# CLASS MEMBER ACCESS OPERATOR − > OVERLOADING IN C++



The class member access operator − > can be overloaded but it is bit trickier. It is defined to give a class type a "pointer-like" behavior. The operator -> must be a member function. If used, its return type must be a pointer or an object of a class to which you can apply.

The operator-> is used often in conjunction with the pointer-dereference operator * to implement "smart pointers." These pointers are objects that behave like normal pointers except they perform other tasks when you access an object through them, such as automatic object deletion either when the pointer is destroyed, or the pointer is used to point to another object.

The dereferencing operator-> can be defined as a unary postfix operator. That is, given a class:

```
class Ptr{
   //...
   X * operator->();
};
```

Objects of class **Ptr** can be used to access members of class **X** in a very similar manner to the way pointers are used. For example:

```
void f(Ptr p )
{
   p->m = 10 ; // (p.operator->())->m = 10
}
```

The statement p->m is interpreted as $p.operator->()$->m. Using the same concept, following example explains how a class access operator -> can be overloaded.

```
#include <iostream>
#include <vector>
using namespace std;

// Consider an actual class.
class Obj {
   static int i, j;
public:
   void f() const { cout << i++ << endl; }
   void g() const { cout << j++ << endl; }
};

// Static member definitions:
int Obj::i = 10;
int Obj::j = 12;

// Implement a container for the above class
class ObjContainer {
   vector<Obj*> a;
public:
   void add(Obj* obj)
   {
      a.push_back(obj);  // call vector's standard method.
   }
   friend class SmartPointer;
};

// implement smart pointer to access member of Obj class.
class SmartPointer {
   ObjContainer oc;
   int index;
public:
   SmartPointer(ObjContainer& objc)
```

```
   {
       oc = objc;
       index = 0;
   }
   // Return value indicates end of list:
   bool operator++() // Prefix version
   {
     if(index >= oc.a.size()) return false;
     if(oc.a[++index] == 0) return false;
     return true;
   }
   bool operator++(int) // Postfix version
   {
      return operator++();
   }
   // overload operator->
   Obj* operator->() const
   {
     if(!oc.a[index])
     {
        cout << "Zero value";
        return (Obj*)0;
     }
     return oc.a[index];
   }
};

int main() {
   const int sz = 10;
   Obj o[sz];
   ObjContainer oc;
   for(int i = 0; i < sz; i++)
   {
       oc.add(&o[i]);
   }
   SmartPointer sp(oc); // Create an iterator
   do {
      sp->f(); // smart pointer call
      sp->g();
   } while(sp++);
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
10
12
11
13
12
14
13
15
14
16
15
17
16
18
17
19
18
20
19
21
```