**Problem No.**: 01

**Problem Name:** Encapsulation in C++.

**Code:**

#include <iostream>

using namespace std;

class temp{

int a;

int b;

public:

int solve(int input){

a=input;

b=a/2;

return b;

}

};

int main() {

int n;

cin>>n;

temp half;

int ans=half.solve(n);

cout<<ans<<endl;

}

**Output:**

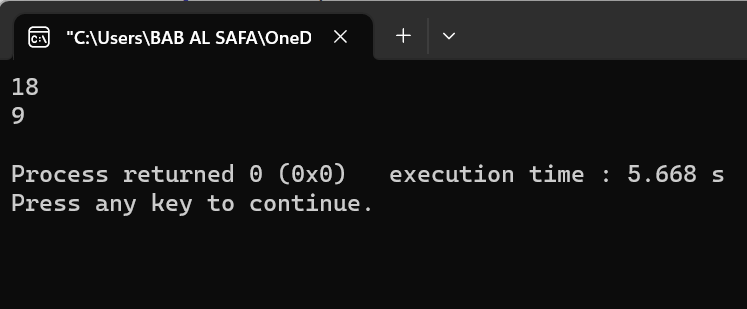


Fig 1.1: Output on console.

**Explanation:**

In this code, defined a class named temp with two private integer data members a and b. The class also has a public member function named solve which takes an integer input and returns half of it. By making the data members private, have encapsulated them within the class and prevented direct access from outside the class. [The public member function solve provides an interface for accessing the data members in a controlled way](https://www.udacity.com/blog/2021/09/cpp-encapsulation-an-overview.html" \t "https://www.bing.com/_blank)

**Problem No.**: 02

**Problem Name**: Encapsulation in C++.

**Code:**

#include <iostream>

#include <string>

using namespace std;

class Person {

private:

string name;

int age;

public:

Person(string name, int age) {

this->name = name;

this->age = age;

}

void setName(string name) {

this->name = name;

}

string getName() {

return name;

}

void setAge(int age) {

this->age = age;

}

int getAge() {

return age;

}

};

int main() {

Person person("John Doe", 30);

cout << "Name: " << person.getName() << endl;

cout << "Age: " << person.getAge() << endl;

person.setName("Jane Doe");

person.setAge(32);

cout << "Name: " << person.getName() << endl;

cout << "Age: " << person.getAge() << endl;

return 0;

}

**Output:**

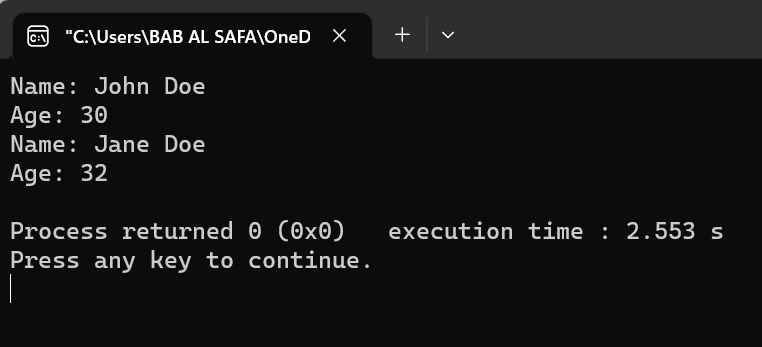


Fig 2.1: Output on console.

**Explanation:**

In this code, defined a class named Person with two private data members name and age. The class also has public member functions named setName, getName, setAge, and getAge which provide an interface for accessing and modifying the private data members in a controlled way. By making the data members private, have encapsulated them within the class and prevented direct access from outside the class. [The public member functions provide an interface for accessing the data members in a controlled way](https://www.udacity.com/blog/2021/09/cpp-encapsulation-an-overview.html" \t "https://www.bing.com/_blank).

In the main function, an object of the Person class named person is created with the name “John Doe” and age 30. The getName and getAge functions are called to print the name and age of the person to the console. Then, the setName and setAge functions are called to modify the name and age of the person to “Jane Doe” and 32, respectively. Finally, the getName and getAge functions are called again to print the updated name and age of the person to the console.

**Problem No.**: 03

**Problem Name**: Encapsulation In C++.

**Code:**

// C++ program to demonstrate

// Encapsulation

#include <iostream>

using namespace std;

class Encapsulation {

private:

// Data hidden from outside world

int x;

public:

// Function to set value of

// variable x

void set(int a) { x = a; }

// Function to return value of

// variable x

int get() { return x; }

};

// Driver code

int main()

{

Encapsulation obj;

obj.set(5);

cout << obj.get();

return 0;

}

**Output:**

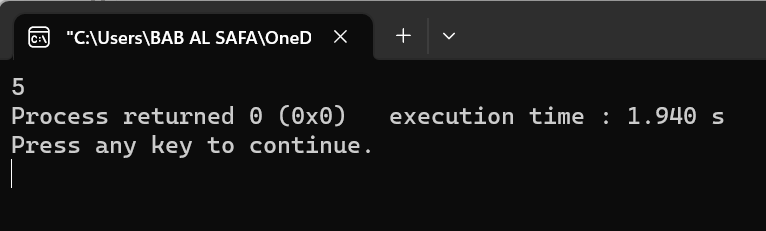


Fig 3.1: Output on console.

**Explanation:**

In this code, defined a class named Encapsulation with a private data member x. The class also has public member functions named set and get which provide an interface for accessing and modifying the private data member in a controlled way. By making the data member private, have encapsulated it within the class and prevented direct access from outside the class. The public member functions provide an interface for accessing the data member in a controlled way.

In the main function, an object of the Encapsulation class named obj is created. The set function is called to set the value of the private data member to 5. Then, the get function is called to return the value of the private data member, which is printed to the console

**Problem No.**: 04

**Problem Name**: Encapsulation In C++.

**Code:**

#include <iostream>

using namespace std;

// declaring class

class Circle {

// access modifier

private:

// Data Member

float area;

float radius;

public:

void getRadius()

{

cout << "Enter radius\n";

cin >> radius;

}

void findArea()

{

area = 3.14 \* radius \* radius;

cout << "Area of circle=" << area;

}

};

int main()

{

// creating instance(object) of class

Circle cir;

cir.getRadius(); // calling function

cir.findArea(); // calling function

}

**Output:**

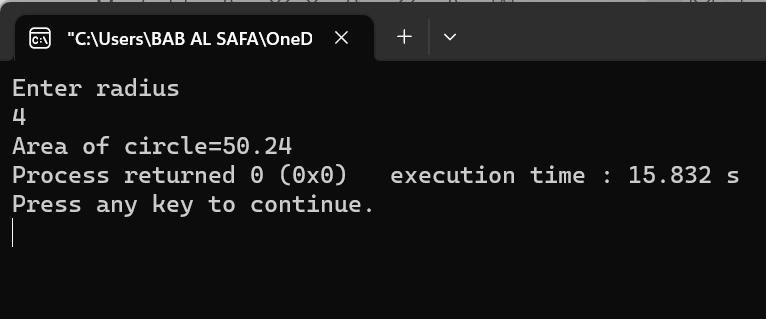


Fig 4.1: Output on console.

**Explanation:**

In this code, defined a class named Circle with two private data members area and radius. The class also has public member functions named getRadius and findArea which provide an interface for accessing and modifying the private data members in a controlled way. By making the data members private, have encapsulated them within the class and prevented direct access from outside the class. The public member functions provide an interface for accessing the data members in a controlled way.

In the main function, an object of the Circle class named cir is created. The getRadius function is called to get the radius of the circle from the user. Then, the findArea function is called to calculate and print the area of the circle to the console.

