VERIGY

C++ Tips
Article 2

Aether Lee
Hachioji, Japan

Abstract

*In this second article of the series, a c++ class template **bitset** will be introduced. It can be used to represent an integer number in binary format and to interpret a string as a binary number. Within the following examples, we will take a peek at the most powerful feature which is also the core of the standard library of C++: templates.*

1. Displaying an integer number in binary format

Though all the numbers are stored and processed in binary, to display it in binary format is not straightforward in C++. A typical solution ever since the era of C is to deploy a bit-mask with iterating loops and to print simply "1" or "0" according to the bitwise AND operation result.

Example 2-1. Using a bit-mask to display an integer number in binary format with MSB at the leftmost position.

```
#include <iostream>
using namespace std;
int main()
{
    unsigned int number = 0xA5A5; //"1010010110100101" in binary;

    unsigned int bitmask = 0x80000000; //a solitary "1" at bit 31
    for(int i=0;i<32;i++){
        cout << (bool)(number bitand (bitmask >> i));
    }
    cout << endl;

    return 0;
}
```

It should display "00000000000000001010010110100101" as the result. It works, however, adding several lines containing a loop only for displaying would not be considered as a piece of art.

With C++, there is something in the standard library which can be used to resolve the task within one line.

*Example 2-2. Using **bitset** to represent an integer number in binary format.*

```
#include <iostream>
#include <bitset>
using namespace std;
int main()
{
    unsigned int number = 0xA5A5; //”1010010110100101” in binary;

    cout << bitset<32>(number) << endl;

    return 0;
}
```

Here, a class template **bitset** is introduced. In the line, it is used to implicitly form a sequence of bits based on the value of “number”, and **cout** simply prints every bit of it.

The angle brackets after **bitset** may look unfamiliar to C users. It is part of the template declaration, and the content can be specified when the template is used. In the case of **bitset**, it is used to specify the size of the sequence of bits. In example 2-2, a sequence of 32 bits is formed then.

There are also several operators and functions in the template of **bitset** which make it very handy when manipulating the bits. For example, the operator [] can be used to access a specific bit by an index, e.g. bitseq[7] accesses the bit 7 in “bits”. Another example is given in the following section.

2. Interpreting a string as a binary number

In the following example, a solution to interpret a string as a binary number with **bitset** is demonstrated.

*Example 2-3. Using **bitset** to interpret a string as a binary number and to convert it to an integer.*

```
#include <iostream>
#include <string>
#include <bitset>
using namespace std;
int main()
{
    string control = “0110101”;
    bitset<7> bitsctrl(control);
    unsigned long numctrl  = bitsctrl.to_ulong();
    unsigned long flipctrl = bitsctrl.flip().to_ulong();

    return 0;
}
```

In the example above, a **bitset** class “bitsctrl” is constructed according to the string “control”, and then the member function **to_ulong** returns the bit sequence as an **unsigned long** integral number to “numctrl”. Furthermore, the next line invokes **flip** to toggle all the bits in bitsctrl before returning the integral number.

In the example above, “7” is assigned to the **bitset** template. If we print those two integral numbers, 53 and 74 shall be displayed, and they are “0110101” and “1001010” in binary. However, if “11” is assigned, “0” will be assigned to the highest 4 bits in bitsctrl. If those two integral numbers are printed in a fashion of example 2-2, we shall see “00000110101” and “11111001010” as results.

3. Cutting several bits from a integer and making it a string

Consider we have a register setting stored in an integer and want to take several bits from it to form an identifier in the name of a file.

Example 2-4. Using **bitset** *to cut a bit sequence.*

```
#include <iostream>
#include <string>
#include <bitset>
using namespace std;
int main()
{
    unsigned int reg = 0xB4; //”10110100”;
    string fileId =
        bitset<4>(bitset<8>(reg).to_string(),1,4).to_string();
    cout << fileId << endl;

    return 0;
}
```

The result should be displayed as “0110”. Here another member function is used: **to_string**. It returns the content of a **bitset** class in string type.

In the construction of “fileId”, the “**bitset<4>**” cuts from the character 1 for 4 characters that is bit 3 to 6 of “reg” (the character 0 is the leftmost character in a string while the bit 0 is the rightmost bit in a binary number).

Before the end of the article, there is a notice about **to_string**. Depending the compiler, it may require specifying explicitly the template parameters of it. It should be then replaced as in the following line for the example above:

```
to_string< char, char_traits<char>, allocator<char> >();
```

4. References

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

5. Copyright