C++ Tutorial

@ Coding Scape

July 27, 2024

1 Introduction

C++ is a general-purpose programming language that supports object-oriented, imperative, and generic programming. It is widely used for system/software development, game programming, and performance-critical applications.

2 Basic Syntax

2.1 Hello World Program

```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Explanation:

- #include ¡iostream¿ includes the input-output stream library.
- `int main()` is the main function where the execution begins.
- `std::cout` is used to output text to the console.
- `return 0;` signifies that the program ended successfully.

Example 1: Write a program that prints your name and age.

```
#include <iostream>

int main() {
    std::string name = "Sachin Kumar Yadav";
    int age = 19;
    std::cout << "Name: " << name << std::endl;
    std::cout << "Age: " << age << std::endl;
    return 0;
}
```

Example 2: Create a program that calculates and displays the sum of two integers provided by the user.

```
#include <iostream>

int main() {
    int a, b;
    std::cout << "Enter two integers: ";
    std::cin >> a >> b;
    int sum = a + b;
    std::cout << "Sum: " << sum << std::endl;
    return 0;
}
```

Unsolved Question: Write a program that asks the user for their favorite number and prints a message including that number.

3 Variables and Data Types

3.1 Basic Data Types

```
#include <iostream>

int main() {
    int age = 25;
    double height = 5.9;
    char grade = 'A';
    bool isStudent = true;

    std::cout << "Age: " << age << std::endl;
    std::cout << "Height: " << height << std::endl;
    std::cout << "Grade: " << grade << std::endl;
    std::cout << "Is Student: " << isStudent << std::endl;

    return 0;
}
```

Explanation:

- `int`, `double`, `char`, and `bool` are basic data types.
- `std::cout` is used to display values of variables.

Example 1: Create a program that stores and prints the value of a float, a string, and a boolean variable.

```
#include <iostream>

int main() {
    float temperature = 23.5;
    std::string city = "New York";
    bool isSunny = true;

    std::cout << "Temperature: " << temperature << std::endl;
    std::cout << "City: " << city << std::endl;
    std::cout << "Is Sunny: " << isSunny << std::endl;

    return 0;
}
```

Example 2: Write a program that calculates and displays the area of a rectangle given its length and width.

```
#include <iostream>

int main() {
    double length, width;
    std::cout << "Enter length and width of the rectangle: ";
    std::cin >> length >> width;
    double area = length * width;
    std::cout << "Area of the rectangle: " << area << std::endl;
    return 0;
}
```

Unsolved Question: Create a program that initializes and prints three different types of variables: an integer, a character, and a float.

4 Control Structures

4.1 If-Else Statement

```
#include <iostream>

int main() {
    int number;
    std::cout << "Enter a number: ";
    std::cin >> number;

    if (number > 0) {
        std::cout << "The number is positive." << std::endl;
    } else if (number < 0) {
        std::cout << "The number is negative." << std::endl;
    } else {
        std::cout << "The number is zero." << std::endl;
    }

    return 0;
}
```

Explanation:

- The `if-else` structure allows branching based on conditions.
- `std::cin` is used to take input from the user.

Example 1: Write a program that determines if a given number is even or odd.

```
#include <iostream>

int main() {
    int number;
    std::cout << "Enter a number: ";
    std::cin >> number;

    if (number % 2 == 0) {
        std::cout << "The number is even." << std::endl;
    } else {
        std::cout << "The number is odd." << std::endl;
    }

    return 0;
}
```

Example 2: Create a program that checks if a user-entered grade is passing or failing based on a threshold of 60.

```
#include <iostream>

int main() {
    int grade;
    std::cout << "Enter your grade: ";
    std::cin >> grade;

    if (grade >= 60) {
        std::cout << "You passed!" << std::endl;
    } else {
        std::cout << "You failed!" << std::endl;
    }

    return 0;
}
```

Unsolved Question: Write a program that accepts a user's age and determines if they are a minor, an adult, or a senior citizen.

5 Loops

5.1 For Loop

```
#include <iostream>
```

```
int main() {
    for (int i = 1; i <= 5; i++) {
        std::cout << "Number: " << i << std::endl;
    }

    return 0;
}
```

Explanation:

- The `for` loop is used for iteration with a counter.
- `i++` increments the counter variable.

Example 1: Create a program that prints the first 10 positive integers using a for loop.

