目 录

1 基于顺序存储结构的线性表实现

1.1 问题描述

本次实验的主要内容是采用顺序表作为线性表的物理结构，实现线性表抽象数据类型（Abstract Data Type，缩略名 ADT）的基本运算，构造一个具有菜单的功能演示系统，并要求实现线性表的文件形式保存和线性表管理。本实验将以上述内容为主要目标，使用 C++语言构建问题解决方案。

1.1.1 问题 1：线性表 ADT 的顺序表实现

根据本次实验要求，实验人员应当以 ADT 的最小完备性和常用性相结合为基本原则，以顺序表的物理结构为基础，通过函数形式定义线性表的初始化表、销毁表、清空表、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素和遍历表总计 12 种基本运算。

1.1.2 问题 2：构造具有菜单的功能演示系统

根据本次实验要求，实验人员应当构造能够良好体现线性表 ADT 功能的演示程序，至少应当显式提供操作线性表的主菜单。其中，应当在主程序中完成函数调用所需实参值的准备和实现函数执行结果的显示，并提供适当的操作提示信息。演示系统应当实现输入验证和异常响应，对于用户的非法输入予以提示，并请求用户重新进行输入；对于程序运行时错误也应当输出异常信息。另外，演示系统应当具有合理的层次，处理好主菜单和子菜单的关系。

1.1.3 问题 3：实现线性表的文件形式保存

根据本次实验要求，实验人员应当至少实现与线性表的文件形式保存相关的如下内容。

（1）设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息。

（2）设计线性表文件保存和加载操作的合理模式。要求实现时不能随意浪费文件存储空间。

1.1.4 问题 4：实现线性表管理

根据本次实验要求，实验人员可选择实现多个线性表管理。线性表管理部分要求实现线性表的新建，加载和删除功能。应当设置线性表数量上限，当数量达最大值时，输出相关的提示信息；对于加载和删除功能，应当在无可用线性表时输出相关的提示信息。

1.2 系统设计

根据本次实验要求，系统设计方面主要包括如下内容。

（1）设计使用顺序表实现下的线性表 ADT 的物理结构。

（2）设计具有菜单的演示系统框架。

（3）设计管理多个多种类型的线性表的模式。

（4）设计文件格式。

1.2.1 顺序表实现下的线性表 ADT 的物理结构设计

线性表 ADT 的物理结构如下：

```
template <typename Type>                //顺序表的模板类定义
class SqList: public SqList_base        //继承自顺序表基类，基类提供虚方法
{                                       //便于使用多态指针
private:
        Type* elem;                     //顺序表起始指针
        int length;                     //顺序表长
        int listsize;                   //顺序表当前大小
        public:
        //类方法此处从略
};
```

考虑到类与线性表 ADT 的良好相性，此处使用定义类的方式实现线性表 ADT 的物理结构。而之所以使用模板类，是为了对多个或多种类型的线性表进行创建和管理。此外还定义了模板类基类，以便使用多态指针，以期为演示系统的实现打下良好基础。

1.2.2 演示系统框架设计

在演示系统中，为实现线性表管理和线性表操作，考虑设计多级菜单。演示系统结构如图 1-1 所示。

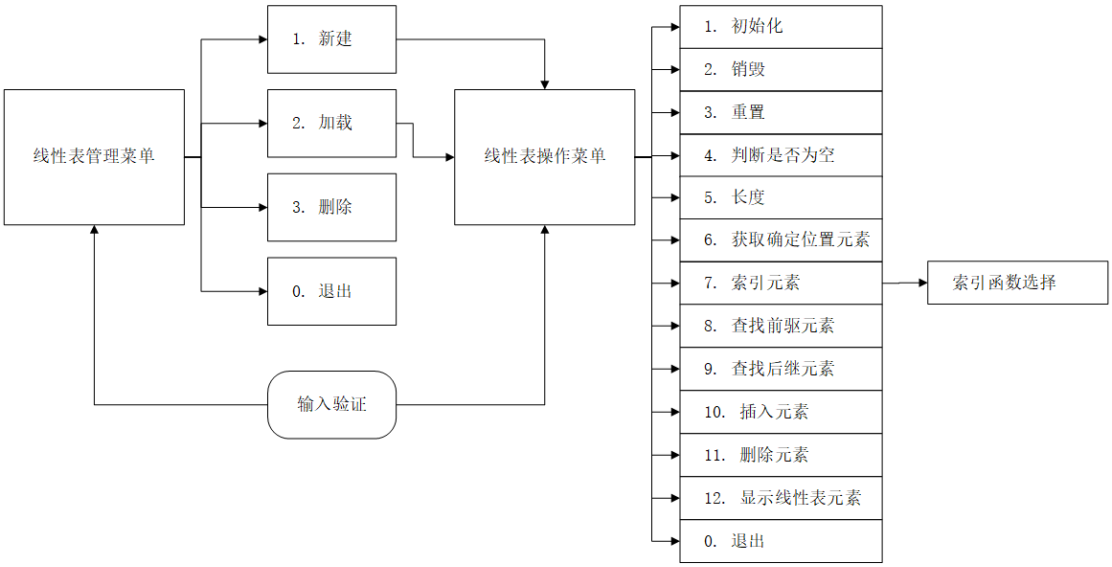

图 1-1　演示系统结构

演示系统包括 2 个主菜单和若干子菜单。线性表管理菜单包含新建、加载、删除、退出总计 4 个选项。线性表操作菜单包括实验所要求的 12 种基本运算选项；其中，“索引元素”选项附设“索引函数选择”子菜单。考虑到演示系统功能完备性，对所有菜单的操作都需要实现输入验证。

1.2.3　管理多个多种类型的线性表的模式设计

以顺序表模板类为基础，考虑设计用户模板类和文件读写类，以实现对线性表的管理。具体考虑下，用户类负责记录一个线性表的所有信息，实现对线性表操作菜单的总体控制，并提供从文件读取线性表和向文件输出线性表的功能；文件读写类负责读取资源文件，与用户区通信，以及修正文件格式等功能。

其中记录线性表信息的结构如下：

```
//使用 string 类、streampos 类以及自定义类型，详情请参见头文件 user.h
struct directory                    //线性表信息存储结构
{
    string source;                  //线性表资源文件
    int cnt;                        //线性表位置，从 0 开始
    streampos beg;                  //线性表信息起始点
    streampos end;                  //线性表信息终止点
    streampos list_beg;             //线性表信息数据区起始点
    string list_name;               //线性表名
    type_code id;                   //线性表数据类型
    int span;                       //线性表数据类型大小
    int length;                     //线性表表长
    int yn_valid;                   //线性表信息状态(有效/无效)开关
```

```
};
```

记录资源文件信息的结构如下：

```
//使用 string 类，详情请参见头文件 file_io.h
struct resource_file_handle                    //资源文件信息存储结构
{
    string file_name;                          //资源文件名
    int file_cnt;                              //资源文件中存储的线性表数量
};
```

