

---

# Chapter Table of Contents

## Chapter 3.2

### Inheritance

Aim.....	175
Instructional Objectives.....	175
Learning Outcomes.....	175
3.2.1 Introduction.....	176
3.2.2 Inheritance .....	176
(i) Base Class and Derived Class.....	176
(ii) Defining Derived Classes .....	176
(iii) Protected Access Specifier .....	178
(iv) Public and Private Inheritance .....	179
(v) Member Accessibility.....	180
(vi) Constructors and Destructors in Derived Classes .....	182
Self-assessment Questions.....	185
3.2.3 Level of Inheritance .....	186
(i) Single Inheritance.....	187
(ii) Multiple Inheritance .....	188
(iii) Multi-level Inheritance.....	189
(iv) Hierarchical Inheritance.....	190
(v) Hybrid Inheritance.....	191
Self-assessment Questions.....	193
Summary .....	194
Terminal Questions.....	194
Answer Keys.....	195
Activity.....	195
Case Study .....	196
Bibliography.....	197
e-References .....	197
External Resources .....	197
Video Links .....	197

---





## **Aim**

To provide the students with a basic knowledge of inheritance in C++ programming



## **Instructional Objectives**

After completing this chapter, you should be able to:

- Describe the concept of inheritance with base class and derived class
- Describe Constructors and destructors in derived classes
- Explain Single, hierarchical and Hybrid inheritance
- Compare Multiple and multi-level inheritance



## **Learning Outcomes**

At the end of this chapter, you are expected to:

- Explain the concept of inheritance in C++ programming
- List advantages of inheritance
- Discuss different levels of inheritance
- Implement constructors and destructors in inheritance

---

### 3.2.1 Introduction

It is always productive if we are able to reuse something that already exists rather than creating the same all over again. This concept is called Reusability. This can be achieved by creating new classes, reusing the properties of existing classes, which is called Inheritance. This chapter deals with introduction to the concept of Inheritance, types of inheritance and their C++ implementations.

The mechanism of deriving a new class from existing or an old class is called “inheritance”. Inheritance is a technique of code reuse. It also allows us to extend existing classes by creating derived classes. The idea of inheritance implements the “is a” relationship.

### 3.2.2 Inheritance

Inheritance is the concept where one class inherits all the properties of base class in addition to its own details or properties. The mechanism of deriving a new class from an old one is called Inheritance.

Inheritance follows the parent child relationships. The base class is called as a parent class and a new class which is derived from base class is known as child class. Child class will always inherit all the features of the base class and is free to add additional features to it. **For example**, the real life example of inheritance is child and parents. The properties of parents are inherited by their child.

#### (i) Base Class and Derived Class

The old class is referred as a base class and the new one is called the derived class or sub class. The derived class inherits some or all of the properties from the base class. A class can also inherit properties from more than one class or from more than one level.

#### (ii) Defining Derived Classes

A derived class can be defined by specifying its relationship with the base class in addition to its own details.

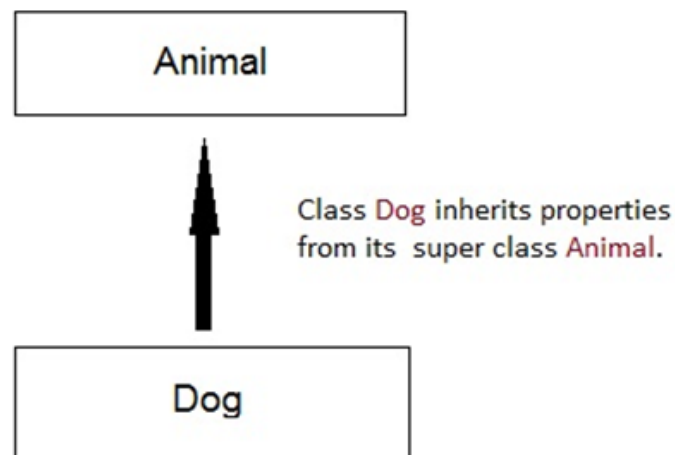
**The general form of defining a derived class is:**

```
Class derived class-name: access /visibility-mode base-class-name
{
    ..... //
```

---

```
.....// members of derived class
.....//
};
```

The colon indicates that the derived class-name is derived from the base-class-name. The visibility mode is optional, if present, may be either private or public.



