**ODD SEMESTER 2023**

**OOPS Lab**

**Submitted for**

# Masters of Computer Applications



Submitted to:

Submitted by:

**Sukaina Inam Naqvi**

Application No: JH23-PGP-02037

MCA 1st Semester,1st Year

**Dr. Shahab Saquib Sohail**

Deptt. of Computer Science & Engg.

Jamia Hamdard, New Delhi, India

## Department Of Computer Science and Engineering

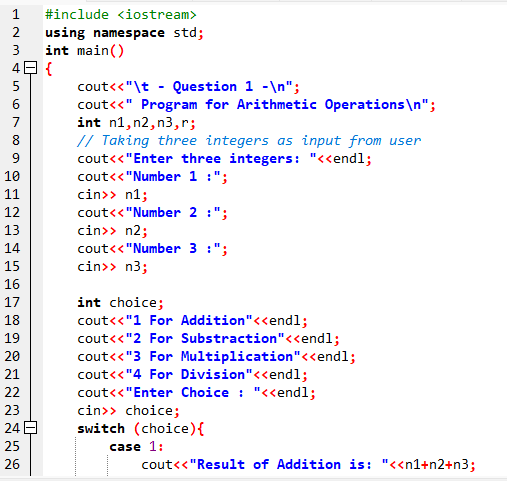
**School of Engineering Sciences and Technology**

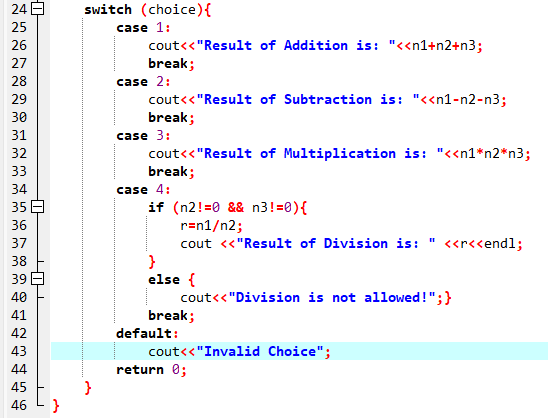
**Jamia Hamdard, New Delhi -110062**

|  |  |  |
| --- | --- | --- |
| Sr.No. | Question | Pg.no. |
| 1 | Wap to perform Arithmetic Operation | 3 |
| 2 | Without using third variable swap the value of two variables | 5 |
| 3 | Wap to print fibonacci series | 6 |
| 4 | Check number is prime or not | 7 |
| 5 | Print triangular pattern | 8 |
| 6 | Wap to print factorial | 9 |
| 7 | Define a class in C++ and provide an example. | 9 |
| 8 | Differentiate between a class and an object in C++. | 10 |
| 9 | Differentiate bw bottom up and top-down approach.  It is stated that oop is a bottom-up approach justify the statement. | 11 |
| 10 | Discuss the concept of inheritance and its types in C++. | 12 |
| 11 | What do you mean by multiple inheritance | 14 |
| 12 | Why do we use inheritance. Wap a program in c++ that illustrate the use of inheritance and its advantage | 15 |
| 13 | Perform the same task without using any oops concepts (inheritance) | 18 |
| 14 | Show how using inheritance is having advantage over procedural approach | 20 |
| 15 | What is a friend function, and how does it relate to OOP in C++? | 20 |
| 16 | What are virtual functions, and why are they essential in polymorphism with example | 21 |
| 17 | Wap to use functions overloading and overriding | 22 |
| 18 | Wap to perform implicit and and explicit type casting | 24 |
| 19 | Explain the concept of encapsulation and how it is implemented in C++. | 25 |
| 20 | How does polymorphism enhance the flexibility of C++ programs? | 27 |

**Question1. Write a program in c++ to perform arithmetic operations with three integer variables.**

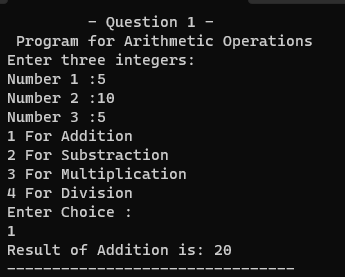
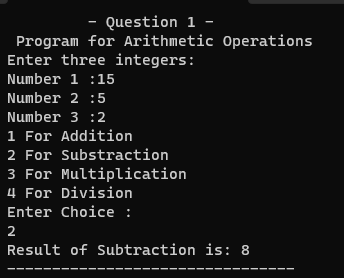
**Code:**

