

实验课程		数据结构实验				成绩	
实验项目		实验一线性表、堆栈和队列的操作与实现				指导老师	
<p>一、实验目的：</p> <p>1、领会单链表存储结构和掌握单链表中的各种基本运算算法设计；</p> <p>2、领会栈链存储结构和掌握栈链中的各种基本运算算法设计；</p> <p>3、领会环形队列存储结构和掌握环形队列中的各种基本运算算法设计；</p> <p>4、深入掌握单链表应用的算法设计；</p> <p>5、掌握栈应用的算法设计；</p> <p>二、使用仪器、器材</p> <p>微机一台</p> <p>操作系统：WinXP</p> <p>编程软件：C/C++编程软件</p> <p>三、实验内容及原理</p> <p>1、教材 P74 实验题 2：实现单链表的各种基本运算的算法</p> <p>编写一个程序 linklist.cpp，实现单链表中的各种基本运算和整体建表算法（假设单链表的元素类型 ElemType 为 char），并在此基础上设计一个程序 exp2-2.cpp 完成以下功能。</p> <p>(1) 初始化单链表 h。</p> <p>(2) 依次采用尾插法插入 a、b、c、d、e 元素。</p> <p>(3) 输出单链表 h。</p> <p>(4) 输出单链表 h 的长度。</p> <p>(5) 判断单链表 h 是否为空。</p> <p>(6) 输出单链表 h 的第 3 个元素。</p> <p>(7) 输出元素 a 的位置。</p> <p>(8) 在第 4 个元素位置上插入 f 元素。</p> <p>(9) 输出单链表 h。</p> <p>(10) 删除单链表 h 的第 3 个元素。</p> <p>(11) 输出单链表 h。</p> <p>(12) 释放单链表 h。</p>							

2、教材 P118 实验题 2：实现链栈的各种基本运算的算法

编写一个程序 linkstack.cpp，实现链栈（假设栈中元素类型 ElemType 为 char）的各种基本运算，并在此基础上设计一个程序 exp3-2.cpp 完成以下功能。

- (1) 初始化栈 s。
- (2) 判断栈 s 是否为空。
- (3) 依次进栈元素 a、b、c、d、e。
- (4) 判断栈 s 是否非空。
- (5) 输出出栈序列。
- (6) 判断栈 s 是否非空。
- (7) 释放栈。

3、教材 P118 实验题 3：实现环形队列的各种基本运算的算法

编写一个程序 sqqueue.cpp，实现环形队列（假设队列中元素类型 ElemType 为 char）的各种基本运算，并在此基础上设计一个程序 exp3-3.cpp 完成以下功能。

- (1) 初始化队列 q。
- (2) 判断队列 q 是否非空。
- (3) 依次进队元素 a、b、c。
- (4) 出队一个元素，输出该元素。
- (5) 依次进队元素 d、e、f。
- (6) 输出出队序列。
- (12) 释放队列。

4、教材 P77 实验题 12：用单链表实现两个大整数的相加运算

编写一个程序 exp2-12.cpp 完成以下功能。

- (1) 将用户输入的十进制整数字符串转化为带头结点的单链表，每个结点存放一个整数位。
- (2) 求两个整数单链表相加的结果单链表。
- (3) 求结果单链表的中间位，如 123 的中间位为 2，1234 的中间位为 2。

