5...

Inheritance

Objectives...

- To study how to define Base class and Derived class.
- To learn about different Types of Inheritance.
- To understand Virtual Base Class.
- To study Abstract class.
- To learn Constructors in derived class.

5.1 INTRODUCTION [W-18, S-19, 23]

- Inheritance is a form of software reusability in which new classes are created from existing classes by absorbing their attributes and behavior and overriding these with capabilities the new classes requires.
- Inheritance is the process by which one object can acquire the properties of another object. Inheritance is the process of creating new classes from an existing class.
- The existing class is known as base class and the newly created class is called as a derived class. The derived class inherits all capabilities of the base class. A derived class is more specific than its base class and represents a smaller group of objects.
- For example, a Red Delicious apple is part of the classification apple, which in turn is part of the fruit class. Hence, inheritance supports the concept of classification; with the help of classification an object need only define those qualities that make it unique within its class.
- The inheritance makes it possible for one object to be a specific instance of a more general case. Example of inheritance hierarchy of shape is shown in Fig. 5.1.

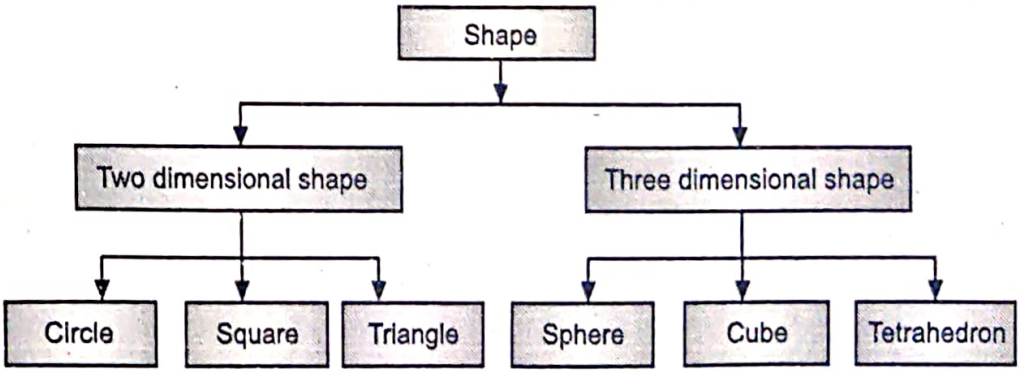

Fig. 5.1: 'Shape' Hierarchy

- The main advantages of inheritance are:
  1. It helps to implement software reusability. Once the hierarchy is done, the specific data to the particular class can be added as per the specification of the derived classes in hierarchy. Reusing existing code saves time and money.
  2. It increases the reliability of the code.
  3. Inheritance can also help in the original conceptualization of a programming problem and in the overall design of the program.
  4. It is useful to avoid redundancy, leading to smaller models that are easier to understand.
  5. It adds some enhancements to the base class.
  6. We can create new class based on base class without modifying base class.

5.2 DEFINING BASE CLASS AND DERIVED CLASS

- In C++, a class that is inherited is referred to as a base class. The class that does the inheriting is called the derived class.
- A derived class can be defined by specifying its relationship with the base class. The pictorial representation of single inheritance is shown in Fig. 5.2.

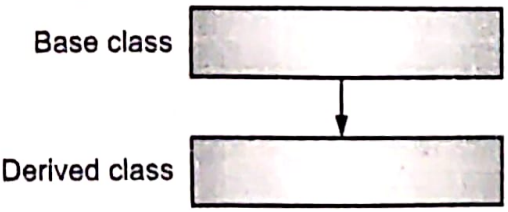

Fig. 5.2: Base Class and Derived Class

- Some example of inheritance is shown in Table 5.1.

Table 5.1: Examples of inheritance

Base Class	Derived Class
Shape	Circle Triangle Rectangle
Employee	Teacher Clerk
Media	Book Tape
Student	Undergraduate Postgraduate

- When a class inherits another, the member of the base class becomes members of the derived class.
- The **syntax** of derived class definition is as follows:

```
class <derived_class_name>: <access modifier> <base_class_name>
{
   // body of class
};
```

It contains:

- Keyword class.
- Derived class name which is user define name for new class.
- The : (colon) indicates that the derived_class_name is derived from the base_class_name.
- The access modifier is optional. If no access modifier is present, the access modifier is private by default. If written, then it must be public, private or protected.
- base_class_name is the existing class name.

- **Example:**

```
class base
{
    ..........
    ..........
};
class derived1: private base        //private derivation
{
    ..........
    ..........
};
class derived2: public base         //public derivation
{
    ..........
    ..........
};
class derived3: protected base      //protected derivation
{
    ..........
    ..........
};
class derived: base                 //by default private derivation
{
    ..........
    ..........
};
```

5.2.1 Public Derivation

- When the access modifier for a base class is **public**, all public members of the base class become public members of the derived class and all protected members of the base class becomes protected members of the derived class.

- The members of derived class can access members of the base class if the base class members are public or protected. They can not access private members.
- The Fig. 5.3 shows the base and derived class relationship.

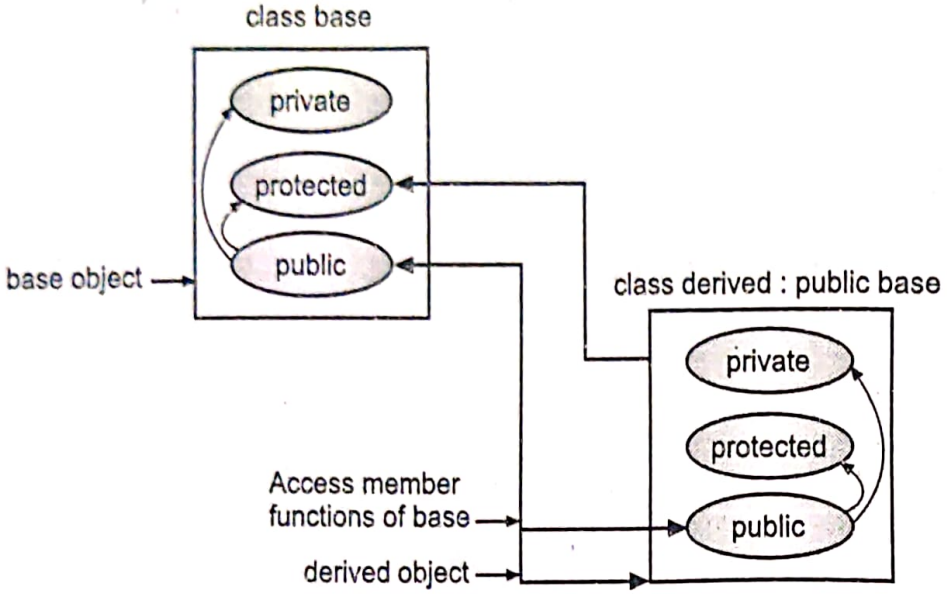

Fig. 5.3: Public Derivation

Program 5.1: Program to demonstrate public derivation.

```
#include<iostream>
using namespace std;
class base
{
   int i, j;
   public:
      void getij()
      {
         cout<<"Enter value of i and j";
         cin>>i;
         cin>>j;
      }
      void show()
      {
         cout<<"i="<<i<<"j="<<j<<"\n";
      }
};
class derived: public base
{
   int k;
   public:
      void getdata()
      {
         getij();
            cout<<"Enter value of k";
            cin>>k;
```

```
            cout<<"k="<<k;
        }
};
int main()
{
    base ob1;        //base object
    derived ob2;     //derived object
    ob1.getij();     //access members of base class
    ob1.show();      //display values of i and j
    ob2.getdata();   //access members of derived class
    ob2.show();      //access members of base class
}
```

Output:

```
Enter value of i and j 2 3
i=2 j=3
Enter value of i and j 5 6
Enter value of k 7
k=7 i=5 j=6
```

- In this program, the base class members are directly access through derived class, because the derived through derived class and derived class is publically derived from base class.
- In inheritance some of the base class data elements and member functions are inherited into the derived class which extends the capability of the existing classes and useful in incremental program development. If the function names of the base and derived class are same then use scope resolution operation (::) and show the scope of the function i.e. to which the class that function belongs.

