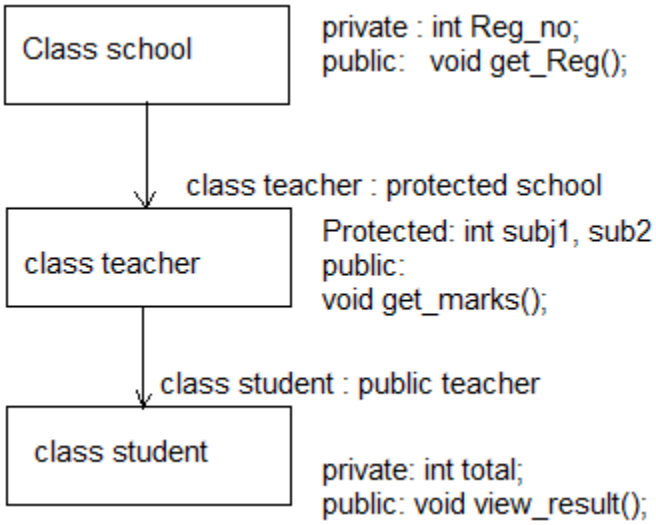
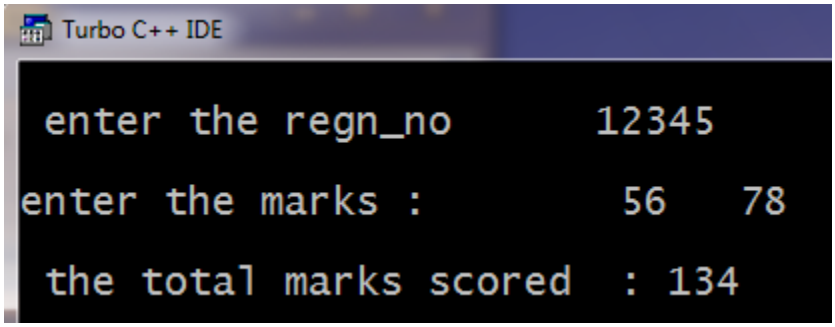
Multilevel Inheritance



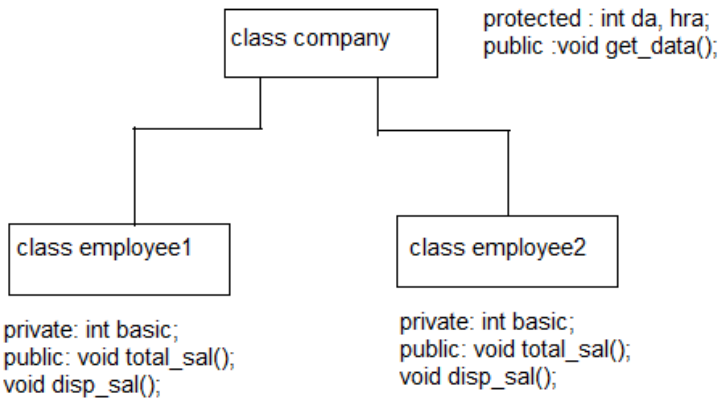

```
#include<iostream.h>
#include<conio.h>
class school
{ int regn_no;
public:
int sub1, sub2;
void get_regn_no()
{
            cout<<"\n enter the
           regn_no\t";
            cin >> regn_no;
}
};
class teacher : protected school
{
 int total;
 public:
 void get_marks()
 {
  get_regn_no();
              cout <<"\nenter the marks : \t ";
              cin >> sub1>> sub2;
 }
int total_marks()
{
            total = sub1 + sub2;
            return total;
}
};
class student : public teacher
{
public:
void see_marks()
{
            get_marks();
            cout <<"\n the total marks scored  : "<<total_marks();
}
};
void main()
{
            clrscr();
            student std;
            std.see_marks();
            getch();

};
```