****

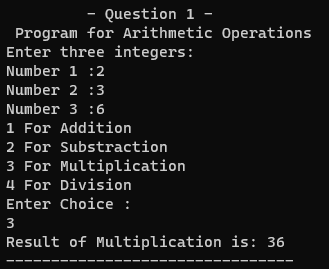
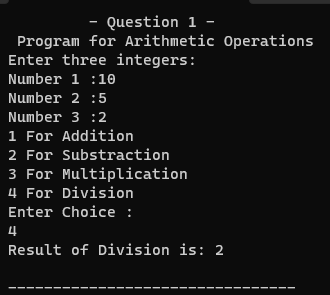
****

**Output:**

**Addition: Subtraction:**

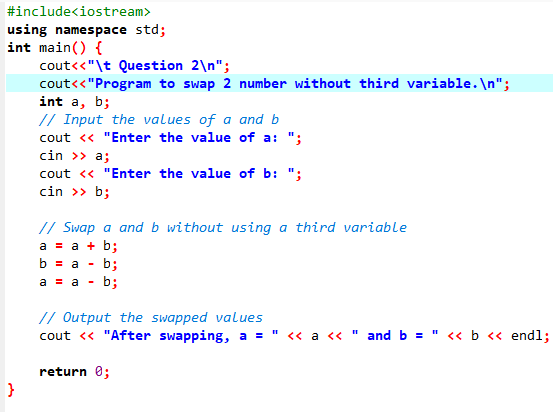
** **

**Multiplication: Division:**

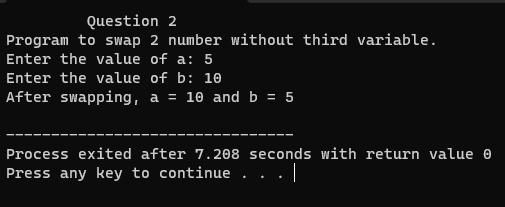
** **

**Question 2. Without using third variable, swap the value of two variables.**

**Code:**

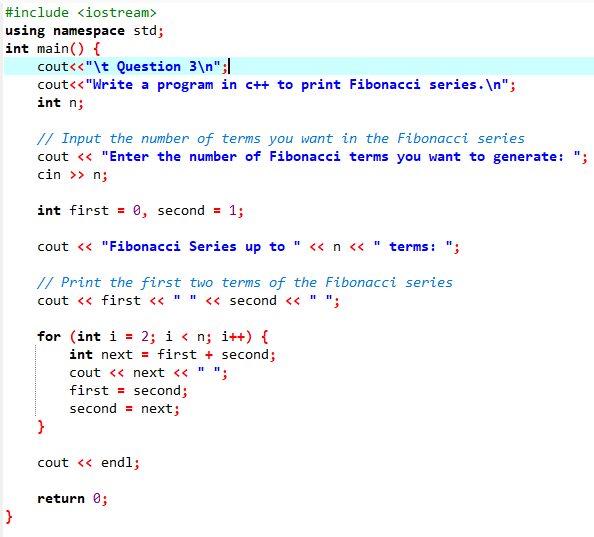
****

**Output:**

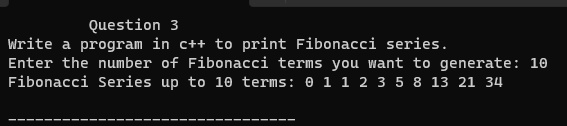
****

**Question 3. Write a program in c++ to print Fibonacci series.**

**Code:**

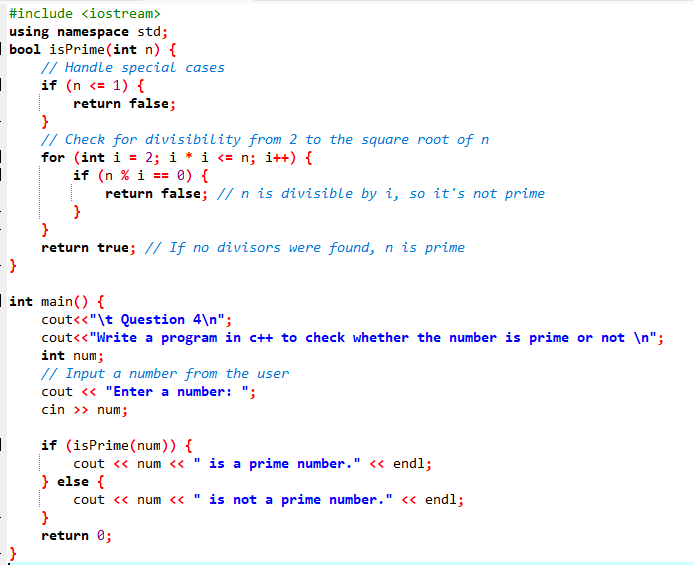
****

**Output:**

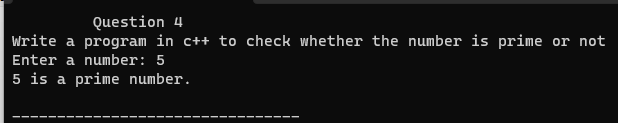
****

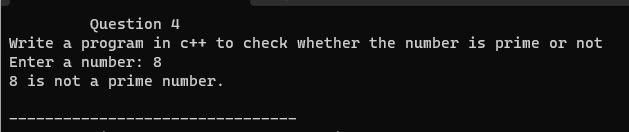
**Question 4. You are given a number, write a program in c++ to check whether the number is prime or not?**

