

实验 2 抽象数据类型的实现

一、 实验目的

1. 了解抽象数据类型 (ADT) 的基本概念及描述方法;
2. 掌握抽象数据类型 (ADT) 的实现方法;
3. 学会使用 VC6.0 建立工程 (project) 来组织程序。

二、 预备知识

1. C 语言中数组、函数、结构体、指针的使用方法。
2. C 语言中动态内存分配和释放方法 (malloc 和 free 库函数的使用)。
3. VC6.0 集成开发环境使用方法, 尤其是工程使用和程序调试方法。
4. 复数的基本知识及四则运算法则:

设 $z1=a+bi$, $z2=c+di$, ($a, b, c, d \in R$)

加减法: $z1 \pm z2 = (a \pm c) + (b \pm d)i$

乘法: $z1 * z2 = (ac - bd) + (ad + bc)i$

三、 实例——三元组抽象数据类型实现

1. 三元组抽象数据类型的定义

ADT Triplet

{

数据对象: $D=\{e1, e2, e3 | e1, e2, e3 \in \text{ElemSet (定义了关系运算的某个集合)}\}$

数据关系: $R1 = \{ \langle e1, e2 \rangle, \langle e2, e3 \rangle \}$

基本操作:

InitTriplet(&T, v1, v2, v3);

操作结果: 构造了三元组 T, 元素 e1, e2 和 e3 分别被赋以参数 v1, v2 和 v3。

DestroyTriplet(&T);

操作结果: 三元组 T 被销毁。

Get(T, i, &e);

初始条件: 三元组 T 已存在, $1 \leq i \leq 3$;

操作结果: 用 e 返回 T 的第 i 元的值。

Put(&T, i, e);

初始条件: 三元组 T 已存在, $1 \leq i \leq 3$;

操作结果: 修改 T 的第 i 元的值为 e。

IsAscending(T);

初始条件：三元组 T 已存在；

操作结果：如果 T 的三个元素按升序排列，则返回 1，否则返回 0。

IsDescending(T);

初始条件：三元组 T 已存在；

操作结果：如果 T 的三个元素按降序排列，则返回 1，否则返回 0。

Max(T, &e);

初始条件：三元组 T 已存在；

操作结果：用 e 返回 T 的三个元素中的最大值。

Min(T, &e);

