

# Chapter 2. Boost.PointerContainer

The library [Boost.PointerContainer](#) provides containers specialized to manage dynamically allocated objects. For example, with C++11 you can use `std::vector<std::unique_ptr<int>>` to create such a container. However, the containers from Boost.PointerContainer can provide some extra comfort.

Example 2.1. Using `boost::ptr_vector`

```
#include <boost/ptr_container/ptr_vector.hpp>
#include <iostream>

int main()
{
    boost::ptr_vector<int> v;
    v.push_back(new int{1});
    v.push_back(new int{2});
    std::cout << v.back() << '\n';
}
```

The class `boost::ptr_vector` basically works like `std::vector<std::unique_ptr<int>>` (see [Example 2.1](#)). However, because `boost::ptr_vector` knows that it stores dynamically allocated objects, member functions like `back()` return a reference to a dynamically allocated object and not a pointer. Thus, the example writes **2** to standard output.

Example 2.2. `boost::ptr_set` with intuitively correct order

```
#include <boost/ptr_container/ptr_set.hpp>
#include <boost/ptr_container/indirect_fun.hpp>
#include <set>
#include <memory>
#include <functional>
#include <iostream>

int main()
{
    boost::ptr_set<int> s;
    s.insert(new int{2});
    s.insert(new int{1});
    std::cout << *s.begin() << '\n';

    std::set<std::unique_ptr<int>, boost::indirect_fun<std::less<int>>> v;
    v.insert(std::unique_ptr<int>(new int{2}));
    v.insert(std::unique_ptr<int>(new int{1}));
    std::cout << **v.begin() << '\n';
}
```

[Example 2.2](#) illustrates another reason to use a specialized container. The example stores dynamically allocated variables of type `int` in a `boost::ptr_set` and a `std::set`. `std::set` is used together with `std::unique_ptr`.

With `boost::ptr_set`, the order of the elements depends on the `int` values. `std::set` compares pointers of type `std::unique_ptr` and not the variables the pointers refer to. To make `std::set` sort the elements based on `int` values, the container must be told how to compare elements. In [Example 2.2](#), `boost::indirect_fun` (provided by Boost.PointerContainer) is used. With `boost::indirect_fun`, `std::set` is told that elements

shouldn't be sorted based on pointers of type `std::unique_ptr`, but instead based on the `int` values the pointers refer to. That's why the example displays **1** twice.

Besides `boost::ptr_vector` and `boost::ptr_set`, there are other containers available for managing dynamically allocated objects. Examples of these additional containers include `boost::ptr_deque`, `boost::ptr_list`, `boost::ptr_map`, `boost::ptr_unordered_set`, and `boost::ptr_unordered_map`. These containers correspond to the well-known containers from the standard library.

#### Example 2.3. Inserters for containers from Boost.PointerContainer

```
#include <boost/ptr_container/ptr_vector.hpp>
#include <boost/ptr_container/ptr_inserter.hpp>
#include <array>
#include <algorithm>
#include <iostream>

int main()
{
    boost::ptr_vector<int> v;
    std::array<int, 3> a{{0, 1, 2}};
    std::copy(a.begin(), a.end(), boost::ptr_container::ptr_back_inserter(v));
    std::cout << v.size() << '\n';
}
```

Boost.PointerContainer provides inserters for its containers. They are defined in the namespace `boost::ptr_container`. To have access to the inserters, you must include the header file `boost/ptr_container/ptr_inserter.hpp`.

[Example 2.3](#) uses the function `boost::ptr_container::ptr_back_inserter()`, which creates an inserter of type `boost::ptr_container::ptr_back_insert_iterator`. This inserter is passed to `std::copy()` to copy all numbers from the array `a` to the vector `v`. Because `v` is a container of type `boost::ptr_vector`, which expects addresses of dynamically allocated `int` objects, the inserter creates copies with `new` on the heap and adds the addresses to the container.

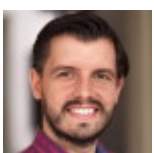
In addition to `boost::ptr_container::ptr_back_inserter()`, Boost.PointerContainer provides the functions `boost::ptr_container::ptr_front_inserter()` and `boost::ptr_container::ptr_inserter()` to create corresponding inserters.

## Exercise

Create a program with multiple objects of a type `animal` with the member variables `name`, `legs` and `has_tail`. Store the objects in a container from Boost.PointerContainer. Sort the container in ascending order based on `legs` and write all elements to standard output.

## Solutions

[theboostcplibraries.com](http://theboostcplibraries.com)



Solutions from  
the expert to all  
exercises in the  
book for \$9.99