Program 5.2: Write a program to read data members of a base class i.e. name, person id and the derived class members are height and weight. Display the data of person.

```
#include<iostream>
using namespace std;
class base
{
   private:
      char name[20];
      int person_id;
   public:
      void getdata()
      {
          cout<<"Enter name and person_id";
          cin>>name>>person_id;
      }
```

```
    void display()
    {
        cout<<"name:"<<name<<"  "<<"person_id:"<<person_id;
    }
};
class derived: public base
{
    private:
        int height, weight;
    public:
        void getdata()
        {
            base:: getdata();
            cout<<"Enter height and weight";
            cin>>height>>weight;
        }
        void display()
        {
            base::display();
            cout<<"height:"<<height<<" "<<"weight:"<<weight;
        }
};
int main()
{
    derived obj;
    obj.getdata();
    obj.display();
}
```

Output:

```
Enter name and person_id prajakta 11
Enter height and weight 6 80
name: prajakta  person_id:11 height:6  weight:80
```

5.2.2 Private Derivation

- When the base class is inherited by using the private access modifier, all public and protected members of the base class become private members of the derived class.
- The Fig. 5.4 shows the private derivation of inheritance.

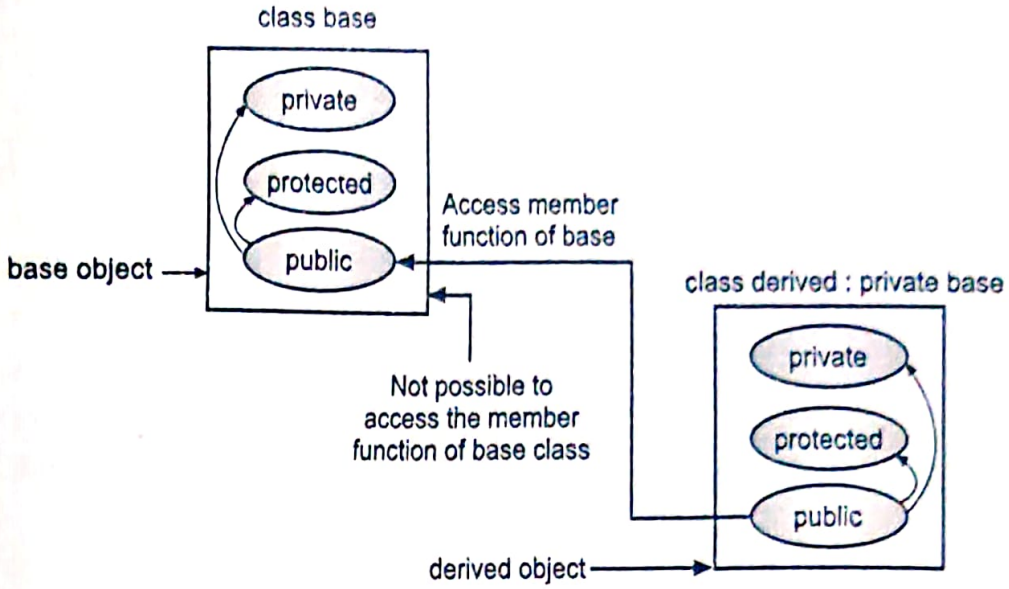

Fig. 5.4: Private Derivation

The following Program 5.3 illustrate this concept.

Program 5.3: Program for private derivation.

```
#include <iostream>
using namespace std;
class Engine
{
    public:
        Engine(int nc)
        {
            cylinder = nc;
        }
        void start()
        {
            cout << getCylinder() <<" cylinder engine started" << endl;
        }
        int getCylinder()
        {
            return cylinder;
        }
    private:
        int cylinder;
};
class Car: private Engine
{
    public:
        Car(int nc = 4): Engine(nc) { }
```

```
        void start()
        {
            cout << "car with " << Engine::getCylinder() <<
            "cylinder engine started" << endl;
            Engine:: start();
        }
};
int main( )
{
    Car c(8);
    c.start();
    return 0;
}
```

Output:

```
car with 8 cylinder engine started
8 cylinder engine started
```

- In this program, derived class object cannot access the functions of base class because they become private. But the public functions of base class access by public functions of derived class.

5.2.3 Protected Members

- The protected keyword provides greater flexibility in the inheritance mechanism. When a member of a class is declared as protected, that member is not accessible by other, non-member elements of program.
- Protected members are as good as private members within the class. Protected member differs from private members when a protected member is inherited. Using the access modifier protected in base class, we could make available the data member in the derived class and restrict its access in other classes or in main().

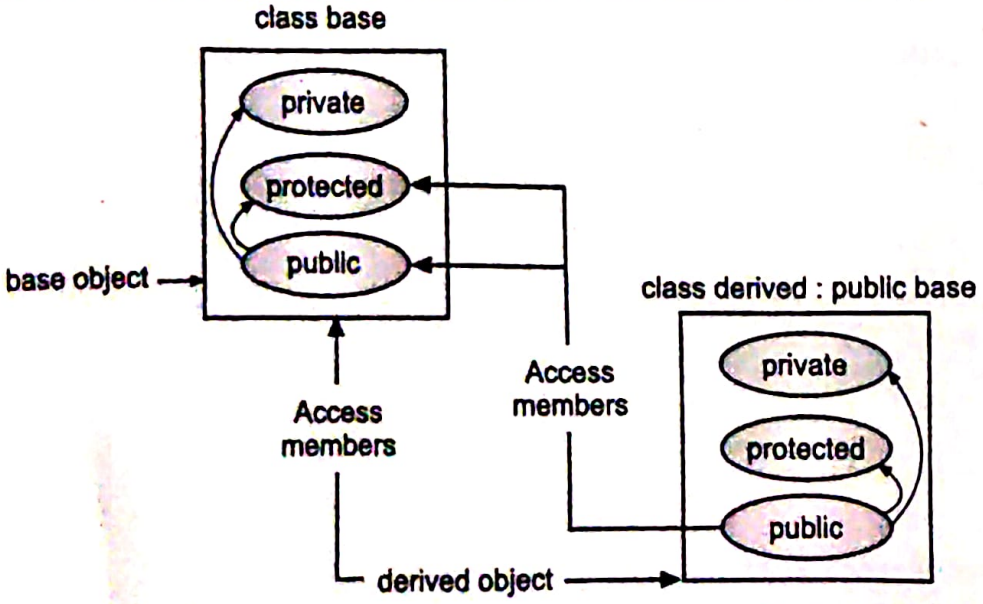

Fig. 5.5: Protected members of base class can be access in derived class

- Data member declare as protected within a base class is accessible by the member within its class and any class immediately derived from it. The Fig. 5.5 shows the public derivation with protected data members.
- If the base class is inherited as public, then the base class protected member becomes protected member of derived class and therefore, can be accessed by derived class. Thus, by using **protected**, we can create class members that are private to their class but can still be inherited and accessed by a derived class.

