

**地理科学与测绘工程学院**

**《空间数据结构》实验报告**

姓 名： 陈昊

学 号： 21200214230

班 级： 地信2112

指导教师： 史守正

日 期： **2023.06.09**

实验报告成绩：

**地理科学与测绘工程学院**

**2022-2023学年第2学期**

实验1.1 线性表输入整形元素

**实验内容**

本次实验包括以下主要任务：

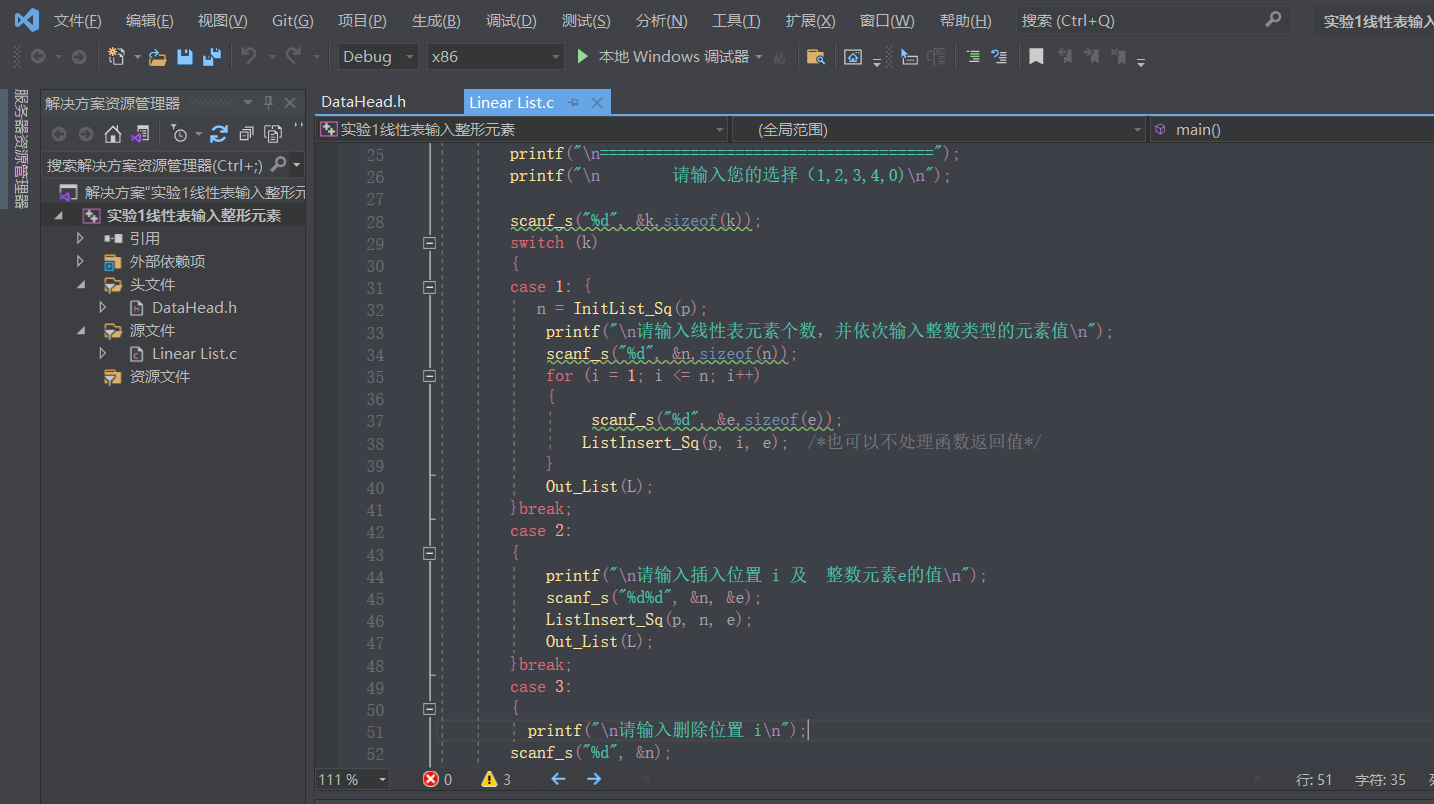
* 定义线性表顺序存储的数据结构
* 实现线性表表的插入、删除、查找和输出等操作
* 创建菜单，进行用户界面操作

**程序的结构**：

主要包括以下函数：

| **函数名** | **功能描述** | **参数说明** |
| --- | --- | --- |
| Status InitList\_Sq(SqList\* L) | 初始化线性表。 | L：指向 SqList 结构体的指针。 |
| void Out\_List(SqList L) | 输出打印线性表。 | L：SqList 结构体。 |
| Status ListInsert\_Sq(SqList\* L, int i, ElemType e) | 在顺序线性表 L 的第 i 个元素之前插入新的元素 e。 | L：指向 SqList 结构体的指针 i：插入位置 e：要插入的元素 |
| Status ListDelete\_Sq(SqList\* L, int i, ElemType\* e) | 在顺序线性表 L 中删除第 i 个元素，并用 e 返回其值。 | L：指向 SqList 结构体的指针 i：删除位置 e：被删除元素的值的指针 |
| int LocateElem\_Sq(SqList L, ElemType e, Status(\*compare)(ElemType, ElemType)) | 在顺序线性表 L 中查找第一个值与 e 满足 compare() 的元素的位序。 | L：SqList 结构体e：要查找的元素 compare：比较两数相等的函数指针 |
| Status compare(ElemType e, ElemType e1) | 比较两个元素是否相等。 | e 和 e1：要比较的两个元素。 |

**程序截图（部分）**



**数据结构：**

1.SqList是线性表的结构体，包含了存储线性表元素的数组指针（elem）、线性表的当前长度（length）以及线性表的最大存储容量（listsize）。

 elem：指向存储线性表元素的数组的指针。

 length：表示线性表的当前长度。

 listsize：表示线性表的最大存储容量。ElemType 是链表中元素的类型，它被定义为 int 类型，可以根据需要修改为其他类型。

2.ElemType 是链表中元素的类型，它被定义为 int 类型，可以根据需要修改为其他类型。

**输入/输出设计：**

程序通过设计一个程序菜单，用户通过键盘输入和屏幕输出进行交互。根据用户选择的操作，程序接收输入并在屏幕上输出相应的结果。

* ListInsert\_Sq(SqList\* L, int i, ElemType e)函数中通过循环将用户输入的数据顺序插入到线性表，用户需要依次输入每个元素的值。
* Out\_List()函数用于打印链表中的所有元素。

**程序变量说明：**

* L：SqList类型的变量，表示线性表。
* i：整数类型的变量，表示插入或删除的位置。
* e：ElemType类型的变量，表示插入或删除的元素值。
* k：整数类型的变量，表示菜单的选择项。

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证线性表的功能：

* 创建一个空线性表表，执行插入操作，查看链表是否正确插入元素。
* 创建一个包含多个元素的线性表，执行删除操作，查看线性表表是否正确删除元素。
* 创建一个线性表，执行查找操作，查看是否能正确找到指定位置的元素。

**实验结果与分析**

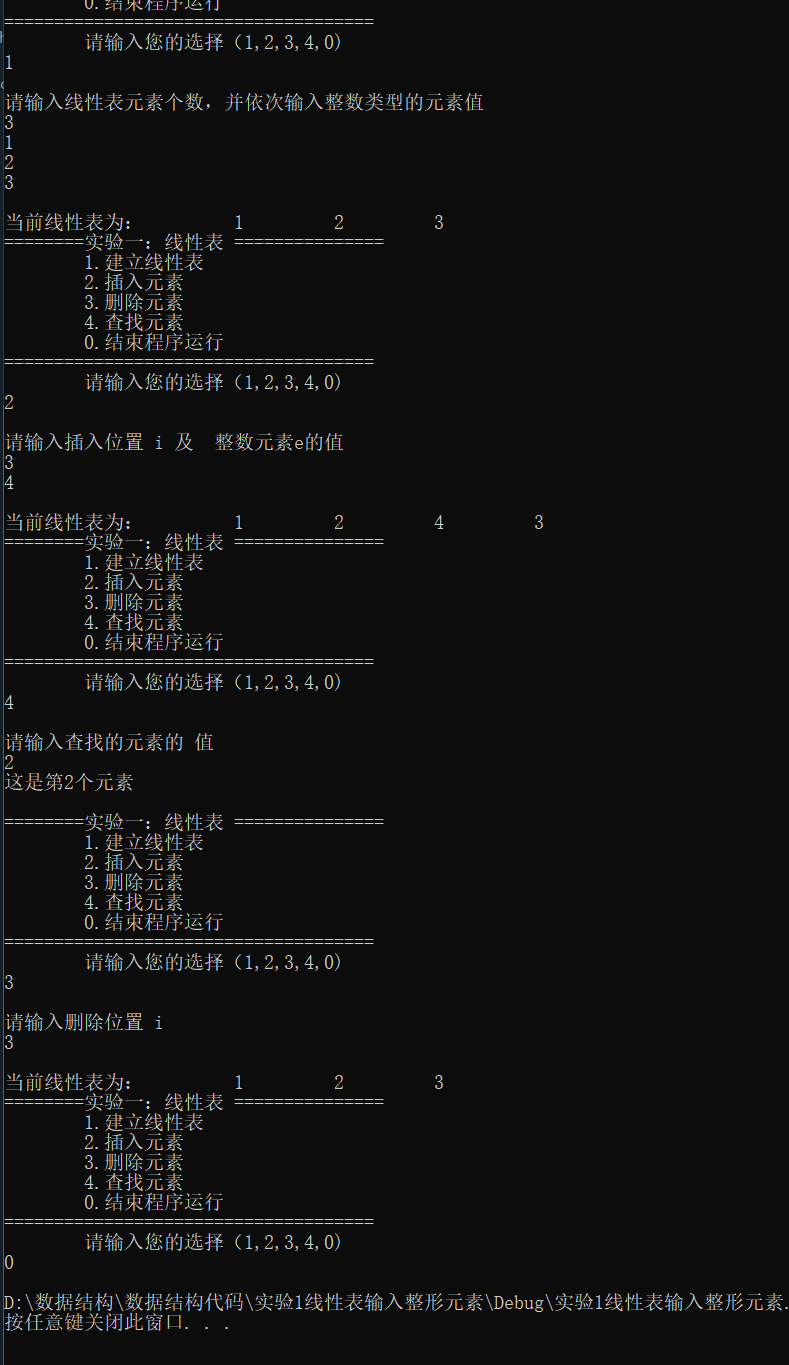
程序正确性验证： 经过多次测试和调试，程序的功能在不同场景下都能正确执行，包括插入、删除和查找操作。输出结果与预期结果一致，表明程序在功能实现上是正确的。

程序时间复杂度分析： 根据程序中各函数的算法描述，插入和删除操作的时间复杂度为O(n)，其中n为线性表的长度。查找操作的时间复杂度为O(n)，在最坏情况下需要遍历整个线性表。因此，程序的时间复杂度为O(n)。

程序的空间复杂度分析： 程序中使用了一个SqList结构体变量L来存储线性表的信息，其中包括元素数组、长度和存储容量等信息。在初始化线性表时，根据LIST\_INIT\_SIZE的大小分配了一定的内存空间。当插入操作需要增加存储容量时，通过realloc函数重新分配内存空间，增加LISTINCREMENT大小的内存。因此，程序的空间复杂度与线性表的存储容量相关，为O(n)。

