→ Ambiguity resolution in inheritance:-

In heritance ambiguity arises when in multiple inheritance a derived class is derived from two base classes. If there is a function in both classes with same name and in derived class that function is called then, it will be called from which class.

This is ambiguity, it can be resolved by using scope resolution operator (::)

example :-

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

Class A
{
   void in(void)
   {
      Cout<<" Hello, Hey"<<endl;
   }
};
```

```
class B
{
    void hi(void)
    {
      cout<<" Namaskar" <<endl;
    }
};

class C : public A, public B
{
      void hi(void)
      {
          A::hi();
      }
      void show(void)
      {
          hi();
      }
};
int main()
{
    A O1;
    O1.hi();

    B O2;
    O2.hi();
```

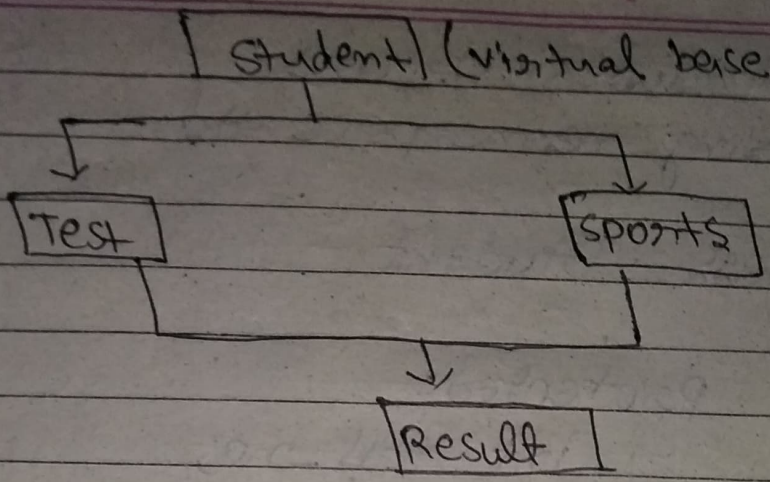
```
C c3;
c3.in();

return 0;
}
```

In single inheritance if a function is defined in base class and in derived class that same function is defined then while invoking that function for derived class, compiler automatically resolve that ambiguity and invoke that function which is defined in derived class.

→ Virtual base class :-

Virtual base class are made to solve ambiguity in inheritance. It controls the flow of data member and member function down the hierarchie. It checks for, there is no same function or variables inherited in a class twice.

For example.



we have a class student, two classes derived from it Test and sports. Both get one-one copy of functions and variable from student. Result is derived from Test and sports, so, in normal case it get 2 copy of function & variable from student.

To prevent any ambiguity we make Student a virtual base class, by doing so, now, complier only pass one copy of Student class to the Result class, and checks for at any point there should not be 2 copy of Student's function and datamember at one derived class.

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

class Student
{
    protected:
        int roll_no;
    Public:
        void set_number(int a)
        {
            roll_no=a;
        }
        void print_number(void)
        {
            cout<<" Your roll no is "<<roll no
                                    <<endl;
        }
};

class Test : virtual public Student
{
    protected:
        float maths, physics;
    pubic:
        void set_number(float m1, float m2)
        {
            maths=m1;
            physics=m2;
        }
```

```
    void print_marks(void)
    {
        cout<<" Your marks is here :"<<endl
            <<" Maths : " <<maths <<endl
            <<" physics :"<<physics <<endl;
    }
};

class sports : virtual public student
{
    protected :
        float score;
    pubic :
        void set_score(float sc)
        {
            score = sc;
        }
        void print_score(void)
        {
            cout<<" Your PT score is"
                << score <<endl;
        }
};
```

```
Class Result : Pubic Test, Pubic Sports
{
    private :
        float total;
    Pubic :
        void display(void)
        {
            total = maths+physics+score;
            print_numbers();
            print_marks();
            print_score();
            Cout<<" Your total score is: "
                <<total <<endl;

        }

};

int main()
{
    Result Harry;

    Harry.set_number(4200);
    Harry.set_marks(78.9, 99.5);
    Harry.set_score(9);
    Harry.display();
    return 0;
}
```

→ Constructors in derived classes and in Inheritence :-

In derive class: to use constructors there are some rules.

- If base class constructor doesn't have any arguments, there is no need of any constructor in derived class. & when object for derived class is made, constructor will automatically called.

- But if there are one or more arguments in the base class constructor, then constructor of base class will be invoked via object of derived class by passing arguments through them.

- If both base and derived classes have constructor, when object of derived class is made base class constructor is execded first then derived class constructor will execute.

- If there is more than one base class like in multiple inheritance, constructor will be invoked in the order of their declaration.

ex.-

```
Class D: Public A, Public B
{

};
```

then first constructor of A will be invoked then that of B, then constructor of derived class D will invoked if any.

- In ~~multiple~~ multilevel inheritance, the constructors are executed in order of inheritance.

- C++ supports an special syntax for passing arguments to multiple base classes constructors.

- The constructor of the derived class receives all the arguments at once and then will pass to the respective base classes.

