# 1 项目简介

二叉排序树就是指将原来已有的数据根据大小构成一棵二叉树，二叉树中的所有结点数据满足一定的大小关系，所有的左子树中的结点均比根结点小，所有的右子树的结点均比根结点大。此后，中序遍历所得到的序列即为从小到大的排序序列。

# 2 程序说明

## 2.1 数据结构设计

二叉树排序本身就是利用树结构进行排序，因此，本程序随着依次输入的关键字建立二叉排序树，接着利用中序遍历，输出从小到大的数字排序序列。

本质上，利用二叉树排序是利用二叉树节点间的左右关系反映数字之间的大小关系，但正因为如此，二叉树之间的左右关系为在树中快速搜索找到某个数字提供了路径方向，即如果当前节点不是目标，那么二叉排序树左右节点与大小关系间的对照可以判断出下一步该往左考察还是往右考察。

/\*\*\*说明：本程序的二叉树排序不仅仅可以实现正数排序，正负数排序都可以实现，而且可以对错误的非法输入作出识别与处理。\*\*\*/

本程序提供了4个操作选项：1表示建立或重建二叉排序树， 2表示插入元素，

3表示查询元素， 4表示退出程序。

此外，为了使程序在空树状态下不出现指针指向等问题，为树增加了根节点。这样一来，即使树中没有任何数字，插入元素、查询元素的功能都可以正常运行。

同时考虑到排序树不可用于char或者string型的类，所以为了防止将二叉排序树实例化为字符型的类，本程序通过静态关键字：

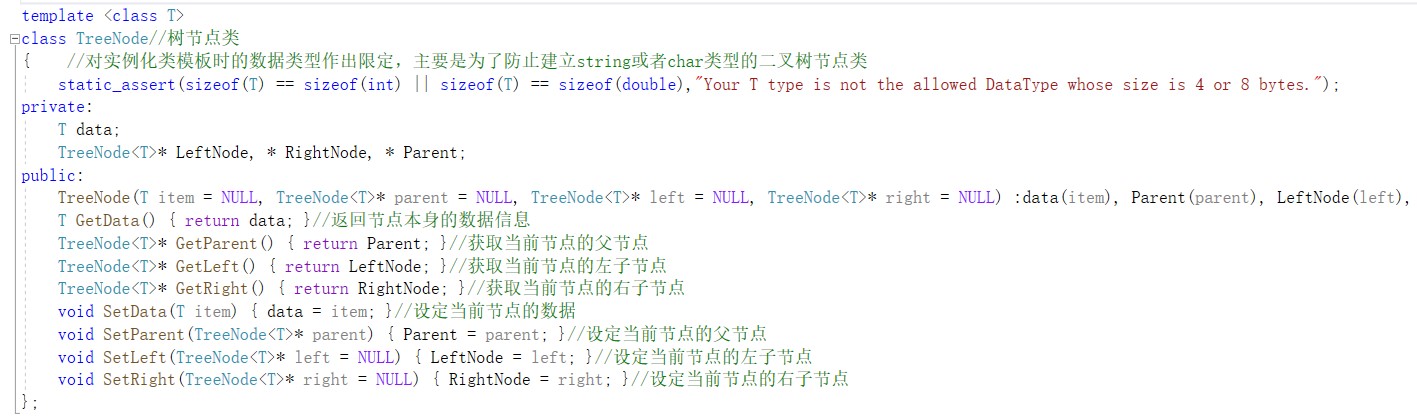
static\_assert(sizeof(T) == sizeof(int) || sizeof(T) == sizeof(double)