5、教材 P119 实验题 7：求解栈元素排序问题

编写一个程序 exp3-7.cpp，按升序对一个字符栈进行排序，即最小元素位于栈顶，最多只能使用一个额外的栈存放临时数据，并输出栈排序过程。

四、实验过程原始数据记录

第 1 题:

```
// StatusCode.h
#pragma once

typedef int Status;           // 函数类型，用于查询函数结果状态代码

// 函数结果状态
#define TRUE      1          // 真
#define FALSE    0          // 假
#define OK       1          // 成功
#define ERROR    0          // 错误
#define INFEASIBLE -1       // 不可行
#define OVERFLOW -2        // 溢出

// linklist.h
#pragma once

#include "StatusCode.h"
#include <iostream> // NULL定义在标准库中
#include <stdlib.h>
#include <time.h>
using namespace std;

typedef char ElemType; // 表里存放的数据类型

typedef struct LNode
{
    ElemType data;
    struct LNode* next;
} LNode, * LinkList;

// 线性表的基本操作
Status InitList(LinkList& L); // 初始化
Status Destroy(LinkList& L); // 销毁
Status ClearList(LinkList& L); // 清空
```

```

Status ListInsert(LinkList& L, int i, ElemType e); // 插入
Status ListDelete(LinkList& L, int i, ElemType& e); // 删除

Status IsEmpty(LinkList L); // 判空
Status ListLength(LinkList L); // 求长
int LocateElem(LinkList L, ElemType e); // 查找
Status GetElem(LinkList L, int i, ElemType& e); // 取值
ElemType GetElem(LinkList L, int i); // 取值, 返回所取元素

Status PrintList(LinkList L); // 打印输出链表

// linklist.cpp
#include "linklist.h"

// 初始化
Status InitList(LinkList& L)
{
    L = new LNode;
    L->next = NULL;
    if (!L)
        return ERROR;
    else
        return OK;
}

// 销毁
Status Destroy(LinkList& L)
{
    LNode* p;
    while (L)
    {
        p = L;
        L = L->next;
        delete p;
    }
    if (!L)
        return OK;
    else
        return ERROR;
}

// 清空

```

```

Status ClearList(LinkList& L)
{
    LNode* q, * p = L->next; // 注意!!! q为前驱指针, p为后驱指针
    while (p)
    {
        q = p;
        p = p->next;
        delete q;
    }
    L->next = NULL;
    return OK;
}

// 插入
Status ListInsert(LinkList& L, int i, ElemType e)
{
    int count = 0;
    LNode* p = L;
    if (i < 1)
        return ERROR;

    while (p->next && count < i - 1)
    {
        p = p->next;
        count++;
    }

    LNode* s = new LNode;
    s->data = e;
    s->next = p->next;
    p->next = s;
    return OK;
}

// 删除
Status ListDelete(LinkList& L, int i, ElemType& e)
{
    int count = 0;
    LNode* q = L;
    if (i < 1)
        return ERROR;
    while (q->next && count < i - 1)
    {
        q = q->next;
        count++;
    }

```

```

    LNode* p = q->next;
    q->next = p->next;
    e = p->data;
    delete p;
    return OK;
}

```

```

// 判空
Status IsEmpty(LinkList L)
{
    if (!L->next)
        return 1;
    else
        return 0;
}

```

```

// 求长
int ListLength(LinkList L)
{
    int count = 0;
    LNode* p = L->next;
    while (p)
    {
        p = p->next;
        count++;
    }
    return count;
}

```

```

/*
 * 查找:
 * 根据指定数据获取数据所在的 位置地址
 * 根据指定数据获取数据 位置序号
 */

```

```

int LocateElem(LinkList L, ElemType e)
{
    LNode* p = L->next;
    int count = 0;
    while (p)
    {
        count++;
        if (p->data == e)
            return count;
        p = p->next;
    }
    return ERROR;
}

```

```

}
// 取值
Status GetElem(LinkList L, int i, ElemType& e)
{
    int count = 0;
    LNode* p = L->next;
    while (p && count <= i)
    {
        count++;
        if (count == i)
        {
            e = p->data;
            return OK;
        }
        p = p->next;
    }
    return ERROR;
}

```

```

// 取值，返回所取元素
ElemType GetElem(LinkList L, int i)
{
    int count = 1;
    LNode* p = L->next;
    if (!p || i < 0)
        return ERROR;
    while (count < i)
    {
        p = p->next;
        count++;
    }
    return p->data;
}

```

```

// 打印输出链表
Status PrintList(LinkList L)
{
    if (!L)
    {
        cout << "链表已经被销毁!" << endl;
        return ERROR;
    }
    if (!L->next)

