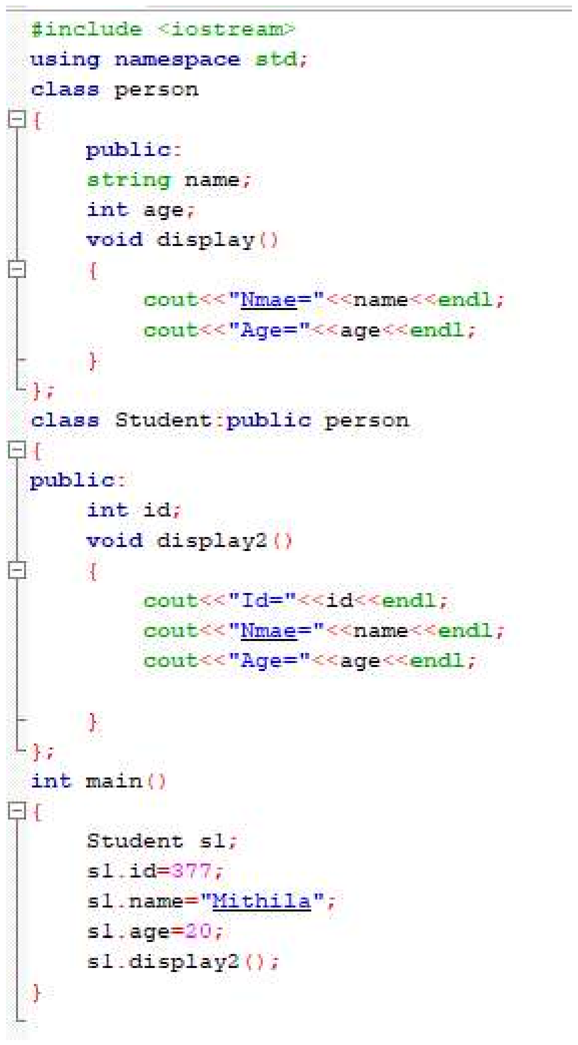
1.What is inheritance?

Inheritance in C++ is a feature that allows a class to inherit the properties and behaviors of another class. The class that inherits is called the derived class, and the class that is inherited from is called the base class.

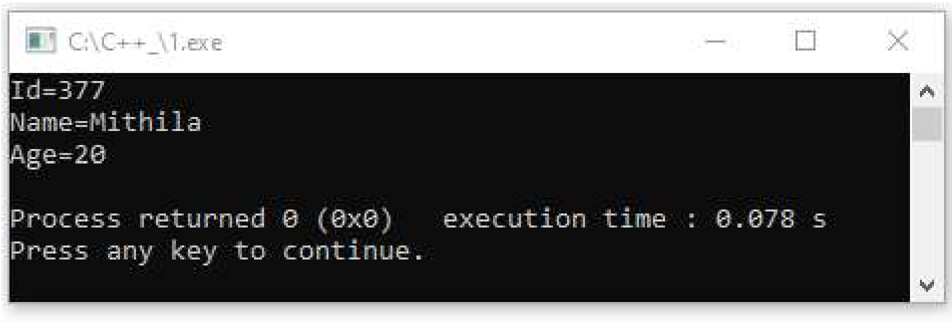
Example:



```
#include <iostream>
using namespace std;
class person
{
    public:
    string name;
    int age;
    void display()
    {
        cout<<"Nmae="<<name<<endl;
        cout<<"Age="<<age<<endl;
    }
};
class Student:public person
{
public:
    int id;
    void display2()
    {
        cout<<"Id="<<id<<endl;
        cout<<"Nmae="<<name<<endl;
        cout<<"Age="<<age<<endl;

    }
};
int main()
{
    Student s1;
    s1.id=377;
    s1.name="Mithila";
    s1.age=20;
    s1.display2();
}
```

Output:



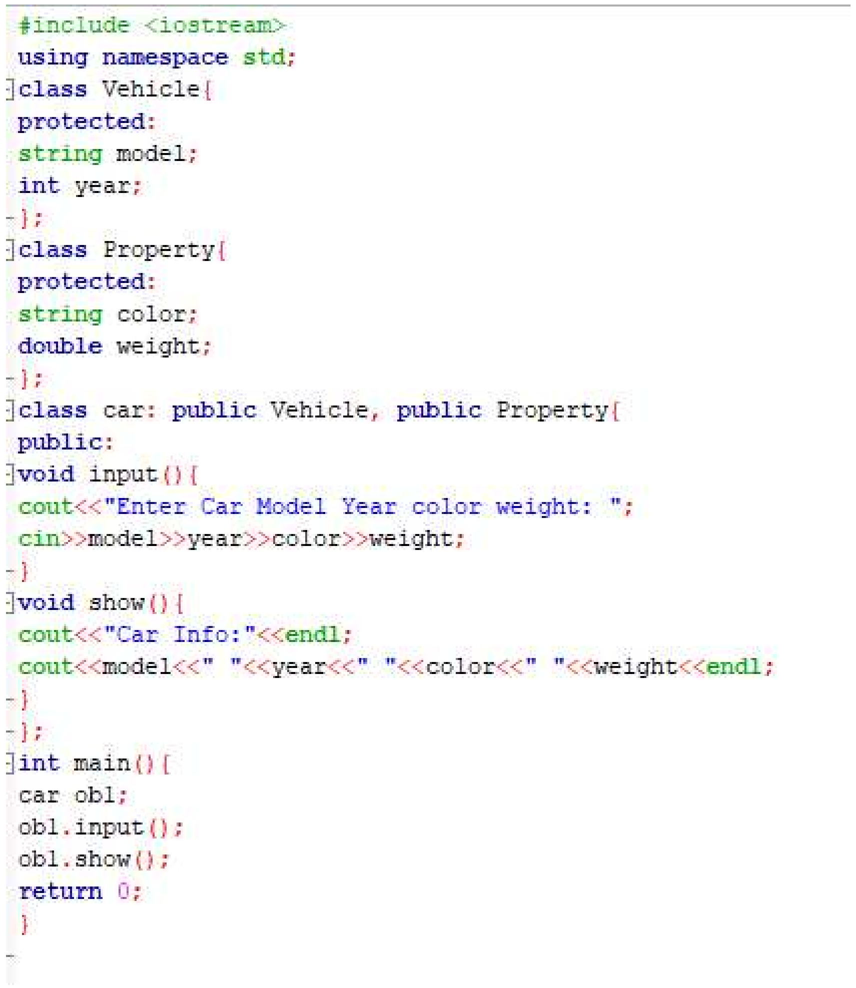
```
Id=377
Name=Mithila
Age=20

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

2.What is protected members?

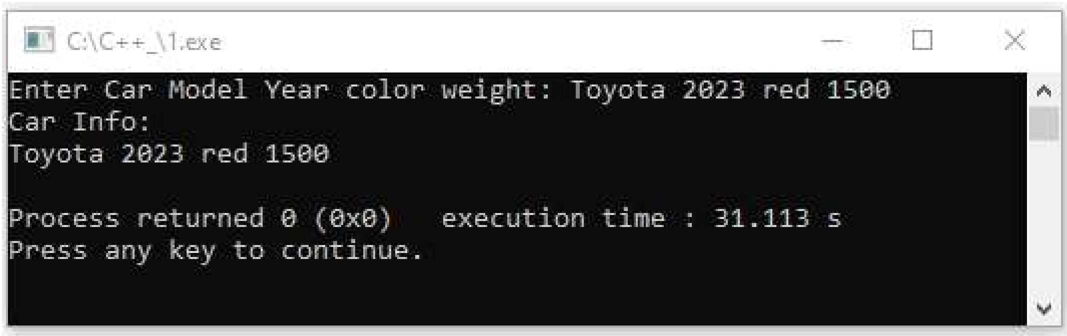
In C++, protected members are a type of access specifier that controls the accessibility of class members. They are less restrictive than private members, which can only be accessed within the class itself, but more restrictive than public members, which can be accessed from anywhere in the program.

Example



```
#include <iostream>
using namespace std;
class Vehicle{
protected:
string model;
int year;
};
class Property{
protected:
string color;
double weight;
};
class car: public Vehicle, public Property{
public:
void input(){
cout<<"Enter Car Model Year color weight: ";
cin>>model>>year>>color>>weight;
}
void show(){
cout<<"Car Info:"<<endl;
cout<<model<<" "<<year<<" "<<color<<" "<<weight<<endl;
}
};
int main(){
car obl;
obl.input();
obl.show();
return 0;
}
```

Output



C:\C++_\1.exe

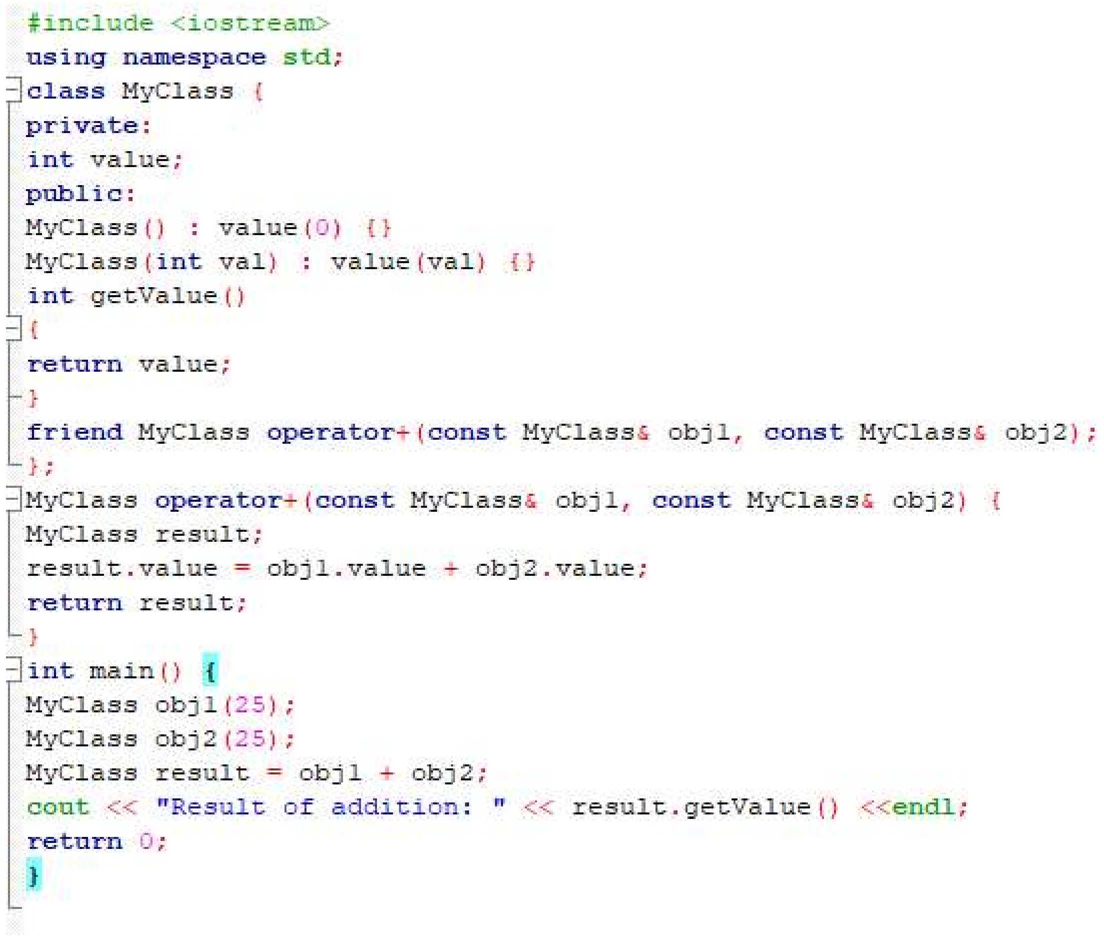
```
Enter Car Model Year color weight: Toyota 2023 red 1500
Car Info:
Toyota 2023 red 1500