Program 5.4: Program using protected members.

```
#include<iostream>
using namespace std;
class base
{
    protected:
        int i, j;    // private to base, but accessible by derived class
    public:
        void getij()
        {
            cout<<"Enter value of i and j";
            cin>>i>>j;

        }
        void show()
        {
            cout<<"i="<<i<<" "<<"j="<<j<<"\n";
        }
};
class derived1: public base
{
    int k;
    public:
        void getdata()
        {
            getij();
                k=i*j;
        }
        void showall()
        {
            cout<<"k="<<k;

        }
}; //end of class derived1
```

```
class derived2: public derived1
{
    int x;
    public:
        void get()
        {
            getij();
            x=i*j;
        } // can access i and j since public inheritance
        void showx()
        {
            cout<<"x="<<x;

        }
}; //end of class derived2
int main()
{
    derived1 ob1;
    derived2 ob2;
    ob1.getdata();
    ob1.showall();
    ob2.get();
    ob2.showx();
}
```

Output:

```
Enter value of i and j 10 20
k=200
Enter value of i and j 15 10
x=150
```

- In the above program, if derived1 class inherits base class privately, then protected member of base becomes private members of derived1 class which cannot inherited in derived2 class. So, when a protected member is inherited in public mode, it becomes protected in the derived class and therefore it is accessible by the member function of the derived class. It is also inherited further.
- A protected member inherited in the private mode becomes private if the derived class and cannot be inherited further.

5.2.4 Protected Derivation

- C++ allows us to use protected as access modifier. When access modifier is protected, all public and protected members of the base class become protected members of the derived class.

Program 5.5: Program for protected derivation.

```
#include<iostream>
using namespace std;
class base
{
    protected:
        int i, j;   //private to base but accessible in derived
    public:
        void getij()
        {
            cin>>i>>j;
        }
        void show()
        {
            cout<<i<<" "<<j<<"\n";
        }
};
class derived: protected base
{
    int k;
    public:
        void getdata()
        {
            getij();
            k = i*j;   // derived may access base protected data members
        }
        void showall()
        {
            cout<<k<<"  ";
            show();

        }
};
int main()
{
    derived ob;
    //ob.getij();    //illegal, getij() is protected member of derived
    ob.getdata();    // correct as public member of derived
    ob.showall();    // correct as public member of derived
    //ob.show();     // illegal
}
```

Output:

```
2   3
6   2   3
```

- In this program, getij( ) and show() are public members of base, but they become protected members of derived. Using derived object is not accessible.

Summary about Access Specifiers:

Table 5.2: Access in a Derived Class

Base class members	Derived Class		
	Public derivation	**Private derivation**	**Protected derivation**
Private	not accessible	not accessible	not accessible
Protected	protected	private	protected
Public	public	private	protected

5.3 TYPES OF INHERITANCE [W-18, 22; S-19]

- There are many ways in which we can create derived class. There are five types of inheritance as follows:
  1. Single inheritance
  2. Multilevel inheritance
  3. Multiple inheritance
  4. Hierarchical inheritance and
  5. Hybrid inheritance

5.3.1 Single Inheritance

- When one base class is derived by one child class, then it is known as single inheritance. The existing base class is known as direct base class whereas, they newly created class is called as **singly derived class.** It is shown in Fig. 5.6.

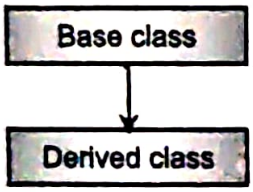

Fig. 5.6: Single Inheritance

- **Example:** Suppose a college want to store information of undergraduate (UG) and postgraduate students (PG). Both are having registration no., name, and address. Each postgraduate student also have additional attributes along with above attributes are placement company name. It can show as in Fig. 5.7.

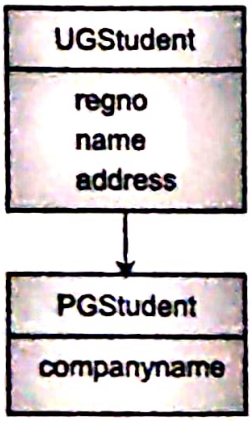

Fig. 5.7

Program 5.6: Program to illustrate single inheritance.

```
#include<iostream>
using namespace std;
class student
{
    protected:
        int regno;
        char name[20];
        char addr[30];
    public:
        void getdata()
        {   cout<<"enter registration no, name and address";
            cin>>regno>>name>>addr;
        }
        void display()
        {
            cout<<"registration no.= "<<regno;
            cout<<"name = "<<name;
            cout<<"address = "<<addr;
        }
};   //end of base class
class PGstudent: public student
{
        private:
                char compname[20];
        public:
                void get_PG_data()
                {
                    getdata();
                    cout<<"\nEnter the company name=";
                    cin>>compname;
                }
                void dispPG()
                {
                    cout<<"\n registration no. ="<<regno;
                    cout<<"\n name ="<<name;
                    cout<<"\n address ="<<addr;
                    cout<<"\n company name = "<<compname;
                }
    };
```

```
int main()
{
    student s;
    PGstudent ps;
    s.getdata();
    s.display();
    ps.get_PG_data();
    ps.dispPG();
}
```

Output:

```
Enter registration no, name and address
101 Ram Pune
registration no = 101  name = Ram     addr = Pune
Enter registration no., name & address
401 Sita Pune
Enter the company name = zenex
Registration no = 401
name = Sita
address = Pune
Company name = Zenex
```

Program 5.7: A base class contains the data of employee name, empno and sex. The derived class contains basic salary. The derived class has been declared as an array of class objects. The program illustrate the single inheritance

```
#include<iostream.h>
class base
{
    private:
        char name[20];
        int empno;
        char sex;
    public:
        void get();
        void disp();
};
class derived: public base
{
    private:
        float bsalary;
    public:
        void get();
        void disp();
};
```

```
void base:: get()
{
    cout<<"\n Enter name:";
    cin>>name;
    cout<<"\n Enter Employee id:";
    cin>>empno;
    cout<<"\n Enter Sex:";
    cin>>sex;
}
void base:: disp()
{
    cout<<"name ="<<name;
    cout<<"Emp no. is ="<<empno;
    cout<<"sex="<<sex;
}
void derived:: get()
{
    base:: get();
    cout<<"Enter basic salary: \n";
    cin>>bsalary;
}
void derived:: disp()
{
    base::disp();
    cout<<"basic salary = "<<bsalary;
}
void main()
{
    derived ob[10];
    int i, n;
    cout<<"How many employees?";
    cin>>n;
    for (i=0;i<n;i++)
    {
       ob[i].get();
    }
    cout<<endl;
    cout<<"name Empno.sex bsalary \n";
```

```
    for (i=0;i<n;i++)
    {
        ob[i].disp();
        cout<<endl;
    }
}
```

Output:

```
How many employees? 2

Enter name: Prajakta
Enter Employee id: 101
Enter Sex: f
Enter basic salary: 10000

Enter name: Pranjal
Enter Employee id: 102
Enter Sex: m
Enter basic salary: 40000

name            Empno              sex          bsalary
name =abc    Emp no. is =101       sex=f        basic salary = 10000
name =xyz    Emp no. is =102       sex=m        basic salary = 40000
```

5.3.2 Multilevel Inheritance

- When a base class is derived by a child class which further derived by another child class, then it is known as Multilevel Inheritance.
- The class which provides link between two classes is known as intermediate base class. This is illustrate in Fig. 5.8.

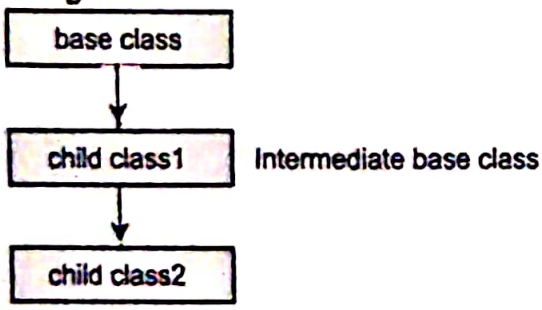

Fig. 5.8: Multilevel Inheritance

Practically, we can have more than two levels also.

- **Syntax of multilevel inheritance:**

```
class base
{
    ----
    ----
};
```

```
class child1: public base
{
    ____
    ____
};
class child2: public child1
{
    ____
    ____
};
```

- Consider an example of bank. In a bank, different customers have savings account. Some customers may have taken loan from the bank. So the bank always maintains information about bank depositors and borrowers. This can be shown using the following Fig. 5.9.

