OOP
Lecture 19

16/12/2021

Inheritance :

The capability of a class to derive properties and characteristics (inherit members) from another class is called Inheritance. Inheritance is one of the most important features of Object Oriented Programming. In inheritance, we extend the same type of the classes. Inheritance is used for the reusability of the code, and to avoid duplication of the code. Inheritance ensures **IS-A** relationship between two classes. Like, we can say that “Cat is an Animal” or “Car is a Vehicle”. We use inheritance only if an **IS-A** relationship is present between the two classes. Inheritance makes Parent-Child relationship between the classes. These are,

Sub Class or Child Class:

The class that inherits properties from another class is called Subclass or Derived Class or Child class.

Super Class or Parent Class:

The class whose properties are inherited by sub class is called Base Class or Superclass or Parent Class.

Syntax :

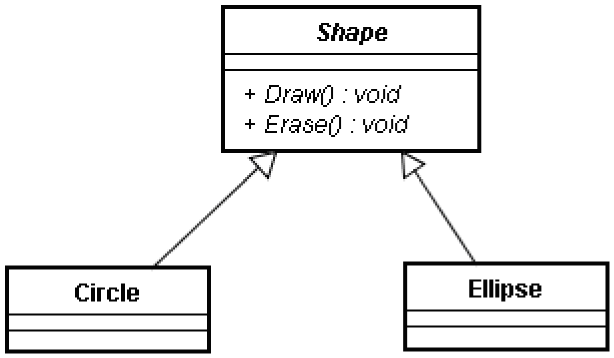
```
class Child_Class_name : access_mode Parent_Class_name
    {
                //body of Child Class
    };
```

Example :

```
class B : public A
    {
            //Body of Child Class
    };
```

UML Notation :

UML (Unified Modelling Language) Notation is used to represent the relationship between two classes graphically or theoretically. The inheritance relationship between 2 classes can be shown by empty head arrow pointing to the parent class.



Ways of Inheritance :

Just like aggregation, there are 2 approaches to do inheritance in Object Oriented Programming. These are,

1. **Top Bottom approach :** In this approach, we first define the parent class and then decide which classes are to be its child class. In other words, we first define our base class then we decide which classes can be derived from the base class.

Example :

We first define an Animal class with basic attributes like eat(), sleep() etc and then we write its child classes like Dog and Cat.

2. **Bottom Up approach :** In this approach, we first define the child classes and then we see what are the similarities between the child classes and then we combine those attributes into one unit and make our parent class.
   Example :
   We first define classes of different vehicles like Car, Truck, Bus etc then we find similarities and combine them in parent Vehicle class like, refuel(), applyBrakes() etc.

Any of these 2 approaches can be used when we are starting our program from scratch. If we are working on the existing classes then we don't have any choice, we have to use the Top Bottom approach, as we cannot change the existing class, we can only make childs of existing classes and derived attributes from them.

Access of Child and Parent Class :

The parent class is always unaware of its child class. It means that the parent is an independent class, it does not have any access to its child class and it does not depend on the members of its child class. While, the child class inherits all the characteristics (Members -> Data Members and Member Functions) of the parent class. So, The members of the parent class are also the members of the child class. But, there is a restriction on the access of the members of the parent class. If the members of the parent class are **private,** then the child class will be unable to access its members. Though, we can solve this problem by making the members of the parent class **public**, it raises the problem that those members of parent class can be accessed from main() function and from anywhere in the program. So, there is another access mode in OOP named as **protected**. If we define our members as protected members, then those members(data members and member functions) can be accessed from the child class only. Those members will be public for child classes and private for the rest of the program.

Access Types :

So, we have studied 3 access types till now,

1. **Public** members can be accessed anywhere in the program.

2. **Private** members can be accessed only within the current class and friend functions.

