lower_bound()函数和upper_bound()函数

参考C++ Refference：

https://www.cplusplus.com/reference/algorithm/lower_bound/

https://www.cplusplus.com/reference/algorithm/upper_bound/

lower_bound()

```
template <class ForwardIterator, class T>
 ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last,
                              const T& val);
```

自定义比较函数版

```
template <class ForwardIterator, class T, class Compare>
ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last,
                              const T& val, Compare comp);
```

第一个first参数是一段连续空间的首地址，last是连续空间末端的地址，val是要查找的值。调用lower_bound()的前提是这段连续的空间里的元素是有序（递增）的。
然后lower_bound()的返回值是第一个大于等于val的值的地址，用这个地址减去first，得到的就是第一个大于等于val的值的下标

在自定义版本里有一个comp参数，它的用处在于，当你要查找的不是基本数据类型，而是自己定义的struct时就需要自己定义一下，怎么比较两个struct的大小。

用法示例

默认版本

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
int main(int argc, char *argv[]){
    int a[] = {1, 3, 5, 7, 9};
    cout << (lower_bound(a, a + 5, 1) - a) << endl;
    // 第一个大于等于1的元素是1，下标是0
    cout << (lower_bound(a, a + 5, 2) - a) << endl;
    // 第一个大于等于2的元素是3，下标是1
    cout << (lower_bound(a, a + 5, 8) - a) << endl;
    // 第一个大于等于8的元素是9，下标是4
    cout << (lower_bound(a, a + 5, 100) - a) << endl;
    // 最大的元素也不比100大，故返回值是last，再减a也就是5
}
```

自定义版本

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
struct point{
    point(){

    }
    point(int _x, int _y){
        x = _x;
        y = _y;
    }
    int x;
    int y;
};

bool cmp(point a, point b){
    return a.x < b.x;
}
int main(int argc, char *argv[]){
    point a[5];

    a[0].x = 1;
    a[0].y = 100;

    a[1].x = 100;
    a[1].y = 1;

    a[2].x = 30;
    a[2].y = 50;

    a[3].x = 25;
    a[3].y = 120;

    a[4].x = 301;
    a[4].y = 103;
    // 随便赋值
```

```
    sort(a, a + 5, cmp);
    // 先排序
    for (int i = 0; i < 5; i++){
        printf("a[%d].x = %d, a[%d].y = %d\n", i, a[i].x, i, a[i].y);
    }
    // 输出会发现他们按照x从小到大排序了
    cout << (lower_bound(a, a + 5, point(1, 1000), cmp) - a) << endl;
    // 第一个x值大于1的元素是(1, 100)这个元素，它的下标为0
    cout << (lower_bound(a, a + 5, point(101, 1000), cmp) - a) << endl;
    // 第一个x值大于101的元素是(301, 103)这个元素，它的下标为4
    cout << (lower_bound(a, a + 5, point(1000, 1000), cmp) - a) << endl;
    // 因为找不到所以返回a + 5，再减a就是5

}
```

upper_bound()

用法跟lower_bound()一样，只不过它返回的是**第一个大于x的值的地址， 而lower_bound()是返回第一个大于等于x的值的地址，> 和 >= 是二者的区别**

示例

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(int argc, char *argv[]){
    int a[] = {1, 3, 5, 7, 9};
    cout << (upper_bound(a, a + 5, 1) - a) << endl;
    // 第一个大于1的元素是3，下标是1
    cout << (upper_bound(a, a + 5, 2) - a) << endl;
    // 第一个大于2的元素是3，下标是1
    cout << (upper_bound(a, a + 5, 7) - a) << endl;
    // 第一个大于7的元素是9，下标是4
    cout << (upper_bound(a, a + 5, 100) - a) << endl;
    // 最大的元素也不比100大，故返回值是last，再减a也就是5
}
```

如果要求第一个小于x的数，或者第一个小于等于x的数怎么办呢？

还是用上面两个函数就可以了

查找第一个小于等于x的数（注：这里的意思为小于x的数里最大的那个）

算法：upper_bound()的返回值 - 1，就是要查找的地址

比如数组是int a[] = {1, 3, 5, 7, 9}要查找的数x是3
用upper_bound()找到的是第一个大于3的数对吧，它的返回值是5的地址，把返回结果再减1就好了，就是3的地址了。第一个小于等于3的元素是3，没错吧！

这是刚好数组里有x的情况，那如果没有呢？比如要查找的元素是4，那upper_bound()的返回值是5的地址，再减1，就是3的地址了，第一个小于等于
4的元素是3，没错吧！

那如果查的元素比第一个元素还要小呢，比如-1，那upper_bound()的返回值是a，再减1就是a的前一个单元了。所以这里得特判了，要不然段错误了。

查找第一个小于x的数

算法：lower_bound()的返回值 - 1，就是要查找的地址

还是用上面的数据为例子
要查找的元素为7，lower_bound的返回值为7的地址，再减一就是5的地址，第一个小于7的元素是5，没错。
要查找8呢，lower_bound()返回的是9的地址，再减一就是7，没错。
查找-1，lower_bound()返回1的地址，也就是a，再减一为a的前一个单元，会越界，需要特判。