对类模板支持的数据类型做了限定。

## 2.2 类结构设计

①树节点TreeNode类：

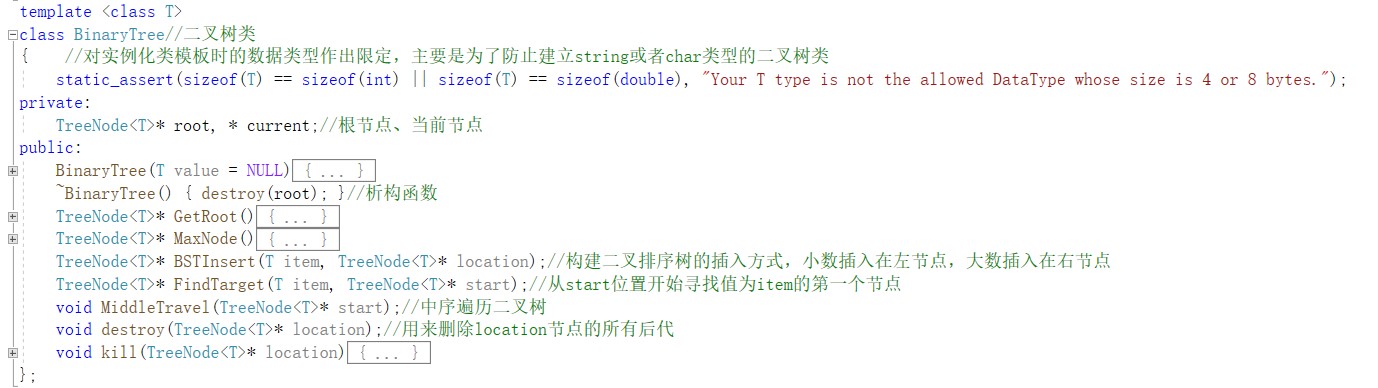
此类包含着4个信息：树节点本身的数据（即元素本身携带的数值）、左节点（小数存放的方向）、右节点（大数存放的方向）、父节点。此外，还设计了一系列函数以为Tree类实现插入、排序等功能提供基础。



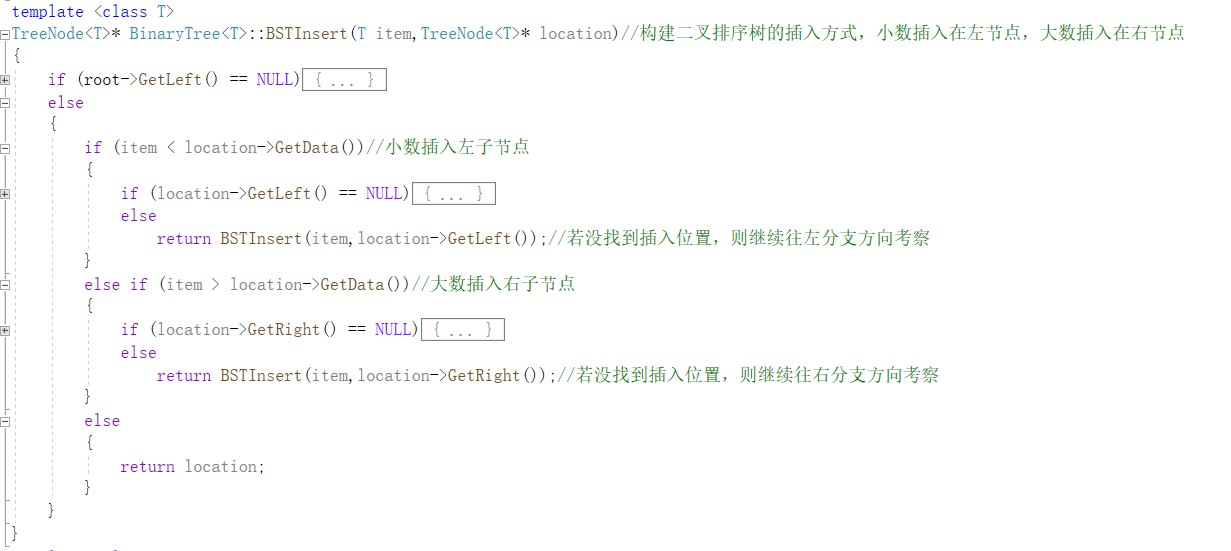
此外，为了在树结构中遍历时直接访问当前节点的父节点，本程序在新建立节点时，直接通过parent指针指向了父节点，而不是每次寻找父节点都需要通过自根节点开始的遍历寻找特定节点的父节点，以空间换取时间，以提升程序效率。

