

C++——bitset

顾名思义，**bitset** 就是比特集合，用于位运算等操作。

固定长度，支持随机访问

同替他 **模板类** 一样，**bitset**的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，如果超出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_argument**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C风格的字符串。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

演示

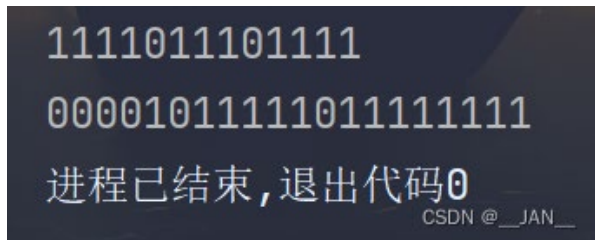
```
1  #include <iostream>
2  #include <bitset>
3  #include <algorithm>
4  using namespace std;
5
6  int main()
7  {
8      const unsigned bit_num = 13;
9      bitset<13> b1 (0xbeef);
```

```

10     for(int i = bit_num -1;i>=0;--i)
11         cout << b1[i];
12     cout << endl;
13     bitset<20> b2 (0xbfff);
14
15     for(int i= 20-1;i>=0;--i)
16         cout << b2[i];
17     return 0;
18 }

```

运行结果



```

1111011101111
00001011111011111111
进程已结束,退出代码0
CSDN @__JAN__

```

演示

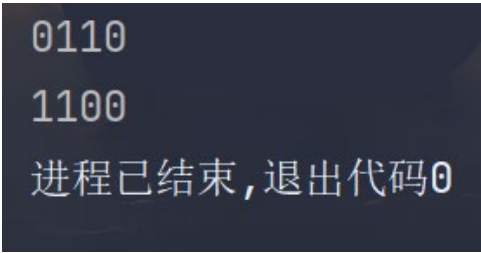
```

1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4
5  int main()
6  {
7      string bit_string = "1001001100";
8      //const unsigned bit_length = bit_string.length(); //error bit_length不能
9      bitset<10> b1 (bit_string); // b1 is 1001001100
10     bitset<4> b2(bit_string,5,4,'0','1');
11     for(int i = 4-1;i>=0;--i)
12     {
13         cout << b2[i];
14     }
15     cout << endl;
16     bitset<4> b3(bit_string,bit_string.length()-4);

```

```
17     for(int i=4-1;i>=0;--i)
18     {
19         cout << b3[i];
20     }
21     return 0;
22 }
```

运行结果



```
0110
1100
进程已结束,退出代码0
```

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)` pos

复位 位

.reset() 无实参的情况下复位所有位

.flip(pos) 切换pos位

.flip() 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 false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

演示

用一个bitset来存放10人的成绩及格状况

```
1  #include <iostream>
2  #include <bitset>
3  #include <algorithm>
4  #define grade_table GT
5  using namespace std;
6
7  int main()
8  {
9      const unsigned stu_num = 10;
10     bitset<stu_num> grade_table;
11     cin >> grade_table;
12     if(GT.none()) //or !GT.any()
13         cout << "没有人及格" << endl;
14     else
15         cout << "一共有" << GT.count() << "人及格" << endl;
16     string bit_string = GT.to_string();
17     cout << ".to_string()返回了一个比特字符串: " << bit_string << endl;
```

```
18     cout << "老师大发慈悲，所有人都过了，好耶！" << endl;
19     GT.set();
20     cout << GT.to_string() << endl;
21     GT.reset(1);
22
23     cout << "但是一号小伙得罪了老师，老师没有给他过：" << GT.to_string() << endl;
24     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();
25     return 0; }
```

运行结果

1001001100

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

进程已结束,退出代码0

CSDN @__JAN__

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。

一些其他的東西

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

一些有趣的东西

```
1 #include <iostream>
2 #include <bitset>
3 #include <algorithm>
```

```
4 using namespace std;
5
6 int main() {
7     const string str = "ynynyynnyn";
8     bitset<10> b(str, 0, string::npos, 'n', 'y');
9     cout << b << endl;
10    cout << b.to_string('n', 'y');
11    return 0;
12 }
```

可以猜猜运行的结果时什么

1010110010

ynynyynnyn

进程已结束,退出代码0

CSDN @__JAN__