通过使用以上 2 个结构类型，合理搭建用户模板类和文件读写类，并实现类间的通信，来完成对多个或多种类型的线性表的管理。基于以上内容，此处做出关于文件读写的补充说明：用户类完成线性表信息的读写，文件读写类则对用户类写入的内容进行整理，使之符合文件格式。

1.2.4 文件格式设计

在符合实验要求的前提下，本次实验将使用二进制文件作为线性表信息的资源文件，并提供文本文件输出。

二进制文件使用“索引+信息”的格式方法设计，其具体格式如表 1-1 所示。

表 1-1 二进制文件格式

位置	类型	值	描述
0000	32 bit integer	??(最大 100)	资源文件保存的线性表数量
0004	32 bit integer	??	第 1 个线性表信息的位置
0008	32 bit integer	??	第 2 个线性表信息的位置
…	…	…	…（以此类推）
0400	32 bit integer	??	第 100 个（最后一个）线性表信息的位置
0404	32 bit integer	x (最大 100)	线性表名长度
0408	x * 8 bit character	??	线性表名
0408 + x	32 bit integer	??(可能值: 0, 1, 2)	线性表类型标识码
0412 + x	32 bit integer	y	类型字长
0416 + x	32 bit integer	len(最大 1000)	线性表长
0420 + x	y * len * 8 bit character	??	线性表数据信息
…	…	…	…（以此类推）

文本文件使用“表名+数据”的格式方法设计，其具体格式如表 1-2 所示。

表 1-2　文本文件格式

（文本文件）
1: （线性表名）10 个整数
1　2　3　4　5
6　7　8　9　10

2: （线性表名）10 个实数
84.8　83.9429　83.0857　82.2286　81.3714
80.5143　79.6571　78.8　77.9429　77.0857

…

二进制文件的主要作用是作为线性表信息的资源文件，以便程序读取和存储线性表信息，文本文件的主要作用是为用户提供直观的参考。

1.3　系统实现

本次实验使用的开发环境为 Microsoft Visual Studio 2017，使用 Microsoft Build Engine 编译器。

以系统设计部分为参考，系统实现方面主要包括如下内容。

（1）项目工程说明。

（2）线性表 ADT 运算的顺序表实现。

（3）多个多类型线性表操作支持实现。

（4）文件输入输出实现。

（5）包含输入验证的演示系统实现。

1.3.1　项目工程说明

项目工程包含 7 个头文件和 6 个 C++源文件。

7 个头文件为 Char.h、generaldef.h、sequence_list.h、exception_res.h、user.h 、file_io.h、main_func.h。

6 个 C++源文件为 Char.cpp、generaldef.cpp、exception_res.cpp、file_io.cpp、main_func.cpp、main_presentation.cpp。

使用的标准库文件有 iostream、fstream、string、exception、string.h、limits.h、float.h、stdlib.h、windows.h。

文件描述如表 1-3 所示。

表 1-3　项目工程文件描述

文件	描述
Char.h	中文字类。提供中文字类 Char 定义，提供 2 字节中文字符存储支持。
generaldef.h	通用内容。提供状态常量定义，函数返回值类型声明，比较和输入验证模板函数定义。
sequence_list.h	顺序表模板类。提供顺序表基类 SqList_base 定义，派生顺序表模板类 SqList 定义。
exception_res.h	异常类。提供异常类 List_except 定义。
user.h	用户类。提供用户基类 User_base 定义，派生用户模板类 User 和类方法定义，线性表信息存储结构 directory 定义。
file_io.h	文件管理类。提供文件读写类 fileIO 定义，用户区信息存储结构 list_handle 定义，资源文件信息存储结构 resource_file_handle 定义。
main_func.h	主程序函数原型。提供默认演示文件名常量定义，主演示函数和功能演示函数原型。
Char.cpp	中文字类方法定义。提供中文字类 Char 类方法和友元函数定义。
generaldef.cpp	通用内容。提供字符串输入验证函数定义。
exception_res.cpp	异常类方法定义。提供异常类 List_except 类方法定义。
file_io.cpp	文件管理类方法定义。提供文件读写类 fileIO 类方法定义。
main_func.cpp	主程序函数定义。提供主演示函数和功能演示函数定义。
main_presentation.cpp	此文件包含 "main" 函数。程序执行将在此处开始并结束。

此处列出 generaldef.h 中的常量定义和类型声明以供参考：

```
typedef int Status;                    //函数返回值结果状态类型
//函数结果状态
const Status TRUE = 1;                 //真
const Status FALSE = 0;                //假
const Status OK = 1;                   //成功
const Status ERROR_M = 0;              //严重错误
const Status INFEASIBLE = -1;          //非法
const Status OVERFLOW_M = -2;          //溢出
const Status FULL = -3;                //满
const Status NOUSE = -4;               //无效
//字符串最大长度限制
const size_t str_maxlen = 100;
```

1.3.2 线性表 ADT 运算的顺序表实现

基于顺序表模板类 SqList，线性表 ADT 运算由类方法实现。

以下为模板类 SqList 定义和常量定义，供参考：

```
const int LIST_INIT_SIZE = 100;      //线性表存储空间的初始分配量
const int LISTINCREMENT = 10;        //线性表存储空间的分配增量
const int MAX_LIST_LEN = 1000;       //最大线性表长度
template <typename Type>             //顺序表的模板类定义
class SqList: public SqList_base     //继承自顺序表基类，基类提供虚方法，
{                                    //便于使用多态指针
private:
    Type* elem;                      //顺序表起始指针
    int length;                      //顺序表长
    int listsize;                    //顺序表当前大小
public:
    //类方法此处从略
};
```

运算 1：初始化表。由类方法 InitList()实现。该函数使用指针 elem 从堆区申请 LIST_INIT_SIZE 的内存，初始化成员变量。

运算 2：销毁表。由类方法 DestroyList()实现。如果指针 elem 不为空，该函数释放指针 elem 指向的内存，并将所有成员变量置零，返回成功(OK)；否则，返回无效(INFEASIBLE)。

运算 3：清空表。由类方法 ClearList()实现。该函数简单地调用类方法 DestroyList()和 InitList()。

运算 4：判定空表。由类方法 ListEmpty()实现。由指针 elem 判断线性表的存在性，由 length 判断线性表是否为空。

运算 5：求表长。由类方法 ListLength()实现。当指针 elem 不为空时，简单地返回 length 成员值。

运算 6：获得元素。由类方法 GetElem(int i, Type &e)实现。当指针 elem 不为空且索引位置有效时，用 e 返回 elem 指向内存的第 i 个元素。

运算 7：查找元素。由类方法 LocateElem(const Type e, Status(*compare)(const Type ie, const Type ix))实现。该函数的第 1 个参数是定位元素值，第 2 个参数是返回值类型为 Status 的比较函数指针。该函数遍历 elem 指向的内存，对每个元素调用 compare 指向的函数，如果比较成功则返回元素位置。函数所使用的算法

