**用于hash表查询**

cout<<hex<<i<<endl;十六

cout<<oct<<i<<endl;八

二进制的赋值是不行的……

**C/C++里面只能支持八、十、十六进制三种**

数字型常量。

如果真的需要的话，那么只能靠std::bitset来完成，通过调用参数为std::string的构造函数就可以把二进制变成bitset，然后用bitset::to\_ulong就可以了……

有些程序要处理二进制位的有序集，每个位可能包含的是0（关）或1（开）的值。位是用来保存一组项或条件的yes/no信息（有时也称标志）的简洁方法。标准库提供了bitset类使得处理位集合更容易一些。要使用bitset类就必须要包含相关的头文件。在本书提供的例子中，假设都使用了std::bitset的using声明：

#include <bitset>

3.5.1 bitset的定义和初始化

bitset<32> bitvec; //32位，全为0。

这条语句把bitvec定义为含有32个位的bitset对象。和vector的元素一样，bitset中的位是没有命名的，程序员只能按位置来访问它们。位集合的位置编号从0开始，因此，bitvec的位序是从0到31。以0位开始的位串是低阶位（low-order bit），以31位结束的位串是高阶位(high-order bit)。(右往左是低到高)

bitset<n> b;

b有n位，每位都为0

bitset<n> b(u);

b是unsigned long型u的一个副本

bitset<n> b(s);

b是string对象s中含有的位串的副本

bitset<n> b(s, pos, n);

b是s中从位置pos开始的n个位的副本

// bitvec1is smaller than the initializer

bitset<16> bitvec1(0xffff); // bits 0 ... 15 are set to 1

// bitvec2same size as initializer

bitset<32> bitvec2(0xffff); [00000000000000001111111111111111]

// on a 32-bit machine, bits 0 to 31 initialized from0xffff

bitset<128> bitvec3(0xffff); // bits 32 through 127 initialized to zero

上面的三个例子中，0到15位都置为1。由于bitvec1位数少于unsigned long的位数，因此bitvec1的初始值的高阶位被丢弃。bitvec2和unsigned long长度相同，因此所有位正好放置了初始值。bitvec3长度大于32，31位以上的高阶位就被置为0。

2. 用string对象初始化bitset对象

当用string对象初始化bitset对象时，string对象直接表示为位模式。从string对象读入位集的顺序是从右向左：

string strval("1100");

bitset<32> bitvec4(strval)

bitvec4的位模式中第2和3的位置为1，其余位置都为0。如果string对象的字符个数小于bitset类型的长度，则高阶位将置为0。

string对象和bitset对象之间是反向转化的：string对象的最右边字符（即下标最大的那个字符）用来初始化bitset对象的低阶位（即下标为0的位）。当用string对象初始化bitset对象时，记住这一差别很重要。

不一定要把整个string对象都作为bitset对象的初始值。相反，可以只用某个子串作为初始值：

string str("1111111000000011001101");

bitset<32> bitvec5(str, 5, 4); // 4 bits starting atstr[5], 1100

bitset<32> bitvec6(str, str.size() - 4); // use last 4 characters

这里用str中从str[5]开始包含四个字符的子串来初始化bitvec5。照常，初始化bitset对象时总是从子串最右边结尾字符开始的，bitvec5的从0到3的二进制位置为1100，其他二进制位都置为0。如果省略第三个参数则意味着取从开始位置一直到string末尾的所有字符。本例中，取出str末尾的四位来对bitvec6的低四位进行初始化。bitvec6其余的位初始化为0。这些初始化过程的图示如下：

3.5.2 bitset对象上的操作

b.any()

b中是否存在置为1的二进制位？

b.none()

b中不存在置为1的二进制位吗？

b.count()

b中置为1的二进制位的个数

b.size()

b中二进制位的个数

b[pos]

访问b中在pos处的二进制位

b.test(pos)

b中在pos处的二进制位是否为1？

b.set()

