OBJECT ORIENTED PROGRAMMING WITH C++

SUB CODE :18BIT23C

UNIT III: Operator Overloading: Overloading Unary – Binary Operators – Overloading Friend Functions – Type Conversion – Inheritance: Types of Inheritance – Single – Multilevel – Multiple – Hierarchical – Hybrid and Multi Path Inheritance – Virtual Base Classes – Abstract Classes.

TEXT BOOK

1. Ashok N Kamthane, "Object Oriented Programming with ANSI and Turbo C++", Pearson Education Publications, 2006.

Prepared By: Dr. M. Soranamageswari

3.1 Operator Overloading

- Overloading is an important feature of c++
- It is similar to function overloading. An operator is a symbol used for an operation.
- C++ has the ability to treat the user-defined data type.
- As a built in data type.
- The Operator + can be used to perform addition of two variables but it is not possible to perform addition of two objects.
- Operator overloading is one of the most valuable concept to perform this type of operation.
- It is a type of polymorphism permit to write multiple definitions for functions and operators.

- The Operator +,-,* and = are used to carry the operations of overloading.
- The Capability to relate the existing operator with a member function and use the resulting operator with object of its class, as its operands is called Operator Overloading.

Syntax:

```
Return type
{
        S+1
        S+2
}
```

Example

```
Number operator +(number D)
{
    Number T;
    T.X=X+D.X;
    T.Y=Y+D.Y;
    Return T;
}
```

- Overloaded Operators are redefined using the keyword Operator followed by an Operator symbol.
- An Operator function should be either a member function or Friend function.
- A Friend Function requires one argument for unary operators and two for binary Operators.
- A Member function requires one argument for binary operator and no arguments for unary Operators.

The prototype for operator overloading can be return as follows:

- Void Operator ++();
- Void Operator - -();
- Void Operator – ();
- Num operator + (num);
- Friend num operator * (int, num);
- Void Operator =(num);

- The Prototype of operator overloading function in classes.
- The Operator Overloading can be carried out in the following steps:
- Define a class to be used for overloading operations.
- In the public section the class contains the prototype of the function operator().
- Define the definition of the operator()

Example:

```
#include<iostream.h>
#include<conio.h>
class number
{
public:
        int X;
        int Y;
number() {}
number (int j, int k)
{
        X=j;
        Y=k;
}
```

- }

```
number operator +(number D)
{
        number T;
        T.X=X+D.X;
        T.Y=Y+D.Y;
        Return T;
}
void show()
{
        cout<<”\n  X=”<<”\n  Y=”<<Y;
}
};
```

```
void main()
{
        clrscr();
        number  A(2,3) ,B(4,5),C;
        A.show();
        B.show();
        C=A+B;
        C.show();
}
```

3.2 Overloading Unary Operators

- The Operator ++, - - and – or unary Operator.
- The unary Operator ++ and - - can be used as prefix and suffix with the functions.
- These operators have only one operand.

Example:

```
#include<iostream.h>
#include<conio.h>
class num
{
private:
        int a,b,c,d;
public:
```

```
num(int j, int k, int m, int l)
{
        a=j;
        b=k;
        c=m;
        d=l;
}
void show(void);
void operator ++();
};
void num::show()
 {
        cout<<”A=”<<a<<”B=”<<b<<”C=”<<c<<”D=”<<d;
}
void num:: operator ++()
 { ++a;++b;++c;++d;}
```

```
void main()
{
        clrscr();
        num X(3,2,5,7);
        cout<<”\n before increment of x:”;
        X.show();
        ++x;
        cout<<”\n after increment of x:”;
        X.show();
        return 0;
}
```

3.3 Overloading binary operator:

- Overloading with a single parameter is called binary operator overloading
- Binary operators requires two operands binary operator or overloaded using member function and friend function
- Overloading binary operator using member function:
- Overloading binary operator using member function require 1 argument
- The argument contains value of the object which is to the right of the operator
- The overloading function should be declared as follows.