1 个参数是插入位置值，第 2 个参数是插入数据元素值。该函数首先判断线性表是否已满，然后由指针 elem 判断线性表的存在性（线性表不存在时表长为零，前述 2 个步骤可颠倒），再判断插入位置的合法性，最后判断是否需要追加存储空间；而后从尾端开始遍历 elem 指向的内存，移动元素，留出空位插入新元素。函数所使用的算法时间复杂度为 O(n)，流程图如图 1-3 所示。

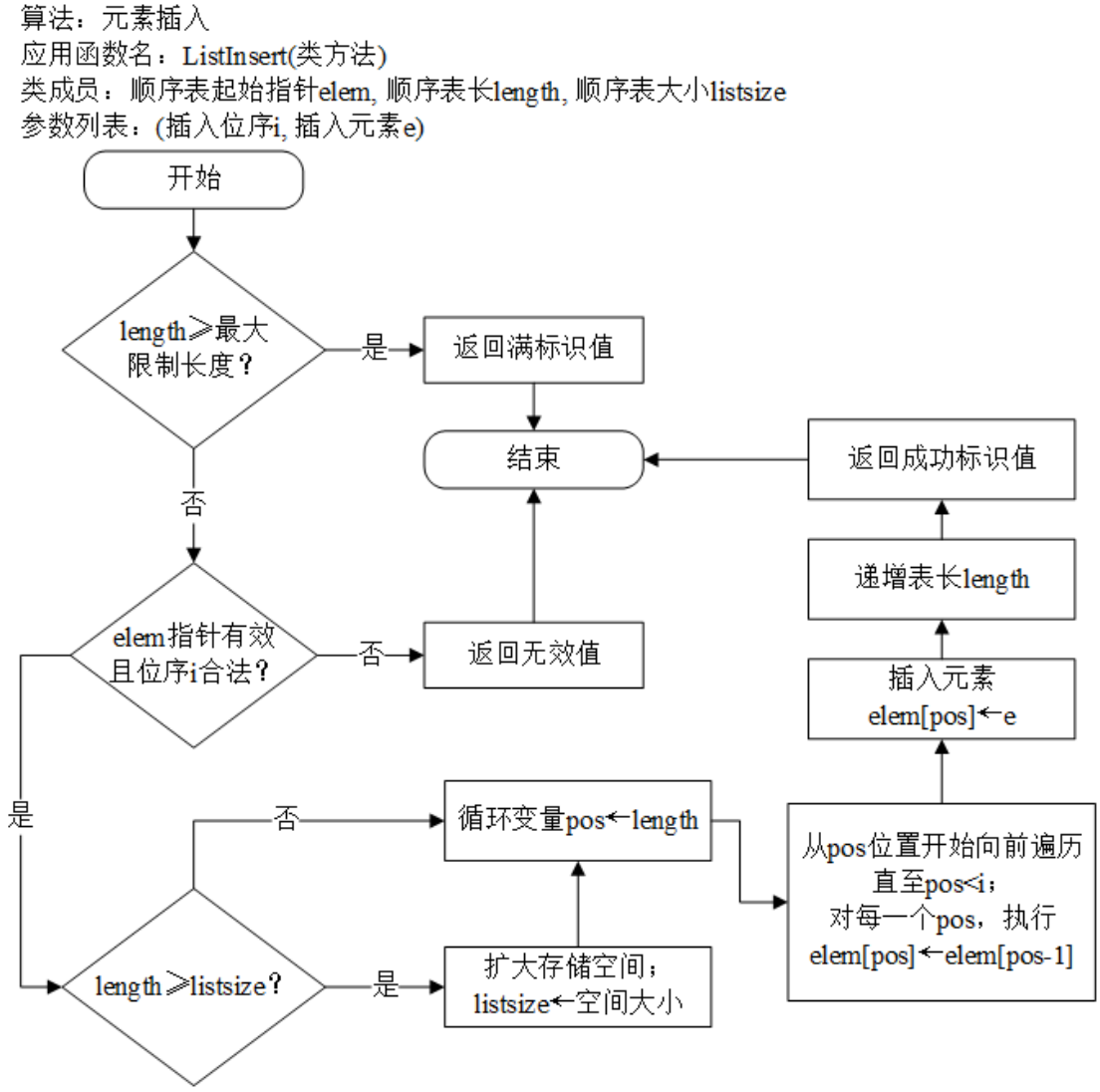

图 1-3　元素插入算法流程图

运算 11：删除元素。由类方法 ListDelete (int i, Type &e)实现。该函数的第 1 个参数是删除位置值，第 2 个参数是存储变量引用。该函数首先由指针 elem 判断线性表的存在性，再判断删除位置的合法性。而后用 e 返回第 i 个元素，从第 i 个位置开始遍历 elem 指向的内存，移动元素，完成删除。函数所使用的算法时间复杂度为 O(n)。

```
        is.setstate(ios::eofbit);
        is.setstate(ios::failbit);
    }
    else if (Ch.Chin[0] == '\n')      //处理请求读入的字符为换行符'\n'的情形
    {
        is.unget();                   //配合性地送回换行符'\n'
    }
    //…余下内容从略
    return is;
}
```

重载运算符“>>”函数的第 1 个参数是输入流类引用，第 2 个参数是 Char 类变量引用。该函数通过读取流状态，分支处理非法情况，设置流状态掩码和配合性送回字符。通过重载，输入验证的实现得到了有力的支持，这也是实现多类型线性表操作支持的典型代表。

包含内置类型和 Char 类在内，User 类也提供了相关支持，而 fileIO 类提供了使用方式。

以下为用户基类 User_base 和用户模板类 User 定义，供参考：

```
class User_base
{
    //相关性较低的内容此处从略
public:
    //标示继承类应当实现的类方法
    virtual Status crtl_proc() = 0;
    virtual Status load_list(const directory& dir, const int yn_ini = NO_INIT) = 0;
    virtual int data_is_del() = 0;
};
template <typename T>
class User: public User_base
{
private:
    directory info;                   //线性表信息
    SqList<T> data;                   //数据区
    //…余下内容从略
};
```

User 类首先是一个模板类，它的成员中有一个是 SqList<T> data，说明它可以对一个线性表进行操作支持，此处以模板的形式声明了相应类型的顺序表类，所以可以建立多种类型的用户区。User 类的另一个特性是对虚基类 User_base 的公有继承，故可以使用 User_base*多态指针操作派生类。

接下来，fileIO 类将实现多个线性表管理，它的类方法确实使用到了 User_base*多态指针。

以下为 file_io.h 节选，供参考：

```
struct list_handle                    //用户区信息存储结构
{
    directory l_dir;                  //用户区线性表信息
    User_base* ub;                    //用户基类指针
};
//资源文件信息存储结构
struct resource_file_handle
{
    string file_name;                 //资源文件名
    int file_cnt;                     //资源文件中存储的线性表数量
};
//资源文件读写类
class fileIO
{
private:
    resource_file_handle rfh;         //资源文件信息
    SqList<directory> dir;            //所有线性表信息
    list_handle lh;                   //用户区信息
    //…余下内容从略
};
```