~~derived class constr~~

Derived class constructor (int a1, int a2, int a3, int a4):
Base 1 Constructor (int a1), Base 2 (int a2)

{

// code

}

- If there is virtual base class among base classes, constructor of virtual class is called then constructor of other non-virtual base classes will be invoked in the order of their declaration.

- If there are multiple virtual base classes and multiple non-virtual base classes then first constructor of virtual base class will invoked in order of declaration, then constructor of non-virtual base class will invoked in their order of declaration then at last Constructor of derived class will be invoked

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

class base1
{
    int data1;
public:

    base1(int i)
    {
        data1 = i;
        cout<<" Base 1 Constructor called"<<endl;
    }
void print_base1(void)
    {
        cout<<" The value of data 1 is:"<<data1
                                        <<endl;
    }

};

class Base2
{
    int data2;
public:
    Base2(int i)
    {
        data2 = i;
        cout<<" Base 2 Constructor called"<<endl;
    }
```

```
void printbase2(void)
{
    cout<<" value of data 2 is:" <<data2
                                 <<endl;
}
};
class Derived: public base1, Pubic base2
{
    int d1, d2;
public:
    Derived(int a, int b, int c, int d):
                    Base1(a), Base2(b)
    {
        d1=c;
        d2=d;
        cout<<" Derived class constructor
                    called"<<endl;
    }

    void printderived(void)
    {
        cout<<" value of d1 is:"<< d1<<endl;
        cout<<" value of d2 is:" <<d2<<endl;
    }
};
```

```
int main()
{
    Derived Ak(5, 7, 9, 12);
    Ak.printdata1();
    Ak.printdata2();
    Ak.printderived();
    return 0;
}
```

→ <u>Initialization list/intialization section:-</u>

Intialization list is used to pass value to the data member directly from the arguments of constructor.

Syntax:-

```
class x
{
    int b, c;
    Public:
        x(int x, int y): b(x), c(y)
        {
            code;
        }
};
```

intialization list.

Same type of systax we have seen when we pass values to the base class constructor through derived class constructor. We can use this with them too.

```
class A
{
    int a;
  public:
    A(int x): a(x)
    {
      //code;
    }
};

class B
{
    int b;
  Public:
    B(int y): b(y)
    {
      //code;
    }
};
```

```
class D : public A, public B
{
    int d1, d2;
  public:
    D(int p, int q, int r, int s) : A(p), B(q),
                                    d1(r), d2(s)
    {
        // code;
    }
};
int main()
{
    // code;
    return 0;
}
```

→ Revisiting pointers (new and delete keyword) :-

New keyword is used to allocate memory dynamically & return its address to a pointer.

ex.-
```
int * ptr = new int(40);
cout<< *(ptr) <<endl;
```

In outcome, we get 40.

```
int * arr = new int [3];
```

By writing this line of code we have dynamically allocation array of size 3.

```
arr[0] = 10;
arr[1] = 30;
arr[2] = 50;
```

or

```
*(arr) = 10;
*(arr+1) = 30;
*(arr+2) = 50;
```

delete operator is used to release the memory allocated by new.

```
delete ptr;
```

It will delete or free memory allocated while declearing ptr as

```
int *ptr = new int(40);
```

To delete or free memory allocated by array we have to write

```
delete [] array name;
```

ex-

```
delete [] arr;
```

→ Pointers to object & arrow operator :-

As we can use pointer to reference and dereference to a variable we can do same ~~by pointer~~ with objects.

ex.-

```
int main()
{
    A Obj1;
    Obj1.getdata();
    Obj1.printdata();
    return 0;
}
```

we can write it also as

```
int main()
{
    A Obj1;
    A *ptr = &Obj1;

    (*ptr).getdata();
    (*ptr).printdata();
}
```

Using objects with pointers will be easy with new operator as:

```
int main
{
    A *obj1 = new A;
// by writing this code we have dynamically
// created an object for class 'A'.

    (*obj1).getdata();
    (*obj1).setdata();

    return 0;
}
```

Arrow operator (->) is abbreviated form of dereferencing and to an address and doing accessing some part of it. For example in the given line of code

```
(*obj1).getdata();
```

we have dereference to an address stored in obj1 and accessed it getdata() function, same this can be written as,

```
obj1 -> getdata();
```

new, delete and → (arrow) keyword are very helpful in complex programms, so we need to learn them.

~~example programme~~.

NOTE:- NOTE if you have a paramatrize constructor and you have to pass arguments to that constructor you can do as,

{{class name}} * Ptr = new {{class name}}(args);

ex-

A *Ptr = new A(3,5);

if A is class and constructor of A takes 2 integers arguments.

Example Code:-

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

Class number
{
    int a, b, c, d;
  public:
    number(int a, int b, int C, int d): a(a), b(b),
    {                                   c(c), d(d)
```

```
cout<<" First complex no is: "<<a<<"+"<<b
                              <<"i"<<endl;
cout<<" second complex no is: "<<c<<"+"<<d<<"i"
                              <<endl;
}

void cal()
{
    cout<<"sum of both no is: "
        <<a+c<<"+"<<b+d<<"i"<<endl;

    }
};

int main()
{
    number *ptr = new number(8, 10, 15, 9);
    ptr->cal();
    return 0;
}
```