Syntax:

Operator(num 02);

Where,

Operator is a symbol

Num is an class

02 is the argument of the class

Example:

03=01 operator + (02)

The callingunction can be written as,

03=01+02

Here the data membr are passed to the called function and performs the number of addition based on number of arguments

Example:

```
#include<iostream.h>
#include<conio.h>
class num
{
  int a,b,c,d;
public:
    void(input(void);
    void show(void);
    num operator + (num);
};
```

```
void num: :input( )
{
cin >>a>>b>>c>>d;
}
void num : :show( )
{
cout <<a<<b<<c<<d;
}
num : : operator+(num t)
{
    m tmp;
    tmp.a=a+t.a;
    tmp.b=b+t.b;
    tmp.c=c+t.c;
    tmp.d=d+t.d;
return(tmp);
}
```

```
void main ( )
{
  num x,y,z;
  x.input( );
  y.input ( );
   z=x+y;
  x.show( );
  y.show( );
  z.show( );
}
```

3.4 Overloading friend functions

- Friend function are more useful in operator overloading
- They are more flexible then member function,
- The different between member function and friend function is that member function arguments explicitly
- The friend functions needs the param eter should be explicitly fast.
- Friend function requires two operands to be passed as arguments

Syntax

- Friend return type operator (variable!, operator symbol variable?)
- {
- ……….
- }

Example:

```
friend num operator + (num n1 num n2)
#include <iostream.h>
#include<conio.h>
class num
{
int a,b,c,d ;
public:
    void input (void);
    void show (void);
    friend num operator*(int,num);
};
void num : : input( )
{
cin >>a>>b>>c>>d;
}
void num : : show ( )
{
  cout<<a<<b<<c<<d;
}
```

```
num operator*(inta, numt)
{
      num tmp;
      tmp.a = a*t.a;
      tmp.b = b*t.b;
      tmp.c = c*t.c;
      tmp.d = d*t.d;
      return(tmp);
}
void main( )
{
  num x,z;
  x.input( );
  z=3*x;
  x.show( );
  z.show( );
}
```

3.5 Type conversion

- The constants and variable of various data types are companied in a single expression can be automatically converted by the compiler.
- The compiler has no knowledge about the user-defined data type and about their conversion of other data type.
- There are three possibilities of data conversion.

  1. Conversion from Basic data type to user-defined data type(class type)
  2. Conversion from class type to basic data type
  3. Conversion from one class type to another class type Conversion type.

S.no	Conversion type	Routine in destination class	Routine in source class
1	class to class	Constructor	Conversion function, (operator function)
2	class to Basic	-	Conversion function, (operator function)
3	Basic to Class	Constructor	-

- Basic source and destination objects are user defined data type the conversion routine can be carried out using operator function is source class or using constructor in destination class.
- If the user – defined object is destination class. The conversion routine should be carried out using constructor in the destination class.
- If the user – defined object is a source object. The conversion routine should be carried out using source object in the operator function.
-

3.5.1 Conversion from Basic – class type

- In this type the left hand operand of (=) equal sign if always the class type. The right hand operand is always basic type.
- The Conversion can be done by the compiler with the helpof build routine or by applying type casting.
- It uses constructors for changing the Basic type to class type.

Example:

```
#include<iostream.h>
#include<conio.h>
class data
{
    int x ;
    float f;
public:
     data( )
{
  x=0; f=0;
}
data(float m)
{
x=2;
f=m;
}
void show( )
{
    cout<<x<<f;
  }
};
int main ( )
{
    data= z
z=1;
z.show( );
z= 2.5
z.show( );
}
```

3.5.2 Conversion from class type – Basic data type

- The compiler does not have any knowledge about the
- user – defined data type using class.
- In this type of conversion the programmer explicitly specify about the conversion.
- There instruction are return in a member function. This type of conversion also known as over loading of type cast operators.
- In this type the left hand operand is Basic data type the right hand operand is class type.
- To perform this conversion it must satisfy the following condition.
-

- The conversion function should not have any argument
- Do not mention return type.
- It should be a class member function.