*Figure 3.2.1: Example of Inheritance*

In the above figure, Animal is the base class and Dog is the derived class of Animal which inherits all the properties of Animal.

The access/visibility mode is optional, if present, may be either private or public. The default visibility mode is private. Visibility mode specifies whether the functions of base class are privately derived or publicly derived.

***For example,***

---

```
1  #include <iostream>
2  using namespace std;
3
4  Class class1 : private class2    //private derivation
5  {
6      //Members of class1
7  };
8  Class class1 : public class2    // public derivation
9  {
10     //Members of class1
11 };
12 Class class1 : class2          // private derivation by default
13 {
14     //Members of class1
15 };
```

---

---

### (iii) Protected Access Specifier

C++ provides a third visibility modifier, protected which serves a limited purpose in inheritance and it is similar to private. Protected members are accessible in the class and derived classes. A member declared as protected is accessible by the member functions within its class and any class immediately derived from it. It cannot be accessed by the function outside these two classes. A class can now use all the three visibility modes as illustrated below.

```
1  #include <iostream>
2  using namespace std;
3
4  class student
5  {
6  protected:           // protected access specifier
7      int age;           // Data Member Declaration
8      void display();    // Member Function declaration
9  }
```

When a protected member is inherited in public mode, it becomes protected in the derived class too. Hence, this becomes accessible by the member functions of the derived class. It is also ready for further inheritance.

A protected member, when inherited in the private mode, becomes private in the derived class. It is not available for further inheritance even though it is available to the member functions of the derived class.

---

The Figure 3.2.2 given below shows the pictorial representation for the two levels of derivation.