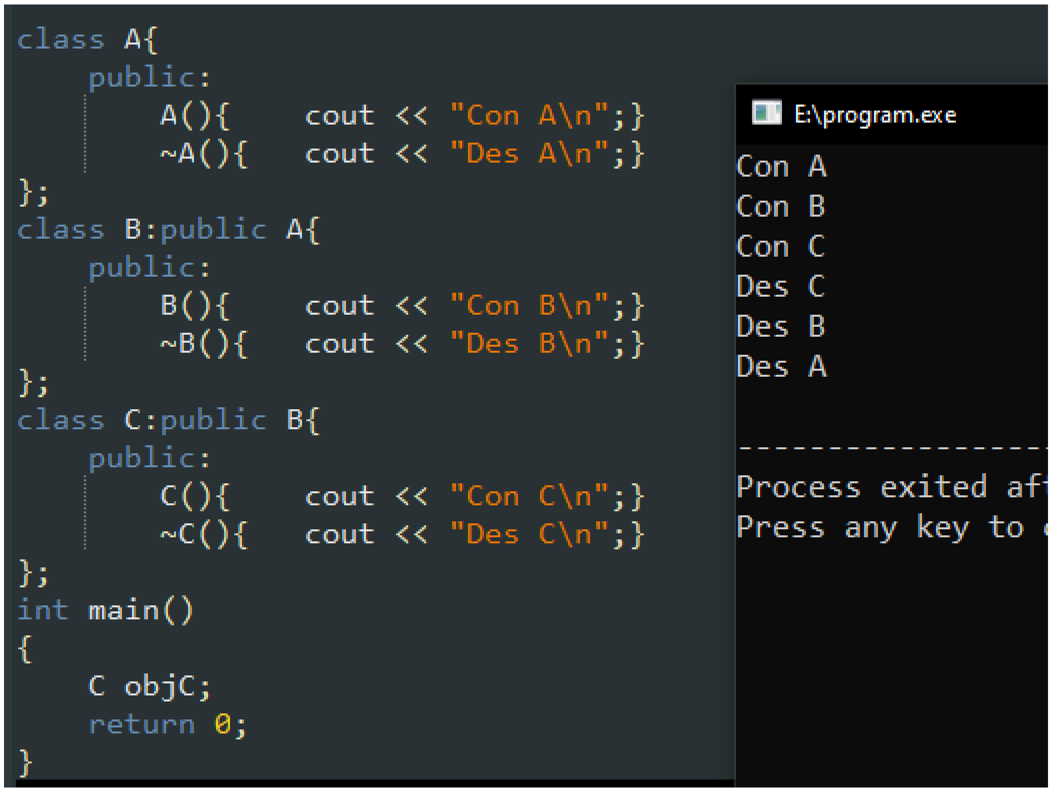
3. **Protected** members can be accessed within the current class and also within all of its Child classes.

Calling Members :

So, we can call the public and protected members of the parent class from the child class. The first and most important member function to be called is the Constructor of the parent class. If a Constructor (Parameterized or non-parameterized) exists in the parent class then it is necessary to write a constructor in the child class as well. When we create an object of child class then the constructor of the parent class is also called with the constructor of the child class. If multiple constructors are available in the parent class then we can decide which constructor to call by using : sign after the constructor of the child class. Next, we have member functions, any public or protected member function of the parent class can be called from the child class and any public function of the parent can also be called from the main() function by using the object of the child class. It means that the object of the child class can access the public member functions of the parent class just like it's own member functions. However, if a member function or data member exists in the child class with same name, a very typical example is show() function, which can occur in both parent and child classes, then the member function of child class will be called with its object, it is called redefining of the function, as we have redefined the function for the child class separately.

Now, Going towards the actual programs to demonstrate the concepts of Inheritance. Let’s go straight into it.

