Chapter 70. Boost.Assign

The library Boost.Assign provides helper functions to initialize containers or add elements to containers. These functions are especially helpful if many elements need to be stored in a container. Thanks to the functions offered by Boost.Assign, you don't need to call a member function like `push_back()` repeatedly to insert elements one by one into a container.

If you work with a development environment that supports C++11, you can benefit from initializer lists. Usually you can pass any number of values to the constructor to initialize containers. Thanks to initializer lists, you don't have to depend on Boost.Assign with C++11. However, Boost.Assign provides helper functions to add multiple values to an existing container. These helper functions can be useful in C++11 development environments.

Example 70.1 introduces a few functions that containers can be initialized with. To use the functions defined by Boost.Assign, include the header file `boost/assign.hpp`.

Example 70.1. Initializing containers

```
#include <boost/assign.hpp>
#include <boost/tuple/tuple.hpp>
#include <vector>
#include <stack>
#include <map>
#include <string>
#include <utility>

using namespace boost::assign;

int main()
{
  std::vector<int> v = list_of(1)(2)(3);

  std::stack<int> s = list_of(1)(2)(3).to_adapter();

  std::vector<std::pair<std::string, int>> v2 =
    list_of<std::pair<std::string, int>>("a", 1)("b", 2)("c", 3);

  std::map<std::string, int> m =
    map_list_of("a", 1)("b", 2)("c", 3);

  std::vector<boost::tuple<std::string, int, double>> v3 =
    tuple_list_of("a", 1, 9.9)("b", 2, 8.8)("c", 3, 7.7);
}
```

Boost.Assign provides three functions to initialize containers. The most important, and the one you usually work with, is `boost::assign::list_of()`. You use `boost::assign::map_list_of()` with `std::map` and `boost::assign::tuple_list_of()` to initialize tuples in a container.

You don't have to use `boost::assign::map_list_of()` or `boost::assign::tuple_list_of()`. You can initialize any container with `boost::assign::list_of()`. However, if you use `std::map` or a container with tuples, you must pass a template parameter to `boost::assign::list_of()` that tells the function how

elements are stored in the container. This template parameter is not required for `boost::assign::map_list_of()` and `boost::assign::tuple_list_of()`.

All three functions return a proxy object. This object overloads the operator `operator()`. You can call this operator multiple times to save values in the container. Even though you access another object, and not the container, the container is changed through this proxy object.

If you want to initialize adapters like `std::stack`, call the member function `to_adapter()` on the proxy. The proxy then calls the member function `push()`, which is provided by all adapters.

`boost::assign::tuple_list_of()` supports tuples of type `boost::tuple` only. You cannot use this function to initialize containers with tuples from the standard library.

Example 70.2 illustrates how values can be added to existing containers.

Example 70.2. Adding values to containers

```
#include <boost/assign.hpp>
#include <vector>
#include <deque>
#include <set>
#include <queue>

int main()
{
  std::vector<int> v;
  boost::assign::push_back(v)(1)(2)(3);

  std::deque<int> d;
  boost::assign::push_front(d)(1)(2)(3);

  std::set<int> s;
  boost::assign::insert(s)(1)(2)(3);

  std::queue<int> q;
  boost::assign::push(q)(1)(2)(3);
}
```

The `boost::assign::push_back()`, `boost::assign::push_front()`, `boost::assign::insert()`, and `boost::assign::push()` functions of Boost.Assign return a proxy. You pass these functions the container you want to add new elements to. Then, you call the operator `operator()` on the proxy and pass the values you want to store in the container.

The four functions `boost::assign::push_back()`, `boost::assign::push_front()`, `boost::assign::insert()`, and `boost::assign::push()` are called in this manner because the proxies returned call the identically named member functions on the container. Example 70.2 adds the three numbers 1, 2, and 3 to the vector **v** with three calls to `push_back()`.

Boost.Assign provides additional helper functions you can use to add values to a container, including `boost::assign::add_edge()`, which you can use to get a proxy for a graph from Boost.Graph.

Exercise

Improve this program with Boost.Assign:

```cpp
#include <vector>
#include <algorithm>
#include <iterator>
#include <iostream>

int main()
{
    std::vector<int> v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout));
    v.push_back(4);
    v.push_back(5);
    v.push_back(6);
    std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout));
}
```