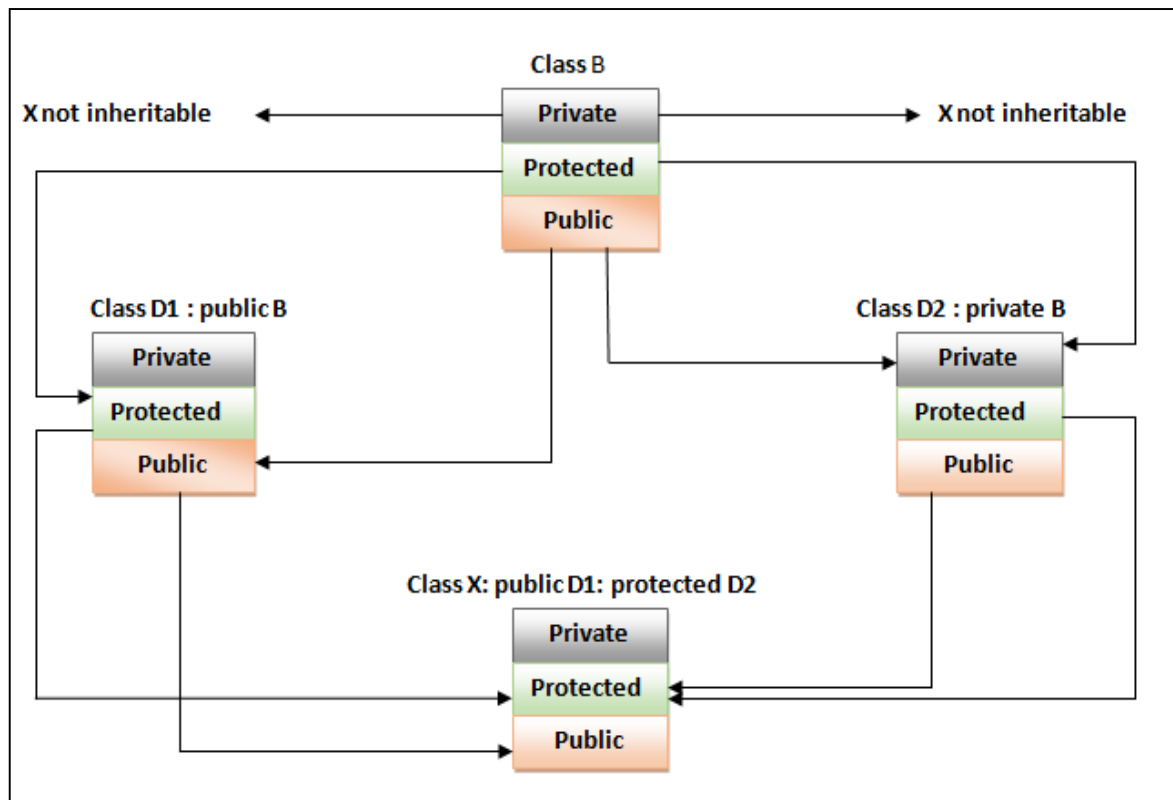


Figure 3.2.2: Effect of Inheritance on the Visibility of Members

#### (iv) Public and Private Inheritance

In Public visibility mode, all the public class members will be available and accessible to everyone.

The data members and member functions declared as public can be accessed by other classes too. Thus if you want data members in your class to be used by other classes then you should declare them as public.

```
1 #include <iostream>
2 using namespace std;
3 class student
4 {
5     public:          // public access specifier
6         int USN;      // Data Member Declaration which is public to all
7         void display(); // Member Function declaration
8 }
```

---

Table 3.2.1: Visibility of Inherited Members

Base Class Visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
<b>Private</b>	Not Inherited	Not Inherited	Not Inherited
<b>Protected</b>	Protected	Private	Protected
<b>Public</b>	Public	Private	Protected

In private visibility mode, no one can access the private class members outside that class. Thus the data remains protected from unauthorised classes. If we try to access the private member, we will get a compile time error.

By default class variables and member functions are private.

```
1 #include <iostream>
2 using namespace std;
3 class student
4 {
5     private:           // private access specifier
6         int salary;      // Data Member Declaration
7         void display();  // Member Function declaration
8 }
```

## (v) Member Accessibility

Each class may contain many data members which can be declared as private, public or protected. Hence we should know how to access the members defined in different classes. Members can be accessed depending on whether a class is privately derived or publicly derived. Hence we can solve many problems by using members of various classes. There are so many ways or possibilities for accessing members. It is necessary to look at an example program that shows how we must access the members, also what works and what doesn't.



---

```

1  #include <iostream>
2  using namespace std;
3  class X //base class
4  {
5      private:
6          int privdata A; //(functions have the same access
7      protected:
8          int protdata A;
9      public:
10         int pubdata A;
11     };
12     class Y : public X //publicly-derived class
13     {
14         public:
15             void funct()
16             {
17                 int a;
18                 a = privdata A; //error: not accessible
19                 a= protdata A; //OK
20                 a= pubdata A; //OK
21             }
22     };
23     class Z : private X//privately-derived class
24     {
25         public:
26             void funct()
27             {
28                 int a;
29                 a = privdata A; //error: not accessible
30                 a = protdata A; //OK
31                 a = pubdata A; //OK
32             }
33     };
34     int main()
35     {
36         int a;
37         Y obj1;
38         a = obj1.privdata A; //error: not accessible
39         a = obj1.protdata A; //error: not accessible
40         a = obj1.pubdata A; //OK (A public to B)
41         Z obj2;
42         a = obj2.privdata A; //error: not accessible
43         a = obj2.protdata A; //error: not accessible
44         a = obj2.pubdata A; //error: not accessible (Base private to Derived2)
45         return 0;
46     }

```

---

The program specifies a base class, Base, with private, protected and public data items. Two classes, Derived1 and Derived2, are derived from Base. Derived1 is publicly derived and Derived2 is privately derived. As we've seen before, functions in the derived classes can access

---

protected and public data in the base class. Objects of the derived classes cannot access private or protected members of the base class.

The difference between publicly derived and privately derived classes is Objects of the publicly derived class Derived1 can access public members of the base class Base, while objects of the privately derived class Derived2 cannot, they can only access the public members of their own derived class. If you do not supply any access specifier when creating a class, private is assumed.

