Chapter 26. Boost.DynamicBitset

The library Boost.DynamicBitset provides the class `boost::dynamic_bitset`, which is used like `std::bitset`. The difference is that the number of bits for `std::bitset` must be specified at compile time, whereas the number of bits for `boost::dynamic_bitset` is specified at run time.

To use `boost::dynamic_bitset`, include the header file `boost/dynamic_bitset.hpp`.

Example 26.1. Using `boost::dynamic_bitset`

```
#include <boost/dynamic_bitset.hpp>
#include <iostream>

int main()
{
  boost::dynamic_bitset<> db{3, 4};

  db.push_back(true);

  std::cout.setf(std::ios::boolalpha);
  std::cout << db.size() << '\n';
  std::cout << db.count() << '\n';
  std::cout << db.any() << '\n';
  std::cout << db.none() << '\n';

  std::cout << db[0].flip() << '\n';
  std::cout << ~db[3] << '\n';
  std::cout << db << '\n';
}
```

`boost::dynamic_bitset` is a template that requires no template parameters when instantiated; default types are used in that case. More important are the parameters passed to the constructor. In Example 26.1, the constructor creates **db** with 3 bits. The second parameter initializes the bits; in this case, the number 4 initializes the most significant bit – the bit on the very left.

The number of bits inside an object of type `boost::dynamic_bitset` can be changed at any time. The member function `push_back()` adds another bit, which will become the most significant bit. Calling `push_back()` in Example 26.1 causes **db** to contain 4 bits, of which the two most significant bits are set. Therefore, **db** stores the number 12.

You can decrease the number of bits by calling the member function `resize()`. Depending on the parameter passed to `resize()`, bits will either be added or removed.

`boost::dynamic_bitset` provides member functions to query data and access individual bits. The member functions `size()` and `count()` return the number of bits and the number of bits currently set, respectively. `any()` returns `true` if at least one bit is set, and `none()` returns `true` if no bit is set.

To access individual bits, use array syntax. A reference to an internal class is returned that represents the corresponding bit and provides member functions to manipulate it. For example, the member function `flip()` toggles the bit. Bitwise operators such as `operator~` are available

as well. Overall, the class `boost::dynamic_bitset` offers the same bit manipulation functionality as `std::bitset`.

Like `std::bitset`, `boost::dynamic_bitset` does not support iterators.