Process returned 0 (0x0)   execution time : 31.113 s
Press any key to continue.
```

3. Friend operator overloading?

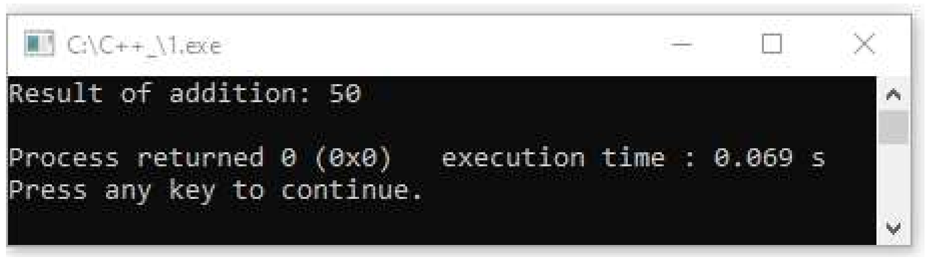
Friend operator overloading is a technique in C++ that allows you to define custom behaviors for operators when applied to objects of a class. By making a function a friend of a class, you grant it access to the class's private members, enabling you to overload operators in ways that would not be possible using regular member functions.

Example



```cpp
#include <iostream>
using namespace std;
class MyClass {
private:
int value;
public:
MyClass() : value(0) {}
MyClass(int val) : value(val) {}
int getValue()
{
return value;
}
friend MyClass operator+(const MyClass& obj1, const MyClass& obj2);
};
MyClass operator+(const MyClass& obj1, const MyClass& obj2) {
MyClass result;
result.value = obj1.value + obj2.value;
return result;
}
int main() {
MyClass obj1(25);
MyClass obj2(25);
MyClass result = obj1 + obj2;
cout << "Result of addition: " << result.getValue() <<endl;
return 0;
}
```

Output



C:\C++_\1.exe

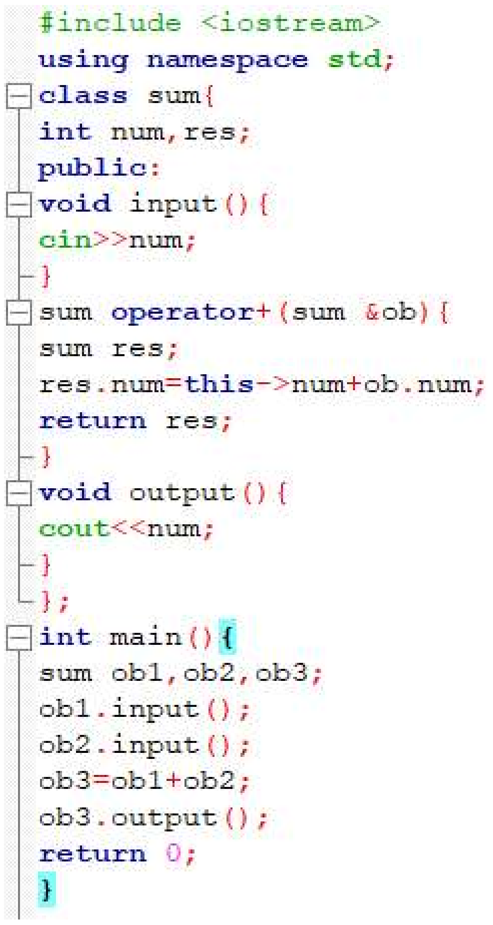
```
Result of addition: 50

Process returned 0 (0x0)   execution time : 0.069 s
Press any key to continue.
```

4. Operator Overloading.

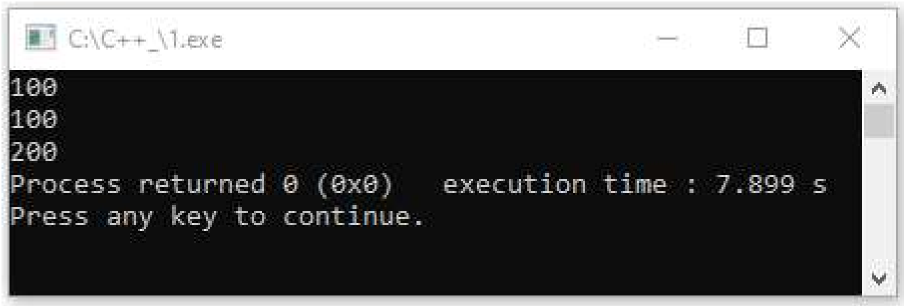
Operator overloading is a feature in object-oriented programming languages that allows you to define custom behaviors for standard operators when applied to user-defined types (classes or structures). In other words, you can redefine the way operators work for your own classes, enabling you to provide meaningful and intuitive implementations.

Example



