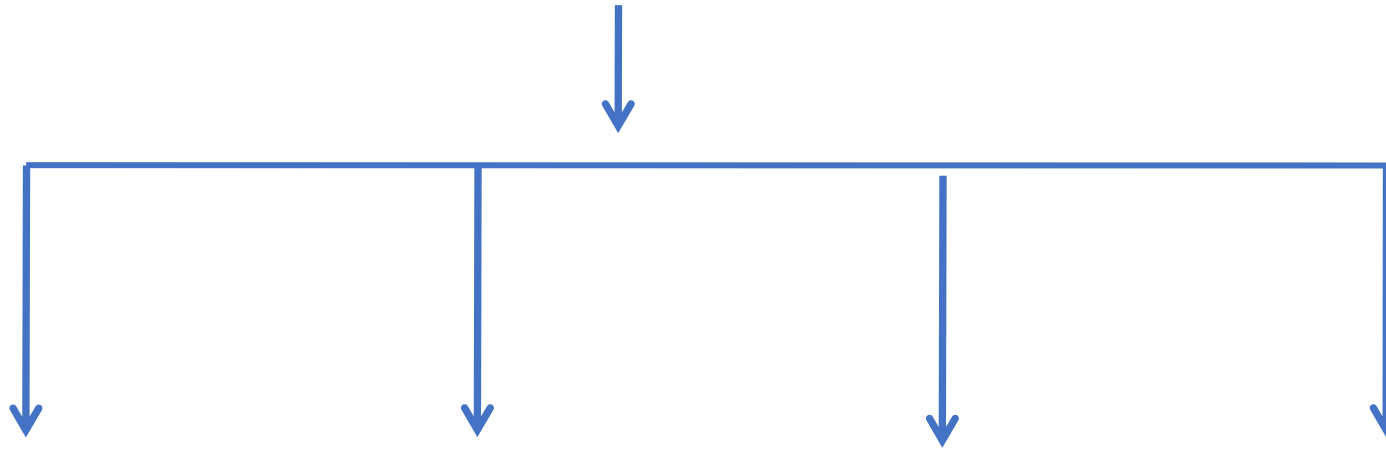


Hierarchical inheritance:

One base class and multiple derived class, one –to-many relationship



Syntax of Hierarchical Inheritance

```
class base_class {  
    ... ..  
}  
  
class first_derived_class: public base_class {  
    ... ..  
}  
  
class second_derived_class: public base_class {  
    ... ..  
}  
  
class third_derived_class: public base_class {  
    ... ..  
}
```

```
// C++ program to demonstrate hierarchical inheritance

#include <iostream>
using namespace std;

// base class
class Animal {
public:
    void info() {
        cout << "I am an animal." << endl;
    }
};

// derived class 1
class Dog : public Animal {
public:
    void bark() {
        cout << "I am a Dog. Woof woof." << endl;
    }
};

// derived class 2
class Cat : public Animal {
public:
    void meow() {
        cout << "I am a Cat. Meow." << endl;
    }
};
```

```
int main() {
    // Create object of Dog class
    Dog dog1;
    cout << "Dog Class:" << endl;
    dog1.info(); // Parent Class function
    dog1.bark();

    // Create object of Cat class
    Cat cat1;
    cout << "\nCat Class:" << endl;
    cat1.info(); // Parent Class function
    cat1.meow();

    return 0;
}
```

Output

```
Dog Class:
I am an animal.
I am a Dog. Woof woof.

Cat Class:
I am an animal.
I am a Cat. Meow.
```

```

#include <iostream>
using namespace std;

class A //single base class
{
    public:
        int x, y;
        void getdata()
        {
            cout << "\nEnter value of x and y:\n"; cin >> x >> y;
        }
};

class B : public A //B is derived from class base
{
    public:
        void product()
        {
            cout << "\nProduct= " << x * y;
        }
};

```

```

class C : public A //C is also derived from class base
{
    public:
        void sum()
        {
            cout << "\nSum= " << x + y;
        }
};

int main()
{
    B obj1;           //object of derived class B
    C obj2;           //object of derived class C
    obj1.getdata();
    obj1.product();
    obj2.getdata();
    obj2.sum();
    return 0;
} //end of program

