Unit-3 Object Oriented Programming with C++

BCA SEM-3

Unit-3	**Type Conversion and Inheritance**	**14**	**17**
	- Type conversion - Categories of type conversion - Basic of inheritance- - Types of inheritance**-** Single level, multiple, multilevel, hierarchical and hybrid - Constructor in derived class - Concept of Abstract class - Nesting of classes		

BCA SEM-3
Assistant Professor: Vinod.M.Makwana
KBPCCS

Que : 1 what is type conversion? Explain categories of type conversion.

- ✓ Data type refers which kinds of data are to be stored in a particular variable.
- ✓ One type of information in stored into the another type is called type conversion.
- ✓ Type conversion is required when data types of both operands are differ, means one is basic and other is complex data type.
- ✓ c/c++ supports two types of type conversion such as
  - Implicit type conversion
  - Explicit type conversion

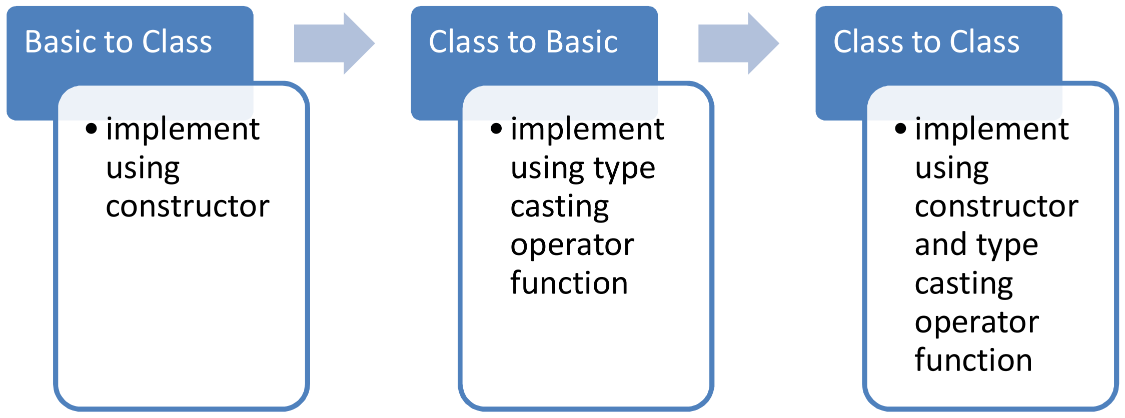
Implicit type conversion:

A conversion done by system automatically is called implicit type conversion for example conversion between basic data type such as int, float, char etc.

Explicit type conversion:

A conversion done by the program is called explicit type conversion.

There are following type of type conversion in c++


Basic to Class
• implement using constructor
Class to Basic
• implement using type casting operator function
Class to Class
• implement using constructor and type casting operator function

[A]Basic to Class type conversion

- Basic data type is converted into the complex of class data type is called basic to class type conversion.
- Right hand operand to equal operator should be basic type and left operand to equal operator should be class type.
- It can be implement using parameterized constructor only, where basic type should be argument of the constructor.
- For example convert seconds into hours, minutes, and seconds.

Syntax:

```
Class_name(argument_1)
{
        //body
}
```

Example

```
#include<iostream.h>
#include<conio.h>

class TIME
{
        int hours,minutes,seconds;
        public:
                TIME() //do nothing constructor
                {
                }
                TIME(int s)
                {
                        int rem;
                        hours=s/3600;
                        rem=s%3600;

                        minutes=rem/60;
                        rem=rem%60;

                        seconds=rem;
                }
```

```
			void put()
			{
				cout<<"Value of Hours:"<<hours<<endl;
				cout<<"Value of Minutes:"<<mitutes<<endl;
				cout<<"Value of Seconds:"<<seconds<<endl;
			}
};
void main()
{
	clrscr();
	TIME t;
	int SECONDS;
	cout<<"Enter value of Seconds:";
	cin>>SECONDS;
	t=SECONDS;		//invoke parameterized constructor
	t.put();
	getch();

}
```

Output:

```
Enter Value of Seconds: 3661
Value of Hours: 1
Value of Minutes: 1
Value of Seconds: 1
```

[B]Class to Basic type conversion

- ❖ Class data type is converted into the basic data type is called class to basic type conversion.
- ❖ Right hand operand to equal operator should be Class type and left operand to equal operator should be Basic Type.
- ❖ It can be implement using operator type casting function only.
- ❖ For example convert Time object into seconds.