**Problem No.**: 05

**Problem Name**: C++ Polymorphism .

**Code:**

// C++ program to demonstrate

// function overloading or

// Compile-time Polymorphism

#include <bits/stdc++.h>

using namespace std;

class Geeks {

public:

// Function with 1 int parameter

void func(int x)

{

cout << "value of x is " << x << endl;

}

// Function with same name but

// 1 double parameter

void func(double x)

{

cout << "value of x is " << x << endl;

}

// Function with same name and

// 2 int parameters

void func(int x, int y)

{

cout << "value of x and y is " << x << ", " << y

<< endl;

}

};

// Driver code

int main()

{

Geeks obj1;

// Function being called depends

// on the parameters passed

// func() is called with int value

obj1.func(7);

// func() is called with double value

obj1.func(9.132);

// func() is called with 2 int values

obj1.func(85, 64);

return 0;

}

**Output:**

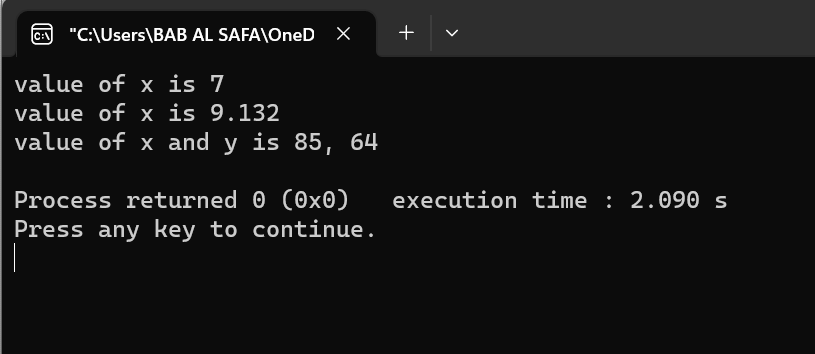


Fig 5.1: Output on console

**Explanation:**

 In this code, defined a class named Geeks with three overloaded functions named func. Each function has a different number and type of parameters, but they all have the same name. [When we call the func function with different parameters, the appropriate version of the function gets called based on the parameters passed](https://www.geeksforgeeks.org/cpp-polymorphism/" \t "https://www.bing.com/_blank).

**Problem No.**: 06

**Problem Name**: C++ Polymorphism .

**Code:**

// C++ program to demonstrate

// Operator Overloading or

// Compile-Time Polymorphism

#include <iostream>

using namespace std;

class Complex {

private:

int real, imag;

public:

Complex(int r = 0, int i = 0)

{

real = r;

imag = i;

}

// This is automatically called

// when '+' is used with between

// two Complex objects

Complex operator+(Complex const& obj)

{

Complex res;

res.real = real + obj.real;

res.imag = imag + obj.imag;

return res;

}

void print() { cout << real << " + i" << imag << endl; }

};

// Driver code

int main()

{

Complex c1(10, 5), c2(2, 4);

// An example call to "operator+"

Complex c3 = c1 + c2;

c3.print();

}

**Output:**

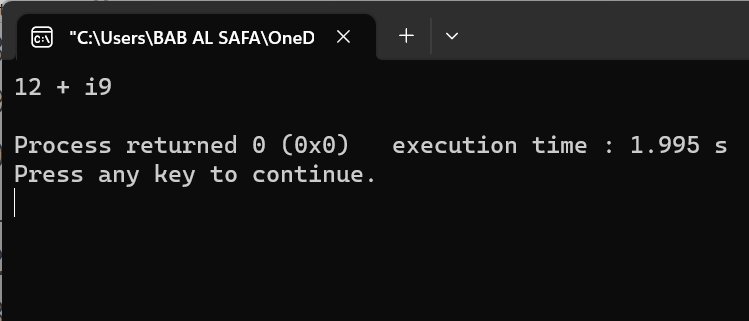


Fig 6.1: Output on console

**Explanation:**

 In this code, defined a class named Complex with two private data members real and image. The class also has a constructor that initializes the data members and an overloaded + operator that adds two Complex objects together. When we call the + operator with two Complex objects, the overloaded version of the operator gets called, which returns a new Complex object that represents the sum of the two input objects.

**Problem No.**: 07

**Problem Name**: C++ Polymorphism .

**Code:**

// C++ program for function overriding with data members

#include <bits/stdc++.h>

using namespace std;

// base class declaration.

class Animal {

public:

string color = "Black";

};

// inheriting Animal class.

class Dog : public Animal {

public:

string color = "Grey";

};

// Driver code

int main(void)

{

Animal d = Dog(); // accessing the field by reference

// variable which refers to derived

cout << d.color;

}

**Output:**

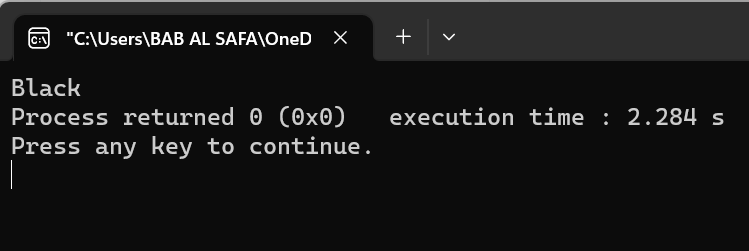


Fig 7.1: Output on console

**Explanation:**

In this code, defined a base class named Animal with a public data member color initialized to “Black”. Have also defined a derived class named Dog that inherits from the Animal class and overrides the color data member with its own value of “Grey”. In the main function, an object of the Dog class is created and assigned to a reference variable of type Animal. When the color data member is accessed through the reference variable, the value of “Black” is printed to the console because the reference variable is of type Animal, not Dog. This is an example of polymorphism in C++ because the same data member name (color) is used in both the base and derived classes, but has different values depending on the context.

**Problem No.**: 08

**Problem Name**: C++ Polymorphism .

**Code:**

// C++ Program to demonstrate

// the Virtual Function

#include <iostream>

using namespace std;

// Declaring a Base class

class GFG\_Base {

public:

// virtual function

virtual void display()

{

cout << "Called virtual Base Class function"

<< "\n\n";

}

void print()

{

cout << "Called GFG\_Base print function"

<< "\n\n";

}

};

// Declaring a Child Class

class GFG\_Child : public GFG\_Base {

public:

void display()

{

cout << "Called GFG\_Child Display Function"

<< "\n\n";

}

void print()

{

cout << "Called GFG\_Child print Function"

<< "\n\n";

}

};

// Driver code

int main()

{

// Create a reference of class GFG\_Base

GFG\_Base\* base;

GFG\_Child child;

base = &child;

// This will call the virtual function

base->GFG\_Base::display();

// this will call the non-virtual function

base->print();

}

**Output:**

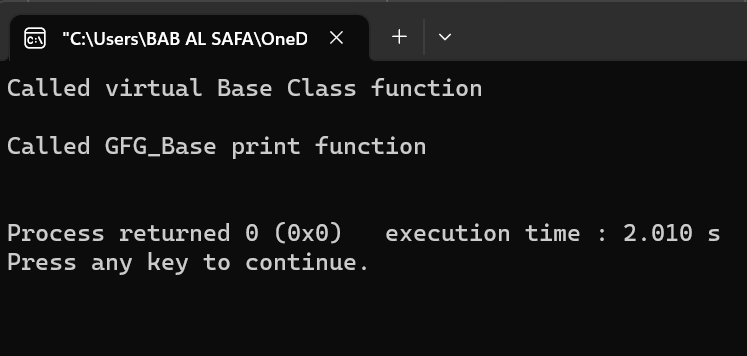


Fig 8.1: Output on console

**Explanation:**

In your code, you have defined a base class named GFG\_Base with a virtual function named display and a non-virtual function named print. You have also defined a derived class named GFG\_Child that overrides the display function with its own implementation. In the main function, an object of the GFG\_Child class is created and assigned to a reference variable of type GFG\_Base. When the display function is called through the reference variable, the overridden version of the function in the GFG\_Child class gets called because it is marked as virtual in the base class. When the print function is called through the reference variable, the non-virtual version of the function in the GFG\_Base class gets called because it is not marked as virtual.