```

Output

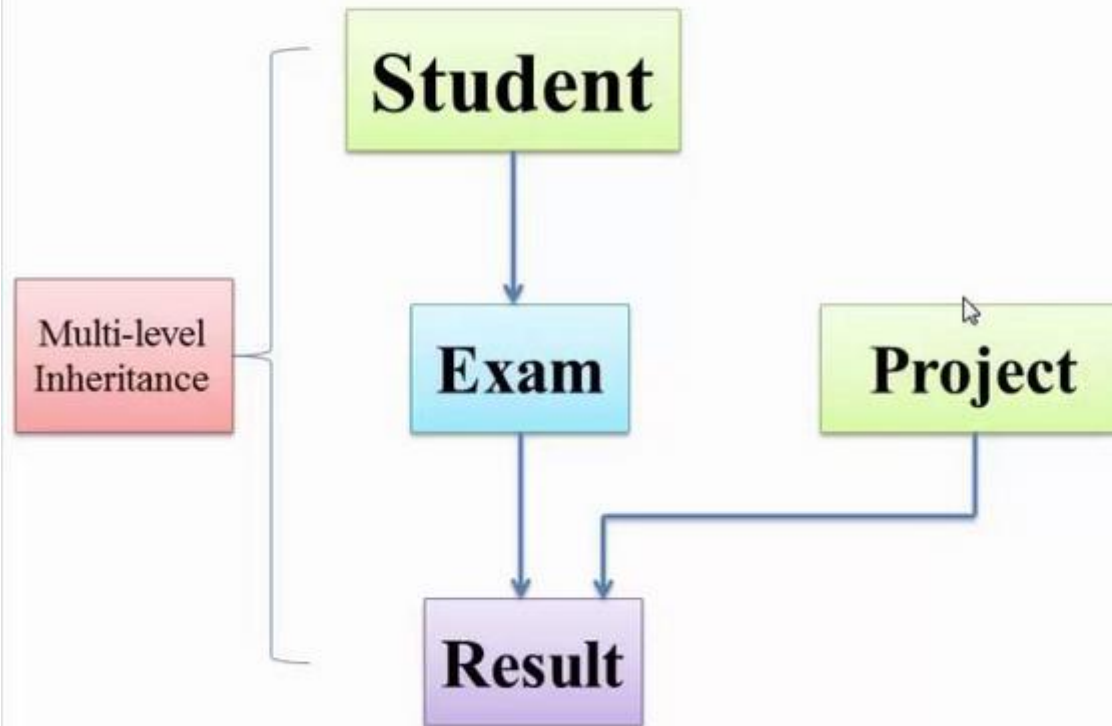
```

Enter value of x and y:
2
3
Product= 6
Enter value of x and y:
2
3
Sum= 5

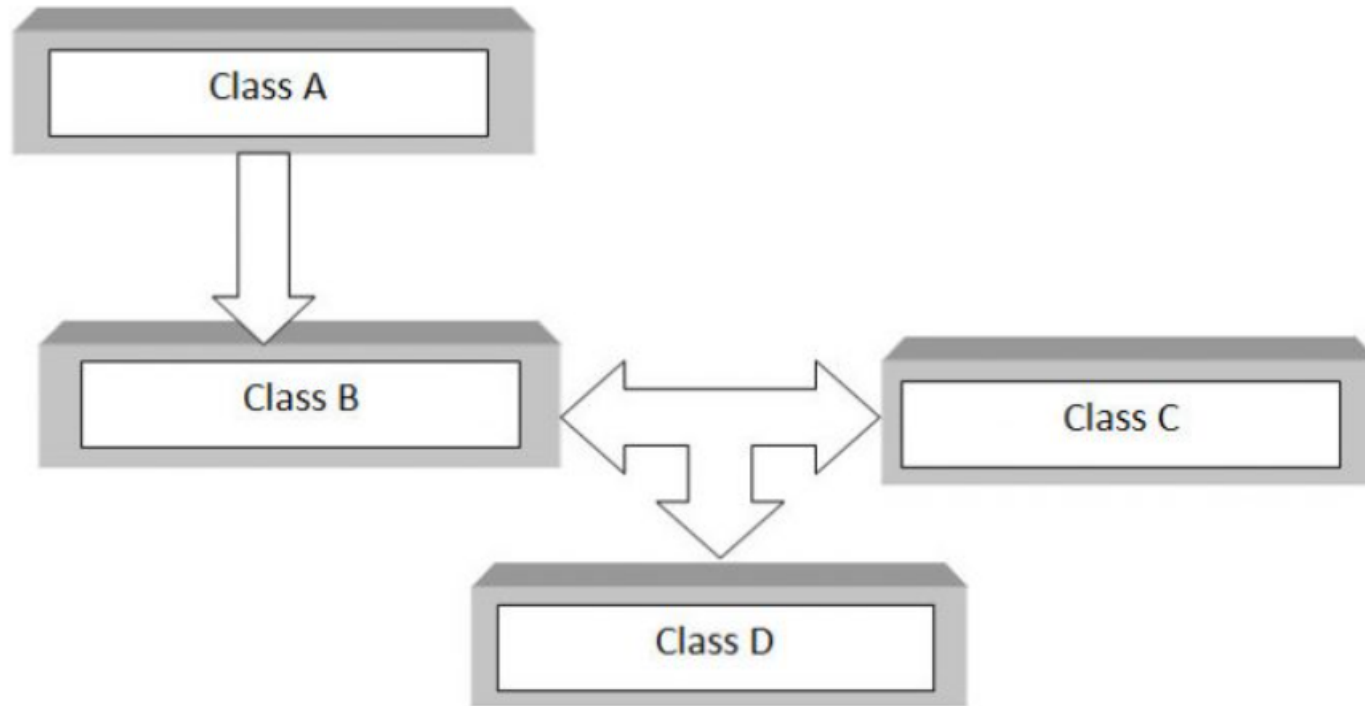
```