Sample Program :

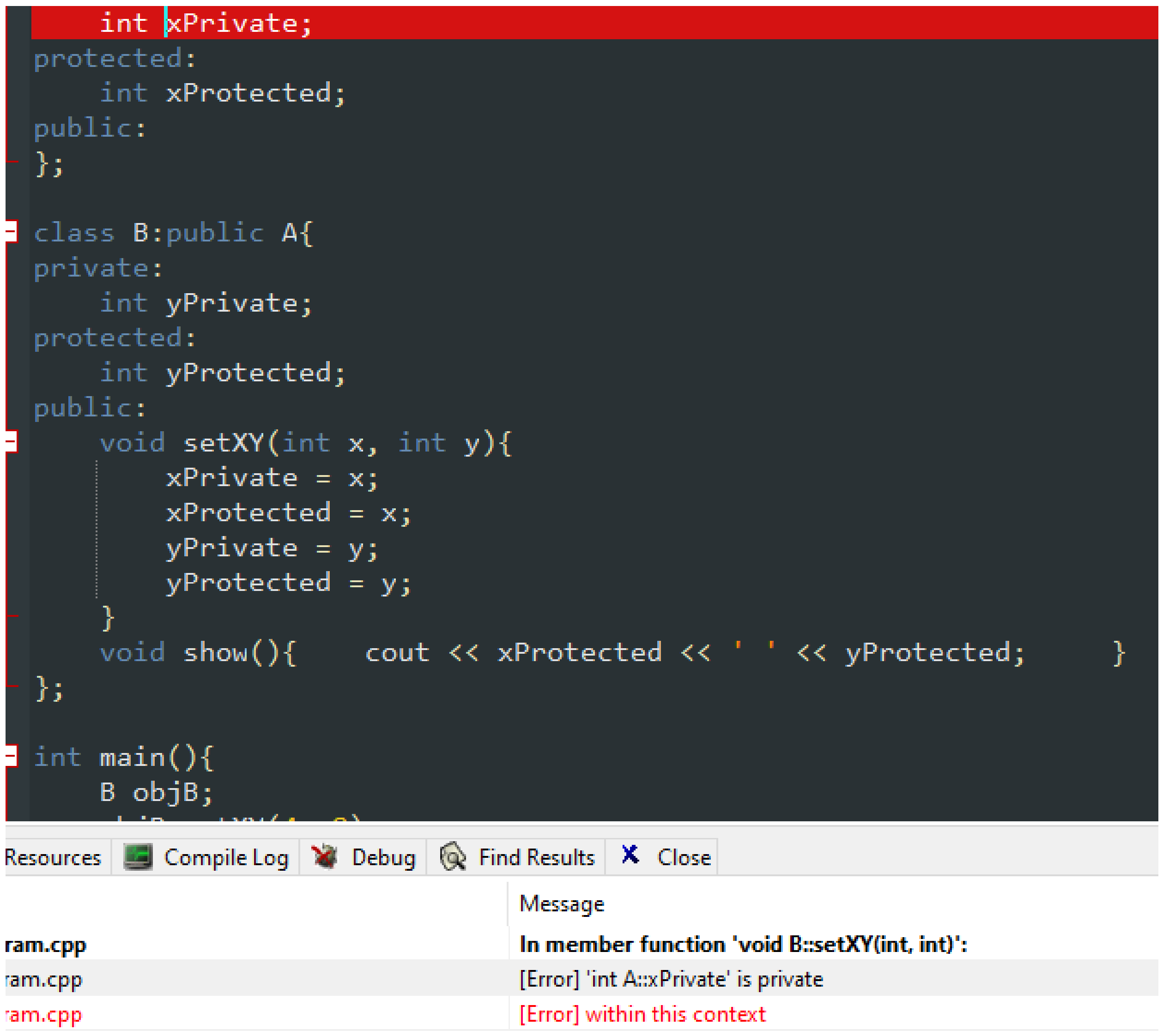


```
class A{
    public:
        A(){    cout << "Con A\n";}
        ~A(){   cout << "Des A\n";}
};
class B:public A{
    public:
        B(){    cout << "Con B\n";}
        ~B(){   cout << "Des B\n";}
};
class C:public B{
    public:
        C(){    cout << "Con C\n";}
        ~C(){   cout << "Des C\n";}
};
int main()
{
    C objC;
    return 0;
}
```

E:\program.exe

```
Con A
Con B
Con C
Des C
Des B
Des A

-----------------
Process exited af
Press any key to
```

- We made class B child of A and class C as a child of B, which means that B has the attributes and A and C has the attributes of both A and B.
- When we created an object of class C, it called the constructor of class A, then class and B and then the Constructor of class C, and when the program finished, the destructors are called in reverse order.
- Can you guess the output of the same program, if we have created 2 objects of class B in the same program instead of the class C?