**Code:**

****

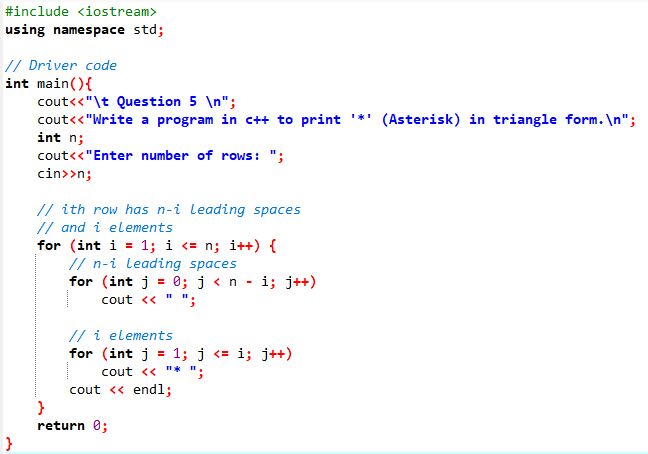
**Output:**

****

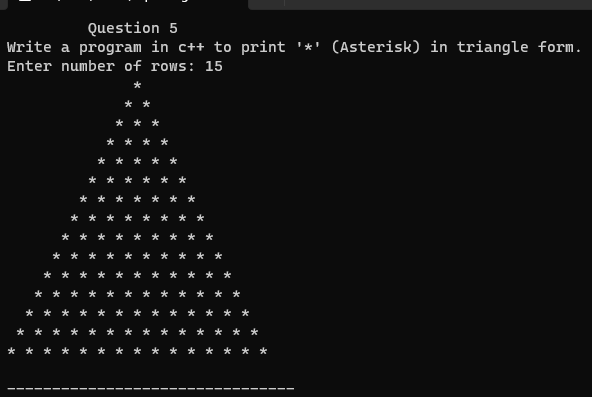
****

**Question 5. Write a program in c++ to print ‘\*’ (Asterisk) in triangle form.**

**Code:**

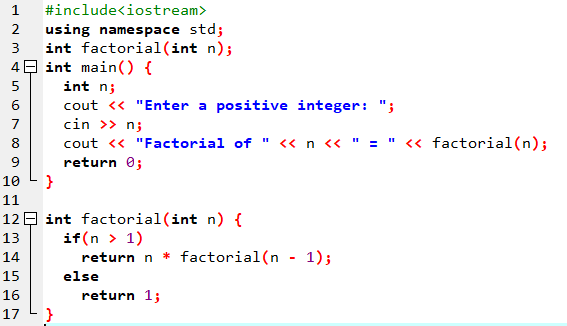
****

**Output:**

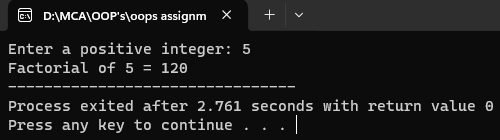
****

**Question 6.** **Wap to print factorial.**

**Code:**

****

**Output:**

****

**Question 7. Define a class in C++ and provide an example.**

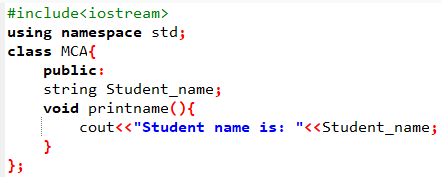
**Answer:**

**Class in C++** is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

* A Class is a user-defined data type that has data members and member functions.
* Data members are the data variables and member functions are the functions used to manipulate these variables together, these data members and member functions define the properties and behavior of the objects in a Class.

For Example: Consider the Class of **MCA**. There may be many cars with different names and student id but all of them will share some common properties like all of them will have *same subjects, same degree, same teachers, fees* etc.

In the above example of class *MCA*, the data member will be *Student Name, Student id* etc.

****

**Question 8. Differentiate between a class and an object in C++.**

**Answer:**

|  |  |
| --- | --- |
| Class | Object |
| Class is used as a template for declaring and  creating the objects. | An object is an instance of a class. |
| When a class is created, no memory is allocated. | Objects are allocated memory space whenever they are created. |
| The class has to be declared first and only once. | An object is created many times as per requirement. |
| A class can not be manipulated as they are not available in the memory. | Objects can be manipulated. |
| A class is a logical entity. | An object is a physical entity. |
| It is declared with the class keyword | It is created with a class name in C++ and  with the **new** keywords in Java. |
| Class does not contain any values which  can be associated with the field. | Each object has its own values, which are associated with it. |
| A class is used to bind data as well as methods together as a single unit. | Objects are like a variable of the class. |
| Syntax: Declaring Class in C++ is as follows:  class <classname> {  }; | **Syntax:**  class Student {   public:        void put(){          cout<<“Function Called”<<endl;}  }; // The class is declared here  int main () {           Student s1; // Object created           s1.put ();  } |