## ②二叉排序树BinaryTree类：

## 二叉排序树中建立二叉树、插入新元素、查找元素、元素大小排序等操作的基础功能都是通过调用BinaryTree类函数来实现的，其类函数也是本程序的**核心代码**，接下来会对上述函数进行详细说明：



1. 构造函数：实例化类时自动调用，动态开辟内存存储新节点，需要注意的是：将current指针与root指针一起指向实例化类时建立的第一个新节点，即以该节点作为二叉树的根节点。
2. 析构函数、destroy函数：析构函数对树所占用的节点空间的释放实际上是通过调用destroy函数实现的，而destroy函数还可以把传入参数指出的节点位置作为删除起始位置，删除树的某一部分子树。简单来说，destroy函数是利用递归的思想，在本程序中递归流程与后续遍历的顺序一样，通过遍历树或者子树，完成删除操作。
3. kill函数：功能是：用来删除特定的叶节点或者孤立节点。需要注意的一点是，在删除树的某一个节点时，若采用if先判断该节点是否为NULL，不为NULL则用delete删除，这一思路是有缺陷的。因为在实际调试过程中发现，通过if判断某节点是否为NULL来表示该节点空不空有时会发生这样的问题：本来上其通过指针连接的节点已经被删除，但指针有时不会因为删除其指向的节点的内容而自动指向NULL,因此，无论是kill函数还是destroy函数，本程序在使用delete释放相应节点内存空间之前，都会显式地将与其相关联的其他节点指针指向NULL。
4. BSTInsert 函数：功能是按照树左右分叉的关系与数字大小关系的对应将新的节点按照二叉排序树的建立规则插入到正确的位置。



结合上图中的代码框架，可以看出此函数的算法逻辑：首先判断插入的节点属于的类型：当插入当前节点之前的二叉排序树除了根节点以外不存在其他节点，则将节点插入根节点的第一个左子节点位置。除了这种情况之外，在插入节点之前先比较待插入节点值与当前节点值的大小关系，若item < location->GetData()，则表明应当将新节点插入到当前节点左分支的方向。查找左分支方向具体插入位置的流程则利用了递归思路：首先明确递归终止条件：如果location->GetLeft() == NULL，那么说明已经遍历到了原树的可插入新节点的位置，从而通过设定节点间的指针关系将新节点插入到定位到的节点的左子节点位置，完成小数的插入，否则继续递归。当item > location->GetData()，同理寻找对应的插入位置将新节点连在其右子节点位置即可。此外，如果出现了item ==location->GetData()的情况，说明待插入节点值与树中已有节点值重复，二叉排序树中不可以存在同值节点，此时直接return结束整个函数调用。

5.FindTarget函数：功能是从start位置开始寻找数中第一个值为target的节点，并返回其位置。由于通过比较节点值所得到的大小关系对应了下一步查找应该往左考察还是往右考察，简介得到了搜索的路径，因此，这里可以通过循环结构更快速地检索：只要当前节点不为空，就一直循环，如果在循环过程中找到目标则退出循环，并返回目标节点位置，否则如果当前节点值小于target，则下一步在当前节点的右分支方向寻找，如果当前节点值大于target，则下一步在当前节点的左分支方向寻找。如果循环执行过程中一直没有跳出循环，或者直接因为start==NULL,未执行循环，那么说明当前树中不存在目标节点，返回NULL。

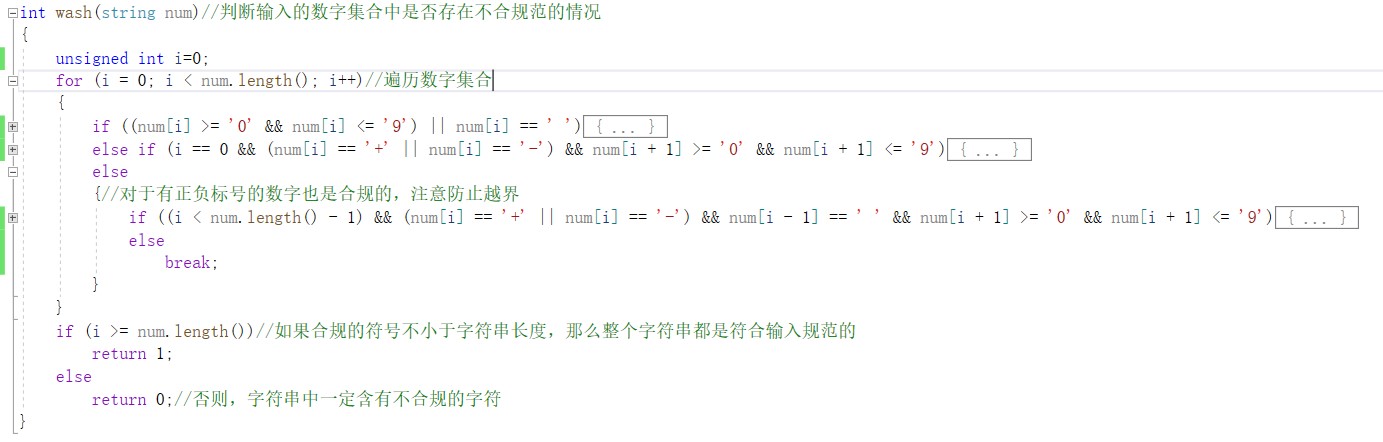
6.中序遍历MiddleTravel函数：本程序中都是利用递归思路实现：按照先中间节点再左子节点最后右子节点的顺序应用于每一个节点，即可完成对整棵树的遍历。