Syntax:

```
operator data_type()
{
	//function body
}
```

As per above syntax some important characteristics of operator type casting function are as below.

1. There is no return type.
2. There are no arguments.
3. operator is a name of the function

Example

```
#include<iostream.h>
#include<conio.h>

class TIME
{
        int hours,minutes,seconds;
        public:
                void get()
                {
                        cout<<"Enter value of Hours:";
                        cin>>hours;

                        cout<<"Enter value of Minutes:";
                        cin>>hours;

                        cout<<"Enter value of  Seconds:";
                        cin>>hours;
                }
};
void main()
{
        clrscr();
        TIME t;
        int SECONDS;
        SECONDS=t;              //invoke operator type casting function
        cout<<"Seconds:"<<SECONDS;
        getch();

}
```

Output:

```
Enter of Hours: 1
Enter of Minutes: 1
Enter of Seconds: 1

Seconds: 3661
```

[C]Class to Class type conversion

- Class data type is converted into another Class data type is called class to class type conversion.
- Right hand operand to equal operator should be Class type and left operand to equal operator should be another class type.
- It can be implement using operator type casting function or constructor.
- For example convert day object converted into calendar object.

Syntax:

```
operator data_type()
{
        //function body
}
```

OR

Syntax:

```
Class_name(argument_1)
{
        //body
}
```

As per above syntax some important characteristics of operator type casting function are as below.

1. There is no return type.
2. There are no arguments.
3. operator is a name of the function
4. Argument must be type of class type.

Example: implement using constructor

```
#include<iostream.h>
#include<conio.h>

class DAY;//early declaration
class CALENDER
{
        int year,month,week,day;
        public:
                void put()
                {
                        cout<<"value of year :"<<year;
                        cout<<"value of Month:"<<month;
                        cout<<"value of  Week:"<<week;
                        cout<<"value of day:"<<day;

                }

                CALENDER(DAY D)
                {
                        int rem;

                        year=D/365;
                        rem=D%365;

                        month=rem/30;
                        rem=rem%30;

                        week=rem/7;
                        rem=rem%7;

                        day=rem;
                }
};
```

```
Class DAY
{
        int days;
        public:
                void get()
                {
                        Cout<<"How many days:";
                        Cin>>days;
                }
};
void main()
{
        clrscr();
        CALENDER C;
        DAY D;
        D.get();
        C=D;            //invoke constructor
        C.put();
        getch();

}
```

Output:

```
How many days: 403

Value of year:1
Value of month:1
Value of week:1
Value of day:1
```

Example: implement using operator type casting function

```
#include<iostream.h>
#include<conio.h>

class DAY;//early declaration
class CALENDER
{
        int year,month,week,day;
        public:
                void put()
                {
                        cout<<"value of year :"<<year;
                        cout<<"value of Month:"<<month;
                        cout<<"value of  Week:"<<week;
                        cout<<"value of day:"<<day;

                }

};

Class DAY
{
        int days;
        public:
                void get()
                {
                        Cout<<"How many days:";
                        Cin>>days;
                }

                operator CALENDER()
                {
                        CALENDER C;
                        int rem;

                        C.year=days/365;
                        rem=days%365;

                        C.month=rem/30;
                        C.rem=rem%30;

                        C.week=rem/7;
                        C.rem=rem%7;

                        C.day=rem;

                        return C;
                }

};
```

```
void main()
{
        clrscr();
        CALENDER C;
        DAY D;
        D.get();
        C=D;            //invoke operator type casting function
        C.put();
        getch();

}
```