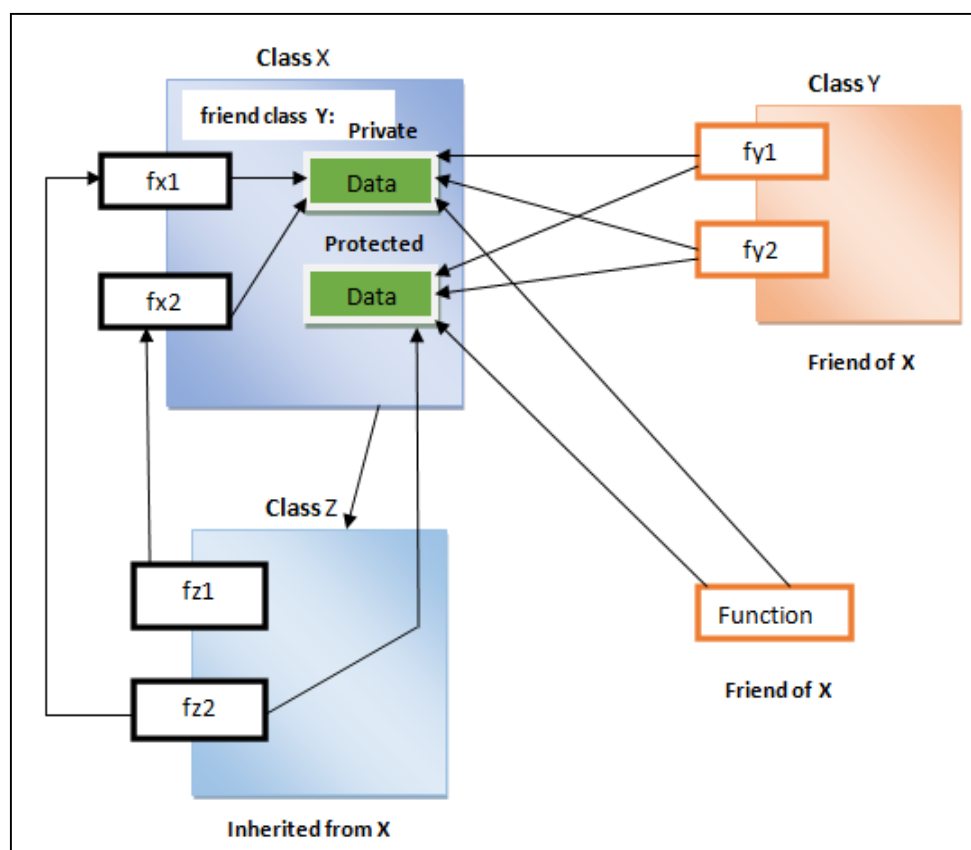


Figure 3.2.3: Access Mechanism in Classes

## (vi) Constructors and Destructors in Derived Classes

In C++, the constructors play an important role in initializing objects. In C++, Base class constructors are always called in the derived class constructors. Whenever you create derived class object, first the base class default constructor is executed and then the derived class's constructor is executed. An important point to note is that as long as no base class constructor takes any arguments, the derived class need not have a constructor function. However, if any base class contains a constructor with one or more arguments to the base class constructors. Thus, it makes sense for the derived class to pass arguments to the base

---

class constructor. When both the derived and base classes contain constructors, the base constructor is executed first and then the constructor in the derived class is executed.

**The general form of defining a derived constructor is,**

```

Derived_constructor      (Arglist1, Arglist2,.....,ArglistN, Arglist(D)
    base1(argumentlist1),
    base2(argumentlist2),
    .....
    .....
    .....
    baseN((argumentlistN),
    {
        Body of derived constructor
    }
  
```

Method of Inheritance	Order of Execution
Class B: Public A { };	A(); base constructor B(); derived constructor
Class A: Public B, Public C	B(); base (first) C(); base(second) A();derived constructor

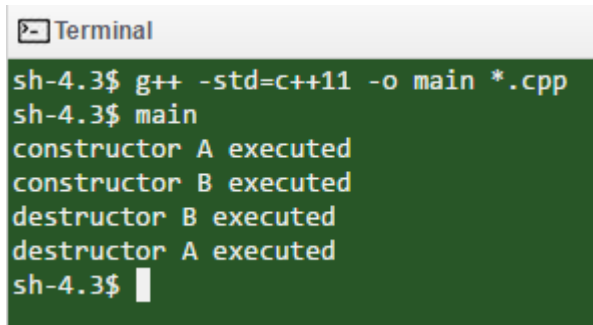
*Table 3.2.2: Execution of Base Class Constructor*

In inheritance, destructors are executed in reverse order of constructor execution. The destructors are executed when an object goes out of scope. The following program illustrates the constructors and destructors in inheritance.