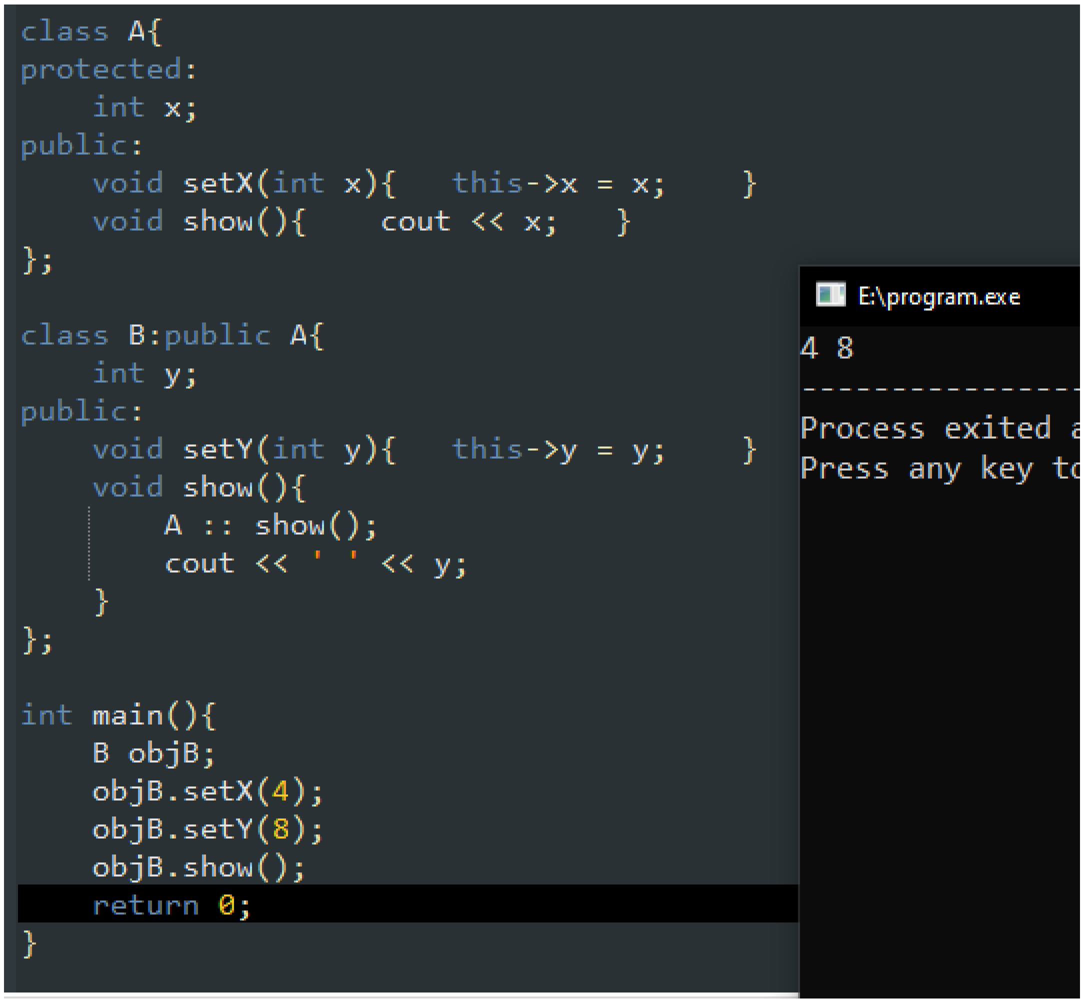


```
    int xPrivate;
protected:
    int xProtected;
public:
};

class B:public A{
private:
    int yPrivate;
protected:
    int yProtected;
public:
    void setXY(int x, int y){
        xPrivate = x;
        xProtected = x;
        yPrivate = y;
        yProtected = y;
    }
    void show(){    cout << xProtected << ' ' << yProtected;     }
};

int main(){
    B objB;
```

```
                                        Message
ram.cpp                                 In member function 'void B::setXY(int, int)':
ram.cpp                                 [Error] 'int A::xPrivate' is private
ram.cpp                                 [Error] within this context
```

Here, When we tried to access the private data members of the class A, it gave an error, however, we can access the protected members easily inside the child class. It is important to note that the protected members can only be accessed within the child class boundaries not from the main function.

