Chapter 20. Boost.Container

Boost.Container is a Boost library that provides the same containers as the standard library. Boost.Container focuses on additional flexibility. For example, all containers from this library can be used with Boost.Interprocess in shared memory – something that is not always possible with containers from the standard library.

Boost.Container provides additional advantages:

- The interfaces of the containers resemble those of the containers in the C++11 standard library. For example, they provide member functions such as `emplace_back()`, which you can use in a C++98 program even though it wasn't added to the standard library until C++11.
- With `boost::container::slist` or `boost::container::stable_vector`, Boost.Container offers containers the standard library doesn't provide.
- The implementation is platform independent. The containers behave the same everywhere. You don't need to worry about possible differences between implementations of the standard library.
- The containers from Boost.Container support *incomplete types* and can be used to define recursive containers.

Example 20.1 illustrates incomplete types.

Note

The examples in this chapters cannot be compiled with Visual C++ 2013 and Boost 1.55.0. This bug is described in ticket 9332. It was fixed in Boost 1.56.0.

Example 20.1. Recursive containers with Boost.Container

```
#include <boost/container/vector.hpp>

using namespace boost::container;

struct animal
{
  vector<animal> children;
};

int main()
{
  animal parent, child1, child2;
  parent.children.push_back(child1);
  parent.children.push_back(child2);
}
```

The class `animal` has a member variable **`children`** of type `boost::container::vector<animal>`. `boost::container::vector` is defined in the header file `boost/container/vector.hpp`. Thus, the type of the member variable **`children`** is based

on the class `animal`, which defines the variable **`children`**. At this point, `animal` hasn't been defined completely. While the standard doesn't require containers from the standard library to support incomplete types, recursive containers are explicitly supported by Boost.Container. Whether containers defined by the standard library can be used recursively is implementation dependent.

Example 20.2. Using `boost::container::stable_vector`

```
#include <boost/container/stable_vector.hpp>
#include <iostream>

using namespace boost::container;

int main()
{
  stable_vector<int> v(2, 1);
  int &i = v[1];
  v.erase(v.begin());
  std::cout << i << '\n';
}
```

Boost.Container provides containers in addition to the well-known containers from the standard library. Example 20.2 introduces the container `boost::container::stable_vector`, which behaves similarly to `std::vector`, except that if `boost::container::stable_vector` is changed, all iterators and references to existing elements remain valid. This is possible because elements aren't stored contiguously in `boost::container::stable_vector`. It is still possible to access elements with an index even though elements are not stored next to each other in memory.

Boost.Container guarantees that the reference **`i`** in Example 20.2 remains valid when the first element in the vector is erased. The example displays **`1`**.

Please note that neither `boost::container::stable_vector` nor other containers from this library support C++11 initializer lists. In Example 20.2 **`v`** is initialized with two elements both set to 1.

`boost::container::stable_vector` is defined in `boost/container/stable_vector.hpp`.

Additional containers provided by Boost.Container are `boost::container::flat_set`, `boost::container::flat_map`, `boost::container::slist`, and `boost::container::static_vector`:

- `boost::container::flat_set` and `boost::container::flat_map` resemble `std::set` and `std::map`. However they are implemented as sorted vectors, not as a tree. This allows faster lookups and iterations, but inserting and removing elements is more expensive.

  These two containers are defined in the header files `boost/container/flat_set.hpp` and `boost/container/flat_map.hpp`.

- `boost::container::slist` is a singly linked list. It is similar to `std::forward_list`, which was added to the standard library with C++11. `boost::container::slist` provides a member function `size()`, which is missing in `std::forward_list`.

`boost::container::slist` is defined in `boost/container/slist.hpp`.

- `boost::container::static_vector` stores elements like `std::array` directly in the container. Like `std::array`, the container has a constant capacity, though the capacity doesn't say anything about the number of elements. The member functions `push_back()`, `pop_back()`, `insert()`, and `erase()` are available to insert or remove elements. In this regard, `boost::container::static_vector` is similar to `std::vector`. The member function `size()` returns the number of currently stored elements in the container.

  The capacity is constant, but can be changed with `resize()`. `push_back()` doesn't change the capacity. You may add an element with `push_back()` only if the capacity is greater than the number of currently stored elements. Otherwise, `push_back()` throws an exception of type `std::bad_alloc`.

  `boost::container::static_vector` is defined in `boost/container/static_vector.hpp`.