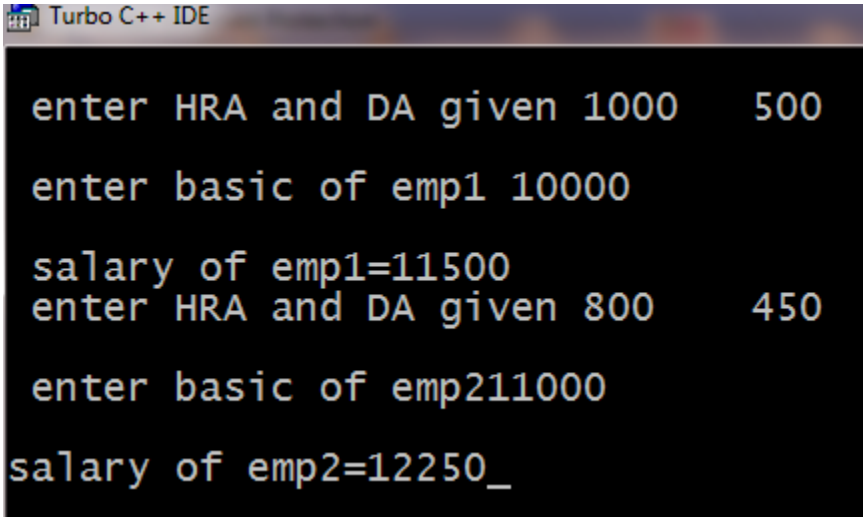


Hierarchical Inheritance

```
#include<iostream.h>
#include<conio.h>
class company
{
protected:
int hra, da;
public:
void get_data()
{
cout<<"\n enter HRA and DA given ";
cin>> hra >> da;
}
};
class employee1: public company
{
int total, basic;
public:
void get_basic()
{
get_data();
cout<<"\n enter basic of emp1 ";
cin>>basic;
}
void total_salary();
void disp_sal1();
};
void employee1::total_salary()
{
      total = basic + hra + da;
}
```

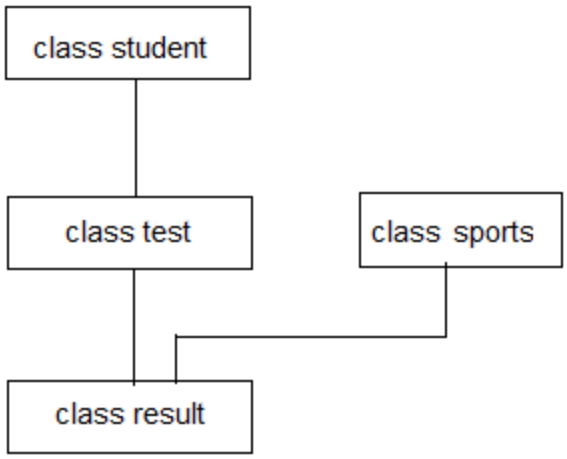


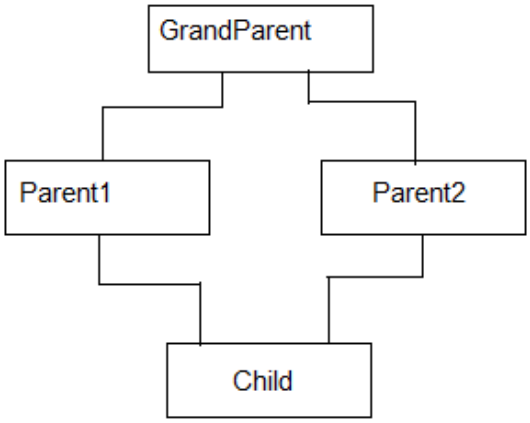
```
void employee1:: disp_sal1()
{
      cout<<"\n salary of emp1="<<total;
}
class employee2: public company
{
          int total, basic;
public:
void get_basic()
{
          get_data();
          cout<<"\n enter basic of emp2";
          cin>>basic;
}
          void total_salary();
          void disp_sal();
};
void employee2::total_salary()
{
    total = basic +hra + da;
}
void employee2::disp_sal()
{
  cout<<"\nsalary of emp2="<<total;
}
void main()
{
          clrscr();
          employee1 obj1;
          obj1.get_basic();
          obj1.total_salary();
          obj1.disp_sal1();
          employee2 obj2;
          obj2.get_basic();
          obj2.total_salary();
          obj2.disp_sal();
}
```



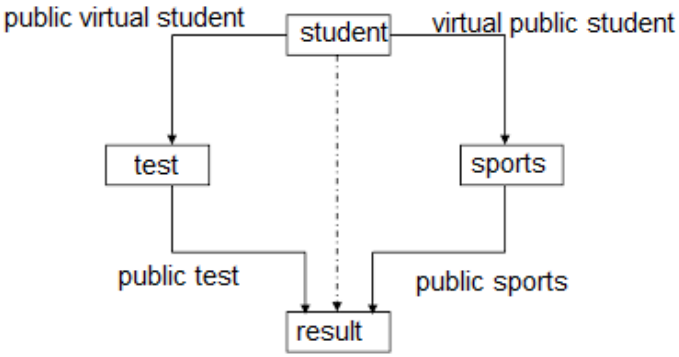
Hybrid inheritance

This type of inheritance employs more than one type of inheritance.





