles(double A, double B, double C);
	  const bool validateTriangle();
	};
	
	Triangle::Triangle(double A, double 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);
	}

Main Function	int main() {
	  Triangle set1(40, 30, 110);
	
	  // Check if the created triangle is valid
	  if (set1.validateTriangle()) {
	  cout << "The shape is a valid triangle.\n";
	  } else {
	  cout << "The shape is NOT a valid triangle.\n";
	  }
	
	  return 0;
	}

Method Definition	/* (Method Definition is already included in the Class Declaration and Method Definition Section.) */

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

Input:



Output:

When the program is executed with the angles 40, 30, and 110 degrees, the output is:

Observations & Comments:

The program's output is based on the 'validateTriangle' method, which checks if the sum of the angles is exactly 180 degrees. In this case, the angles add up to 180, confirming that the shape is a valid triangle. However, we should note that the 'validateTriangle' method only checks if the total angle equals 180 and does not account for potential edge cases like negative or zero angles, which could be added for a more accurate validation. I also note that minimal alterations from the original code from the PDF were made for the sake of fixing coding and syntax errors.

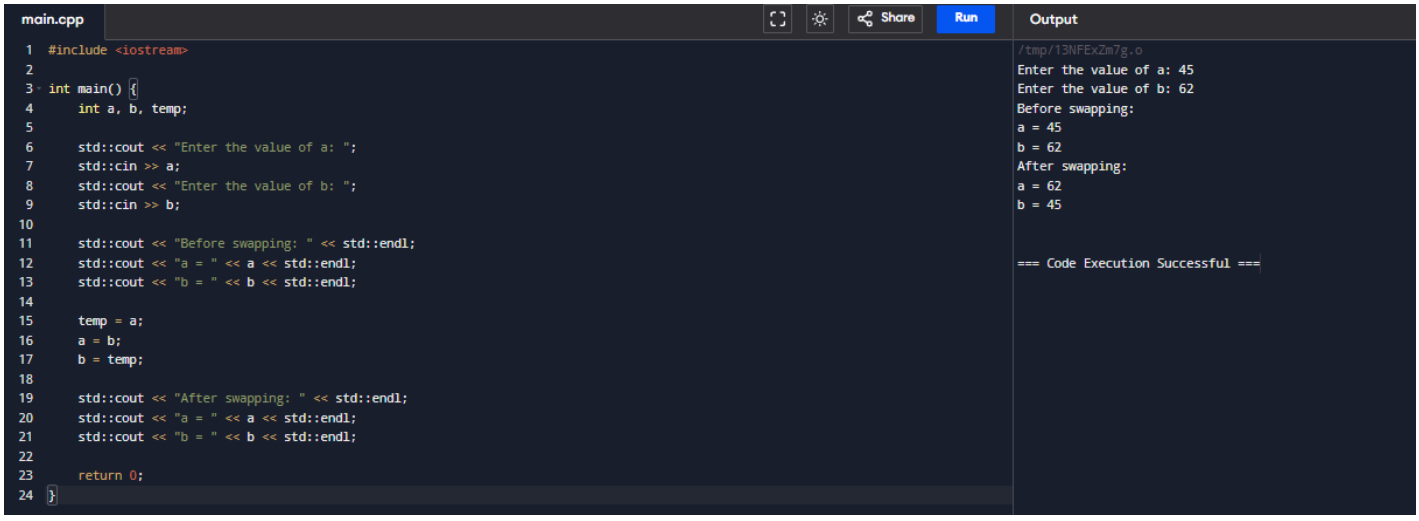
Table 1-2. ILO B output observations and comments.

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.



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

main.cpp

```cpp
#include <iostream>

double kelvinToFahrenheit(double kelvin) {
    return (kelvin - 273.15) * 9.0 / 5.0 + 32.0;
}

int main() {
    double kelvin, fahrenheit;

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

    fahrenheit = kelvinToFahrenheit(kelvin);

    std::cout << kelvin << " Kelvin is equal to " << fahrenheit << " Fahrenheit." << std::endl;

    return 0;
}
```

Output

```
/tmp/6tA063Xx81.o
Enter temperature in Kelvin: 247
247 Kelvin is equal to -15.07 Fahrenheit.


=== Code Execution Successful ===
```

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

main.cpp

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

double calculateDistance(double x1, double y1, double x2, double y2) {
    double distance = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
    return distance;
}

int main() {
    double x1, y1, x2, y2;

    std::cout << "Enter the coordinates of the first point (x1, y1): ";
    std::cin >> x1 >> y1;

    std::cout << "Enter the coordinates of the second point (x2, y2): ";
    std::cin >> x2 >> y2;

    double distance = calculateDistance(x1, y1, x2, y2);

    std::cout << "The distance between the two points is: " << distance << std
        ::endl;

    return 0;
}
```

Output

```
/tmp/Tq9OZkBHIm.o
Enter the coordinates of the first point (x1, y1): 3 5
Enter the coordinates of the second point (x2, y2): 7 6
The distance between the two points is: 4.12311


