**Bitsets**

Bitsets model fixed sized arrays of bits or Boolean values. They are useful to manage sets of flags, where variables may represent any combination of flags.

The class bitset has the advantage that bitsets may contain any number of bits, and additional operations are provided.

You can’t change the number of bits in a bitset. The number of bits is the template parameter.

If you need a container for a variable number of bits or Boolean values, you can use the class vector<bool>.

header file

#include <bitset>

The class bitset is defined as a class template, with the number of bits as the template parameter:

namespace std {

template <size\_t Bits>

class bitset;

}

In this case, the template parameter is not a type but an unsigned integral value.

Templates with different template arguments are different types.

You can compare and combine bitsets only with the same number of bits.

# Important Changes with C++11

1. Bitsets now can be initialized by string literals
2. Conversions to and from numeric values now support type unsigned long long. For this, to\_ullong() was introduced
3. Conversions to and from strings now allow you to specify the character interpreted as set and unset bit.
4. Member all() is now provided to check whether all bits are set.
5. To use bitsets in unordered containers, a default hash function is provided

The number of bits in the bitset should be smaller than sizeof(unsigned long long).

The reason is that you get an exception when the value of the bitset can’t be represented as unsigned long long.

# Example

## Using Bitsets as Sets of Flags

Demonstrates how to use bitsets to manage a set of flags.

Each flag has a value that is defined by an enumeration type. The value of the enumeration type is used as the position of the bit in the bitset.

#include <bitset>

#include <iostream>

using namespace std;

int main() {

// enumeration type for the bits each bit represents a color

enum Color { white, black, red, yellow, green, numColors };

// create bitset for all bits/colors

bitset<numColors> usedColors;

// set bits for two colors

usedColors.set(red);

usedColors.set(white);

// print some bitset data

cout << "bitfield of used colors: " << usedColors << endl;

cout << "number of used colors: " << usedColors.count() << endl;

cout << "bitfield of unused colors: " << ~usedColors << endl;

// if any color is used

if (usedColors.any()) {

// loop over all colors

for (int c = 0; c < numColors; ++c) {

// if the actual color is used

if (usedColors[(Color)c]) {

} // end of if

} // end of for

} // end of if

return 0;

}

Output:

bitfield of used colors: 00101

number of used colors: 2

bitfield of unused colors: 11010

## Using Bitsets for I/O with Binary Representation

A useful feature of bitsets is the ability to convert integral values into a sequence of bits, and vice versa. This is done simply by creating a temporary bitset:

#include <bitset>

#include <iostream>

#include <string>

#include <limits>

using namespace std;

int main() {

// print some numbers in binary representation

cout << "267 as binary short: "

<< bitset<numeric\_limits<unsigned short>::digits>(267) << endl;

cout << "267 as binary long: "

<< bitset<numeric\_limits<unsigned long>::digits>(267) << endl;

cout << "10,000,000 with 24 bits: " << bitset<24>(1e7) << endl;

// write binary representation into string

/\* Before C++11

string s =

bitset<42>(12345678).to\_string<char, char\_traits<char>, allocator<char> >();

\*/

string s = bitset<42>(12345678).to\_string();

cout << "12,345,678 with 42 bits: " << s << endl;

// transform binary representation into integral number

cout << "\"1000101011\" as number: "

<< bitset<100>("1000101011").to\_ullong() << endl;

/\*

Before C++11 bitset<100>(string("1000101011"))

\*/

return 0;

}

Output:

267 as binary short: 0000000100001011

267 as binary long: 0000000000000000000000000000000000000000000000000000000100001011

10,000,000 with 24 bits: 100110001001011010000000

12,345,678 with 42 bits: 000000000000000000101111000110000101001110

"1000101011" as number: 555

# END