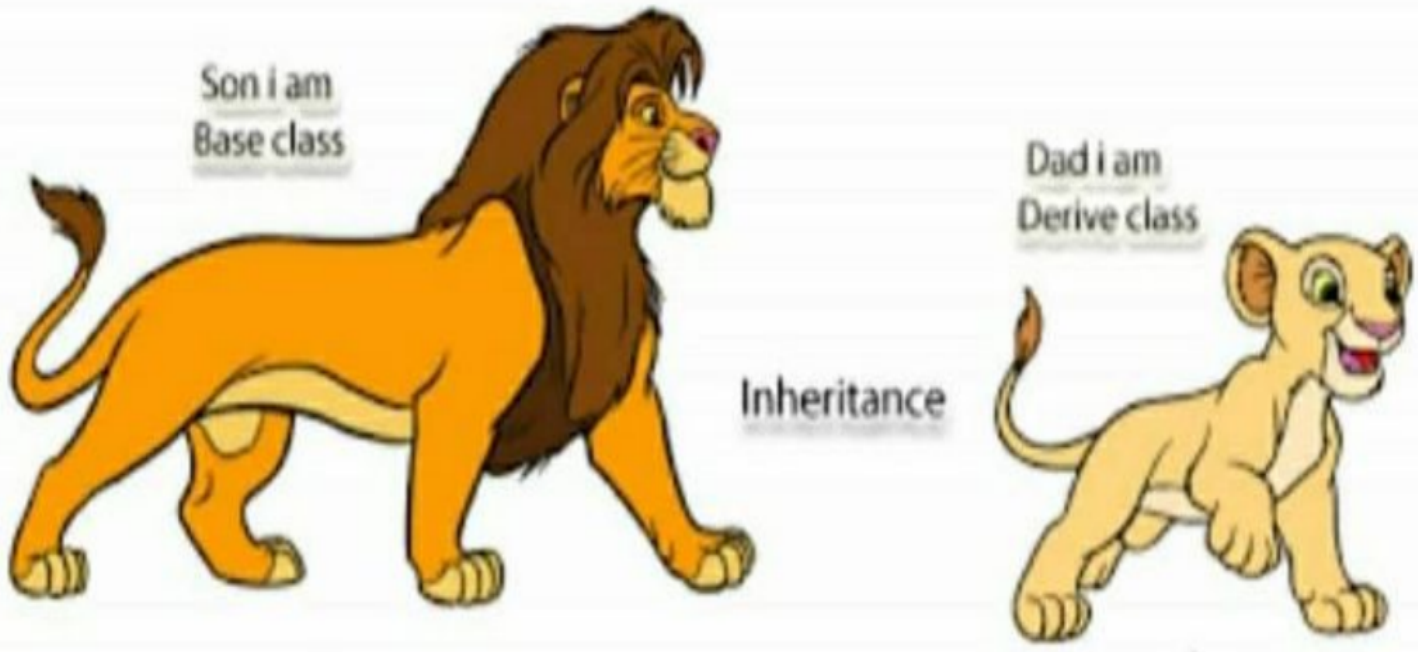
Unit-3

Operator overloading,

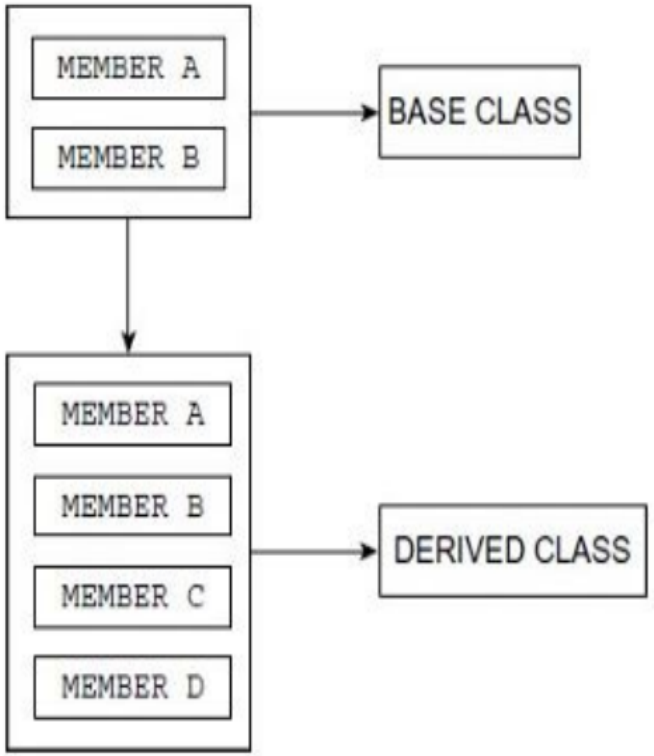
Inheritance



Inheritance is a mechanism in which one class acquires the property of another class.

C++ Inheritance

Inheritance is the process by which new classes called *derived* classes are created from existing classes called *base* classes.



The derived classes have all the features of the base class and the programmer can choose to add new features specific to the newly created derived class.

C++ Inheritance

Features or Advantages of Inheritance:

Reusability:

✔Inheritance helps the code to be reused in many situations.

✔The base class is defined and once it is compiled, it need not be reworked.