7.MaxNode函数：功能：获取二叉排序树中最大的节点

需要明确的一点是：按照排序二叉树的逻辑，二叉排序树中最大的节点，位置在二叉排序树的最右分支的最后一个节点，这也是中序遍历最后一个访问的节点。所以通过循环结构访问根节点第一个左子节点的右分支，只要没到最右分支的最后一个节点，就一直循环，最终循环结束之后，返回暂存结构所储存的最大值节点的地址。

**3.程序实现流程**

# 首先需要说明的是：由于实例化的类中，要求节点数据为int型数据，因此，由用户输入的字符的格式规范性直接影响着排序二叉树能否正确建立以及能否正确排序。所以，这里先介绍本程序对用户输入的数据的非法无效输入判定以及数字识别机制：

对于用户输入的字符串中是否均为数字，是否掺着字母、符号（正负号除外）等其他非数字字符的判断是通过wash函数来判断与处理的：

# 首先按行读入用户输入的所有字符（事实上，用户没敲一个回车即表示输入结束，所以用户输入的数字也只会有一行），将字符串作为参数传入wash函数，通过循环结构遍历字符串，在此过程中，若遇到0-9纯数字字符或者空格，则计数，而有时我们会在数字前面添加正负号表示数字的正负，因此，在遇到表示正负的正负号时也应该计数，但是有一点需要注意的是：在对符号进行考察时，需要结合num[i-1]、num[i]、num[i+1]综合判断该字符是否是表示正负的符号，这里就需要注意当i==0时i-1会越界，当i==length()-1时，i+1会越界，因此程序中对这两种情况单独给出了分支结构进行判断。这样一来，循环结束后，比较有效字符数量和原来的字符串长度length，若前者不少于后者，则说明用户输入的字符串不存在非法的无效字符，否则需要提示用户非法输入，重新输入。

此外，由于直接输入的是字符串，而二叉排序树要求的是int型数据，因此，在判断完用户输入的字符串中不存在非法无效数字之后，通过调用string类型的stoi函数与substr函数将表示数字的字符段截取出来后转换成int型数据，即可再调用BSTInsert函数即可在二叉排序树中插入新节点。

接下来对程序的执行流程进行说明：

首先实例化二叉排序树类，即带根节点的树结构，紧接着根据用户的不同选择进入下面不同的分支程序：

1.建立或重建二叉排序树：首先要判断当前的树是不是除了根节点以外还存在其他的节点，若存在，则将此操作视为重建二叉排序树，而且在重建二叉树之前要先删除之前已经存在的二叉树，若不存在，则直接进行后续操作：读入用户输入的数字，之后调用wash函数对非法输入的判断与处理，若存在非法字符，则重新输入。若不存在非法输入，则进入建树流程：先调用FindTarget函数判断当前二叉树中是否已经存在与待插入节点相同数值的节点，由于二叉排序树中不支持相同节点，所以当判断完不存在同值节点后，再调用BSTInsert函数将节点插入，最后调用MiddleTravel函数中序遍历二叉树给出排序结果。