**Problem No.**: 09

**Problem Name**: C++ Polymorphism .

**Code:**

// C++ program for virtual function overriding

#include <bits/stdc++.h>

using namespace std;

class base {

public:

virtual void print()

{

cout << "print base class" << endl;

}

void show() { cout << "show base class" << endl; }

};

class derived : public base {

public:

// print () is already virtual function in

// derived class, we could also declared as

// virtual void print () explicitly

void print() { cout << "print derived class" << endl; }

void show() { cout << "show derived class" << endl; }

};

// Driver code

int main()

{

base\* bptr;

derived d;

bptr = &d;

// Virtual function, binded at

// runtime (Runtime polymorphism)

bptr->print();

// Non-virtual function, binded

// at compile time

bptr->show();

return 0;

}

**Output:**

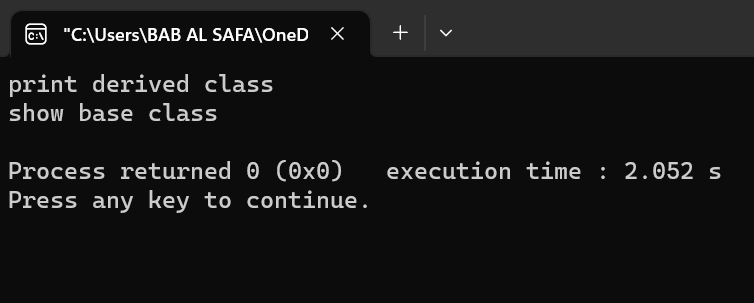


Fig 9.1: Output on console

**Explanation:**

In this code, defined a base class named base with a virtual function named print and a non-virtual function named show. Have also defined a derived class named derived that overrides the print function with its own implementation. In the main function, an object of the derived class is created and assigned to a reference variable of type base. When the print function is called through the reference variable, the overridden version of the function in the derived class gets called because it is marked as virtual in the base class. When the show function is called through the reference variable, the non-virtual version of the function in the base class gets called because it is not marked as virtual.

**Problem No.**: 10

**Problem Name**: Inheritance in C++ .

**Code:**

// Example: define member function without argument within the class

#include<iostream>

using namespace std;

class Person

{

int id;

char name[100];

public:

void set\_p()

{

cout<<"Enter the Id:";

cin>>id;

fflush(stdin);

cout<<"Enter the Name:";

cin.get(name,100);

}

void display\_p()

{

cout<<endl<<id<<"\t"<<name<<"\t";

}

};

class Student: private Person

{

char course[50];

int fee;

public:

void set\_s()

{

set\_p();

cout<<"Enter the Course Name:";

fflush(stdin);

cin.getline(course,50);

cout<<"Enter the Course Fee:";

cin>>fee;

}

void display\_s()

{

display\_p();

cout<<course<<"\t"<<fee<<endl;

}

};

main()

{

Student s;

s.set\_s();

s.display\_s();

return 0;

}

**Output:**

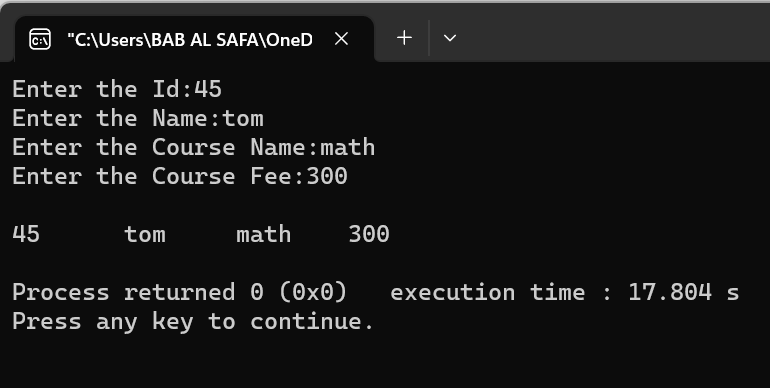


Fig 10.1: Output on console

**Explanation:**

In this code, defined a base class named Person with two private data members id and name. The class also has public member functions named set\_p and display\_p which provide an interface for accessing and modifying the private data members in a controlled way. Have also defined a derived class named Student that inherits from the Person class and adds two private data members course and fee. The class also has public member functions named set\_s and display\_s which provide an interface for accessing and modifying the private data members in a controlled way. In the main function, an object of the Student class is created and the set\_s function is called to set the values of the private data members. Then, the display\_s function is called to print the values of all the data members to the console.

This is an example of inheritance in C++. Inheritance allows new classes to be created from existing classes by inheriting their properties and characteristics. In this example, the Student class is derived from the Person class, which means that it inherits all of its properties and characteristics. This allows us to reuse code that has already been written in the base class, which can save time and reduce errors.

**Problem No.**: 11

**Problem Name**: Inheritance in C++ .

**Code:**

// Example: define member function without argument outside the class

#include<iostream>

using namespace std;

class Person

{

int id;

char name[100];

public:

void set\_p();

void display\_p();

};

void Person::set\_p()

{

cout<<"Enter the Id:";

cin>>id;

fflush(stdin);

cout<<"Enter the Name:";

cin.get(name,100);

}

void Person::display\_p()

{

cout<<endl<<id<<"\t"<<name;

}

class Student: private Person

{

char course[50];

int fee;

public:

void set\_s();

void display\_s();

};

void Student::set\_s()

{

set\_p();

cout<<"Enter the Course Name:";

fflush(stdin);

cin.getline(course,50);

cout<<"Enter the Course Fee:";

cin>>fee;

}

void Student::display\_s()

{

display\_p();

cout<<"\t"<<course<<"\t"<<fee;

}

main()

{

Student s;

s.set\_s();

s.display\_s();

return 0;

}

**Output:**

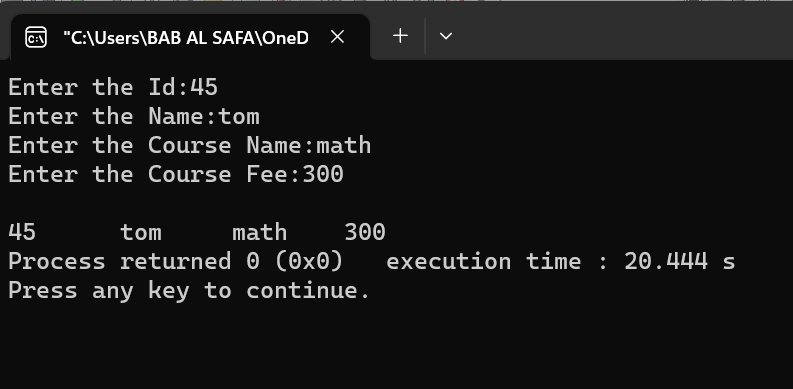


Fig 11.1: Output on console

**Explanation:**

In this code, defined a base class named Person with two private data members id and name. The class also has public member functions named set\_p and display\_p which provide an interface for accessing and modifying the private data members in a controlled way. Have also defined a derived class named Student that inherits from the Person class and adds two private data members course and fee. The class also has public member functions named set\_s and display\_s which provide an interface for accessing and modifying the private data members in a controlled way. In the main function, an object of the Student class is created and the set\_s function is called to set the values of the private data members. Then, the display\_s function is called to print the values of all the data members to the console.

