Inside STL: The vector

devblogs.microsoft.com/oldnewthing/20230802-00

August 2, 2023

Raymond Chen

The C++ language comes with a standard library, and although implementations are welcome to implement the library types in whatever manner they choose, they are constraints imposed by the standard which often force one of a small number of possible implementations.

The `std::vector` is one of those types which is constrained to the point that there's really only one viable implementation.

Internally, a `std::vector` basically looks like this:

```
template<typename T>
struct vector
{
    T* first;
    T* last;
    T* end;
};
```

The `first` is a pointer to the beginning of a single memory allocation that holds the vector contents.

The `last` is a pointer one past the end of the last valid vector element.

The `end` is a pointer one past the end of the last allocated memory for the vector.

The picture for this is as follows:

The `first` points to the start of the allocation.

The memory between `first` and `last` is used to hold live elements of the vector. The `size()` of the vector is `last - first`.

The memory between `last` and `end` is not being used for anything. The `capacity()` of the vector is `end - first`.

If you actually dump the `std::vector` in the debugger, you'll see that it's actually more complicated than this due to the need to store an allocator. The allocator is usually an empty class (the default allocator certainly is), so I didn't show it in the conceptual version above.

Since the allocator is usually an empty class, it is stored as a compressed pair with one of the other members so that it usually occupies no memory.

The Visual Studio debugger contains a visualizer to view the contents of a `std::vector` more conveniently, but if you need to dig out the three magic pointers yourself, here's how you can do it with the Microsoft implementation of the standard library:

```
0:001> ?? v
class std::vector<T,std::allocator<T> >
   +0x000 _Mypair          :
std::_Compressed_pair<std::allocator<T>,std::_Vector_val<std::_Simple_types<T> >,1>
0:000> ?? v._Mypair
class std::_Compressed_pair<std::allocator<T>,std::_Vector_val<std::_Simple_types<T>
>,1>
   +0x000 _Myval2          : std::_Vector_val<std::_Simple_types<T> >
0:000> ?? v._Mypair._Myval2
class std::_Vector_val<std::_Simple_types<T> >
   +0x000 _Myfirst         : T* ⇐
   +0x008 _Mylast          : T* ⇐
   +0x010 _Myend           : T* ⇐
```

The language requirements pretty much require this layout for a vector because the `data()` method must return a pointer to contiguous memory, and the constraints on the `capacity()` require that the excess elements come immediately after.

In particular, the language forbids a "small vector optimization" analogous to the small string optimization (which we'll see soon), because the standard requires that[1] moving a vector not invalidate iterators or references. The vector elements have to remain where they are. A small-vector optimization would put them in vector-internal storage, which would require that the element move from one vector to another on a vector move operation, which is not allowed.

[1] Assuming you're using a well-behaved allocator.