实验 1　顺序表的操作及其应用

一、实验目的

1）掌握线性表的顺序存储结构；
2）熟练掌握顺序表基本算法的实现；
3）掌握利用线性表数据结构解决实际问题的方法和基本技巧；
4）按照实验题目要求独立正确地完成实验内容

二、实验内容

要求：数据元素类型ElemType 取整型int　或者char 。顺序存储实现如下算法：
1）创建一顺序表；
2）输出该顺序表；
3）在顺序表中查找第i 个元素，并返回其值；
4）在顺序表中第i 个元素之前插入一已知元素；
5）在顺序表中删除第i 个元素；
6）实现顺序表的合并。（选做）

源程序：

```
//A Sequential List  顺序表
#include <stdio.h>
#include <malloc.h>
#include <process.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
#define InitSize 100        /线性表存储空间的初始分配量
#define ListIncrement 10       /线性表存储空间的分配增量
typedef int ElemType;
typedef struct
{
      ElemType *elem;
      int length;
      int listsize;
}SqList;
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
typedef int Status;
Status InitList(SqList &L) //  初始化
```

```
{
        L.elem=(ElemType *)malloc(InitSize*sizeof(ElemType));
        if(!L.elem)
        exit(OVERFLOW);
        L.length=0;
        L.listsize=InitSize;
        return OK;
}
/求表长
int ListLength(SqList &L)
{
   return L.length;
}
/输入元素
int DataInput(SqList &L)
{

    int i=1,j=1;
            输入数据后，按“0”结束输入
    while(j)
  {

     if(j!=0)
    {
          L.elem[i]=j;
          L.length++;
          i++;
          if(i>InitSize)
               break;
     }
  }
     return FALSE;
}
/输出顺序表
Status ListTraverse(SqList L)
{
   ElemType *p;
   int i;
   p=L.elem;
   for(i=0;i<L.length;i++)


   return OK;
}
```

```
/查找元素
Status GetElem(SqList L,int i,ElemType &e)
{
    if(i<1||i>L.length)
            exit(ERROR);
    e=L.elem[i-1];
    return OK;
}
/插入元素
Status ListInsert(SqList &L,int i,ElemType e)
{
   ElemType *newbase,*q,*p;
   if(i<1||i>L.length+1)
      return     ERROR;
   if(L.length>=L.listsize)
   {
      newbase=(ElemType                                                    *)
realloc(L.elem,(L.listsize+ListIncrement)*sizeof(ElemType));
     if(!newbase)
         exit(OVERFLOW);
     L.elem=newbase;
     L.listsize=L.listsize+ListIncrement;
   }
  q=&(L.elem[i-1]);
  for(p=&(L.elem[L.length-1]);p>=q;--p)
        *(p+1)=*p;
         *q=e;
       ++L.length;
     return OK;
}
/删除元素
Status ListDeletSq(SqList &L,int i,ElemType &e)
{
    ElemType *p,*q;
    if(i<1||i>L.length)
    return ERROR;
    p=&(L.elem[i-1]);
    e=*p;
    q=L.elem+L.length-1;
    for(++p;p<=q;++p)
       *(p-1)=*p;
    --L.length;
    return OK;
}
```

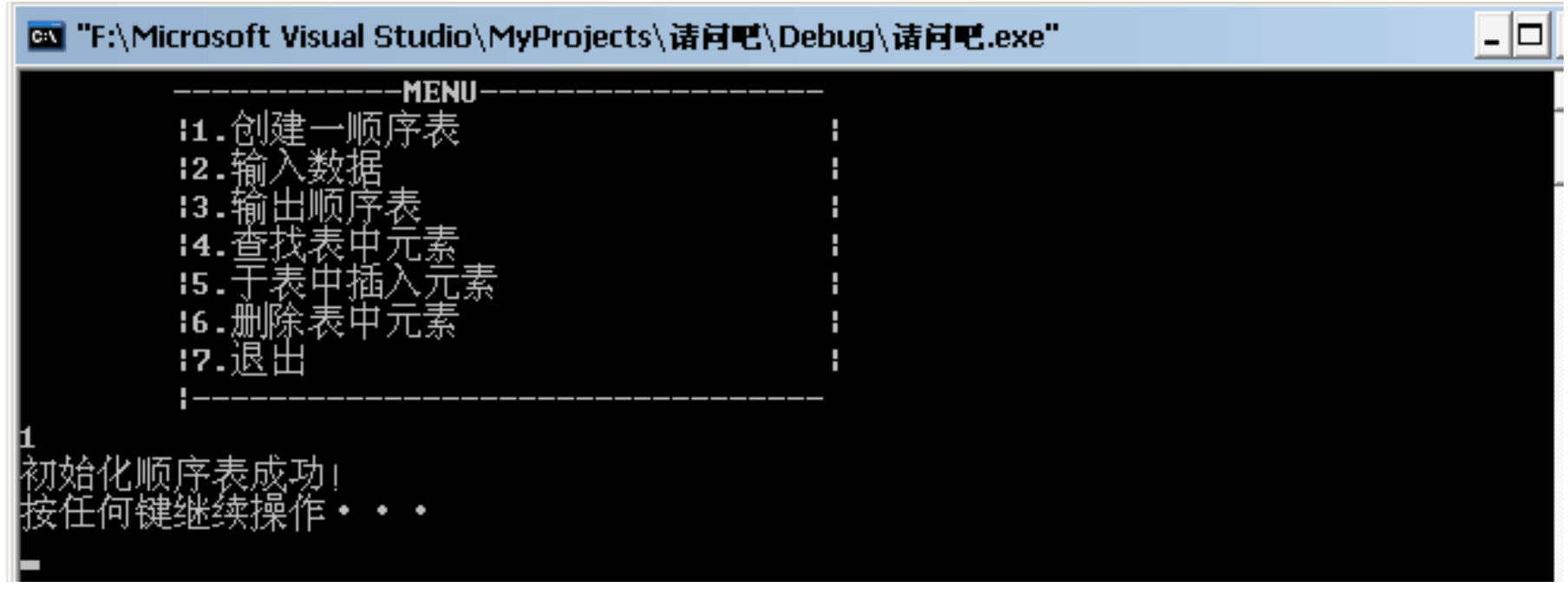
```
/生函数
int main()
{
      SqList L;
      ElemType e;
      char ch;
      int t;
    while(1)
{

                创建一顺序表
                输入数据
                输出顺序表
                查找表中元素
                于表中插入元素
                删除表中元素
                退出

fflush(stdin);
ch=getchar();
if(ch=='7')
   break;
switch(ch)
   {
        case '1': InitList(L);
                              初始化顺序表成功!
                            按任何键继续操作•••
                  getch();
                  break;
        case '2':DataInput(L);
                              数据输入成功!
                            按任何键继续操作•••
                  getch();
                  break;
case '3':ListTraverse(L);
                  getch();
                  break;
                      你查找是第几个元素:
                  fflush(stdin);

                  GetElem(L,t,e);
                        你查找的元素是:
                      按任何键继续操作•••
```

```
  getch();
  break;
        输入你要插入的元素：

  ListInsert(L,t,e);
            成功插入！
        按任何键继续操作 •••
  getch();
  break;
       你想删除第几个数据：

ListDeletSq(L,t,e);
            成功删除！
                按任何键继续操作 •••
                    getch();
                    break;
          default:break;
        }
  }
    return     FALSE;
}
```