This is an example of inheritance in C++. Inheritance allows new classes to be created from existing classes by inheriting their properties and characteristics. In this example, the Student class is derived from the Person class, which means that it inherits all of its properties and characteristics. This allows us to reuse code that has already been written in the base class, which can save time and reduce errors.

**Problem No.**: 12

**Problem Name**: Inheritance in C++ .

**Code:**

// Example: define member function with argument outside the class

#include<iostream>

#include<string.h>

using namespace std;

class Person

{

int id;

char name[100];

public:

void set\_p(int,char[]);

void display\_p();

};

void Person::set\_p(int id,char n[])

{

this->id=id;

strcpy(this->name,n);

}

void Person::display\_p()

{

cout<<endl<<id<<"\t"<<name;

}

class Student: private Person

{

char course[50];

int fee;

public:

void set\_s(int,char[],char[],int);

void display\_s();

};

void Student::set\_s(int id,char n[],char c[],int f)

{

set\_p(id,n);

strcpy(course,c);

fee=f;

}

void Student::display\_s()

{

display\_p();

cout<<"t"<<course<<"\t"<<fee;

}

main()

{

Student s;

s.set\_s(1001,"Ram","B.Tech",2000);

s.display\_s();

return 0;

}

**Output:**

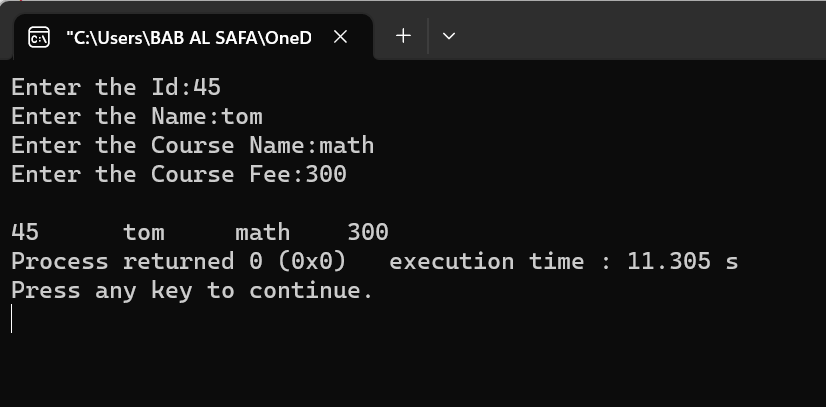


Fig 12.1: Output on console

**Explanation:**

In this code, defined a base class named Person with two private data members id and name. The class also has public member functions named set\_p and display\_p which provide an interface for accessing and modifying the private data members in a controlled way. Have also defined a derived class named Student that inherits from the Person class and adds two private data members course and fee. The class also has public member functions named set\_s and display\_s which provide an interface for accessing and modifying the private data members in a controlled way. In the main function, an object of the Student class is created and the set\_s function is called to set the values of the private data members. Then, the display\_s function is called to print the values of all the data members to the console.

This is an example of inheritance in C++. Inheritance allows new classes to be created from existing classes by inheriting their properties and characteristics. In this example, the Student class is derived from the Person class, which means that it inherits all of its properties and characteristics. This allows us to reuse code that has already been written in the base class, which can save time and reduce errors.

**Problem No.**: 13

**Problem Name**: Inheritance in C++ .

**Code:**

// Example: define member function with argument outside the class

#include<iostream>

#include<string.h>

using namespace std;

class Person

{

int id;

char name[100];

public:

void set\_p(int,char[]);

void display\_p();

};

void Person::set\_p(int id,char n[])

{

this->id=id;

strcpy(this->name,n);

}

void Person::display\_p()

{

cout<<endl<<id<<"\t"<<name;

}

class Student: private Person

{

char course[50];

int fee;

public:

void set\_s(int,char[],char[],int);

void display\_s();

};

void Student::set\_s(int id,char n[],char c[],int f)

{

set\_p(id,n);

strcpy(course,c);

fee=f;

}

void Student::display\_s()

{

display\_p();

cout<<"t"<<course<<"\t"<<fee;

}

main()

{

Student s;

s.set\_s(1001,"Ram","B.Tech",2000);

s.display\_s();

return 0;

}

**Output:**

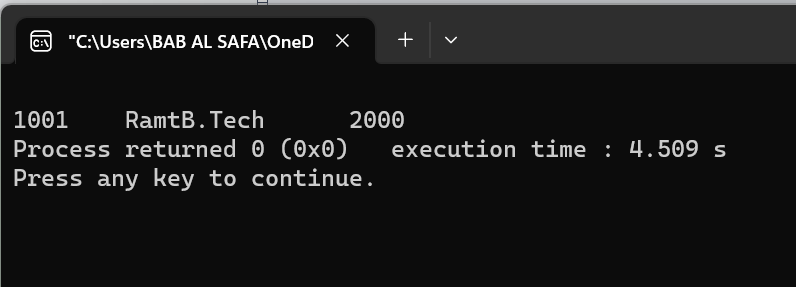


Fig 13.1: Output on console

**Explanation:**

[In this code, Student is derived from Person using private inheritance](https://www.geeksforgeeks.org/inheritance-in-c/" \t "https://www.bing.com/_blank). [This means that the public and protected members of the base class become private members of the derived class](https://www.geeksforgeeks.org/inheritance-in-c/" \t "https://www.bing.com/_blank). [The set\_p and display\_p functions are public members of the base class Person and can be accessed by the derived class Student using the scope resolution operator](https://www.geeksforgeeks.org/inheritance-in-c/" \t "https://www.bing.com/_blank). [The set\_s function in the derived class sets the values of the member variables of both the base and derived classes using the set\_p function in the base class](https://www.geeksforgeeks.org/inheritance-in-c/" \t "https://www.bing.com/_blank). [The display\_s function in the derived class displays the values of member variables of both classes using the display\_p function in the base class](https://www.geeksforgeeks.org/inheritance-in-c/" \t "https://www.bing.com/_blank)

**Problem No.**: 14

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program to demonstrate implementation

// of Inheritance

#include <bits/stdc++.h>

using namespace std;

// Base class

class Parent {

public:

int id\_p;

};

// Sub class inheriting from Base Class(Parent)

class Child : public Parent {

public:

int id\_c;

};

// main function

int main()

{

Child obj1;

// An object of class child has all data members

// and member functions of class parent

obj1.id\_c = 7;

obj1.id\_p = 91;

cout << "Child id is: " << obj1.id\_c << '\n';

cout << "Parent id is: " << obj1.id\_p << '\n';

return 0;

}

**Output:**

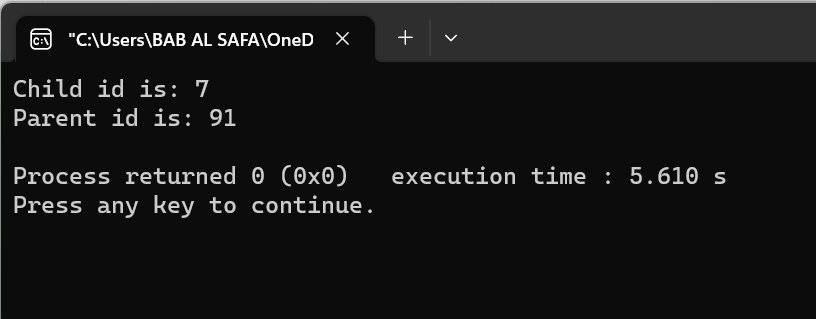


Fig 14.1: Output on console

**Explanation:**

 In this code, Child is derived from Parent using public inheritance . This means that the public and protected members of the base class become public and protected members of the derived class respectively . The id\_p member variable in the base class is inherited by the derived class Child and can be accessed using the dot operator . The id\_c member variable in the derived class is unique to it and can also be accessed using the dot operator . When an object of the derived class is created, it has all data members and member functions of the base class . In your example, obj1 is an object of the derived class Child and has both id\_p and id\_c member variables .