```

```

    {
        cout << "空表" << endl;
        return ERROR;
    }

    LNode* p = L->next;

    while (p)
    {
        cout << p->data << endl;
        p = p->next;
    }

    return OK;
}

// exp2-2.cpp
#include <iostream>
#include "linklist.h"

using namespace std;

int main()
{
    LinkList h = NULL;
    ElemType e;

    cout << "1. 初始化单链表h" << endl;
    InitList(h);

    char InsertElem[] = { 'a', 'b', 'c', 'd', 'e' };
    cout << endl << "2. 依次采用尾插法插入a、b、c、d、e元素" << endl;
    for (int i = 1; i < 6; i++)
        ListInsert(h, i, InsertElem[i - 1]);

    cout << endl << "3. 输出单链表h如下：" << endl;
    PrintList(h);

    cout << endl << "4. 输出单链表h的长度：" << ListLength(h) << endl;

    cout << endl << "5. 判断单链表h是否为空（1空/0非空）：" << IsEmpty(h) << endl;
}

```



```

cout << endl << "6. 输出单链表h的第3个元素: " << GetElem(h, 3) << endl;

cout << endl << "7. 输出元素a的位置: " << LocateElem(h, 'a') << endl;

cout << endl << "8. 在第4个元素位置上插入f元素" << endl;
ListInsert(h, 4, 'f');

cout << endl << "9. 插入后输出链表h如下: " << endl;
PrintList(h);

cout << endl << "10. 删除单链表h的第3个元素" << endl;
ListDelete(h, 3, e);
cout << endl << "被删除的元素为: " << e << endl;
cout << endl << "11. 删除后输出链表h如下: " << endl;
PrintList(h);

cout << endl << "12. 删除链表" << endl;
Destroy(h);
PrintList(h);

return 0;
}

```

第 2 题:

```

// StatusCode.h
#pragma once

typedef int Status; // 函数类型，用于查询函数结果状态代码

// 函数结果状态
#define TRUE 1 // 真
#define FALSE 0 // 假
#define OK 1 // 成功
#define ERROR 0 // 错误
#define INFEASIBLE -1 // 不可行

#define OVERFLOW -2 // 溢出

// linkstack.h
#include "StatusCode.h"

```

```

#define ElemType char

typedef struct StackNode
{
    ElemType data;
    struct StackNode* next;
}StackNode, *LinkStack;

Status InitStack(LinkStack& S);           // 初始化
Status StackEmpty(LinkStack S);          // 栈是否为空
Status Push(LinkStack& S, ElemType e);    // 入栈
Status Pop(LinkStack& S, ElemType& e);    // 出栈
ElemType GetTop(LinkStack S);            // 获取栈顶元素
Status DestoryStack(LinkStack& S);        // 销毁

// linkstack.cpp
#include "linkstack.h"
#include <iostream>
using namespace std;

// 初始化
Status InitStack(LinkStack& S)
{
    S = new StackNode;
    if (!S)
        exit(OVERFLOW);
    S->next = NULL;; //栈顶指针置空
    return OK;
}

// 销毁
Status DestoryStack(LinkStack& S)
{
    StackNode* p = S, * tp;
    while (p)
    {
        tp = p;
        p = p->next;
        delete tp;
    }
    S = NULL;
    return OK;
}

```

```

// 栈是否为空
Status StackEmpty(LinkStack S)
{
    if (!S->next)
        return OK;
    else
        return FALSE;
}

// 入栈
Status Push(LinkStack& S, ElemType e)
{
    StackNode* p = new StackNode;
    if (!p)
        return OVERFLOW;
    p->data = e;
    p->next = S->next;
    S->next = p;    // 移动栈顶指针
}

// 出栈
Status Pop(LinkStack& S, ElemType& e)
{
    if (!S->next)
    {
        cout << "栈空!" << endl;
        return ERROR;
    }
    e = S->next->data;    // 获取出栈元素
    StackNode* tp = S->next;
    S->next = S->next->next;    // 移动栈顶指针
    delete tp;    // 释放出栈元素空间
    return OK;
}

// 获取栈顶元素
ElemType GetTop(LinkStack S)
{
    if (!S->next)
    {
        cout << "栈空!" << endl;
        return ERROR;
    }
}