把b中所有二进制位都置为1

b.set(pos)

把b中在pos处的二进制位置为1

b.reset()

把b中所有二进制位都置为0

b.reset(pos)

把b中在pos处的二进制位置为0

b.flip()

把b中所有二进制位逐位取反

b.flip(pos)

把b中在pos处的二进制位取反

b.to\_ulong()

用b中同样的二进制位返回一个unsigned long值

os << b

把b中的位集输出到os流

1. 测试整个bitset对象

如果bitset对象中有一个或多个二进制位置为1，则any操作返回true，也就是说，其返回值等于1;相反，如果bitset对象中的二进制位全为0,则none操作返回true。

bitset<32> bitvec; // 32 bits, all zero

bool is\_set = bitvec.any(); // false, all bits are zero

bool is\_not\_set = bitvec.none(); // true, all bits are zero

如果需要知道置为1的二进制位的个数，可以使用count操作，该操作返回置为1的二进制位的个数：

size\_t bits\_set = bitvec.count(); // returns number of bits that are on

count操作的返回类型是标准库中命名为size\_t的类型。size\_t类型定义在cstddef头文件中，该文件是C标准库的头文件stddef.h的C++版本。它是一个与机器相关的unsigned类型，大小可以保证存储内存中对象。

与vector和string中的size操作一样，bitset的size操作返回bitset对象中二进制位的个数，返回值的类型是size\_t:

size\_t sz = bitvec.size(); // returns 32

2. 访问bitset对象中的位

可以用下标操作符来读或写某个索引位置的二进制位，同样地，也可以用下标操作符测试给定二进制位的值或设置某个二进制位的值：

// assign 1 to even numbered bits

for (int index = 0; index != 32; index += 2)

bitvec[index] = 1;

上面的循环把bitvec中的偶数下标的位都置为1。

除了用下标操作符，还可以用set、test和reset操作来测试或设置给定二进制位的值：

// equivalent loop using set operation

for (int index = 0; index != 32; index += 2)

bitvec.set(index);

为了测试某个二进制位是否为1，可以用test操作或者测试下标操作符的返回值：

if (bitvec.test(i))

// bitvec[i] is on

// equivalent test using subscript

if (bitvec[i])

// bitvec[i] is on

如果下标操作符测试的二进制位为1，则返回的测试值的结果为true，否则返回false。

3. 对整个bitset对象进行设置

set和reset操作分别用来对整个bitset对象的所有二进制位全置1和全置0：

bitvec.reset(); // set all the bits to 0.

bitvec.set(); // set all the bits to 1

flip操作可以对bitset对象的所有位或个别位按位取反：

bitvec.flip(0); // reverses value of first bit

bitvec[0].flip(); // also reverses the first bit

bitvec.flip(); // reverses value of all bits

4. 获取bitset对象的值

to\_ulong操作返回一个unsignedlong值，该值与bitset对象的位模式存储值相同。仅当bitset类型的长度小于或等于unsignedlong的长度时，才可以使用to\_ulong操作：

unsigned long ulong = bitvec3.to\_ulong();

cout << "ulong = " << ulong << endl;

to\_ulong操作主要用于把bitset对象转到C风格或标准C++之前风格的程序上。如果bitset对象包含的二进制位数超过unsigned long的长度，将会产生运行时异常。本书将在6.13节介绍异常（exception），并在17.1节中详细地讨论它。

5. 输出二进制位

可以用输出操作符输出bitset对象中的位模式：

bitset<32> bitvec2(0xffff); // bits 0 ... 15 are set to 1; 16 ... 31 are 0

cout << "bitvec2: " << bitvec2 << endl;

输出结果为：

bitvec2: 00000000000000001111111111111111

6. 使用位操作符

bitset类也支持内置的位操作符。C++定义的这些操作符都只适用于整型操作数，它们所提供的操作类似于本节所介绍的bitset操作。5.3节将介绍这些操作符。