Hybrid Inheritance

Combination of two or more type of inheritance to design a program.



C++ Hybrid Inheritance Block Diagram



```
class A
{
    .....
};
class B : public A
{
    .....
} ;
class C
{
    .....
};
class D : public B, public C
{
    .....
};
```

```

#include <iostream>
using namespace std;

class A
{
    public:
    int x;
};

class B : public A
{
    public:
    B()    //constructor to initialize x in base class A
    {
        x = 10;
    }
};

```

```

class C
{
    public:
    int y;
    C()    //constructor to initialize y
    {
        y = 4;
    }
};

class D : public B, public C    //D is derived from class B and class C
{
    public:
    void sum()
    {
        cout << "Sum= " << x + y;
    }
};

```

```

int main()
{
    D obj1;    //object of derived class D
    obj1.sum();
    return 0;

}    //end of program

```


Type Conversion



- A type cast is basically a conversion from one type to another. There are two types of type conversion:

Implicit Type Conversion Also known as ‘automatic type conversion’.

- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (**type promotion**) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.
- **bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double**

Example of Type Implicit Conversion:

```
// An example of implicit conversion

#include <iostream>
using namespace std;

int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    cout << "x = " << x << endl
         << "y = " << y << endl
         << "z = " << z << endl;

    return 0;
}
```

Output:

```
x = 107
y = a
z = 108
```

Explicit Type Conversion:

- This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

can be also considered as forceful casting.

Syntax:

```
(type) expression
```

where *type* indicates the data type to which the final result is converted.

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

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    cout << "Sum = " << sum;

    return 0;
}
```

Output:

```
Sum = 2
```

Type Conversion in user-defined data

- User define data type conversion not done automatically
- A user-defined data types are designed by the user to suit their requirements, the compiler does not support automatic type conversions for such data types therefore, the user needs to design the conversion routines by themselves if required.
- User define data type conversion done by using either constructor or by using casting operator

Three type of situation occurs during user define type conversion:

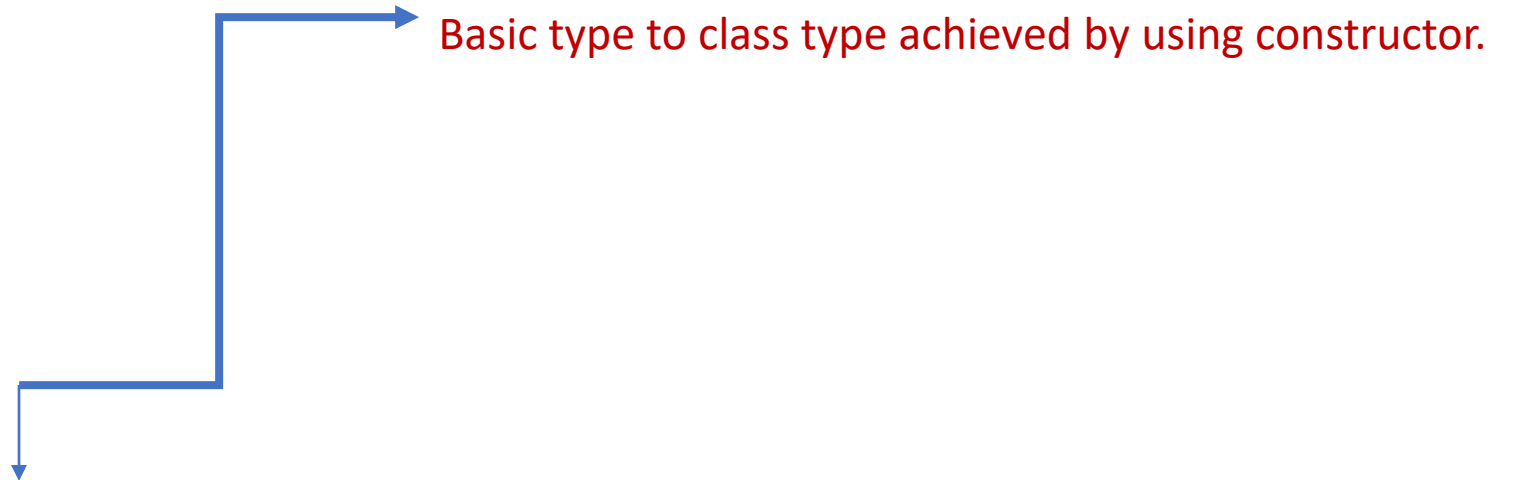
1. basic type to class type(using constructor)
2. class type to basic type(using casting operator function)
3. class type to class type (using constructor and casting operator function both)

1. basic type to class type-using constructor

To perform this conversion, the idea is to use the [constructor](#) to perform type conversion during the [object creation](#).

```
#include <iostream>
using namespace std;
class A
{

};
int main()
{
A a1;
int x=8;
a1=x ;//basic to class type
return 0;
}
```



convert int to user-defined data type:

```
#include <bits/stdc++.h>
using namespace std;

// Time Class
class Time {
    int hour;
    int mins;

public:
    // Default Constructor
    Time()
    {
        hour = 0;
        mins = 0;
    }

    // Parameterized Constructor
    Time(int t)
    {
        hour = t / 60;
        mins = t % 60;
    }

    // Function to print the value
    // of class variables
    void Display()
    {
        cout << "Time = " << hour
              << " hrs and "
              << mins << " mins\n";
    }
};
```

```
// Driver Code
int main()
{
    // Object of Time class
    Time T1;
    int dur = 95;

    // Conversion of int type to
    // class type
    T1 = dur;
    T1.Display();

    return 0;
}
```

Time = 1 hrs and 35 mins

2. class type to basic type-using casting operator function

Class type to basic type done by using casting operator function

1.It must be a define inside in class.

2.It must not specify a return type in function signature.

3.It must not have any arguments.

```
class A
```

```
{};
```

```
A a1;
```


```
int x;
```

```
x=a1 //class type to basic type
```

casting operator function

Syntax:

```
operator dest_typename()  
{  
    return statement;  
}
```



```
operator int()  
{  
    return a;  
}
```

```

#include <bits/stdc++.h>
using namespace std;

// Tie Class
class Time {
    int hrs, mins;

public:
    // Constructor
    Time(int, int);

    // Casting operator
    operator int();

    // Destructor
    ~Time()
    {
        cout << "Destructor is called."
              << endl;
    }
};

// int() operator is used for Data
// conversion of class to primitive
Time::operator int()
{
    cout << "Conversion of Class"
          << " Type to Primitive Type"
          << endl;

    return (hrs * 60 + mins);
}

```

```

// Function performs type conversion
// from the Time class type object
// to int data type
void TypeConversion(int hour, int mins)
{
    int duration;

