【C++】Day59

Class	C++
Date	@February 20, 2022
Material	
Series Number	
Summary	

【Ch13】Copy Control

13.3 Swap

Classes that manage resources often also define a function named `swap`. Defining `swap` is particularly important for classes that we plan to use with algorithms that reorder elements. Such algorithms calls `swap` whenever they need to exchange two elements.

If a class defines its own `swap`, then the algorithm uses that class-specific version. Otherwise, it uses the `swap` function defined by the library.

For example, code to swap two objects of our valuelike `HasPtr` class might look something like this:

```
HasPtr temp = v1; // make a temporary copy of the value of v1
v1 = v2;
v2 = temp;
```

In principle, none of the memory allocation is necessary. Rather than allocating new copies of the string, we'd like swap to swap the pointers.

```
string *temp = v1.ps;
v1.ps = v2.ps;
v2.ps = temp;
```

Writing Our Own Swap Function

We can override the default behaviour of `swap` by defining a version of swap that operates on our class. The typical implementation of swap is:

```
class HasPtr {
  friend void swap(HasPtr&, HasPtr&);
private:
  string *ps;
  int i;
};

inline void swap(HasPtr& v1, HasPtr& v2) {
  using std::swap;
  swap(v1.ps, v2.ps); // swap the pointers not the string data
  swap(v1.i, v2.i); // swap the int members
}
```

Note: Unlike the copy-control members, swap is never necessary. However, defining swap can be an important optimization for classes that allocate resources.

Using swap in Assignment Operators

Classes that define `swap` often use `swap` to define their assignment operator. These operators use a technique known as copy and swap. This technique swaps the left-hand operand with a copy of the right-hand operand.

```
// note rhs is passed by value, which means the HasPtr copy constructor
// copies the string in the right-hand operand into rhs
HasPtr& HasPtr::operator=(HasPtr rhs) {
  // swap the contents of the left-hand operand with the local variable rhs
  swap(*this, rhs); // rhs now points to the memory this object has used
  return *this; // rhs is destroyed, which deletes the pointer in rhs
}
```

Exercise

Exercise 13.30: Write and test a `swap` function for your valuelike version of `HasPtr`. Give your `swap` a print statement that notes when it is executed.

Exercise 13.31: Give your class a `<` operator and define a `vector` of `HasPtr`s. Give that `vector` some elements and then `sort` the `vector`.

See 13_30.cpp for code