| Example: Car | Example: wheels, brakes |

**Question 9: Difference between Top-Down Approach and Bottom-Up Approach?**

**Answer:** In object-oriented programming (OOP), the concepts of "top-down" and "bottom-up" can be applied to the design and implementation of software systems. A "top-down" approach in OOP involves starting with a high-level view of the system, often defining the overall architecture, class hierarchies, and major functionalities before delving into the details of individual classes and methods. Design patterns, inheritance, and abstraction play significant roles in the top-down approach, allowing developers to create a well-structured and organized system from the outset.

On the other hand, a "bottom-up" approach in OOP involves building the system from specific components or classes, gradually combining them to form higher-level structures. Developers using the bottom-up approach might begin by implementing individual classes with specific functionalities and then composing them to create more complex objects or systems. This approach emphasizes code reusability and modular design, allowing for greater flexibility and adaptability.

The main difference between the top-down and bottom-up approaches is the

process's starting point and focus. The top-down approach prioritizes high-level

planning and decision-making, while the bottom-up approach prioritizes the

execution of individual tasks and the development of detailed knowledge. Both

approaches have advantages and disadvantages, and the best approach will

depend on the specific context, including the nature of the problem, the

resources available, the timeline, and the desired outcome.

**It is stated that OOP is a bottom-up approach, justify the statement.**

**Answer:** It’s because you first define a class and its functions. Then you initialize an object of that class and calls functions as per the need. Now, though process looks like to-bottom but the execution takes place in bottom-up approach. While executing, execution flow finds object initialization first and then it looks up for declared class and then functions.

**Question 10. Discuss the concept of inheritance and its types in C++.**

**Answer:** Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class.

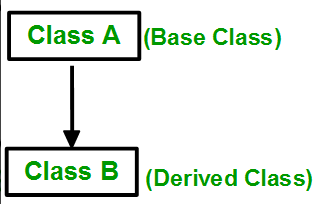
* **Sub Class:** The class that inherits properties from another class is called Subclass or Derived Class.
* **Super Class:**The class whose properties are inherited by a subclass is called Base Class or Superclass.