**Problem No.**: 15

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program to explain

// Single inheritance

#include<iostream>

using namespace std;

// base class

class Vehicle {

public:

Vehicle()

{

cout << "This is a Vehicle\n";

}

};

// sub class derived from a single base classes

class Car : public Vehicle {

};

// main function

int main()

{

// Creating object of sub class will

// invoke the constructor of base classes

Car obj;

return 0;

}

**Output:**

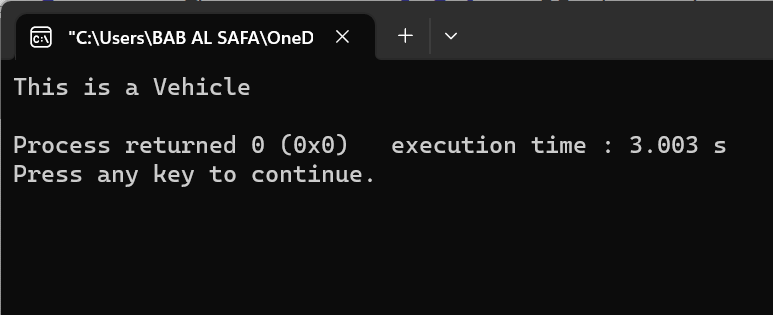


Fig 15.1: Output on console

**Explanation:**

In this code, Car is derived from the Vehicle class using public inheritance . This means that the public members of the base class become public members of the derived class . When an object of the derived class is created, it invokes the constructor of the base class . In your example, when an object of Car is created, it invokes the constructor of Vehicle and prints “This is a Vehicle” to the console .

**Problem No.**: 16

**Problem Name**: Inheritance in C++ .

**Code:**

// Example:

#include<iostream>

using namespace std;

class A

{

protected:

int a;

public:

void set\_A()

{

cout<<"Enter the Value of A=";

cin>>a;

}

void disp\_A()

{

cout<<endl<<"Value of A="<<a;

}

};

class B: public A

{

int b,p;

public:

void set\_B()

{

set\_A();

cout<<"Enter the Value of B=";

cin>>b;

}

void disp\_B()

{

disp\_A();

cout<<endl<<"Value of B="<<b;

}

void cal\_product()

{

p=a\*b;

cout<<endl<<"Product of "<<a<<" \* "<<b<<" = "<<p;

}

};

main()

{

B \_b;

\_b.set\_B();

\_b.cal\_product();

return 0;

}

**Output:**

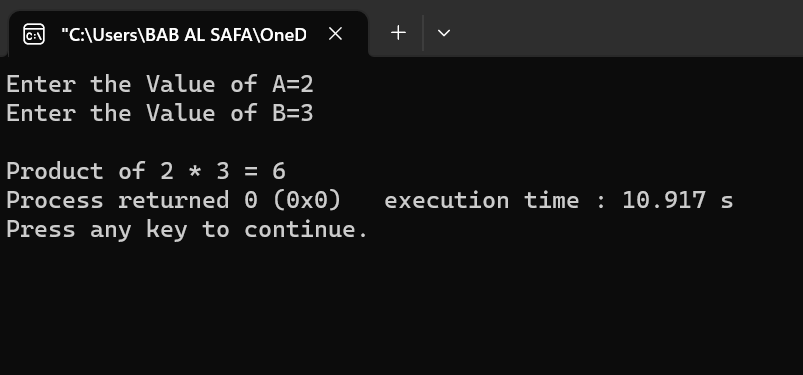


Fig 16.1: Output on console

**Explanation:**

In this code, B is derived from A using public inheritance . This means that the public and protected members of the base class become public and protected members of the derived class respectively . The set\_A and disp\_A functions are public members of the base class A and can be accessed by the derived class B using the scope resolution operator . The set\_B function in the derived class sets the values of the member variables of both the base and derived classes using the set\_A function in the base class . The disp\_B function in the derived class displays the values of member variables of both classes using the disp\_A function in the base class . The cal\_product function in the derived class calculates the product of member variables of both classes .

**Problem No.**: 17

**Problem Name**: Inheritance in C++ .

**Code:**

// Example:

#include<iostream>

using namespace std;

class A

{

protected:

int a;

public:

void set\_A(int x)

{

a=x;

}

void disp\_A()

{

cout<<endl<<"Value of A="<<a;

}

};

class B: public A

{

int b,p;

public:

void set\_B(int x,int y)

{

set\_A(x);

b=y;

}

void disp\_B()

{

disp\_A();

cout<<endl<<"Value of B="<<b;

}

void cal\_product()

{

p=a\*b;

cout<<endl<<"Product of "<<a<<" \* "<<b<<" = "<<p;

}

};

main()

{

B \_b;

\_b.set\_B(4,5);

\_b.cal\_product();

return 0;

}

**Output:**

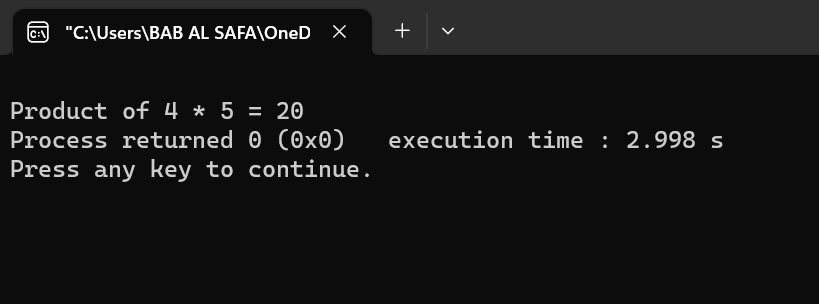


Fig 17.1: Output on console

**Explanation:**

In this code, B is derived from A using public inheritance . This means that the public and protected members of the base class become public and protected members of the derived class respectively . The set\_A function in the base class sets the value of the member variable a . The disp\_A function in the base class displays the value of the member variable a . The set\_B function in the derived class sets the values of the member variables of both the base and derived classes using the set\_A function in the base class . The disp\_B function in the derived class displays the values of member variables of both classes using the disp\_A function in the base class . The cal\_product function in the derived class calculates the product of member variables of both classes .

**Problem No.**: 18

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program to explain

// multiple inheritance

#include <iostream>

using namespace std;

// first base class

class Vehicle {

public:

Vehicle() { cout << "This is a Vehicle\n"; }

};

// second base class

class FourWheeler {

public:

FourWheeler()

{

cout << "This is a 4 wheeler Vehicle\n";

}

};

// sub class derived from two base classes

class Car : public Vehicle, public FourWheeler {

};

// main function

int main()

{

// Creating object of sub class will

// invoke the constructor of base classes.

