Contents

- Operator Overloading
- Overloading Unary and Binary Operator
- Overloading using friend function
- Type casting and Type conversion

Quiz 1

- **What is Operator Overloading?**
  - **Operator overloading** is a specific case of polymorphism in which some or all **operators** like +, = or == are treated as polymorphic functions and as such have different behaviours depending on the types of its arguments.

Operator Overloading

- Operator overloading is a specific case of polymorphism in which some or all of operators like +, =, or == have different implementations depending on the types of their arguments.
- Operator overloading gives us the opportunity to redefine the C++ language.
- C++ permits us to add two variables of user defined types with the same syntax that is applied to the basic data types.

Restrictions on Operator Overloading

- **Overloading restrictions**
  - Precedence of an operator cannot be changed
  - Associatively of an operator cannot be changed
  - Arity (number of operands) cannot be changed
    - Unary operators remain unary, and binary operators remain binary
    - Operators **&, *, + and - each have unary and binary versions**
    - Unary and binary versions can be overloaded separately
- **No new operators can be created**
  - Use only existing operators
- **No overloading operators for built-in types**
  - Cannot change how two integers are added
  - Produces a syntax error

Restrictions on Operator Overloading

- Following operators can't be overloaded:
  - Class member access operators (., .*)
  - Scope resolution operator (::)
  - Size operator (sizeof)
  - Conditional operator (?:)
- All other operators can be overload

Operators that can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Defining Operator Overloading

- General syntax for operator overloading is:

```
return type classname :: operator op(arglist)
{
    Function body
}
```

For e.g.:

```
vector operator +(vector);
```

- vector is a data type of class.

Operator Overloading Process

- The process of overloading involves the following steps:
  - Create a class that defines the data type that is to be used in the overloading operation.
  - Declare the operator function operator op() in the public part of the class.
  - Define the operator function to implement the required operations.

Overloading Unary Operators

- Consider unary minus operator (changes the sign of the operand)
- The unary minus when applied to an object should change the sign of each of its data items.

Overloading Unary Operators

```
class space
{
   int x, y, z;
   public:
          void getdata( int a, int b, int c);
          void display(void);
          void operator - ();       // overload unary minus operator
};
```

Overloading Unary Operators

```
void  space :: getdata(int a, int b, int c)
{
        x = a;
        y = b;
        z = c;
}
void space :: display(void)
{
        cout <<  x ;
        cout <<  y ;
        cout <<  z ;
}
```

Overloading Unary Operators

```
void  space :: operator - ()
{
        x = -x;
        y = -y;
        z = -z;
}
int main()
{
        space S;
        S.getdata(10, 20, 30);
        S.display();
        -S;                    // activates operator-() function
        S.display();
        return 0;
}
```

Practice Statement 1

- Create a class date with day, month and year as its members. Accept the date from the user and display it. Overload the increment and decrement operators for displaying the next and previous date for the given date.

Overloading Binary Operators

- As a unary operator is overloaded we can also overload a binary operator.
- For e.g: A binary operator + can be overloaded to add two objects rather than adding two variables.
- Using operator overloading a functional notation,

```
C = sum(A, B);
```

Can be replaced by,

```
C = A + B;
```

Overloading Binary Operators

```
class complex
{
        float x;
        float y;
        public:
                complex(){}
                complex(float  real, float  imag)
                {
                        x = real;
                        y = imag;
                }
                complex  operator + (complex);
                void  display(void);
};
```

Overloading Binary Operators

```
complex  complex :: operator + (complex c)
{
        complex temp;
        temp.x = x + c.x;
        temp.y = y + c.y;
        return (temp);
}
void complex :: display(void)
{
        cout << x << " + j " << y ;
}
```

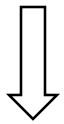
Overloading Binary Operators

```
int main()
{
        complex C1, C2, C3;
        C1 = complex(2.5, 3.5);
        C2 = complex(1.6, 2.7);
        C3 = C1 + C2;          // invokes operator+() function
        cout << " C1 = "; C1.display();
        cout << " C2 = "; C2.display();
        cout << " C3 = "; C3.display();
        return 0;
}
```

Overloading binary operators using Friends

- Friend functions may be used in place of member functions for overloading a binary operator.
- A friend function requires two arguments to be explicitly passed to it, while a member function requires only one.
- For e.g:

```
complex operator + (complex);
```

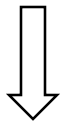


```
friend complex operator + (complex, complex);
```

Overloading binary operators using Friends

- The operator function definition would also be modified as:

```
complex  complex :: operator+(complex  c)
{
        complex temp;
        temp.x = x + c.x;
        temp.y = y + c.y;
        return(temp);
}
```



```
complex operator + (complex  a, complex  b)
{
        return  complex((a.x + b.x), (a.y  + b.y));
}
```

Overloading binary operators using Friends

- The statement,

```
C3 = C1  +  C2;
```

Is equivalent to

```
C3 = operator+(C1, C2);
```

Overloading Stream Insertion and Extraction Operators

- It is possible to overload the stream insertion (<<) and extraction operators (>>) to work with classes.
- This has advantages, in that
  - It makes programs more readable
  - It makes programs more extensible
  - It makes input and output more consistent

Overloading Stream Insertion and Extraction Operators

```
istream & operator >> (istream &din, vector &b)
    {
        for (int i = 0; i < size; i++)
        {
                din>>b.v[i];
        }
        return(din);
    }
```