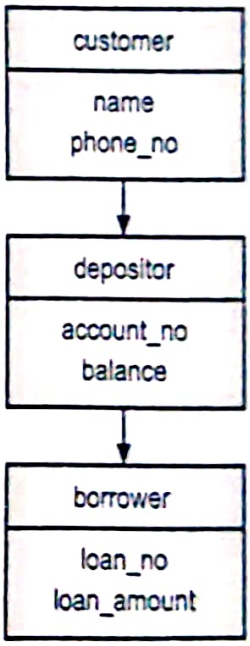

Fig. 5.9: Example of Bank

Program 5.8: Program to illustrate use of multilevel inheritance.

```
#include<iostream>
using namespace std;
class customer      // base class
{
   protected:
      char name[30];
      int phone_no;
}; // end of base class
```

```
class depositor: public customer      // child class1
{
   private:
      int account_no;
      float balance;
   public:
      void get_depositor_data( )
      {
         cout<<"\n Enter name & phone no ";
         cin>>name>>phone_no;
         cout<<"\n Enter account no and balance ";
         cin>>account_no>>balance;
      }
      void display_depositor_data( )
      {
         cout<<"\n Name = "<<name;
         cout<<"\n Phone no = "<<phone_no;
         cout<<"\n Account no = "<<account_no;
         cout<<"\n Balance = "<<balance;
      }
}; // end of child class1
class borrower: public depositor            // child class2
{
   protected:
      int loan_no;
      float loan_amount;
    public:
      void get_loan_data( )
      {
         cout<<"\n Enter loan no and loan amount";
         cin>>loan_no>>loan_amount;
      }
      void display_loan_data( )
      {
         cout<<"\n Loan no = "<<loan_no;
         cout<<"\n Loan amount = "<<loan_amount;
      }
}; // end of child class2
```

```
int main( )
{
    int choice;
    cout<<"1-Read & display depositor information "<<endl;
    cout<<"2-Read & display depositor & borrower information"<<endl;
    cout<<"Enter your choice = ";
    cin>>choice;
    switch (choice)
    {
        case 1:
            depositor d;
            d.get_depositor_data( );
            d.display_depositor_data( );
            break;
        case 2:
            borrower b;
            b.get_depositor_data( );
            b.get_loan_data( );
            b.display_depositor_data( );
            b.display_loan_data( );
            break;
    }
} // end of main( )
```

Output:

```
1-Read & display depositor information
2-Read & display depositor & borrower information
Enter your choice = 1
Enter name & phone no Prajakta 98776677
Enter account no and balance 123456789 10000
Name = Prajakta
Phone no = 98776677
Account no = 123456789
Balance = 10000
```

- In the above program, though the data members of depositor class are private, they can be accessed indirectly by the borrower class through the member functions of the depositor class indirectly.

Program 5.9: Consider the example of student data. Base class is student. Class test is intermediate base class and result is child class. The class result inherits the details of marks obtained in the test and the roll no. of student from base class through multilevel inheritance. [S - 18]

```
#include<iostream>
using namespace std;
class student
{
   protected:
      int roll;
   public:
      void get(int x)
      {
         roll = x;
      }
      void display()
      {
         cout<<"roll="<<roll;
      }
};
class test: public student
{
   protected:
      int mark1, mark2;
   public:
      void getmark(int a, int b)
      {
         mark1 = a;
         mark2 = b;
      }
      void dispmark()
      {
         cout<<"mark1 = \n"<<mark1<<"mark2 ="<<mark2;
      }
};
class result: public test
{
   private:
      int total;
```

```
    public:
        void displayall()
        {
            total = mark1 + mark2;
            display();
            dispmark();
            cout<<"total="<<total;
        }
};
int main()
{
    result r;
    r.get(10);
    r.getmark(60,75);
    r.displayall();
}
```

Output:

```
roll = 10
mark1 = 60
mark2 = 75
total = 135
```

5.3.3 Multiple Inheritance

- When more than one base classes are inherited by a derived class, then such type of inheritance is called as multiple inheritance. Fig. 5.10 illustrates the point.

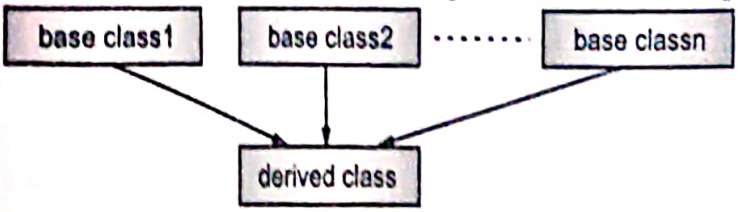

Fig. 5.10: Multiple inheritance

Example: A child inherits the features from father and intelligence from mother.

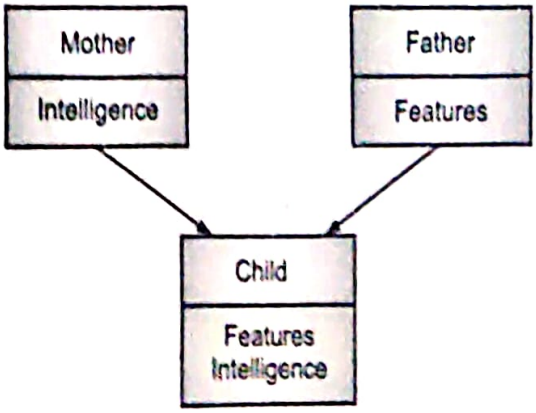

Fig. 5.11: Multiple Inheritance

- **Syntax of Multiple Inheritance:**

1. **Multiple inheritance with all public derivation:**

```
class base1
{
    —
    —
    —
};
class base2
{
    —
    —
    —
};
class derived: public base1, public base2
{
    —
    —
    —
};
```

Here access specifier may be public or private and the base classes are separated by commas.

2. **Multiple inheritance with all private derivation:**

```
class base1
{
    —
    —
    —
};
class base2
{
    —
    —
    —
};
class derived: private base1, private base2
{
    —
    —
    —
};
```

3. Multiple inheritance with all mixed derivation:

```
class base1
{
    —
    —
    —
};
class base2
{
    —
    —
    —
};
class base3
{
    —
    —
    —
};
class derived: pubic base1, public base2, private base3
{
    —
    —
    —
};
```

Let us discuss the program how multiple inheritance works.

Program 5.10: Program for multiple inheritance.

```
#include<iostream>
using namespace std;
class basex
{
    protected:
        int x;
    public:
        void get_x (int a) {x = a;}
};
class basey
{
    protected:
        int y;
    public:
        void get_y (int b) {y = b;}
};
```

```
class derived: public basex, public basey
{
    public:
        void display()
        {
            cout<<"x = " <<x<<"\n";
            cout<<"y = "<<y<<"\n";
            cout<<"x+y = "<<x+y<<"\n";
        }
};
int main()
{
    derived ob;
    ob.get_x(50);
    ob.get_y(60);
    ob.display();
}
```