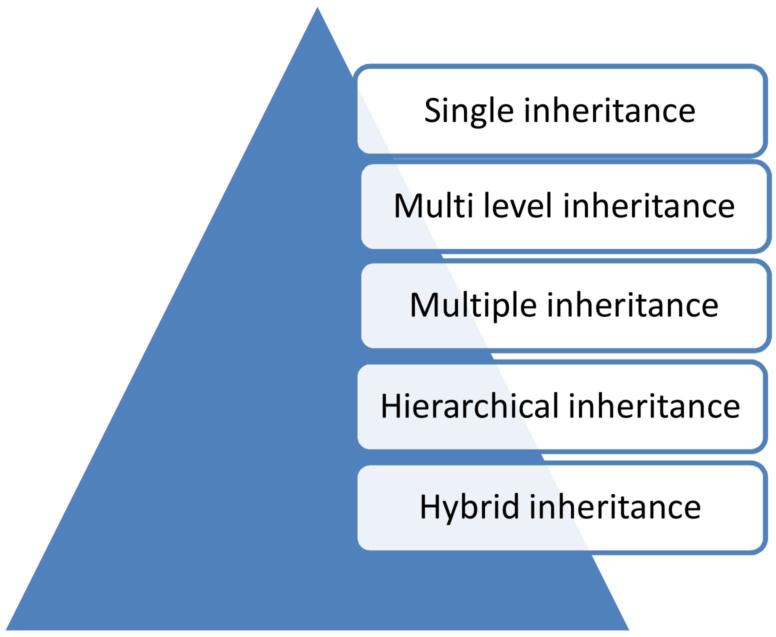
Output:

```
How many days: 403

Value of year:1
Value of month:1
Value of week:1
Value of day:1
```

Que: 2 what is Inheritance? Explain types of inheritance with proper example.

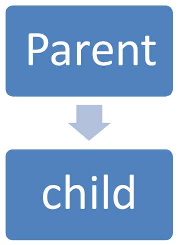
- ✓ Acquiring the properties of one class into another class is called inheritance.
- ✓ In general we cannot access the members of one class into another class but inheritance provides the facility to access member into another class.
- ✓ Reusability of code is also known as inheritance.
- ✓ A class is derived into is called base or parent or root class.
- ✓ A class is derived from another class is called child or sub or derived class.
- ✓ In short properties of one class are access into another class in called inheritance.



[1]Single Inheritance

An inheritance which has only single parent and single child class is called single level or single inherence.

Diagram



Syntax:

```
Class base_class
{
        //members and member functions
};
Class child_class:visibility_mode base_class
{
                //members and member functions

};
```

```
class Parent
{
        public:
                char Father_name[20];
        parent()
        {
                strcpy(Father_name,"Dashrath");
        }
};

class Child:public Parent
{

        public:
                char name[20];
        Child()
        {
                Strcpy(name,"Ram")
        }

        void display()
        {
                cout<<"My Name is :"<<name<<endl;
                cout<<"\nFather Name is:"<<Father_name;
        }

};

void main()
{
        Child C;
        C.display();
        getch();
}
```

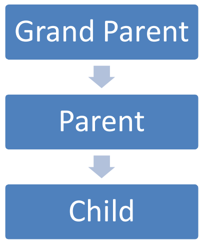
Output:

```
My name is:Ram
Father Name is:Dashrath
```

[2]Multilevel Inheritance

An inheritance which has more than one level is called multilevel inheritance. In this form sometimes parent class become child class or child class become parent class.

Diagram



Syntax:

```
Class base_class
{
        //members and member functions
};
Class child_class-1:visibility_mode base_class
{
                //members and member functions

};
Class child_class-2:visibility_mode child_class-1
{
                //members and member functions

};
```

Example

```
class Grand_Parent
{
        public:
                 char Surname[20];
        Grand_parent()
        {
                strcpy(Surname,"Yadav");
        }
};

class Parent:public Grand_Parent
```

```
{
        public:
                char Father_name[20];


          Parent()
          {
                Strcpy(Father_name,"Vasudev");
          }
};


class Child:public Parent
{
        public:
                char name[20];
           Child()
           {
                Strcpy(name,"Krishna");
           }
           void display()
           {
                cout<<"My Name is :"<<name<<endl;
                cout<<"\nFather Name is:"<<Father_name;
                cout<<"\nSurname is :"<<surname;
           }

};

void main()
{
           Child C;
           C.display();
           getch();
}
```