Car obj;

return 0;

}

**Output:**

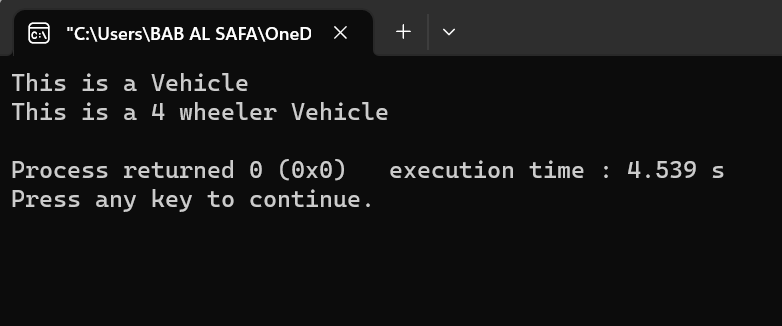


Fig 18.1: Output on console

**Explanation:**

 In this code, Car is derived from both Vehicle and FourWheeler classes using public inheritance . This means that the public members of both base classes become public members of the derived class . When an object of the derived class is created, it invokes the constructor of both base classes . In your example, when an object of Car is created, it invokes the constructors of both Vehicle and FourWheeler and prints “This is a Vehicle” and “This is a 4 wheeler Vehicle” to the console respectively.

**Problem No.**: 19

**Problem Name**: Inheritance in C++ .

**Code:**

// Example:

#include<iostream>

using namespace std;

class A

{

protected:

int a;

public:

void set\_A()

{

cout<<"Enter the Value of A=";

cin>>a;

}

void disp\_A()

{

cout<<endl<<"Value of A="<<a;

}

};

class B: public A

{

protected:

int b;

public:

void set\_B()

{

cout<<"Enter the Value of B=";

cin>>b;

}

void disp\_B()

{

cout<<endl<<"Value of B="<<b;

}

};

class C: public B

{

int c,p;

public:

void set\_C()

{

cout<<"Enter the Value of C=";

cin>>c;

}

void disp\_C()

{

cout<<endl<<"Value of C="<<c;

}

void cal\_product()

{

p=a\*b\*c;

cout<<endl<<"Product of "<<a<<" \* "<<b<<" \* "<<c<<" = "<<p;

}

};

main()

{

C \_c;

\_c.set\_A();

\_c.set\_B();

\_c.set\_C();

\_c.disp\_A();

\_c.disp\_B();

\_c.disp\_C();

\_c.cal\_product();

return 0;

}

**Output:**

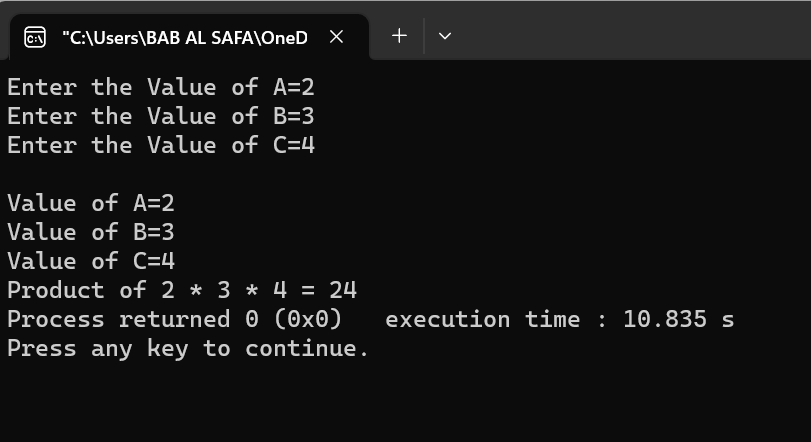


Fig 19.1: Output on console

**Explanation:**

In this code, C is derived from both A and B classes using public inheritance . This means that the public members of both base classes become public members of the derived class . The set\_A and disp\_A functions are public members of the base class A and can be accessed by the derived class C using the scope resolution operator . The set\_B and disp\_B functions in the derived class set and display the value of member variable b respectively . The set\_C and disp\_C functions in the derived class set and display the value of member variable c respectively . The cal\_product function in the derived class calculates the product of member variables of all three classes .

**Problem No.**: 20

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program to implement

// Hierarchical Inheritance

#include <iostream>

using namespace std;

// base class

class Vehicle {

public:

Vehicle() { cout << "This is a Vehicle\n"; }

};

// first sub class

class Car : public Vehicle {

};

// second sub class

class Bus : public Vehicle {

};

// main function

int main()

{

// Creating object of sub class will

// invoke the constructor of base class.

Car obj1;

Bus obj2;

return 0;

}

**Output:**

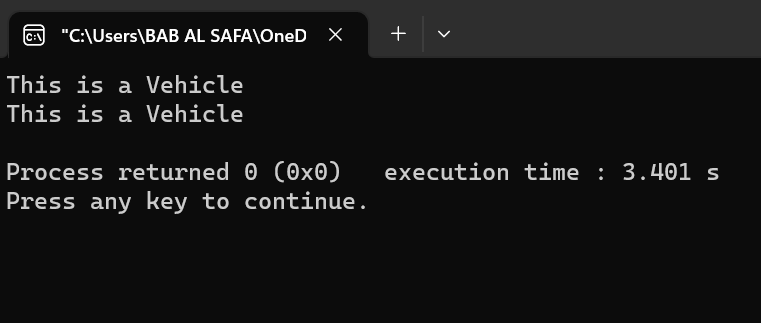


Fig 20.1: Output on console

**Explanation:**

In this code, Car and Bus are derived from the Vehicle class using hierarchical inheritance . This means that the public members of the base class become public members of the derived classes . When an object of the derived class is created, it invokes the constructor of the base class . In your example, when an object of Car is created, it invokes the constructor of Vehicle and prints “This is a Vehicle” to the console . Similarly, when an object of Bus is created, it also invokes the constructor of Vehicle and prints “This is a Vehicle” to the console .

**Problem No.**: 21

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program to implement

// Multilevel Inheritance

#include <iostream>

using namespace std;

// base class

class Vehicle {

public:

Vehicle() { cout << "This is a Vehicle\n"; }

};

// first sub\_class derived from class vehicle

class fourWheeler : public Vehicle {

public:

fourWheeler()

{

cout << "Objects with 4 wheels are vehicles\n";

}

};

// sub class derived from the derived base class fourWheeler

class Car : public fourWheeler {

public:

Car() { cout << "Car has 4 Wheels\n"; }

};

// main function

int main()

{

// Creating object of sub class will

// invoke the constructor of base classes.

Car obj;

return 0;

}

**Output:**

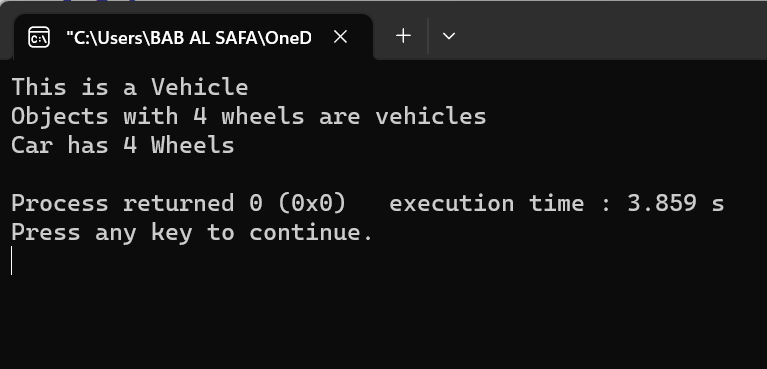


Fig 21.1: Output on console

**Explanation:**

In this code, Car is derived from both Vehicle and fourWheeler classes using public inheritance . This means that the public members of both base classes become public members of the derived class . When an object of the derived class is created, it invokes the constructor of both base classes . In your example, when an object of Car is created, it invokes the constructors of both Vehicle and fourWheeler and prints “This is a Vehicle”, “Objects with 4 wheels are vehicles” and “Car has 4 Wheels” to the console respectively .

