2. 二分查找算法实现

在模块 LinearSearch 中增加实现二分查找算法的 BinarySearch 函数模板。该函数通过返回一个整数值来表明查找的结果，这不同于 C++标准库的 binary_search 函数。

```
template <typename T>
int BinarySearch(const T& k, const T* items, int len) {
    int mid = 0, left = 0;
    int right = len - 1;
    while (left <= right) {
        mid = (left + right) / 2;
        if (k == items[mid])
            return mid;
        else if (k < items[mid])
            right = mid - 1;
        else
            left = mid + 1;
    }
    if (k > items[mid])  mid++;
    return ~mid;
}
```

返回整数的方案与 C#/Java 类库中的 BinarySearch 方法相同。如果在数组 items 的区域 range=[0, len)中找到 k，则返回某个值为 k 的元素的索引（零或正整数，如果区域中含有多个值为 k 的元素，则无法保证找到的是哪一个）。如果找不到 k，则返回值为一个负数 r，其反码（又称按位补码）i（即 i = ~r）正好是将 k 插入原序列并保持其排序的正确位置。即，如果 k 小于区域中的一个元素，则返回区域中大于 k 的第一个元素的索引 i 的按位补码 r。r 和 i 之间存在如下关系：i＝~r＝-r-1，r＝~i＝-i-1。如果 k 大于数组 ar 中的所有元素，则返回最后一个元素的索引加 1 的按位补码。

整数返回值表达出更丰富且准确的内涵，便于在查找操作后实施其他操作。根据返回值的规则，如果返回值 r＜0，则说明数组 items 中没有要查找的数据 k；如果返回值 r＝-1，则说明 k < items[~r] = items[0]；如果~r = len，则说明 k > items[~r - 1] = items[len -1]；其他情况则有 items[~r - 1] < k < items[~r]。

【例11.1】　创建一个具有随机值的数组，对其进行排序后测试顺序和二分查找算法。

```
#include <iostream>
#include <algorithm>
#include "../dsa/LinearSearch.h"
#include "../dsa/dsaUtils.h"
int main() { // LinearSearchTest.cpp
    const int CNT = 12; int items[CNT];
    RandomizeData(items,CNT,7,1,100); // seed=7, [1,100)
    cout << "随机排列： "; Show(items, CNT);
    cout << "排序后 ： ";
    sort(items, items + CNT); Show(items, CNT);
    int k = 50;
    int re = Index(items, items + CNT, k); cout << re << endl;
```

```
    re = BinarySearch(k, items, CNT);
    cout << "k= " << k << ", re= " << re << ", i=~re= " << ~re << endl;
    return 0;
}
```

程序运行结果如下：

```
随机排列：  1 53 46 22 47 20 92 36 84 62 15 86
排序后 ：   1 15 20 22 36 46 47 53 62 84 86 92
-1
k= 50, re= -8, i=~re= 7
```

结果说明，在随机产生的一组数中不包含 50 这个数；如果要将 50 插入排好序的一组数中，应该将它插入到 7 号位置，即 47 和 53 之间，才能保持其排序。