- If you still have any confusion try to change the protected, public and private members of both of the classes to understand the access of data members from the child classes.



```
class A{
protected:
    int x;
public:
    void setX(int x){   this->x = x;     }
    void show(){     cout << x;    }
};

class B:public A{
    int y;
public:
    void setY(int y){   this->y = y;     }
    void show(){
        A :: show();
        cout << ' ' << y;
    }
};

int main(){
    B objB;
    objB.setX(4);
    objB.setY(8);
    objB.show();
    return 0;
}
```

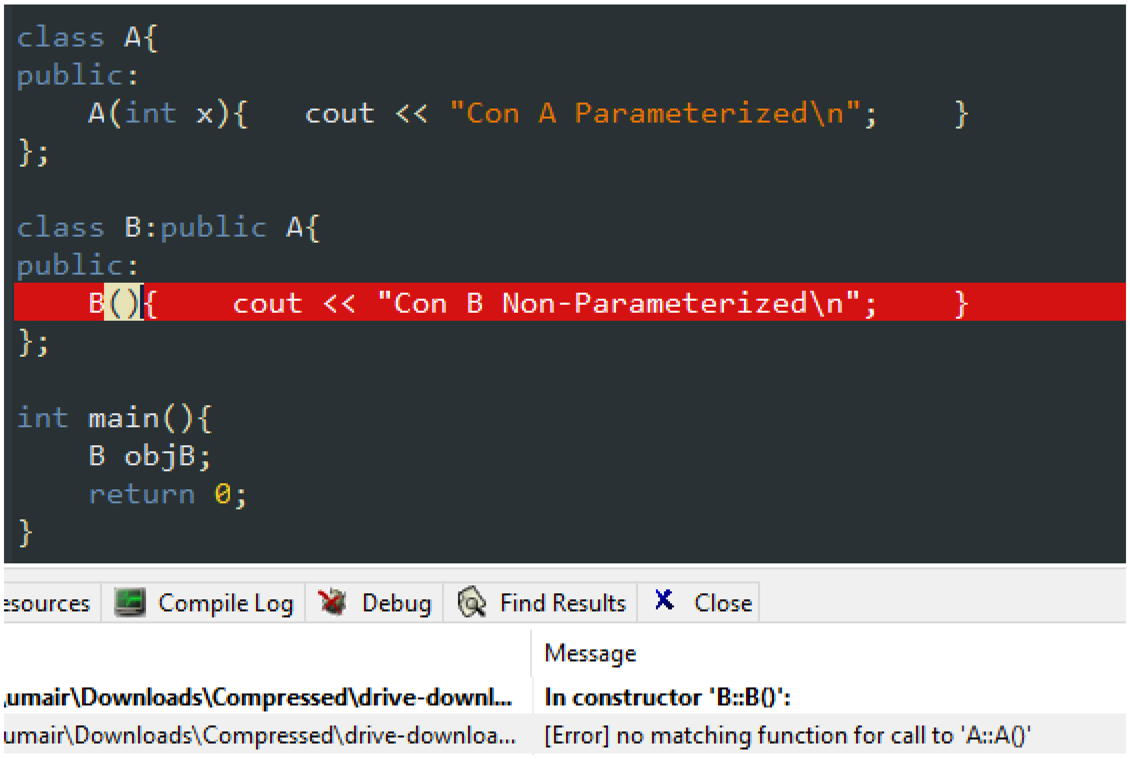
E:\program.exe

```
4 8
----------------
Process exited a
Press any key to
```

- Here, The first thing to check is that we can call setX and setY in the same way but they are defined in the different classes. As the class B is the child class of the class A, so we can access its public members from anywhere without any error.
- We defined the show function in both of the classes. When we called the show function from main() it called the show function of the class B, as we are calling with the object of B. Then, we called the show function of class A using A :: show(). We used scope resolution operator (::) to call the function of the specified class otherwise if we just wrote the show() here it will call itself again and wil stuck in an infinite recursion.
- If we have multiple classes, suppose a class C inheriting B. Class C does not contain any show function. So, when we call the show

function with C, it will call the show function of its nearest Parent. i.e the show() function of class B will be called.