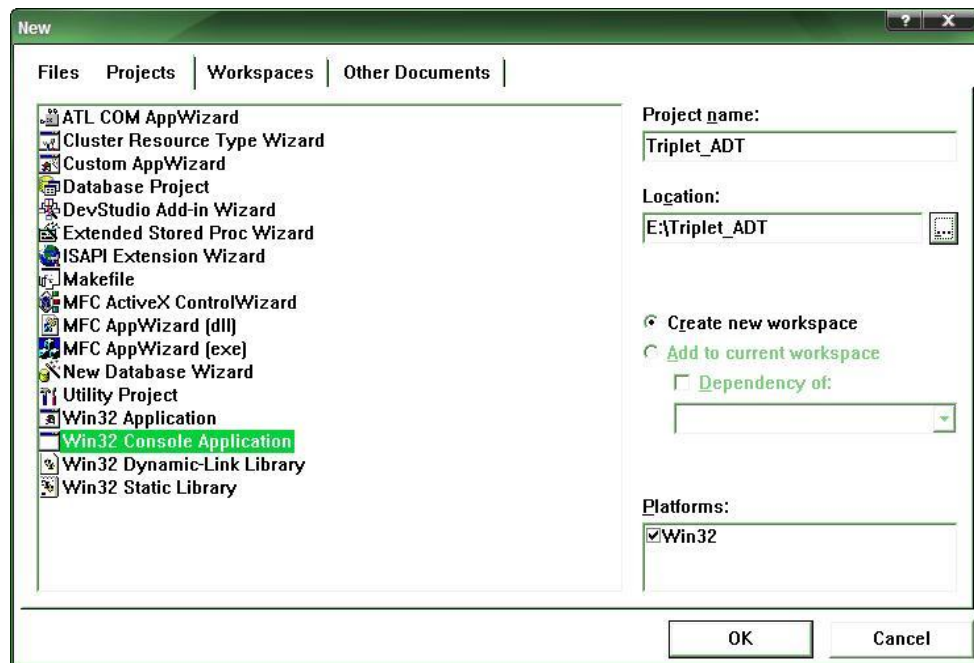
初始条件：三元组 T 已存在；

操作结果：用 e 返回 T 的三个元素中的最小值。

} ADT Triplet

2. 用 C 语言类型声明定义三元组类型和基本操作函数声明

①创建工程：Triplet_ADT，如下图 2-1 所示：



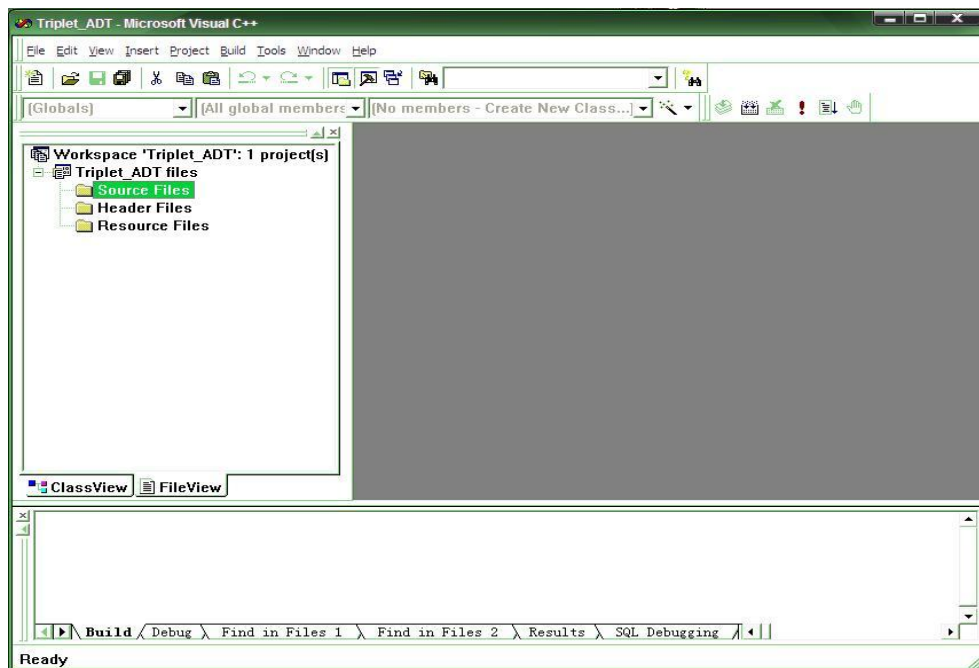
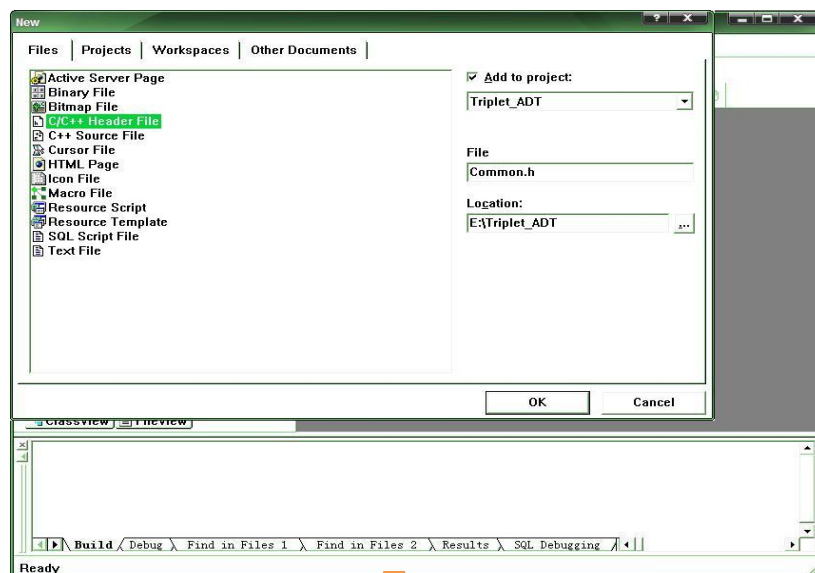