以下为 directory 结构，供参考：

```
struct directory                      //线性表信息存储结构
{
    string source;                    //线性表资源文件
    int cnt;                          //线性表位置，从 0 开始
    streampos beg;                    //线性表信息起始点
    streampos end;                    //线性表信息终止点
    streampos list_beg;               //线性表信息数据区起始点
    string list_name;                 //线性表名
    type_code id;                     //线性表数据类型
    int span;                         //线性表数据类型大小
    int length;                       //线性表表长
    int yn_valid;                     //线性表信息状态(有效/无效)开关
};
```

fileIO 类的 rfh 成员存储资源文件信息，dir 成员存储所有线性表信息，lh 成员存储用户区信息。

首先，dir 成员复用模板类 SqList 将所有线性表信息存入，但并不存入线性

表的海量数据，由此可以在节省内存空间同时获取所有必要信息，这样就完成了线性表管理的第一步。

而后，lh 成员用于处理 User 类和 fileIO 类的通信。lh 成员拥有一个 directory 变量和一个 User_base*基类指针，借助虚基类指针的多态性，使得创建多种类型的 User 类成为可能。

另外，从 fileIO 类的设计可以发现：fileIO 类最多只能同时创建一个用户区，每当用户区使用完毕，就需要交换数据，更新 dir 成员，销毁用户区，为下一次创建做好准备。

在 fileIO 类对多类型线性表的使用方面，主要体现为 User_base*多态指针的使用。以下为 fileIO 类 load_list 类方法的定义节选，供参考：

```
Status fileIO::load_list(const int cnt_in)
{   //相关性较低的内容此处从略
    //加载线性表功能函数: 提供对加载功能的支持
    //执行用户区构造
    switch (in.id)          //构造类型处理开始，使用用户基类指针创建类
    {
        case TYPE_INT:
            lh.ub = new User<int>;
            if (!lh.ub)
                std::exit(OVERFLOW_M);
            break;
        case TYPE_DBL:
            lh.ub = new User<double>;
            if (!lh.ub)
                std::exit(OVERFLOW_M);
            break;
        case TYPE_CH:
            lh.ub = new User<Char>;
            if (!lh.ub)
                std::exit(OVERFLOW_M);
            break;
    }
    //加载线性表信息存储结构，加载原有线性表
    if (lh.ub->load_list(in) != OK)
        return ERROR_M;
    //…余下内容从略
}
```

在此类方法中，lh 的 User_base*成员 ub 被用于多种用户区的创建，并被用

于调用 User 类方法：lh.ub->load_list(in)。此处调用的类方法取决于 lh.ub 指向的 User 类类型。例如，如果 lh.ub 指向 User<int>类，则将调用 User<int>类的类方法 load_list<int>。借由这种模式，对多种类型的线性表管理得以实现。

1.3.4 文件输入输出实现

文件输入输出部分是由 User 模板类和 fileIO 类共同实现的。

文件输入方面。fileIO 类从文件将所有线性表信息存入 dir 成员，然后依据用户请求创建类型恰当的用户区，将线性表信息存入用户区 lh.l_dir，提请 User 类读取相关线性表数据，并将此线性表的权限转交给 User 类，由 User 类响应用户输入。

文件输出方面。由于 fileIO 类并不存入线性表数据，所以线性表数据的输出依赖用户区 User 类来完成，即由 User 类将线性表数据写入资源文件。但是 User 类并不知道资源文件的所有格式信息，所以它只负责将线性表数据写入到资源文件的末尾。因此，fileIO 类还需要对资源文件的格式进行修正，依据用户区传回的 directory 线性表信息，fileIO 类重新打开资源文件进行数据区的移动操作，并更新索引信息。

此处提供一个 fileIO 类修正资源文件的示例，取自 file_io.cpp，供参考：

```
Status fileIO::load_list(const int cnt_in)
{
    //…前文内容从略
    dir.GetElem(rfh.file_cnt, in);              //读取最后一个线性表信息
    //执行用户区数据向资源文件写入，写入到文件有效数据的最后
    if (lh.ub->write_list(in) != OK)
        return ERROR_M;
    in = *lh.ub->dir();                         //读取用户区线性表信息
    directory out;
    streampos pos_beg, pos_end;
    fs.open(rfh.file_name, ios::in | ios::out | ios::binary);
      //对资源文件进行读写，修正文件结构
    if (fs.is_open())
    {
        dir.GetElem((cnt_in + 1), out);
        int new_gap;
        int old_gap;
        int gap, remain, mem;
        char limbuf[bufsize];
```

```
    cout << max_size << "]" << "\t2. 加载"                << endl;
    cout << "              3. 删除\t\t0. 退出"             << endl;
    cout << "-------------------------------------------------" << endl;
    //输入验证开始
    op = checkin<int>("****请选择你的操作[0~3]: ",
                      "****(输入非法)请重新选择操作[0~3]: ", true, 0, 3);
    //输入验证结束
    return op;
}
```

线性表操作菜单，节选自 user.h，供参考：

```
//控制菜单函数: 提供控制菜单显示和输入验证
template <typename T>
StateType User<T>::crtl_menu()
{   //相关性较低的内容此处从略
    StateType op = -1;
    cout << "                    线 性 表 演 示 功 能 菜 单 "    << endl;
    cout << "-------------------------------------------------"              << endl;
    cout << "            1. 初始化                  7. 索引元素"      << endl;
    cout << "            2. 销毁                    8. 查找前驱元素"     << endl;
    cout << "            3. 重置                    9. 查找后继元素 "    << endl;
    cout << "            4. 判别是否为空           10. 插入元素"      << endl;
    cout << "            5. 长度                   11. 删除元素"      << endl;
    cout << "            6. 获取确定位置元素       12. 显示线性表元素" << endl;
    cout << "            0. 退出"                                     << endl;
    cout << "-------------------------------------------------"              << endl;
    //输入验证开始
    op = checkin<int>("****请选择你的操作[0~12]: ",
                      "****(输入非法)请重新选择你的操作[0~12]: ", true, 0, 12);
    //输入验证结束
    return op;
}
```

演示系统的输入验证主要通过 generaldef.h 中定义的 checkin 模板函数和 generaldef.cpp 中定义的 mygetline 函数实现。