**Types Of Inheritance: -**

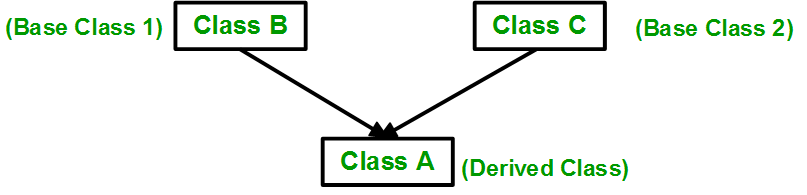
1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

**Types of Inheritance in C++**

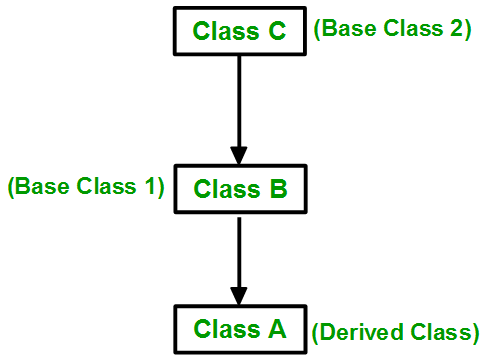
**1.Single Inheritance**: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.



**2. Multiple Inheritance:**Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one subclass is inherited from more than one base class.

****

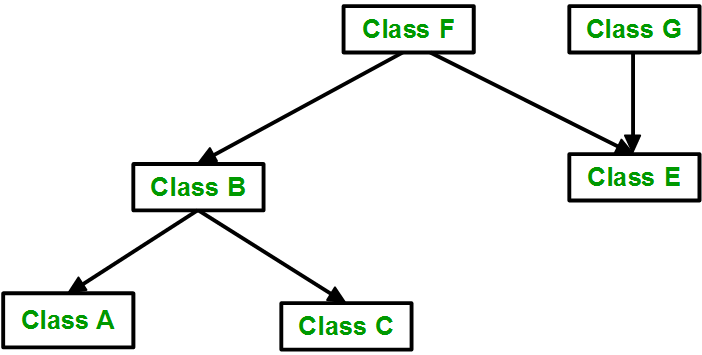
**3. Multilevel Inheritance**: In this type of inheritance, a derived class is created from another derived class.



**4. Hierarchical Inheritance**: In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



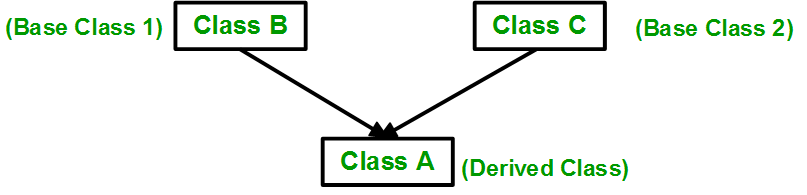
**5. Hybrid (Virtual) Inheritance**: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.   
Below image shows the combination of hierarchical and multiple inheritances:



**Question 11: What do you mean by multiple inheritance?**

**Answer:** Multiple inheritance is a feature of some object-oriented computer programming languages in which an object or class can inherit features from more than one parent object or parent class. It is distinct from single inheritance, where an object or class may only inherit from one particular object or class.

A diagram that demonstrates multiple inheritance is given below –

****

**Question 12. Why do we use inheritance?**

**Answer:** Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors (realizing an interface), to reuse code and to independently extend original software via public classes and interfaces.

We use inheritance in C++ for the reusability of code from the existing class. C++ strongly supports the concept of reusability. Reusability is yet another essential feature of OOP (Object Oriented Programming).

It is always good to reuse something that already exists rather than trying to create the one that is already present, as it saves time and increases reliability.

We use inheritance in C++ when both the classes in the program have the same logical domain and when we want the class to use the properties of its superclass along with its properties.

For example, there is a base class or parent class named “Animal,” and there is a child class named “Dog,” so, here dog is an animal, so in “Dog class,” all the common properties of the “Animal” class should be there, along with its property of dog animal.