Example:

```
#include<iostream.h>
#include<conio.h>
class data
{
   int x;
   float f ;
```

```
public;
    data( )
{
X=0; y=0;
}
operator int ( )
{
return(x) ;
}
data (float (m)
{
x=2;
f=m;
}
void show( )
{
cout<<x<<f;
}
};
int main( )
{
int j;
float f;
data a;
a=5.5;
j=a;
f=a;
cout<<j;
cout<<f;
}
```

3.5.3 Conversion from one class type _ another class type

- There are two ways to convert one class type to another class type
- One is to define a conversion operator function in source class or a constructor in a destination class.

Example:

```
#include<iostream.h>
#include<conio.h>
class stock2:
{
int code, item;
float price;
```

```
public:
stock1 (int a, int b, int c)
{
   code =a;
   item =b;
   price =c;
}
void disp( )
{
    cout<<code;
    cout<<item;
    cout<<price;
}
```

```
int getcode( )
{
return code;
}
int getitem()
{
return item;
}
int get price()
{
return price;
}
```

```
operator float()
{
return(item*price);
}
};
class stock2
{
int code;
float val;
public:
stock2()
{
code=0;value=0;
}
```

```
stock2(int x,float  y)
{
code=x;
val=y;
}
void disp()
{
cout<<code;
cout<<val;
}
stock2(stock1 p)
{
code=p.getcode();
val=p.getitem()*p.getprice();
}
};
```

```
void main()
{
stock 1 i1(10,10,100.5)
stock i2;
Float tot=i1;
i2=i2;
i1.disp();
i2.disp();
}
```

3.5.4 Rules For Overloading Operators

- Overloading of an operator cannot change the basic idea of an operator. when an operator is overloaded. its properties like syntax, precedence, and associativity remain constant.

Example:

- A and B are objects.
- A+=B
- Assigns additions of objects A and B to A. The overloaded operator must carry the same task the original operator according to the language.
- The floating statement must perform the same operation like the last statement.
-

- A=A+B
- Overloading of an operator must never change its natural meaning.
- An overloaded operator+ can be used for subtraction of two objects. but this type of code decrease the utility of the program.
- Remember that the aine of operator overloading is to comfort the programmer to carry various operations with objects.

3.6 Inheritance

- It is one of the most useful characteristic of object oriented programming.
- New classes are created from existing classes.
- The properties of existing classes are extended to new   classes.
- The new classes are called are derived classes.
- The existing classes are known as base classes.
- The term reusability means to reuse the properties of base class in the derived class.
- Reusability is achieve using inheritance the outcome of inheritance is reusability.
- The base class is called is called super class or parent class or ancestors class.
- The derived class is called as sub class or child class or descendent class.
- It is also possible to derive a class from previously derived class.
- A class can be derived from more than one class.

3.6.1 Access specifies and simple inheritance:

- The public members of a class can be accessed by objects, directly outside the class.
- The private members of the class can be accessed by public member function of the same class.
- The protected access specified this same as private.
- The only difference is that it allows its derived classes to access protected members directly with out member function.

Syntax:

Derived class

Class name of derived class:access specifier name of the base class

{

Member variables of derived class

}

Example:

```
1.class B:public A
{
.
.
}
2.class B:private A
{
.
.
}
```

```
3.class B:protected A
{
.
.
}
4.class B:A(default definition private)
{
.
.
}
```

- 1. When public access specified is used public members of the derived class. similarly the protected members of the base class or protected member of the derived class.

- 2. When a private access specified is used public and protected members of the base class or private members of the derived class.

3.6.2 Public inheritance

- A class can be derived publicly or privately.when a class is derived publicly all the public members of the base class can be accessed directly in the derived class.
- The public derivation does not allow the derived class to access private member variables of the base class.

Example:

- Write a program to derive a class publicly from base class.declare the base class with its member under public section.

