# 【C++】Day65

| | |
|---|---|
| Class | C++ |
| Date | @February 28, 2022 |
| Material | |
| Series Number | |
| Summary | Overloaded Subscript, Increment, and Decrement Operator |

## 【Ch14】Overloaded Operations and Conversions

### 14.5 Subscript Operator

Classes that represent containers from which elements can be retrieved by position often define the subscript operator, `operator[]`.

*Note: The subscript operator must be a member function.*

The subscript operator usually returns a reference to the elements that is fetched. By returning a reference, subscript can be used on either side of an assignment.

Consequently, it is also usually a good idea to define both const and nonconst versions of this operator. When applied to a `const` object, subscript should return a reference to const so that it is not possible to assign to the returned object.

*Best Practices: If a class has a subscript operator, it usually should define two versions: one that returns a plain reference and the other that is a const member and returns a reference to const.*

As an example, we'll define subscript for `StrVec`:

```
class StrVec {
public:
  std::string &operator[](size_t n) {
    return elements[n];
  }
```

```
  const string &operator(size_t n) const {
    return elements[n];
  }
};
```

## 14.6 Increment and Decrement Operators

Because these operators change the state of the object on which they operate, our preference is to make the increment and decrement operators members.

We can define both the prefix versions and the postfix ones.

*Best Practices: Classes that define increment or decrement operators should define both the prefix and postfix versions. These operators usually should be defined as members.*

### *Defining Prefix Operators*

We'll define these operators for our `StrBlobPtr` class:

```
class StrBlobPtr {
public:
  // increment and decrement
  StrBlobPtr &operator++();
  StrBlobPtr &operator--();
};
```

*Best Practices: To be consistent with the built-in operators, the prefix operators should return a reference to the incremented or decremented object.*

In this case of increment, we pass the current value of `curr` to check. If that value is less than the size of the underlying vector, check will return.

```
StrBlobPtr StrBlobPtr::&operator++() {
  check(curr, "increment past end of StrBlobPtr");
  ++curr;
  return *this;
}
```

The decrement operator decrements curr before calling check. That way, if curr is already zero, the value that wee pass to check will be a large positive value representing an invalid subscript.

```
StrBlobPtr &StrBlobPtr::operator--() {
  // if curr is zero, decrementing it will yield an invalid subscript
  --cur;
  check(-1, "decrement past begin of StrBlobPtr");
  return *this;
}
```

### *Differentiating Prefix and Postfix Operators*

To differentiate the prefix and postfix operators, the postfix versions take an extra(unused) parameter of type int. When we use a postfix operator, the compiler supplies 0 as the argument for this parameter.

Although the postfix function can use this extra parameter, it usually should not. That parameter is not needed for the work normally performed by a postfix operator. Its sole purpose is to distinguish a postfix function from the prefix version.

We can now add the postfix operators to `StrBlobPtr`:

```
class StrBlobPtr {
public:
  StrBlobPtr operator++(int);
  StrBlobPtr operator--(int);
};
```

*Best Practices: To be consistent with the built-in operators, the postfix operators should return the old value. That value is returned as a value, not a reference.*

The postfix versions have to remember the current state of the object before incrementing the object

```
StrBlobPtr StrBlobPtr::operator++(int) {
  // no check needed here; the call to prefix increment will do the check
  StrBlobPtr ret = *this; // save the current value
  ++*this;
  return ret;
}

StrBlobPtr StrBlobPtr::operator--(int) {
  StrBlobPtr ret = *this;
  --*this;
  return ret;
}
```

Each of our operator calls its own prefix version to do the actual work. For example, the postfix increment operator executes

```
++*this
```

This expression calls the prefix increment operator. That operator checks that the increment is safe and either throws an exception or increments curr.

*Note: The int parameter is not used, so we do not give it a name.*