**Write a program in C++ that illustrate the use of inheritance and its advantage.**

**Code:**

#include <iostream>

// Base class 1

class Engine {

public:

Engine() {

std::cout << "Engine constructor" << std::endl;

}

void start() {

std::cout << "Engine started" << std::endl;

}

void stop() {

std::cout << "Engine stopped" << std::endl;

}};

// Base class 2

class Wheels {

public:

Wheels() {

std::cout << "Wheels constructor" << std::endl;

}

void rotate() {

std::cout << "Wheels rotating" << std::endl;

}

void stopRotation() {

std::cout << "Wheels stopped rotating" << std::endl;

}

};

// Derived class inheriting from both Engine and Wheels

class Car : public Engine, public Wheels {

public:

Car() {

std::cout << "Car constructor" << std::endl;

}

void drive() {

start(); // From Engine

rotate(); // From Wheels

std::cout << "Car is moving" << std::endl;

}

void park() {

stop(); // From Engine

stopRotation(); // From Wheels

std::cout << "Car is parked" << std::endl;

}

};

int main() {

// Creating an object of the derived class

Car myCar;

// Accessing members from base classes

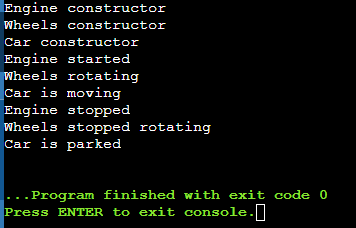
myCar.drive(); // Uses members from both Engine and Wheels

myCar.park(); // Uses members from both Engine and Wheels

return 0;

}

**Output:-**

****

**Advantages of Inheritance in C++**

Inheritance promotes reusability. When a class inherits or derives another class, it can access all the functionality of inherited class.

Reusability enhanced reliability. The base class code will be already tested and debugged.

As the existing code is reused, it leads to less development and maintenance costs.

Inheritance makes the sub classes follow a standard interface.

Inheritance helps to reduce code redundancy and supports code extensibility.

Inheritance facilitates creation of class libraries.

**Question 13. Perform same task without using any OOP concept(inheritance).**

**Code:**

#include <iostream>

// Engine functions

void startEngine() {

std::cout << "Engine started" << std::endl;

}

void stopEngine() {

std::cout << "Engine stopped" << std::endl;

}

// Wheels functions

void rotateWheels() {

std::cout << "Wheels rotating" << std::endl;

}

void stopWheelRotation() {

std::cout << "Wheels stopped rotating" << std::endl;

}

// Car-related functions

void driveCar() {

startEngine();

rotateWheels();

std::cout << "Car is moving" << std::endl;

}

void parkCar() {

stopEngine();

stopWheelRotation();

std::cout << "Car is parked" << std::endl;

}

int main() {

// Perform the same task without using inheritance

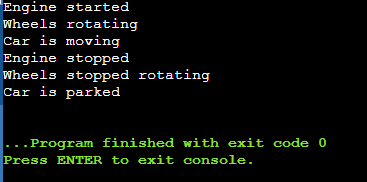
driveCar();

parkCar();

return 0;

}

**Output:**

****

**Question 14. Show how using inheritance is having advantage over procedural approach.**

**Answer:** The length of the programs developed using OOP language is much larger than the procedural approach. Since the program becomes larger in size, it requires more time to be executed that leads to slower execution of the program.

We cannot apply OOP everywhere as it is not a universal language. It is applied only when it is required. It is not suitable for all types of problems.

Programmers need to have brilliant designing skill and programming skill along with proper planning because using OOP is little bit tricky.

OOPs take time to get used to it. The thought process involved in object-oriented programming may not be natural for some people.

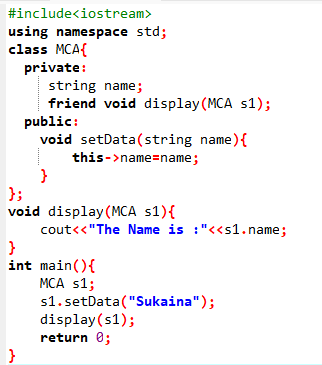
Everything is treated as object in OOP so before applying it we need to have excellent thinking in terms of objects.

**Question 15. What is a friend function, and how does it relate to OOP in C++?**

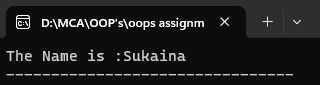
The friend function is a function that can access the private member of class, it’s written outside of class and declared in class. It will take an object of the class as a parameter.

Object Oriented Programming has a future that is an abstraction where we can declare data members of the class as private, public, and protected for security purposes.

