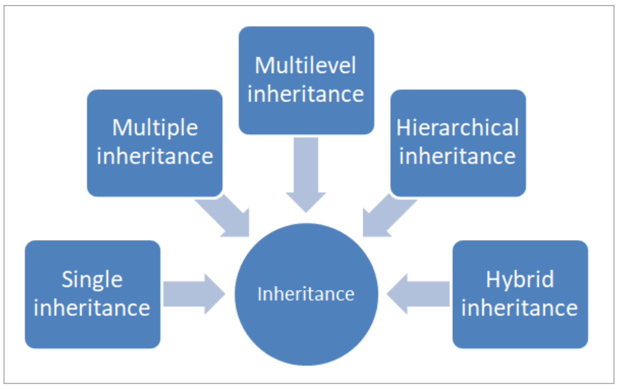
Levels of Inheritance

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Types Of Inheritance

Given below is a pictorial representation of the various types of inheritance.



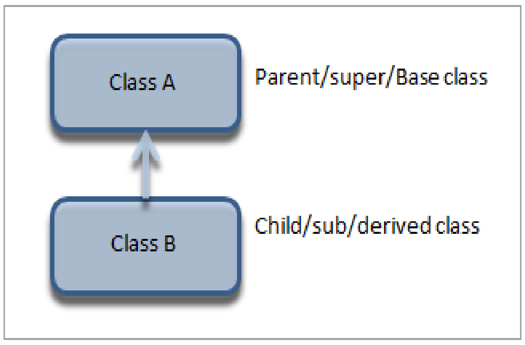
We will see each type of inheritance with examples in the below sections.

#1) Single Inheritance

In single inheritance, a class derives from one base class only. This means that there is only one subclass that is derived from one superclass.

Single inheritance is usually declared as follows:

```
class subclassname : accessspecifier superclassname {
            //class specific code;
};
```



<u>Given below is a complete Example of Single Inheritance.</u>

```
#include <iostream>
#include <string>
using namespace std;
class Animal
{
  string name="";
  public:
  int tail=1;
  int legs=4;

};
class Dog : public Animal
{
  public:
  void voiceAction()
  {
    cout<<"Barks!!!";
  }
};
int main()
{
  Dog dog;
  cout<<"Dog has "<<dog.legs<<" legs"<<endl;
  cout<<"Dog has "<<dog.tail<<" tail"<<endl;
  cout<<"Dog ";
  dog.voiceAction();
}
```

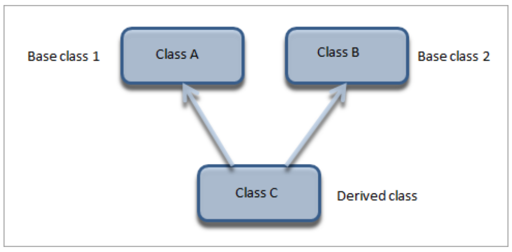
Output:

```
Dog has 4 legs
Dog has 1 tail
Dog Barks!!!
```

We have a class Animal as a base class from which we have derived a subclass dog. Class dog inherits all the members of the Animal class and can be extended to include its own properties, as seen from the output.

Single inheritance is the simplest form of inheritance.

#2) Multiple Inheritance



We present the below program to demonstrate Multiple Inheritance.

```
#include <iostream>
using namespace std;
//multiple inheritance example
class student_marks {
protected:
int rollNo, marks1, marks2;
public:
void get() {
cout << "Enter the Roll No.: "; cin >> rollNo;
cout << "Enter the two highest marks: "; cin >> marks1 >> marks2;
  }
};
class cocurricular_marks {
protected:
int comarks;
public:
void getsm() {
cout << "Enter the mark for CoCurricular Activities: "; cin >> comarks;
  }
};

//Result is a combination of subject_marks and cocurricular activities marks
class Result : public student_marks, public cocurricular_marks {
  int total_marks, avg_marks;
  public:
  void display()
  {
    total_marks = (marks1 + marks2 + comarks);
    avg_marks = total_marks / 3;
    cout << "\nRoll No: " << rollNo << "\nTotal marks: " << total_marks;
    cout << "\nAverage marks: " << avg_marks;
  }
};
int main()
{
  Result res;
res.get(); //read subject marks
res.getsm(); //read cocurricular activities marks
res.display(); //display the total marks and average marks
}
```

Output:

```
Enter the Roll No.: 25
Enter the two highest marks: 40 50
Enter the mark for CoCurricular Activities: 30Roll No: 25
```

```
Total marks: 120
Average marks: 40
```

In the above example, we have three classes i.e. student_marks, cocurricular_marks, and Result. The class student_marks reads the subject mark for the student. The class cocurricular_marks reads the student's marks in co-curricular activities.