✔Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

C++ Inheritance

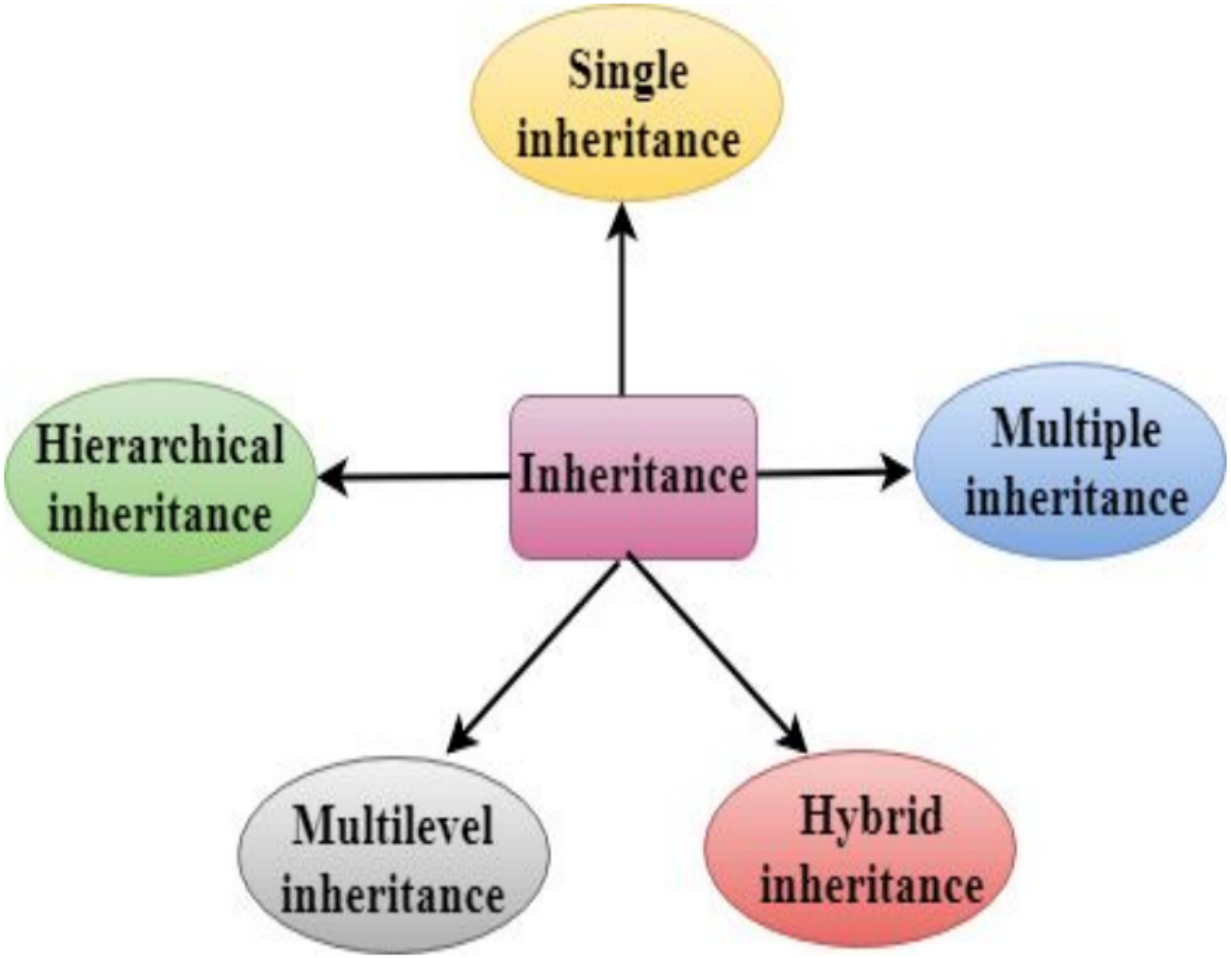
Features or Advantages of Inheritance:

✔***Saves Time and Effort:***

The above concept of reusability achieved by inheritance saves the programmer time and effort. The main code written can be reused in various situations as needed.

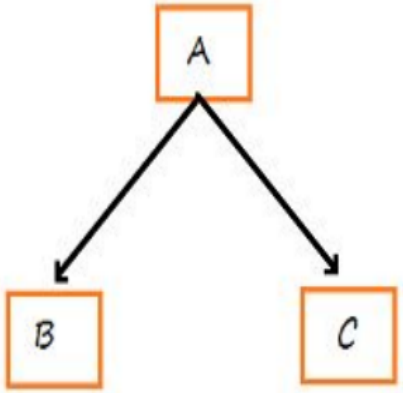
✔**Increases Program Structure which results in greater reliability.**

C++ Inheritance Types

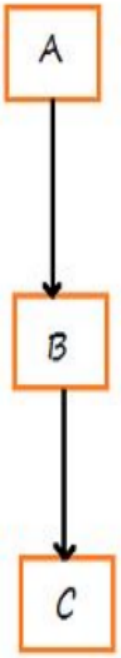


C++ Inheritance Types

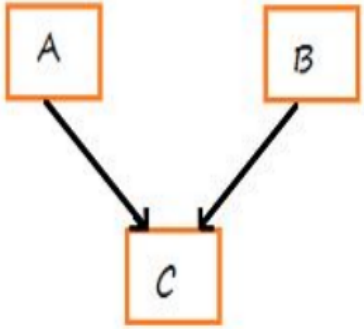
Single Inheritance



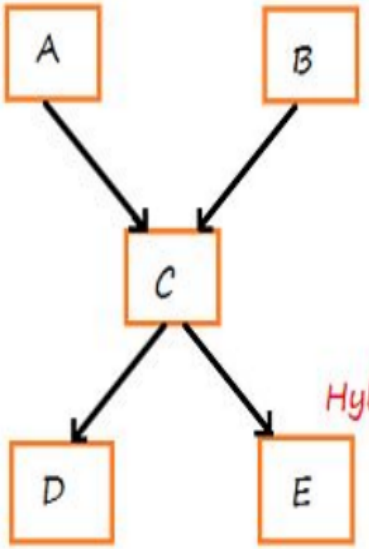
Heirarchical Inheritance



Multilevel Inheritance



Multiple Inheritance



Hybrid Inheritance

Defining derived Class:

- A derived class can be defined by specifying its relationship with the base in addition to its own details.
- The general form of defining a derived class is

```
class derived-class-name : visibility-mode  base-class-name
{
……..
……..
……. // members of derived class
……..
};
```

Single Level Inheritance Example

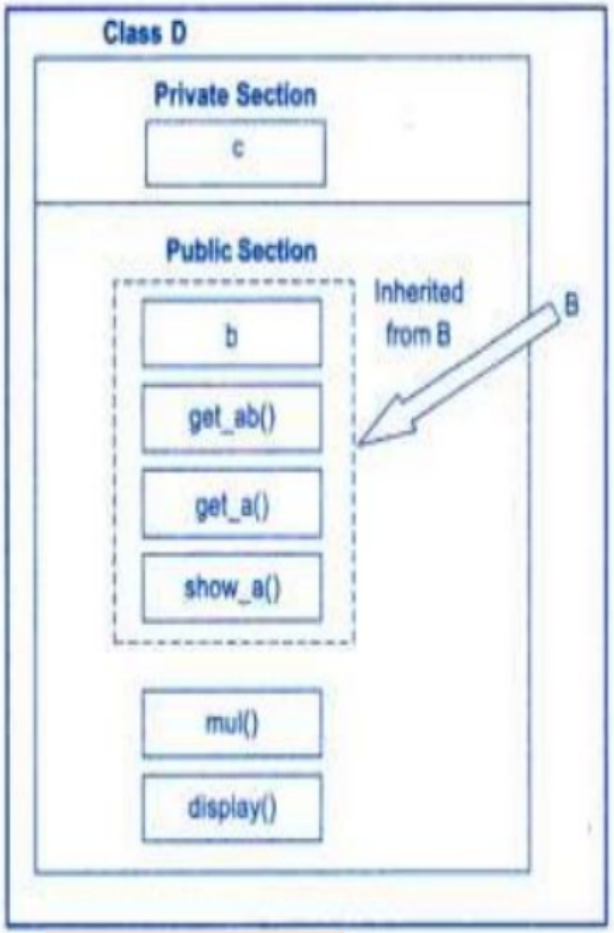
```
#include <iostream>
using namespace std;
 class Animal {
   public:
 void eat() {
    cout<<"Eating..."<<endl;
    }
  };
```

```
class Dog: public Animal
  {
     public:
     void bark(){
     cout<<"Barking...";
    }
  };
int main(void) {
   Dog d1;
   d1.eat();
   d1.bark();
   return 0;
}
```