所以，此次实验的程序在功能实现上是正确的，时间复杂度为O(n)，空间复杂度为O(n)。通过对程序的多次测试和调试，可以验证其在不同场景下的正确性和稳定性。

**二、运行输出结果：**



**三:调试和运行程序过程中产生的问题及采取的措施：**

在做此次实验前，先去了解了线性表的存储结构，线性表有两种主要的存储结构：顺序存储结构和链式存储结构。

1. 顺序存储结构：

顺序存储结构使用一段连续的存储空间来存储线性表的元素，可以是数组或是类似数组的数据结构。

元素在内存中的存储位置是连续的，通过元素在序列中的位置（索引）和存储空间的起始地址来进行访问。

顺序存储结构支持随机访问，可以通过索引直接访问任意位置的元素。

插入和删除操作可能需要移动其他元素，以保持线性表的顺序性。

顺序存储结构适用于元素的访问频繁、大小固定或者预先知道大小的情况。

1. 链式存储结构：

链式存储结构使用节点（Node）来存储线性表的元素，每个节点包含数据域和指针域。

数据域存储元素的值，指针域指向下一个节点（或前一个节点，对于双向链表）。

节点在内存中可以是离散的，通过指针连接在一起形成链表。

链式存储结构支持动态分配内存，可以根据需要插入和删除节点，不需要移动其他元素。

访问链式存储结构中的元素需要从头节点开始按照指针依次遍历。

链式存储结构适用于元素的插入和删除频繁、大小不固定或者未知大小的情况。

在了解基本概念后，线性表的顺序存储就较好实现

定义线性表的长度，以及每添加一个元素需要多少增量空间，即可实现线性表的顺序存储

实验1.2 线性表输入结构体元素

**实验内容**

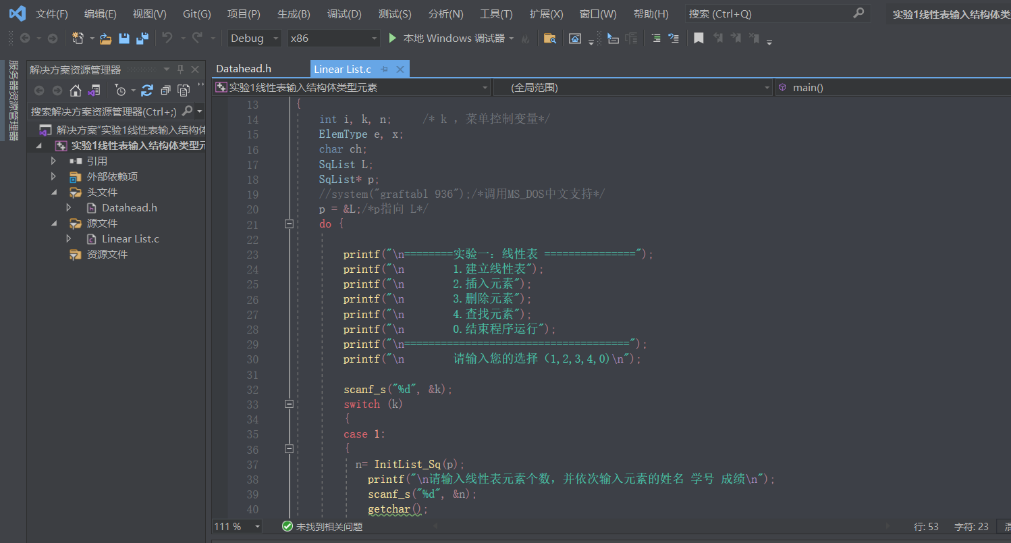
本次实验包括以下主要任务：

* 定义线性表顺序存储的数据结构
* 实现线性表表的插入、删除、查找和输出等操作
* 创建菜单，进行用户界面操作

**程序的结构**：

主要包括以下函数：

| **函数名** | **功能描述** | **参数说明** |
| --- | --- | --- |
| InitList\_Sq | 初始化线性表 | SqList\* L：指向线性表的指针 |
| Out\_List | 输出线性表 | SqList L：线性表结构体类型 |
| ListInsert\_Sq | 在顺序线性表中插入元素 | SqList\* L：指向线性表的指针 int i：插入位置 ElemType e：插入的元素 |
| ListDelete\_Sq | 在顺序线性表中删除元素 | SqList\* L：指向线性表的指针 int i：删除位置 ElemType\* e：存储被删除元素的指针 |
| LocateElem\_Sq | 在顺序线性表中查找元素位置 | SqList L：线性表 ElemType e：待查找的元素 Status(\*compare)(ElemType, ElemType)：比较两数的函数指针 |

程序截图(部分):

**数据结构：**

* typedef struct ElemType：定义线性表内的元素类型为学生数据类型，包括姓名、学号和分数。
* typedef struct SqList：定义线性表的动态分配顺序存储结构，包括元素数组指针、长度和存储容量。

**输入/输出设计：**

输入：通过函数参数传入需要操作的线性表、插入或删除的位置、要插入或删除的元素等。

输出：在控制台输出线性表的内容，包括姓名、学号和分数。

* ListInsert\_Sq(SqList\* L, int i, ElemType e)函数中通过循环将用户输入的学生数据顺序插入到线性表，包括姓名，学号，分数三个部分组成一个线性表数据，用户需要依次输入每个元素的值。
* Out\_List()函数用于打印线性表中的所有学生属性元素。

**程序变量说明：**

* SqList\* L：指向线性表的指针，用于传递线性表的地址。
* ElemType e：用于存储要插入或删除的元素。
* int i：用于表示插入或删除的位置。
* k：整数类型的变量，表示菜单的选择项。

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证线性表的功能：

 创建一个空线性表，执行插入操作，查看线性表是否正确插入元素。

 创建一个包含多个元素的线性表，执行删除操作，查看线性表是否正确删除元素

 创建一个线性表，执行查找操作，查看是否能正确找到指定位置的元素实验结果与分析

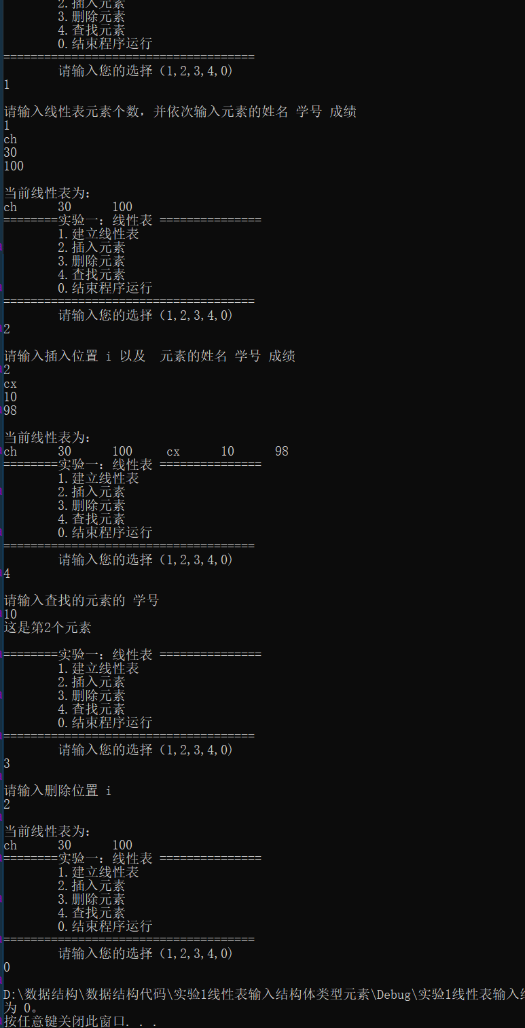
程序正确性验证： 经过多次测试和调试，程序的功能在不同场景下都能正确执行，包括插入、删除和查找操作。输出结果与预期结果一致，表明程序在功能实现上是正确的。

程序的时间复杂度分析： 根据代码实现，可以得出以下几个函数的时间复杂度分析：

* InitList\_Sq：时间复杂度为O(1)，它只涉及一次内存分配操作。
* Out\_List：时间复杂度为O(n)，其中n为线性表的长度。该函数需要遍历线性表中的所有元素并进行输出。
* ListInsert\_Sq：时间复杂度为O(n)，其中n为线性表的长度。插入操作需要将插入位置之后的元素依次后移，最坏情况下需要移动n个元素。
* ListDelete\_Sq：时间复杂度为O(n)，其中n为线性表的长度。删除操作需要将删除位置之后的元素依次前移，最坏情况下需要移动n个元素。
* LocateElem\_Sq：时间复杂度为O(n)，其中n为线性表的长度。查找操作需要遍历线性表中的所有元素进行比较，最坏情况下需要遍历n个元素。

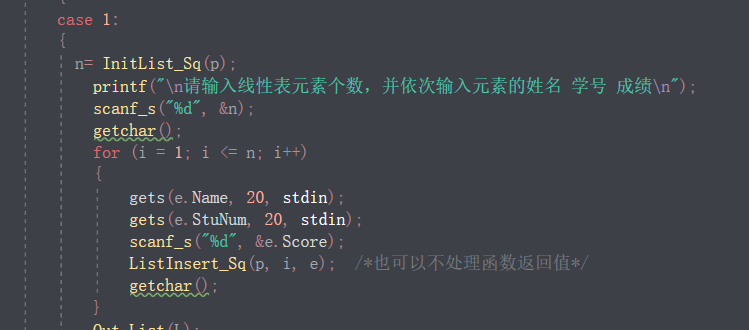
程序的空间复杂度分析： 根据代码实现，程序的空间复杂度主要体现在线性表的存储空间上。具体分析如下：

* InitList\_Sq：空间复杂度为O(1)，它只涉及一次线性表的初始化和内存分配操作。
* Out\_List：空间复杂度为O(1)，它只涉及输出操作，不需要额外的存储空间。
* ListInsert\_Sq：空间复杂度为O(1)，它只涉及插入操作，不需要额外的存储空间。
* ListDelete\_Sq：空间复杂度为O(1)，它只涉及删除操作，不需要额外的存储空间。
* LocateElem\_Sq：空间复杂度为O(1)，它只涉及比较和查找操作，不需要额外的存储空间。

**二、运行输出结果：**

**三:调试和运行程序过程中产生的问题及采取的措施：**

本次实验在进行输入数据的时候，会发现输入的学生类型的属性数据会被程序给吞掉一个，这是因为scanf函数输入时候会存在内存缓冲区读取字符的原因，利用getchar函数读取掉换行的字符，就不会出现这个问题了。



实验1.3 线性表的链式存储结构

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

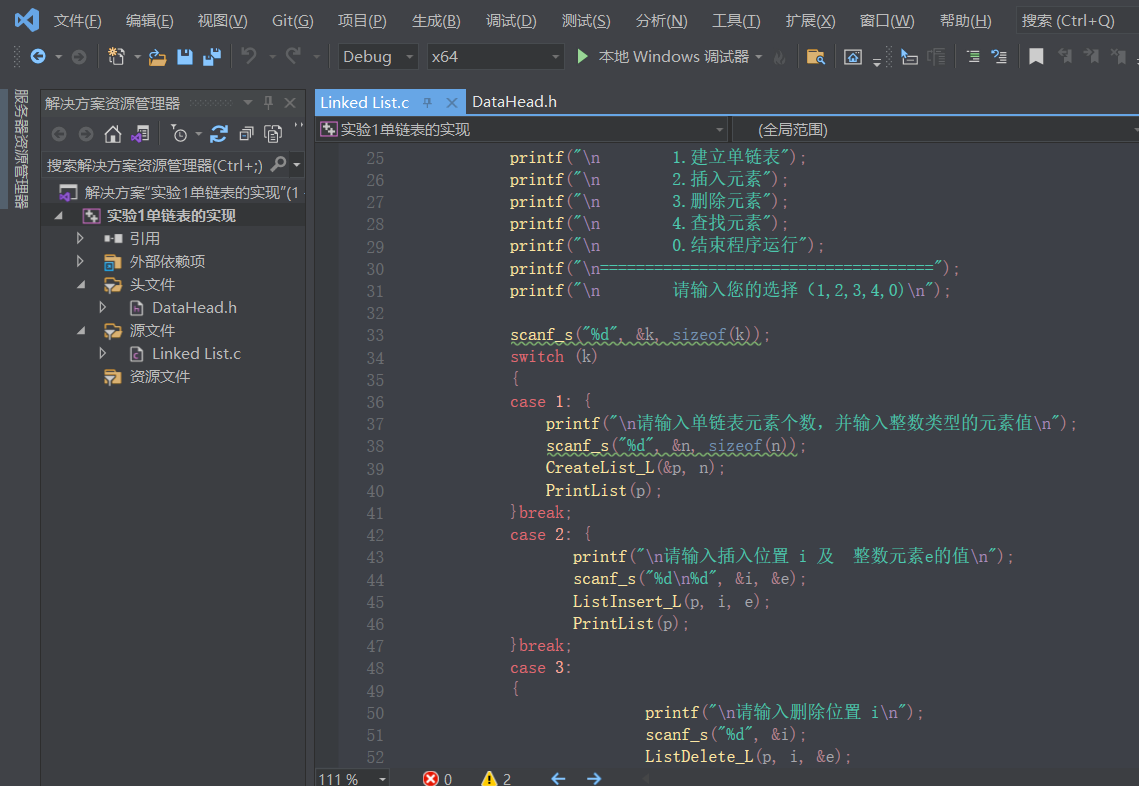
* 定义线性链表节点的数据结构
* 实现链表的插入、删除、查找和输出等操作
* 创建菜单，进行用户界面操作

**程序的结构**：

主要包括以下函数：

| **函数名** | **功能描述** | **参数说明** |
| --- | --- | --- |
| GetElem\_L | 查找链表中第i个元素的值，并将其存储在指针e所指向的变量中。 | L: 链表的头指针。 i: 要查找的元素位置。 e: 用于存储查找到的元素值的指针。 |
| ListInsert\_L | 在链表的第i个位置之前插入元素e。 | L: 链表的头指针的指针，传入指针的地址以修改指针的值。 i: 要插入的位置。 e: 要插入的元素值。 |
| ListDelete\_L | 删除链表中的第i个元素，并将其值存储在指针e所指向的变量中。 | L: 链表的头指针的指针，传入指针的地址以修改指针的值。 i: 要删除的位置。 e: 用于存储删除的元素值的指针。 |
| CreateList\_L | 根据用户输入创建一个具有n个元素的链表。 | L: 链表的头指针的指针，传入指针的地址以修改指针的值。 n: 链表中元素的个数。 |
| PrintList | 打印链表中的所有元素。 | L: 链表的头指针。 |

**程序截图（部分）**



**数据结构：**

* LNode：定义了链表节点的结构，包括数据域data和指针域next，用于存储结点元素值和指向下一个节点的指针。
* ElemType 是链表中元素的类型，它被定义为 int 类型，可以根据需要修改为其他类型。
* LinkList 是指向链表节点的指针类型，用于表示链表的头指针。

**输入/输出设计：**

程序通过设计一个程序菜单，用户通过键盘输入和屏幕输出进行交互。根据用户选择的操作，程序接收输入并在屏幕上输出相应的结果。

* CreateList\_L()函数中通过用户输入逆序创建一个链表，用户需要依次输入每个元素的值。
* PrintList()函数用于打印链表中的所有元素。

**程序变量说明：**

* L：链表的头指针，用于指向链表的第一个节点。
* p、q、s：用于辅助节点操作的指针变量。
* i、j：循环计数器变量。

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证链表的功能：

* 创建一个空链表，执行插入操作，查看链表是否正确插入元素。
* 创建一个包含多个元素的链表，执行删除操作，查看链表是否正确删除元素。
* 创建一个链表，执行查找操作，查看是否能正确找到指定位置的元素。