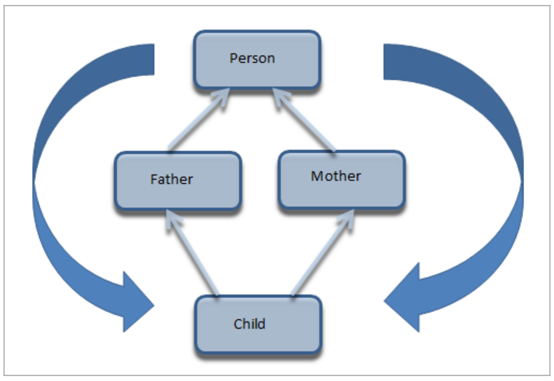
The Result class calculates the total_marks for the student along with the average marks.

In this model, the Result class is derived from student_marks and cocurricular_marks as we calculate results from the subject as well as co-curricular activities marks.

This exhibits multiple inheritances.

Diamond problem

Diamond Problem is pictorially represented below:



Here, we have a child class inheriting two classes Father and Mother. These two classes, in turn, inherit the class Person.

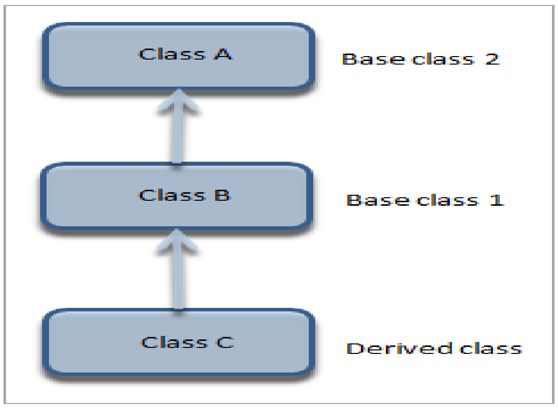
As shown in the figure, a class Child inherits the traits of a class Person twice i.e. once from Father and the second time from Mother. This gives rise to ambiguity as the compiler fails to understand which way to go.

Since this scenario arises when we have a diamond-shaped inheritance, this problem is famously called "**The Diamond Problem**".

The Diamond problem implemented in C++ results in ambiguity error at compilation. We can resolve this problem by making the root base class virtual. We will learn more about the "virtual" keyword in our upcoming tutorial on polymorphism.

#3) Multilevel Inheritance

Multilevel inheritance is represented below.



In multilevel inheritance, a class is derived from another derived class. This inheritance can have as many levels as long as our implementation doesn't go wayward. In the above diagram, class C is derived from Class B. Class B is in turn derived from class A.

Let us see an example of Multilevel Inheritance.

```
#include <iostream>
#include <string>
using namespace std;
class Animal
{
  string name="";
  public:
  int tail=1;
  int legs=4;

};
class Dog : public Animal
{
  public:
  void voiceAction()
  {
    cout<<"Barks!!!";
  }
};
class Puppy:public Dog{
  public:
  void weeping()
  {
    cout<<"Weeps!!";
  }
};
int main()
{
  Puppy puppy;
cout<<"Puppy has "<<puppy.legs<<" legs"<<endl;
cout<<"Puppy has "<<puppy.tail<<" tail"<<endl;
cout<<"Puppy ";
puppy.voiceAction();
cout<<" Puppy ";
puppy.weeping();
}
```

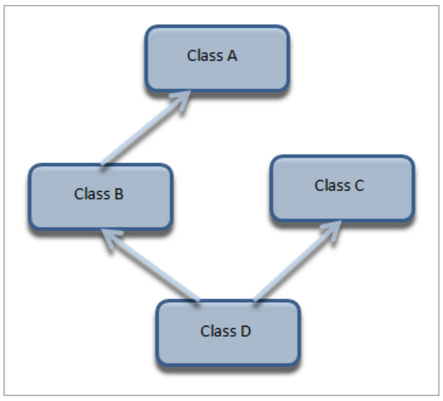
Output:

```
Puppy has 4 legs
Puppy has 1 tail
Puppy Barks!!! Puppy Weeps!!
```

Here we modified the example for Single inheritance such that there is a new class Puppy which inherits from the class Dog that in turn inherits from class Animal. We see that the class Puppy acquires and uses the properties and methods of both the classes above it.

#4) Hybrid Inheritance

Hybrid inheritance is depicted below.



Hybrid inheritance is usually a combination of more than one type of inheritance. In the above representation, we have multiple inheritance (B, C, and D) and multilevel inheritance (A, B, and D) to get a hybrid inheritance.