以下为 checkin 模板函数的定义节选，供参考：（代码后方列出了流程图）

```
//通用输入验证函数:
//tip: 输入提示信息
//err: 错误和重新输入提示信息
//yn_lim: 输入范围开关
//down: 允许的下限(包含下限)
//up: 允许的上限(包含上限)
template <typename T>
T checkin(const char* tip, const char* err, const bool yn_lim = false,
```

```
            T down = (T)0, T up = (T)0)
{   //相关性较低的内容此处从略
    T tmp;
    int st;
    char ch;
    cout << tip;
    //第 1 次输入验证开始
    cin >> tmp;                                    //析取输入
    st = cin.rdstate();                            //获取输入流状态
    if (st != 0 && st != 3)                        //处理非 eof 的析取失败情况
    {
        cin.clear();                               //重置输入流
        while (1)                                  //清空输入缓冲
        {
            ch = cin.get();
            if (ch == '\x1a' || ch == '\n')        //'\x1a'表示'crtl+z'控制字符
                break;
        }
    }
    else if (st == 3)                              //处理 eof
        cin.clear();                               //重置输入流
    else                                           //有效输入
    {
        ch = cin.get();
        if (ch == '\n')                            //说明没有后续非法输入
        {
if (!yn_lim || (tmp >= down && tmp <= up))
            {  //输入范围开关打开则进行范围验证
                return tmp;
            }
        }
        while (1)                                  //输入非法，清空输入缓冲
        {
            if (ch == '\x1a' || ch == '\n')        //'\x1a'表示'crtl+z'控制字符
                break;
            ch = cin.get();
        }
    }
    //第 1 次输入验证结束
    //循环输入验证，重复与第 1 次相似的过程
    while (1)
    {
    //…循环验证内容从略
}
```

```
    return tmp;
}
```

其验证算法的主要流程如图 1-4 所示。

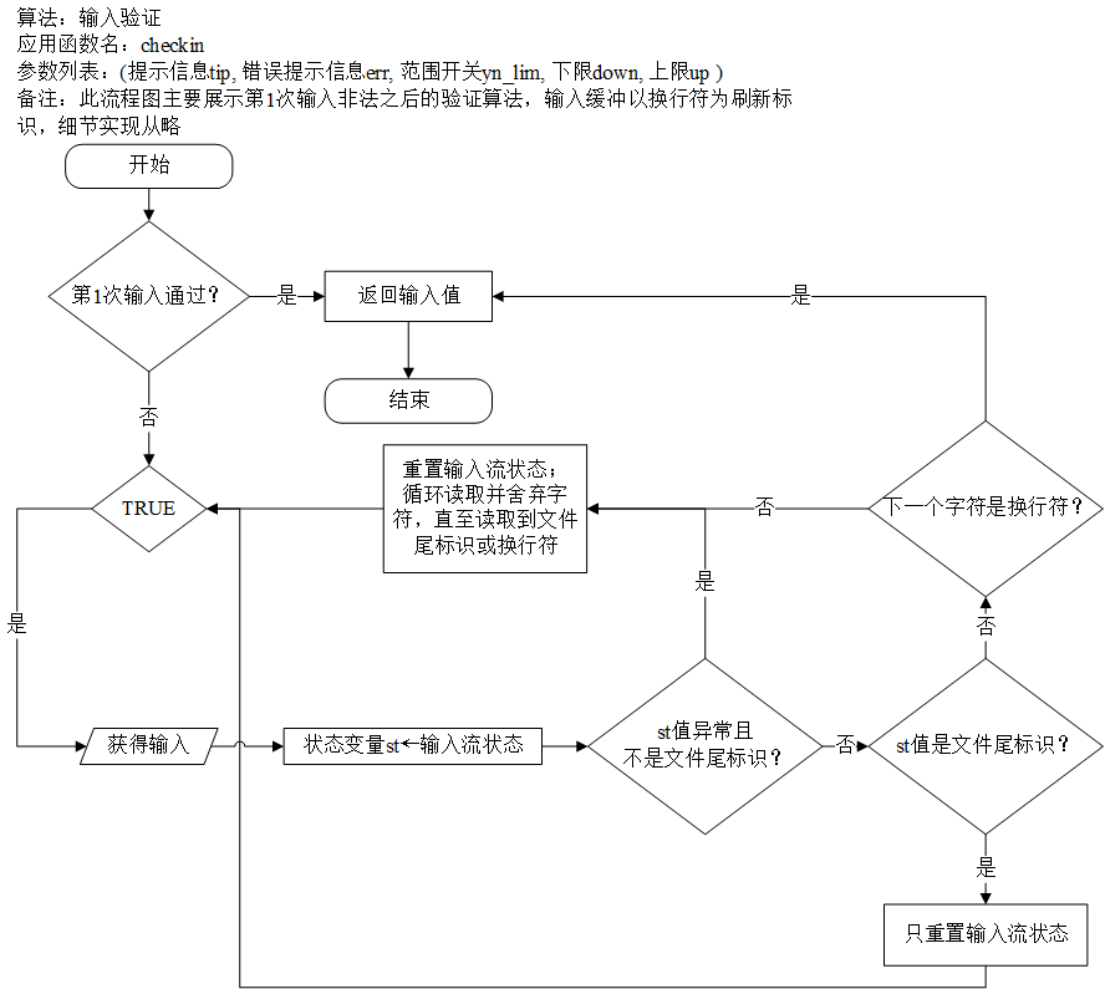

图 1-4　输入验证算法流程图

基本上，checkin 函数通过读取流状态，分支处理非法情况，取消流阻塞，并设法清空缓冲区，为处理后续输入做好准备。具体而言，checkin 函数接受最多 5 个参数：tip 和 err 分别为输入提示信息、错误和重新输入提示信息；yn_lim 是输入范围开关，决定 down 和 up 信息的有效性；down 和 up 是输入有效范围指标。实际输入响应过程中，值得注意的是控制字符“Crtl+Z”的输入阻塞作用，即“Crtl+Z”后面的输入不会进入缓冲区，所以换行符将会丢失。因此在输入验证时，同时将“Crtl+Z”和换行符作为输入结束的标志进行处理。

总而言之，提示信息显示和输入验证是演示系统完备性和友好度的重要体现，在本次演示系统的实现过程中尽可能地考虑了所有输入情形，而接下来将对此进行充分的测试。

1.4 系统测试

根据本次实验要求，系统测试主要包括如下内容。

（1）测试准备。

（2）测试计划。

（3）测试结果。

（4）测试分析。

1.4.1 测试准备

为方便对演示系统各项功能进行测试，另外编写了一个输出测试文件的简单程序，该程序依照文件格式设计输出 2 份测试文件 Demo1.bin，Demo2.bin。

测试文件 Demo1.bin 的数据说明如下：

1 个　长度为 999 的整数类型线性表-数据内容为 0 至 998

1 个　长度为 100 的实数类型线性表-数据内容为随机实数

1 个　长度为 388 的中文字符类型线性表-数据内容为短文

997 个 长度为 1000 的实数类型线性表-数据内容为实数

测试文件 Demo2.bin 的数据说明如下：

1 个　长度为 999 的整数类型线性表-数据内容为 0 至 998

1 个　长度为 100 的实数类型线性表-数据内容为随机实数

1 个　长度为 388 的中文字符类型线性表-数据内容为短文

1.4.2 测试计划

测试计划包含 3 个模块：线性表管理菜单测试，线性表操作菜单测试，输入验证和文件输出测试。

测试模块 1：线性表管理菜单测试（如表 1-4 所示）

表 1-4　线性表管理菜单测试内容

测试序号	测试前总体情况描述	测试描述	测试操作	预计输出	测试文件
1	线性表数量已达最大值（100）	测试新建功能	提供合法输入，选择“新建”	新建失败，提示已满	Demo1.bin