**实验结果与分析**

程序正确性验证：

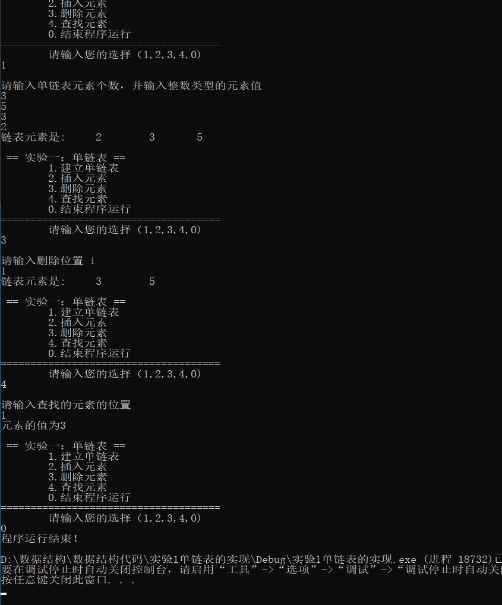
通过创建链表并执行插入、删除和查找，输出操作，可以验证程序的正确性。可以根据输入的数据和预期的结果进行比对，确保链表的操作和输出符合预期。

程序时间复杂度分析：

* GetElem\_L()函数的时间复杂度为O(n)，其中n为链表的长度。最坏情况下，需要遍历整个链表才能找到目标元素。
* ListInsert\_L()函数的时间复杂度为O(n)，其中n为链表的长度。最坏情况下，需要遍历整个链表找到插入位置，并进行插入操作。
* ListDelete\_L()函数的时间复杂度为O(n)，其中n为链表的长度。最坏情况下，需要遍历整个链表找到删除位置，并进行删除操作。
* CreateList\_L()函数的时间复杂度为O(n)，其中n为输入的元素个数。需要逐个输入元素，并创建相应的链表节点。
* PrintList()函数的时间复杂度为O(n)，其中n为链表的长度。需要遍历整个链表并输出每个元素的值。

程序空间复杂度分析：

* 空间复杂度取决于链表中节点的数量，即O(n)，其中n为链表的长度。需要分配内存来存储链表节点。

**二:运行输出结果：**

**四、调试和运行程序过程中产生的问题及采取的措施：**

链式存储，利用malloc开辟指针空间的时候会报空指针的错误，这是因为我们定义了指针但是没有指向任何变量，将他指向一个变量后就不会报错了

实验2.1 栈的顺序表示和实现

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

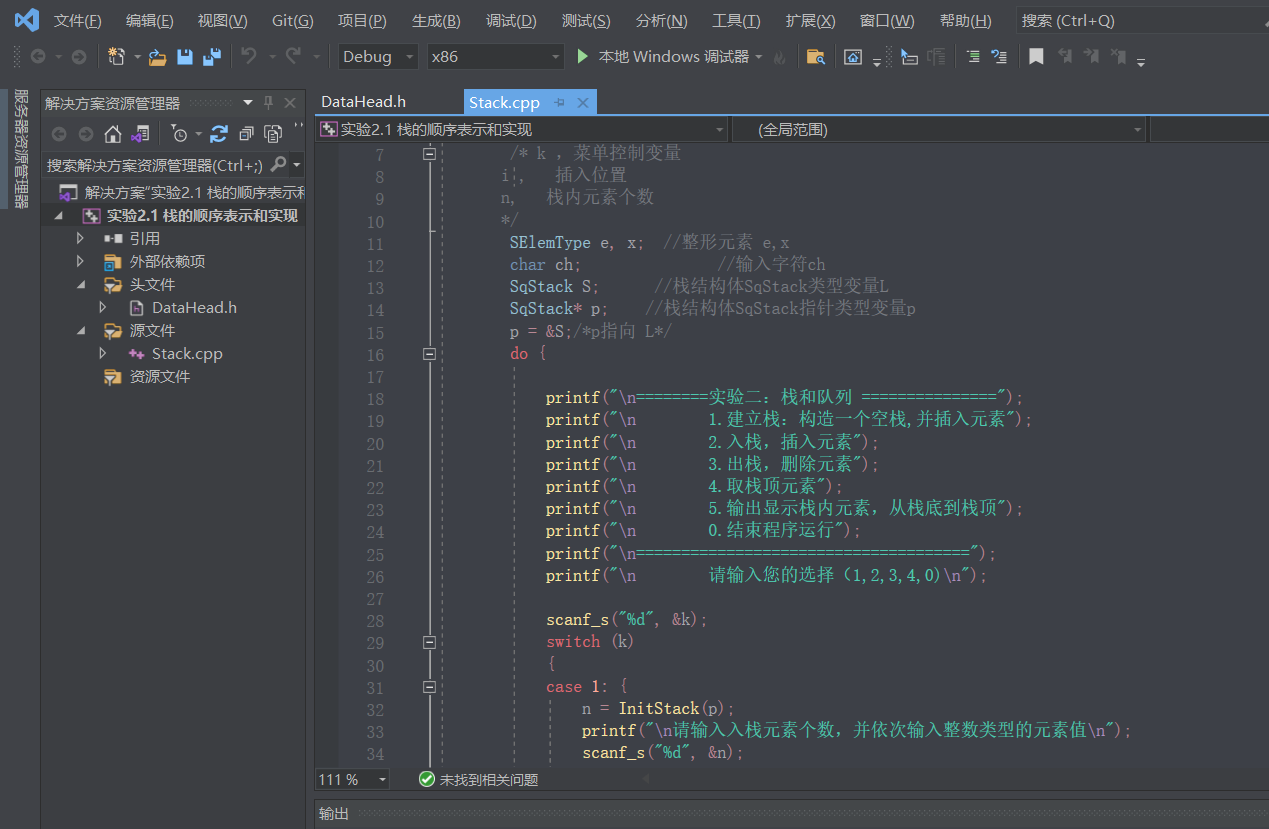
本次实验包括以下主要任务：

* 定义栈的顺序存储数据结构
* 实现栈的插入、删除、查找和输出等操作
* 创建菜单，进行用户界面操作

**程序的结构**：

主要包括以下函数：

| **函数名称** | **功能描述** | **参数说明** |
| --- | --- | --- |
| InitStack | 构建一个空栈 | SqStack\* L：指向栈的指针 |
| GetTop | 返回栈顶元素 | SqStack S：栈对象， SElemType\* e：指向栈顶元素的指针 |
| Push | 将元素入栈 | SqStack\* L：指向栈的指针， SElemType e：要入栈的元素 |
| Pop | 将元素出栈 | SqStack\* L：指向栈的指针， SElemType\* e：指向出栈元素的指针 |
| Out\_Stack | 从栈底到栈顶输出栈内元素 | SqStack L：栈对象 |

**程序截图（部分）**

**数据结构：**

* LNode：定义了链表节点的结构，包括数据域data和指针域next，用于存储结点元素值和指向下一个节点的指针。
* ElemType 是链表中元素的类型，它被定义为 int 类型，可以根据需要修改为其他类型。
* LinkList 是指向链表节点的指针类型，用于表示链表的头指针。

**输入/输出设计：**

* 程序根据用户输入的选项执行相应的操作，并输出结果。
* Push()函数中通过用户输入将元素压入栈，用户需要依次输入每个元素的值。
* Out\_Stack()函数用于打印栈中的所有元素。

**程序变量说明：**

* i：循环计数变量。
* k：菜单控制变量，用于选择操作。
* n：栈内元素个数。
* e：临时变量，用于存储元素值。
* x：临时变量。
* ch：输入字符。

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证栈的功能：

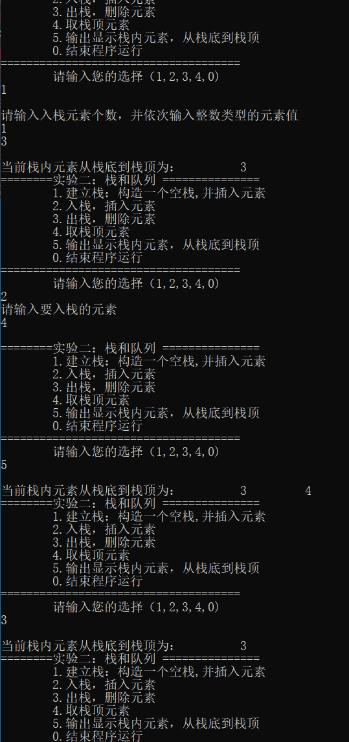
* 创建一个空栈，执行插入操作，查看栈是否正确插入元素。
* 创建一个包含多个元素的栈，执行删除操作，查看栈是否正确删除元素。
* 创建一个栈，执行查找操作，查看是否能正确找到栈顶元素。

**实验结果与分析**

经过测试，程序能够正常运行，并实现了栈的基本功能，包括栈的初始化、入栈、出栈、获取栈顶元素和输出栈内元素等操作。

程序的时间复杂度分析： 栈的基本操作的时间复杂度均为O(1)，即常数时间复杂度，不随输入规模的增加而增加。

程序的空间复杂度分析： 栈的空间复杂度主要取决于栈的存储空间，即数组的大小。在程序中，栈的存储空间会根据需要进行动态扩容，因此空间复杂度为O(n)，其中n为栈的元素个数。

**二:运行输出结果：**

**三:调试和运行程序过程中产生的问题及采取的措施：**

在编写这部分代码时，先去回顾了栈的存储结构，栈是一种后进先出的数据结构。类比现实生活中的火车进站，栈的元素按照插入的顺序排列，但最后插入的元素将最先被访问和移除。也就是火车头最先进来，但是出去的时候却是最晚的，在栈中，新的元素被插入到栈的顶部，而访问和移除操作也发生在栈的顶部。这种顺序确保了最近插入的元素优先处理。

实验2.2栈的链式表示与实现

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

1. 初始化栈
2. 链栈置空
3. 入栈
4. 出栈
5. 取栈顶元素
6. 遍历链栈

**程序的结构**：

主要包括以下函数：

| **InitStack** | **构建一个空栈** | **LinkStack\* S：指向栈的指针** |
| --- | --- | --- |
| ClearStack | 链栈置空 | LinkStack\* S：指向栈的指针 |
| Push | 将元素入栈 | LinkStack\* S：指向栈的指针， SElemType e：要入栈的元素 |
| Pop | 将元素出栈 | LinkStack\* S：指向栈的指针， SElemType\* e：指向出栈元素的指针 |
| GetTop | 返回栈顶元素 | LinkStack\* S：指向栈的指针， SElemType\* e：指向栈顶元素的指针 |
| Out\_Stack | 从栈底到栈顶输出栈内元素 | LinkStack\* S：指向栈的指针 |

**数据结构：**

// 链栈节点结构体

typedef struct StackNode {

SElemType data; // 数据域

struct StackNode\* next; // 指针域，指向下一个节点

} StackNode;

// 链栈结构体

typedef struct {

StackNode\* top; // 指向栈顶节点的指针

int count; // 栈的元素个数

} LinkStack;

**输入/输出设计：**

程序根据用户输入的选择执行相应的操作，包括建立栈、入栈、出栈、取栈顶元素和输出显示栈内元素。

* Push()函数中通过用户输入将元素压入栈，用户需要依次输入每个元素的值。
* Out\_Stack()函数用于打印栈中的所有元素。

**程序变量说明：**

* k：菜单控制变量，用于记录用户选择的操作。
* i：插入位置。
* n：栈内元素个数。
* L：链栈对象。
* e：整型元素。

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证链栈的功能：

* 创建一个空链栈，执行插入操作，查看链栈是否正确插入元素。
* 创建一个包含多个元素的链栈，执行删除操作，查看链栈是否正确删除元素。
* 创建一个链栈，执行查找操作，查看是否能正确找到指定位置的元素。

**实验结果与分析**

程序正确性验证：

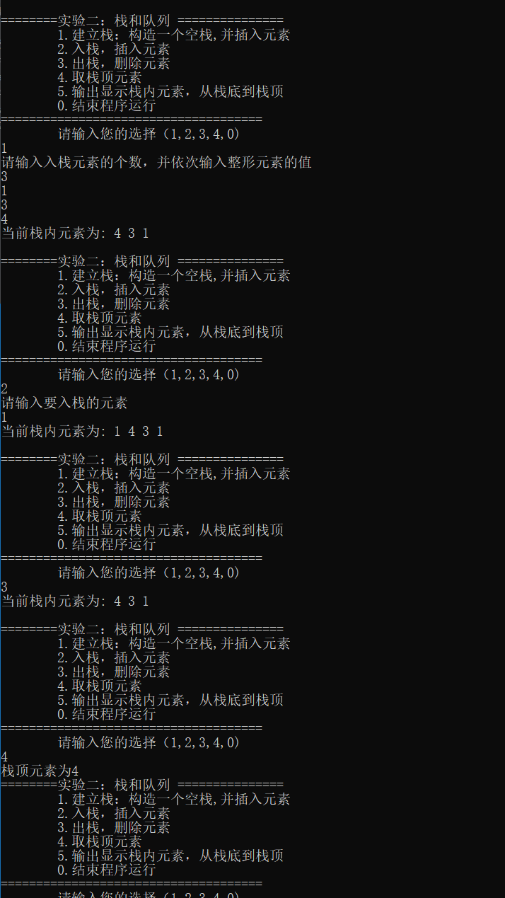
通过多次运行程序，并根 据输入不同的操作选择，验证了程序的正确性。在测试中，创建了空链栈并进行插入操作，确认插入的元素正确地添加到链栈中。同时，创建包含多个元素的链栈，执行删除操作，确保链栈能正确删除元素。还创建了链栈并执行查找操作，验证了能够正确找到指定位置的元素。

程序时间复杂度分析：

栈的链式存储结构中，插入和删除操作的时间复杂度为O(1)，因为它们只涉及到链表的插入和删除操作。取栈顶元素的时间复杂度也为O(1)，只需返回栈顶节点的数据域即可。

