UNIT -4

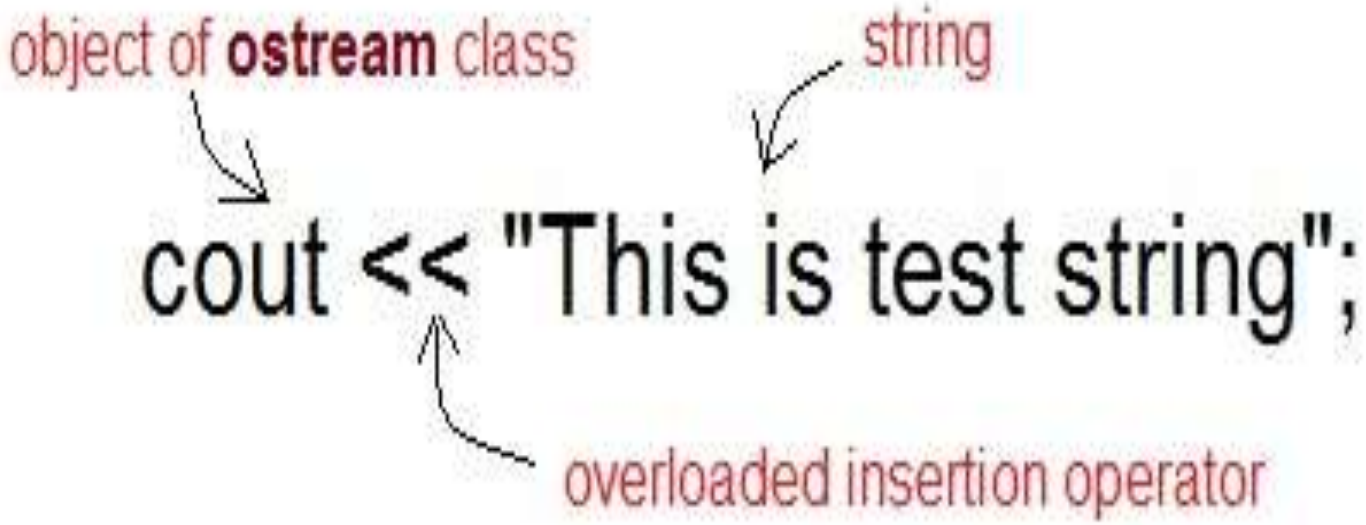
Operator Overloading

BCA-2nd Sem

Introduction 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.

Operator Overloading



Operator Overloading

- 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

Keyword Operator to be overloaded

```
ReturnType classname :: Operator OperatorSymbol (argument list)
{
        \\ Function body
}
```

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

Implementing Operator Overloading

- 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.

Restrictions on Operator Overloading

- Precedence and Associativity of an operator cannot be changed.
- Arity (numbers of Operands) cannot be changed. Unary operator remains unary, binary remains binary etc.
- 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.

Overloading Unary Operator using member Functions

There are two types of operator overloading:

- Unary operator overloading
- Binary operator overloading

Overloading Unary Operator using member Functions

```
#include<iostream>
using namespace std; {
 private:
 int count ;
 public: Inc() {
 count = 0 ;}
 Inc(int C) {
 count = C ; }
 Inc operator ++ () {
Inc(++count); }
```

Overloading Unary Operator using member Functions

```
Inc operator -- (){
 return Inc(--count);}
 void display(void){
cout << count << endl}};
 void main(void) {
 Inc a, b(4), c, d, e(1), f(4);
 cout << "Before using the operator ++()\n";
cout << "a = ";
```

Overloading Unary Operator using member Functions

```
a.display();
cout << "b = ";
 b.display();
 ++a;
 b++;
cout << "After using the operator ++()\n";
 cout << "a = ";
 a.display();
cout << "b = ";
 b.display();c = ++a; d = b++;
```

Overloading Unary Operator using member Functions

```
cout << "Result prefix (on a) and postfix (on b)\n";
cout << "c = ";
c.display();
 cout << "d = ";
d.display();
cout << "Before using the operator --()\n";
 cout << "e = ";
 e.display();
cout << "f = ";
 f.display();
```

Overloading Unary Operator using friend Functions

- When you overload a unary operator using a friend function you would have to pass one argument to the friend function.