Calling Parameterized Constructors :

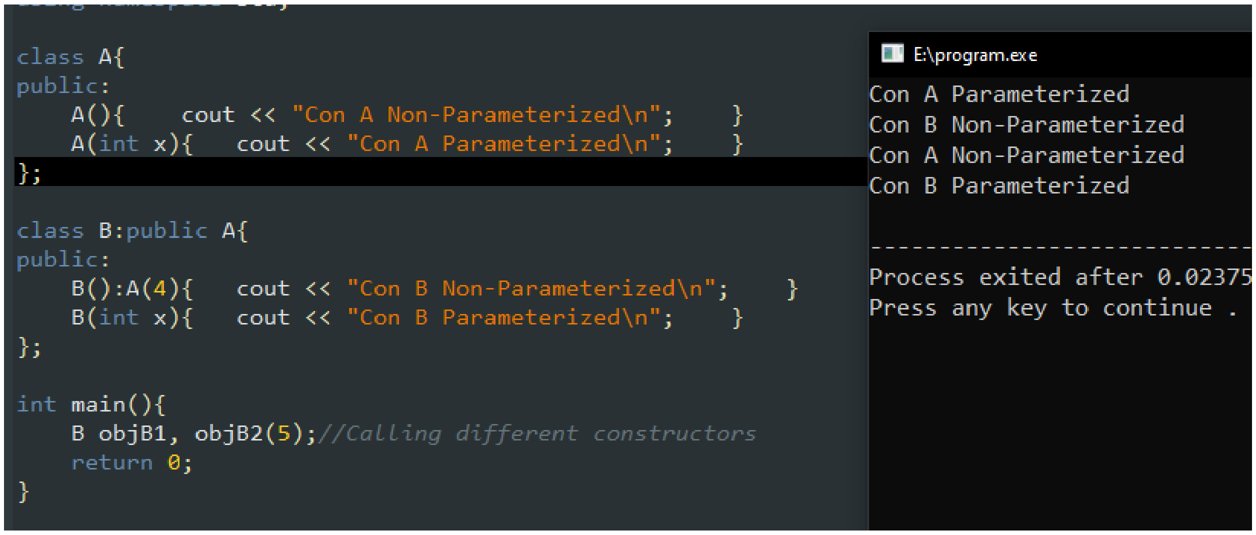


```
class A{
public:
    A(int x){    cout << "Con A Parameterized\n";     }
};

class B:public A{
public:
    B(){     cout << "Con B Non-Parameterized\n";     }
};

int main(){
    B objB;
    return 0;
}
```

```
umair\Downloads\Compressed\drive-downl...    In constructor 'B::B()':
umair\Downloads\Compressed\drive-downloa...  [Error] no matching function for call to 'A::A()'
```

Here, the class A has only parameterized constructor and class B has non-parameterized constructor, so we need to call the constructor of class A explicitly with the a parameter as well, otherwise it is giving error that no non-parameterized constructor exists in the class A.

If we made the constructor of class B parameterized, this error will still exist, because class A will still be called without any parameter. So, we need to pass the parameters explicitly in any case, if the non-parameterized of the parent class does not exist.



```
class A{
public:
    A(){    cout << "Con A Non-Parameterized\n";    }
    A(int x){   cout << "Con A Parameterized\n";    }
};

class B:public A{
public:
    B():A(4){   cout << "Con B Non-Parameterized\n";    }
    B(int x){   cout << "Con B Parameterized\n";    }
};

int main(){
    B objB1, objB2(5);//Calling different constructors
    return 0;
}
```

E:\program.exe

```
Con A Parameterized
Con B Non-Parameterized
Con A Non-Parameterized
Con B Parameterized

----------------------------
Process exited after 0.02375
Press any key to continue .
```

Here, when we created objB1 without parameters it goes to the non-parameterized constructor of the class B and before entering into the constructor body it calls the parameterized constructor of the class A with parameter 4. So, the first 2 lines will be printed.Then, we created objB2 without parameters it goes to the parameterized constructor of the class B and before entering into the constructor body it calls the non-parameterized constructor of the class A. So, the last 2 lines will be printed.

