Inside STL: The array

devblogs.microsoft.com/oldnewthing/20230811-00

August 11, 2023

Raymond Chen

The C++ standard library `array` is just a C-style array wrapped inside a class so that it behaves like a normal C++ object instead of a wacky thing that undergoes decay.

```
template<typename T, size_t N>
class array
{
    T elements[N];
};

template<typename T>
class array<T, 0>
{
};
```

The only weird case is *N* = 0. You are allowed to create a zero-length `std::array`, but C++ does not allow zero-length C-style arrays. The zero-length `std::array` is just an empty class.

Visual Studio and the Windows debugger come with a visualizer:

```
0:000> dx a
a                 : { size=5 } [Type: std::array<int,5>]
    [<Raw View>]     [Type: std::array<int,5>]
    [0]              : 3 [Type: int]
    [1]              : 1 [Type: int]
    [2]              : 4 [Type: int]
    [3]              : 1 [Type: int]
    [4]              : 5 [Type: int]
```

Or you can just dig out the elements yourself.

```
0:000> ?? a
class std::array<int,5>
   +0x000 _Elems           : [5] 0n3
0:000> ?? a._Elems[1]
int 0n1
```

The `dt` command with the `-a` option (array) is handy for dumping C-style arrays.

```
0:000> ?? &a
class std::array<int,5> * 0x00000094`bed0f698
   +0x000 _Elems           : [5] 0n3
0:000> dt -a5 int 0x00000094`bed0f698
[0] @ 00000094`bed0f698
---------------------------------------------
0n3

[1] @ 00000094`bed0f69c
---------------------------------------------
0n1

[2] @ 00000094`bed0f6a0
---------------------------------------------
0n4

[3] @ 00000094`bed0f6a4
---------------------------------------------
0n1

[4] @ 00000094`bed0f6a8
---------------------------------------------
0n5
```

The simplicity of the `std::array` is a nice break from the complexity of `std::deque`.