Output:

```
x = 50
y = 60
x + y = 110
```

Program 5.11: Consider an example to print bio-data of a diploma student having personal and academic information. Program to illustrate use of multiple inheritance. **[S - 19]**

```
#include<iostream>
#include<cstring>
using namespace std;
class personal                                    // base class1
{
    private:
        char name [30], email_id[50];
    protected:
    void get_personal_info( )
    {
        cout<<"\n Enter name & email id of a person = ";
        cin>>name>>email_id;
    }
```

```
    void display_personal_info( )
    {
        cout<<"\n Name = "<<name;
        cout<<"\n Email_id = "<<email_id;
    }
};  // end of base class1
class academic                                  // base class 2
{
    private:
        float tenth_marks;
        char tenth_class[20];
    protected:
        void get_academic_info( )
        {
            cout<<"\n Enter tenth marks";
            cin>>tenth_marks;
        }
        void find_class( )
        {
            if (tenth_marks>=70)
                strcpy (tenth_class, "Distinction");
            else
            if (tenth_marks>=60)
                strcpy (tenth_class, "First class");
            else
            if (tenth_marks>=50)
                strcpy (tenth_class, "Second class");
            else
            if (tenth_marks>=40)
                strcpy (tenth_class, "Pass class");
            else
                strcpy (tenth_class, "Fail");
        }
        void display_academic_info( )
        {
            cout<<"\n Tenth marks = "<<tenth_marks;
            find_class( );
            cout<<"\n Tenth class = ";
            cout<<tenth_class;
        }
}; // end of base class2
```

```
class bio_data: private personal, private academic      // Derived class
{
   public:
      void display_biodata( );
      void get_info( );
}; // end of derived class
   // Member functions of derived class
void bio_data::display_biodata( )
{
   cout<<"BIO DATA "<<endl;
   cout<<"Personal info = "<<endl;
   display_personal_info( );
   cout<<"Academic info= "<<endl;
   display_academic_info( );
}
void bio_data::get_info( )
{
   get_personal_info( );
   get_academic_info( );
}
int main( )
{
   bio_data b;
   // First read info
   b.get_info( );
   // Now print biodata
   b.display_biodata( );
}
```

Output:

```
Enter name & email id of a person = Prajakta abc@yahoo
Enter tenth marks 80.5
BIO DATA
Personal info =
Name = Prajakta
Email_id = abc@yahoo
Academic info=
Tenth marks = 80.5
Tenth class = Distinction
```

Ambiguity in Multiple Inheritance:

- One common cause of ambiguity in multiple inheritance is when two or more base classes have methods with the same name, while there is no method of that name in the derived class. In this case objects of the derived class have no way of knowing which of the parent methods is to be executed. The scope resolution operator can be used to resolve the ambiguity.
- Another element of ambiguity in multiple inheritance arises when two or more derived classes inherit the same member from a common base class. If now another derived class inherits from these classes, two copies of the original member function could be inherited.
- Fig. 5.12 shows one possible scenario that leads to inheritance of multiple copies.
- One possible solution to this duplicity is the use of the scope resolution operator. Another solution of this problem is to declare the base classes to be virtual at the time of they are inherited.

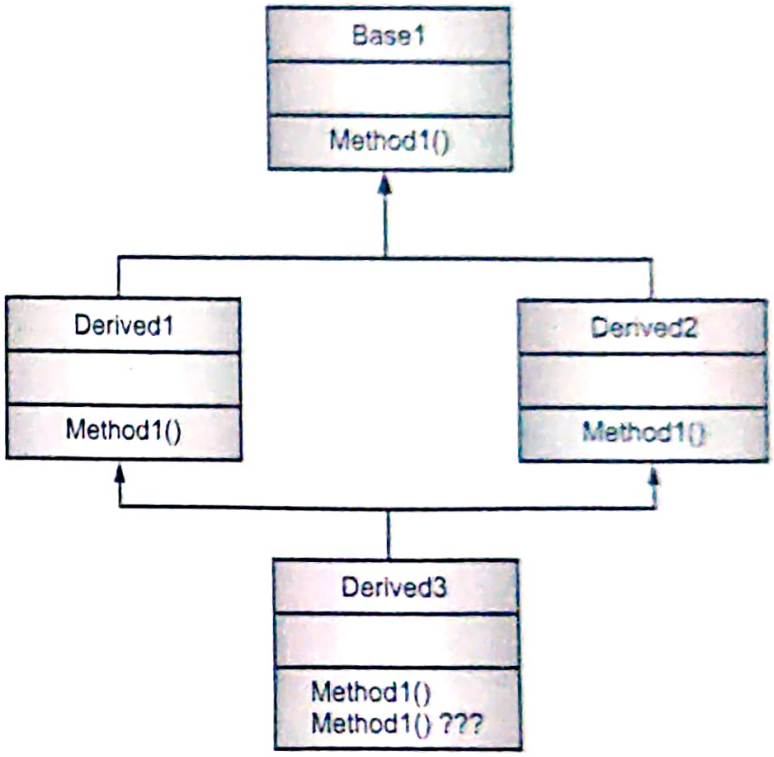

Fig. 5.12: Inheritance of Multiple Copies

Advantages of Multiple Inheritance:

1. Derived class typically represents a combination of its base classes.
2. It has rich semantics
3. It has ability to directly express complex structures.

5.3.4 Hierarchical Inheritance

S-23

- When one base class is inherited by more than one derived classes, it is known as hierarchical inheritance. In this, many programming problems can be cast into a hierarchy where certain features of one level are shared by many other below that level.
- The Fig. 5.13 illustrate this point.

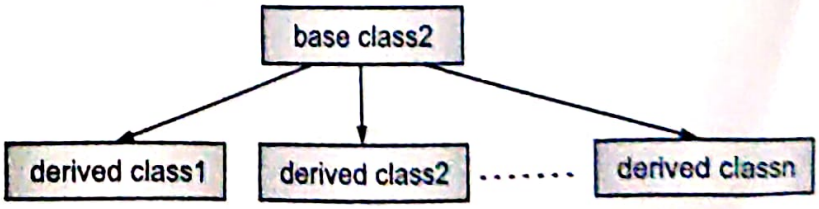

Fig. 5.13: Hierarchical Inheritance

- **Syntax of Hierarchical Inheritance:**

```
class base
{
    .....
    .....
    .....
};
class derived1: public base
{
    .....
    .....
    .....
};
class derived2: public base
{
    .....
    .....
    .....
};
```

- **Example:** Consider an example of an organisation having full time and part time employees. Depending on it, the employee salary is calculated. The hierarchy is:

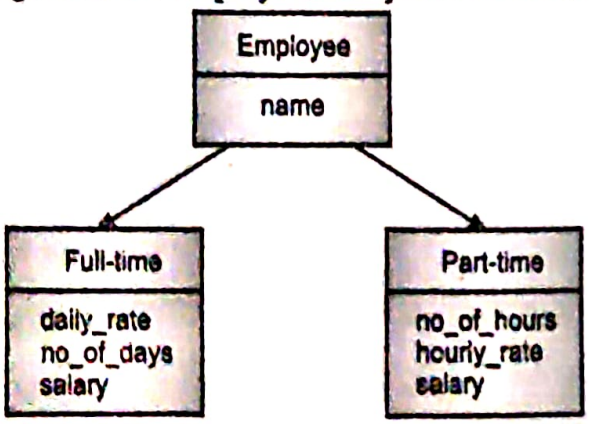

Fig. 5.14: Hierarchical Inheritance

Program 5.12: Program to illustrate use of hierarchical inheritance.

```
#include<iostream>
using namespace std;
class employee  // base class
{
   protected:
      char name[30];
}; // end of base class
```

```
class full_time_emp: protected employee
{
    protected:
        float daily_rate, salary;
        int no_of_days;
    public:
        void read_emp_data( );
        void display_emp_data( );
        void calculate_salary( );
}; // end of derived class1

void full_time_emp::read_emp_data( )
{
    cout<<endl<<"Enter name of employee =";
    cin>>name;
    cout<<endl<<"Enter working rate for one day = ";
    cin>>daily_rate;
    cout<<endl<<"Enter no. of working days of employee=";
    cin>>no_of_days;
}
void full_time_emp::calculate_salary( )
{
    salary = daily_rate *no_of_days;
}
void full_time_emp::display_emp_data( )
{
    cout<<endl<<"For the full time employee:=";
    cout<<endl<<"Name="<<name;
    cout<<endl<<"Daily rate ="<<daily_rate;
    cout<<endl<<"No. of working days ="<<no_of_days;
    cout<<endl<<"Salary ="<<salary;
}
class part_time_emp:protected employee                    // derived class2
{
    protected:
            float hourly_rate, salary;
            int no_of_hours;
    public:
            void read_emp_data( );
            void display_emp_data( );
            void calculate_salary( );
}; // end of derived class 2
```

```
void part_time_emp::read_emp_data( )
{
   cout<<endl<<"Enter name of employee=";
   cin>>name;
   cout<<endl<<"Enter working rate for one hour =";
   cin>>hourly_rate;
   cout<<endl<<"Enter no. of working hours for employee =";
   cin>> no_of_hours;
}
void part_time_emp::calculate_salary( )
{
   salary = hourly_rate * no_of_hours;
}
void part_time_emp::display_emp_data( )
{
   cout<<endl<<"For the part time employee";
   cout<<endl<<"Name ="<<name;
   cout<<endl<<"Hourly rate ="<<hourly_rate;
   cout<<endl<<"No. of working hours="<<no_of_hours;
   cout<<endl<<"Salary="<<salary;
}
int main( )
{
   int choice;
   cout<<endl<<"You have following choices";
   cout<<endl<<"1-Read & display information of full time employee";
   cout<<endl<<"2-Read & display information of part time employee";
   cout<<endl<<"Enter your choice=";
   cin>>choice;
   switch(choice)
   {
      case 1:
            full_time_emp ft;
            // First read data
            ft.read_emp_data( );
            ft.calculate_salary( );
            // Now display data
            ft.display_emp_data( );
            break;
```

```
        case 2:
                part_time_emp pt;
                // First read data
                pt.read_emp_data( );
                pt.calculate_salary( );
                // Now display data
                pt.display_emp_data( );
                break;
    }
}
```