Class Activity :

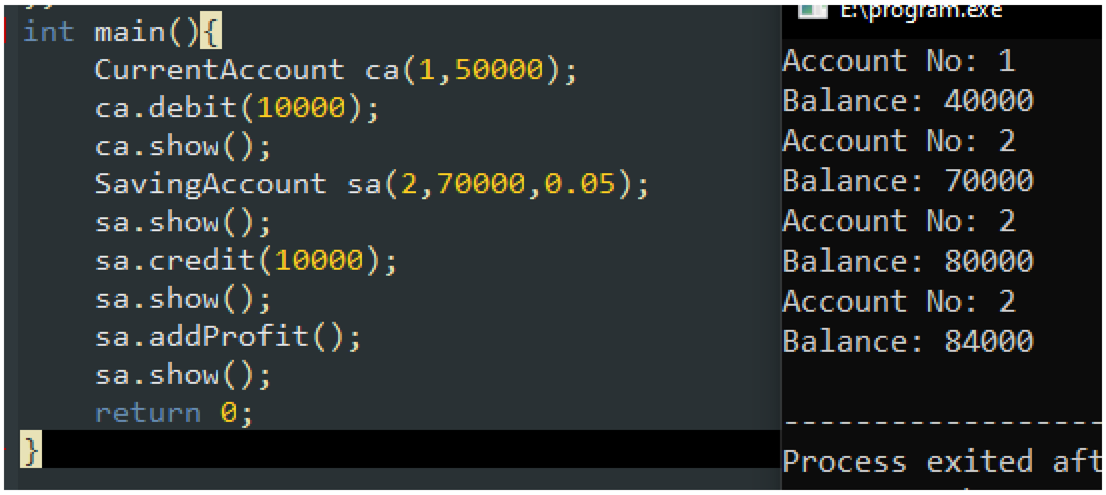
Write a class Accounts with data members : account number and balance and member functions credite(amount) and debit(amount). Then write two child classes: Current Account and Saving Account. Write member functions to add profit into the balance according to profit percentage.

```
class Account
{
        int accountNo;
protected:
        int balance;
```

```
public:
	Account(int accountNo, int balance)
		{
			this->accountNo = accountNo;
			this->balance = balance;
		}
	void debit(int amt)
		{
			balance-=amt;
		}
	void credit(int amt)
		{
			balance+=amt;
		}
	void show() const
		{
			cout << "Account No: " << accountNo <<'\n';
			cout << "Balance: " << balance <<'\n';
		}
};

class CurrentAccount:public Account
{
public:
	CurrentAccount(int accountNo, int balance)
	: Account(accountNo, balance) { }
};

class SavingAccount:public Account
{
	float profitPercentage;
public:
	SavingAccount(int accountNo, int balance, float
profitPercentage)
	: Account(accountNo, balance)
```

```
		{
			this->profitPercentage=profitPercentage;
		}
	void addProfit()
		{
			balance = balance + balance * profitPercentage;
		}
};
```

Output with main() function :



```
int main(){
	CurrentAccount ca(1,50000);
	ca.debit(10000);
	ca.show();
	SavingAccount sa(2,70000,0.05);
	sa.show();
	sa.credit(10000);
	sa.show();
	sa.addProfit();
	sa.show();
	return 0;
}
```

```
E:\program.exe
Account No: 1
Balance: 40000
Account No: 2
Balance: 70000
Account No: 2
Balance: 80000
Account No: 2
Balance: 84000


------------------
Process exited aft
```

- THE END -