2.插入元素：首先有用户输入待查找元素值并判断是否输入非法，并判断输入是否非法，此后同样先调用FindTarget函数判断当前二叉树中是否存在同值节点，若不存在，则调用BSTInsert函数插入新节点，否则给出数值重复信息提示，且不执行插入操作。

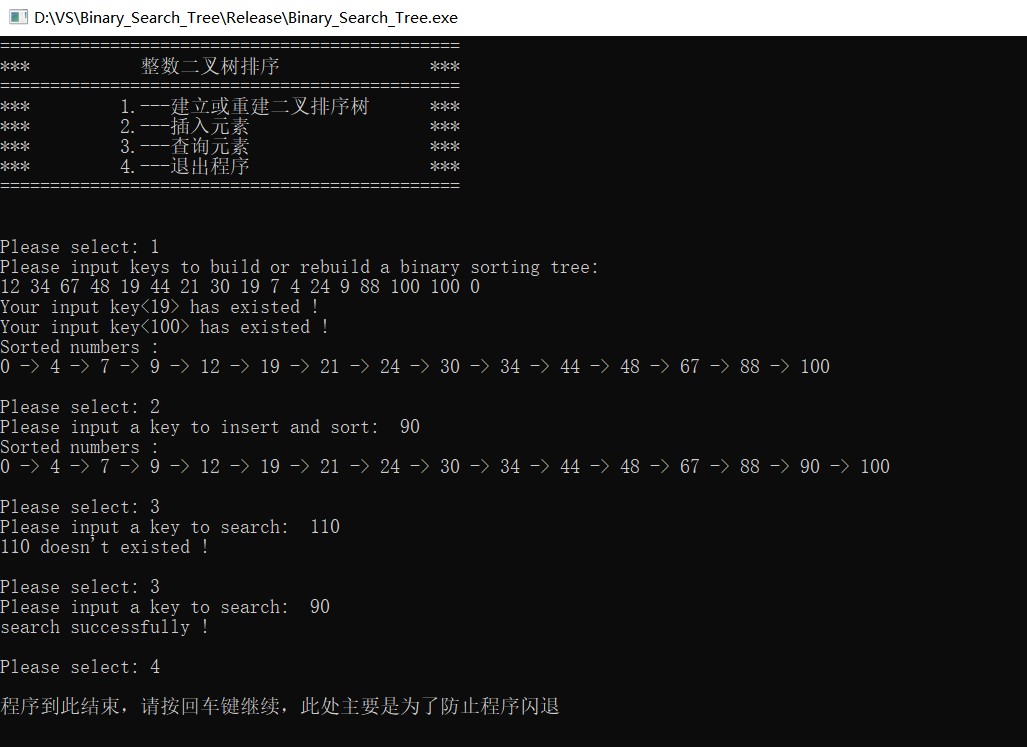
3.查询元素：首先有用户输入待查找元素值并判断是否输入非法，若存在非法无效输入则重新输入数值。此后直接调用FindTarget函数依据返回值输出查询结果即可。