程序空间复杂度分析：

空间复杂度主要取决于链栈中节点的个数，即栈的元素个数。假设栈中有n个元素，则空间复杂度为O(n)，因为需要存储每个节点的数据和指针。

**二:运行输出结果：**

**三：调试和运行程序过程中产生的问题及采取的措施：**

本次实验暂无其余问题

实验2.3 队列的顺序表示与实现

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

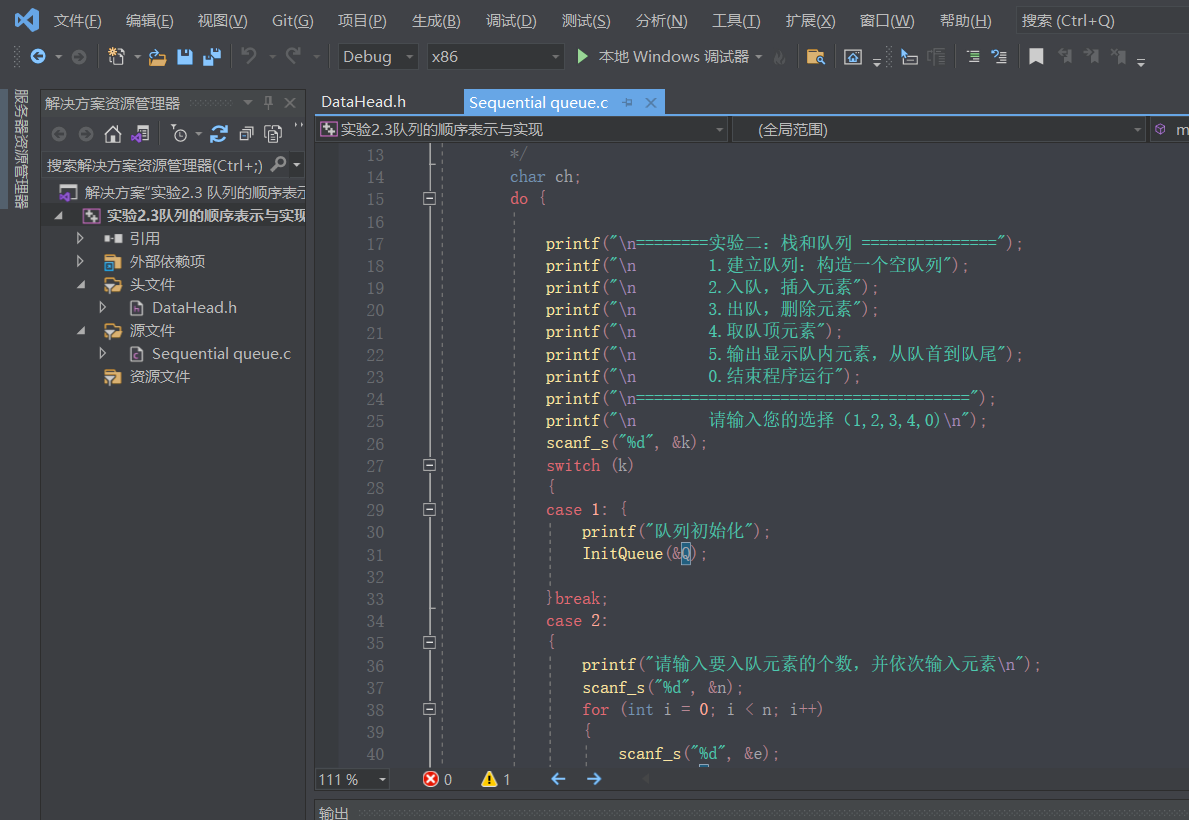
1. 初始化队列
2. 建立顺序队列
3. 入队
4. 出队
5. 判断队列是否为空
6. 取队列头元素
7. 遍历队列

**程序的结构**：

主要包括以下函数：

| **函数名称** | **功能描述** | **参数说明** |
| --- | --- | --- |
| InitQueue | 初始化队列 | SqQueue\* Q |
| QueueEmpty | 判断队列是否为空 | SqQueue Q |
| QueueFull | 判断队列是否已满 | SqQueue Q |
| EnQueue | 入队操作 | SqQueue\* Q, QElemType e |
| DeQueue | 出队操作 | SqQueue\* Q, QElemType\* e |
| GetHead | 取队列头元素 | SqQueue Q |
| TraverseQueue | 遍历队列 | SqQueue Q |

**程序截图（部分）**



**数据结构：**

队列的顺序存储结构由结构体SqQueue定义，包含一个数组存储队列元素，以及队头和队尾的数组下标。

typedef struct {

int data[MAXSIZE]; // 队列中的元素

int front; // 队头下标

int rear; // 队尾下标

} SqQueue;

**输入/输出设计：**

程序通过命令行进行交互式输入和输出。根据用户选择的菜单操作，输入相应的参数并执行相应的操作，然后输出结果。

* EnQueue()函数中通过用户输入将元素压入队列，用户需要依次输入每个元素的值。
* TraverseQueue()函数用于打印队列中的所有元素。

**程序变量说明：**

* Q: 队列的顺序存储结构
* e: 栈和队列中的元素

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证链表的功能：

* 创建一个空队列，执行入队操作，查看队列是否正确插入元素。
* 创建一个非空队列，执行出队操作，查看队列是否正确删除元素。
* 创建一个非空队列，执行取队头元素操作，查看是否能正确取得队头元素。

**实验结果与分析**

程序正确性验证：

经过测试和验证，队列的基本操作均能正常运行，入队、出队、取队头元素等操作符合预期结果，符合队列的定义和规则。

程序时间复杂度分析：

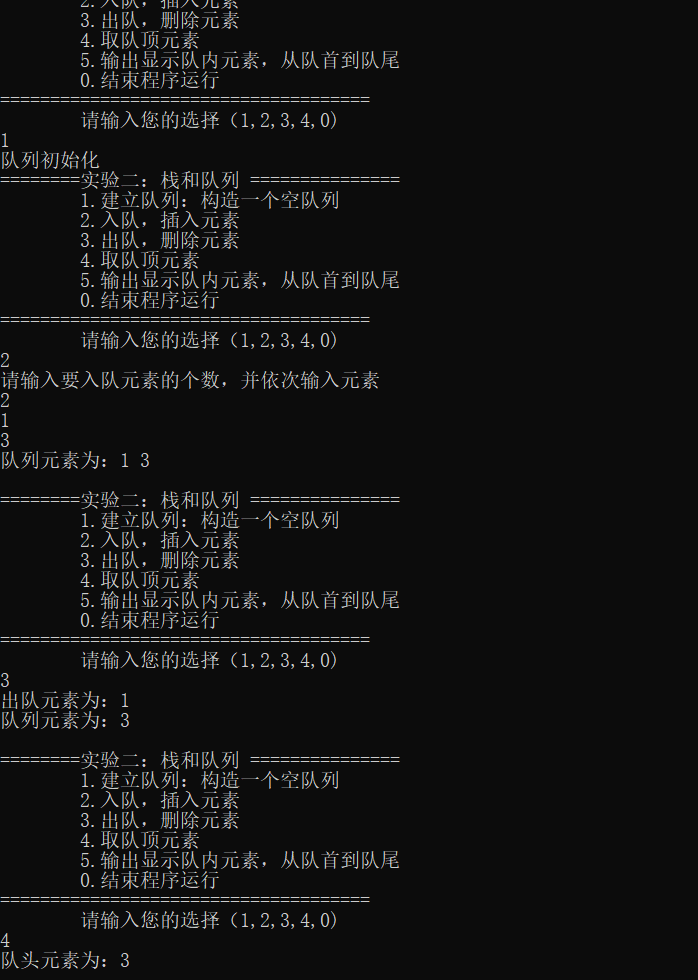
* 初始化队列：O(1)
* 入队：O(1)
* 出队：O(1)
* 取队头元素：O(1)
* 判断队列是否为空：O(1)

程序空间复杂度分析：

栈和队列的空间复杂度取决于存储元素所需的空间和指针变量的空间。

* 队列的空间复杂度：O(n)，其中n为队列中元素的个数。

**二:运行输出结果：**



**三：调试和运行程序过程中产生的问题及采取的措施：**

在编写程序时去回顾了队列的相关知识，队列是一种先进先出的数据结构。类比现实生活中的排队，队列的元素按照插入的顺序排列，最先插入的元素将最先被访问和移除。在队列中，新的元素被插入到队列的末尾，而访问和移除操作发生在队列的前端。这种顺序确保了先进入队列的元素优先处理。在了解后再去编写程序，无其他问题。

实验2.4 队列的链式表示和实现

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

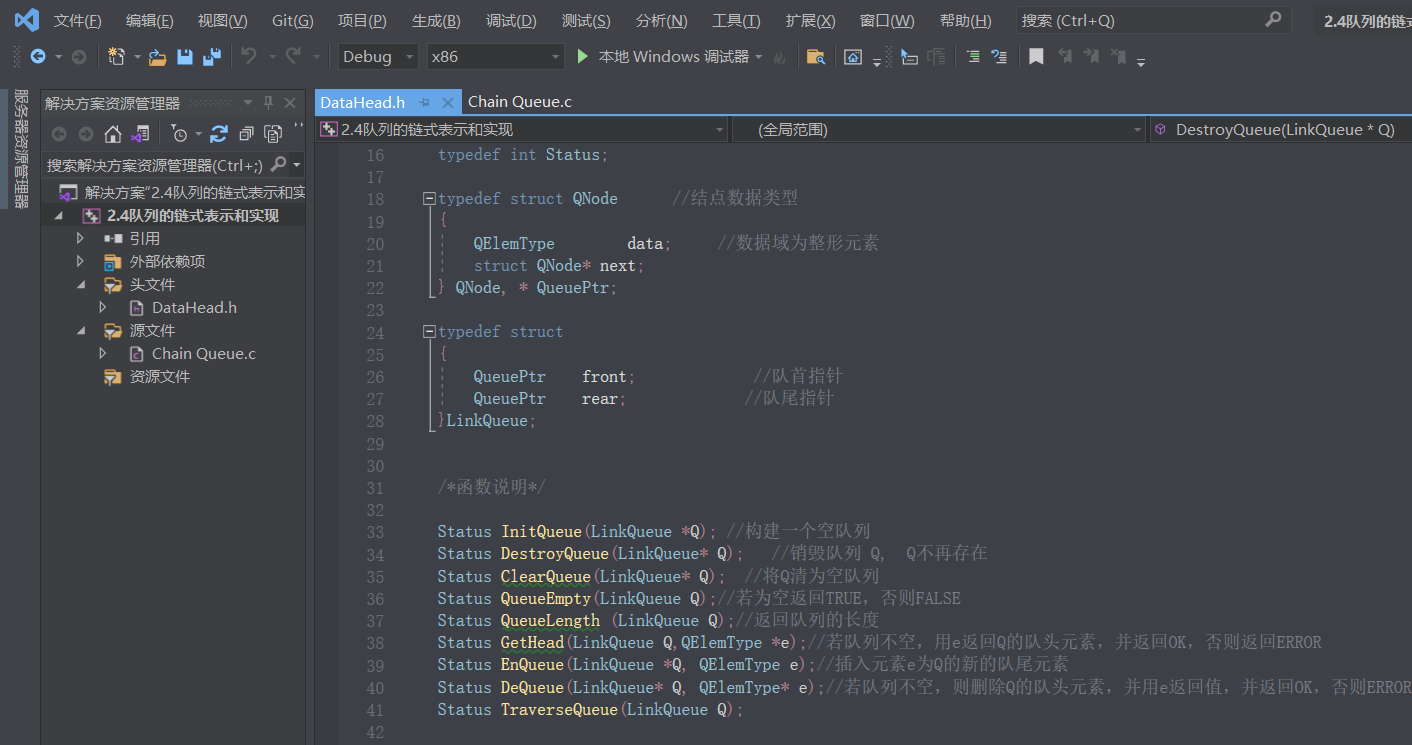
* 初始化并建立链队列
* 入链队列
* 出链队列
* 遍历链队列

**程序的结构**：

主要包括以下函数：

| **InitQueue** | **构建一个空队列** | **LinkQueue\* Q** |
| --- | --- | --- |
| DestroyQueue | 销毁队列 | LinkQueue\* Q |
| ClearQueue | 将队列清空 | LinkQueue\* Q |
| QueueEmpty | 判断队列是否为空 | LinkQueue Q |
| QueueLength | 返回队列的长度 | LinkQueue Q |
| GetHead | 若队列不为空，用e返回队头元素，并返回OK | LinkQueue Q, QElemType\* e |
| EnQueue | 插入元素e为队尾元素 | LinkQueue\* Q, QElemType e |
| DeQueue | 若队列不为空，则删除队头元素，并用e返回值 | LinkQueue\* Q, QElemType\* e |
| TraverseQueue | 遍历输出队列元素 | LinkQueue Q |

**程序截图（部分）**



**数据结构：**

程序使用链队列实现队列功能。链队列使用链表来存储队列元素，具体的数据结构如下:

typedef struct QNode

{

QElemType data;

struct QNode\* next;

} QNode, \* QueuePtr;

typedef struct

{

QueuePtr front; // 队首指针

QueuePtr rear; // 队尾指针

} LinkQueue;

**输入/输出设计：**

程序通过命令行菜单与用户交互，用户可以选择不同的操作来进行队列的初始化、插入元素、删除元素和显示队列元素等操作

* EnQueue()函数中通过用户输入将元素压入队列，用户需要依次输入每个元素的值。
* TraverseQueue()函数用于打印队列中的所有元素。

**程序变量说明：**

* Q：链队列对象，用于存储队列的头指针和尾指针。
* e：队列元素，用于接收出队操作时被删除的队头元素。
* x：临时变量，用于接收用户输入的元素值。
* k：菜单控制变量，用于接收用户选择的操作。
* n：队列内的元素个数。