<u>Let us see an example of Hybrid Inheritance.</u>

```
#include <iostream>
#include <string>
using namespace std;
//Hybrid inheritance = multilevel + multilpe
class student{ //First base Class
            int id;
            string name;
            public:
            void getstudent(){
                        cout << "Enter student Id and student name"; cin >> id >> name;
  }
};
```

```
class marks: public student{ //derived from student
                protected:
                int marks_math,marks_phy,marks_chem;
                public:
                void getmarks(){
                               cout << "Enter 3 subject marks:"; cin >>marks_math>>marks_phy>>marks_chem;
   }
};
class sports{
              protected:
              int spmarks;
              public:
              void getsports(){
                              cout << "Enter sports marks:"; cin >> spmarks;
   }
};
class result : public marks, public sports{//Derived class by multiple inheritance//
                int total_marks;
                float avg_marks;
                public :
               void display(){
                             total_marks=marks_math+marks_phy+marks_chem;
                             avg_marks=total_marks/3.0;

                           cout << "Total marks =" << total_marks << endl;
                           cout << "Average marks =" << avg_marks << endl;
                           cout << "Average + Sports marks =" << avg_marks+spmarks;
   }
};


int main(){
                result res;//object//
                res.getstudent();
                res.getmarks();
                res.getsports();
                res.display();
               return 0;
}
```

Output:

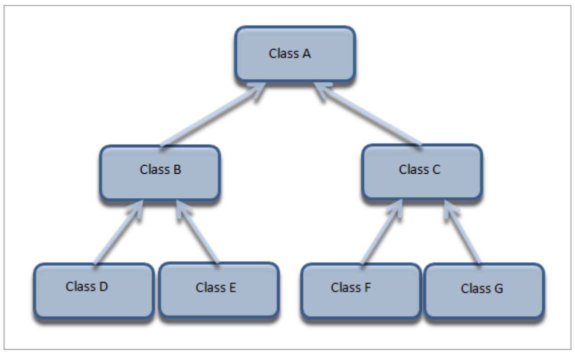
```
Enter student Id and student name 25 Ved
Enter 3 subject marks:89 88 87
Enter sports marks:40
Total marks =264
Average marks =88
Average + Sports marks =128
```

Here we have four classes i.e. Student, Marks, Sports, and Result. Marks are derived from the student class. The class Result derives from Marks and Sports as we calculate the result from the subject marks as well as sports marks.

The output is generated by creating an object of class Result that has acquired the properties of all three classes.

Note that in hybrid inheritance as well, the implementation may result in “Diamond Problem” which can be resolved using “virtual” keyword as mentioned previously.

#5) Hierarchical Inheritance



In hierarchical inheritance, more than one class inherits from a single base class as shown in the representation above. This gives it a structure of a hierarchy.

Given below is the Example demonstrating Hierarchical Inheritance.

```
#include <iostream>
using namespace std;
//hierarchical inheritance example
class Shape                    // shape class -> base class
{
public:
int x,y;

void get_data(int n,int m) {
     x= n;
     y = m;
   }
};
class Rectangle : public Shape // inherit Shape class
{
public:
int area_rect() {
int area = x*y;
return area;
   }
};
class Triangle : public Shape // inherit Shape class
```

```
{
public:
int triangle_area() {
float area = 0.5*x*y;
return area;
  }
};
class Square : public Shape // inherit Shape class
{
public:
int square_area() {
float area = 4*x;
return area;
  }
};
int main()
{ Rectangle r;
  Triangle t;
  Square s;
  int length,breadth,base,height,side;
  //area of a Rectangle
  std::cout << "Enter the length and breadth of a rectangle: "; cin>>length>>breadth;
  r.get_data(length,breadth);
  int rect_area = r.area_rect();
  std::cout << "Area of the rectangle = " <<rect_area<< std::endl;
  //area of a triangle
  std::cout << "Enter the base and height of the triangle: "; cin>>base>>height;
  t.get_data(base,height);
  float tri_area = t.triangle_area();
  std::cout <<"Area of the triangle = " << tri_area<<std::endl;
  //area of a Square
  std::cout << "Enter the length of one side of the square: "; cin>>side;
  s.get_data(side,side);
  int sq_area = s.square_area();
  std::cout <<"Area of the square = " << sq_area<<std::endl;
  return 0;
}
```

Output:

```
Enter the length and breadth of a rectangle: 10 5
Area of the rectangle = 50
Enter the base and height of the triangle: 4 8
Area of the triangle = 16
Enter the length of one side of the square: 5
Area of the square = 20
```

The above example is a classic example of class Shape. We have a base class Shape and three classes i.e. rectangle, triangle, and square are derived from it.

We have a method to read data in the Shape class while each derived class has its own method to calculate area. In the main function, we read data for each object and then calculate the area.