    // Create Time Class object
    Time t(hour, mins);

    // Conversion OR duration = (int)t
    duration = t;
    cout << "Total Minutes are "
          << duration << endl;

    // Conversion from Class type to
    // Primitive type
    cout << "2nd method operator"
          << " overloading " << endl;

    duration = t.operator int();

    cout << "Total Minutes are "
          << duration << endl;

    return;
}

```

```

// Driver Code
int main()
{
    // Input value
    int hour, mins;
    hour = 2;
    mins = 20;

    // Function call to illustrate
    // type conversion
    TypeConversion(hour, mins);

    return 0;
}

```

Output

```

Conversion of Class Type to Primitive Type
Total Minutes are 140
2nd method operator overloading
Conversion of Class Type to Primitive Type
Total Minutes are 140
Destructor is called.

```

Conversion of one class type to another class type

Ex: A obj1; B obj2;

obj1 = obj2 ; // obj1 and obj2 are objects of different classes

➤ First approach using Constructor:-

Left side of assignment operator(=) which is class object we have to create constructor in that class here in Class A.

➤ Second approach using casting operator function:

Right side of assignment operator(=) which is class object we have to create casting operator function in that class here class B.

1.Using constructor :

```
#include<iostream>
using namespace std;
//CGS system
class CGS
{
    int mts; //meters
    int cms; //centimeters
public:
    void showdata()
    {
        cout<<"Meters and centimeters in CGS system:";
        std::cout << mts<<" meters "<<cms<<" centimeters" << std::endl;
    }
    CGS(int x,int y) // parameterized constructor
    {
        mts=x;
        cms=y;
    }
    int getcms()
    {
        return cms;
    }
    int getmts()
    {
        return mts;
    }
};
```

In the Destination class we use the constructor method

```
//Objects of different types
ObjectX=ObjectY;
Here ObjectX is Destination object and ObjectY is source object
```

```
class FPS
{
    int feet;
    int inches;
public:
    FPS() // default constructor
    {
        feet=0;
        inches=0;
    }
    FPS(CGS d2)
    {
        int x;
        x=d2.getcms()+d2.getmts()*100;
        x=x/2.5;
        feet=x/12;
        inches=x%12;
    }
    void showdata()
    {
        cout<<"feet and inches in FPS system:";
        std::cout << feet<<" feet "<<inches<<" inches" << std::endl;
    }
};

int main()
{
    CGS d1(9,10);
    FPS d2;
    d2=d1;
    d1.showdata(); //to display CGS values
    d2.showdata(); //to display FPS values
    return 0;
}
```

Output

```
Meters and centimeters in CGS system:9 meters 10 centimeters
feet and inches in FPS system:30 feet 4 inches
```

2.Using Overloading casting operator

```
// Objects of different types  
objectX = objectY;
```

- Here we use Overloading casting operator in source class i.e. overloading destination class in source class
- we have two classes Time and Minute respectively and will convert one class Time to another Minute class.
- In the below example minute class is destination class and time class is source class
- so we need to overload the destination class in the source class
- Here we should not tell the return type but we returns the overloaded class object
- i.e. returning value without specifying return type

```

#include <bits/stdc++.h>
using namespace std;
//minutes class
class Minute {

public:
    int mins;
    // Constructors
    Minute()
    {
        mins = 0;
    }

    // Function to print the value of
    // hours and minutes
    void show()
    {
        cout << "\nTotal Minute : " << mins << endl;
    }
};

// Time Class
class Time {
    int hr, mins;

public:
    // Constructors
    Time(int h, int m)
    {
        hr = h;
        mins = m;
    }
    Time()
    {
        cout << "\nTime's Object Created";
    }
    operator Minute () //overloading minute class
    {
        Minute m;
        m.mins = (hr * 60) + mins;
        return m;
    } //driver code

    // Function to print the value of
    // hours and minutes
    void show()
    {
        cout << "Hour: " << hr << endl;
        cout << "Minute : " << mins << endl;
    }
};

// Minutes Class
int main()
{
    Time T1(3,40);
    Minute m;
    m=T1; //minute class is destination and Time class is source class
    T1.show();
    m.show();
    return 0;
}

```

Output

```

Hour: 3
Minute : 40

Total Minute : 220

```



Any Query?