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证链表的功能：

* 创建一个空链队列，执行插入操作，查看链队列是否正确插入元素。
* 创建一个包含多个元素的链队列，执行删除操作，查看查看链队列是否正确删除元素。
* 创建一个链队列，执行查找操作，查看是否能正确找到指定位置的元素

**实验结果与分析**

**程序正确性验证**

在调试过程中，使用上述测试数据进行验证，确保程序在插入、删除和查找操作中能够正确地执行，并且队列的状态和元素值符合预期。

**程序的时间复杂度分析**

* 初始化队列：O(1)
* 入队操作：O(1)
* 出队操作：O(1)
* 遍历队列：O(n)

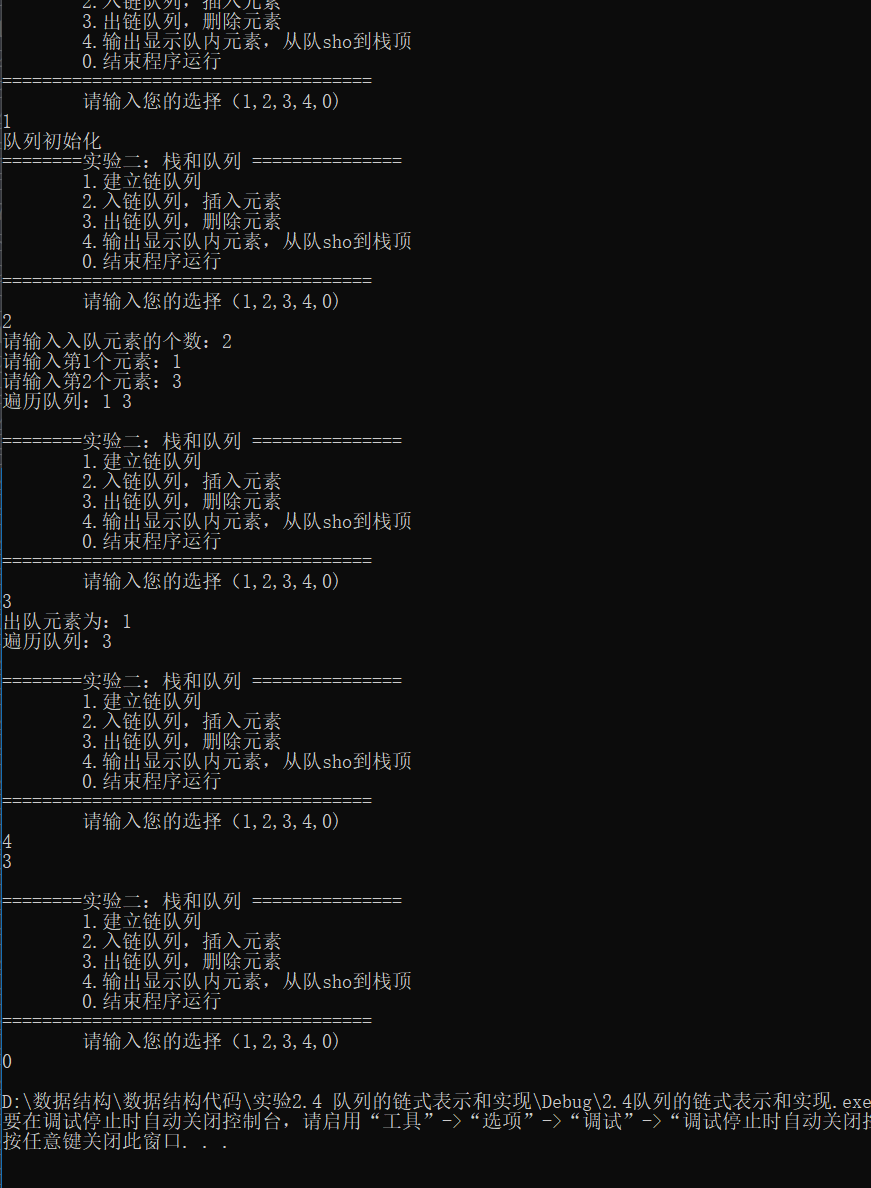
其中，n为队列中的元素个数。

**程序的空间复杂度分析**

* 空间复杂度：O(n)

程序使用链表作为底层数据结构，随着元素的插入和删除，链表的节点会动态地分配和释放内存。因此，程序的空间复杂度与队列中的元素个数成正比。

**二：运行输出结果：**



**三：调试和运行程序过程中产生的问题及采取的措施：**

本次实验暂无其余问题。

实验3 二叉树

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

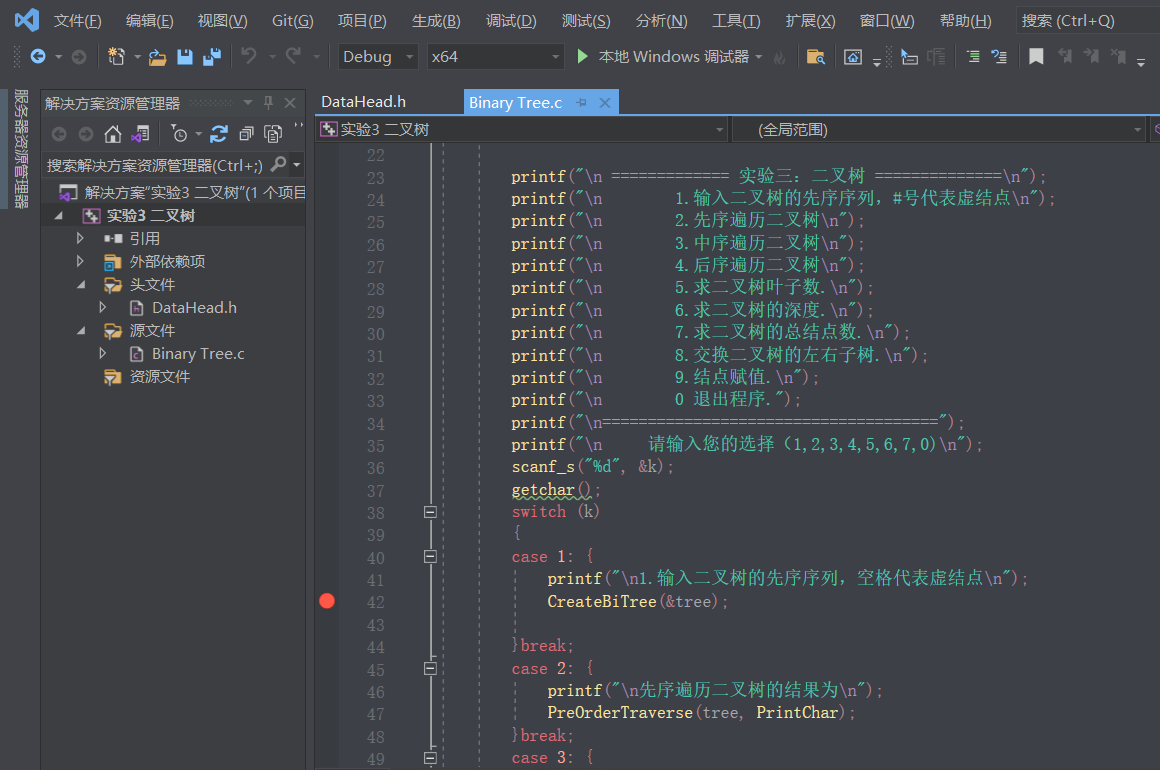
1. 输入二叉树的先序序列字符，建立二叉链表。
2. 按先序、中序和后序遍历二叉树
3. 求二叉树结点总数，观察输出结果。
4. 求二叉树叶子数，观察输出结果。
5. 求二叉树的深度（或称高度），观察输出结果。
6. 交换二叉树的左右子树

在主函数中设计一个简单的菜单，分别调试上述算法**程序的结构**：

主要包括以下函数：

| **函数名称** | **功能描述** | **参数说明** |
| --- | --- | --- |
| CreateBiTree | 构造二叉链表，输入二叉树的先序序列字符 | T: 指向二叉树的指针 |
| PrintChar | Visit函数，用于打印字符 | e: 待打印的字符 |
| PreOrderTraverse | 前序遍历二叉树 | T: 指向二叉树的指针, Visit: 访问函数 |
| InOrderTraverse | 中序遍历二叉树 | T: 指向二叉树的指针 |
| Postorder | 后序遍历二叉树 | T: 指向二叉树的指针 |
| nodes | 计算二叉树的节点总数 | T: 指向二叉树的指针 |
| leafs | 计算二叉树的叶子数 | T: 指向二叉树的指针 |
| swap | 交换二叉树的左右子树 | T: 指向二叉树的指针 |
| depth | 计算二叉树的深度（高度） | T: 指向二叉树的指针 |
| ChangeValue | 结点赋值 | T: 指向二叉树的指针 |

程序截图（部分）



**数据结构：**

* BiTNode：二叉树节点结构体，包含数据域和左右孩子指针。
* BiTree：二叉树指针类型，用于指向二叉树的根节点。

**输入/输出设计：**

程序通过设计一个程序菜单，用户通过键盘输入和屏幕输出进行交互。根据用户选择的操作，程序接收输入并在屏幕上输出相应的结果。

* 输入：通过菜单选项，可以输入二叉树的先序序列字符。
* 输出：根据菜单选项，输出对应的结果，如先序、中序和后序遍历的结果，节点总数、叶子数和深度等。

**程序变量说明：**

* k：菜单控制变量，用于控制菜单选项。
* ch：输入字符，用于接收用户输入的字符。
* bnode：二叉树节点

**程序调试使用的数据说明：**

在调试过程中，可以使用以下数据进行测试和验证链表的功能：

* 创建一个空二叉树，执行插入操作，查看二叉树是否正确插入元素。
* 创建一个包含多个元素的二叉树，执行删除操作，查看二叉树是否正确删除元素。
* 创建一个二叉树，执行查找操作，查看是否能正确找到指定位置的元素。

**实验结果与分析**

程序正确性验证：

在程序中，通过调用CreateBiTree函数创建了一颗二叉树，并对该二叉树进行了前序、中序和后序遍历，计算了二叉树的结点总数和叶子数。另外，还实现了交换二叉树的左右子树功能。

为了验证程序的正确性，可以通过给定的输入数据进行调试，并比较程序输出的结果与预期结果是否一致。例如，可以使用以下数据进行测试：

输入数据（先序序列）：AB#D##C##

预期输出结果：

* 前序遍历结果：ABDC
* 中序遍历结果：BDAC
* 后序遍历结果：DBCA
* 结点总数：4
* 叶子数：2

通过与预期输出结果进行对比，可以验证程序的正确性。

程序时间复杂度分析：

对于二叉树的遍历操作，包括前序、中序和后序遍历，其时间复杂度都为O(n)，其中n为二叉树的结点数。

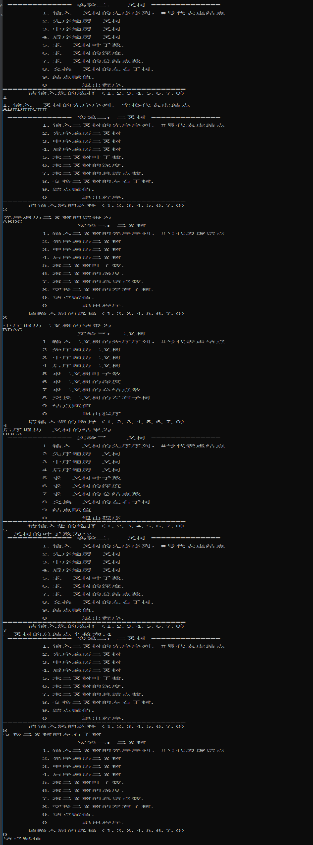
对于计算二叉树的结点总数和叶子数，需要遍历整个二叉树的结点。因此，时间复杂度也为O(n)。

交换二叉树的左右子树的操作是通过指针交换来实现的，时间复杂度为O(1)，与二叉树的规模无关。

综上所述，该程序的时间复杂度主要取决于二叉树的结点数，为O(n)。

程序空间复杂度分析：

主要取决于创建二叉树所需的内存空间和递归调用所占用的栈空间。对于二叉树的创建，空间复杂度为O(n)，其中n为二叉树的节点数量。递归调用所占用的栈空间取决于递归的深度，最坏情况下可能达到O(n)，其中n为二叉树的高度。