```

```

        return S->next->data;
    }

// exp3-2.cpp
#include "linkstack.h"
#include <iostream>
using namespace std;

int main()
{
    LinkStack s;
    ElemType e;

    cout << "1. 初始化栈s" << endl;
    InitStack(s);

    cout << endl << "2. 判断栈s是否为空(1空/0非空): " << StackEmpty(s) << endl;

    cout << endl << "3. 依次进栈元素a、b、c、d、e" << StackEmpty(s) << endl;
    ElemType elem[] = { 'a', 'b', 'c', 'd', 'e' };
    for (int i = 1; i < 6; i++)
        Push(s, elem[i - 1]);

    cout << endl << "4. 判断栈s是否为空(1空/0非空): " << StackEmpty(s) << endl;

    cout << endl << "5. 输出出栈序列:" << endl;
    while (!StackEmpty(s))
    {
        Pop(s, e);
        cout << "    " << e << endl;
    }

    cout << endl << "6. 判断栈s是否为空(1空/0非空): " << StackEmpty(s) << endl;
    cout << endl << "7. 释放栈" << endl;
    DestoryStack(s);
    return 0;
}

```

第 3 题:

```
// StatusCode.h
#pragma once

typedef int Status;           // 函数类型，用于查询函数结果状态代码

// 函数结果状态
#define TRUE 1               // 真
#define FALSE 0             // 假
#define OK 1                 // 成功
#define ERROR 0              // 错误
#define INFEASIBLE -1        // 不可行
#define OVERFLOW -2          // 溢出

// sqqueue.h
#include "StatusCode.h"

#define MaxSize 10           // 队列最大长度
#define ElemType char        // 队列元素的数据类型

// 循环队列的定义
typedef struct
{
    ElemType* base;          // 动态数组指针
    int front;               // 头指针，指向队头
    int rear;                // 尾指针，指向队尾
} SqQueue;

Status InitQueue(SqQueue& Q);           // 初始化队列
Status DestroyQueue(SqQueue& Q);        // 销毁队列
Status ClearQueue(SqQueue& Q);          // 清空队列
Status QueueEmpty(SqQueue Q);           // 队列判空
Status QueueLength(SqQueue Q);          // 队列求长
Status GetHead(SqQueue Q, ElemType& e); // 获取队头
Status EnQueue(SqQueue& Q, ElemType e); // 入队

Status DeQueue(SqQueue& Q, ElemType& e); // 出队
```

```

// sqqueue.cpp
#include "sqqueue.h"
#include <iostream>
using namespace std;

// 初始化队列
Status InitQueue(SqQueue& Q)
{
    Q.base = new ElemType[MaxSize]; // 分配动态空间
    // 如果base指针为NULL, 代表空间分配失败
    if (!Q.base)
    {
        cout << "队列初始化失败!" << endl;
        return OVERFLOW;
    }
    Q.front = Q.rear = 0; // 空间分配成功, 初始化队头与队尾指针
    return OK;
}

// 销毁队列
Status DestroyQueue(SqQueue& Q)
{
    delete Q.base; // 释放动态分配的空间
    Q.base = NULL; // 数组指针置空
    Q.front = Q.rear = 0; // 头尾置0, 队列空
    return OK;
}

// 清空队列
Status ClearQueue(SqQueue& Q)
{
    Q.front = Q.rear = 0; // 头尾置0, 队列空
    return OK;
}

// 队列判空
Status QueueEmpty(SqQueue Q)
{
    if (Q.front == Q.rear)
        return TRUE;
    else
        return FALSE;
}

// 队列求长
Status QueueLength(SqQueue Q)

```

```

{
    return (Q.rear - Q.front + MaxSize) % MaxSize;
}

// 获取队头
Status GetHead(SqQueue Q, ElemType& e)
{
    // 队列非空才可以获取队头
    if (Q.front != Q.rear)
    {
        e = Q.base[Q.front];
        return OK;
    }
    else
        return ERROR;
}

// 入队
Status EnQueue(SqQueue& Q, ElemType e)
{
    // 判断队是否已满，满则返回错误
    if ((Q.rear + 1) % MaxSize == Q.front)
    {
        cout << "队满!!!" << endl;
        return ERROR;
    }
    Q.base[Q.rear] = e; // 元素e入队尾
    Q.rear = (Q.rear + 1) % MaxSize; // 根据循环队列逻辑，使尾指针逻辑后移
    return OK;
}

// 出队
Status DeQueue(SqQueue& Q, ElemType& e)
{
    // 判断队是否已空，空则返回错误
    if (Q.front == Q.rear)
    {
        cout << "队空!!!" << endl;
        return ERROR;
    }
    e = Q.base[Q.front];
    Q.front = (Q.front + 1) % MaxSize; // 根据循环队列逻辑，使头指针逻辑后移
    return OK;
}