---

```
1 #include <iostream>
2 using namespace std;
3 class A
4 {
5     public:
6     A ( ) //Constructor A
7     {
8         cout << "constructor A executed" << endl;
9     }
10    ~A ( ) //Destructor A
11    {
12        cout << "destructor A executed" << endl;
13    }
14 };
15 class B : public A
16 {
17     public:
18     B ( )
19     {
20         cout << "constructor B executed" << endl;
21     }
22     ~B ( )
23     {
24         cout << "destructor B executed" << endl;
25     }
26 };
27 int main( )
28 {
29     B b;
30     return 0;
31 }
```

**Output:**



```
Terminal
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
constructor A executed
constructor B executed
destructor B executed
destructor A executed
sh-4.3$
```



## Self-assessment Questions

- 1) The technique of creating a new class from an existing class is called \_\_\_\_\_.
- 2) A derived class inherits the constructors as well as destructors.
  - a) True
  - b) False
- 3) If class B uses the properties or features of class A, then B is called the \_\_\_\_\_ class and A is called the \_\_\_\_\_.
  - a) Derived, Base
  - b) Base, Derived
  - c) Derived, Derived
  - d) Base, Base
- 4) The visibility mode can be \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.
- 5) The object of a derived class cannot be assigned to object of its base class.
  - a) True
  - b) False

---

### 3.2.3 Level of Inheritance

The different levels of inheritance are discussed in detail below. A derived class with only one base class is called single inheritance and one with several base classes is called multiple inheritance. The properties of one class may be inherited by more than one class. This process is known as hierarchical inheritance. The mechanism of deriving a class from another derived class is known as multilevel inheritance. The figure 3.2.4 given below shows various forms of inheritance. The direction of arrow indicates the direction of inheritance.