Single Inheritance: Public

```
#include<iostream>
using namespace std;
class B
{
   int a; // private its not inherit
   public:
      int b; // public its ready for inherit
      void get_ab();
      int get_A();
      void show_A();
};

void B::get_ab(){
   a=8; b=90;
}
int B:: get_A() {   return a;}

void B::show_A(){
   cout << "a=" << a << "\n";
}
```

```
class D: public B //public derivation
{
    int c;
public:
   void mul();
   void display();
};

void D::mul(){ c=b*get_A(); }

void D::display()
{
   cout << "a=" << get_A() << "\n";
   cout << "b=" << b << "\n";
   cout << "c=" << c << "\n";
}
int main()
{
D d;
d.get_ab();
d.mul();
d.show_A();
d.display();

d.b=20;
d.mul();
d.display();
}
```

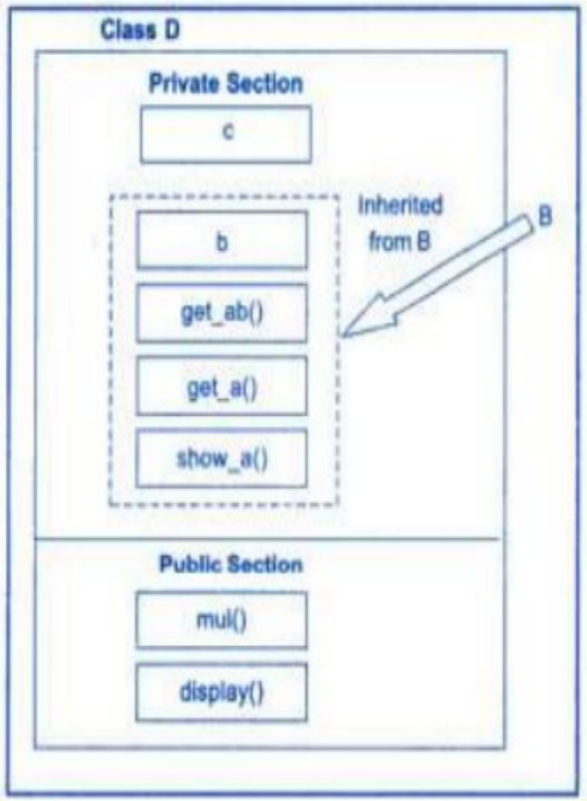



```
a=8

a=8
b=90
c=720

a=8
b=20
c=160
```

Single Inheritance: Private

```
#include<iostream>
using namespace std;
class B
{
   int a; // private its not inherit
   public:
      int b; // public its ready for inherit
      void get_ab();
      int get_A();
      void show_A();
};

void B::get_ab(){
   a=8; b=90;
}
int B:: get_A() {   return a;}

void B::show_A(){
   cout << "a=" << a << "\n";
}
```

```
class D: private B //private derivation
{
    int c;
public:
   void mul();
   void display();
};

void D::mul(){
    get_ab();
    c=b*get_A(); }

void D::display()
{
   cout << "a=" << get_A() << "\n";
   cout << "b=" << b << "\n";
   cout << "c=" << c << "\n";
}
```




```
int main()
{
   D d;
   //d.get_ab(); Wont work
   d.mul();
   //d.show_A(); Wont work
   d.display();

   //d.b=20; Wont work
   d.mul();
   d.display();
}
```

Making a private member inheritable:

How to inherit a private data by a derived class?

- C++ provides a third, protected, visibility modifier for restricted inheritance use.
- A member declared protected is accessible by the functions of the member in its class and any class immediately deriving from it.
- It is not accessible by functions outside of both classes.

```
Class a
{
 private:
     int a;          // Optional, visible to member functions within its class
     - - - -
 Protected:
      int b;              // visible to member functions of its own and
     - - -          // that of derived class
 public:
      - - -  // visible to all member functions in the program
}
```

```
class ABC: private XYZ      // private derivation
{
members of ABC
};

class ABC: public XYZ // its public derivation
{
members of ABC
};

class ABC: XYZ            // its private derivation by default
{
members of ABC
};
```

C++ Inheritance

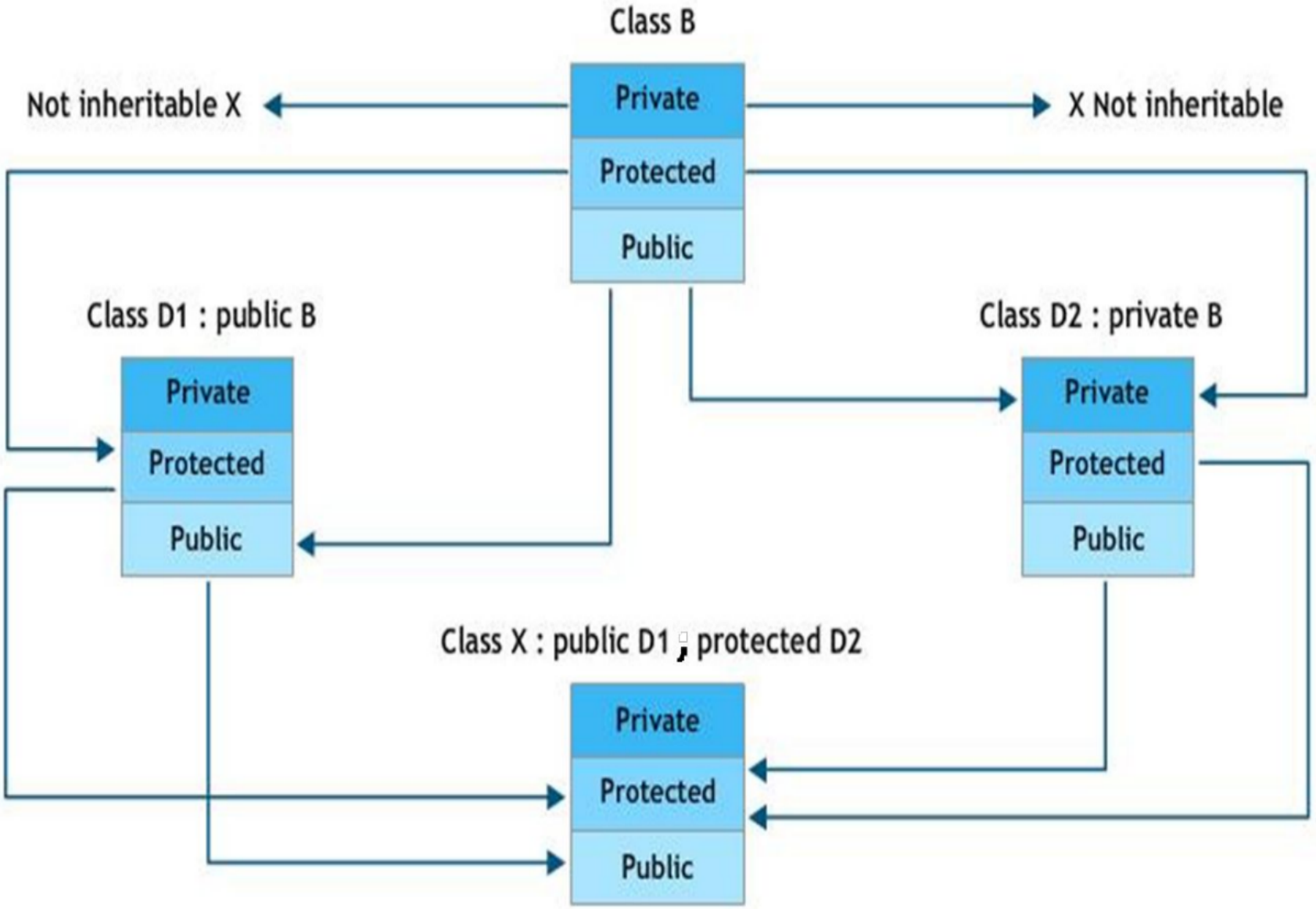
About public, private and protected access specifiers:

1. If a member or variables defined in a class is private, then they are accessible by members of the same class only and cannot be accessed from outside the class.

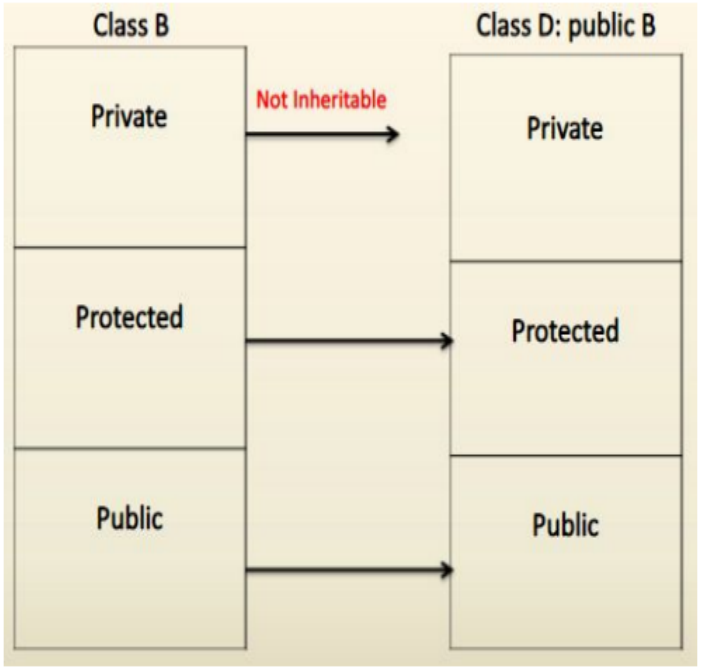
2. Public members and variables are accessible from outside the class.

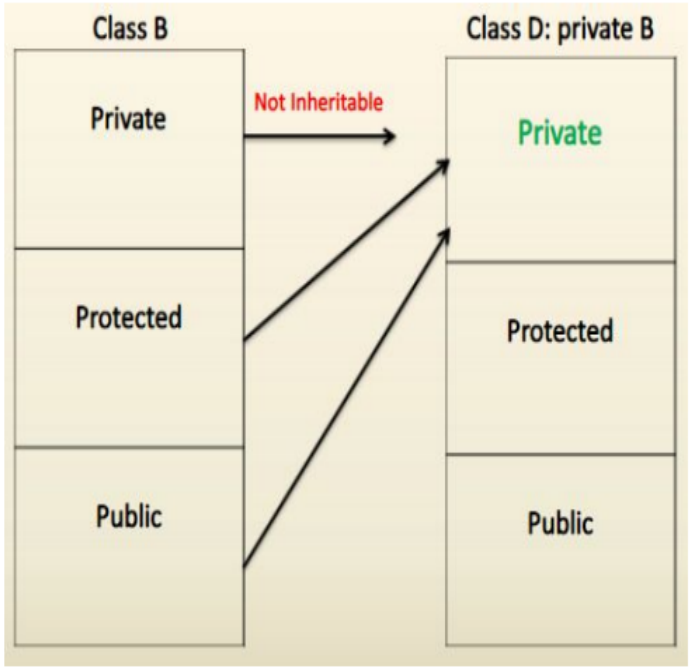
3. Protected access specifier is a stage between private and public. If a member functions or variables defined in a class are protected, then they cannot be accessed from outside the class but can be accessed from the derived class.

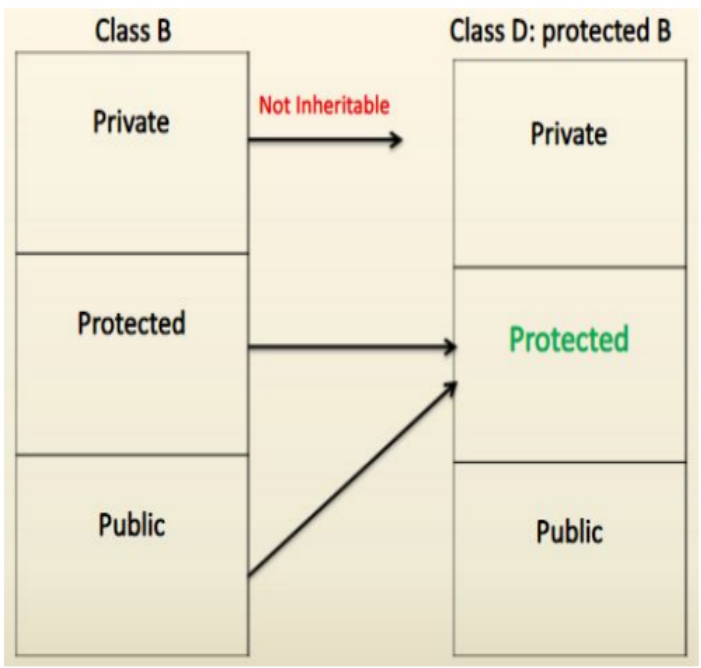
Effects of Inheritance on Members Visibility