Output:

```
You have following choices
1-Read & display information of full time employee
2-Read & display information of part time employee
Enter your choice=1
Enter name of employee = Prajakta
Enter working rate for one day = 250
Enter no. of working days of employee=30
For the full time employee:=
Name=Prajakta
Daily rate =250
No. of working days =30
Salary =7500
```

- In the above program, both derived classes have same names for the member functions. You can also use different names.
- Also in the above program, you can use array of objects to store information about more than one employee.

5.3.5 Hybrid Inheritance

- Sometimes, we may require to combine two or more types of inheritance to design a program; the result is known as Hybrid Inheritance. Fig. 5.15 illustrates this point.

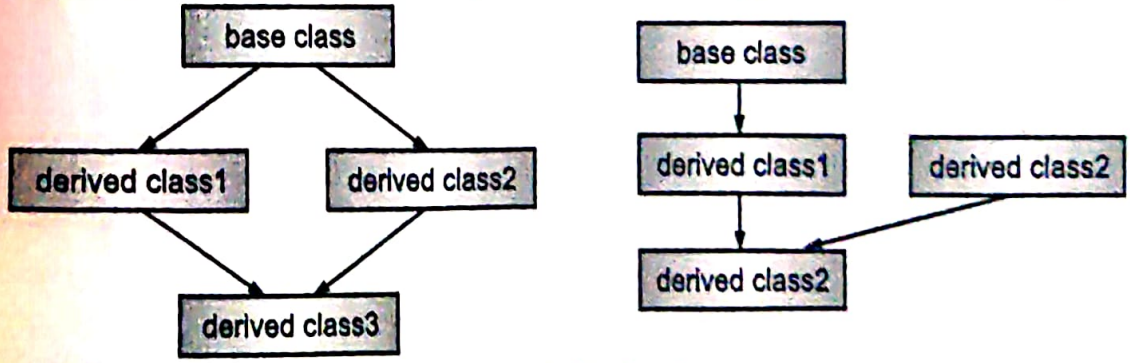

Fig. 5.15: Hybrid Inheritance

- **Example:** Consider an example where we have three classes' student, test and result. But some authority wants that some weightage should be given for sports to finalize the result. This is shown in Fig. 5.16.

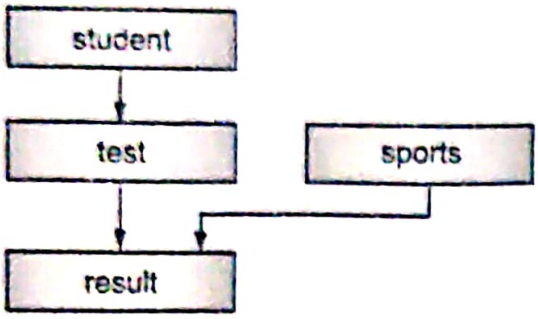

Fig. 5.16: Hybrid Inheritance (Multiple and Multilevel)

Program 5.13: Program for hybrid inheritance.

```
#include<iostream.h>
class student
{
   protected:
      int rollno;
   public:
      void get (int x)
      { rollno = x; }
      void display()
      { cout<<rollno; }
};
class test: public student
{
   protected:
      int mark1, mark2;
   public:
      void getmark(int a, int b)
      {
         mark1 = a;
         mark2 = b;
      }
      void dispmark()
      { cout<<mark1<<"  "<<mark2; }
};
class sport
{
   protected:
      int smark;
```

```
    public:
        void getsm(int s)
        { smark = s; }
        void disps()
        { cout<<smark<<"\n"; }
};
class result: public test, public sport
{
    private:
        int total;
    public:
        void displaytotal()
        {
            total = mark1 + mark2 + smark;
            display();
            dispmark();
            disps();
            cout<<"total = "<<total;
        }
};
void main()
{
    result r;
    r.get(10);
    r.getmark(50, 60);
    r.getsm(70);
    r.displaytotal();
}
```

Output:

```
10    50    60    70
total = 180
```

5.4 VIRTUAL BASE CLASS [W-18, 22; S-19, 22]

- Consider a situation where multilevel, multiple and hierarchical all the three kinds of inheritance are involved. This is illustrate in Fig. 5.17.
- In the Fig. 5.17, the base class is inherited by both Derived1 and Derived2. Derived3 directly inherits both Derived1 and Derived2. All the public and protected members of Base are inherited into Derived3 twice through both Derived1 and Derived2. Therefore, Derived3 would have duplicate sets of members inherited. This causes ambiguity when a member of Base is used by Derived3.

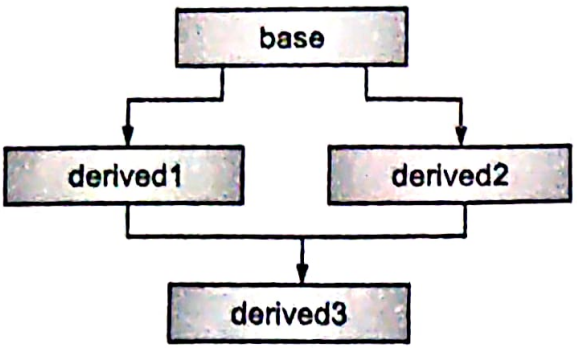

Fig. 5.17: Multipath Inheritance

- To resolve this ambiguity, C++ includes a mechanism by which only one copy of Base will be included in Derived3. This feature is called a virtual base class. When a class is made virtual, C++ take necessary care to inherit only one copy of that class. The keyword virtual precedes the base class access specifier when it is inherited by a derived class.
- The situation in Fig. 5.17 can be declared as:

```
class Base
{ ..... };
class Derived1: virtual public Base
{ ..... };
class Derived2: virtual public Base
{ ..... };
class Derived3: public Derived1, public Derived2
{ ..... };
```

Program 5.14: Program using virtual base class.

```
#include<iostream.h>
//using namespace std;
class base
{
   public:
   int x;
};
class derived1: virtual public base
{
   public:
   int y;
};
class derived2: virtual public base
{
   public:
   int z;
};
```

```
/* derived3 inherits derived1 and derived2. However, only one copy of base
is present */
class derived3: public derived1, public derived2
{
    public:
    int mult()
    { return x * y * z; }
};
void main()
{
    derived3 d;
    d.x = 10;
    d.y = 20;
    d.z = 30;
    cout<<"product is"<<d.mult();
}
```

Output:

```
product is 6000
```

Note:

- If derived1 and derived2 classes had not inherited base as virtual, then statement d.x = 10; becomes ambiguous and a program would have resulted as compile-time error.
- If virtual base classes are used when an object inherits the base more than once, only one base class is present in the object.
- If normal base classes are used when an object inherits the base more than once, the multiple copies will be found; and it is compile-time error.