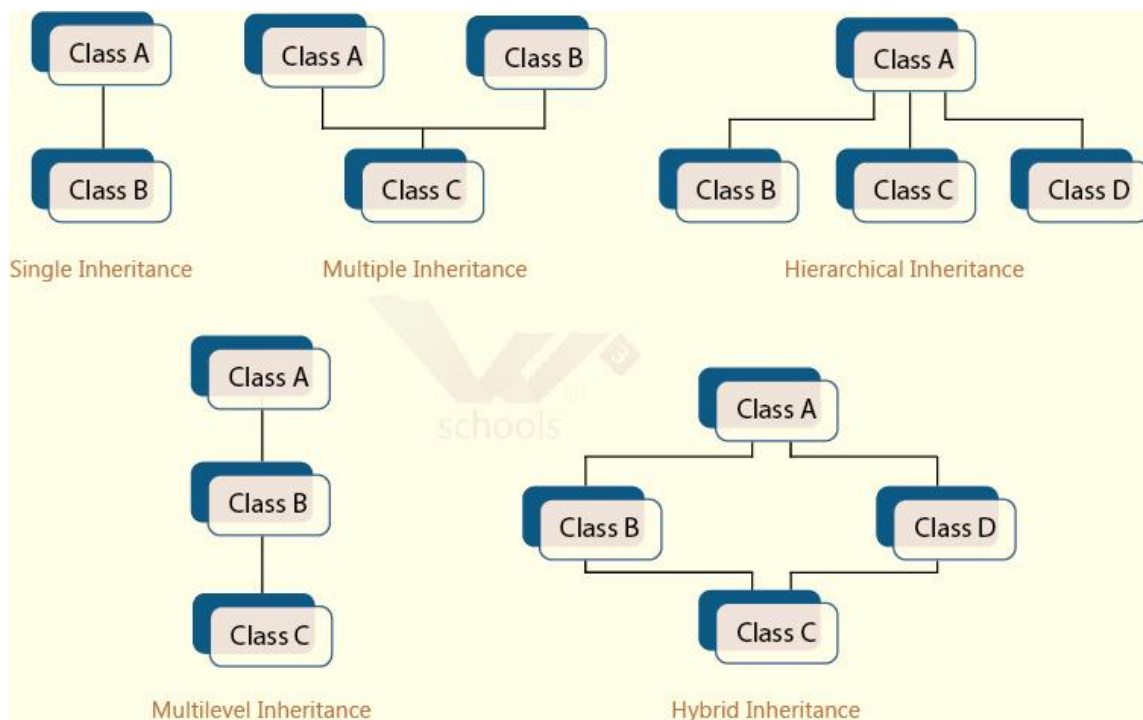


Figure 3.2.4: Forms of Inheritance

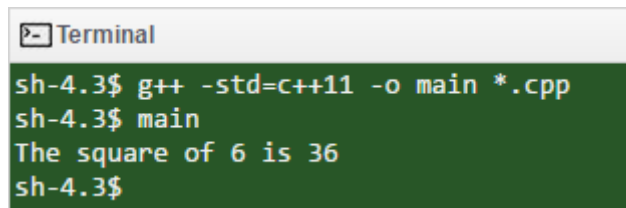
---

## (i) Single Inheritance

A derived class with only one base class is called single inheritance. Below program illustrates the concept of single inheritance. Here the derived class square has only one base class Val.

```
1  #include <iostream>
2  using namespace std;
3  class Val
4  {
5      protected:
6          int v;
7      public:
8          void set_values (int x)
9          {
10             v=x;
11         }
12     };
13     class Square:public Val
14     {
15     public:
16         int square()
17         {
18             return (v*v);
19         }
20     };
21     int main ()
22     {
23         Square S;
24         S.set_values (6);
25         cout << "The square of 6 is " <<S.square() << endl;
26         return 0;
27     }
```

**Output:**



```
Terminal
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
The square of 6 is 36
sh-4.3$
```

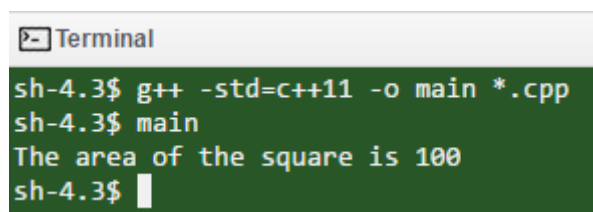
---

## (ii) Multiple Inheritance

A derived class with several base classes is called multiple inheritance. The below program illustrates the concept of multiple inheritance.

```
1  #include <iostream>
2  using namespace std;
3  class Square
4  {
5      protected:
6          int l;
7      public:
8          void set_values (int a)
9          {
10             l=a;
11         }
12 };
13 class Show1
14 {
15     public:
16         void show(int i);
17 };
18 void Show1::show (int i)
19 {
20     cout << "The area of the square is " << i << endl;
21 }
22 class Area: public Square, public Show1
23 {
24     public:
25         int area()
26         {
27             return (l * l);
28         }
29 };
30 int main ()
31 {
32     Area a;
33     a.set_values (10);
34     a.show(a.area());
35     return 0;
36 }
```

**Output:**



```
Terminal
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
The area of the square is 100
sh-4.3$
```



---

### (iii) Multi-level Inheritance

The mechanism of deriving a class from another derived class is known as multilevel inheritance. Below program illustrates the concept of Multi-level inheritance.

```
1  #include <iostream>
2  using namespace std;
3  class student
4  {
5      protected:
6          int USN;
7      public:
8          void get_num(int x)
9          {
10             USN = x;
11         }
12         void put_num()
13         {
14             cout << "USN Is:" << USN << "\n";
15         }
16     };
17     class marks : public student
18     {
19         protected:
20             int s1;
21             int s2;
22         public:
23             void get_marks(int a,int b)
24             {
25                 s1 = a;
26                 s2 = b;
27             }
28         void put_marks(void)
29         {
30             cout << "Subject 1:" << s1 << "\n";
31             cout << "Subject 2:" << s2 << "\n";
32         }
33     };
34     class result : public marks
35     {
36         protected:
37             float total;
38         public:
39             void disp(void)
```