**Problem No.**: 21

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program to implement

// Multilevel Inheritance

#include <iostream>

using namespace std;

// base class

class Vehicle {

public:

Vehicle() { cout << "This is a Vehicle\n"; }

};

// first sub\_class derived from class vehicle

class fourWheeler : public Vehicle {

public:

fourWheeler()

{

cout << "Objects with 4 wheels are vehicles\n";

}

};

// sub class derived from the derived base class fourWheeler

class Car : public fourWheeler {

public:

Car() { cout << "Car has 4 Wheels\n"; }

};

// main function

int main()

{

// Creating object of sub class will

// invoke the constructor of base classes.

Car obj;

return 0;

}

**Output:**

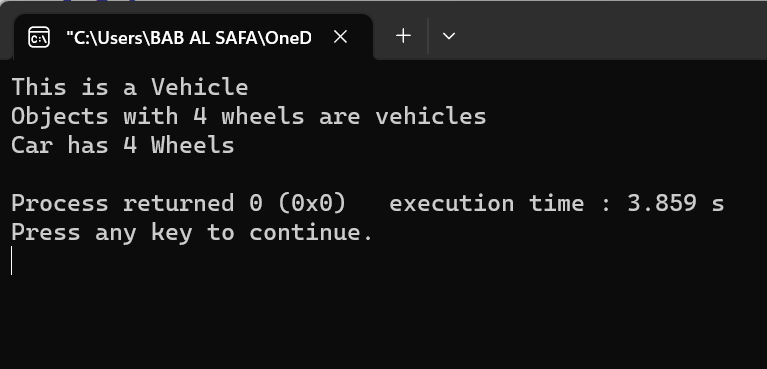


Fig 21.1: Output on console

**Explanation:**

In this code, Car is derived from both Vehicle and fourWheeler classes using public inheritance . This means that the public members of both base classes become public members of the derived class . When an object of the derived class is created, it invokes the constructor of both base classes . In your example, when an object of Car is created, it invokes the constructors of both Vehicle and fourWheeler and prints “This is a Vehicle”, “Objects with 4 wheels are vehicles” and “Car has 4 Wheels” to the console respectively .

**Problem No.**: 22

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program for Hybrid Inheritance

#include <iostream>

using namespace std;

// base class

class Vehicle {

public:

Vehicle() { cout << "This is a Vehicle\n"; }

};

// base class

class Fare {

public:

Fare() { cout << "Fare of Vehicle\n"; }

};

// first sub class

class Car : public Vehicle {

};

// second sub class

class Bus : public Vehicle, public Fare {

};

// main function

int main()

{

// Creating object of sub class will

// invoke the constructor of base class.

Bus obj2;

return 0;

}

**Output:**

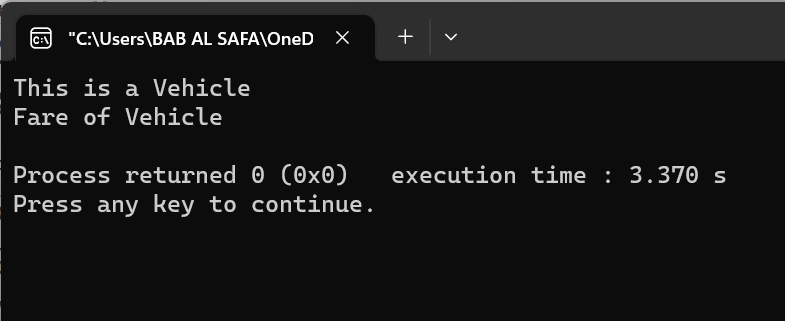


Fig 22.1: Output on console

**Explanation:**

In this code, Bus is derived from both Vehicle and Fare classes using hybrid inheritance . This means that the public members of both base classes become public members of the derived class . When an object of the derived class is created, it invokes the constructor of both base classes . In your example, when an object of Bus is created, it invokes the constructors of both Vehicle and Fare and prints “This is a Vehicle” and “Fare of Vehicle” to the console respectively .

**Problem No.**: 23

**Problem Name**: Inheritance in C++ .

**Code:**

// Example:

#include <iostream>

using namespace std;

class A

{

protected:

int a;

public:

void get\_a()

{

cout << "Enter the value of 'a' : ";

cin>>a;

}

};

class B : public A

{

protected:

int b;

public:

void get\_b()

{

cout << "Enter the value of 'b' : ";

cin>>b;

}

};

class C

{

protected:

int c;

public:

void get\_c()

{

cout << "Enter the value of c is : ";

cin>>c;

}

};

class D : public B, public C

{

protected:

int d;

public:

void mul()

{

get\_a();

get\_b();

get\_c();

cout << "Multiplication of a,b,c is : " <<a\*b\*c;

}

};

int main()

{

D d;

d.mul();

return 0;

}

**Output:**

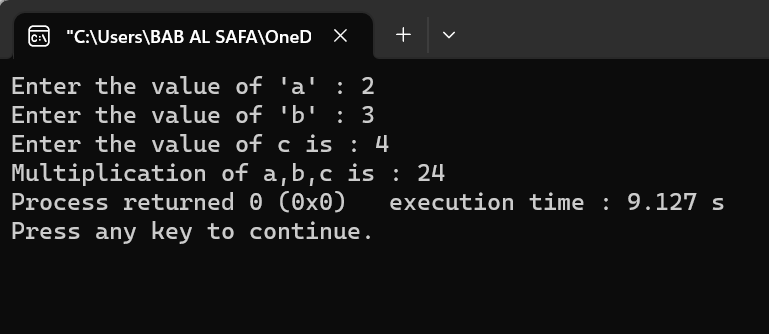


Fig 23.1: Output on console

**Explanation:**

In this code, D is derived from both B and C classes using multiple inheritance . This means that the public members of both base classes become public members of the derived class . The get\_a, get\_b and get\_c functions are public members of the base classes and can be accessed by the derived class using the scope resolution operator . The mul function in the derived class gets the values of member variables of all three classes using the get\_a, get\_b and get\_c functions in the base classes . It then calculates the product of member variables of all three classes and displays it to the console .

**Problem No.**: 24

**Problem Name**: Inheritance in C++ .

**Code:**

// C++ program demonstrating ambiguity in Multipath

// Inheritance

#include <iostream>

using namespace std;

class ClassA {

public:

int a;

};

class ClassB : public ClassA {

public:

int b;

};

class ClassC : public ClassA {

public:

int c;

};

class ClassD : public ClassB, public ClassC {

public:

int d;

};

int main()

{

ClassD obj;

// obj.a = 10; // Statement 1, Error

// obj.a = 100; // Statement 2, Error

obj.ClassB::a = 10; // Statement 3

obj.ClassC::a = 100; // Statement 4

obj.b = 20;

obj.c = 30;

obj.d = 40;

cout << " a from ClassB : " << obj.ClassB::a;

cout << "\n a from ClassC : " << obj.ClassC::a;

cout << "\n b : " << obj.b;

cout << "\n c : " << obj.c;

cout << "\n d : " << obj.d << '\n';

}

**Output:**

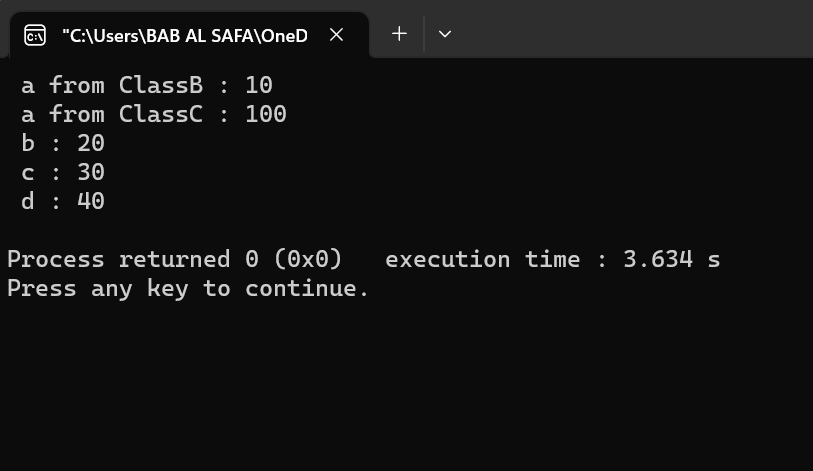


Fig 24.1: Output on console

**Explanation:**

In this code, ClassD is derived from both ClassB and ClassC classes using multiple inheritance . This means that the public members of both base classes become public members of the derived class . In your example, there is an ambiguity in the a member variable of the ClassA base class because it is inherited by both ClassB and ClassC classes . Therefore, when an object of the derived class is created, it cannot access the member variable a directly . In your example, you can access the member variable a of both base classes using the scope resolution operator . For example, you can use obj.ClassB::a = 10; to set the value of a in the ClassB base class and obj.ClassC::a = 100; to set the value of a in the ClassC base class .

