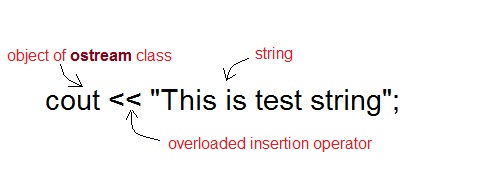
**Operator Overloading**

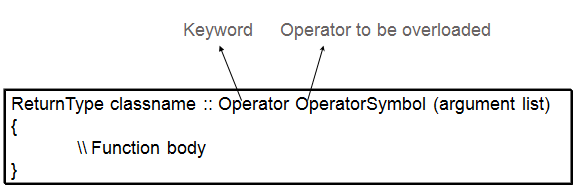
Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example '+' operator can be overloaded to perform addition on various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



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

* scope operator - ::
* sizeof
* member selector - .
* member pointer selector - \*
* ternary operator - ?:

**Operator Overloading Syntax**



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

**Example: overloading '+' Operator to add two time object**

#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; //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**

* Overloaded to perform input/output for user defined datatypes.
* Left Operand will be of types ostream& and istream&
* Function overloading this operator must be a Non-Member function because left operand is not an Object of the class.
* 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 overload **<<**in **time** class to display time object using cout.

time t1(3,15,48);

cout << t1;

**NOTE:** When the operator does not modify its operands, the best way to overload the operator is via friend function.

**Example: overloading '<<' Operator to print time object**

#include< iostream.h>

#include< conio.h>

class time

{

int hr,min,sec;

public:

time()

{

hr=0, min=0; sec=0;

}

time(int h,int m, int s)

{

hr=h, min=m; sec=s;

}

**friend ostream& operator << (ostream &out, time &tm);** //overloading '<<' operator

};

ostream& operator<< (ostream &out, time &tm) //operator function

{

out << "Time is " << tm.hr << "hour : " << tm.min << "min : " << tm.sec << "sec";

return out;

}

void main()

{

time tm(3,15,45);

cout << tm;

}

**Output**

Time is 3 hour : 15 min : 45 sec

**Overloading Relational operator**

You can also overload Relational operator like == , != , >= , <= etc. to compare two user-defined object.

**Example**

class time

{

int hr,min,sec;

public:

time()

{

hr=0, min=0; sec=0;

}

time(int h,int m, int s)

{

hr=h, min=m; sec=s;

}

**friend bool operator==(time &t1, time &t2);**//overloading '==' operator

};

**bool operator== (time &t1, time &t2)** //operator function

{

return ( t1.hr == t2.hr &&

t1.min == t2.min &&

t1.sec == t2.sec );

}

**Copy constructor Vs. Assignment operator**

**Assignment operator** is used to copy the values from one object to another **already existing object**. For example

time tm(3,15,45); //**tm** object created and initialized

time t1; //**t1** object created

t1 = tm; //initializing **t1** using **tm**

**Copy constructor** is a special constructor that initializes a **new object** from an existing object.

time tm(3,15,45); //**tm** object created and initialized

time t1(tm); //**t1** object created and initialized using **tm** object