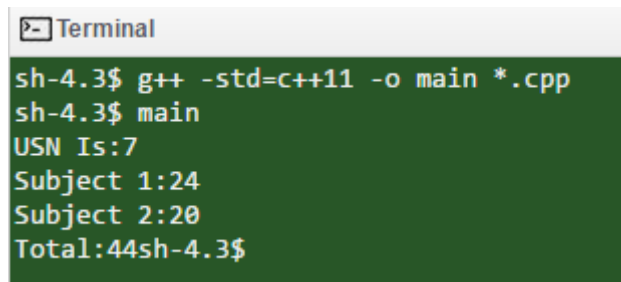
---

```

40 ▾      {
41          total = s1+s2;
42          put_num();
43          put_marks();
44          cout << "Total:"<< total;
45      }
46  };
47  int main()
48  ▾  {
49      result student1;
50      student1.get_num(07);
51      student1.get_marks(24,20);
52      student1.disp();
53      return 0;
54  }

```

**Output:**



```

Terminal
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
USN Is:7
Subject 1:24
Subject 2:20
Total:44sh-4.3$

```

In the above example, the derived function "result" uses the function "put\_num()" from another derived class "marks".

## (iv) Hierarchical Inheritance

The properties of one class may be inherited by more than one class. This process is known as hierarchical inheritance.

```

1  #include <iostream>
2  using namespace std;
3  class S
4  ▾  {
5      protected:
6          int l;
7      public:
8          void set_values (int a)
9  ▾      {
10         l=a;
11     }
12 };
13 class Square: public S
14 ▾  {
15     public:
16         int sq()

```

---

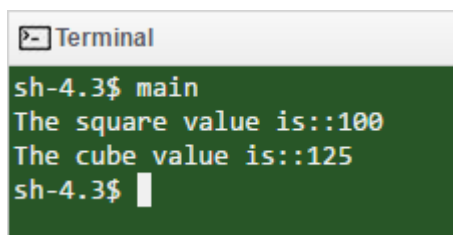
---

```

17 {
18     return (1 *1);
19 }
20 };
21 class Cube:public S
22 {
23     public:
24     int cub()
25     {
26         return (1 *1*1);
27     }
28 };
29 int main ()
30 {
31     Square s1;
32     s1.set_values (10);
33     cout << "The square value is::" << s1.sq() << endl;
34     Cube c1;
35     c1.set_values (5);
36     cout << "The cube value is::" << c1.cub() << endl;
37     return 0;
38 }

```

**Output:**



```

Terminal
sh-4.3$ main
The square value is::100
The cube value is::125
sh-4.3$

```

## (v) Hybrid Inheritance

Hybrid inheritance is the one where one or more types of inheritance are combined together and used. Below program illustrates the concept of hybrid inheritance.

---

```

1  #include <iostream>
2  using namespace std;
3  class student
4  {
5      protected:
6          int USN;
7      public:
8          void get_num(int x)
9          {
10             USN = x;
11         }
12         void put_num()

```

---

---

```

13     {
14         cout << "USN Is:"<< USN << "\n";
15     }
16 };
17 class marks : public student
18 {
19     protected:
20         int s1;
21         int s2;
22     public:
23     void get_marks(int a,int b)
24     {
25         s1 = a;
26         s2 = b;
27     }
28     void put_marks(void)
29     {
30         cout << "Subject 1:" << s1 << "\n";
31         cout << "Subject 2:" << s2 << "\n";
32     }
33 };
34 class xtra
35 {
36     protected:
37         float z;
38     public:
39     void get_xtra(float s)
40     {
41         z=s;
42     }
43     void put_xtra(void)
44     {
45         cout << "Extra Score::" << z << "\n";
46     }
47 };
48 class result : public marks, public xtra
49 {
50     protected:
51         float total;
52     public:
53     void disp(void)
54     {
55         total = s1+s2+z;
56         put_num();
57         put_marks();
58         put_xtra();
59         cout << "Total:"<< total;
60     }
61 };

```

---

```

62 int main()
63 {
64     result student1;
65     student1.get_num(10);
66     student1.get_marks(20,24);
67     student1.get_xtra(30.12);
68     student1.disp();
69     return 0;
70 }

```

**Output:**

```

Terminal
sh-4.3$ g++ -std=c++11 -o main *.cpp
main
sh-4.3$ main
USN Is:10
Subject 1:20
Subject 2:24
Extra Score::30.12
Total:74.12sh-4.3$

```



### Did you Know?

An abstract class is one that is not used to create objects and is designed only to act as a base class.