```
#include <iostream>

int main() {
    for (int i = 1; i <= 10; i++) {
        std::cout << "Number: " << i << std::endl;
    }

    return 0;
}
```

Example 2: Write a program that computes and displays the factorial of a given number using a for loop.

```
#include <iostream>

int main() {
    int number;
    std::cout << "Enter a number: ";
    std::cin >> number;
    int factorial = 1;

    for (int i = 1; i <= number; i++) {
        factorial *= i;
    }

    std::cout << "Factorial: " << factorial << std::endl;
    return 0;
}
```

Unsolved Question: Write a program that prints the multiplication table of a given number using a for loop.

6 Functions

6.1 Defining and Calling Functions

```
#include <iostream>

void greet() {
    std::cout << "Hello from the function!" << std::endl;
}

int add(int a, int b) {
    return a + b;
}

int main() {
    greet();
    int result = add(5, 3);
    std::cout << "Sum: " << result << std::endl;
    return 0;
}
```

Explanation:

- Functions allow code reuse and modular programming.
- `void` indicates that the function does not return a value.
- `int add(int a, int b)` is a function that returns the sum of two integers.

Example 1: Write a function that calculates and returns the square of a number. Call this function from `main`.

```
#include <iostream>

int square(int n) {
    return n * n;
}

int main() {
    int number = 4;
    std::cout << "Square of " << number << " is " << square(number) << std::endl;
    return 0;
}
```

Example 2: Create a function that takes a string as an argument and prints it in uppercase. Use this function in your program.

```
#include <iostream>
#include <cctype>
#include <string>
```

```
void printUppercase(const std::string& s) {
    for (char c : s) {
        std::cout << char(toupper(c));
    }
    std::cout << std::endl;
}

int main() {
    std::string text = "hello";
    printUppercase(text);
    return 0;
}
```

Unsolved Question: Write a function that takes two numbers as input and returns the larger of the two. Test this function in your main program.

7 Classes and Objects

7.1 Basic Class Structure

```
#include <iostream>

class Rectangle {
public:
    int length;
    int width;

    int area() {
        return length * width;
    }
};

int main() {
    Rectangle rect;
    rect.length = 10;
    rect.width = 5;
    std::cout << "Area: " << rect.area() << std::endl;
    return 0;
}
```

Explanation:

- Classes encapsulate data and functions.
- The `area()` method calculates the area of the rectangle.

Example 1: Create a class called `Circle` with a method to calculate the circumference. Instantiate the class and display the circumference.

```
#include <iostream>
#define PI 3.14159

class Circle {
public:
    double radius;

    double circumference() {
        return 2 * PI * radius;
    }
};

int main() {
    Circle circle;
    circle.radius = 7.5;
    std::cout << "Circumference: " << circle.circumference() << std::endl;
    return 0;
}
```

Example 2: Write a class `Student` with attributes for name and grade, and a method to display these attributes.

```
#include <iostream>
#include <string>

class Student {
public:
    std::string name;
    char grade;

    void display() {
        std::cout << "Name: " << name << std::endl;
        std::cout << "Grade: " << grade << std::endl;
    }
};

int main() {
    Student student;
    student.name = "Alice";
    student.grade = 'A';
    student.display();
    return 0;
}
```

Unsolved Question: Define a class `BankAccount` with methods to deposit and withdraw money. Implement this class and test it with various transactions.

8 Inheritance

8.1 Basic Inheritance

```
#include <iostream>

class Shape {
public:
    virtual void draw() {
        std::cout << "Drawing a shape" << std::endl;
    }
};

class Circle : public Shape {
public:
    void draw() override {
        std::cout << "Drawing a circle" << std::endl;
    }
};

int main() {
    Shape* shape = new Circle();
    shape->draw();
    delete shape;
    return 0;
}
```

Explanation:

- Inheritance allows one class to inherit attributes and methods from another.
- The `Circle` class overrides the `draw()` method from the `Shape` class.

Example 1: Create a base class `Vehicle` with derived classes `Car` and `Bike`. Each derived class should override a method to display its specific type.