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

```
Class A
{
Public:
Int X;
};
Class B:public A
{
Public:
Int Y;
};
```

```
void main()
{
clrscr();
B b;
b.x=20;
b.y=10;
count<<”\n member of A:”<<b.x;
count<<”\n member of B:”<<b.y;
}
```

3.6.3 Private inheritance

- The object of privately derived class cannot access the public members of the base class directly.
- The member function are used to access the member of the base class.

Example:

- Write a program to derive a class privately. Declare the member of base class under public section.

```
#include<iostream.h>
#include<conio.h>
class A
{
public:
int x;
};
```

```
Class B:private A
{
Public:
Int y;
B()
{
X=20;
Y=40;
}
void show()
{
cout<<”\n x=”<<x;
cout<<”\n y=”<<y;
}
};
```

```
void main()
{
clrscr();
B b;
b.show();
}
```

3.6 4 Protected data with private inheritance

- The member functions of derived class cannot access the private member variables of base class.
- The private members of base class can be accessed using public member functions of the same class.
- To overcome this problem the protected access specifier is used.
- The protected is same as private but it allows the derived class to access the private members directly.

Example:

```
#include<iostream.h>
#include<conio.h>
class A
{
protected:
int x;
};
class B:private A
{
int y;
public:
B( )
{
x=30;
y=40;
}
void shows( )
{
cout<<"\n x="<<x;
cout<<"\n y="<<y;
}
};
void main( )
{
clrscr( )
B.b;
b.show( );
}
```

3.7 Types of Inheritance

- The process of inheritance can depends on the following points.

    1.Number of base classes:

- The program may contain one or more base classes.

    2.Number of derived classes:

- A program may contain one or more derived classes.

The types of inheritance are as follows:

1. Single inheritance or simple
2. Multiple inheritance
3. Hierarchical inheritance
4. Multilevel inheritance
5. Hybird inheritance
6. Multipath inheritance

3.8 single inheritance:

- When only one base class is used for derivation of a class and the derived class is not used for base class.
- Inheritance between one base class and one a derived class is known as single inheritance.
- The new class is termed as derived class and the old class is called base class.
- A Derived class inherit data members and member functions of base class.
- The Constructor and destructor of base class are not inherited.

Example:

```
#include<iostream.h>
#include<conio.h>
Class ABC
{
protected:
    char name[20];
    int age;
};
class abc:public ABC
{
    float height,weight;
public:
    void getdata()
{
    cin>>name>>age;
    cin>>height>>weight;
}
```

```
void display()
{
    cout<<name<<age;
    cout<<height<<weight;
};
void main()
{
    abc  x;
    x.getdata();
    x.display();
}
```

3.9. Multiple Inheritance

- Two or more base classes are used for derivation of a class.
- That is this type of inheritance contains one or more base classes and a single derived class it is known as multiple inheritance.
- When a class is derived from more than one base class is known as multiple inheritance.
- Properties of various base classes are transferred to single derived class.

Example:

```
#include<iostream.h>
#include<conio.h>
class A
{
protected:
int x:
}
class B
{
protected:
int y;
}
```

```
class C
{
protected:
int z;
}
class D: public A,B,C
{
int d;
public:
void getdata()
{
cin>>x>>y>>z>>d;
}
void display()
{
cout<<x<<y<<z<<d;
}
};
void main()
D.d1;
d1.getdata();
d1.display();
}
```

3.10.Hierarchical inheritance

- A single base class is used for derivation of two or more derived classes is known as hierarchical inheritance.
- Inheritance also support hierarchical arrangement of programs.
- Hierarchical unit source the top down arrangement of classes.

Example:

```
#include<iostream.h>
#include<conio.h>
class A
{
 protected:
int x;
}
class B
{
 protected:
int y;
}
```

```
class C
{
  protected:
int z;
}
class D: public A,public B
{
int d;
public :
void getdata()
{
cin>>x>>y;
}
```

```
void display()
{
cin<<x<<y;
}
};
class E:public  D,public  C

{
int e;
public:
```

```
void get()
{
cout<<e<<z;
}
};
void main()
{
E e1;
e1.getdata();
e1.display();
e1.get();
e1.put();
}
```

3.11 Multilevel inheritance

- When a class is from another derived class that it the derived class act is a base class.
- This type of inheritance is known as multilevel inheritance.