```



```

// exp3-3.cpp
#include <iostream>
#include "squeue.h"
#include "StatusCode.h"
using namespace std;

int main()
{
    ElemType e;
    ElemType a[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int length;

    cout << "1. 初始化队列q" << endl;
    SqQueue q;
    InitQueue(q);

    cout << endl << "2. 判断队列q是否非空 (1空/0非空): " << QueueEmpty(q) << endl;

    cout << endl << "3. 依次进队元素a、b、c: " << endl;
    for (int i = 0; i < 3; i++)
        EnQueue(q, a[i]);

    cout << endl << "4. 出队一个元素，输出该元素" << endl;
    DeQueue(q, e);
    cout << " 该元素为: " << e << endl;

    cout << endl << "5. 依次进队元素d、e、f: " << endl;
    for (int i = 3; i < 6; i++)
        EnQueue(q, a[i]);

    cout << endl << "6. 输出出队序列: " << endl;
    length = QueueLength(q);
    for (int i = 0; i < length; i++)
    {
        DeQueue(q, e);
        cout << " " << e << endl;
    }

    cout << endl << "12. 释放队列" << endl;
    DestroyQueue(q);
    return 0;
}

```


第 4 题:

```
// StatusCode.h
#pragma once

typedef int Status;           // 函数类型，用于查询函数结果状态代码

// 函数结果状态
#define TRUE 1               // 真
#define FALSE 0             // 假
#define OK 1                 // 成功
#define ERROR 0              // 错误
#define INFEASIBLE -1        // 不可行
#define OVERFLOW -2          // 溢出

// Linklist.h
#include "StatusCode.h"
#include <iostream> // NULL定义在标准库中
#include <stdlib.h>
#include <time.h>
using namespace std;

typedef int ElemType; // 表里存放的数据类型

typedef struct LNode
{
    ElemType data;
    struct LNode* next;
} LNode, * LinkList;

// 线性表的基本操作
Status InitList(LinkList& L); // 初始化
Status Destroy(LinkList& L);  // 销毁

Status IsEmpty(LinkList L);    // 判空
Status ListLength(LinkList L); // 求长
Status GetElem(LinkList L, int i, ElemType& e); // 取值

Status PrintList(LinkList L); // 打印输出链表
```

```
Status CreateList_Head(LinkList& L, ElemType e);
```

```
// linklist.cpp
```

```
#include "linklist.h"
```

```
// 初始化
```

```
Status InitList(LinkList& L)
```

```
{
```

```
    L = new LNode;
```

```
    L->next = NULL;
```

```
    if (!L)
```

```
        return ERROR;
```

```
    else
```

```
        return OK;
```

```
}
```

```
// 销毁
```

```
Status Destroy(LinkList& L)
```

```
{
```

```
    LNode* p;
```

```
    while (L)
```

```
    {
```

```
        p = L;
```

```
        L = L->next;
```

```
        delete p;
```

```
    }
```

```
    if (!L)
```

```
        return OK;
```

```
    else
```

```
        return ERROR;
```

```
}
```

```
// 判空
```

```
Status IsEmpty(LinkList L)
```

```
{
```

```
    if (!L->next)
```

```
        return 1;
```

```
    else
```

```
        return 0;
```

```
}
```

```
// 求长
```

```
int ListLength(LinkList L)
```

```

{
    int count = 0;
    LNode* p = L->next;
    while (p)
    {
        p = p->next;
        count++;
    }
    return count;
}

// 取值
Status GetElem(LinkList L, int i, ElemType& e)
{
    int count = 0;
    LNode* p = L->next;
    while (p && count <= i)
    {
        count++;
        if (count == i)
        {
            e = p->data;
            return OK;
        }
        p = p->next;
    }
    return ERROR;
}

// 打印输出链表
Status PrintList(LinkList L)
{
    if (!L)
    {
        cout << "链表已经被销毁!" << endl;
        return ERROR;
    }

    if (!L->next)
    {
        cout << "空表" << endl;
        return ERROR;
    }
}