Program 5.15: To calculate the total mark of a student using the concept of virtual base class.

```
#include <iostream>
using namespace std;
class student
{
  int rno;
  public:
  void getnumber()
  {
       cout<<"Enter Roll No:";
       cin>>rno;
  }
```

```
    void putnumber()
    {
    cout<<"\n\n\tRoll No:"<<rno<<"\n";
    }
};
class test:virtual public student
{
  public:
   int part1,part2;
   void getmarks()
   {
       cout<<"Enter Marks\n";
      cout<<"Part1:";
      cin>>part1;
      cout<<"Part2:";
      cin>>part2;
   }
   void putmarks()
   {
      cout<<"\tMarks Obtained\n";
      cout<<"\n\tPart1:"<<part1;
      cout<<"\n\tPart2:"<<part2;
   }
};
class sports:public virtual student
{
  public:
    int score;
    void getscore()
    {
       cout<<"Enter Sports Score:";
      cin>>score;
    }
    void putscore()
    {
      cout<<"\n\tSports Score is:"<<score;
    }
};
```

```
class result:public test,public sports
{
    int total;
    public:
    void display()
    {
        total=part1+part2+score;
        putnumber();
        putmarks();
        putscore();
        cout<<"\n\tTotal Score:"<<total;
    }
};
int main()
{
    result obj;
    obj.getnumber();
    obj.getmarks();
    obj.getscore();
    obj.display();
}
```

Output:

```
Enter Roll No: 151
Enter Marks

Part1: 90
Part2: 80
Enter Sports Score: 80

Roll No: 151
Marks Obtained
Part1: 90
Part2: 80
Sports Score is: 80
Total Score is: 250
```

5.5 ABSTRACT CLASS [S - 18]

- A class that contains at least one pure virtual function is said to be abstract. A pure virtual function is a member function i.e., declared in an abstract class, but defined in a derived class.
- An abstract class does not create any object and it contains one or more functions for which there is no definition (i.e., a pure virtual function).

- We can create pointers and references to an abstract class. This allows abstract classes to support runtime polymorphism, which depends upon base class pointers and references to select the proper virtual function.
- An abstract class is used only to derive other classes. For example: Shape is abstract base class which derives square, circle and rectangle.

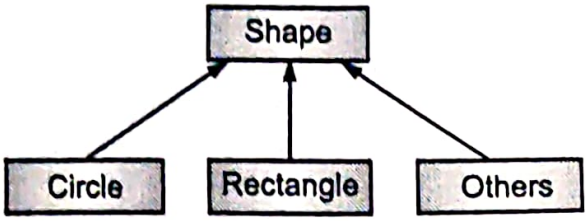

Fig. 5.18

Program 5.16: Following program shows use of abstract classes.

```
#include <iostream>
using namespace std;
// Abstract class
class Shape
{
    protected:
       float l;
    public:
       void getData()
       {
           cin >> l;
       }
       // virtual Function
       virtual float calculateArea() = 0;
};
class Square: public Shape
{
    public:
       float calculateArea()
       {   return l*l;  }
};
class Circle: public Shape
{
    public:
       float calculateArea()
       { return 3.14*l*l; }
```

```
int main()
{
    Square s;
    Circle c;
    cout << "Enter length to calculate the area of a square: ";
    s.getData();
    cout<<"Area of square: " << s.calculateArea();
    cout<<"\nEnter radius to calculate the area of a circle: ";
    c.getData();
    cout << "Area of circle: " << c.calculateArea();
    return 0;
}
```

Output:

```
Enter length to calculate the area of a square: 4
Area of square: 16
Enter radius to calculate the area of a circle: 5
Area of circle: 78.5
```

5.6 CONSTRUCTORS IN DERIVED CLASS

- There are two major questions that arise relative to constructors and destructors when inheritance is involved. First, when are base class and derived class constructor and destructor functions called? Second, how can parameters be passed to base class constructor functions?
- As long as no base class constructor takes any argument, the derived class need not have a constructor function. If a base class contains a constructor with one or more arguments then it is mandatory for a derived class to have a constructor pass the arguments to the base class constructor.

1. Order of calling constructors:

- When both the derived and base classes contain constructors, then base constructor is executed first and then the constructor is the derived class is executed.
- In case of multiple and multilevel inheritance, base classes are constructed in the order in which they appear in the declaration of derived class.
- When a derived object is destroyed its destructor is called first and then base class destructor.
- Constructor's functions are executed in their order of derivation. Destructor functions are executed in reverse order of derivation.

Program 5.17: Program for order of calling constructor.

```
#include<iostream.h>
class base
{
    public:
        base()
        {
            cout<<"constructing base \n";
        }
```

```
        ~base()
        {
            cout<<"destructing base \n";
        }
};
class derived1: public base
{
    public:
        derived1()
        {
            cout<<"constructing derived1 \n";
        }
        ~derived1()
        {
            cout<<"destructing derived1 \n";
        }
};
class derived2: public derived1
{
    public:
        derived2()
        {
            cout<<"constructing derived2 \n";
        }
        ~derived2()
        {
            cout<<"destructing derived2 \n";
        }
};
int main()
{
    derived2 ob;
    //construct and destruct ob
    return 0;
}
```

Output:

```
constructing base
constructing derived1
constructing derived2
destructing derived2
destructing derived1
destructing base
```

2. Passing parameter to base class constructor:

- In case where derived class constructor requires one or more parameters, you simply use standard parameterized syntax. However, how do we pass arguments to a constructor in a base class?
- The constructor of the derived class receives the entire list of arguments with their values and passes them to base constructor. Following is the general syntax of derived constructor.
- **The syntax of derived class constructor:**

```
derived constructor (arg1, arg2, ..... argn,    arg(D)):
                        base1(arg1),
                        base2(arg2),
                            .
                            .
                            .
                        basen(argn)
                        {
                          //body of derived constructor
                        }
```

- Here, base1 through basen are the names of base classes inherited by derived class. A colon separates the derived class constructor declaration from base_class specification and base class specifications are separated from each other by commas.
- In case of multiple base classes arg1, arg2 .....argn represents the actual parameters that are passed to the base constructor and arg(D) provides the parameters that are necessary to initialize the number of derived class.
- **For example:**

```
derived (int a, int b, int x)
base1(a);          //call constructor of base1
base2(b);          //call constructor of base2
{
   t = x;          //executes its own body
}
```

Program 5.18: Program for passing parameter to base constructor.

```
#include <iostream>
using namespace std;
class BaseClass1
{
   public:
      BaseClass1()
      {
```

```
        cout << "BaseClass1 constructor." << endl;
      }
};
class BaseClass2
{
    public:
      BaseClass2()
      {
        cout << "BaseClass2 constructor." << endl;
      }
};
class BaseClass3
{
    public:
      BaseClass3()
      {
        cout << "BaseClass3 constructor." << endl;
      }
};
class DerivedClass: public BaseClass1, public BaseClass2, public BaseClass3
{
    public:
      DerivedClass()
      {
        cout << "DerivedClass constructor." << endl;
      }
};
int main()
{
    DerivedClass dc;
}
```

Output:

```
BaseClass1 constructor.
BaseClass2 constructor.
BaseClass3 constructor.
DerivedClass constructor.
```

- In inheritance, destructors are executed in reverse order of constructor execution. The destructors are executed when an object goes out of scope.
- Base class constructors are called first and the derived class constructors are called next in single inheritance.

- Destructor is called in reverse sequence of constructor invocation. The destructor of the derived class is called first and the destructor of the base is called next.