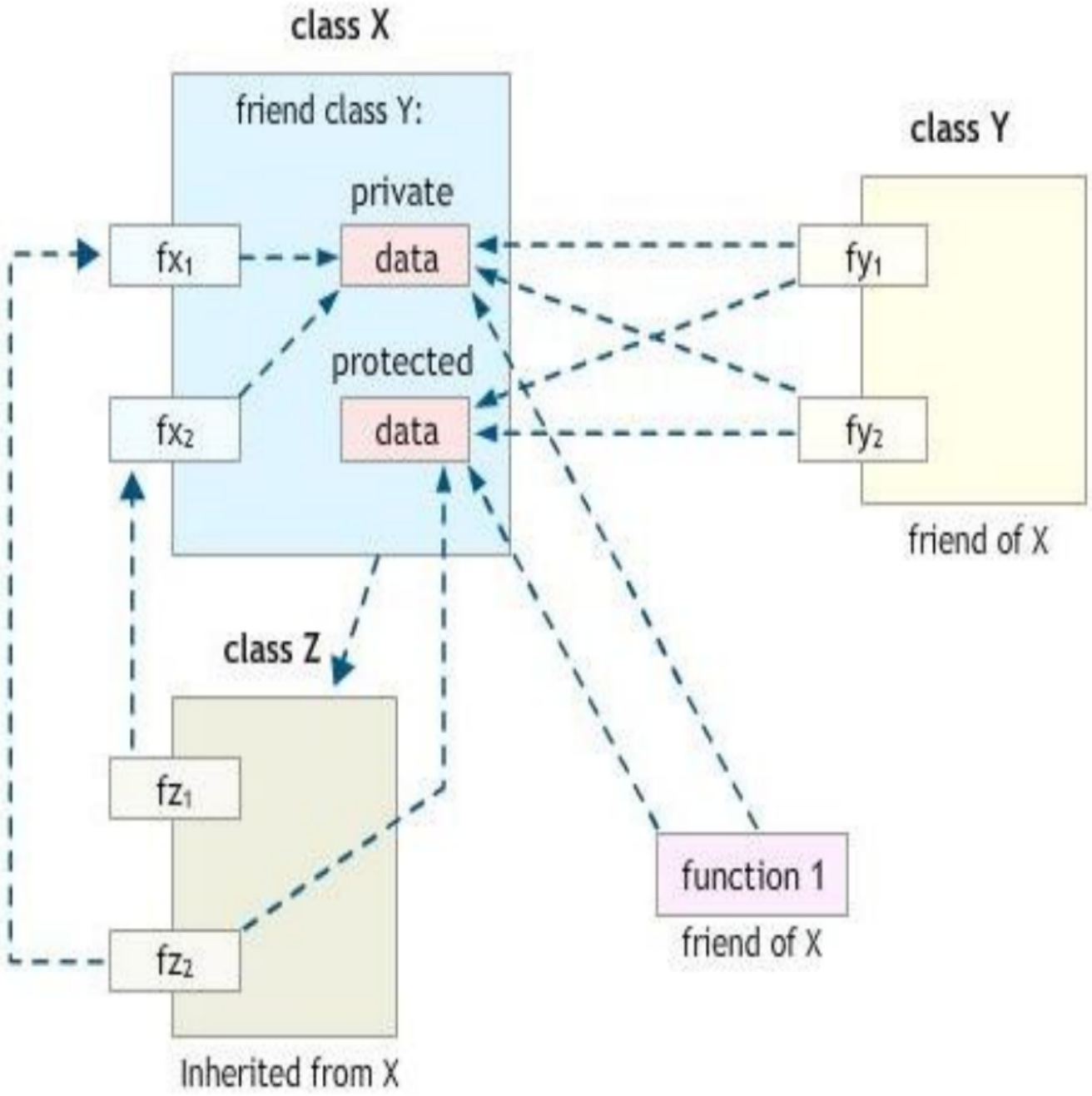
visibility of inherited members







Base class visibility	Derived class visibility		
	Public	Private	Protected
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

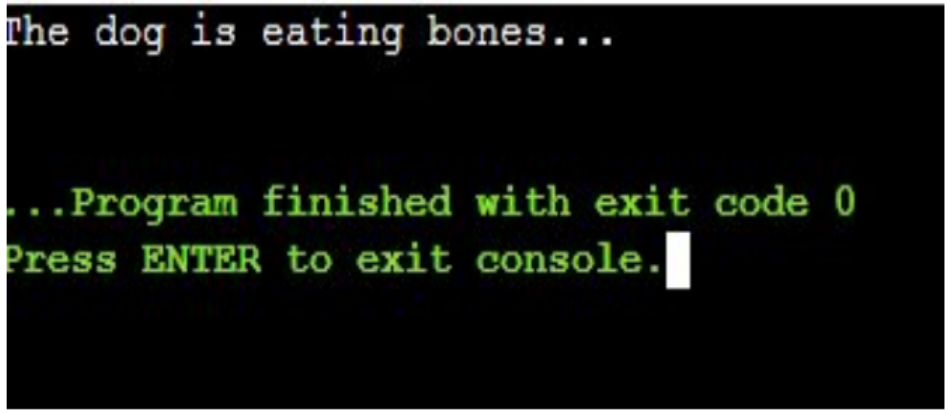

class X
friend class Y:
private
data
protected
data
fx1
fx2
class Y
fy1
fy2
friend of X
class Z
fz1
fz2
Inherited from X
function 1
friend of X

Overriding

- If derived class defines same function as defined in its base class, it is known as function overriding in C++.
- It is used to achieve runtime polymorphism.
- It involves providing a new implementation for a method in a subclass that is already present in one of its superclasses.
- When a subclass overrides a method, it provides its own implementation of that method, which replaces the implementation inherited from the superclass.

```cpp
#include <iostream>
// Base class
class Animal {
public:
    // Virtual function eat
    virtual void eat() {
        std::cout << "The animal is eating..." << std::endl;
    }
};

// Derived class
class Dog : public Animal {
public:
    // Overriding the eat function of the base class
    void eat() override {
        std::cout << "The dog is eating bones..." <<
std::endl;
    }
};
```

```cpp
int main() {
    // Create an instance of the derived class
    Dog dog;

    // Call the overridden eat function
    dog.eat();

    return 0;
}
```



