【C++】Day24(2)

Class	C++
Date	@December 15, 2021
Material	
Series Number	
Summary	Return *this

【Ch7】Classes

7.3.2 Functions that Return *this

Next we'll add functions to set the character at the cursor or at a given location:

```
class Screen {
public:
  Screen &set(char);
  Screen &set(pos, pos, char);
  //other members as before
};

inline Screen &Screen::set(char c) {
  contents[cursor] = c; //set the new value at the current cursor location
  return *this; //return this object as an lvalue
}

inline Screen &Screen::set(pos r, pos col, char ch) {
  contents[r * width + col] = ch; //set specified location to given value
  return *this;
}
```

Functions that return a reference are lvalues, which means that they return the object itself, not a copy of the object.

If we concatenate a sequence of these actions into a single expression:

```
//move the cursor to a given position, and set that character
myScreen.move(4, 0).set('#');
```

these operations will execute on the same object.

If we define `move` and `set` to return `Screen` rather than `Screen&`, this statement would execute quite differently. In this case, it would be equivalent to:

```
//if move returns Screen not Screen&
Screen temp = myScreen.move(4, 0); //the return value would be copied
temp.set('#'); //the contents inside myScreen would be unchanged
```

If move had a nonreference return type, then the return value of move would be a copy of `*this`. The call to `set` would change the temporary copy, not `myScreen`.

*Returning *this from a const Member Function*

We define a new function display to display the contents of our Screen. However, if display is a const member, then this is a pointer to const and *this is a const object. Hence, the return type of display must be const.

However, if display returns a reference to const, we won't be able to embed display into a series of actions:

```
Screen myScreen;
//if display returns a const reference, the call to set is an error
myScreen.display(cout.set('*');
```

*Note: A const member function that returns *this as a reference should have a return type that is a reference to const.*

Overloading Based on const

We can overload a member function based on whether it is const for the same reason that we can overload a function based on whether a pointer parameter points to const.

In this case, we will define two display functions for Screen:

```
class Screen {
public:
```

```
  //display overloaded on whether this is const or not
  Screen &display(std::ostream os) {
    do_display(os);
    return *this;
  }

  const Screen &display(std::ostream os) const {
    do_display(os);
    return *this;
  }
private:
  //function that do the display for Screen
  void do_display(std::ostream os) const {
    os << contents;
  }
};
```