```

```

    LNode* p = L->next;

    while (p)
    {
        cout << p->data;
        p = p->next;
    }

    return OK;
}

// 前插法创建链表
Status CreateList_Head(LinkList& L, ElemType e)
{
    LNode* s = new LNode;
    s->data = e;
    s->next = L->next;
    L->next = s;
    return OK;
}

// exp2-12.cpp
#include <iostream>
#include "linklist.h"

using namespace std;
#define MaxSize 100

void BigIntAdd(LinkList& opL, LinkList& opR, LinkList& result);
void GetMid(LinkList& result, ElemType& e);

int main()
{
    int len;
    ElemType e;
    LinkList opL, opR, result;
    InitList(opL);
    InitList(opR);
    InitList(result);

    char* str = new char[MaxSize];
    cout << "单链表实现两个大整数的相加运算" << endl;

```

```

    cout << "请输入左操作数：";
    cin >> str;
    len = strlen(str);
    opL->data = len; // 利用头节点存储链表长度信息
    for (int i = 0; i < len; i++)
        CreateList_Head(opL, int(str[i] - '0'));

    cout << "请输入右操作数：";
    cin >> str;
    len = strlen(str);
    opR->data = len; // 利用头节点存储链表长度信息
    for (int i = 0; i < len; i++)
        CreateList_Head(opR, int(str[i] - '0'));

    BigIntAdd(opL, opR, result);
    GetMid(result, e);

    cout << "结果单链表的中间位为：" << e << endl;

    Destroy(opL);
    Destroy(opR);
    Destroy(result);
    return 0;
}

void BigIntAdd(LinkList& opL, LinkList& opR, LinkList& result)
{
    int n = min(opR->data, opL->data);
    int out = 0; // 进位
    int temp = 0;
    // 分别建立左右操作数、结果链表的工作指针
    LNode* pL = opL->next;
    LNode* pR = opR->next;

    for (int i = 0; i < n; i++)
    {
        temp = pL->data + pR->data + out;
        out = temp >= 10 ? 1 : 0;
        CreateList_Head(result, temp % 10);
        pL = pL->next;
        pR = pR->next;
    }

    pL = pL ? pL : pR;

```

```

while (pL)
{
    temp = pL->data + out;
    out = temp >= 10 ? 1 : 0;
    CreateList_Head(result, temp % 10);
    pL = pL->next;
}

if (out)
    CreateList_Head(result, out);
cout << "结果为: ";
PrintList(result);
cout << endl;
}

void GetMid(LinkList& result, ElemType& e)
{
    int len = ListLength(result);
    GetElem(result, len / 2, e);
}

```

第 5 题:

```

// StatusCode.h
#pragma once

typedef int Status;           // 函数类型，用于查询函数结果状态代码
// 函数结果状态
#define TRUE 1                // 真
#define FALSE 0               // 假
#define OK 1                  // 成功
#define ERROR 0               // 错误
#define INFEASIBLE -1         // 不可行

#define OVERFLOW -2           // 溢出

// linkstack.h
#pragma once

#include "StatusCode.h"

#define ElemType int

typedef struct StackNode

```

```

{
    ElemType data;
    struct StackNode* next;
}StackNode, * LinkStack;

Status InitStack(LinkStack& S);           // 初始化
Status DestoryStack(LinkStack& S); // 销毁
Status StackEmpty(LinkStack S);    // 栈是否为空
Status Push(LinkStack& S, ElemType e); // 入栈
Status Pop(LinkStack& S, ElemType& e); // 出栈
ElemType GetTop(LinkStack S); // 获取栈顶元素

```

```

// linkstack.cpp
#include "linkstack.h"
#include <iostream>
using namespace std;

// 初始化
Status InitStack(LinkStack& S)
{
    S = new StackNode;
    if (!S)
        exit(OVERFLOW);
    S->next = NULL;; //栈顶指针置空
    return OK;
}

// 销毁
Status DestoryStack(LinkStack& S)
{
    StackNode* p = S, * tp;
    while (p)
    {
        tp = p;
        p = p->next;
        delete tp;
    }
    S = NULL;
    return OK;
}