Example:

```
#include<iostream.h>
#include<conio.h>
class A1
{
protected :
int age;
char name[20];
};
```

```
void put()
{
cout<<age<<name;
cout<<height<<weight;
cout<<sex;
}
};
void main()
{
    A3.x;
x.get();
x.put();
}
```

```
class  A2:public A1
{
protected:
float height;
float weight;
};
class A3:publc A2
{
protected:
char  sex;
public:
void get()
{
cin>>age>>name;
cin>>height>t>weight;
cin>>sex;
}
```

3.12 Hybrid Inheritance

- The combination one or more type of inheritance is known as hybrid inheritance.
- Here two types of inheritance is used. That is single and multiple inheritance.
- x-base class
- y-derived class and base class of z.
- w-base class

Example:

```
#include<iostream.h>
#include<conio.h>
class A1
{
protected:
int age;
char name[20];
};
class A2:public A1
{
protected:
float heirght;
float weight;
};
class A3
{
protected:
char sex;
};
```

```
class A4:public A2,A3
{
protected:
char address[20];
Public:
void get( )
{
cin>>age>>name;
cin>>height>>weight;
cin>>sex;
cin>>address;
}
void put( )
{
cout<<age<<name;
cout<<height<<weight;
cout<<sex;
cout<<address;
}
};
void main( )
{
A4 x;
x.get( );
x.put( );
}
```

3.13.Multipath Inheritance:

- When a class is derived from two or more classes which are derived from the same base class is called multipath inheritance.
- It consists of many types of inheritance such as Multiple, Multilevel, Inheritance.
- x-base class
- y, z, w-derived classes of x.
- y, z-base class for w.

Example:

```
#include<iostream.h>
#include<conio.h>
{
protected:
int age;
char name[20];
};
class A2:public A1
{
protected:
float height;
float weight;
};
```

```
class A3:public A1
{
protected:
char sex;
};
class A4:public A1,A2,A3
{
protected:
char address[20];
public:
void get( )
{
cin>>age>>name;
cin>>height>>weigjht;
cin>>sex;
cin>>address;
}
void put( )
{
cout<<age<<name;
cout<<height<<weight;
cout<<sex;
cout<<address
}
};
void main( )
{
A4 x;
x.get( );
x.put( );
}
```

3.14 Virtual base class

- To overcome the ambiguity occurd in multipath inheritance c++ provides the keyword virtual.
- The keyword virtual declares the specified classes as virtual.
- It can avoid the duplication of member variables defined in the base classes.

Example:

```
#include<iostream.h>
#include<conio.h>
class A1
```

```
{
protected:
int age;
char name[20];
};
class A2:public virtual A1
{
protected:
float height;
float weight;
};
class A3:public virtual A1
{
protected:
char sex;
};
class A4:public A2,A3
{
protected:
char address[20];
Public:
void get( )
{
cin>>age>>name;
cin>>height>>weight;
cin>>sex;
cin>>address;
}
void put( )
{
cout<<age<<name;
cout<<height<<weight;
cout<<sex;
cout<<address;
}};
```

```
void main( )
{
A4.x;
x.get( );
x.put( );
}
```

3.15 Abstract classes

- When a class is not used for creating object is called abstract classes.
- The abstract classes can act as a base class. It is the layout abstraction in a program and it allows the base class on several levels of inheritance.
- An abstract classes developed only to act as a base class for inheriting the properties and no object of these classes are declared.
- **Simple inheritance**

Example:

```
#include <iostream.h>
#include<conio.h>
class ABC
{
```

```
protected:
int age;
char name[20];
};
class abc:public ABC
{
float height,weight:
public:
void get data( )
{
cin>>age>>name;
cin>>height>>weight;
}
void display( )
{
cout<<age<<name;
cout<<height<<weight;
} };
```

```
void main( )
{
abc x;
x.get data( );
x.display( );
}
```