图 2-1 创建工程 Triplet_ADT

②为该工程创建公共头文件 Common.h，如图 2-2 所示：



```

/*Common.h*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2

typedef int Status;
typedef int Boolean;

```

- 包含所需要的标准头文件；
- 定义所需要的符号常量；
- 为已有类型定义所需要的新类型名。
- 不需要的头文件、符号常量等不必出现。

图 2-2 创建头文件 Common.h

③创建头文件 Trilet.h, 对用来实现基本操作的函数进行声明, 如图 2-3 所示:

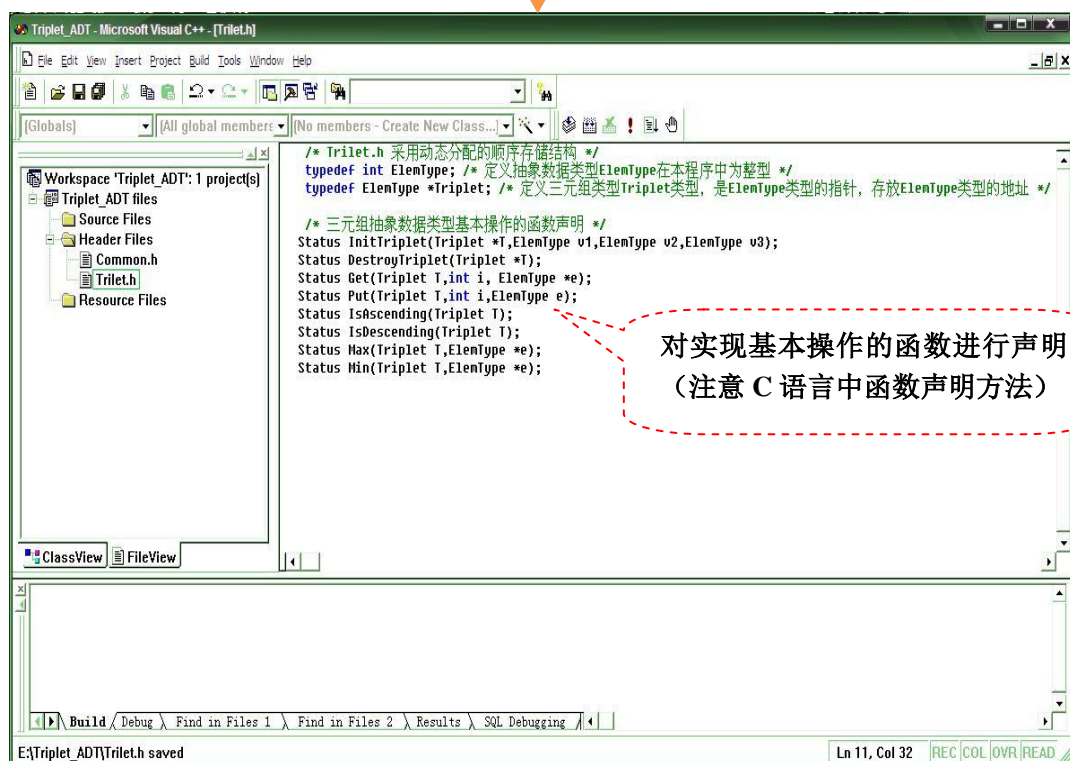
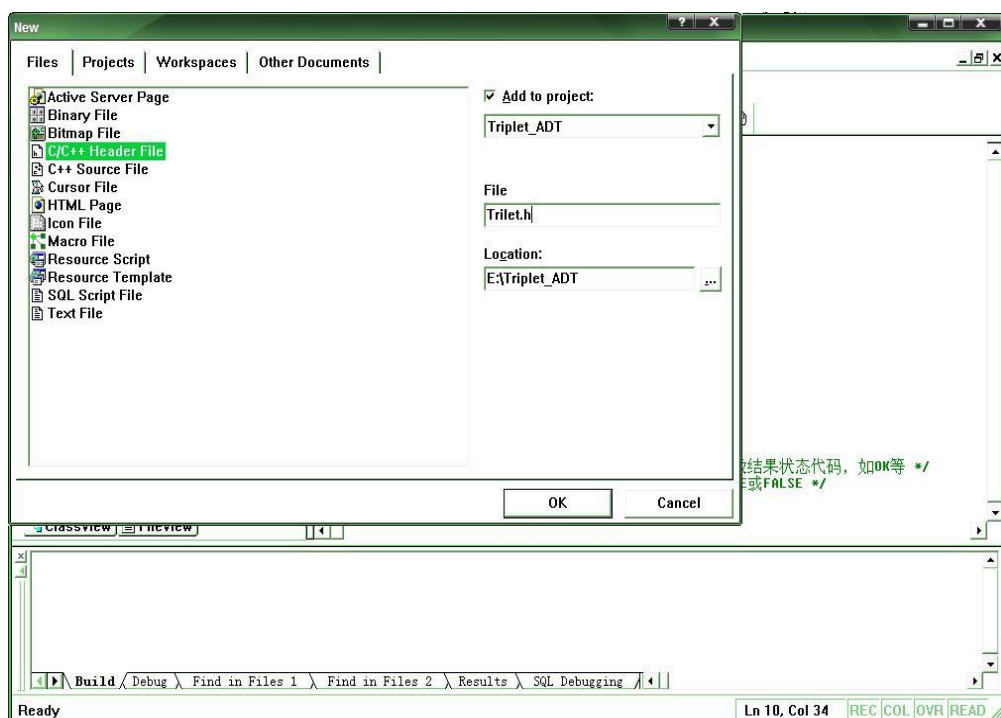