**Problem No.**: 25

**Problem Name**: Inheritance in C++ .

**Code:**

#include<iostream>

Using namespace std;

class ClassA

{

public:

int a;

};

class ClassB : virtual public ClassA

{

public:

int b;

};

class ClassC : virtual public ClassA

{

public:

int c;

};

class ClassD : public ClassB, public ClassC

{

public:

int d;

};

int main()

{

ClassD obj;

obj.a = 10; // Statement 3

obj.a = 100; // Statement 4

obj.b = 20;

obj.c = 30;

obj.d = 40;

cout << "\n a : " << obj.a;

cout << "\n b : " << obj.b;

cout << "\n c : " << obj.c;

cout << "\n d : " << obj.d << '\n';

}

**Output:**

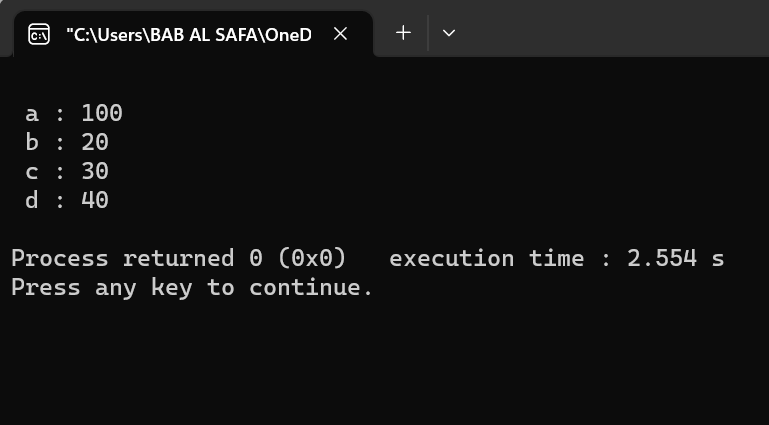


Fig 25.1: Output on console

**Explanation:**

 In this code, ClassD is derived from both ClassB and ClassC classes using multiple inheritance . This means that the public members of both base classes become public members of the derived class . In your example, there is an ambiguity in the a member variable of the ClassA base class because it is inherited by both ClassB and ClassC classes . Therefore, when an object of the derived class is created, it cannot access the member variable a directly . In your example, you can access the member variable a of both base classes using the scope resolution operator . For example, you can use obj.ClassB::a = 10; to set the value of a in the ClassB base class and obj.ClassC::a = 100; to set the value of a in the ClassC base class .

**Problem No.**: 26

**Problem Name**: Abstraction in C++ .

**Code:**

// C++ Program to Demonstrate the

// working of Abstraction

#include <iostream>

using namespace std;

class implementAbstraction {

private:

int a, b;

public:

// method to set values of

// private members

void set(int x, int y)

{

a = x;

b = y;

}

void display()

{

cout << "a = " << a << endl;

cout << "b = " << b << endl;

}

};

int main()

{

implementAbstraction obj;

obj.set(10, 20);

obj.display();

return 0;

}

**Output:**

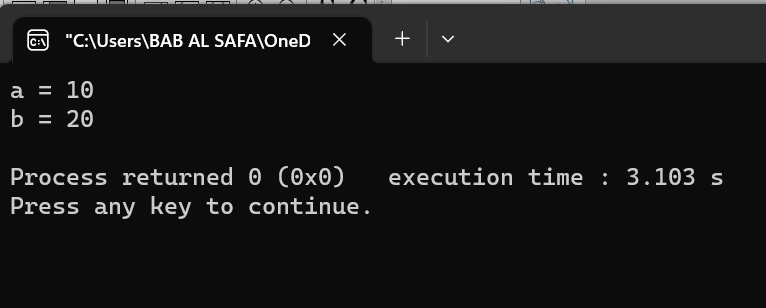


Fig 26.1: Output on console

**Explanation:**

In this code, the program demonstrates the working of abstraction in C++ . Abstraction is a process of hiding the implementation details and showing only the functionality to the user . In this code, the implement Abstraction class has two private member variables a and b that can only be accessed by the member functions of the class . The set function sets the values of these member variables and the display function displays them to the console . The user does not need to know how these functions work internally, but only how to use them to set and display values . This is an example of abstraction in C++ .

**Problem No.**: 27

**Problem Name**: Abstraction in C++ .

**Code:**

#include<iostream>

using namespace std;

class Vehicle

{

public:

void company()

{

cout<<"GFG\n";

}

public:

void model()

{

cout<<"SIMPLE\n";

}

public:

void color()

{

cout<<"Red/GREEN/Silver\n";

}

public:

void cost()

{

cout<<"Rs. 60000 to 900000\n";

}

public:

void oil()

{

cout<<"PETRO\n";

}

private:

void piston()

{

cout<<"4 piston\n";

}

private:

void manWhoMade()

{

cout<<"Markus Librette\n";

}

};

int main()

{

Vehicle obj;

obj.company();

obj.model();

obj.color();

obj.cost();

obj.oil();

}

**Output:**

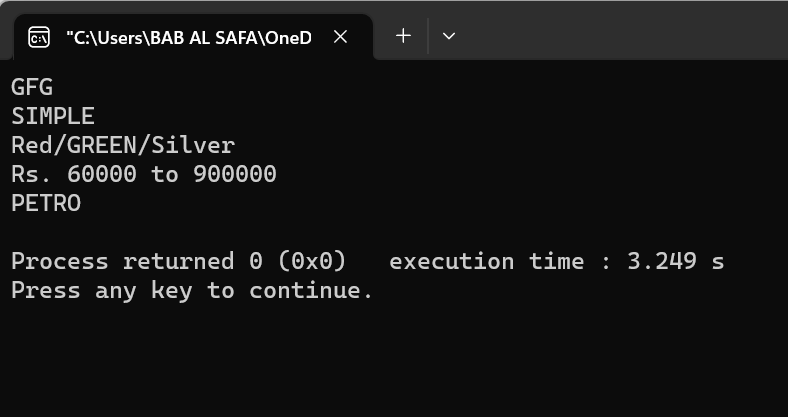


Fig 27.1: Output on console

**Explanation:**

In this code, the program demonstrates the working of abstraction in C++ . Abstraction is a process of hiding the implementation details and showing only the functionality to the user . In your example, the Vehicle class has several public member functions that can be accessed by the user to get information about a vehicle such as its company, model, color, cost, and oil type. These functions provide only the necessary information to the user and hide the implementation details such as the number of pistons or the name of the person who made it. This is an example of abstraction in C++ .