```

```

// 栈是否为空
Status StackEmpty(LinkStack S)
{
    if (!S->next)
        return OK;
    else
        return FALSE;
}

// 入栈
Status Push(LinkStack& S, ElemType e)
{
    StackNode* p = new StackNode;
    if (!p)
        return OVERFLOW;
    p->data = e;
    p->next = S->next;
    S->next = p;    // 移动栈顶指针
}

// 出栈
Status Pop(LinkStack& S, ElemType& e)
{
    if (!S->next)
    {
        cout << "栈空!" << endl;
        return ERROR;
    }
    e = S->next->data;    // 获取出栈元素
    StackNode* tp = S->next;
    S->next = S->next->next;    // 移动栈顶指针
    delete tp;    // 释放出栈元素空间
    return OK;
}

// 获取栈顶元素
ElemType GetTop(LinkStack S)
{
    if (!S->next)
    {
        cout << "栈空!" << endl;
        return ERROR;
    }
    return S->next->data;
}

```



```

}

// exp3-7.cpp
#include <iostream>
#include "linkstack.h"
using namespace std;

#define MaxSize 100

int main()
{
    int len = 0;
    char str[MaxSize];
    LinkStack S, Stemp; // 字符栈, 辅助栈
    ElemType e, temp; // 出栈元素, 对应字符栈, 辅助栈
    // 初始化两个栈
    InitStack(S);
    InitStack(Stemp);

    // 输入一个字符串然后全部入栈
    cout << "求解栈元素排序问题" << endl;
    cout << "请输入:";
    cin >> str;
    len = strlen(str);
    for (int i = 0; i < len; i++)
        Push(S, int(str[i] - '0'));

    // 开始栈排序
    while (!StackEmpty(S)) // 栈S不空一直循环
    {
        Pop(S, e); // 出栈元素e
        cout << "栈S出栈" << e << ", ";
        while (!StackEmpty(Stemp))
        {
            temp = GetTop(Stemp);
            cout << "栈Stemp栈顶" << temp << endl;
            if (temp > e)
            {
                cout << "因为" << temp << ">" << e;
                cout << ", " << temp << "出栈Stemp, 入栈S" << endl;
                Pop(Stemp, temp);
                Push(S, temp);
            }
        }
        else

```

```

    {
        cout << "因为" << temp << "<" << e << ", Stemp出栈循环结束" << endl;
        break;
    }
}

// 辅助栈空直接入栈
Push(Stemp, e);
cout << e << "进栈Stemp" << endl;
}

cout << endl;
cout << "排序后: ";

// 将排序好的元素从辅助栈压回原栈
while (!StackEmpty(Stemp))
{
    Pop(Stemp, e);
    Push(S, e);
}

// 从原栈出栈输出
while (!StackEmpty(S))
{
    Pop(S, e);
    cout << e;
}

// 程序结束，销毁两个栈
DestoryStack(S);
DestoryStack(Stemp);

return 0;
}

```

五、实验结果及分析

1、
运行结果：

```
C:\Users\LinFeng\Desktop\无标题1.exe
1. 初始化单链表h
2. 依次采用尾插法插入a、b、c、d、e元素
3. 输出单链表h如下：
a
b
c
d
e
4. 输出单链表h的长度： 5
5. 判断单链表h是否为空（1空/0非空）： 0
6. 输出单链表h的第3个元素： c
7. 输出元素a的位置： 1
8. 在第4个元素位置上插入f元素
9. 插入后输出链表h如下：
a
b
c
f
d
e
10. 删除单链表h的第3个元素
    被删除的元素为： c
11. 删除后输出链表h如下：
a
b
f
d
e
12. 删除链表
链表已经被销毁！

-----
Process exited after 0.0454 seconds with return value 0
Press ANY key to exit...
```

运行结果正确

2、

运行结果：

选择Microsoft Visual Studio 调试控制台

1. 初始化栈s
2. 判断栈s是否为空 (1空/0非空): 1
3. 依次进栈元素a、b、c、d、e
4. 判断栈s是否为空 (1空/0非空): 0
5. 输出出栈序列:
e
d
c
b
a
6. 判断栈s是否为空 (1空/0非空): 1
6. 释放栈

C:\Users\LinFeng\iCloudDrive\课程资料\大二上学期\数
\exp3-2.exe (进程 26400) 已退出, 代码为 0。
按任意键关闭此窗口. . .