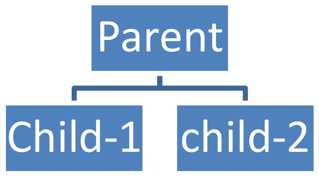
Output:

```
My name is:Krishna
Father Name is:Vasudev
Surname is:Yadav
```

[3]Hierarchical Inheritance

An inheritance which has single parent class and more than one child class is called hierarchical inheritance.

Diagram



Syntax:

```
Class base_class
{
        //members and member functions
};
Class child_class-1:visibility_mode base_class
{
                //members and member functions

};
Class child_class-2:visibility_mode base_class
{
                //members and member functions

};
```

Example

```
class Parent
{
      public:
                char Father_name[20];


        Parent()
         {
                Strcpy(Father_name,"Dashrath");
         }
};
```

```
class Child_1:public Parent
{
        public:
                    char name[20];
           Child()
           {
                    Strcpy(name,"Ram");
           }
           void display()
           {
                    cout<<"My Name is :"<<name<<endl;
                    cout<<"\nFather Name is:"<<Father_name;
           }

};

class Child_2:public Parent
{
        public:
                    char name[20];
           Child()
           {
                    Strcpy(name,"Laxman");
           }
           void display()
           {
                    cout<<"My Name is :"<<name<<endl;
                    cout<<"\nFather Name is:"<<Father_name;
           }

};

void main()
{
           Child_1  C1;
           C1.display();

           Child_2 C2;
           C2.display();
           getch();
}
```

Output:

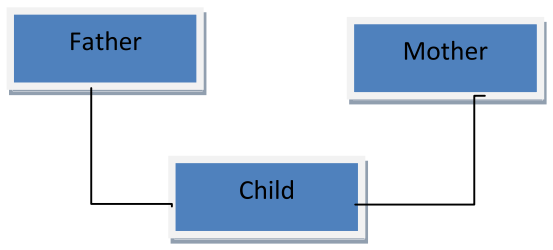
```
My name is:Ram
Father Name is:Dashrath

My name is:Laxman
Father Name is:Dashrath
```

[4]Multiple Inheritance

An inheritance which has more than one parent class but only single child class in known as multiple inheritance.

Diagram



Syntax:

```
Class base_class_1
{
        //members and member functions
};
Class base_class_2
{
                //members and member functions

};
Class child_class:visibility_mode base_class-1,visibility_mode base_class-2
{
                //members and member functions

};
```

Note: as per above syntax multiple inheritance separated by comma(,)

Example

```
class Father
{
        public:
                char Father_name[20];


          Father()
          {
                Strcpy(Father_name,"Dashrath");
          }
};

class Mother
{
        public:
                char mother_name[20];
           Child()
           {
                Strcpy(name,"Kaushalya");
           }


};

class Child:public Father,public Mother
{
        public:
                char name[20];
           Child()
           {
                Strcpy(name,"Ram");
           }
           void display()
           {
                cout<<"My Name is :"<<name<<endl;
                cout<<"\nFather Name is:"<<Father_name;
                cout<<"\nMother Name is:"<<Mother_name;
           }

};
```

```
void main()
{
        Child C;
        C.display();
        getch();
}
```

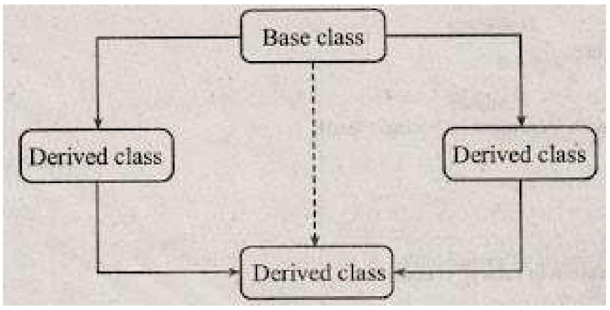
Output:

```
My name is:Ram
Father Name is:Dashrath
Mother Name:Kaushalya
```