（续表）

2	线性表数量为 3	测试加载功能	提供合法输入，选择“加载”，分别加载编号为 1, 2, 3 的线性表并显示内容	成功加载并显示编号为 1, 2, 3 的线性表内容	Demo2.bin
3	线性表数量为 3	测试删除功能	提供合法输入，选择“删除”，分别删除编号为，1, 2, 3 的线性表	成功删除编号为 1, 2, 3 的线性表，且管理菜单现有线性表数量显示为 0	Demo2.bin
4	线性表数量为 0	测试加载功能	提供合法输入，选择“加载”	加载失败，提示无可用线性表	Demo2.bin
5	线性表数量为 0	测试删除功能	提供合法输入，选择“删除”	删除失败，提示无线性表	Demo2.bin
6	线性表数量为 3	测试新建功能	提供合法输入，选择“新建”，新建一个“整数”类型的线性表，并返回查看	新建成功，管理菜单现有线性表数量显示为 4	Demo2.bin
7	线性表数量为 4	测试新建功能	提供合法输入，选择“新建”，新建一个“实数”类型的线性表，并返回查看	新建成功，管理菜单现有线性表数量显示为 5	Demo2.bin
8	线性表数量为 5	测试新建功能	提供合法输入，选择“新建”，新建一个“中文字符”类型的线性表，并返回查看	新建成功，管理菜单现有线性表数量显示为 6	Demo2.bin
9	线性表数量为 6	测试退出功能	提供合法输入，选择“退出”	延时退出，提示生成文本文件	Demo2.bin

测试模块 2：线性表操作菜单测试（如表 1-5 所示）

表 1-5　线性表操作菜单测试内容

测试序号	测试前线性表状态描述	测试描述	测试操作	预计输出	测试文件/线性表编号
1	表长为 999	测试判别是否为空功能	提供合法输入，选择“判别是否为空”	提示非空	Demo2.bin/1
2	表长为 999	测试长度功能	提供合法输入，选择“长度”	提示长度为 999	Demo2.bin/1
3	表长为 999	测试显示线性表元素功能	提供合法输入，选择“显示线性表元素”	显示 0 至 998 总计 999 个整数	Demo2.bin/1
4	表长为 999	测试获取确定位置元素功能	提供合法输入，选择“获取确定位置元素”，索引“501”	显示第 501 个元素 500	Demo2.bin/1
5	表长为 999	测试索引元素功能	提供合法输入，选择“索引元素”，分别使用 3 种比较函数，输入定位元素“750”	分别显示第 751 个元素 750，第 1 个元素 0，第 752 个元素 751	Demo2.bin/1
6	表长为 999	测试查找前驱元素功能	提供合法输入，选择“查找前驱元素”，分别输入定位元素“600”，“0”和“1500”	分别显示第 600 个元素 599，无匹配，无匹配	Demo2.bin/1
7	表长为 999	测试查找后继元素功能	提供合法输入，选择“查找后继元素”，分别输入定位元素“600”，“-5”和“998”	分别显示第 602 个元素 601，无匹配，无匹配	Demo2.bin/1
8	表长为 999	测试插入元素功能	提供合法输入，选择“插入元素”，于位序 1000 插入元素 999	插入成功，显示 0 至 999 总计 1000 个整数	Demo2.bin/1

（续表）

9	表长为 1000	测试插入元素功能	提供合法输入，选择“插入元素”	提示空间已满	Demo2.bin/1
10	表长为 100	测试删除元素功能	提供合法输入，选择“删除元素”，删除位序为97的元素	删除成功，提示已删除的元素为6.8	Demo2.bin/2
11	表长为 100	测试重置功能	提供合法输入，选择“重置”	提示重置成功	Demo2.bin/2
12	表长为 0	测试销毁功能	提供合法输入，选择“销毁”	提示已销毁	Demo2.bin/2
13	表被销毁	测试判别是否为空功能	提供合法输入，选择“判别是否为空”	提示线性表不存在	Demo2.bin/2
14	表被销毁	测试初始化功能	提供合法输入，选择“初始化”	提示初始化成功	Demo2.bin/2
15	表长为 0	测试判别是否为空功能	提供合法输入，选择“判别是否为空”	提示线性表为空	Demo2.bin/2
16	表长为 0	测试退出功能	提供合法输入，选择“退出”，并选择“加载”进行查看	提示线性表保存成功，查看到的描述为“实数类型长度为0的线性表”	Demo2.bin/2

测试模块 3：输入验证和文件输出测试（如表 1-6 所示）

表 1-6　输入验证和文件输出测试内容

测试序号	测试前总体情况描述	测试描述	测试操作	预计结果	测试文件
1	线性表数量为 3	测试文本文件输出功能	加载第 1 个线性表，删除其第 1 个元素，并多次退出，直至退出管理菜单，查看输出的文本文件	显示输出的文本文件名，打开文本文件后，其内容格式符合前述设计，总计显示 3 个线性表，第 1 个线性表的第 1 个元素不为 0	Demo2.bin

（续表）

2	线性表数量为 3	测试输入验证功能	进入任意菜单选项，提供 23 次不同类型的非法输入（非法输入类型如表 1-7 所示）	提示输入非法，可以重新输入	Demo2.bin

表 1-7　非法输入类型

序号	非法输入类型	用例
1	超大数字	123456789123456789, 6e400
2	过小数字	-123456789123456789, 6e-400
3	过多字符	12，a1, 过多，1st 开始
4	控制字符	Crtl+D, Crtl+G
5	数字+控制字符	1123+ Crtl+D, 1123+ Crtl+T
6	阻塞信号/文件尾标识	Crtl+Z
7	中断信号	Crtl+C
8	数字+阻塞信号/文件尾标识	1123+ Crtl+Z
9	数字+字符	123 始
10	字符+数字	无 89
11	字符+阻塞信号/文件尾标识	哦 Crtl+Z
12	阻塞信号/文件尾标识+控制字符	Crtl+Z+ Crtl+D+ Crtl+T
13	阻塞信号/文件尾标识+阻塞信号/文件尾标识	Crtl+Z+Crtl+Z
14	数字+字符+阻塞信号/文件尾标识+数字+字符	123 始+Crtl+Z+无 89
15	控制字符+阻塞信号/文件尾标识	Crtl+D+Crtl+Z
16	过长字符串	循环 4 次输入字母 a 至 z

