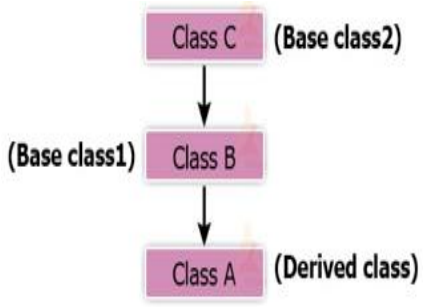
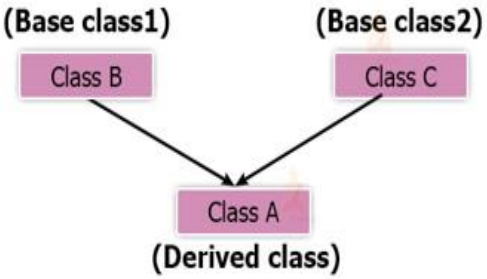
Order of execution of constructors in inheritance: For the multi-level inheritance as shown below, the constructor of class C is executed first, then the constructor of class B and finally of the class A. The order of execution is from the highest level base class to the other subsequent classes.



For the multiple-inheritance, the constructors of the base class are executed in the order of inheritance in which they are coded. That means, for the inheritance as coded below, the constructor of the class B is executed at first, then the constructor of class C, and finally of the derived class A.



Order of Inheritance:

```
Class A: public B, public C
{
   //body of class A
};
```

Destructors in inheritance:

Destructors are executed in the opposite order of that of constructors. That means, the constructor of the derived class is executed at first and then the constructors of its base classes in order.

An example that demonstrates the order of execution of destructors in base and derived class is as follows: [Example program- 37]

Example program- 37:

```
class Russia{
   public:
  ~Russia(){
     cout<<"I am destructor from Russia."<<endl;
    }
};
class Ukraine: public Russia{
   public:
  ~Ukraine(){
     cout<<"I am destructor from Ukraine."<<endl;
    }
};
```

Output: I am destructor from Ukraine.
I am destructor from Russia.

Few Program Tasks:

Example program- 38: Create a class called Person with suitable data members to represent their name and age. Use member functions to initialize and display these information. Derive Student and Employee from the Person class with their unique features. Initialize objects of these classes using constructor and display the information.

Solution:

```
#include <iostream>
using namespace std;

class Person
{
    string name;
    int age;
    public:
    void getData(string n, int a){
        name=n;
        age=a;
    }
    void displayData(){
        cout<<"Name: "<<name<<endl;
        cout<<"Age: "<<age<<endl;
    }
};
```

```
class Student: public Person
{
    int rollNos;
    public:
    Student(){  //default constructor

    }
    Student(string name, int age, int r){  //Parameterized constructor
        Person::getData(name,age); //calling member function of base class
        rollNos=r;
    }
    displayData(){
        Person::displayData();
        cout<<"RollNos: "<<rollNos<<endl;
    }
};
class Employee: public Person
{
    int id;
    public:
    Employee( ){     //default constructor

    }
    Employee(string name, int age, int i){   //Parameterized constructor
        Person::getData(name,age); //calling member function of base class
        id =i;
    }
    void displayData(){
        Person::displayData();
        cout<<"ID: "<<id<<endl;
    }
};
int main()
{
    Student s1("Neha", 20, 15);
    Employee e1("Aashish", 25, 101);

    cout<<"Displaying student info.."<<endl;
    s1.displayData();

    cout<<"Displaying employee info.."<<endl;
    e1.displayData();

    return 0;
}
```

Output:
Displaying student info..
Name: Neha
Age: 20
RollNos: 15
Displaying employee info..
Name: Aashish
Age: 25
ID: 101

Example program- 39: Create a class Person with data members Name, Age and Address. Create another class Teacher with data members Qualification and Department. Also create another class Student with data members Program and Semester. Both classes are inherited from the class Person. Every class has at least one constructor which uses base class constructor. Create a member function showData( ) in each to display the information of the class member. Create objects of the derived classes and display the information.