```
#include <iostream>
using namespace std;
class sum{
int num,res;
public:
void input(){
cin>>num;
}
sum operator+(sum &ob){
sum res;
res.num=this->num+ob.num;
return res;
}
void output(){
cout<<num;
}
};
int main(){
sum ob1,ob2,ob3;
ob1.input();
ob2.input();
ob3=ob1+ob2;
ob3.output();
return 0;
}
```

Output

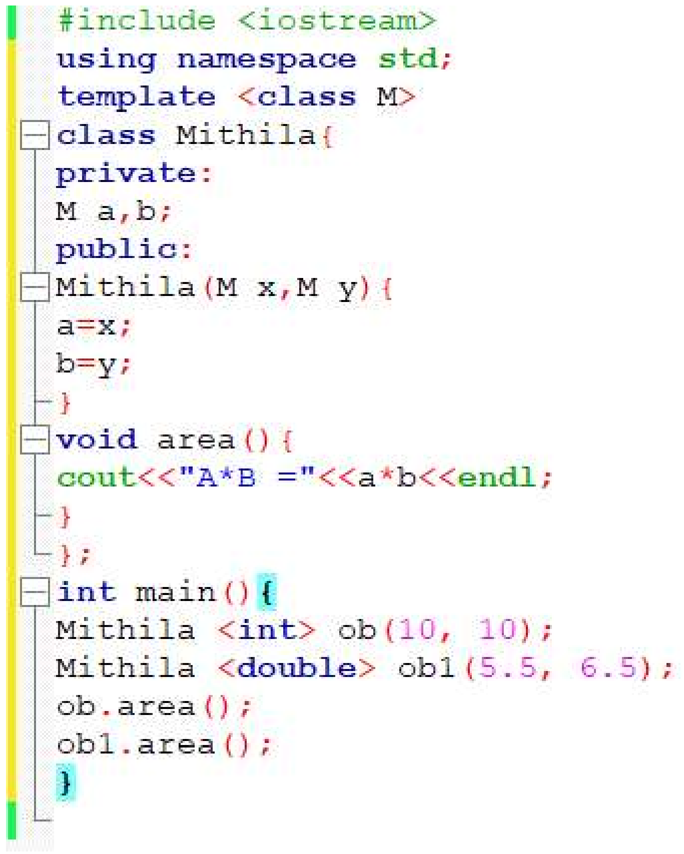


```
100
100
200
Process returned 0 (0x0)   execution time : 7.899 s
Press any key to continue.
```

5.What is Generic class?

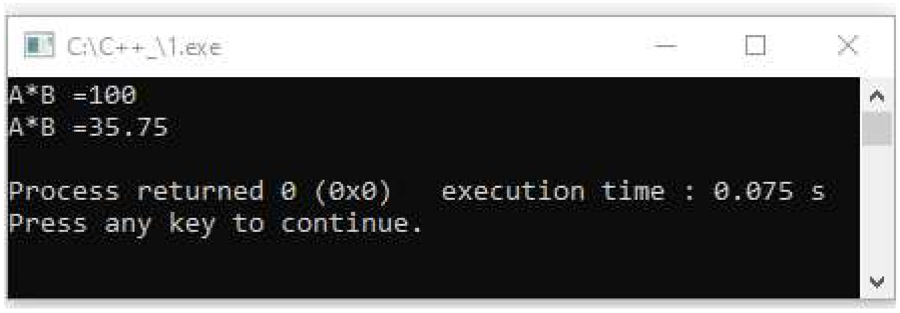
A generic class, often referred to as a template class in C++, is a class that is defined with one or more type parameters. These type parameters can represent any data type, and the class can be used with different data types without the need to create separate class implementations for each type. This promotes code reusability and flexibility.

Example



```
#include <iostream>
using namespace std;
template <class M>
class Mithila{
private:
M a,b;
public:
Mithila(M x,M y){
a=x;
b=y;
}
void area(){
cout<<"A*B ="<<a*b<<endl;
}
};
int main(){
Mithila <int> ob(10, 10);
Mithila <double> ob1(5.5, 6.5);
ob.area();
ob1.area();
}
```

Output



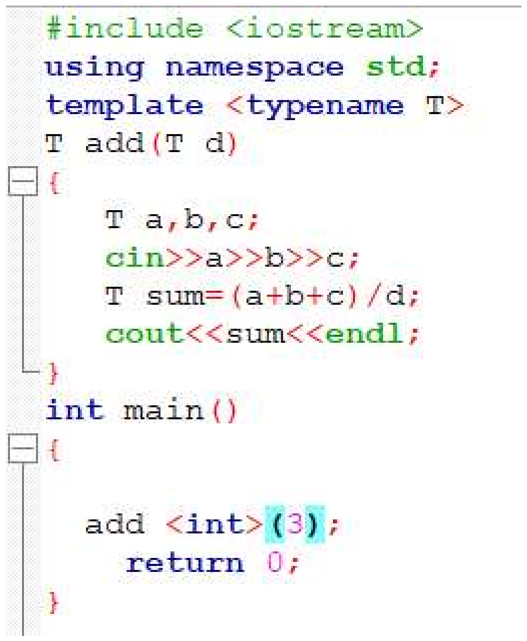
```
A*B =100
A*B =35.75