**二：运行输出结果：**

**三：调试和运行程序过程中产生的问题及采取的措施：**

本次实验暂无其余问题

实验4 图

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

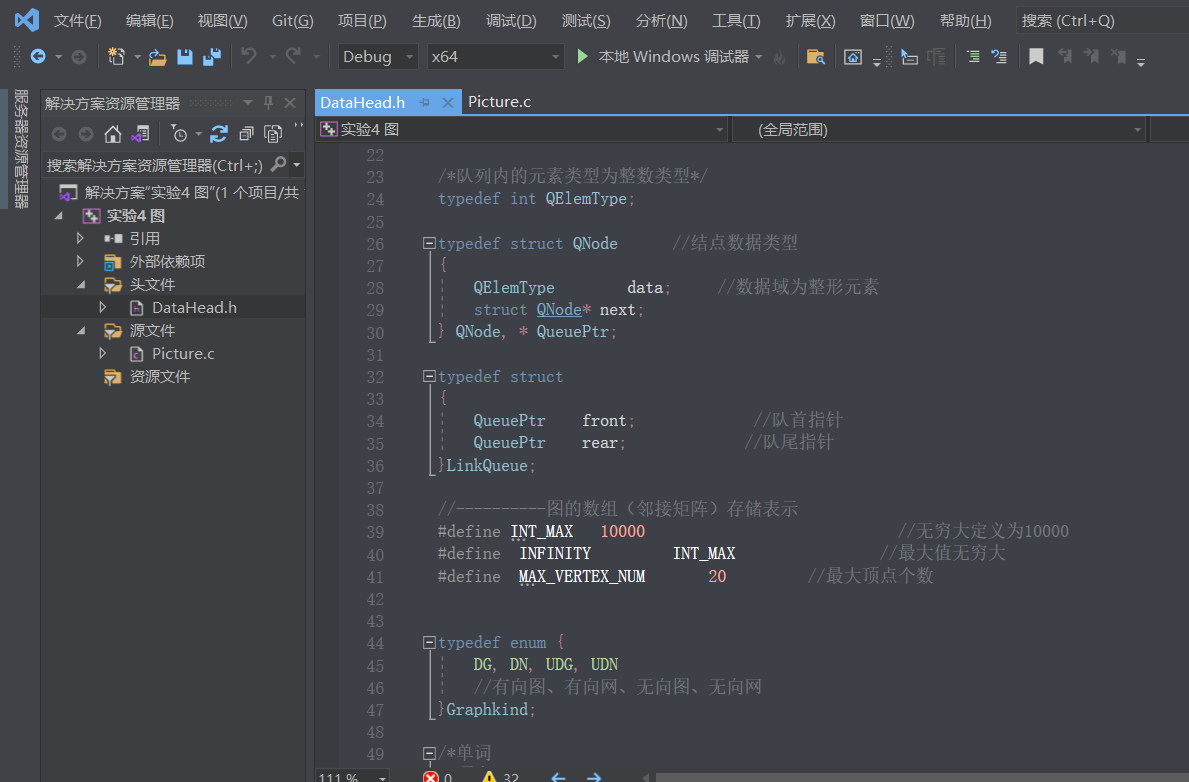
* 建立无向网的邻接矩阵存储并输出，参考程序清单，实现无向图的邻接矩阵表示。
* 建立图的邻接表存储并在此基础上实现图的深度优先遍历和广度优先遍历

**程序的结构**：

主要包括以下函数：

| **函数名称** | **功能描述** | **参数说明** |
| --- | --- | --- |
| CreateGraph | 创建图G | G - 指向MGraph类型的指针 |
| CreateDG | 创建有向图 | G - 指向MGraph类型的指针 |
| CreateDN | 创建有向网 | G - 指向MGraph类型的指针 |
| CreateUDG | 创建无向图 | G - 指向MGraph类型的指针 |
| CreateUDN | 创建无向网 | G - 指向MGraph类型的指针 |
| CreateALGraph | 创建邻接表 | M - 指向MGraph类型的指针, AL - 指向ALGraph类型的指针 |
| OutputMG | 输出邻接矩阵 | G - MGraph类型的变量 |
| OutputAL | 输出邻接表 | AL - ALGraph类型的变量 |
| LocateVex | 确定顶点在矩阵中的位置 | G - MGraph类型的变量, v - 顶点名称 |
| PrintInt | 打印整数 | e - 整数变量 |
| PrintChar | 打印字符 | vex - 字符变量 |
| DFSTraverse | 对图G进行深度优先遍历 | G - ALGraph类型的变量, Visit - 函数指针，用于访问顶点 |

程序截图（部分）



**数据结构：**

* MGraph：图的邻接矩阵存储结构，包括顶点向量、邻接矩阵、顶点数、弧数和图的种类标志。
* ALGraph：图的邻接表存储结构，包括顶点数组、顶点数、弧数和图的种类标志。

**输入/输出设计：**

程序通过设计一个程序菜单，用户通过键盘输入和屏幕输出进行交互。根据用户选择的操作，程序接收输入并在屏幕上输出相应的结果。

 输入：通过函数参数和用户交互输入图的相关信息。

 输出：将图的邻接矩阵和邻接表输出到屏幕

**程序变量说明：**

* G: MGraph结构体指针，表示图G。
* M: MGraph结构体指针，表示图G的邻接矩阵存储结构。
* AL: ALGraph结构体指针，表示图G的邻接表存储结构。

**程序调试使用的数据说明：**

**实验结果与分析**

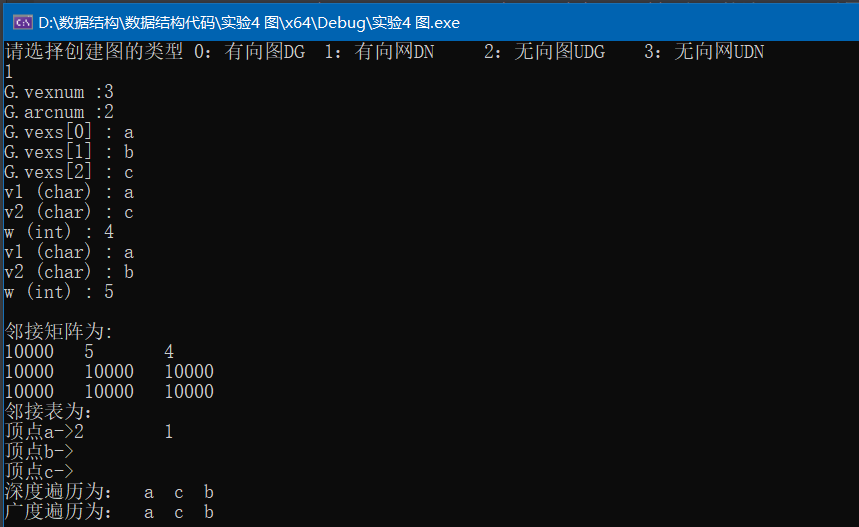
程序正确性验证：

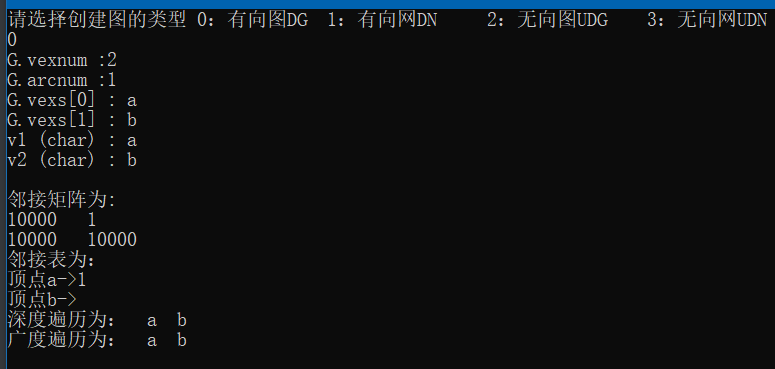
通过测试，程序能够正确创建图的邻接矩阵和邻接表，并输出正确的结果。程序时间复杂度分析：

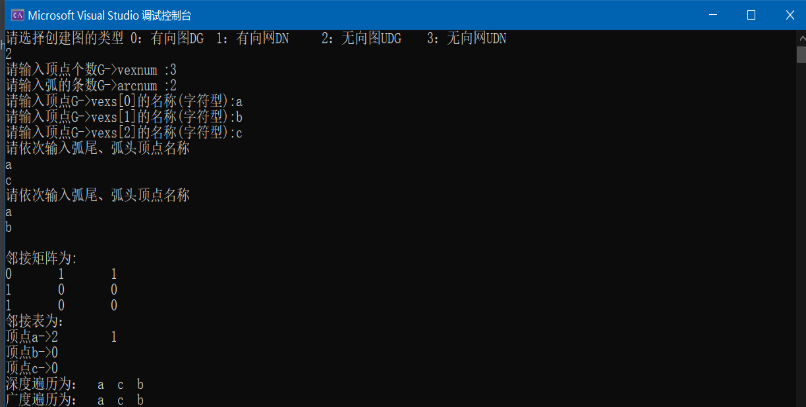
程序空间复杂度分析：

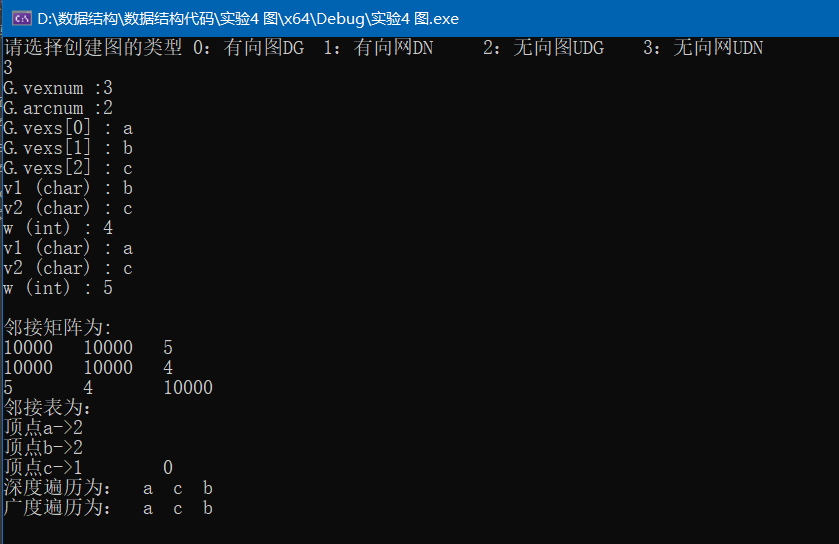
创建图的邻接矩阵和邻接表的时间复杂度为O(V^2)，其中V为顶点数。遍历图的时间复杂度为O(V+E)，其中E为边数。

**二:运行输出结果：**

1.有向图 2.有向网



3.无向图 4.无向网



**三：调试和运行程序过程中产生的问题及采取的措施：**

在做本次实验时，对于邻接表和邻接矩阵，无向图，有向图的概念有些混淆，不知道代码运行的逻辑是什么，所以去刷了浙江大学陈越教授的数据结构视频，在理解之后去编写了此次代码，后续只出现了一些小问题，就是在涉及到输出函数的时候内存缓冲的问题，重新设计好了输入语法后问题解决。

实验5 查找

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

* 用二叉链表做存储结构，输入键值序列，利用查找、插入算法建立一棵二叉排序树。
* 按中序遍历这棵二叉排序树。
* 在二叉排序树上插入结点。
* 删除二叉排序树上的结点。

**程序的结构**：

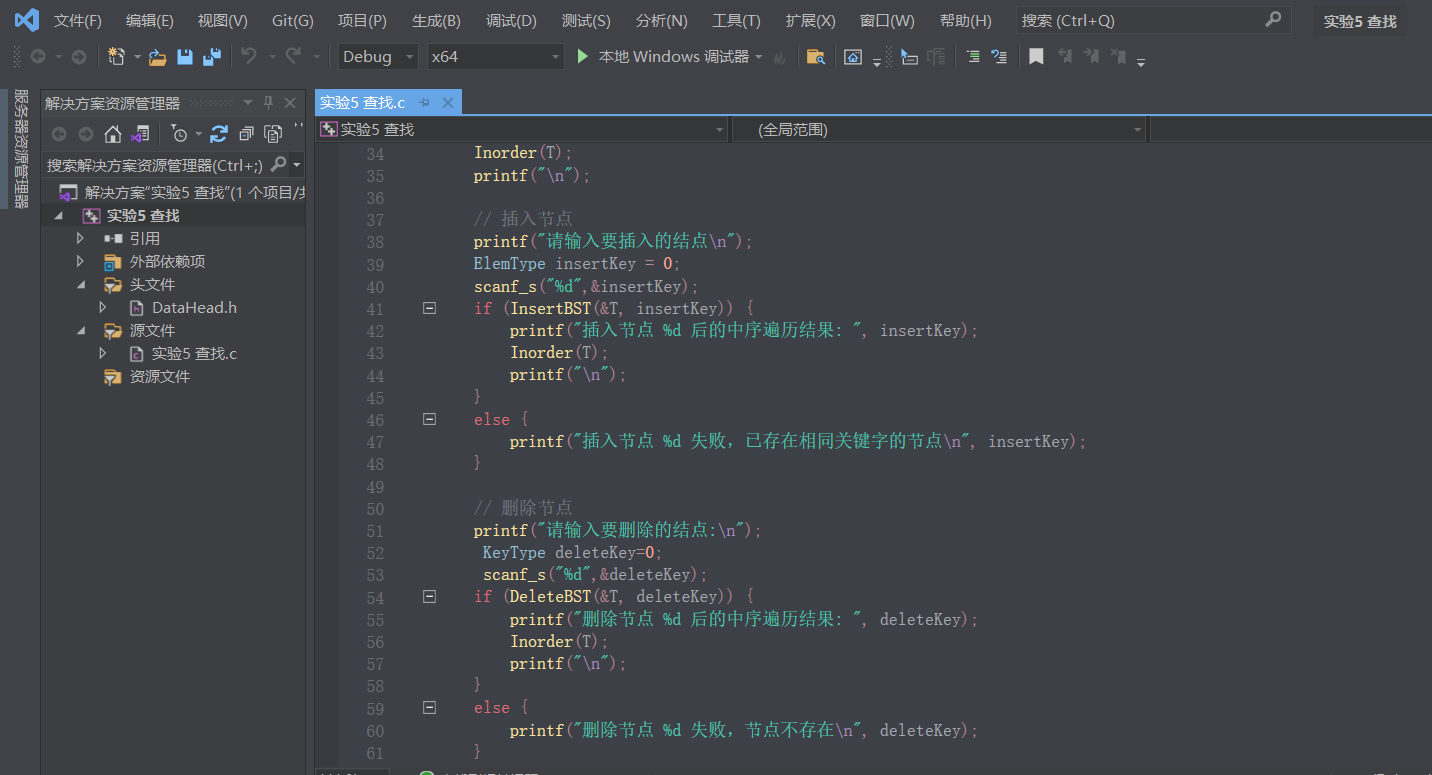
主要包括以下函数：

| **函数名称** | **功能描述** | **参数说明** |
| --- | --- | --- |
| **Inorder** | **中序遍历二叉树** | **T:** **二叉树的根节点** |
| **SearchBST** | **在二叉排序树中查找元素** | **T:** **二叉排序树的根节点，key: 关键字，f: 指向T的双亲节点的指针，p: 指向查找到的节点的指针** |
| **InsertBST** | **在二叉排序树中插入节点** | **T:** **指向二叉排序树的根节点的指针，e: 要插入的节点的值** |
| **DeleteBST** | **在二叉排序树中删除节点** | **T:** **指向二叉排序树的根节点的指针，key: 要删除的节点的关键字** |
| **Delete** | **删除二叉排序树中的节点** | **p:** **指向要删除的节点的指针** |
| **EQ** | **判断两个关键字是否相等** | **key1:** **关键字1，key2: 关键字2** |
| **LT** | **判断关键字1是否小于关键字2** | **key1:** **关键字1，key2: 关键字2** |

