
4.6. Allocators

In several places, the C++ standard library uses special objects to handle the allocation and deallocation of memory. Such objects are called *allocators*. They represent a special memory model and are used as an abstraction to translate the *need* to use memory into a raw *call* for memory. The use of different allocator objects at the same time allows you to use different memory models in a program.

Originally, allocators were introduced as part of the STL to handle the nasty problem of different pointer types on PCs (such as near, far, and huge pointers). Now, allocators serve as a base for technical solutions that use certain memory models, such as shared memory, garbage collection, and object-oriented databases, without changing the interfaces. However, this use is relatively new and not yet widely adopted (this will probably change).

The C++ standard library defines a *default allocator* as follows:

```
namespace std {
    template <typename T>
    class allocator;
}
```

An object of the default allocator type is used as the default value everywhere an allocator can be used as an argument. It does the usual calls for memory allocation and deallocation; that is, it calls the `new` and `delete` operators. However, when or how often these operators are called is unspecified. Thus, an implementation of the default allocator might, for example, cache the allocated memory internally.

The default allocator is used in most programs. However, other libraries sometimes provide allocators to fit certain needs. In such cases, you must simply pass them as arguments. Only occasionally does it make sense to program allocators. In practice, the default allocator is typically used. The discussion of allocators is deferred until Chapter 19, which covers in detail not only allocators but also their interfaces.