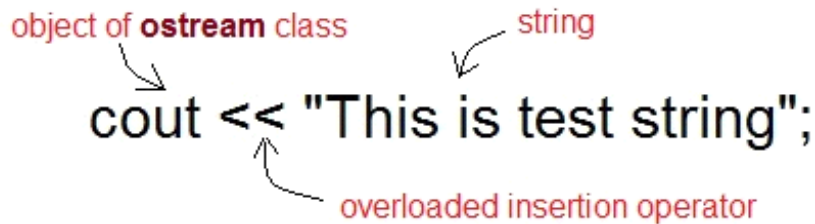


CHAPTER 8: OPERATOR OVERLOADING

Meaning and importance of operator overloading

Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.

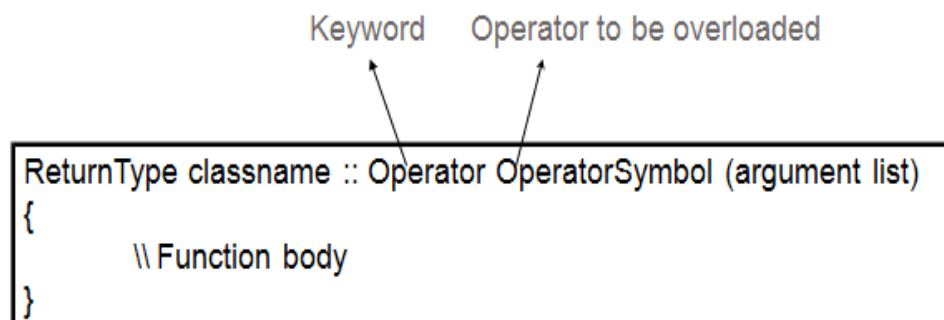


The diagram shows the code snippet `cout << "This is test string";`. Three annotations with arrows point to parts of the code: "object of ostream class" points to `cout`, "string" points to the string literal `"This is test string"`, and "overloaded insertion operator" points to the `<<` operator.

Almost any operator can be overloaded in C++. However there are few operator which can not be overloaded. **Operator that are not overloaded** are follows

1. scope operator - `::`
2. `sizeof`
3. member selector - `.`
4. member pointer selector - `*`
5. ternary operator - `?:`

Operator Overloading Syntax



The diagram shows the syntax for operator overloading: `ReturnType classname :: Operator OperatorSymbol (argument list)` followed by a function body in curly braces. Two arrows point from the labels "Keyword" and "Operator to be overloaded" to the `::` and `Operator` parts of the syntax respectively.

Implementing Operator Overloading

Operator overloading can be done by implementing a function which can be :

1. Member Function
2. Non-Member Function
3. Friend Function

Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading

function must be a non-member function.

Operator overloading function can be made friend function if it needs access to the private and protected members of class.

Restrictions on Operator Overloading

Following are some restrictions to be kept in mind while implementing operator overloading.

1. Precedence and Associativity of an operator cannot be changed.
2. Arity (numbers of Operands) cannot be changed. Unary operator remains unary, binary remains binary etc.
3. No new operators can be created, only existing operators can be overloaded.

Cannot redefine the meaning of a procedure. You cannot change how integers are added.

Operator Overloading Examples

Almost all the operators can be overloaded in infinite different ways. Following are some examples to learn more about operator overloading. All the examples are closely connected.

Overloading Arithmetic Operator

Arithmetic operators are most commonly used operators in C++. Almost all arithmetic operators can be overloaded to perform arithmetic operations on user-defined data types. In the below example we have overridden the + operator, to add to Time (hh:mm:ss) objects.

Example: overloading '+' Operator to add two time objects

```
#include< iostream.h>
#include< conio.h>
class time
{
    int h,m,s;
public:
    time()
    {
        h=0, m=0; s=0;
    }
    void getTime();
    void show()
    {
        cout<< h<< ":"<< m<< ":"<< s;
    }
    time operator+(time); //overloading '+' operator
};
time time::operator+(time t1) //operator function
{
    time t;
    int a,b;
    a=s+t1.s;
```

```

        t.s=a%60;
        b=(a/60)+m+t1.m;
        t.m=b%60;
        t.h=(b/60)+h+t1.h;
        t.h=t.h%12;
        return t;
    }
void time::getTime()
{
    cout<<"\n Enter the hour(0-11) ";
    cin>>h;
    cout<<"\n Enter the minute(0-59) ";
    cin>>m;
    cout<<"\n Enter the second(0-59) ";
    cin>>s;
}
void main()
{
    clrscr();
    time t1,t2,t3;
    cout<<"\n Enter the first time ";
    t1.getTime();
    cout<<"\n Enter the second time ";
    t2.getTime();
    t3=t1+t2; <font color="green">//adding of two time object using '+' operator
    cout<<"\n First time ";
    t1.show();
    cout<<"\n Second time ";
    t2.show();
    cout<<"\n Sum of times ";
    t3.show();
    getch();
}

```

Overloading I/O operator

1. Overloaded to perform input/output for user defined datatypes.
2. Left Operand will be of types **ostream&** and **istream&**
3. Function overloading this operator must be a Non-Member function because left operand is not an Object of the class.
4. It must be a friend function to access private data members.

You have seen above that **<<** operator is overloaded with **ostream** class object

cout to print primitive type value output to the screen. Similarly you can overload **<<** operator in your class to print user-defined type to screen. For example we will