Solution:

```
class Person
{
    string name;
    int age;
    string address;

    public:
    Person(){  //Default constructor

    }

    Person(string n, int a, string b){
      name=n;
      age=a;
      address=b;
    }

    void showData(){
        cout<<"Name: "<<name<<endl;
        cout<<"Age: "<<age<<endl;
        cout<<"Address: "<<address<<endl;
    }
};
```

```
class Teacher: public Person
{
    string qualification;
    string department;

    public:
    Teacher( ){     //default constructor

    }

    Teacher(string name, int age, string add, string q, string d): Person(name, age, add)
    {
        qualification=q;
        department=d;
    }
    void displayData(){
        Person::showData();
        cout<<"Qualification: "<<qualification<<endl;
        cout<<"Department: "<<department<<endl;
    }
};
class Student: public Person
{
    string program;
    int semester;
    public:
    Student(){  //default constructor

    }
    Student(string name, int age, string add, string p, int s): Person(name, age, add)
    {
        program=p;
        semester=s;
    }
    displayData(){
        Person::showData();
        cout<<"Program: "<<program<<endl;
        cout<<"Semester: "<<semester<<endl;
    }
};
```

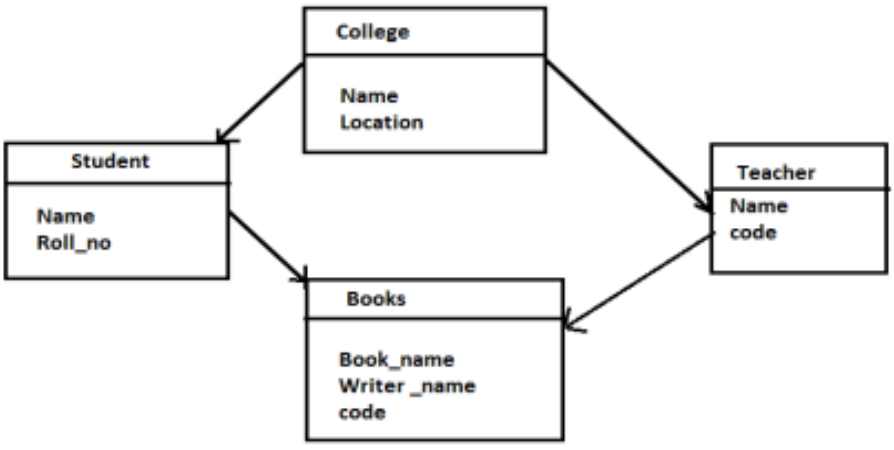
```
int main()
{
    Teacher t1("Manoz", 30, "Bharatpur", "Masters", "Computer Science");
    Student s1("Kiran", 20, "Bharatpur", "BE", 2);


    cout<<"Displaying Teacher info.."<<endl;
    t1.displayData();

    cout<<endl;
    cout<<"Displaying Student info.."<<endl;
    s1.displayData();

    return 0;
}
```

Output:

Displaying Teacher info..
Name: Manoz
Age: 30
Address: Bharatpur
Qualification: Masters

Department: Computer Science
Displaying Student info..
Name: Kiran
Age: 20
Address: Bharatpur
Program: BE
Semester: 2

Example program- 40: The following figure shows the minimum information required for each class. Write a program by realizing the necessary member functions to read and display information of individual object. Every class should contain at least one constructor and should be inherited to other classes as well.



Solution:

```
#include <iostream>
using namespace std;

class College
{
    string name;
    string location;

public:
    College(string a, string b)
    {
        name=a;
        location=b;
    }

    void displayCollegeInfo()
    {
        cout<<"College Name: "<<name<<endl;
        cout<<"Location: "<<location<<endl;
    }

};

class Student: virtual public College
{
    string name;
    int roll_no;

public:
    Student(string a, string b, string c, int d): College(a,b)
    {
        name=c;
        roll_no=d;
    }

    void displayStudentInfo()
    {
        cout<<"Student Name: "<<name<<endl;
        cout<<"Roll no: "<<roll_no<<endl;
    }

};
```

```
class Teacher: virtual public College
{
     string name;
     int code;

public:
     Teacher(string a, string b, string e, int f): College(a,b)
     {
          name=e;
          code=f;
     }
     void displayTeacherInfo()
     {
          cout<<"Teacher Name: "<<name<<endl;
          cout<<"Code: "<<code<<endl;
     }

};
class Books: public Student, public Teacher
{
    string book_name;
    string writer_name;
    int code;
public:
    Books(string a, string b, string c, int d, string e, int f, string g, string h, int i):
        College(a,b), Student(a,b,c,d), Teacher(a,b,e,f)
    {
        book_name=g;
        writer_name=h;
        code=i;
    }

    void displayInfo(){
        displayCollegeInfo();
        displayStudentInfo();
        displayTeacherInfo();
        cout<<"Book Name: "<<book_name<<endl;
        cout<<"Write Name: "<<writer_name<<endl;
        cout<<"Code: "<<code<<endl;
    }
};
int main()
{
    Books b1("Utech", "Bharatpur", "Kiran", 12, "Manoz", 105, "C++", "Robert", 1010);
    b1.displayInfo();

    return 0;
}
```