Code: -



Output:

****

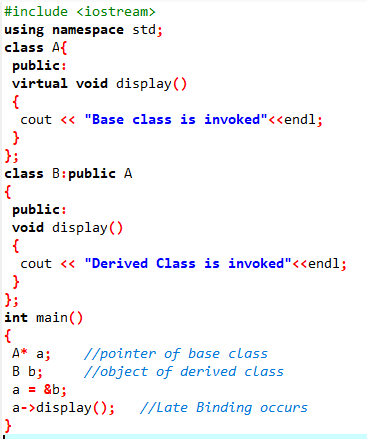
**Question 16. What are virtual functions, and why are they essential in polymorphism with example.**

A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword. It is used to tell the compiler to perform dynamic linkage or late binding on the function.

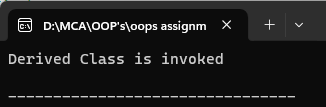
There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when the base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.

A 'virtual' is a keyword preceding the normal declaration of a function. When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

**Code:**

****

**Output:-**

****

**Question 17. Wap to use functions overloading and overriding**

**Function Overloading**

The concept by which we can define different function in a class with the same name but with different parameters is known as **function overloading**. Function overloading takes place during compile time. Therefore, it is also called **compile time polymorphism**. Function overloading happens without inheritance.

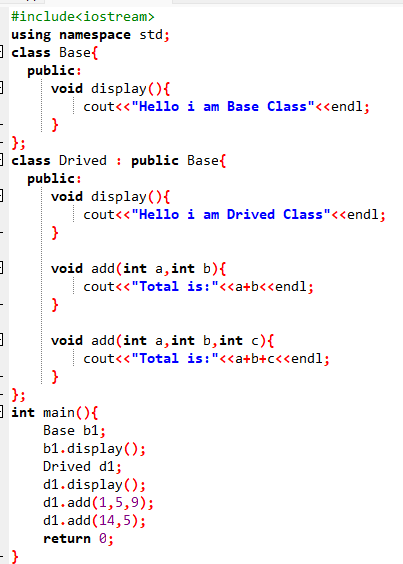
For example, consider two functions **add (float a, float b)** and **add (int a, int b)**. Here, the two functions have the same name but different types of parameters.

**Function Overriding**

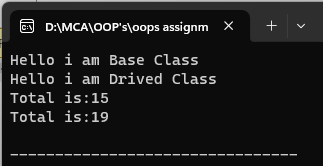
**Function overriding** is the concept that allows two classes to have a function with the same name. Function overriding is accomplished by using inheritance and virtual functions.

As we know every derived class inherits all the functions of its base class, in this case all the member functions of a derived class override the member functions of the base class, hence this is known as the **function overriding**. **Function overriding** is achieved during runtime, hence it is also known as runtime polymorphism.

Code:-



Output:-



**Question 18. Wap to perform implicit and explicit type casting**

**Answer:** C++ allows us to convert data of one type to that of another. This is known as type conversion.

There are two types of type conversion in C++.

1.Implicit Conversion

2.Explicit Conversion (also known as Type Casting)

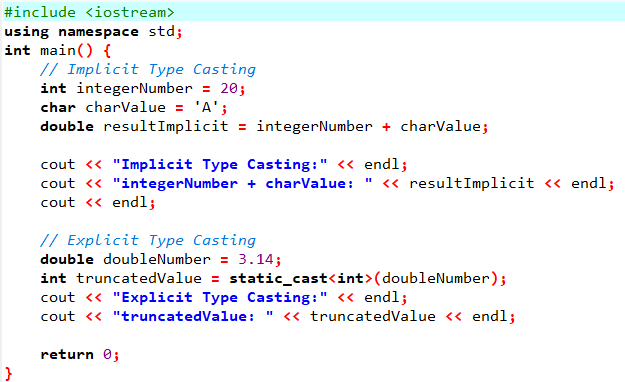
**Implicit Type Conversion**

The type conversion that is done automatically done by the compiler is known as implicit type conversion. This type of conversion is also known as automatic conversion.

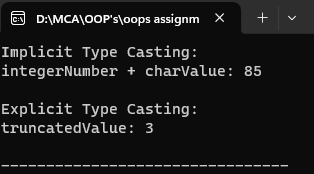
**Explicit type conversion**

An explicit conversion allows the programmer to manually changes or typecasts the data type from one variable to another type. Hence, it is also known as typecasting. Generally, we force the explicit type conversion to convert data from one type to another because it does not follow the implicit conversion rule.