运行结果正确

3、

运行结果：

Microsoft Visual Studio 调试控制台

1. 初始化队列q
2. 判断队列q是否非空 (1空/0非空): 1
3. 依次进队元素a、b、c:
4. 出队一个元素, 输出该元素
该元素为: a
5. 依次进队元素d、e、f:
6. 输出出队序列:
b
c
d
e
f
12. 释放队列

C:\Users\LinFeng\iCloudDrive\课程资料\大
\exp3-3.exe (进程 18360) 已退出, 代码为 0。
按任意键关闭此窗口. . .

运行结果正确

4、

运行结果：

```
选择Microsoft Visual Studio 调试控制台

单链表实现两个大整数的相加运算
请输入左操作数：999999
请输入右操作数：99
结果为：1000098
结果单链表的中间位为：0

C:\Users\LinFeng\iCloudDrive\课程资料\大二上学期\数据结构\实验2\exp2-12.exe (进程 14716) 已退出，代码为 0。
按任意键关闭此窗口。 . . .
```

运行结果正确

5、

运行结果：

```
Microsoft Visual Studio 调试控制台

求解栈元素排序问题
请输入：891
栈S出栈1，1进栈Stemp
栈S出栈9，栈Stemp栈顶1
因为1<9，Stemp出栈循环结束
9进栈Stemp
栈S出栈8，栈Stemp栈顶9
因为9>8，9出栈Stemp，入栈S
栈Stemp栈顶1
因为1<8，Stemp出栈循环结束
8进栈Stemp
栈S出栈9，栈Stemp栈顶8
因为8<9，Stemp出栈循环结束
9进栈Stemp

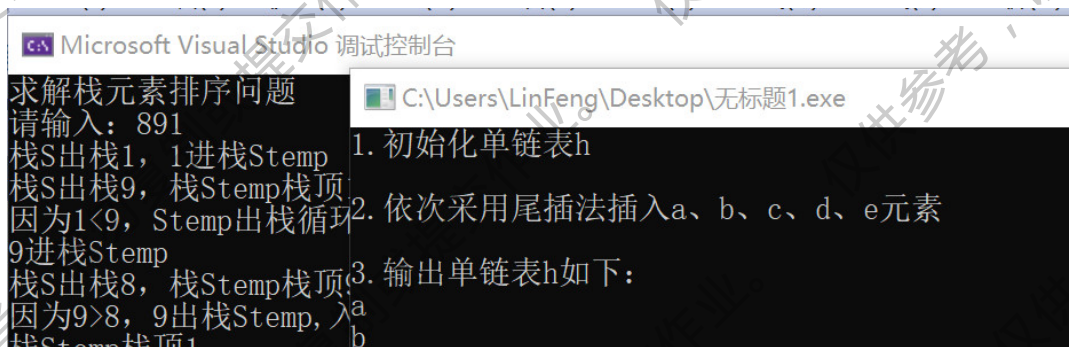
排序后：189
C:\Users\LinFeng\iCloudDrive\课程资料\大二上学期\数据结构\实验3\exp3-7.exe (进程 22492) 已退出，代码为 0。
按任意键关闭此窗口。 . . .
```

运行结果正确

实验体会与心得:

这次实验是一个非常简单的数据结构实验，主要是实现我们上课学习的线性链表、队列、栈基本数据结构。然后最后两题是利用栈和单链表进行一些应用，使我认识到了数据结构的魅力。

此外，在这个实验中也遇到了一些令人哭笑不得的问题，比如我在实验室所编写的vs项目在我的电脑上无法运行，出现一些奇怪的报错，所以第一题的我将所有分文件编写内容整合，然后使用 dev c++进行编译运行，故该程序截图与其他程序截图可能有所区别。



```
Microsoft Visual Studio 调试控制台
C:\Users\LinFeng\Desktop\无标题1.exe

求解栈元素排序问题
请输入: 891
栈S出栈1, 1进栈Stemp
栈S出栈9, 栈Stemp栈顶
因为1<9, Stemp出栈循环
9进栈Stemp
栈S出栈8, 栈Stemp栈顶
因为9>8, 9出栈Stemp, 入a
栈Stemp栈顶1

1. 初始化单链表h
2. 依次采用尾插法插入a、b、c、d、e元素
3. 输出单链表h如下:
```