**数据结构：**

* BiTNode：二叉树节点结构体，包括数据域data和左右孩子指针lchild、rchild。
* BiTree：二叉树指针，指向二叉树的根节点。

程序截图（部分）



**输入/输出设计：**

* 输入：通过用户交互输入二叉排序树的元素个数和每个元素的值。
* 输出：输出中序遍历结果以及插入和删除节点后的中序遍历结果。

**程序变量说明：**

* T: BiTree指针，表示二叉排序树的根节点。
* n: 整型变量，表示二叉排序树的元素个数。
* keys: 整型数组，用于存储输入的键值序列。
* insertKey: 整型变量，表示要插入的节点的值。
* deleteKey: 整型变量，表示要删除的节点的关键字。

**程序调试使用的数据说明：**

**实验结果与分析**

程序正确性验证：

通过测试，程序能够正确建立二叉排序树，并按要求输出中序遍历结果，插入节点和删除节点后的中序遍历结果。

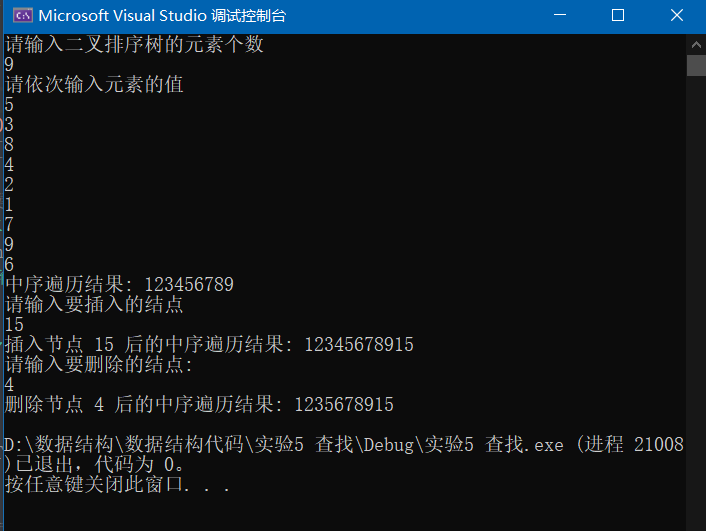
程序时间复杂度分析：

* 查找操作（SearchBST）：在二叉排序树中递归地查找关键字等于key的数据元素。最坏情况下，二叉排序树为一条链表，需要遍历所有节点才能找到目标节点，时间复杂度为O(n)。平均情况下，树的高度为log(n)，时间复杂度为O(log n)，其中n为二叉排序树的节点数。
* 插入操作（InsertBST）：在二叉排序树中插入节点。插入操作需要先进行查找，查找的时间复杂度为O(log n)，然后在合适的位置插入新节点，时间复杂度为O(1)。因此，插入操作的时间复杂度为O(log n)。
* 删除操作（DeleteBST）：在二叉排序树中删除节点。删除操作需要先进行查找，查找的时间复杂度为O(log n)，然后根据不同情况进行节点的删除操作。删除操作涉及重接子树和释放内存，时间复杂度为O(1)。因此，删除操作的时间复杂度为O(log n)。

程序空间复杂度分析：

程序变量和数据结构的空间复杂度：

* BiTNode和BiTree结构体：这两个结构体用于表示二叉排序树的节点和指针，它们的空间复杂度为O(1)，因为它们只包含固定数量的数据成员。
* TElemType、KeyType和ElemType：这些数据类型用于存储二叉排序树节点的数据值和关键字，根据实际情况而定，假设每个节点的数据值和关键字占用常数空间，则它们的空间复杂度为O(1)。
* keys数组：用于存储输入的键值序列，根据输入的元素个数而定，假设输入的元素个数为n，则keys数组的空间复杂度为O(n)。

**二:运行输出结果：**

**三：调试和运行程序过程中产生的问题及采取的措施：**

本次实验暂无其他问题

实验6 矢量数据结构

**一、程序设计的基本思想，原理和算法描述：**

**实验内容**

本次实验包括以下主要任务：

1. 认真阅读和掌握本实验中超图对象的使用语法。
2. 读取所给的gps航迹点文件内的坐标（文本文件格式）
3. **添加几何对象：** 航迹点、航迹线、部分航迹围成的面
4. **修改几何对象：**搜索特定点、线、面，对其坐标进行局部加减，生成新的对象替代原对象，添加到地图中显示
5. **修改添加对象的属性值**
6. 保存和打印出程序的运行结果

**输入/输出设计：**

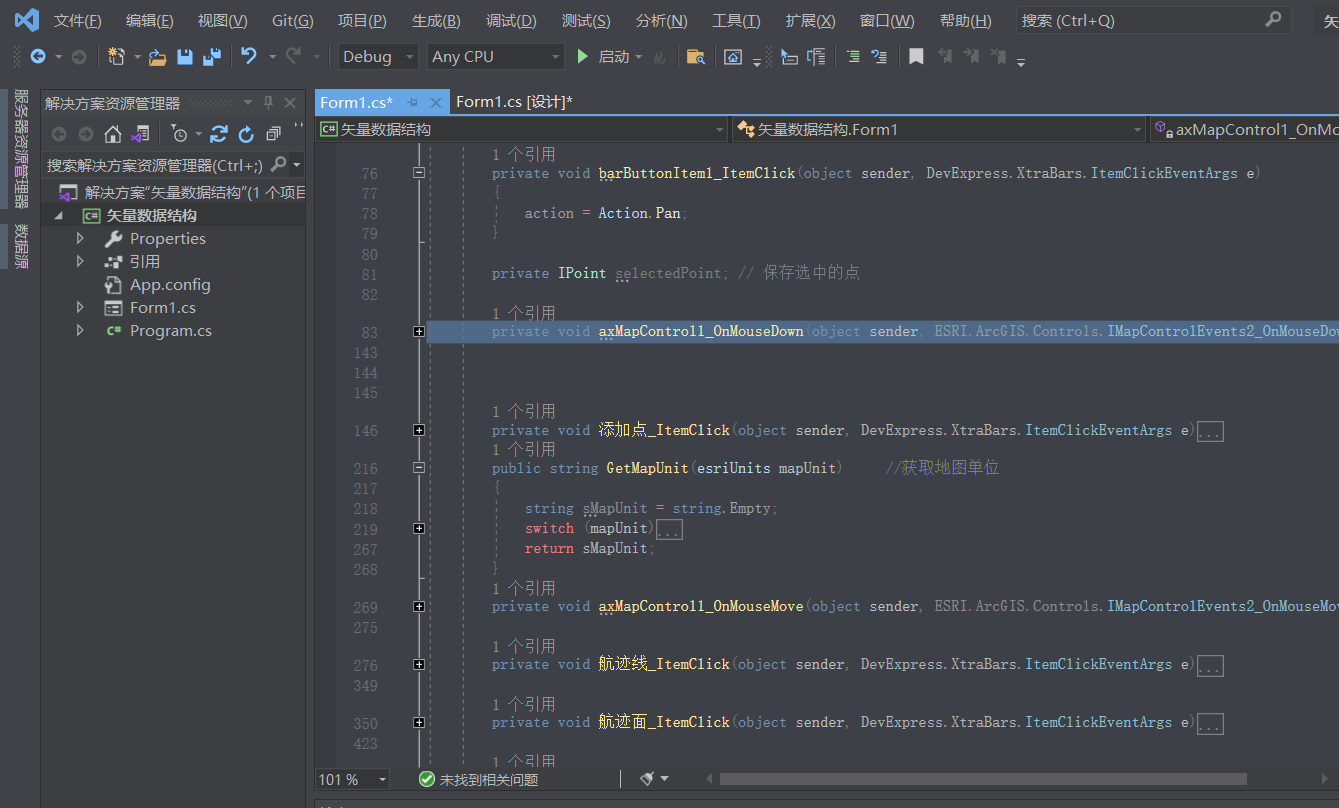
输入数据为矢量底图数据以及航迹数据（2007去莫干山.txt）

**程序变量说明：**

* sMapUnits:地图单位，用于底图坐标状态栏的显示。
* Action:枚举类型变量，用于存储用户点击菜单栏时具体是做哪一种操作
* mapControl：当前的底图地图控件，用于显示地图

