16.12 — std::vector<bool>

In lesson O.1 -- Bit flags and bit manipulation via std::bitset (https://www.learncpp.com/cpp-tutorial/bit-flags-and-bit-manipulation-via-stdbitset/), we discussed how std::bitset has the capability to compact 8 Boolean values into a byte. Those bits can then be modified via the member functions of std::bitset.

std::vector has an interesting trick up its sleeves. There is a special implementation for
std::vector<bool> that may be more space efficient for Boolean values by similarly
compacting 8 Boolean values into a byte.

For advanced readers

When a template class has a different implementation for a particular template type argument, this is called **class template specialization**. We discuss this topic further in lesson <u>26.4 -- Class template specialization (https://www.learncpp.com/cpp-tutorial/class-template-specialization/)</u>.

Unlike std::bitset, which was designed for bit manipulation, std::vector<bool> lacks bit manipulation member functions.

Using std::vector<bool>

For the most part, std::vector<bool> works just like a normal std::vector:

```
#include <iostream>
#include <vector>
int main()
{
    std::vector<bool> v { true, false, false, true, true };

    for (int i : v)
        std::cout << i << ' ';
    std::cout << '\n';

    // Change the Boolean value with index 4 to false
    v[4] = false;

    for (int i : v)
        std::cout << i << ' ';
    std::cout << '\n';
    return 0;
}</pre>
```

On the author's 64-bit machine, this prints:

```
1 0 0 1 1
1 0 0 1 0
```

std::vector<bool> tradeoffs

However, std::vector<bool> has some tradeoffs that users should be aware of.

First, std::vector<bool> has a fairly high amount of overhead (sizeof(std::vector<bool>) is 40 bytes on the author's machine), so you won't save memory unless you're allocating more Boolean values than the overhead for your architecture.

Second, the performance of std::vector<bool> is highly dependent upon the implementation (as implementations aren't even required to do optimization, let alone do it well). Per this article (https://isocpp.org/blog/2012/11/on-vectorbool), a highly optimized implementation can be significantly faster than alternatives. However, a poorly optimized implementation will be slower.

Third and most importantly, std::vector<bool> is not a vector (it is not required to be contiguous in memory), nor does it hold bool values (it holds a collection of bits), nor does it meet C++'s definition of a container.

. . .

0

Although std::vector<bool> behaves like a vector in most cases, it is not fully compatible with the rest of the standard library. Code that works with other element types may not work with std::vector<bool>.

For example, the following code works when T is any type except bool:

```
template<typename T>
void foo( std::vector<T>& v )
{
   T& first = v[0]; // get a reference to the first element
   // Do something with first
}
```

Avoid std::vector<bool>

The modern consensus is that std::vector<bool> should generally be avoided, as the performance gains are unlikely to be worth the incompatibility headaches due to it not being a proper container.

Unfortunately, this optimizing version of std::vector<bool> is enabled by default, and there is no way to disable it in favor of a nonoptimized version that is actually a container. There have been calls to deprecate std::vector<bool>, and work is underway to determine
what a replacement compacted vector of bool might look like (perhaps as a future std::dynamic bitset).

. . .

- Use (constexpr) std::bitset when the number of bits you need is known at compile-time, you don't have more than a moderate number of Boolean values to store (e.g. under 64k), and the limited set of operators and member functions (e.g. lack of iterator support) meets your requirements.
- Prefer std::vector<char> when you need a resizable container of Boolean values and space-savings isn't a necessity. This type behaves like a normal container.
- Favor a 3rd party implementation of a dynamic bitset (such as boost::dynamic_bitset) when you need a dynamic bitset to do bit operations on. Such types won't pretend to be standard library containers when they aren't.

Best practice

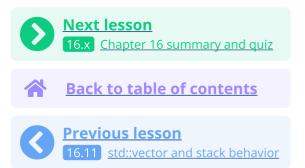
@ Email*

provided email address.

if Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/ are connected to your

Favor constexpr std::bitset, std::vector<char>, or 3rd party dynamic bitsets over std::vector<bool>.



Leave a comment...

Notify me about replies: POST COMMENT

. . .

4 COMMENTS Newest •

?

We and our partners share information on your use of this website to help improve your experience.

×

Do not sell my info:

OKAY