Program 5.19: Program for constructor and destructors in derived classes.

```
#include<iostream>
using namespace std;
class base
{
    public:
     base()
     {
         cout<<"base class constructor"<<endl;
     } ~base()
     {
         cout<<"base class destructor"<<endl;
     }
};
class derived:public base
{
    public:
     derived()
     {
         cout<<"derived class constructor"<<endl;
     } ~derived()
     {
         cout<<"derived class destructor"<<endl;
     }
};
int main()
{
     derived d;
     return 0;
}
```

Output:

```
base class constructor
derived class constructor
derived class destructor
base class destructor
```

5.6.1 Constructors for Virtual Base Classes

- Constructors for virtual base classes are invoked before any non-virtual base class. For example,

```
class A: public B, virtual C
{
    .....
    .....
};
```

- Virtual base class C constructors first executed then B's constructor and then A's constructor.
- If the hierarchy contains multiple virtual base classes, then the virtual base class constructors are invoked in the order in which they are declared.

Table 5.3: Order of invocation of constructors

Method of Inheritance	Order of Execution

`class D: public B`
`{`
`  .....`
`};`
	B ( ): base constructor
	D ( ): derived constructor

`class D: public B1, public B2`
`{`
`  .....`
`};`
	B1 ( ): base constructor
	B2 ( ): base constructor
	D ( ): derived constructor

`class D: public B2, public B1`
`{`
`  .....`
`};`
	B2 ( ): base constructor
	B1 ( ): base constructor
	D ( ): derived constructor

`class D: public B1, virtual B2`
`{`
`  .....`
`};`
	B2 ( ): virtual base constructor
	B1 ( ): base constructor
	D ( ): derived constructor

`class D1: public B`
`{`
`  .....`
`};`
`class D2: public D1`
`{`
`  .....`
`};`
	B ( ): superbase constructor
	D1 ( ): base constructor
	D2 ( ): derived constructor

Summary

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.
- The technique by which features and properties of an old class is transferred into a new class is called inheritance or derivation. The old class is referred to as the base class or super class and new one is called the derived class or subclass.
- A derived class with only one base class is called single inheritance and one with several base classes is called multiple inheritance. A class may be inherited by more than one class is known as hierarchical inheritance.
- When a class is derived from another one of more base classes, this process is termed hybrid inheritance.
- The mechanism of deriving a class from another 'derived class' is known as multilevel inheritance.
- When classes are declared virtual, the compiler takes essential caution to avoid duplication of member variables. Thus, we make a class virtual if it is a base class that has been used by more than one derived class as their base class.
- When deriving a class from a base class, the base class may be inherited through public, protected or private.
- When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.
- When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class.
- A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.
- When deriving from a private base class, public and protected members of the base class become private members of the derived class.
- The execution of constructors takes place from base class to derived class. The execution of destructors is in opposite order as compared with constructors i.e., from derived class to base class.
- If more than one subclass is derived from the same base class, then the base class is called a virtual base class.
- A class containing a pure virtual function is called an abstract class. It is called abstract because we cannot define objects of a class containing a pure virtual function. It exists only for the purpose of defining classes that are derived from it.

Check Your Understanding

1. The process of creating new classes from an existing class is called as .....
   (a) polymorphism (b) polyinheritance
   (c) inheritance (d) none
2. If a base class is privately inherited by a derived class become ..... member to derived class.
   (a) public (b) private
   (c) visible (d) none
3. A class which is used only to derive other classes is known as ..... class.
   (a) abstract (b) virtual
   (c) derived (d) none.
4. Hybrid inheritance is .....
   (a) multiple inheritance (b) multilevel inheritance
   (c) multipath inheritance (d) both (a) & (b)
5. A class is inherited known as .....
   (a) base class (b) hybrid
   (c) derived class (d) both (a) & (b)
6. Following ..... operator is used for resolve ambiguity.
   (a) + (b) ::
   (c) - (d) none of these
7. Inheritance helps in making a general class into a more specific class.
   (a) True (b) False
8. Inheritance facilitates the creation of class libraries.
   (a) True (b) False
9. Defining a derived class requires some changes in the base class.
   (a) True (b) False
10. A base class is never used to create objects.
    (a) True (b) False

Answers

1. (c)	2. (b)	3. (a)	4. (d)	5. (a)	6. (b)	7. (b)	8. (a)	9. (b)	10. (b)

Practice Questions

Q.I Answer the following questions in short:

1. What is main advantage of Inheritance?
2. List out different types of inheritance.
3. What is the difference between the Base and Derived Classes?
4. Differentiate between Public and Private Inheritance.
5. What is a Virtual Base Class?

Q.II Answer the following questions:

1. Explain different types of Inheritance with example.
2. What are advantages and disadvantages of declaring Inheritance?
3. What is the scope of the accessibility of variables in the protected section and in the private section of a class?
4. When can Multiple Inheritance lead to ambiguity?
5. How is a Single Inheritance different from Multiple Inheritance?
3. When do we make a class Virtual?
4. Write short note on:
   (i) Multiple Inheritance
   (ii) Base Class and Derived Class
5. Can a Member Function in the Derived Class have the same name as that of one in the Base Class?

Q.III Define the term:

1. Inheritance
2. Single level inheritance
3. Multiple level inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

Previous Exam Questions

April 2018

1. What is an Abstract Class? **[2 M]**

Ans. Refer to Section 5.5.

2. Create a base class student(rollno, name) which derives two classes test(mark1, mark2), sport(score). Class result(total_marks, grade) inherits both test and sport classes. Write C++ program to calculate total_marks, percentage and to display marksheet. **[4 M]**

Ans. Refer to Program 5.9.

3. Trace output of the following program. Assume there is no error: **[4 M]**

```
#include<iostream>
using namespace std;
class P
{
    public:
    void print( )
    {
        cout<<"Inside P";
    }
};
```

```
class Q: public P
{
    public:
    void print( )
    {
       cout<<"Inside Q";
    }
};
class R: public Q
{
};
int main(void)
{
    R r;
    r.print( );
    return 0;
}
```

Output:

```
Inside Q
```

October 2018

1. What is inheritance? Explain its three types. **[4 M]**

Ans. Refer to Section 5.1, 5.3

2. Write a program which will implement the concept of virtual base class. **[4 M]**

Ans. Refer to Section 5.4

3. Trace the output of the following program and explain it. Assume there is no syntax error: **[4 M]**

```
#include <iostream>
using namespace std;
class Base1
{
    public:
    ~ Base1 ( )
    {
        cout << "Base1's destructor" << endl;
    }
};
class Base2
{
    public:
    ~ Base2 ( )
    {
        cout << "Base 2's destructor" << endl;
    }
};
```

```
class Derived: public Base1, public Base2
{
    public:
    ~ Derived ( )
    {
        cout << "Derived's destructor" << endl;
    }
};
int main ( )
{
    Derived d;
    return 0;
}
```

Output:

```
Derived's destructor
Base 2's destructor
Base1's destructor
```

April 2019

1. What is inheritance? Explain it with its types. [4 M]

Ans. Refer to Section 5.1, 5.3

2. Explain virtual base class with suitable diagram. [4 M]

Ans. Refer to Section 5.4

3. Design a C++ program to create two base classes personnel (name, address, e-mail-id, DOB) and academic (10th std marks, 12th std marks, class obtained). Derive a class Bio_data from both these classes and prepare a bio data of a student having personal and academic information. [4 M]

Ans. Refer to Program 5.11

4. Trace the output of the following program and explain it. Assume there is no syntax error: [4 M]

```
#include <iostream>
using namespace std;
class Base1
{
    public:
    Base1()
    {
        cout << "Base1's constructor called" << endl;
    }
};
```

```
class Base2
{
    public:
    Base2()
    {
        cout << "Base2's constructor called" << endl;
    }
};
class Derived: public Base2, public Base1
{
    public:
    Derived()
    {
        cout << "Derived's constructor called" <<endl;
    }
};
int main()
{
    Derived d;
    return 0;
}
```

Output:

```
Base2's constructor called
Base1's constructor called
Derived's constructor called
```