- **Beware:** When you use friend functions, they will not have the ‘this’ pointer.
- If you attempt to modify some value of an object passed as argument, then the friend function actually only operates on a copy (it does not act on the original object).
- This is because it is passed by value (and not as reference).

Overloading Unary Operator using friend Functions

- Thus you would actually have to work on a copy of the object, and then return a newly initialized object having the modified values (we did the same process in using friend function for binary operator overloading).
- To work directly on the original object, you can make use of reference parameters in the operator overloaded friend function.

Overloading Unary Operator using friend Functions

```
#include<iostream.h>
#include<conio.h>
class point
{
private:
int x,y;
public:
point(int i,int j)
{
```

Overloading Unary Operator using friend Functions

```
x=i;
y=j;
}
void show()
{
cout<<"point="<<"("<<x<<","<<y<<")"<<endl;
}
```

Overloading Unary Operator using friend Functions

```
friend void operator++(point &p1);
};
void operator++(point &p1)
{
p1.x++;
p1.y++;
}
```

Overloading Unary Operator using friend Functions

```
void main()
{
clrscr();
point p1(2,2);
p1.show();
cout<<"after increment"<<endl;
++p1;
p1.show();
getch();}
```

Overloading Binary Operator using member Functions

```
#include<iostream.h>
include<conio.h>
#include<stdlib.h>
class stud{
int rno;
char *name;
int marks;
public:
```

Overloading Binary Operator using member Functions

```
friend istream &operator>>(istream &,stud &);
friend void operator<<(ostream &,stud &);
};
```

Overloading Binary Operator using member Functions

```
istream &operator>>(istream &tin,stud &s)
{
cout<<"\n Enter the no";
tin>>s.rno;
#cout<<"\n Enter the name";
tin>>s.name;
```

Overloading Binary Operator using member Functions

```
cout<<"\n Enter Marks";
tin>>s.marks;
return tin;
}
void operator<<(ostream &tout,stud &s)
{
```

Overloading Binary Operator using member Functions

```
tout<<”\n”<<s.rno;
tout<<”\n”<<s.name;
tout<<”\n”<<s.marks;
}
void main()
{
clrscr();
cout<<"\t\tBinaryOperator Overloading
   Using FriendFunction";
```

Overloading Binary Operator using member Functions

```
stud s;
for(int i=0;i<3;i++)
{
cin>>s;
}
for( i=0;i<3;i++)
{
cout<<s;
}getch();
```

Overloading Binary Operator using friend Functions

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class stud
{
int rno;
char *name;
int marks;
public:
friend istream
    &operator>>(istream
    &,stud &);
friend void
    operator<<(ostream
    &,stud &);
};
```

Overloading Binary Operator using friend Functions

```
istream&operator>>(istream &tin,stud &s){
cout<<"\n Enter the no";
tin>>s.rno;
cout<<"\n Enter the name";
tin>>s.name;
cout<<"\n Enter Marks";
tin>>s.marks;
   return tin;
```

Overloading Binary Operator using friend Functions

```
}void operator<<(ostream &tout,stud &s)
{tout<<”\n”<<s.rno;
cout<<"\n Enter Marks";
tin>>s.marks;
return tin;}
void operator<<(ostream &tout,stud &s)
{tout<<”\n”<<s.rno;
```

Rules for Operator Overloading.

1) Only built-in operators can be overloaded. New operators can not be created.

2) Arity of the operators cannot be changed.

3) Precedence and associativity of the operators cannot be changed.

Rules for Operator Overloading.

4) Overloaded operators cannot have default arguments except the function call operator () which can have default arguments.

5) Operators cannot be overloaded for built in types only. At least one operand must be used defined type.

Rules for Operator Overloading

6) Assignment (=), subscript ([]), function call (“()”), and member selection (->) operators must be defined as member functions.

7) Except the operators specified in point 6, all other operators can be either member functions or a non member functions.

8) Some operators like (assignment)=, (address)& and comma (,) are by default overloaded.

REFRENCES

- Learn Programming in C++ By Anshuman Sharma, Anurag Gupta, Dr.Hardeep Singh, Vikram Sharma