LECTURE-18

Class to Basic Type

The constructor functions do not support conversion from a class to basic type. C++ allows us to define a overloaded casting operator that convert a class type data to basic type. The general form of an overloaded casting operator function, also referred to as a conversion function, is:

```
operator typename ( )
{
//Program statmerit .
}
```

This function converts a class type data to typename. For example, the operator double( ) converts a class object to type double, in the following conversion function:

```
vector:: operator double ( )
{ double sum = 0 ; for(int I = 0; ioize;
sum = sum + v[i] * v[i ] ; //scalar magnitude return sqrt(sum);
}
```

The casting operator should satisfy the following conditions.

- It must be a class member.
- It must not specify a return type.
- It must not have any arguments. Since it is a member function, it is invoked by the object and therefore, the values used for, Conversion inside the function belongs to the object that invoked the function. As a result function does not need an argument.

In the string example discussed earlier, we can convert the object string to char* as follows:

```
string:: operator char*( )
{ return (str) ;
}
```

One Class to Another Class Type

We have just seen data conversion techniques from a basic to class type and a class to basic type. But sometimes we would like to convert one class data type to another class type.

Example

Obj1 = Obj2 ; //Obj1 and Obj2 are objects of different classes.

Objl is an object of class one and Obj2 is an object of class two. The class two type data is converted to class one type data and the converted value is assigned to the Objl. Since the conversion takes place from class two to class one, two is known as the source and one is known as the destination class.

Such conversion between objects of different classes can be carried out by either a constructor or a conversion function. Which form to use, depends upon where we want the typeconversion function to be located, whether in the source class or in the destination class.

We studied that the casting operator function

Operator typename( )

Converts the class object of which it is a member to typename. The type name may be a built-in type or a user defined one(another class type) . In the case of conversions between objects, typename refers to the destination class. Therefore, when a class needs to be converted, a casting operator function can be used. The conversion takes place in the source class and the result is given to the destination class object.

Let us consider a single-argument constructor function which serves as an instruction for converting the argument's type to the class type of which it is a member. The argument belongs to the source class and is passed to the destination class for conversion. Therefore the conversion constructor must be placed in the destination class.

Table 7.3

Conversion	**Conversion takes place in**	
	Source class	**Destination class**
Basic to class	Not applicable	Constructor
Class to Basic	Casting operator	Not applicable
Class to class	Casting operator	Constructor

When a conversion using a constructor is performed in the destination class, we must be able to access the data members of the object sent (by the source class) as an argument. Since data members of the source class are private, we must use special access functions in the source class to facilitate its data flow to the destination class.

Consider the following example of an inventory of products in a store. One way of keeping record of the details of the products is to record their code number, total items in the stock and the cost of each item. Alternatively we could just specify the item code and the value of the item in the stock. The following program uses classes and shows how to convert data of one type to another.

```
#include<iostream.h> #include<conio.h>  class stock2;  class stock1
{ int code, item; float price; public:
stockl (int a, int b, float c)
{
code=a; item=b;  price=c; }
void disp( )
{ cout<<"code"<<code <<"\n"; cout<<"Items"<<item <<"\n";
cout<<"Price per item Rs . "<<price <<"\n";
}
int getcode( )  {return code; }  int getitem( ) {return item; }  int getprice( )  {return price;}
operator float( ) {
return ( item*price );
}
};

class stock2 { int code;  float val; public: stock2() { code=0; val=0; }
```

```
stock2(int x, float y)
{ code=x; val=y; } void disp( ) { cout<< “code”<<code  << “\n”; cout<< “Total Value Rs . “
<<val  <<”\n”
} stock2 (stockl p) { code=p . getcode ( ) ; val=p.getitem( ) * p. getprice ( ) ;
}
};

void main ( )
{          '
Stockl   il(101, 10,125.0); Stock2   12; float  tot_val; tot_val=i1  ; i2=il   ;
cout<<” Stock Details-stockl-type” <<”\n”; i 1 . disp ( ) ;
cout<<” Stock value”<<”\n”; cout<< tot_val<<”\n”; cout<<”  Stock Details-stock2-type”<<
“\n”; i2 .disp( ) ; getch ( ) ;
}
```

You should get the following output. Stock Details-stock1-type code 101 Items 10
Price per item Rs. 125
Stock value
1250

Stock Details-stock2-type code 10 1
Total Value Rs. 1250