1.4.3　测试结果

根据测试计划，测试结果包含 3 个部分：线性表管理菜单测试结果，线性表操作菜单测试结果，输入验证和文件输出测试结果。

线性表管理菜单测试结果如图 1-5 所示（见第 24 至 25 页）。

测试序号1

```
                    线 性 表 管 理 菜 单
------------------------------------------------
          1. 新建[100/100]        2. 加载
          3. 删除                 0. 退出
------------------------------------------------
****请选择你的操作[0~3]: 1
----存储空间::已满。
请按任意键继续. . .
```

测试序号2

```
----线性表内容如下:
----(元素总数: 999)
          0           1           2           3
          9          10          11          12
         18          19          20          21
         27          28          29          30
         36          37          38          39
         45          46          47          48

----线性表内容如下:
----(元素总数: 100)
    89.0857     88.2286     87.3714     86.5143
    81.3714     80.5143     79.6571        78.8
    73.6571        72.8     71.9429     71.0857
    65.9429     65.0857     64.2286     63.3714
    58.2286     57.3714     56.5143     55.6571

----线性表内容如下:
----(元素总数: 387)
    这几天心里颇不宁静。今晚在院子里坐着乘凉
亮渐渐地升高了，墙外马路上孩子们的欢笑，已经
，带上门出去。
```

测试序号3

```
****请输入线性表序号[0~100]: 1
----线性表::Sequence Number: ZERO to 998::删除成功。

                ****请输入线性表序号[0~99]: 2
                ----线性表::Random Floating::删除成功。

****请输入线性表序号[0~100]: 3
----线性表::《荷塘月色(节选)》朱自清::删除成功。
```

测试序号4

```
                    线 性 表 管 理 菜 单
------------------------------------------------
          1. 新建[0/100]          2. 加载
          3. 删除                 0. 退出
------------------------------------------------
****请选择你的操作[0~3]: 2
----无可用线性表。
```

测试序号5

```
                    线 性 表 管 理 菜 单
------------------------------------------------
          1. 新建[0/100]          2. 加载
          3. 删除                 0. 退出
------------------------------------------------
****请选择你的操作[0~3]: 3
----无线性表。
```

（续图）

测试序号2

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------
          1. 初始化                 7. 索引元素
          2. 销毁                   8. 查找前驱元素
          3. 重置                   9. 查找后继元素
          4. 判别是否为空          10. 插入元素
          5. 长度                  11. 删除元素
          6. 获取确定位置元素      12. 显示线性表元素
          0. 退出
------------------------------------------------------
****请选择你的操作[0~12]: 5
----线性表长度::999。
```

测试序号3

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------
          1. 初始化                 7. 索引元素
          2. 销毁                   8. 查找前驱元素
          3. 重置                   9. 查找后继元素
          4. 判别是否为空          10. 插入元素
          5. 长度                  11. 删除元素
          6. 获取确定位置元素      12. 显示线性表元素
          0. 退出
------------------------------------------------------
****请选择你的操作[0~12]: 12
----线性表内容如下:
----(元素总数: 999)
          0          1          2          3          4
          5          6          7          8          9
         10         11         12         13         14
         15         16         17         18         19
         20         21         22         23         24
         25         26         27         28         29
         30         31         32         33         34
         35         36         37         38         39
         40         41         42         43         44
         45         46         47         48         49
         50         51         52         53         54
         55         56         57         58         59
         60         61         62         63         64
         65         66         67         68         69
         70         71         72         73         74
         75         76         77         78         79
         80         81         82         83         84
         85         86         87         88         89
         90         91         92         93         94
         95         96         97         98         99

        960        961        962        963        964
        965        966        967        968        969
        970        971        972        973        974
        975        976        977        978        979
        980        981        982        983        984
        985        986        987        988        989
        990        991        992        993        994
        995        996        997        998
----显示::完毕。
```

（续图）

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 8
****请输入定位元素: 0
----元素前驱::无匹配。

                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 8
****请输入定位元素: 1500
----元素前驱::无匹配。
```

测试序号7

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 9
****请输入定位元素: 600
----匹配到的元素后继::[602]601。

                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 9
****请输入定位元素: -5
----元素后继::无匹配。
```

（续图）

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                7. 索引元素
          2. 销毁                  8. 查找前驱元素
          3. 重置                  9. 查找后继元素
          4. 判别是否为空          10. 插入元素
          5. 长度                  11. 删除元素
          6. 获取确定位置元素      12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 9
****请输入定位元素: 998
----元素后继::无匹配。
```

测试序号8

```
****请输入插入位置[1~1000]: 1000
****请输入插入元素: 999
----插入元素::成功。
----已完成插入
----线性表内容如下:
----(元素总数: 1000)
   0    1    2    3    4
   5    6    7    8    9
  10   11   12   13   14
  15   16   17   18   19
  20   21   22   23   24
  25   26   27   28   29
  30   31   32   33   34
  35   36   37   38   39
  40   41   42   43   44
  45   46   47   48   49
  50   51   52   53   54
  55   56   57   58   59
  60   61   62   63   64
  65   66   67   68   69
  70   71   72   73   74

 960  961  962  963  964
 965  966  967  968  969
 970  971  972  973  974
 975  976  977  978  979
 980  981  982  983  984
 985  986  987  988  989
 990  991  992  993  994
 995  996  997  998  999
----显示::完毕。
```

测试序号9

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                7. 索引元素
          2. 销毁                  8. 查找前驱元素
          3. 重置                  9. 查找后继元素
          4. 判别是否为空          10. 插入元素
          5. 长度                  11. 删除元素
          6. 获取确定位置元素      12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 10
----线性表::空间已满。
```

（续图）

测试序号12

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 2
----线性表::已销毁。
```

测试序号13

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 4
----线性表::不存在。
```

测试序号14

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 1
----线性表::初始化成功。
```

测试序号15

```
                    线 性 表 演 示 功 能 菜 单
------------------------------------------------------------
          1. 初始化                  7. 索引元素
          2. 销毁                    8. 查找前驱元素
          3. 重置                    9. 查找后继元素
          4. 判别是否为空           10. 插入元素
          5. 长度                   11. 删除元素
          6. 获取确定位置元素       12. 显示线性表元素
          0. 退出
------------------------------------------------------------
****请选择你的操作[0~12]: 4
---线性表状态::空。
```

（续图）

2: Random Floating

89.0857	88.2286	87.3714	86.5143	85.6571
84.8	83.9429	83.0857	82.2286	81.3714
80.5143	79.6571	78.8	77.9429	77.0857
76.2286	75.3714	74.5143	73.6571	72.8
71.9429	71.0857	70.2286	69.3714	68.5143
20.5143	19.6571	18.8	17.9429	17.0857
16.2286	15.3714	14.5143	13.6571	12.8
11.9429	11.0857	10.2286	9.37143	8.51429
7.65714	6.8	5.94286	5.08571	4.22857

3: 《荷塘月色(节选)》朱自清

这几天心里颇不宁静。今晚在院子里坐着乘凉，忽然想起日日走过的荷塘，在这满月的光里，总该另有一番样子吧。月亮渐渐地升高了，墙外马路上孩子们的欢笑，已经听不见了；妻在屋里拍着闰儿，迷迷糊糊地哼着眠歌。我悄悄地披了大衫，带上门出去。

测试序号2