Process returned 0 (0x0)   execution time : 0.075 s
Press any key to continue.
```

6. Generic Function?

A generic function, often referred to as a template function in C++ and similar languages, is a function that is defined with one or more type parameters. These type parameters allow the function to work with different data types without the need for separate function implementations for each type. Generic functions enhance code reusability and flexibility.

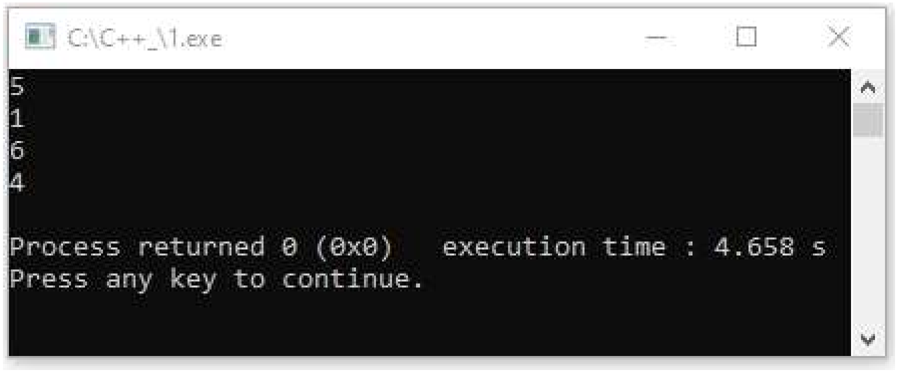
Example



```
#include <iostream>
using namespace std;
template <typename T>
T add(T d)
{
    T a,b,c;
    cin>>a>>b>>c;
    T sum=(a+b+c)/d;
    cout<<sum<<endl;
}
int main()
{

  add <int>(3);
   return 0;
}
```

Output



```
5
1
6
4

Process returned 0 (0x0)   execution time : 4.658 s
Press any key to continue.
```

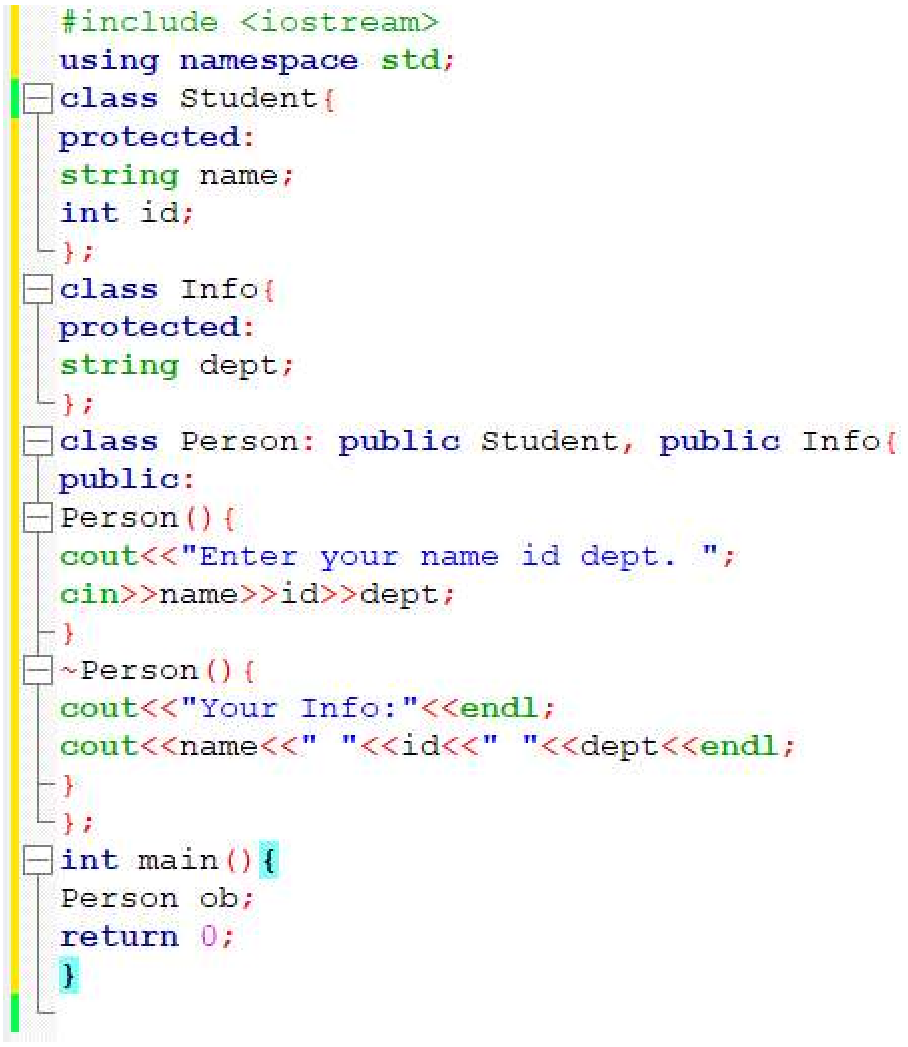
7.Defination Multiple Inheritence,Constructor,Destructor

Multiple Inheritence: Multiple inheritance allows a class to inherit from more than one class, acquiring the attributes and behaviors of each parent class. This can be a powerful mechanism for code reuse and building complex class hierarchies.

Constructor: A constructor in object-oriented programming is a special member function of a class that is automatically called when an object of the class is created. The primary purpose of a constructor is to initialize the object's data members or perform any other setup necessary for the object to be in a valid state.

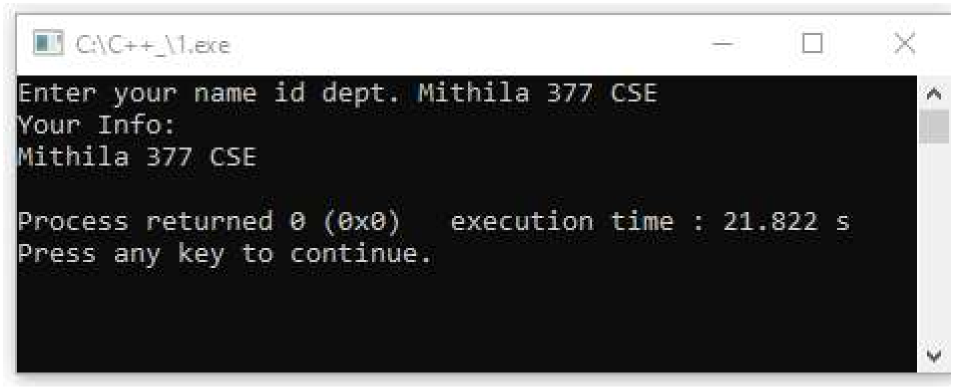
Destructor: In C++, a destructor is a special member function of a class that is called automatically when an object of that class is destroyed. Destructors are used to perform cleanup tasks, such as releasing memory or closing files.A destructor has the same name as its class, but it is preceded by a tilde (`~`). For example, the destructor for the class `MyClass` is named `~MyClass()`. Destructors do not take any arguments and do not return any values.

8.Constructor & Destructor With Multiple Inheritenc Example