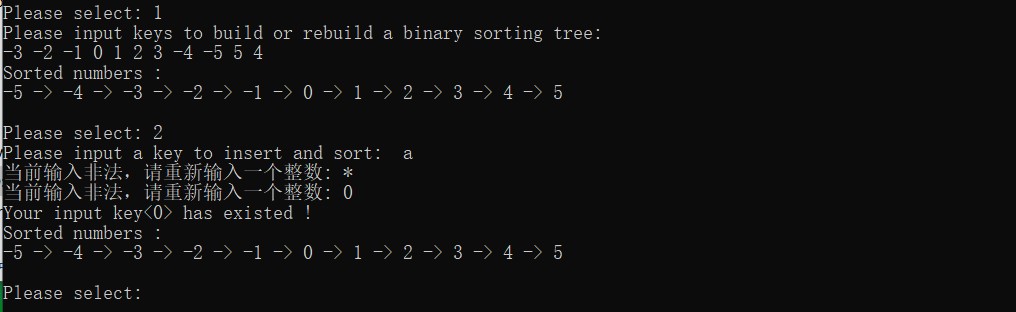
4.当用户输入选项4，则结束整个程序的最外层循环，退出程序。

# 4.测试

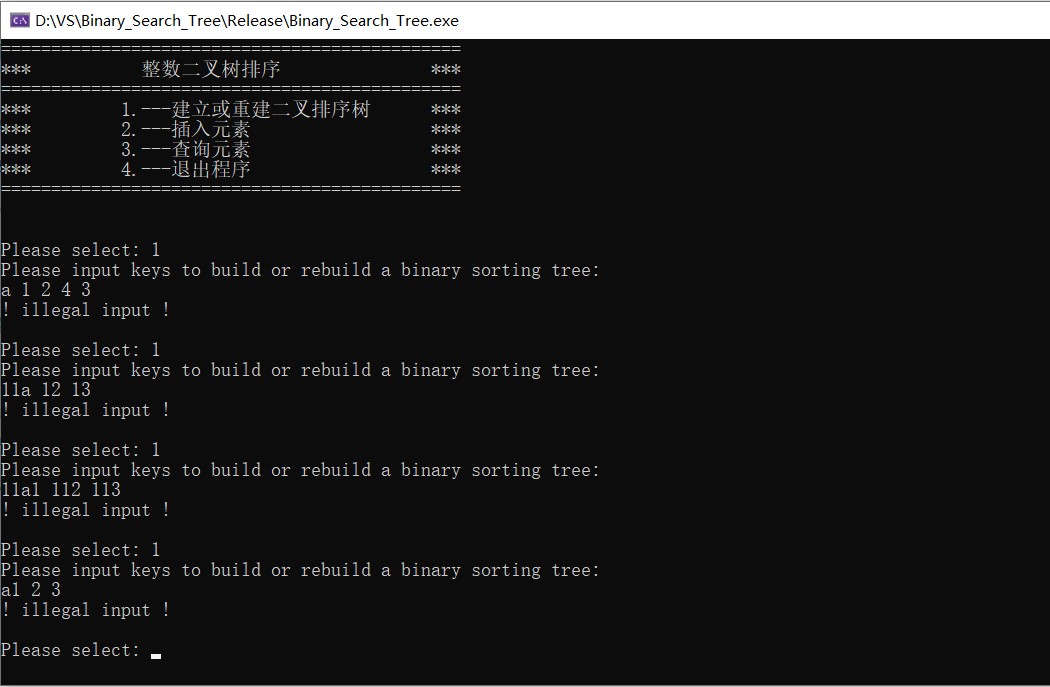
## 4.1 功能测试

# ①正数建立二叉排序树、插入元素、查询元素功能测试：

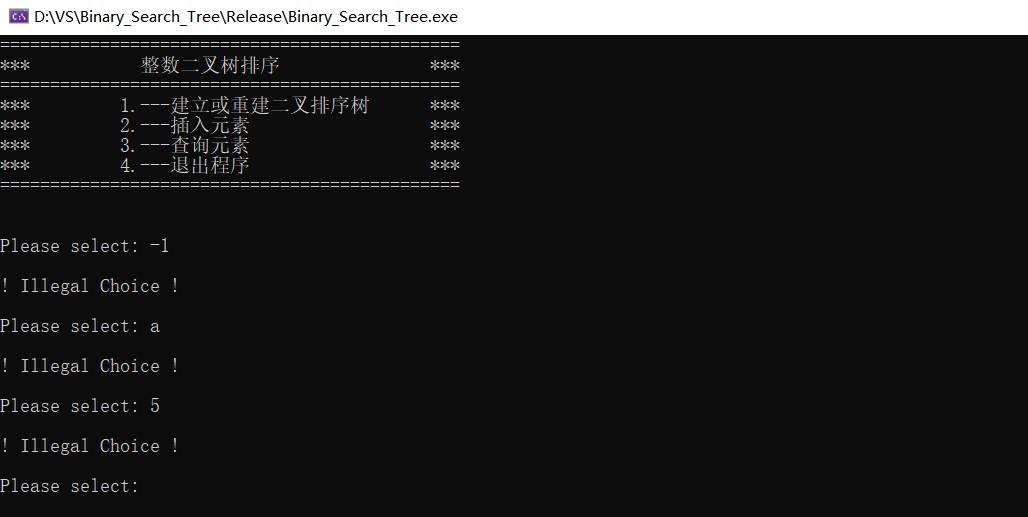


②功能2--插入元素时错误输入处理功能测试：