Ambiguity resolution in inheritance

- Function with the same inheritance appears in more than one base class

```
class A
{
public:
void display()
{
    cout<<"A\n";
};
class B: public A
{
public:
void display()
{
    cout<<"B\n";
}
};
```

```
void main()
{
B b; // derived class object
b.display(); //invokes display() in B
b.A::display(); // invokes display() in A
b.B::display(); // invokes display() in B
}
```

Ambiguity resolution in inheritance

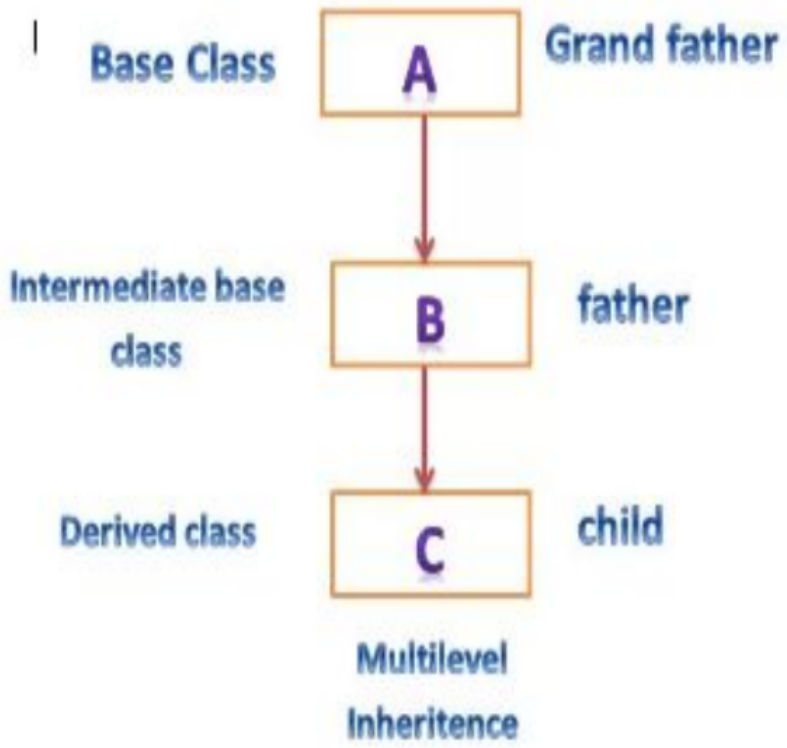
```
class M
{
public:
void display()
{
    cout << "class M\n";
}
};

class N
{
public:
void display()
{
    cout << "class N\n";
}
};
```

```
class P: public M, public N
{
public:
void display()
{
M::display();

}
};

//we can now use the derived class as follows
int main()
{
P p;
p.display();
}
```

Multilevel inheritance

- The class A is a base(super) class for the derived(child) class B , which in turn serves as a base class for the derived class C.
- The class B is know as intermediate base class since it provides a link for the inheritance between A and C
- The chain ABC is know as inheritance path

```
class A // Base class
{
      .....
};
class B: public A
{
      .....  //B derived from A
}
class C: public B
{
      ..... // C derived from B
}
```



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

class student {
protected:
    int roll;

public:
    void get_number(int);
    void put_number();
};

void student::get_number(int a) {
    roll = a;
}

void student::put_number() {
    cout << "Roll number: " << roll << "\n";
}
```

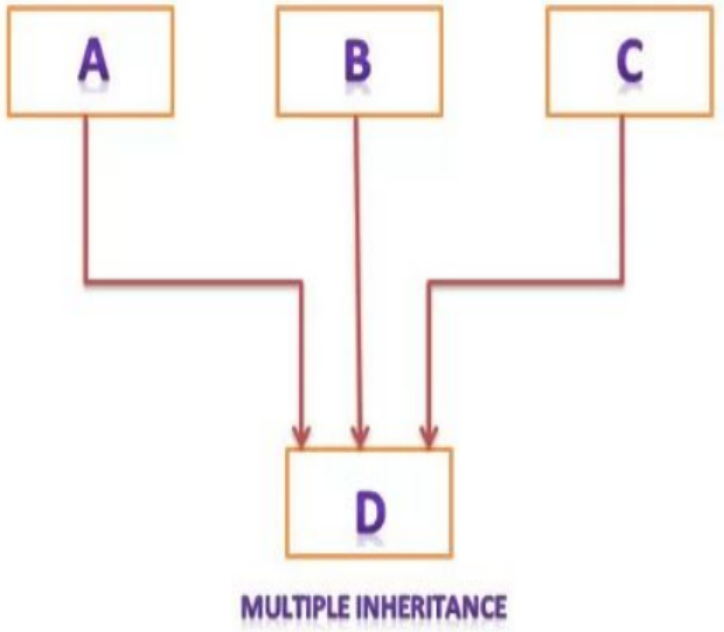
```cpp
class test : public student {
protected:
    float sub1;
    float sub2;

public:
    void get_marks(float, float);
    void put_marks();
};

void test::get_marks(float x, float y) {
    sub1 = x;
    sub2 = y;
}

void test::put_marks() {
    cout << "Marks in sub1: " << sub1 << "\n";
    cout << "Marks in sub2: " << sub2 << "\n";
}
```

```cpp
class result : public test {
    float total;

public:
    void display();
};

void result::display() {
    total = sub1 + sub2;
    put_number();
    put_marks();
    cout << "Total: " << total << "\n";
}

int main() {
    result student1;
    student1.get_number(111);
    student1.get_marks(80.8, 78.5);
    student1.display();

    return 0;
}
```

Multiple inheritance

- Multiple inheritance allows users to combine the features of several existing classes as a starting point for defining new class.
- It is like a child(subclass) inheriting the physical features of one parent and the intelligence of another

```
class D: visibility A, visibility B,.....
{
//body of D class
.....
.....
.....
};
```



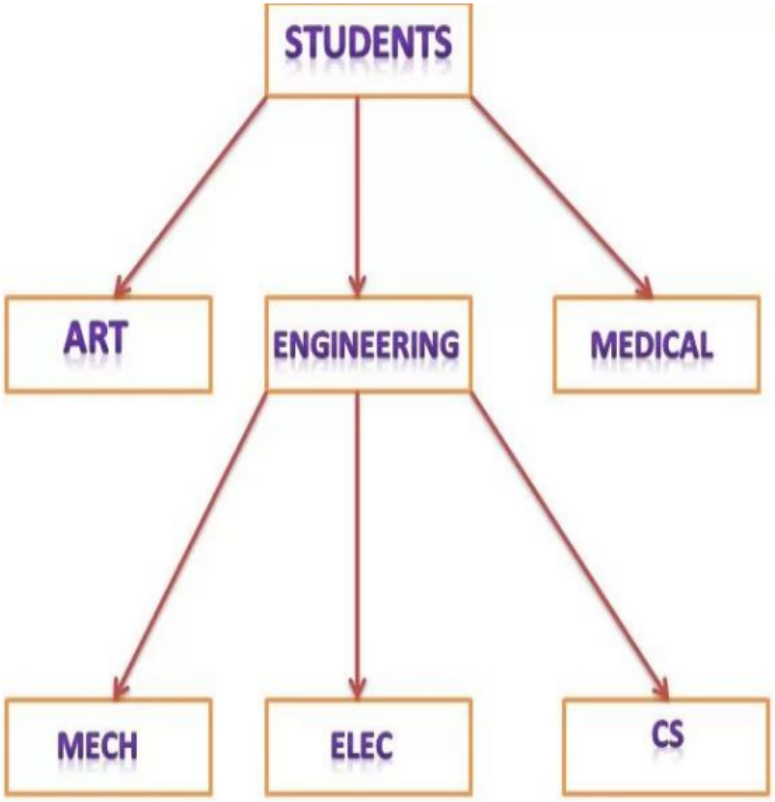
```
#include<iostream>
using namespace std;