图 2-3 创建头文件 Trilet.h

3. 用 C 语言实现三元组类型的基本操作

创建源文件 Trilet.c，如图 2-4 所示。

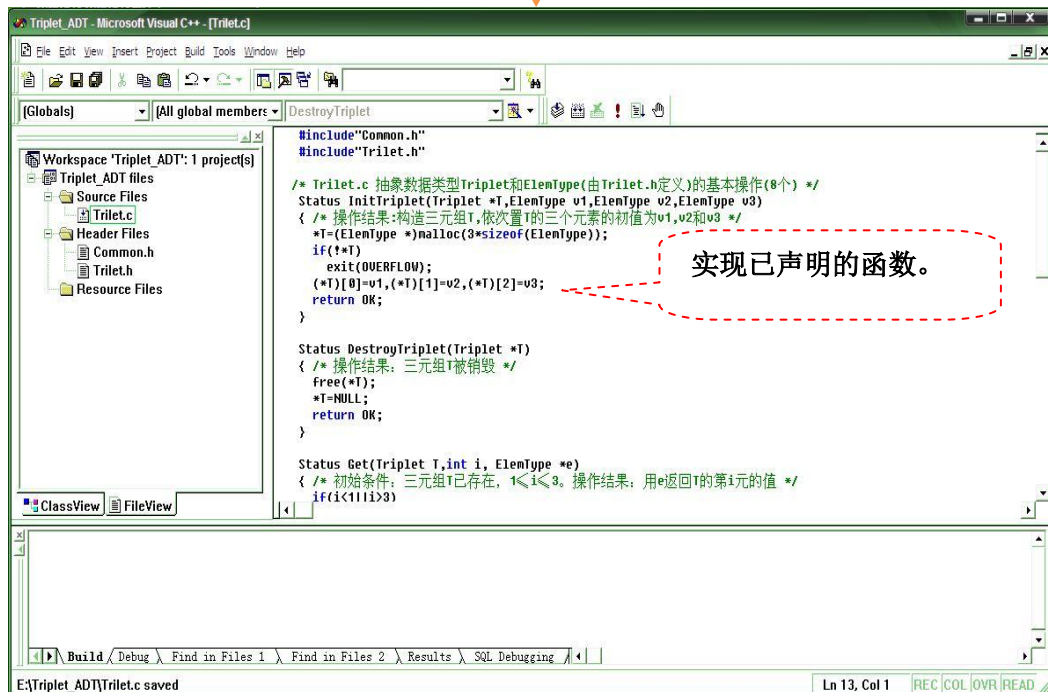
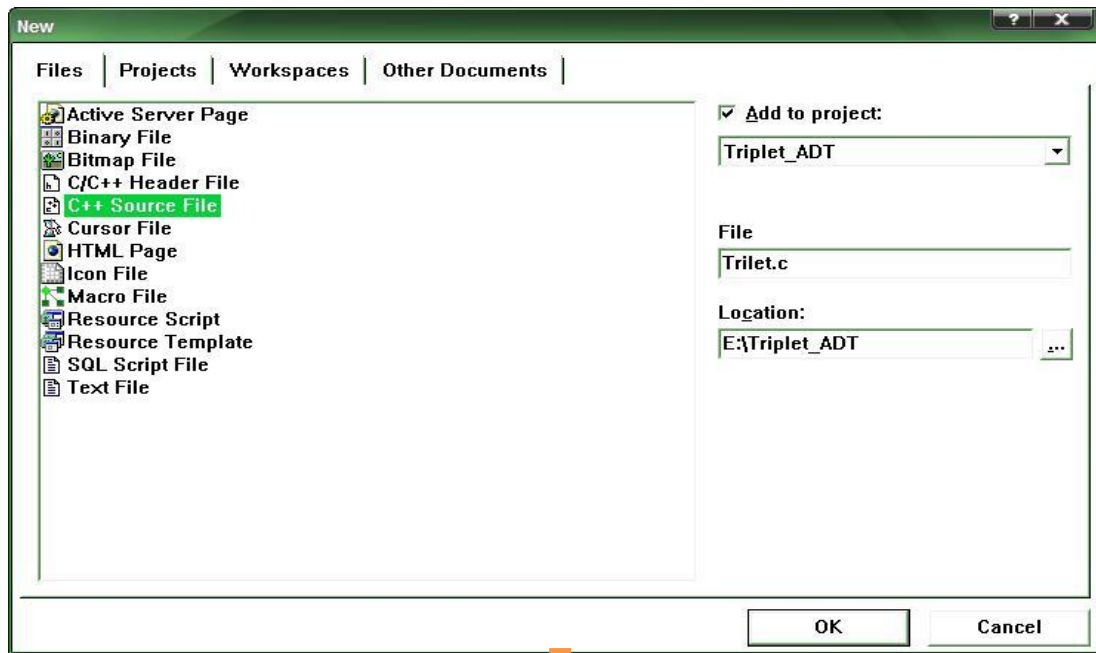