```
****请输入插入位置[1~999]: 1
****请输入插入元素: 123456789123456789
****(输入非法)请重新输入插入元素: -123456789123456789
****(输入非法)请重新输入插入元素:

****请输入插入位置[1~101]: 1
****请输入插入元素: 6e400
****(输入非法)请重新输入插入元素: 6e-400
****(输入非法)请重新输入插入元素:

****请输入插入位置[1~388]: 1
****请输入插入元素: 12
****(输入非法)请重新输入插入元素: a1
****(输入非法)请重新输入插入元素: 过多
****(输入非法)请重新输入插入元素: 1st开始
****(输入非法)请重新输入插入元素:

****请选择你的操作[0~3]: 1
****请输入新建线性表名: ^D
****(表名过长或输入非法)请重新输入新建线性表名: ^G
****(表名过长或输入非法)请重新输入新建线性表名: 1123^D
****(表名过长或输入非法)请重新输入新建线性表名: 1123^T
****(表名过长或输入非法)请重新输入新建线性表名: ^Z
****(表名过长或输入非法)请重新输入新建线性表名: 1123^Z
****(表名过长或输入非法)请重新输入新建线性表名:

****请选择你的操作[0~3]: 123始
****(输入非法)请重新选择操作[0~3]: 无89
****(输入非法)请重新选择操作[0~3]: 哦^Z
****(输入非法)请重新选择操作[0~3]: ^Z^D^T
****(输入非法)请重新选择操作[0~3]: ^Z^Z
****(输入非法)请重新选择操作[0~3]: 123始^Z无89
****(输入非法)请重新选择操作[0~3]: ^D^Z
****(输入非法)请重新选择操作[0~3]:
```

（续图）

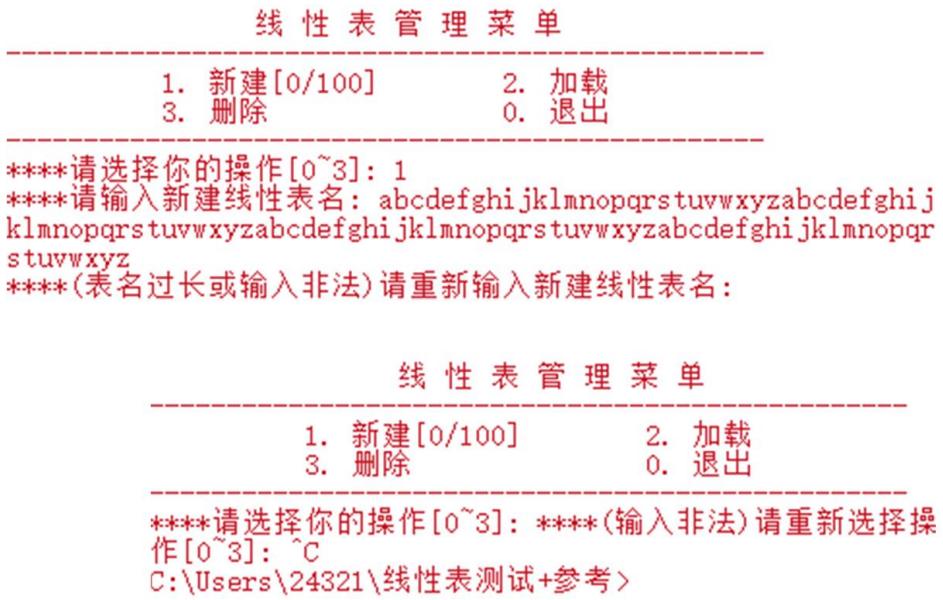

图 1-7　输入验证和文件输出测试结果

1.4.4　测试分析

根据实验要求，以测试计划所述的 3 个模块的测试结果为根本依据，现将测试结果陈述如下：

线性表管理菜单测试：总计 9 项测试内容全部通过，通过率为 100%。

线性表操作菜单测试：总计 16 项测试内容全部通过，通过率为 100%。

输入验证和文件输出测试：总计 2 项测试内容，通过 1 项，通过率为 50%。

失败测试项为输入验证和文件输出测试第 2 项，如表 1-8 所示。

表 1-8　输入验证和文件输出测试失败项

测试序号	测试前总体情况描述	测试描述	测试操作	预计结果	测试文件
2	线性表数量为 3	测试输入验证功能	进入任意菜单选项，提供 23 次不同类型的非法输入（非法输入类型如表 1-7 所示）	提示输入非法，可以重新输入	Demo2.bin

失败表现为：输入中断信号“Crtl+C”时，程序将连续显示 2 次错误提示信息，然后异常终止。

失败原因为：程序设计时未考虑对中断信号的处理。

另外，在打开演示程序时多次被杀毒软件拦截并隔离。其主要原因是程序可以用二进制模式打开并截短任何传入的无读写保护的文件，因此在使用演示程序时务必确保隔离使用。

1.5 实验小结

本次实验是第1次数据结构实验。演示程序编写、调试和修改总计历时9天。

本次实验是本实验员使用C++语言和Microsoft Visual Studio 2017开发环境编写较为完备的演示系统的第一次尝试。由于使用C++语言编程的实践经历尚不充足，在编程过程中多次出现结构性错误，以致开发效率严重下降。但幸运的是，在文件格式设计出现结构性错误后，本实验员及时更正了编程方向，主动舍弃了部分已写好的程序代码，并引入信息存储结构，重新设计文件格式；最终得以按时完成实验。

具体而言，本实验员认为线性表ADT的顺序表实现相对于演示系统的实现较为容易。但不可否认的是，演示系统的实现离不开线性表ADT，本实验员发现在编写演示系统程序代码时，需要多次复用线性表ADT，由此可见其重要性。

在本次实验中，本实验员不仅尝试实现线性表 ADT，也大量尝试使用线性表ADT。从结构决定功能的角度来看，在设计和实现线性表ADT时，应当先确定该ADT的功能蓝图，并以此作为设计导向。本实验员最初关于此方面的思考不足，导致了一部分盲目性工作的出现。

关于本次实验的最大难点，本实验员认为体现友好的交互设计上。为了实现较为完备的输入验证功能，本实验员参考了部分C++参考书和其他程序员博客，对输入流 std::cin 的状态掩码进行了学习和测试；最终以模板函数的形式较好地实现了输入验证。但对系统指令方面尚缺乏了解。

总而言之，本次实验是本实验员实践设计和使用线性表ADT的重要契机，同时也是融汇式学习C/C++语言程序设计的重要契机。在程序设计过程中，本实验员使用到的编程元素有结构、联合（最后放弃使用，改用多态指针）、类、模板类、函数、模板函数、枚举，尝试的设计方法有继承、包含、引用、多态、覆盖/划分、复用等。总体上，包含创作能力在内，本实验员各方面能力都得到了锻炼，本次实验是一次良好的实践契机。