=== Code Execution Successful ===
```

4. Modify the code given in ILO B and add the following functions:
   a. A function to compute for the area of a triangle
   b. A function to compute for the perimeter of a triangle
   c. A function that determines whether the triangle is acute-angled, obtuse-angled or 'others.'

main.cpp

Share Run

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

using namespace std;

class Triangle {
private:
    double angleA, angleB, angleC;
    double sideA, sideB, sideC;
    double totalAngle;

public:
    Triangle(double A, double B, double C, double a, double b, double c);
    void setAngles(double A, double B, double C);
    void setSides(double a, double b, double c);
    const bool validateTriangle();
    double calculateArea();
    double calculatePerimeter();
    string determineTriangleType();
};

Triangle::Triangle(double A, double B, double C, double a, double b, double c) {
    angleA = A;
    angleB = B;
    angleC = C;
    sideA = a;
    sideB = b;
    sideC = c;
    totalAngle = A + B + C;
}

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

void Triangle::setSides(double a, double b, double c) {
    sideA = a;
    sideB = b;
    sideC = c;
}

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

double Triangle::calculateArea() {
    if (!validateTriangle()) {
        cout << "Invalid triangle. Cannot calculate area.\n";
        return 0;
    }
    double s = (sideA + sideB + sideC) / 2;
    double area = sqrt(s * (s - sideA) * (s - sideB) * (s - sideC));
    return area;
}

double Triangle::calculatePerimeter() {
    return sideA + sideB + sideC;
}

string Triangle::determineTriangleType() {
    if (!validateTriangle()) {
        return "Invalid triangle";
    }

    if (angleA < 90 && angleB < 90 && angleC < 90) {
        return "Acute-angled triangle";
    } else if (angleA > 90 || angleB > 90 || angleC > 90) {
        return "Obtuse-angled triangle";
    } else if (angleA == 90 || angleB == 90 || angleC == 90) {
        return "Right-angled triangle";
    } else {
        return "Others";
    }
}

int main() {
    Triangle set1(40, 30, 110, 3, 4, 5);

    if (set1.validateTriangle()) {
        cout << "The shape is a valid triangle.\n";
    } else {
        cout << "The shape is NOT a valid triangle.\n";
    }

    double area = set1.calculateArea();
    cout << "Area of the triangle: " << area << endl;

    double perimeter = set1.calculatePerimeter();
    cout << "Perimeter of the triangle: " << perimeter << endl;

    string type = set1.determineTriangleType();
    cout << "The triangle is: " << type << endl;

    return 0;
}
```

Output

```
/tmp/0mylcpw4ov.o
The shape is a valid triangle.
Area of the triangle: 6
Perimeter of the triangle: 12
The triangle is: Obtuse-angled triangle


=== Code Execution Successful ===
```

8. Conclusion

Provide the following:

- ***Summary of lessons learned***

  From the activity, the focus was on revisiting core C++ programming concepts, particularly the basics of object-oriented programming (OOP), code structuring, and problem-solving using the C++ language. The key lessons include:

  - Understanding the C++ code structure (header files, global declarations, class and method definitions, main function, etc.).
  - Applying OOP principles by creating classes, methods, and objects.
  - Familiarity with functions, specifically how to create function prototypes and define methods outside the main function.
  - Knowledge of conditional logic and validation techniques to check inputs and constraints (such as validating whether a set of angles forms a valid triangle).
  - Developing practical problem-solving skills by writing functions that handle specific calculations (e.g., perimeter, area, and triangle type).

- ***Analysis of the procedure***
  The procedure followed in the activity effectively reinforces the basics of C++ programming. Key elements of the process included:
  - Creating and manipulating classes and objects, as seen in the task of creating a Triangle class that handles various properties of a triangle (angles and sides).
  - Defining methods that are clearly separated into declarations and definitions, following best practices for code readability.
  - Implementation of conditional checks like validating triangles based on their angles.

  The activities align well with typical C++ problem-solving scenarios, encouraging us to apply control flow, use loops and functions, and focus on methodical code design.

- ***Analysis of the supplementary activity***
  The supplementary activities allowed for the application of learned concepts in increasingly challenging contexts:

  - Swapping numbers: Reinforces basic variable manipulation.
  - Converting temperature: Tests simple arithmetic and understanding of functions.
  - Distance between points: Applies the distance formula, emphasizing how math can be implemented programmatically.
  - Triangle calculations: The most challenging, involving both geometric concepts and OOP to calculate the area, perimeter, and classification of triangles (acute, obtuse, or others).

  These activities help bridge theoretical knowledge with practical applications, challenging us to refine our coding practices while enhancing our problem-solving skills as students.

- ***Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?***
  Overall, the activities helped me demonstrate proficiency in C++ programming and object-oriented principles, successfully applying them to solve real-world problems.

  I believe that my key strengths included proper code structure, class creation, and function usage. On the other hand, I find that my areas for improvement include enhancing error handling, optimizing efficiency, and addressing edge cases more thoroughly.

9. Assessment Rubric

Honor Pledge:

"I affirm that I will not give or receive any unauthorized help on this activity and that all work will be my own."