Output:

College Name: Utech
Location: Bharatpur
Student Name: Kiran
Roll no: 12
Teacher Name: Manoz
Code: 105
Book Name: C++
Write Name: Robert
Code: 1010

Pointer to object (Object Pointer): C++ allows us to have pointer to objects. The pointers pointing to objects are known as object pointers. These pointers are declared by putting (*) in front of an object pointer's name.

Syntax for object-pointer declaration:

```
Class_name *object-pointer-name;
```

For example: Student *ptr;
Here, 'Student' is an already defined class-name. And 'ptr' is an object pointer that points to an object of type 'Student' class.

Example for object-pointer initialization:

```
Student s1;          //creating object 's1' of class 'Student'
Student *ptr = &s1;  // object pointer 'ptr' pointing to object 's1'
```

- The object pointer can be used to access the members of a class. The arrow operator (->) is used for this purpose instead of dot(.) operator.

For example:

```
ptr -> display( );
```

It calls the 'display' member function of class 'Student'.

To use the dot operator (.), we have to dereference the object pointer.
For example: (*ptr).display( );

An example program that demonstrates the use of object pointer is given below.
[Example program- 41]

Example program- 41:

```
class Russia {
  public:
   void declareWar(){
    cout<<"Attack on Ukraine."<<endl;
   }
   void goAhead(){
    cout<<"We are coming."<<endl;
   }
};
int main(){
  Russia r;  //Creating r object
  Russia *ptr; //Declaring an object pointer
  ptr = &r; //Initialing the object pointer
  ptr->declareWar(); //Calling member function via object pointer
 (*ptr).goAhead(); //Alternative way to call member function
                   //from object pointer
  return 0;
}
```

Output: Attack on Ukraine.
We are coming.

this pointer: A special pointer that points to the current object of the class with the help of 'this' keyword is known as 'this pointer'. The 'this' pointer is passed as a hidden argument to all the non-static member function calls. It is available as a local variable within their body and may be used to refer to the invoking object. It is not available in static member functions and friend functions.

Examples of use cases:

1. It can be used to know the address of the current object inside the called member function. *[Example program- 42]*
2. It can be used for name conflict resolution. That means, when the name of the local variable and data members is the same, 'this' pointer is used to distinguish between them. *[Example program- 43]*
3. It is used to return the object or reference of the object it is pointing to. *[An example of this was demonstrated in the class. Try it again by yourself.]*

Example program- 42:

```
class A{
 public:
     void fun(){
    //Here, 'this' pointer points to current object,i.e. obj1
      cout<<"Address of current object: "<<this<<endl;
     }
};

int main(){
  A obj1;
  obj1.fun(); //It implicitly sends 'this' pointer as hidden
              //argument to the fun() function.
  return 0;
}
```

Output: Address of current object: 0x61fe1f

Example program- 43:

```
class A{
 int x,y;    //data members
 public:

 //names of local variables are same as that of data members
   void getData(int x, int y){
     /* x=x;   (name conflicts and so, values in local
        y=y;  variables cannot be assigned to the private
              members in such way) */

     this->x=x;     // 'this' pointer solves this problem
     this->y=y;
   }
   void display(){
    cout<<"The value of x: "<<x<<endl;
    cout<<"The value of y: "<<y<<endl;
   }
};

int main(){
  A obj1;
  obj1.getData(5,7); /*It implicitly sends 'this'
                     pointer as hidden argument to the
                     getData() member function. */
  obj1.display();
  return 0;
}
```

Output: The value of x: 5
The value of y: 7