```
#include <iostream>
using namespace std;
class Student{
protected:
string name;
int id;
};
class Info{
protected:
string dept;
};
class Person: public Student, public Info{
public:
Person(){
cout<<"Enter your name id dept. ";
cin>>name>>id>>dept;
}
~Person(){
cout<<"Your Info:"<<endl;
cout<<name<<" "<<id<<" "<<dept<<endl;
}
};
int main(){
Person ob;
return 0;
}
```

Output



C:\C++_\1.exe

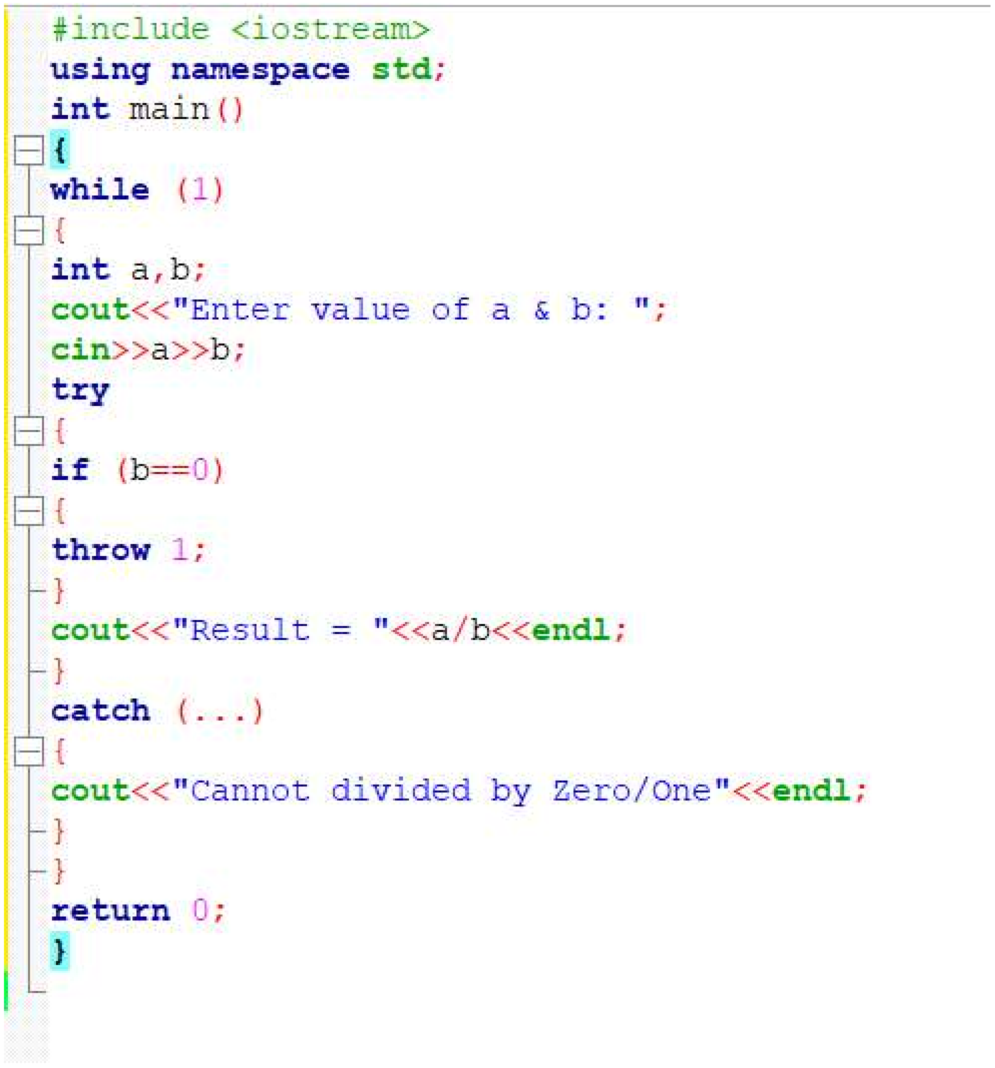
```
Enter your name id dept. Mithila 377 CSE
Your Info:
Mithila 377 CSE