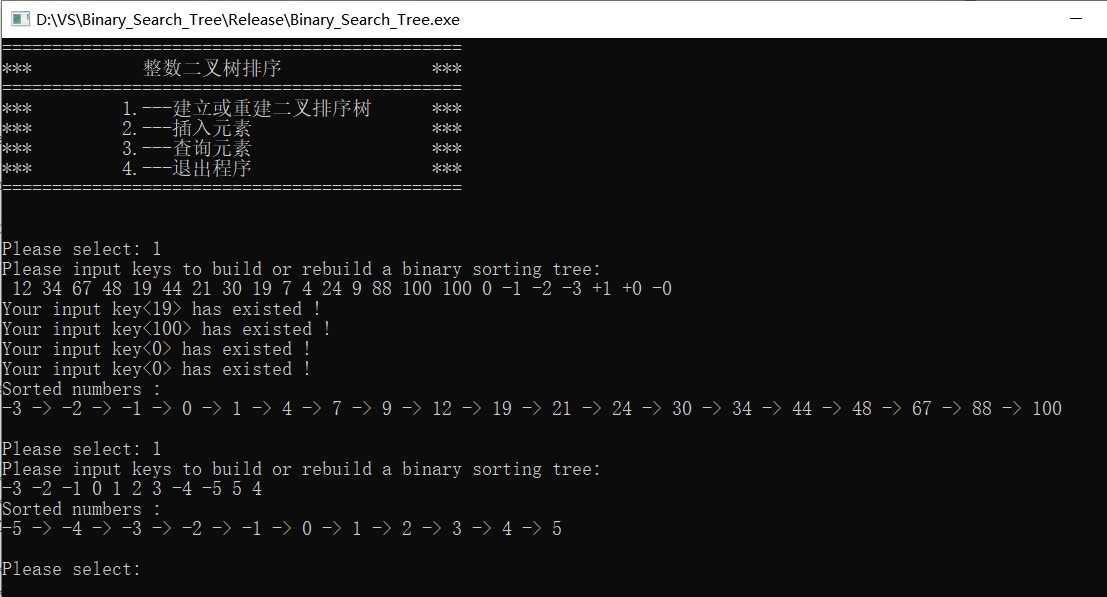
③建立二叉排序树时用户输入的数字中掺杂非法字符时的非法输入处理功能：

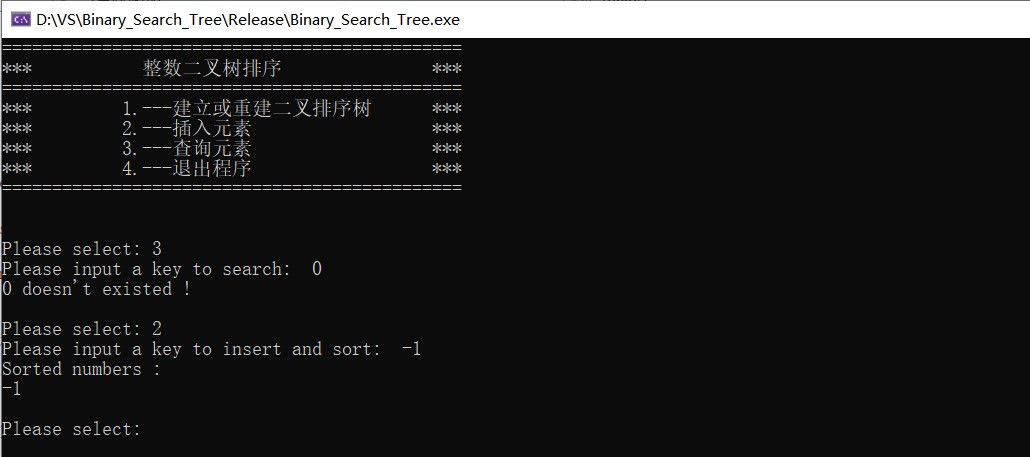


④用户输入1-4以外的无效选项时的错误输入处理功能测试：



⑤数字前有正负号表示数字正负的数字正常排序功能测试：





⑥边界测试：当树中只剩下一个根节点时，程序正常执行功能测试：