class M {
protected:
    int m;

public:
    void get_m(int);
};

class N {
protected:
    int n;

public:
    void get_n(int);
};
```

```
class P : public M, public N {
public:
    void display();
};

void M::get_m(int x) {
    m = x;
}

void N::get_n(int y) {
    n = y;
}

void P::display() {
    cout << "m=" << m << "\n";
    cout << "n=" << n << "\n";
    std::cout << "m*n=" << m * n << "\n";
}
```

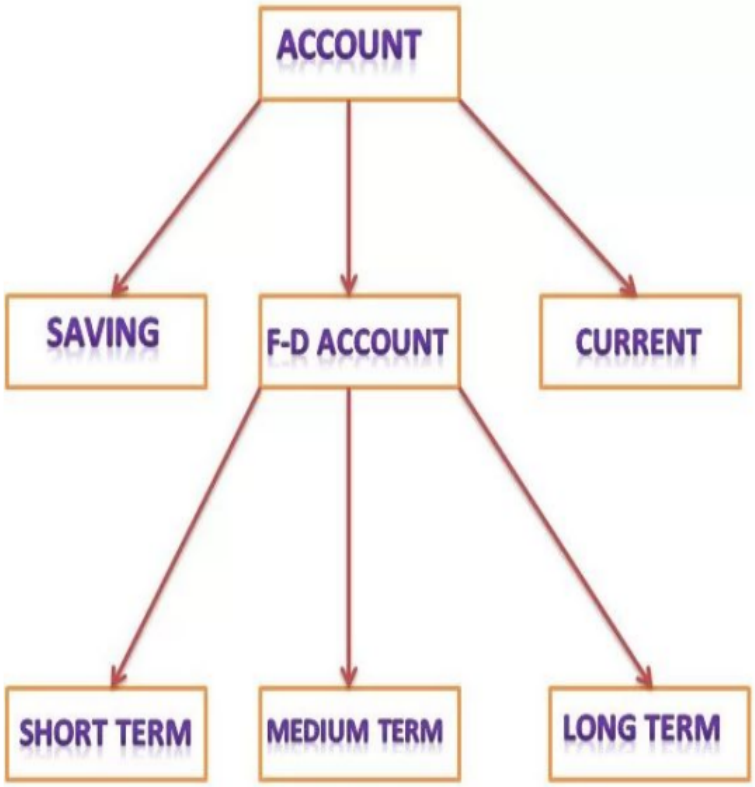
```
int main() {
    P p;
    p.get_m(10);
    p.get_n(20);
    p.display();
    return 0;
}
```

Hierarchical inheritance

- The base class will include all features that are common to subclasses.
- A subclass can be constructed by inheriting the properties of the base class.
- A subclass can serve as a base class for the lower level class and so on..



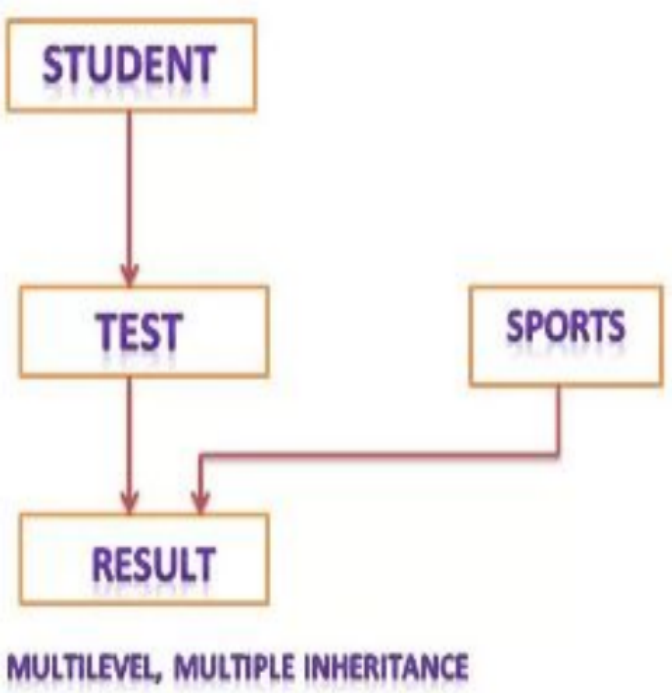
Hieriarchical classification of students



Hieriarchical classification of bank accounts

Hybrid inheritance

- There could be situation where we need to apply two or more types of inheritance to design a program




```
class sports
{
protected:
float score;
public:
void getscore(float);
void putscore();
};
```