图 2-4 创建源文件 Trilet.c

4. 编写应用程序测试上述三元组类型

创建源文件 TriletTestApp.c，如图 2-5 所示。

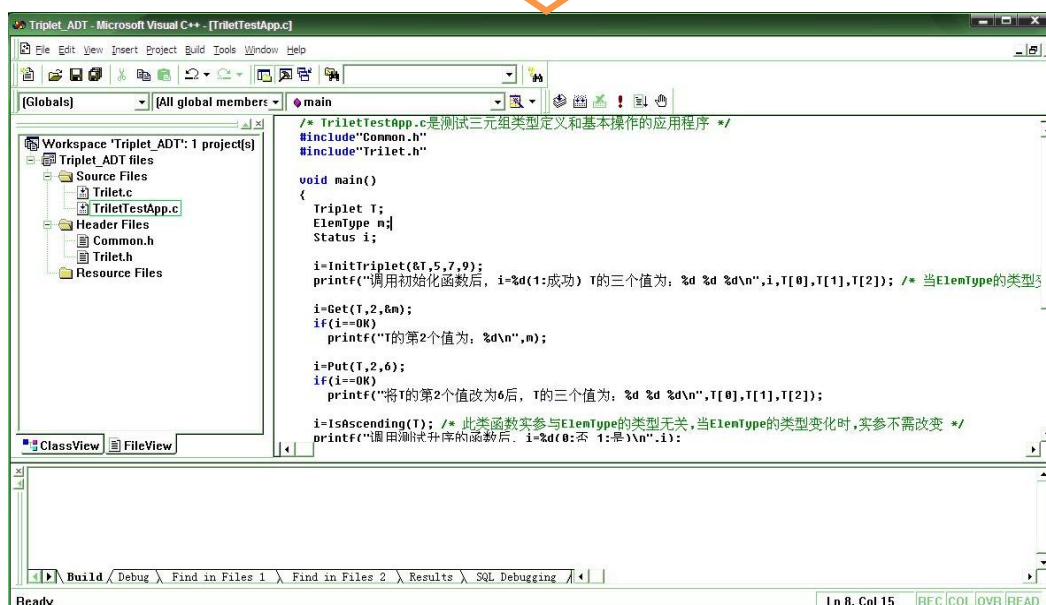


图 2-5 创建源文件 TripletTestApp.c

