

C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com



Friend function

- If we want to access private members inside derived class
 - Either we should use member function(getter/setter).
 - Or we should declare a facilitator function as a friend function.
 - Or we should declare derived class as a friend inside base class.



Operator Overloading

- operator is token in C/C++.
- It is used to generate expression.
- operator is keyword in C++.
- Types of operator:
 - Unary operator (++,--,&!,~,sizeof())
 - Binary Operator (Arithmetic, relational, logical , bitwise, assignment)
 - Ternary operator (conditional)
- In C++, also we can not use operator with objects of user defined type directly.
- If we want to use operator with objects of user defined type then we should overload operator.
- To overload operator, we should define **operator function**.
- **We can define operator function using 2 ways:**
 - Using member function
 - Using non member function



Program Demo without operator overloading

- Create a class point, having two fields, x, y.
 - Point(void)
 - Point(int x, int y)

```
int main( void )  
{  
    Point pt1(10,20);  
    Point pt2(30,40);  
    Point pt3;  
    pt3 = pt1 + pt2; //Not OK( implicitly)  
    return 0;  
}
```



Need Of Operator Overloading

- we extend the meaning of the operator.
- If we want to use operator with the object of use defined type, then we need to overload operator.
- To overload operator, we need to define operator function.
- In C++, operator is a keyword
 - Suppose we want to use plus(+) operator with objects then we need to define operator+() function.

We define operator function either inside class (as a member function) or outside class (as a non-member function).

Point pt1(10,20), pt2(30,40), pt3;

pt3 = pt1 + pt2; //pt3 = pt1.operator+(pt2); //using member function

OR

pt3 = pt1 + pt2; //pt3 = operator+(pt1, pt2); //using non member function



Operator Overloading

using member function

- operator function must be member function
- If we want to overload, binary operator using member function then operator function should take only one parameter.
- Example : `c3 = c1 + c2;` //will be called as
 - `c3 = c1.operator+(c2)`

• Example :

`Point operator+(Point &other) //Member Function`

```
{
    Point temp;
    temp.xPos = this->xPos + other.xPos;
    temp.yPos = this->yPos + other.yPos;
    return temp;
}
```

using non member function

- Operator function must be global function
- If we want to overload binary operator using non member function then operator function should take two parameters.
- Example : `c3 = c1 + c2;` //will be called as
 - `c3 = operator+(c1,c2);`

• Example:

`Point operator+(Point &pt1, Point &pt2) //Non Member Function`

```
{
    Point temp;
    temp.xPos = pt1.xPos + pt2.xPos;
    temp.yPos = pt1.yPos + pt2.yPos;
    return temp;
}
```



We can not overload following operator using member as well as non member function

1. dot/member selection operator(.)
2. Pointer to member selection operator(.*)
3. Scope resolution operator(::)
4. Ternary/conditional operator(? :)
5. sizeof() operator
6. typeid() operator
7. static_cast operator
8. dynamic_cast operator
9. const_cast operator
10. reinterpret_cast operator



We can not overload following operators using non member function:

- Assignment operator(=)
- Subscript / Index operator([])
- Function Call operator[()]
- Arrow / Dereferencing operator(->)



Template

- If we want to write generic program in C++, then we should use template.
- This feature is mainly designed for implementing generic data structure and algorithm.
- If we want to write generic program, then we should pass data type as a argument. And to catch that type we should define template.
- Using template we can not reduce code size or execution time but we can reduce developers effort.



Template

<pre>int num1 = 10, num2 = 20; swap_object<int>(num1, num2); string str1="Pune", str2="Karad"; swap_object<string>(str1, str2);</pre>	<p>In this code, <int> and <string> is considered as type argument.</p>
<pre>template<typename T> //or template<class T> //T : Type Parameter void swap(b obj1, T obj2) { T temp = obj1; obj1 = obj2; obj2 = temp; }</pre>	<p>template and typename is keyword in C++. By passing datatype as argument we can write generic code hence parameterized type is called template</p>

- **Types of Template**

- Function Template
- Class Template



Example of Function Template

//template<typename T>//T : Type Parameter

template<class T> //T : Type Parameter

void swap_number(T &o1, T &o2)

{ T temp = o1;

o1 = o2;

o2 = temp;

}

int main(void)

{

int num1 = 10;

int num2 = 20;

swap_number<int>(num1, num2); //Here int is type argument

cout<<"Num1 : "<<num1<<endl;

cout<<"Num2 : "<<num2<<endl;

return 0;

}



Example of Class Template

```
template<class T>
class Array // Parameterized type
{
private:
    int size;
    T *arr;
public:
    Array( void ) : size( 0 ), arr( NULL )
    {
    }
    Array( int size )
    {
        this->size = size;
        this->arr = new T[ this->size ];
    }
    void acceptRecord( void ){}
    void printRecord( void ){ }
    ~Array( void ){ }
};
```

```
int main(void)
{
    Array<char> a1( 3 );
    a1.acceptRecord();
    a1.printRecord();
    return 0;
}
```



Thank You