```
class test: public student
{
.....
.....
};
```

```
class sports: public test, public sports
{
.....
.....
};
```

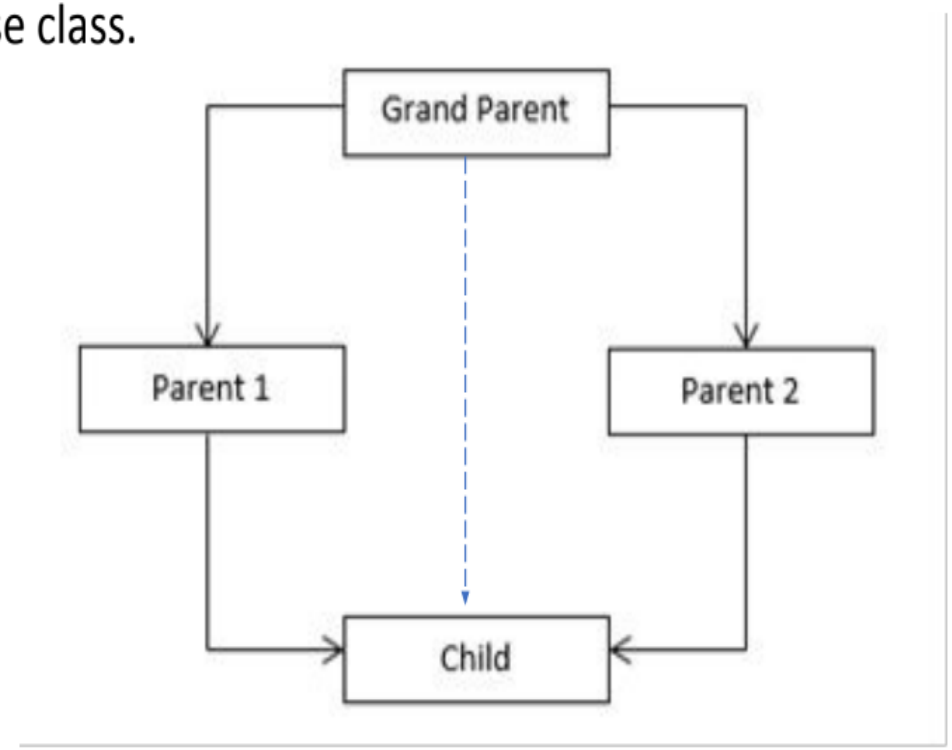
```
#include <iostream>
using namespace std;

class student {
protected:
    int roll;

public:
    void getnumber(int a) {
        roll = a;
    }

    void putnumber() {
        std::cout << "roll number"
<< roll << "\n";
    }
};
```

```
class test : public student {
protected:
    float part1, part2;

public:
    void getmarks(float x, float y) {
        part1 = x;
        part2 = y;
    }

    void putmarks() {
        std::cout << "marks obtain:" << "\n" << "part1=" <<
part1 << "\n" << "part2=" << part2 << "\n";
    }
};
class sports {
protected:
    float score;

public:
    void getscore(float s) {
        score = s;
    }

    void putscore() {
        std::cout << "sports wt:" << score << "\n\n";
    }
};
```

```
class result : public test, public sports {
    float total;

public:
    void display();
};

void result::display() {
    total = part1 + part2 + score;
    putnumber();
    putmarks();
    putscore();
    cout << "total score :" << total << "\n";
}

int main() {
    result student1;
    student1.getnumber(1234);
    student1.getmarks(27.5, 33.0);
    student1.getscore(6.0);
    student1.display();
    return 0;
}
```

Virtual base classes

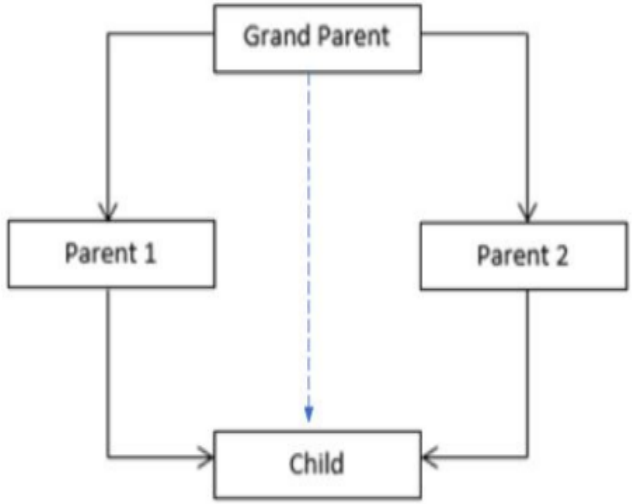
- The child has two direct base class parent1 and parent2 which themselves have a common base class grandparent.
- The child inherits the traits of grandparent via two separate paths, it can also inherit directly as shown by the broken line.
- The grandparent is sometimes referred to as indirect base class.



Virtual base classes

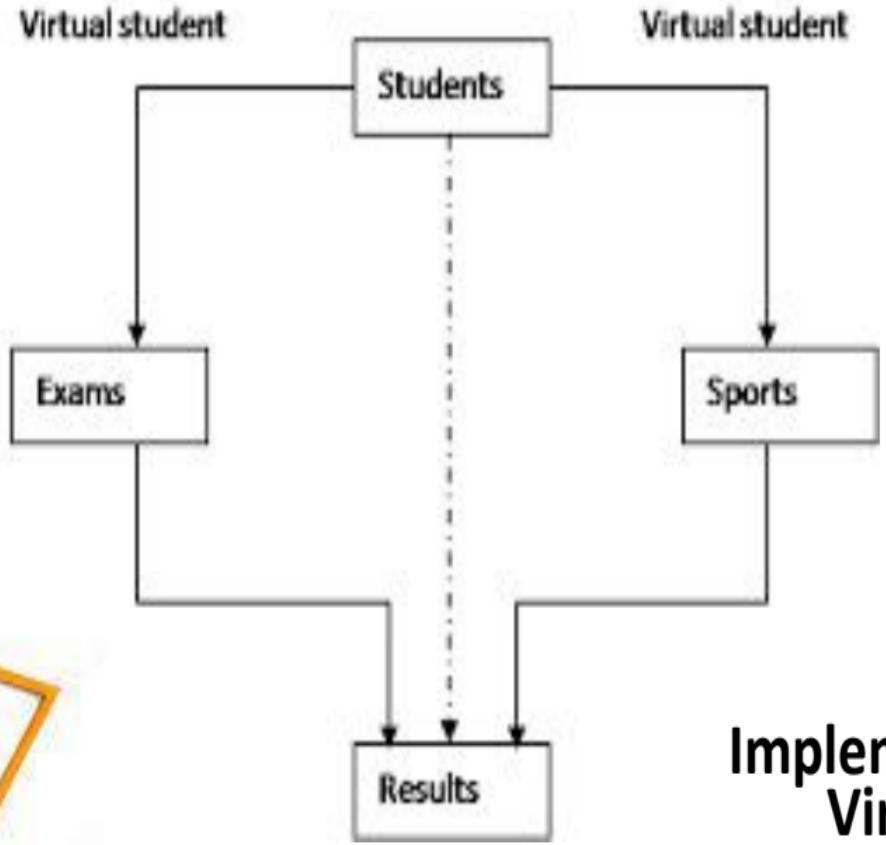
The duplication of the inherited members can be avoided by making common base class as the virtual base class:

```
class g_parent
{
    //Body
};
class parent1: virtual public g_parent
{
    // Body
};
class parent2: public virtual g_parent
{
    // Body
};
class child : public parent1, public parent2
{
    // body
};
```



When a class is virtual base class, C++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exists between virtual base class and derived class.

Note that keywords 'virtual' and 'public' can be used in either order.



Implement the concept of Virtual base class:

Modify the previous program of student results processing system with the classes student, test, sports and results with Virtual base class