编译、连接，如图 2-6 所示。先对工程中每个源文件进行编译，编译通过后，再进行连接，连接成功后，点击运行。

编译

连接

运行

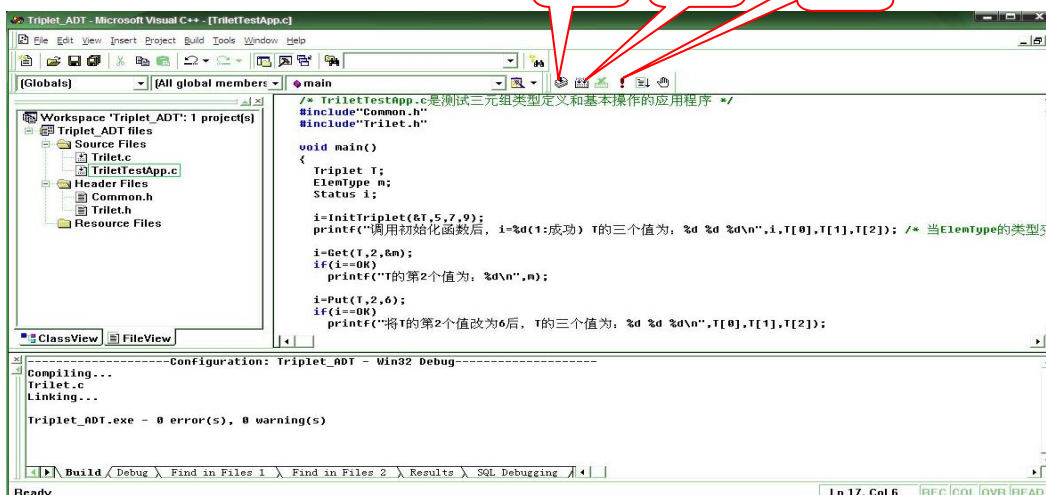
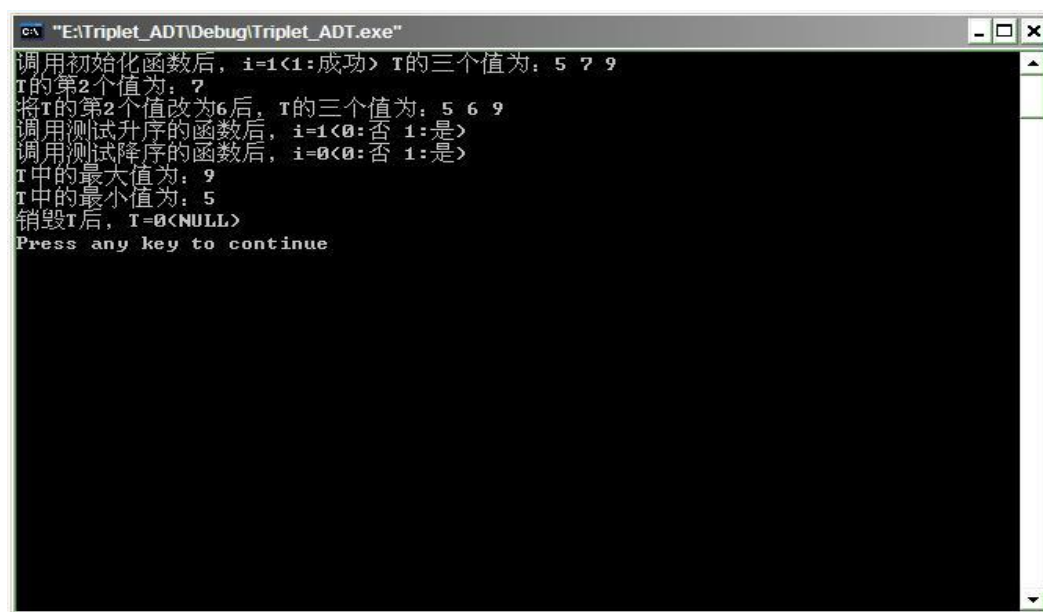