Process returned 0 (0x0)   execution time : 21.822 s
Press any key to continue.
```

9.Excaption Handling

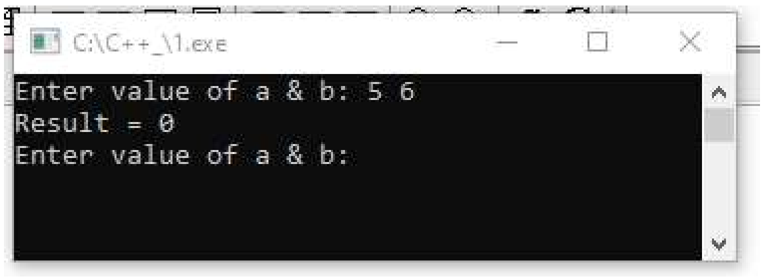
Exception handling is a programming construct that deals with the occurrence of exceptional or unexpected events during the execution of a program. These events, often referred to as exceptions, can include runtime errors, logical errors, or other exceptional conditions that may disrupt the normal flow of program execution.

Example



```
#include <iostream>
using namespace std;
int main()
{
while (1)
{
int a,b;
cout<<"Enter value of a & b: ";
cin>>a>>b;
try
{
if (b==0)
{
throw 1;
}
cout<<"Result = "<<a/b<<endl;
}
catch (...)
{
cout<<"Cannot divided by Zero/One"<<endl;
}
}
return 0;
}
```

Output

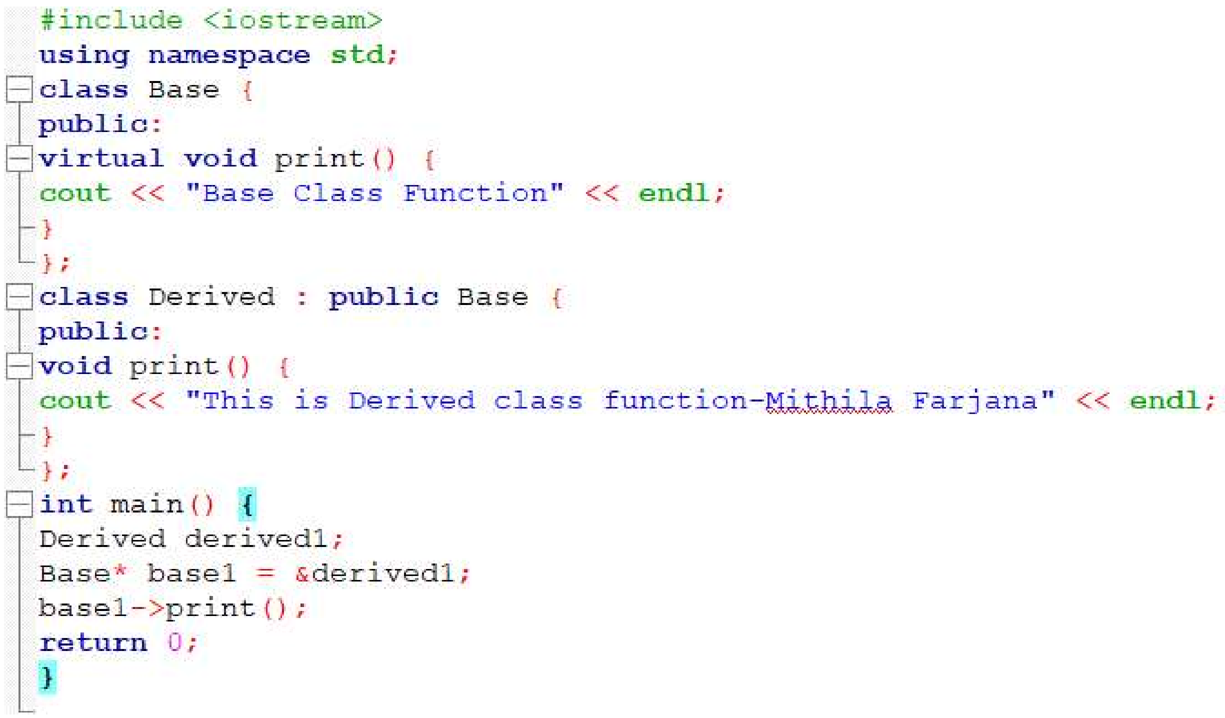


```
Enter value of a & b: 5 6
Result = 0
Enter value of a & b:
```

10.Virtual Function

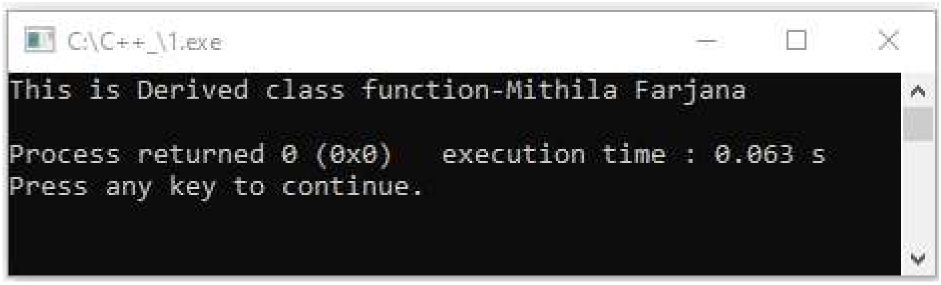
In C++, a virtual function is a function in a base class that is declared using the `virtual` keyword and can be overridden by a function with the same signature in a derived class. This allows for dynamic polymorphism, where the correct function to call is determined at runtime based on the actual type of the object.

Example



