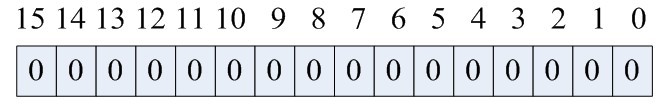
**位图\_BitMap**

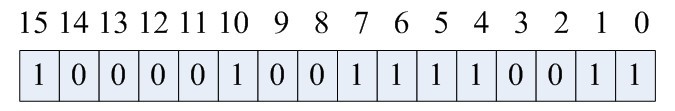
位图法就是bitmap的缩写。所谓bitmap，就是用每一位来存放某种状态，适用于大规模数据，但数据状态又不是很多的情况。通常是用来判断某个数据存不存在的。在STL中有一个bitset容器，其实就是位图法.

unsigned int bit[N];

在这个数组里面，可以存储 N \* sizeof(int) \* 8个数据，但是最大的数只能是N \* sizeof(int)  \* 8 - 1。假如，我们要存储的数据范围为0-15，则我们只需要使得N=1，这样就可以把数据存进去。如下图：



数据为【5，1，7，15，0，4，6，10】，则存入这个结构中的情况为



# 三、相关操作

## 1，写入数据

定义一个数组： unsigned char bit[8 \* 1024];这样做，能存 8K\*8=64K 个 unsigned short 数据。bit 存放的字节位置和位位置（字节 0~8191 ，位 0~7 ）

比如写 1234 ，字节序： 1234/8 = 154; 位序： 1234 &0b111 = 2 ，那么 1234 放在 bit 的下标 154 字节处，把该字节的 2 号位（ 0~7）置为 1

字节位置： int nBytePos =1234/8 = 154;

位位置：   int nBitPos = 1234 & 7 = 2;

**[cpp]** [view plain](http://blog.csdn.net/wypblog/article/details/8237956" \o "view plain) [copy](http://blog.csdn.net/wypblog/article/details/8237956" \o "copy) [print](http://blog.csdn.net/wypblog/article/details/8237956" \o "print)[?](http://blog.csdn.net/wypblog/article/details/8237956" \o "?)[IMG_256](https://code.csdn.net/snippets/141053" \o "在CODE上查看代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)[IMG_257](https://code.csdn.net/snippets/141053/fork" \o "派生到我的代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)

1. // 把数组的 154 字节的 2 位置为 1
2. unsigned **short** val = 1<<nBitPos;
3. bit[nBytePos] = bit[nBytePos] |val;  // 写入 1234 得到arrBit[154]=0b00000100

  再比如写入 1236 ，

字节位置： int nBytePos =1236/8 = 154;

位位置：   int nBitPos = 1236 & 7 = 4

**[cpp]** [view plain](http://blog.csdn.net/wypblog/article/details/8237956" \o "view plain) [copy](http://blog.csdn.net/wypblog/article/details/8237956" \o "copy) [print](http://blog.csdn.net/wypblog/article/details/8237956" \o "print)[?](http://blog.csdn.net/wypblog/article/details/8237956" \o "?)[IMG_258](https://code.csdn.net/snippets/141053" \o "在CODE上查看代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)[IMG_259](https://code.csdn.net/snippets/141053/fork" \o "派生到我的代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)

1. // / 把数组的 154 字节的 4 位置为 1
2. val = 1<<nBitPos;
3. arrBit[nBytePos] = arrBit[nBytePos] |val;  // 再写入 1236 得到arrBit[154]=0b00010100

函数实现：

**[cpp]** [view plain](http://blog.csdn.net/wypblog/article/details/8237956" \o "view plain) [copy](http://blog.csdn.net/wypblog/article/details/8237956" \o "copy) [print](http://blog.csdn.net/wypblog/article/details/8237956" \o "print)[?](http://blog.csdn.net/wypblog/article/details/8237956" \o "?)[IMG_260](https://code.csdn.net/snippets/141053" \o "在CODE上查看代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)[IMG_261](https://code.csdn.net/snippets/141053/fork" \o "派生到我的代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)

1. #define SHIFT 5
2. #define MAXLINE 32
3. #define MASK 0x1F
4. **void** setbit(**int** \*bitmap, **int** i){
5. bitmap[i >> SHIFT] |= (1 << (i & MASK));
6. }

## 2，读指定位

**[cpp]** [view plain](http://blog.csdn.net/wypblog/article/details/8237956" \o "view plain) [copy](http://blog.csdn.net/wypblog/article/details/8237956" \o "copy) [print](http://blog.csdn.net/wypblog/article/details/8237956" \o "print)[?](http://blog.csdn.net/wypblog/article/details/8237956" \o "?)[IMG_262](https://code.csdn.net/snippets/141053" \o "在CODE上查看代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)[IMG_263](https://code.csdn.net/snippets/141053/fork" \o "派生到我的代码片" \t "http://blog.csdn.net/wypblog/article/details/_blank)

1. **bool** getbit(**int** \*bitmap1, **int** i){
2. **return** bitmap1[i >> SHIFT] & (1 << (i & MASK));
3. }

# 四、位图法的缺点

1. 可读性差
2. 位图存储的元素个数虽然比一般做法多，但是存储的元素大小受限于存储空间的大小。位图存储性质：存储的元素个数等于元素的最大值。比如， 1K 字节内存，能存储 8K 个值大小上限为 8K 的元素。（元素值上限为 8K ，这个局限性很大！）比如，要存储值为 65535 的数，就必须要 65535/8=8K 字节的内存。要就导致了位图法根本不适合存 unsigned int 类型的数（大约需要 2^32/8=5 亿字节的内存）。
3. 位图对有符号类型数据的存储，需要 2 位来表示一个有符号元素。这会让位图能存储的元素个数，元素值大小上限减半。 比如 8K 字节内存空间存储 short 类型数据只能存 8K\*4=32K 个，元素值大小范围为 -32K~32K 。

# 五、位图法的应用

  1、给40亿个不重复的unsigned int的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那40亿个数当中  
　　首先，将这40亿个数字存储到bitmap中，然后对于给出的数，判断是否在bitmap中即可。  
2、使用位图法判断整形数组是否存在重复  
      遍历数组，一个一个放入bitmap，并且检查其是否在bitmap中出现过，如果没出现放入，否则即为重复的元素。  
       3、使用位图法进行整形数组排序  
      首先遍历数组，得到数组的最大最小值，然后根据这个最大最小值来缩小bitmap的范围。这里需要注意对于int的负数，都要转化为unsigned int来处理，而且取位的时候，数字要减去最小值。位图也可以初始全为0，扫描完一遍之后将所有的数对应到位图之后然后顺序输出位图中为1对应的书就是排好序的  
       4、在2.5亿个整数中找出不重复的整数，注，内存不足以容纳这2.5亿个整数  
      参 考的一个方法是：采用2-Bitmap（每个数分配2bit，00表示不存在，01表示出现一次，10表示多次，11无意义）。共需内存\*2=1GB。，然后扫描这2.5亿个整数，，查看位图中相对应的位，如果是00就变成01，如果是01就变成10，如果是10就不变，扫描完之后，查看位图，把对应为01的整数输出来即可