Overloading Stream Insertion and Extraction Operators

```
ostream & operator << (ostream &dout, rational &b)
{
   dout<<"(";
   for (int i = 0; i < size; i++)
   {
            din>>b.v[i];
   }
   dout << ")";
   return(dout);
}
```

Type Conversion

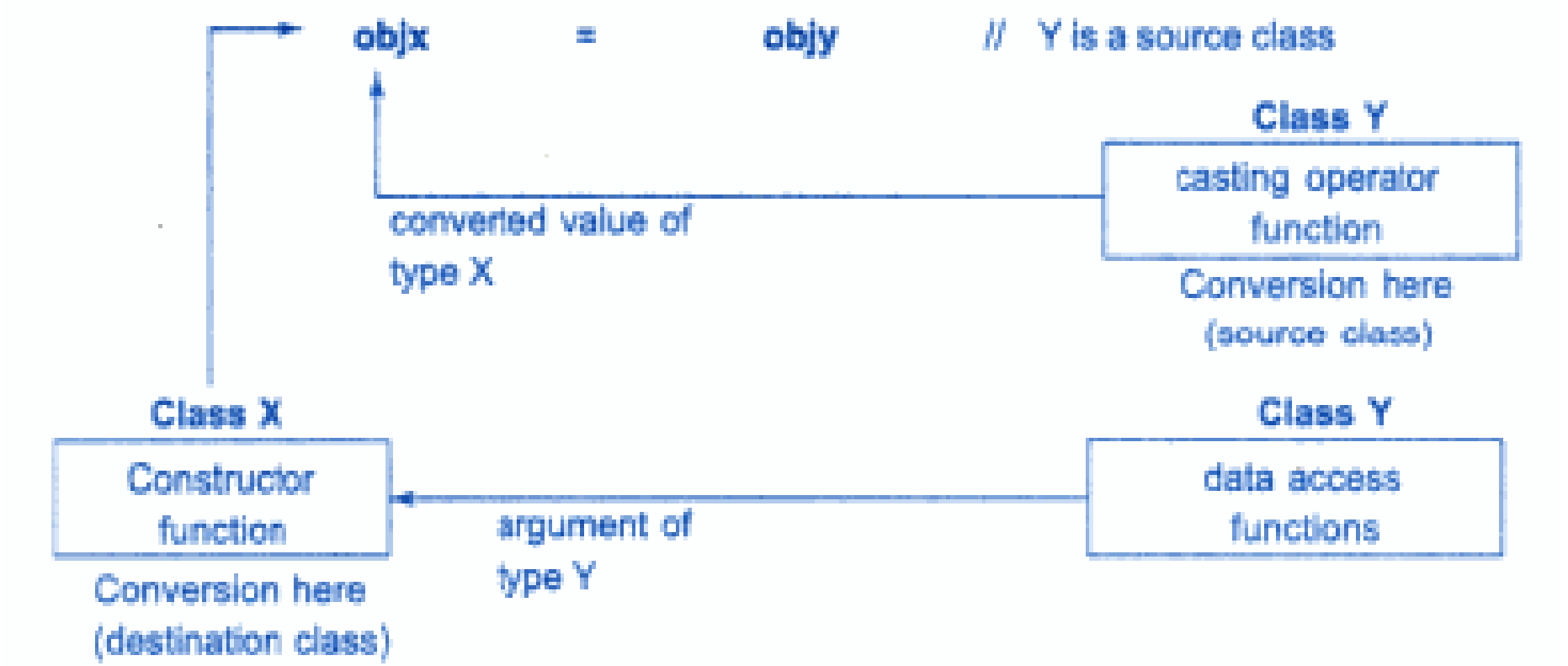
- Type conversion is a process of converting one data type to another.
- Type conversion is done automatically for the built-in datatypes by the compiler.
- Example:

```
int m;
float x = 3.14159;
m = x;
```

- The above statements convert x to an integer before assigning it to m.
- For user defined data type the compiler has to be provided with the type conversion function.

One class to Another class type

- When to use constructor and type conversion function?



One class to Another class type (Example)

- Consider an example of an inventory of products in store.
- One way of recording the details of the product is to record their code number, total items in the stock and the cost of each item.
- Another approach is to just specify the item code and the value of the item in the stock.
- Example.

Type conversion summary

Conversion required	*Conversion takes place in*	
	Source class	*Destination class*
Basic → class	Not applicable	Constructor
Class → basic	Casting operator	Not applicable
Class → class	Casting operator	Constructor

Short Answer Questions

- **Why is it necessary to overload and operator?**
  - Operator overloading allows us to provide new implementations for existing operators. Using operator overloading one can perform the operation on the user defined data types in the same way as that of built-in data types.
  - Example: Two integer values can be added as c = a + b. With the use of operator overloading the + operator can also be used to perform addition of two complex numbers as:

    C3 = C1 + C2 where C1, C2 and C3 are the objects of class complex.

Short Answer Questions

- **When is a friend function compulsory? Give example.**
  - Friend function is used in a situation where we need to use two different types of operands for a binary operator. One an object and another a built-in data type.

$$A = 2 + B$$

  - Here the statement will not be executed by the member functions as the left hand operand which is responsible for invoking the member function is not the object of the same class. However the friend function allows the statement to be executed.

End of unit