VERIGY

C++ Tips
Article 4

Aether Lee
Hachioji, Japan

Abstract

In the previous article, we used "vector" and its iterators along with algorithms to introduce powerful features of C++. This time we will use another C++ container, "map", to look further into capabilities of C++.

1. Using "map"

In a first quick glance, the container "map" looks like an array.

Example 4-1. Deploying a map.

```
#include <iostream>
#include <map>

using namespace std;
int main()
{
    map<int, int> intMAPint;

    intMAPint[1] = 10;
    intMAPint[5] = 20;
    intMAPint[17]= 30;

    /*not a good way to access a map, but let's try it first*/
    for (int i=0; i<intMAPint.size(); i++)
        cout << intMAPint[i] << endl;

    return 0;
}
```

The declaration of "intMAPint" can be read as "to map an integer to another integer" or "to associate an integer value with an integer key". The next 3 lines can be interpreted as "associate 10 with the key 1, 20 with 5, and 30 with 17". So, 3 pairs are constructed here. Furthermore, as these lines imply, keys have to be unique in a map.

The for-loop in example 4-1 will output 18 lines on the screen, which list 15 "0"s in addition to those 3 explicitly assigned values. Obviously, it is something unexpected, since

only 3 pairs are created before the for-loop. The answer to this riddle is embedded in the operator "[]".

When a map calls "[]", it will first check whether the key exists. If the key is not in the map, it will associate a default object (here, an integer whose value is zero) with the key and return a reference to the object. This is exactly how those 3 pairs are created. On the other hand, when we loop over the map with the index i, and deploy "[]" to access the contents, for those indexes that are not in the map, "[]" will create pairs for them.

In this case, iterators are better choices.

Example 4-2. Accessing the contents with iterators.

```
#include <iostream>
#include <map>

using namespace std;
int main()
{
    map<int, int> intMAPint;

    intMAPint[1] = 10;
    intMAPint[5] = 20;
    intMAPint[17]= 30;

    for (map<int,int>::iterator it  = intMAPint.begin();
                                it != intMAPint.end(); ++it)
        cout << (*it).first << " : " << (*it).second << endl;

    return 0;
}
```

With iterators, only those created pairs will be printed on the screen. The way the iterators are used in the for-loop also implies that keys in maps are ordered. In the last example of this article, we will see that they are ordered in ascending order.

2. Using bitset to simulate a linear feedback shift register

Before continuing on the usage of a map, an example of using bitset to simulate a linear feedback shift register (LFSR) will be introduced here.

Example 4-3. Using bitset to simulate a linear feedback register

```
#include <iostream>
#include <bitset>

using namespace std;
int main()
{
    bitset<7> seed(string("1010101"));
    unsigned int tap1 = 7;
```

```
    unsigned int tap2 = 6;
    /* polynomial: x^7 + x^6 + 1 */
    do {
        bool bit0 = seed[tap1-1] xor seed[tap2-1] xor 1;
        seed <<= 1;
        seed[0] = bit0;
        cout << seed << endl;
    } while (seed != bitset<7>(string("1010101")));

        return 0;
}
```

This example is in fact a PRBS-7 generator. Each seed is correspondent to a code in the bit stream. If the generation is invoked several times, an algorithm “generate” can help to simplify the code appearance.

Example 4-3-1. Using generate to fulfill a vector

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

using namespace std;
class prbs{
public:
    prbs(const bitset<7>& start, unsigned int _tap1, unsigned int
_tap2, bool _xor_xnor) : next(start), tap1(_tap1), tap2(_tap2),
xor_xnor(_xor_xnor) {}
    bitset<7> operator()() {
        bool bit0 = next[tap1-1] xor next[tap2-1] xor xor_xnor;
        next <<= 1;
        next[0] = bit0;
        return next;
    }
private:
    bitset<7> next;
    unsigned int tap1;
    unsigned int tap2;
    bool xor_xnor;
};

int main()
{
    vector < bitset<7> > vCode;
    vCode.resize(127);
    bitset<7> seed(string("1010101"));
    generate(vCode.begin(), vCode.end(), prbs(seed,7,6,1));
```

```
    return 0;
}
```

The algorithm “generate” calls the class “prbs” to generate objects to fulfill the vector. When the class “prbs” is called for the first time, the private members of the class are initialized with the values assigned. The generation in the do-while loop in example 4-3 is composed in the operator(). So, when “generate” iterates, prbs() is repeatedly called and then the next code is returned to the vector.

3. Associating objects with string type keys in a map

Consider now several polynomials are used for LFSRs; the generated bit streams have to be saved along with taking polynomials as indexes.

Example 4-4. Associating objects with string type keys in a map.

```
#include <iostream>
#include <vector>
#include <map>
#include <bitset>
#include <algorithm>
#include <iterator>

using namespace std;
class prbs{
public:
    prbs(const bitset<7>& start, unsigned int _tap1,
         unsigned int _tap2, bool _xor_xnor) :
         next(start), tap1(_tap1), tap2(_tap2),
         xor_xnor(_xor_xnor) {}
    bitset<7> operator()() {
        bool bit0 = next[tap1-1] xor next[tap2-1] xor xor_xnor;
        next <<= 1;
        next[0] = bit0;
        return next;
    }
private:
    bitset<7> next;
    unsigned int tap1;
    unsigned int tap2;
    bool xor_xnor;
};

void cout_find(const pair<string, vector< bitset<7> > >& lfsr)
{
    cout << "Polynomial: " << lfsr.first << endl;
    copy(lfsr.second.begin(), lfsr.second.end(),
         ostream_iterator< bitset<7> >(cout,"\t"));
```

```
    cout << endl;

    size_t n = find(lfsr.second.begin(), lfsr.second.end(),
               bitset<7>(string("1111111")))
             - lfsr.second.begin();
    if (n < lfsr.second.size())
         cout << "1111111 is at " << n << "th code." << endl;
    else cout << "There is no 1111111." << endl;
}

int main()
{
    map<string, vector< bitset<7> > > mapLFSR;
    vector < bitset<7> > vCode;
    vCode.resize(127);
    bitset<7> seed(string("1010101"));

    generate(vCode.begin(), vCode.end(), prbs(seed,7,6,1));
    mapLFSR["X^7+X^6+1_XNOR"] = vCode;

    generate(vCode.begin(), vCode.end(), prbs(seed,7,6,0));
    mapLFSR["X^7+X^6+1_XOR"] = vCode;

    generate(vCode.begin(), vCode.end(), prbs(seed,7,4,1));
    mapLFSR["X^7+X^4+1_XNOR"] = vCode;

    generate(vCode.begin(), vCode.end(), prbs(seed,7,4,0));
    mapLFSR["X^7+X^4+1_XOR"] = vCode;

    for_each(mapLFSR.begin(), mapLFSR.end(), cout_find);
    return 0;
}
```

String type keys are used as indexes in the map “mapLFSR” and each key associates a vector, which saves the whole bit stream generated by a polynomial. Each element of the vector is a bitset object. As demonstrated above, a map provides a simple and convenient way to index objects other than consecutive integers. Maps are hence often called dictionaries. Also, to use maps is another way to build multi-dimensional arrays.

Last but not least, as introduced in the previous article, for_each is called to loop over all pairs in the map, and fetch them to the sub function “cout_find”. In the function, copy is used to print out the contents and find is used to find the occurrence of a specific code.

4. References

Ray Lischner, *C++ in a nutshell*, O’Reilly, USA, 2003.

5. Copyright