

**Username:** Pralay Patoria **Book:** The C++ Standard Library: A Tutorial and Reference, Second Edition. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

---

## 19.1. Using Allocators as an Application Programmer

For the application programmer, using different allocators should be no problem. You simply have to pass the allocator as a template argument. For example, the following statements create different containers and strings, using the special allocator `MyAlloc<>` :

[Click here to view code image](#)

```
// a vector with special allocator
std::vector<int, MyAlloc<int>> v;

// an int/float map with special allocator
std::map<int, float, std::less<int>,
        MyAlloc<std::pair<const int, float>>> m;

// a string with special allocator
std::basic_string<char, std::char_traits<char>, MyAlloc<char>> s;
```

If you use your own allocator, it probably is a good idea to make some type definitions. For example:

[Click here to view code image](#)

```
// special string type that uses special allocator
typedef std::basic_string<char, std::char_traits<char>,
                        MyAlloc<char>> MyString;

// special string/string map type that uses special allocator
typedef std::map<MyString, MyString, std::less<MyString>,
                MyAlloc<std::pair<const MyString, MyString>>> MyMap;

// create object of this type
MyMap mymap;
```

Since C++11, you can use *alias templates* (template typedefs; [see Section 3.1.9, page 27](#)) to define the allocator type while leaving the element type open:

[Click here to view code image](#)

```
template <typename T>
using Vec = std::vector<T, MyAlloc<T>>;    // vector using own allocator

Vec<int> coll;    // equivalent to: std::vector<int, MyAlloc<int>>
```

When you use objects with other than the default allocator, you'll see no difference.

You can check whether two allocators use the same memory resource by using operators `==` and `!=`. If operator `==` returns `true`, you can deallocate storage allocated from one allocator via the other. To access the allocator, all types that are parametrized by an allocator provide the member function `get_allocator()`. For example:

[Click here to view code image](#)

```
if (mymap.get_allocator() == s.get_allocator()) {
    //OK, mymap and s use the same or interchangeable allocators
    ...
}
```

In addition, since C++11, a type trait ([see Section 5.4, page 122](#)) is provided to check whether a type `T` has an `allocator_type`, which a passed allocator may be converted into:

[Click here to view code image](#)

```
std::uses_allocator<T, Alloc>::value    // true if Alloc is convertible
                                        // into T::allocator_type
```