**程序UI界面设计:**

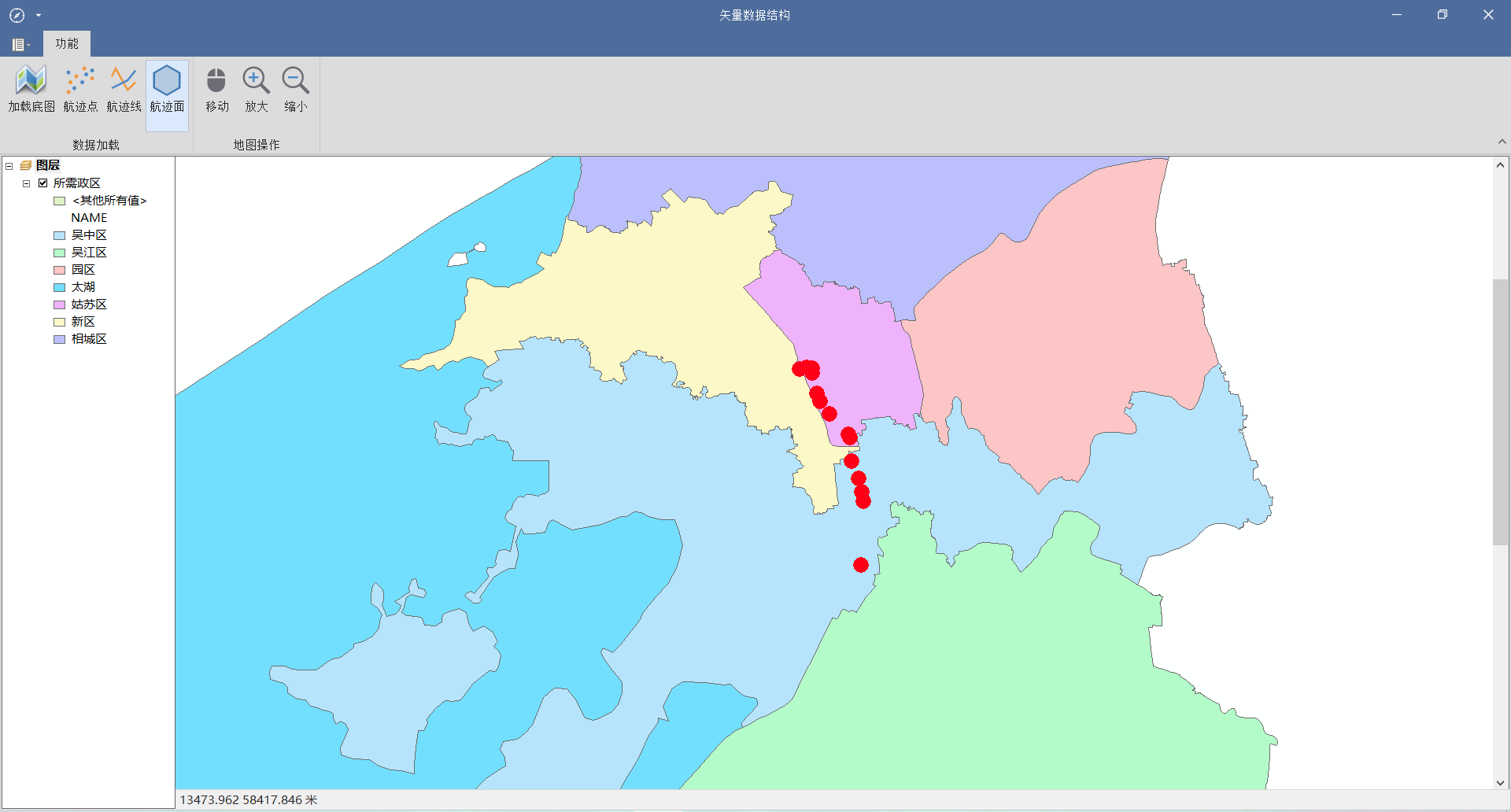
**程序截图（部分）**

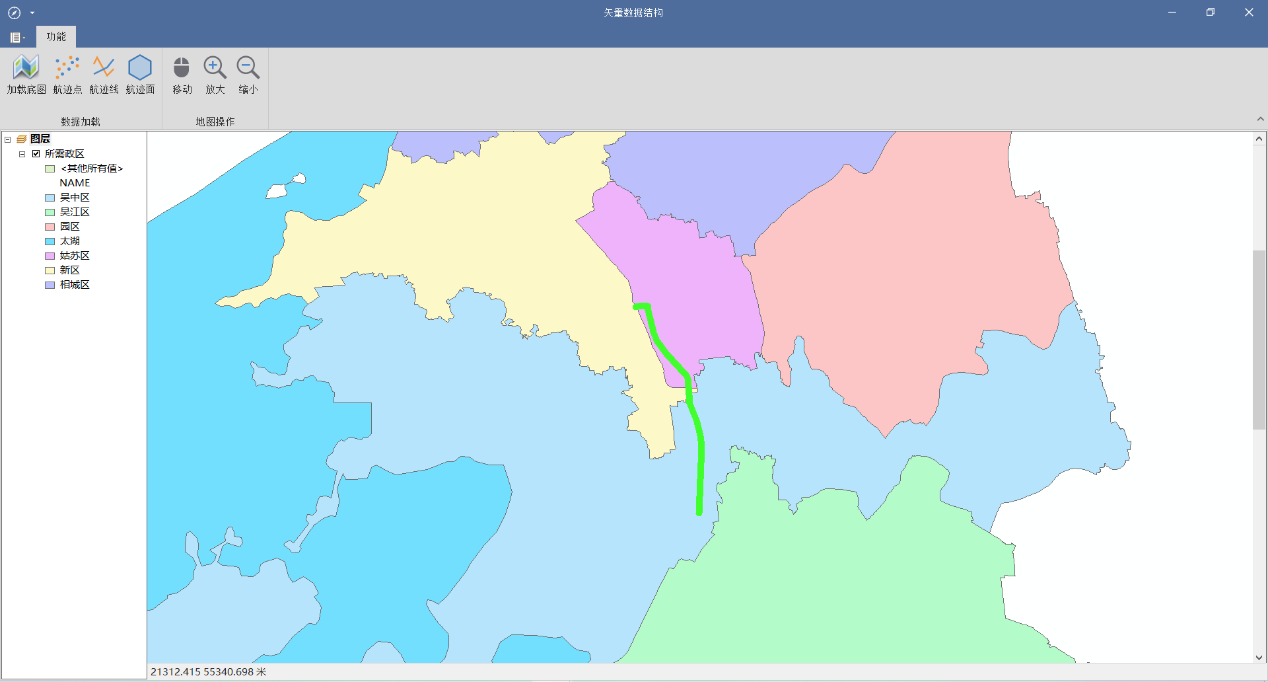


**程序调试使用的数据说明：**

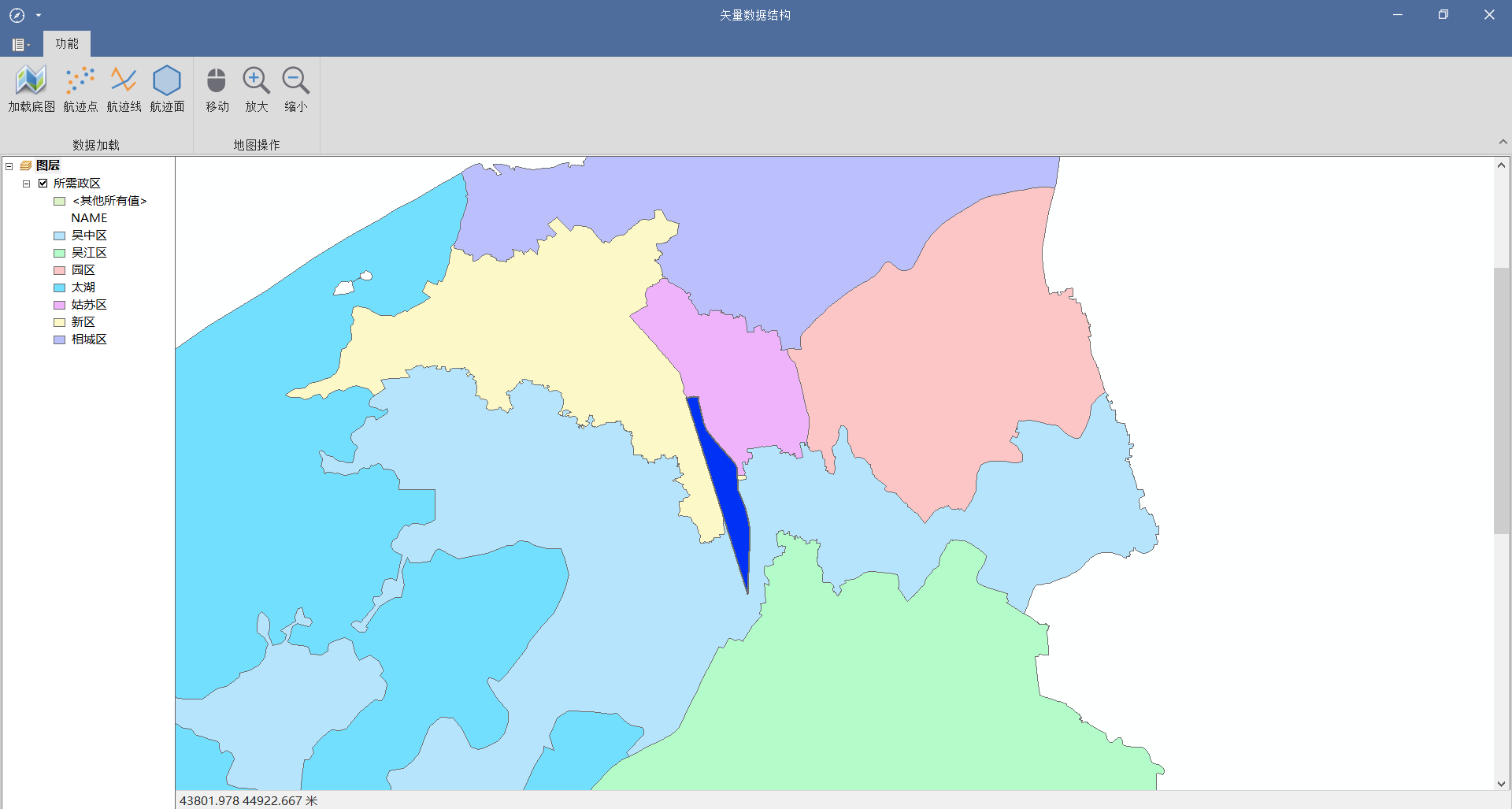
调试数据是2007去莫干山.txt，对原始数据进行了处理，去除了一些不必要内容，保留了点的经纬度坐标。

**二:运行输出结果：**

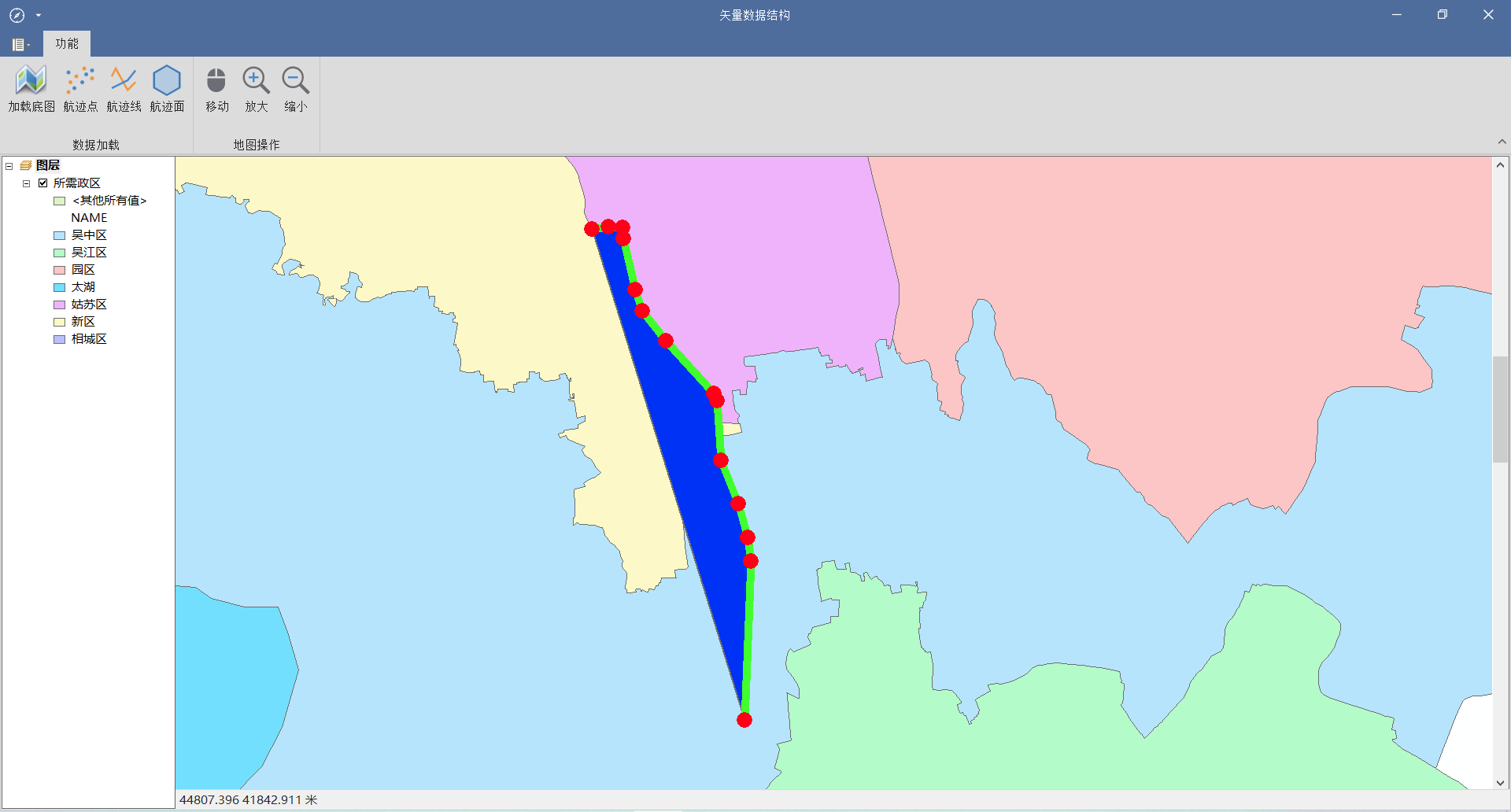


图一:添加航迹点

图二：添加航迹线

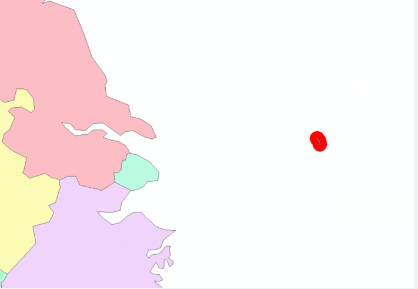


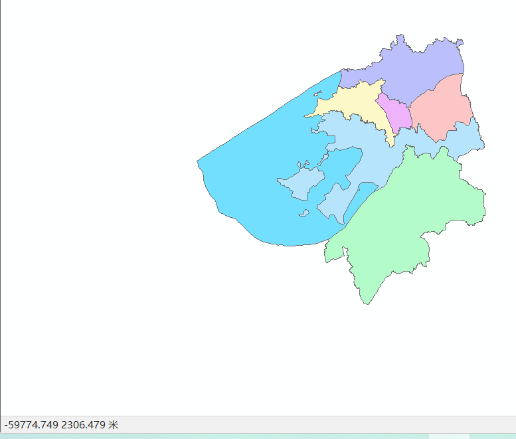
**图三:添加航迹面**

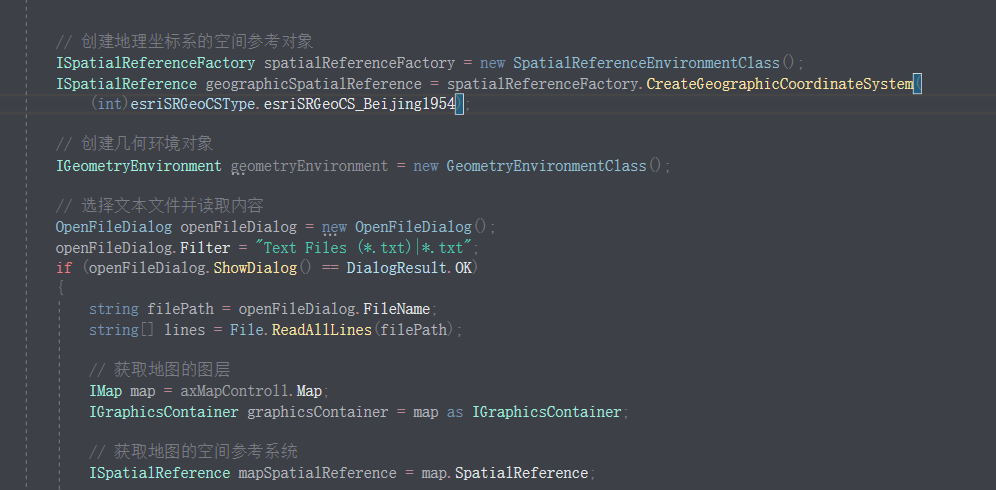


**图四：叠加显示**

**三:调试和运行程序过程中产生的问题及采取的措施：**

刚开始进行添加航迹点显示的时候，发现自己的添加的点显示到中国地图的位置不正确，如下图所示:

所以我想可能是坐标系的问题导致显示的位置有偏差，所以给地图添加了一个状态栏，然后底图数据改成了苏州的底图，想看看地图显示的是地理坐标还是投影坐标，如下图:

由图可以发现地图控件显示的坐标是投影坐标，而我们读取的2007去莫干山.txt是经纬度坐标，所以我在代码中进行了坐标的空间转换，如下图:

经过坐标转换后，成功将点线面添加到了地图上，问题解决。

**四：总结**

空间数据结构实验报告自此全部结束，感谢史老师一学期以来的辛劳的教学，我谨代表地信2112班向你表示衷心的感谢与祝福，希望老师未来身体健康，万事胜意！