```
#include <iostream>

class Vehicle {
public:
    virtual void display() {
        std::cout << "This is a vehicle" << std::endl;
    }
};

class Car : public Vehicle {
public:
    void display() override {
        std::cout << "This is a car" << std::endl;
    }
};

class Bike : public Vehicle {
public:
    void display() override {
        std::cout << "This is a bike" << std::endl;
    }
};

int main() {
    Vehicle* vehicle1 = new Car();
    Vehicle* vehicle2 = new Bike();
    vehicle1->display();
    vehicle2->display();
    delete vehicle1;
    delete vehicle2;
    return 0;
}
```

Example 2: Write a base class `Animal` and derived classes `Dog` and `Cat`. Implement a method `makeSound()` in each derived class.

```
#include <iostream>

class Animal {
public:
    virtual void makeSound() {
        std::cout << "Animal sound" << std::endl;
    }
};

class Dog : public Animal {
public:
    void makeSound() override {
        std::cout << "Woof!" << std::endl;
    }
};

class Cat : public Animal {
public:
    void makeSound() override {
        std::cout << "Meow!" << std::endl;
    }
};

int main() {
    Animal* animal1 = new Dog();
    Animal* animal2 = new Cat();
    animal1->makeSound();
    animal2->makeSound();
    delete animal1;
    delete animal2;
    return 0;
}
```

Unsolved Question: Implement a base class `Employee` with derived classes `Manager` and `Intern`. Each class should have a method to display its role.

9 Polymorphism

9.1 Function Overloading

```
#include <iostream>

void print(int i) {
    std::cout << "Integer: " << i << std::endl;
}

void print(double d) {
    std::cout << "Double: " << d << std::endl;
}

void print(const std::string& s) {
    std::cout << "String: " << s << std::endl;
}

int main() {
    print(10);
    print(10.5);
    print("Hello");
    return 0;
}
```

Explanation:

- Function overloading allows multiple functions with the same name but different parameters.

Example 1: Define overloaded functions `multiply()` that handle integers, doubles, and floats.

```
#include <iostream>

int multiply(int a, int b) {
    return a * b;
}

double multiply(double a, double b) {
    return a * b;
}

float multiply(float a, float b) {
    return a * b;
}

int main() {
    std::cout << "Multiply integers: " << multiply(3, 4) << std::endl;
    std::cout << "Multiply doubles: " << multiply(3.5, 4.5) << std::endl;
    std::cout << "Multiply floats: " << multiply(3.5f, 4.5f) << std::endl;
    return 0;
}
```

Example 2: Write a function `display()` that takes different types of arguments and prints a message accordingly.

```
#include <iostream>

void display(int i) {
    std::cout << "Displaying integer: " << i << std::endl;
}

void display(double d) {
    std::cout << "Displaying double: " << d << std::endl;
}

void display(const std::string& s) {
    std::cout << "Displaying string: " << s << std::endl;
}

int main() {
    display(10);
    display(10.5);
    display("Hello");
    return 0;
}
```

Unsolved Question: Implement overloaded functions for `add()` that work with different combinations of integer and double types.

10 Templates

10.1 Function Templates

```
#include <iostream>

template <typename T>
T add(T a, T b) {
    return a + b;
}

int main() {
    std::cout << add(5, 3) << std::endl;
    std::cout << add(5.5, 3.5) << std::endl;
    return 0;
}
```

Explanation:

- Templates allow writing generic functions or classes.
- `template <typename T>` defines a template that can handle different data types.

Example 1: Write a template function `max()` that returns the maximum of two values of any type.

```
#include <iostream>

template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
    std::cout << "Max of 3 and 7: " << max(3, 7) << std::endl;
    std::cout << "Max of 3.5 and 2.1: " << max(3.5, 2.1) << std::endl;
    return 0;
}
```

Example 2: Create a template class `Box` that can store any data type and provide methods to set and get the value.

```
#include <iostream>

template <typename T>
class Box {
private:
    T value;
public:
    void setValue(T v) {
        value = v;
    }

    T getValue() {
        return value;
    }
};

int main() {
    Box<int> intBox;
    intBox.setValue(123);
    std::cout << "Integer Value: " << intBox.getValue() << std::endl;

    Box<std::string> strBox;
    strBox.setValue("Hello, World!");
    std::cout << "String Value: " << strBox.getValue() << std::endl;

    return 0;
}
```

Unsolved Question: Implement a template function `swap()` that exchanges the values of two variables of any type.

11 Conclusion

This tutorial has covered the basics of C++ programming, including syntax, variables, control structures, loops, functions, classes, inheritance, polymorphism, and templates. Practice with the examples and unsolved questions to enhance your understanding and proficiency in C++.