c++ bitset

bitset是比特集合，用于位运算等操作。固定长度，支持随机访问。其使用方法和其他模板类差别不大。

```
bitset<n> b         b有n位，被默认设置位0，n必须为常量表达式

bitset<n> b(u)     b是unsigned long long的低n位比特串拷贝。超出u的位数，剩余的被设置为0

//有时候可能将字符串与比特串之间互相转换，可用到下面的构造函数

bitset<n> b(s, pos, m, zero, one)

//b是string s 从pos位开始m个字符的拷贝s只能包含zero/one， 否则会抛出一个
invalid_arguement异常。
字符在b中分别保存位zero one. pos默认值为0， m默认为std::string::npos，zero默认为'0', one
默认为'1'

bitset<n> b(cp, pos, m, zero, one)

//同上面的构造函数相同，但是从cp指向的字符数组中拷贝字符。如果没有提供m，则cp必须指向一个字符
串。如果提供了m，则从cp开始必须至少有m个zero/one字符

//注意：上述两个构造函数，即接受string或者字符指针的构造函数是explicit的。在新标准中增加了为0
和1指定其他字符的功能
```

1.演示

```
#include <iostream>
#include <bitset>
using namespace std;

int main()
{
    const unsigned bit_num = 13;
    bitset<13> b1 (0xbeef);
    for(int i = bit_num -1;i>=0;--i)
        cout << b1[i];
    cout << endl;
    bitset<20> b2 (0xbeff);

    for(int i= 20-1;i>=0;--i)
        cout << b2[i];
    return 0;
}
```

1.运行结果

```
1111011101111
00001011111011111111
```

2.演示

```
#include <iostream>
#include <bitset>
using namespace std;

int main()
{
    string bit_string = "1001001100";
    bitset<10> b1 (bit_string); // b1 is 1001001100
    bitset<4> b2(bit_string,5,4,'0','1');
    for(int i = 4-1;i>=0;--i){
        cout << b2[i];
    }
    cout << endl;
    bitset<4> b3(bit_string,bit_string.length()-4);
    for(int i=4-1;i>=0;--i){
        cout << b3[i];
    }
    return 0;
}
```

2.运行结果

```
0110
1100
```

可以看出，类似头迭代器和超尾迭代器的规则在bitset的构造时候依然适用

先介绍一些概念

置位：将某一位设置为1

复位：将某一位设置为0

bitset方法

```
//关于bitset的状态

.any()          是存在置位的二进制返回true

.none()         不存在置位的二进制返回true

.all()            所有位都置位返回true

.count()        返回置位的位数

.size()         一个constexpr函数，返回位数
```

```
.test(pos)   pos位为真返回true


//关于设置bitset状态的函数

.set(pos,v)         设置pos位为v，v默认值为真值

.set()                  无实参的情况下将所有位置位

.reset(pos)         复位pos位

.reset()                无实参的情况下复位所有位

.flip(pos)              切换pos位

.filp()                     无实参的情况下切换所有位

b[pos]                  下标访问


//其它功能

返回一个unsigned long / unsigned long long值，如果b中位模式不能放入指定的结果类型，抛出一个overflow_error异常

.to_ulong()

.to_ullong()

返回一个true为one flase为zero的字符串默认为'1', '0'

.to_string(zero, one)

os << b         将b中的二进制位打印到流os

is >> b          从is流中输入二进制位
```

3.演示

用一个bitset来存放10人的成绩及格状况

```
#include <iostream>
#include <bitset>
#define grade_table GT
using namespace std;

int main()
{
    const unsigned stu_num = 10;
    bitset<stu_num> grade_table;
    cin >> grade_table;
    if(GT.none()) //or !GT.any()
```

```
        cout << "没有人及格" << endl;
    else
        cout << "一共有" << GT.count() << "人及格" << endl;
    string bit_string = GT.to_string();
    cout << ".to_string()返回了一个比特字符串: " << bit_string << endl;
    cout << "老师大发慈悲，所有人都过了，好耶！" << endl;
    GT.set();
    cout << GT.to_string() << endl;
    GT.reset(1);
    cout << "但是一号小伙得罪了老师，老师没有给他过: " << GT.to_string() << endl;
    cout << "不知道将这个比特串转换为无符号整形有什么意义的操作" << GT.to_ulong();
    return 0;
}
```

3.运行结果

```
//输入
1001001100
//输出
一共有4人及格
.to_string()返回了一个比特字符串: 1001001100
老师大发慈悲，所有人都过了，好耶！
1111111111
但是一号小伙得罪了老师，老师没有给他过: 1111111101
不知道将这个比特串转换为无符号整形有什么意义的操作1021
```

可以看见，适用bitset比适用传统的位运算要方便许多，不用我们自己设计mask，不用进行令人头疼的位移操作，甚至输入输出比特串也经为我们定义好了。

上面可以看出bitset为了泛型的功能，有着zero和one这两个东西，提供让用户自定义的功能和一些有趣的东西。

4.演示

```
#include <iostream>
#include <bitset>
using namespace std;

int main() {
    const string str = "ynynyynnyn";
    bitset<10> b(str, 0, string::npos, 'n', 'y');
    cout << b << endl;
    cout << b.to_string('n', 'y');
    return 0;
}
```

4.运行结果

```
1010110010
ynynyynnyn
```