Ex-1



Ex-2

Virtual Base Class

In figure ex-1 the child class has two direct base classes parent1 and parent2, which themselves are the child of a common base class called class grandparent. In this case the child class gets two traits of the grandparent class, one through paren1 and another through parent2. This type of inheritance may pose some problems as all the public and protected member of grandparent are inherited into the child class twice; thus would have duplicate members inherited from grandparents thus introducing ambiguity in class/references and must be avoided.
To overcome this problem C++ offers the concept of virtual base class as shown in figure ex-2. Once the base class is inherited as virtual, then C++ ensure that only one copy of the members are visible to the child classes no matter how many inherited path exists.

Example-1: without virtual:

```
#include<iostream.h>
#include<string.h>
#include<conio.h>
class student
{
protected:
```

```
int regn_no;
public:
void get_regn_no()
{
        cout<<”\nenter regn no”;
        cin>> regn_no;
}
};
class test : public student
{
protected:
int sub1, sub2;
public:
void get_marks()
{
         cout<<"\n enter marks of sub1, and sub2 : ";
         cin >>sub1>>sub2;
}
void disp()
{
        cout<<"\nthe marks obtained are :"<<sub1<<" and "<<sub2<<" total="<<sub1+sub2;
}
};

class sports : public student

{
  protected:
  int score;
  public:
void get_score()
 {
         cout<<”\n enter score;
        cin>>score;
 }
 };
 class result : public test, public sports
  {
   public:
int result;
   void disp()
 {
        get_regn_no();
        get_marks();
         get_score();
        result=marks1+marks2+score;

        cout<<”\nmarks obtained are:”;
```

```
        disp();
        cout <<"\n the overall result of regn_no”<<regn_no<< “  is = "<<result;

 }

};
void main()
{
        clrscr();
        result R;
        R.disp();
        getch();
}
```

//On compilation the system will thow errors as ambiguous function get_regn_no();
Beacuase it gets duplicated in class result.
To overcome this problem we inherit the base class as virtual class in parents class so that the compiler sends only one copy of the regn_no.

Ex-2 With virual

```
#include<iostream.h>
#include<string.h>
#include<conio.h>
class student
{
protected:
int regn_no;
public:
int get_regn_no(int r_no)
{
        regn_no=r_no;
        return regn_no;
}
};
class test : virtual public student
{
protected:
int sub1, sub2;
public:
void get_marks()
{
        cout<<"\n enter marks of sub1, and sub2 : ";
        cin >>sub1>>sub2;
}
```

```
void disp()
{
        cout<<"\nthe marks obtained are :"<<sub1<<" and "<<sub2<<" total="<<sub1+sub2;
}
};

class sports : public virtual student

{
 protected:
 int score;
 public:
void get_data(int sc)
 {
        score=sc;
 }
 };
 class result : public test, public sports
  {
   public:
   void disp()
 {
          cout<< "\n reg_no ="<<regn_no;
          cout<<"\n the score = "<<score;
          cout <<"\n the overall result = "<<sub1+sub2+score;
  }

};
void main()
{
        clrscr();
        result R;
        R.get_regn_no(1234);
        R.get_data(200);
        R.disp();
        getch();
}
```

Constructors in derived class

In the above examples we have not used the constructors neither in base class nor in derived class. It should be understood that as long as a base class constructor take no argument, the derived class need not define a constructor function. If however, the base class contain constructor with one or more argument, then it is mandatory to have a constructor in the derived class.

When using constructor in base class and hence in derived class, the constructor of the derived class will receive all the argument of base class constructor and its own and pass the argument to the base class constructor. The values are passed in the order in which the class name appear in the visibility mode list.

The syntax of the derived class constructor with all the argument of(base and derived class) is:

```
Derived_class_name(arg1, arg2,……..argN)
Base1_class(arg1);
Base2-class(arg2)
--
{
  //body of derive class constructor
}
```

Here is an example of the declaring a constructor in base and derived class