```
#include <iostream>
using namespace std;
class Base {
public:
virtual void print() {
cout << "Base Class Function" << endl;
}
};
class Derived : public Base {
public:
void print() {
cout << "This is Derived class function-Mithila Farjana" << endl;
}
};
int main() {
Derived derived1;
Base* base1 = &derived1;
base1->print();
return 0;
}
```

Output



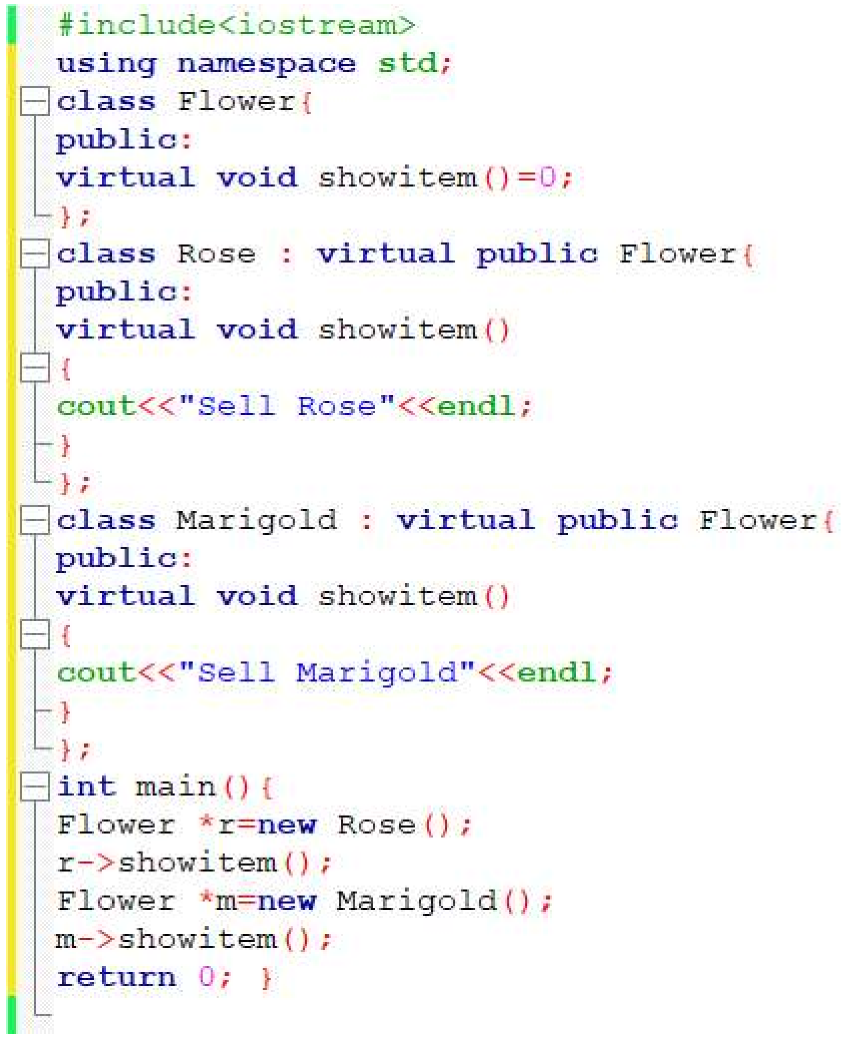
```
This is Derived class function-Mithila Farjana

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

11.Pure Virtual function:

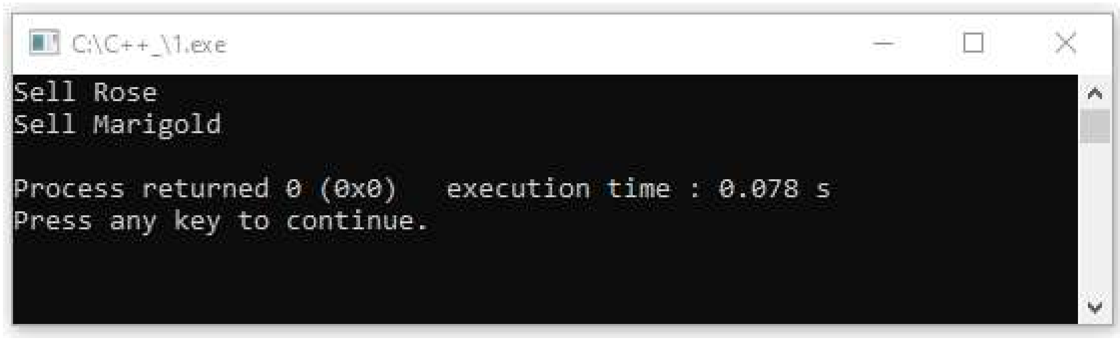
A pure virtual function in C++ is a virtual function in a base class that has no implementation in the base class and is marked with the = 0 syntax. A class containing at least one pure virtual function is considered an abstract class, and you cannot create an object of an abstract class. Instead, you must derive a concrete (non-abstract) class from the abstract class and provide implementations for all pure virtual functions.

Example



```
#include<iostream>
using namespace std;
class Flower{
public:
virtual void showitem()=0;
};
class Rose : virtual public Flower{
public:
virtual void showitem()
{
cout<<"Sell Rose"<<endl;
}
};
class Marigold : virtual public Flower{
public:
virtual void showitem()
{
cout<<"Sell Marigold"<<endl;
}
};
int main(){
Flower *r=new Rose();
r->showitem();
Flower *m=new Marigold();
m->showitem();
return 0; }
```

Output



```
Sell Rose
Sell Marigold

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```