[5]Hybrid inheritance

Combination of more than one form of inheritance is called hybrid inheritance.
It is also known as multipath or diamond shape inheritance.
When a class inherits the properties of another class two or more times through two or more different paths it is known as multipath inheritance.
The properties are inherited more than once, so more memory is consumed and ambiguity (doubt) gets created, in order to avoid this extra consumption of memory and ambiguity, we declare the base class as virtual while giving the definition of the derived class.

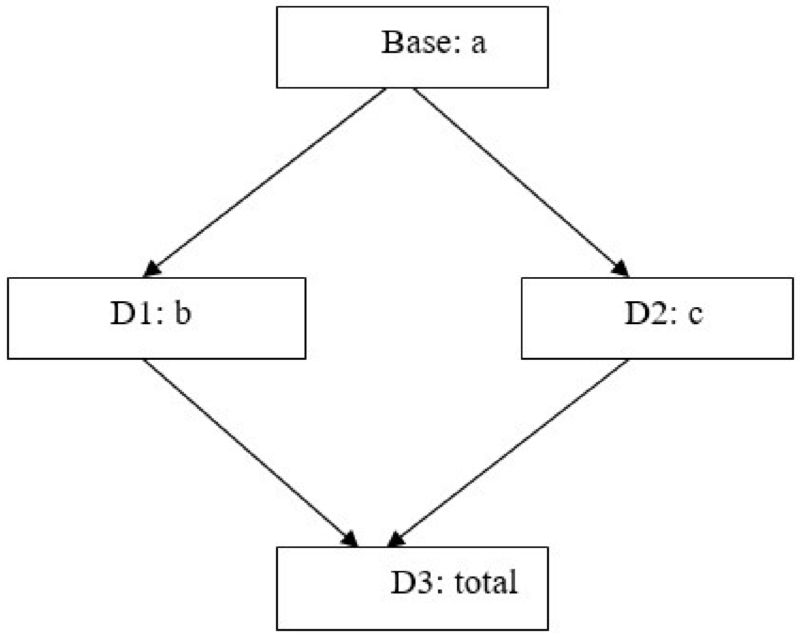
On declaring the base class as virtual the properties of the base class are inherited only once and hence no extra memory is consumed and the problem of ambiguity automatically gets solved.



Syntax:

```
class base
{
};
class derived1:public virtual / virtual public base
{
};
class derived2:public virtual/ virtual public base
{
};
class derived3:public derived1,public derived2
{
};
```

Example:



We want to calculate the sum of all the data elements (“a” of class Base, “b” of class D1 and “c” of class D2) and store them in “total” of class D3.

```
//virtual base  inheritance
#include<iostream>
#include<conio.h>
using namespace std;
class base
{
            public:
             int a;
             base()
            {
                        a=10;
            }
};
class D1:virtual public base
{
            public:
             int b;
            D1()
            {
                        b=20;
            }
};
class D2:virtual public base
{
            public:
              int c;
             D2()
            {
              c=30;
            }
};
class D3:public D1,public D2
{        public:
         int total;
         D3()
         {
            total=(a+b+c);
           cout<<”Total:”<<total;
         }
};
```

```
void main()
{
    D3 ob;
    getch();
}
```

Output:

```
total 60
```

Que : 3 write a note on : Constructor in derived class

- We can use constructors in derived classes in C++
- If the base class constructor does not have any arguments, there is no need for any constructor in the derived class
- But if there are one or more arguments in the base class constructor, derived class need to pass argument to the base class constructor
- If both base and derived classes have constructors, base class constructor is executed first
- We can implements in following ways in different forms of inheritance.