图 2-6 编译、连接

测试程序的执行结果，如图 2-7 所示。



```
c:\ "E:\Triplet_ADT\Debug\Triplet_ADT.exe"
调用初始化函数后, i=1<1:成功> T的三个值为: 5 7 9
T的第2个值为: 7
将T的第2个值改为6后, T的三个值为: 5 6 9
调用测试升序的函数后, i=1<0:否 1:是>
调用测试降序的函数后, i=0<0:否 1:是>
T中的最大值为: 9
T中的最小值为: 5
销毁T后, T=0<NULL>
Press any key to continue
```

图 2-7 运行结果

四、实验：实现复数抽象数据类型及基本操作，并进行测试

上述实例（三元组抽象数据类型实现）的源代码，我们给出了两种实现方法（triplet1 和 triplet2），它们只在细微上有些差别：**一种是用引用&作形参，一种用指针*作形参**。请同学们参考并理解 triplet1 或 triplet2 来解决本实验内容。

具体步骤如下（依据上述实例进行）：

1. 复数抽象数据类型的定义

ADT Complex

{

数据对象： $D = \{e1, e2 | e1, e2 \in \text{RealSet}\}$

数据关系： $R1 = \{ \langle e1, e2 \rangle | e1 \text{ 是复数的实部, } e2 \text{ 是复数的虚部} \}$

基本操作：

AssignComplex(&Z, v1, v2); //创建复数 Z

操作结果：构造复数 Z，其实部和虚部分别被赋以参数 v1, v2 的值。

PrintComplex(Z);

初始条件：复数 Z 已存在。

操作结果：输出复数 Z（形如：a+bi）。

GetReal(Z, &RealPart);

初始条件：复数 Z 已存在。

操作结果：用 RealPart 返回复数 Z 的实部值。

GetImag(Z, &ImagPart);

初始条件：复数 Z 已存在。

操作结果：用 ImagPart 返回复数 Z 的虚部值。

AddComplex (Z1, Z2, &Sum);

初始条件：复数 Z1 和 Z2 都存在。

操作结果：用 Sum 返回两个复数 Z1 和 Z2 的和。

SubComplex (Z1, Z2, &Sub)

初始条件：复数 Z1 和 Z2 都存在。

操作结果：用 Sub 返回两个复数 Z1 和 Z2 的差。

MulComplex (Z1, Z2, &Mul)

初始条件：复数 Z1 和 Z2 都存在。

操作结果：用 Sub 返回两个复数 Z1 和 Z2 的乘积。

} ADT Complex

2. 用 C 语言类型声明定义复数类型

1) 选择存储结构，定义复数类型。

（要求：复数的实部和虚部为浮点型）。

3. 用 C 语言实现复数类型的基本操作

在复数类型定义的基础上，用 C 语言实现复数类型的主要基本操作（一个复数的创建，输出，求两个复数的和、差、乘积）。

1) 建立 Complex.h 文件，将复数类型定义和基本操作的函数声明放在其中。

2) 建立 Complex.cpp 文件，将基本操作对应的函数放在其中。

4. 编写应用程序 ComplexTestApp.cpp 测试上述复数类型，要求：

1) 由输入的实部和虚部生成两个复数；

2) 求两个复数的和；

3) 求两个复数的差；

4) 求两个复数的乘积；

5) 设计合适的显示输出方式，输出复数及相应运算结果。

6) 用户可看到如下界面：

* 1. 创建复数 C1 *


```
* 2.  创建复数 C2          *
* 3.  输出复数 C1          *
* 4.  输出复数 C2          *
* 5.  求 C1 和 C2 的和，并输出和    *
* 6.  求 C1 和 C2 的差，并输出差    *
* 7.  求 C1 和 C2 的积，并输出积    *
* 0.  结束                  *
*****
```