## Self-assessment Questions

- 6) In multi-level inheritance, the constructor is executed in the order of \_\_\_\_\_.
- 7) The only keyword allowed in the declaration of constructor.
  - a) Protected
  - b) Static
  - c) Inline
  - d) Private
- 8) The one where one or more types of inheritance are combined together and used is called \_\_\_\_\_.
  - a) Hierarchical
  - b) Hybrid
  - c) Single
  - d) Multiple



## Summary

- The technique of creating a new class from an existing class is called inheritance.
- The derived class inherits some or all of the properties of base class.
- A derived class with only one base class is called single inheritance.
- In multiple inheritance, a class can inherit features from more than one base class.
- A class can be derived from another derived class is known as multi-level inheritance
- When the features of one class are inherited by more than one class it is called Hybrid inheritance.
- A private member of a class cannot be inherited either in public or in private mode.
- A protected member inherited in public mode becomes protected, whereas inherited in private mode becomes private in the derived class.
- A public member inherited in public mode becomes public, whereas inherited in private mode becomes private in the derived class.



## Terminal Questions

1. Describe the concept of inheritance with base class and derived class.
2. Describe Constructors and destructors in derived classes.
3. Explain Single, hierarchical and Hybrid inheritance.
4. Compare Multiple and multi-level inheritance.
5. Explain member accessibility in inheritance.



## Answer Keys

Self-assessment Questions	
Question No.	Answer
1	Inheritance
2	b
3	a
4	Private, Public and Protected
5	b
6	Inheritance
7	a
8	b



## Activity

**1. Activity Type:** Online

**Duration:** 30 Minutes

**Description:**

1. Use private and protected inheritance to create two new classes from a base class. Then attempt to upcast the objects of the derived class to the base class. Explain what happens
2. Design and execute program to calculate gross and net pay of employee from basic salary. Create employee class which consists of employee name, employee id and basic salary as its data members. Use parameterized constructor in the derived class to initialize data members of the base class and calculate gross and net pay of the employee in the derived class.

---

## Case Study

**Problem Statement:** Let us look at the illustration of the constructors and destructors in inheritance

```
#include<iostream>
using namespace std;
class Base
{
    public:
        Base()  { cout << "Base's constructor called" << endl; }
};
class Derived
{
    public:
        Derived()  { cout << "Derived's constructor called" << endl; }
};
class Final: public Base, public Derived  // Note the order
{
    public:
        Final()  { cout << "Final's constructor called" << endl; }
};
int main()
{
    Final f;
    return 0;
}
```

- In which order the constructor is called?
- In which order the destructor is called?
- What is the output of the program?
- Which constructor is called first?



---

## Bibliography



### e-References

- programiz.com,(2016). *C++ Inheritance* .Retrieved 7 April, 2016. from, <http://www.programiz.com/cpp-programming/inheritance>
- cppforschool.com,(2016). *Inheritance - C++ Tutorial for School Students*. Retrieved 7 April, 2016. from, <http://www.cppforschool.com/tutorial/inheritance.html>



### External Resources

- Balaguruswamy, E. (2008). *Object Oriented Programming with C++*. Tata McGraw Hill Publications.
- Lippman. (2006). *C++ Primer* (3rd ed.). Pearson Education.
- Robert, L. (2006). *Object Oriented Programming in C++* (3rd ed.). Galgotia Publications References.
- Schildt, H., & Kanetkar, Y. (2010). *C++ completer* (1st ed.). Tata McGraw Hill.
- Strousstrup. (2005). *The C++ Programming Language* (3rd ed.). Pearson Publications



### Video Links

Topic	Link
Inheritance in C++	<a href="https://www.youtube.com/watch?v=EPffB2mFaSw">https://www.youtube.com/watch?v=EPffB2mFaSw</a>
Destructors in C++	<a href="https://www.youtube.com/watch?v=4P4Im0vF_mU">https://www.youtube.com/watch?v=4P4Im0vF_mU</a>
Multilevel Inheritance in C++	<a href="https://www.youtube.com/watch?v=uiH-X0GNtL8">https://www.youtube.com/watch?v=uiH-X0GNtL8</a>



**Notes:**

