第 1 章　抽象数据类型项目实训

对于一些基本数据类型无法描述的信息结构，可以将其逻辑特性抽象出来，形成一组数据以及对这些数据的一组逻辑操作，这称为**抽象数据类型**。

逻辑特征抽象层次越高，抽象数据类型的复用性程度就越高，因此，有必要掌握如何定义一个抽象数据类型，以及如何用代码来实现该数据类型。

一、本章实训目的

- 用 C 或 C++语言实现一个抽象数据类型；
- 实现一个用户操作界面来验证该数据类型；
- 运行程序并对其进行测试。

二、实训项目要求

编写程序实现以下抽象数据类型，并实现一个主函数调用三元组的所有函数操作。

ADT Triplet {

数据对象：D = {e1, e2, e3 | e1, e2, e3 ∈ Elemset}

数据关系：R1 = {<e1, e2>, <e2, e3>}

基本操作：

InitTriplet(&T, v1, v2, v3)

操作结果：构造三元组 T，元素 e1、e2、e3 分别赋值为 v1、v2、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≤i≤3。

操作结果：用 e 返回 T 的第 i 元值。

Put(&T, i, e)

初始条件：三元组 T 已存在，1≤i≤3。

操作结果：改变 T 的第 i 元值为 e。

IsAscending(T)

初始条件：三元组 T 已存在。

操作结果：若 T 的 3 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：若 T 的 3 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 3 个元素最大值。

Min(T, &e)

初始条件：三元组 T 已存在。

操作结果：用 e 返回 T 的 3 个元素最小值。

} ADT Triplet

三、重要代码提示

使用连续的空间来存储三元组中的元素，如果是动态三元组的内存，则只定义三元组类型 Triplet 为其元素类型的指针即可。

```
// Triplet类型是ElemType类型的指针，用于存放ElemType类型的地址
typedef ElemType *Triplet; // 由InitTriplet分配3个元素存储空间
```

三元组初始化函数 InitTriplet 传入已定义的 Triplet 变量的引用 T（因此，函数要将新申请的空间指针赋予三元组变量，所以参数必须传入三元组变量的引用），同时传入三元组的 3 个初值 v1、v2、v3。在函数中先为 T 申请 3 个 ElemType 类型的空间，如果申请失败，则直接退出程序，否则将 v1、v2、v3 分别赋值给 T[0]、T[1]、T[2]，参考代码如下。

```
Status InitTriplet(Triplet &T, ElemType v1, ElemType v2, ElemType v3)
{
    // 操作结果：构造三元组T，依次置T的3个元素的初值为v1、v2和v3
    if (!(T = (ElemType *) malloc(3 * sizeof(ElemType))))
    {
        exit(OVERFLOW);
    }
    T[0] = v1, T[1] = v2, T[2] = v3;
    return OK;
}
```

因为三元组 T 是连续分配的空间，所以销毁三元组的函数 DestroyTriplet 只需要直接释放 T 所指向的空间，随机将 T 置为空即可。

```
Status DestroyTriplet(Triplet &T)
{
    // 操作结果：三元组T被销毁
    free(T);
    T = NULL;
    return OK;
}
```

获取三元组元素值的函数 Get 通过参数 i 传入元素顺序，通过引用 e 传出元素值。首先，判断 i 的合法性，因为是三元组，所以 i 值只能为 1～3。因为下标编号是从 0 开始的，所以 T[i-1] 是三元组第 i 个元素的值，最后将 T[i-1]的值赋予 e。

```
Status Get(Triplet T, int i, ElemType &e)
{
    // 初始条件：三元组T已存在，1≤i≤3
    // 操作结果：用e返回T的第i个元素的值
    if (i < 1 || i > 3)
    {
        return ERROR;
    }
    e = T[i - 1];
```

```
    return OK;
}
```

修改三元组元素值的函数 Put 和函数 Get 类似，只是将要修改的值通过参数 e 传递进来，并将其赋值给 T[i-1]，在此之前同样要对 i 的合法性进行判断，见如下代码。

```
Status Put(Triplet T, int i, ElemType e)
{
    // 初始条件：三元组T已存在，1≤i≤3
    // 操作结果：改变T的第i元素的值为e
    if (i < 1 || i > 3)
    {
        return ERROR;
    }
    T[i - 1] = e;
    return OK;
}
```

判断三元组是否升序排列的函数 IsAscending 实现起来很简单，只需要比较相邻的两个元素是不是后者比前者大，如果后者比前者大则返回 1，否则返回 0。

```
Status IsAscending(Triplet T)
{
    // 初始条件：三元组T已存在
    // 操作结果：如果T的3个元素按升序排列，则返回1，否则返回0
    return (T[0] <= T[1] && T[1] <= T[2]);
}
```

判断降序排列的函数 IsDescending 和 IsAscending 类似，代码略过。从三元组中找出最大元素的函数 Max 先比较 T[0]和 T[1]，将大的赋值给引用参数 e，再比较 e 和 T[2]，将大的赋值给 e 即可。找寻最小数的函数 Min 和函数 Max 类似，具体代码这里不再赘述。

```
Status Max(Triplet T, ElemType &e)
{
    // 初始条件：三元组T已存在
    // 操作结果：用e返回指向T的最大元素的值
    e = T[0] >= T[1] ? T[0] : T[1];
    e = e >= T[2] ? e : T[2];
    return OK;
}
```