Code:-



Ouput:-



1.Implicit Type Casting:

The integerNumber (int) and charValue (char) are implicitly converted to a double when added together. The result is a double value.

2.Explicit Type Casting:

The doubleNumber (double) is explicitly cast to an integer using static\_cast<int>(doubleNumber). This type of conversion truncates the decimal part, and the result is assigned to the truncatedValue.

**Question 19. Explain the concept of encapsulation and how it is implemented in C++.**

Answer: Encapsulation is one of the fundamental principles of object-oriented programming (OOP) that promotes the bundling of data and methods that operate on that data within a single unit, known as a class. The idea behind encapsulation is to hide the internal details of an object and restrict access to its internal state. This helps in achieving data integrity and protecting the object's internal implementation.

In C++, encapsulation is implemented using classes. Here are the key components and techniques involved in encapsulation in C++:

**1.Class Definition:**

* + A class is a user-defined data type that encapsulates data members (attributes) and member functions (methods).
  + Data members represent the internal state of the object.
  + Member functions define the operations that can be performed on the object.

**2.Access Specifiers:**

C++ provides three access specifiers: private, protected, and public.

Members declared as private are accessible only within the class, not outside it.

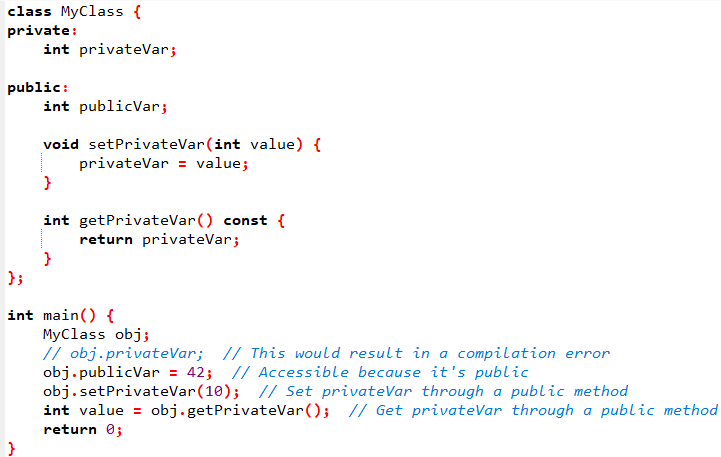
Members declared as public are accessible from outside the class.

**3.Data Hiding:**

Members declared as private are hidden from the outside world, ensuring that only the class's methods can access or modify them.

Public methods (getters and setters) are used to control the access to private members.

Code:



**4.Encapsulation Benefits:**

* + Improved code organization: Members related to each other are grouped within a class.
  + Information hiding: Internal details are hidden, reducing complexity and preventing unauthorized access.
  + Modularity: Changes to the internal implementation do not affect the external code as long as the public interface remains unchanged.

By applying encapsulation in C++, you create more robust and maintainable code, and it becomes easier to understand and extend your programs over time.

**Question 20. How does polymorphism enhance the flexibility of C++ programs?**

Answer: Polymorphism is a fundamental concept in object-oriented programming (OOP) and is a key feature in C++. It enhances the flexibility of C++ programs in several ways:

**1.Code Reusability:** Polymorphism allows you to write code that can work with objects of multiple classes through a common interface. This promotes code reuse because you can write functions or methods that operate on a base class type, and these can be used with derived classes without modification.

**2.Flexibility in Function Arguments:** Functions can accept objects of different types, as long as they share a common base class. This allows you to pass objects of derived classes to functions that expect objects of the base class type, providing flexibility in function arguments.

**3.Run-time Binding (Dynamic Binding):** Polymorphism allows for dynamic dispatch, meaning that the correct function or method is called at runtime based on the actual type of the object rather than the declared type. This enables more flexibility during program execution and supports extensibility by allowing new derived classes to be added without modifying existing code.

**4.Virtual Functions and Abstract Classes:** Polymorphism is often implemented using virtual functions and abstract classes in C++. Virtual functions allow a function in a base class to be overridden by a function in a derived class. Abstract classes, which may contain pure virtual functions, provide a common interface for a group of related classes.

In summary, polymorphism in C++ enhances program flexibility by promoting code reuse, allowing flexibility in function arguments, supporting dynamic binding, and facilitating the creation of extensible and modular code through the use of virtual functions and abstract classes.