1. Constructors in Multiple Inheritances
2. Constructor in multilevel inheritance

- **In multiple inheritances,** base classes are constructed in the order in which they appear in the class deceleration. For example if there are three classes "A", "B", and "C", and the class "C" is inheriting classes "A" and "B". If the class "A" is written before class "B" then the constructor of class "A" will be executed first. But if the class "B" is written before class "A" then the constructor of class "B" will be executed first.
- **In multilevel inheritance**, the constructors are executed in the order of inheritance. For example if there are three classes "A", "B", and "C", and the class "B" is inheriting classes "A" and the class "C" is inheriting classes "B". Then the constructor will run according to the order of inheritance such as the constructor of class "A" will be called first then the constructor of class "B" will be called and at the end constructor of class "C" will be called.

Special Syntax

- C++ supports a special syntax for passing arguments to multiple base classes
- The constructor of the derived class receives all the arguments at once and then will pass the call to the respective base classes
- The body is called after the constructors is finished executing
-

Syntax Example:
Derived-Constructor (arg1, arg2, arg3….): Base 1-Constructor (arglist_1), Base 2-Constructor(arglist_2)…baseN(arglistN)
{
//body of derived class
}

Example

```
/*
        CONSTRUCTOR IN DERIEVED CLASS
*/
#include<iostream.h>
#include<conio.h>
class theory
{
        int os,cpp,dfs,sad;
        public:
                theory(int o,int c,int d,int s)
                {
                        os=o,cpp=c,dfs=d,sad=s;
                }
};
class practical
{
        int pr;
        public:
        practical(int p)
        {
                pr=p;
        }
};
```

```
class result:public practical,public theory
{
        public:
        int total;
        result(int os,int cp,int df,int sa,int p):theory(os,cp,df,sa),practical(p)
        {
                total=(os+cp+df+sa+p);
        }
        void display()
        {
                cout<<"\nTotal:"<<total;
        }
};
void main()
{
        clrscr();
        result R(45,65,76,65,43);
        R.display();
        getch();
}
```

Output:

```
Total: 294
```

Que: 4 Write a detail note on : Abstract Class

- A class which contains at least one pure virtual function is called abstract class.
- It provides interface for its sub classes.
- Classes inheriting an abstract class must provide definition to the pure virtual function otherwise they become abstract class.

Characteristics of abstract class

- Abstract class cannot be instantiated, but pointer and reference of abstract class type can be created.
- Abstract class can have normal functions and variables along with a pure virtual function.
- It is mainly used for up casting. Up casting is a process to create pointer of base class and refers address of sub class.
- Classes in inheriting an abstract class must implement all pure virtual functions else they will become abstract classes too.

Pure virtual function:

- A virtual function with no definition is called pure virtual function.
- It start with virtual keyword and end with=0.
- Pure virtual function must me overrides in derived class.

Example:

```
class Base
{
        public:
                virtual void show()=0;//pure virtual function
};

class derived:public Base
{
        public:
                void show()
                {
                cout<<"Implemetation of virtual function";//method overriding
                }
};
```

```
void main()
{
        Base *bptr;
        derived d;
        B=&d;
        B->show();
        getch();
}
```

Output:

```
Implements of virtual function
```

Que: 5 Write a detail note on: Nesting of classe

Defining object as a member in a particular class is known as nesting of class.

We can also define member as object of another class in a particular class.

We can inherit the properties without inheritance using nesting of class.

Example:

```
class apha
{
        public:
                int a;
                alpha(int x)
                {
                        x=a;
                        cout<<"Alpha initialized with:"<<a<<endl;
                }
};
```

```
class beta
{
        public:
                int b;
                beta(int y)
                {
                        b=y;
                        cout<<"Beta initialized with:"<<b<<endl;
                }
};

class gamma
{
        public:
                int c;
                alpha A;
                beta B;

                gamma(int a,int b,int z):A(a),B(b)
                {
                        c=z;
                        cout<<"Gamma initialized with:"<<c<<endl;
                }
};

void main()
{
        gamma G(10,20,30);
        getch();
}
```

Output:

```
Alpha initialized with:10
Beta initialized with:20
Gamma initialized with:20
```

As per above example we can also access the properties of class alpha and beta without inheritance.