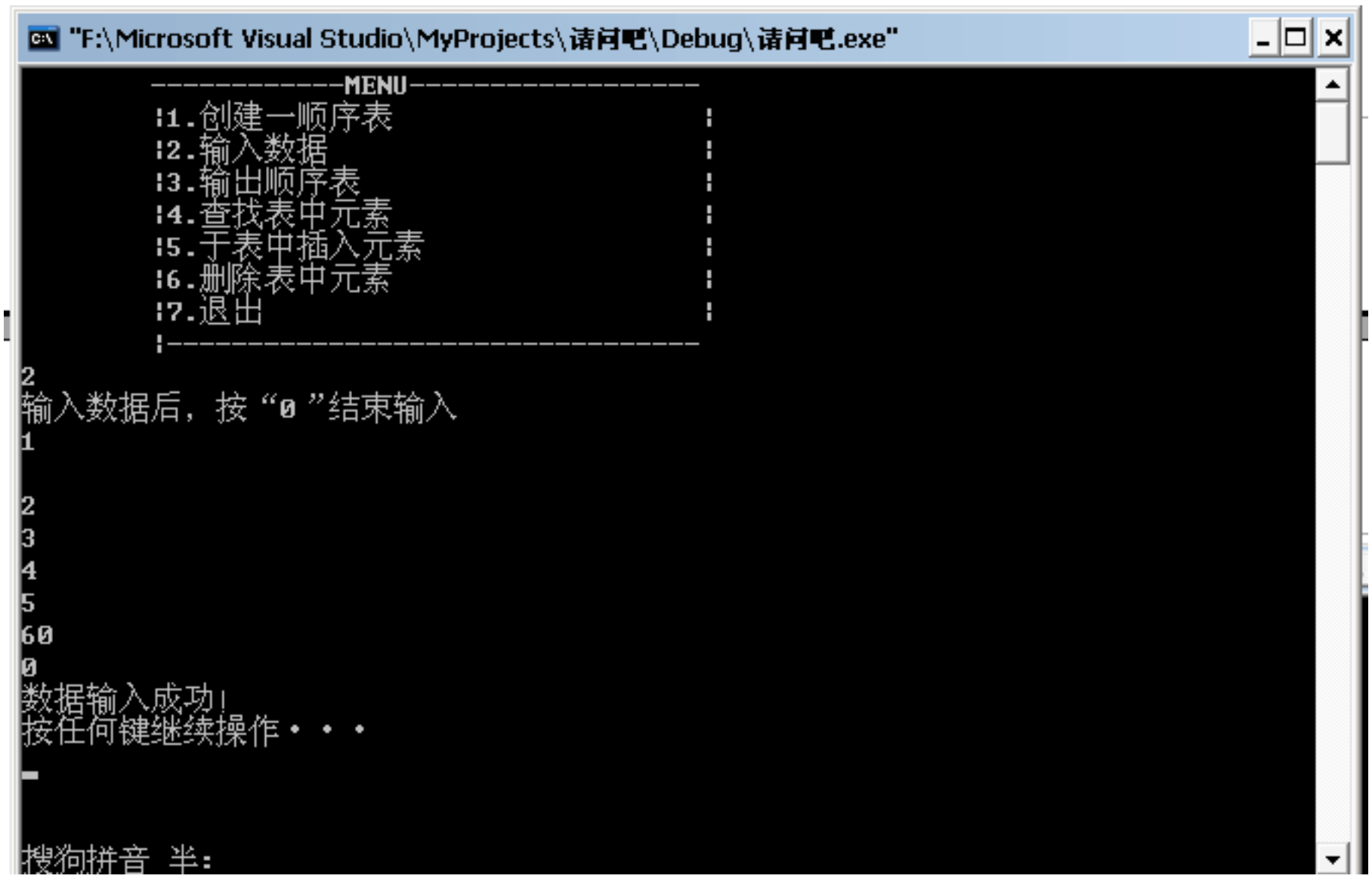
运行截图：

主菜单：

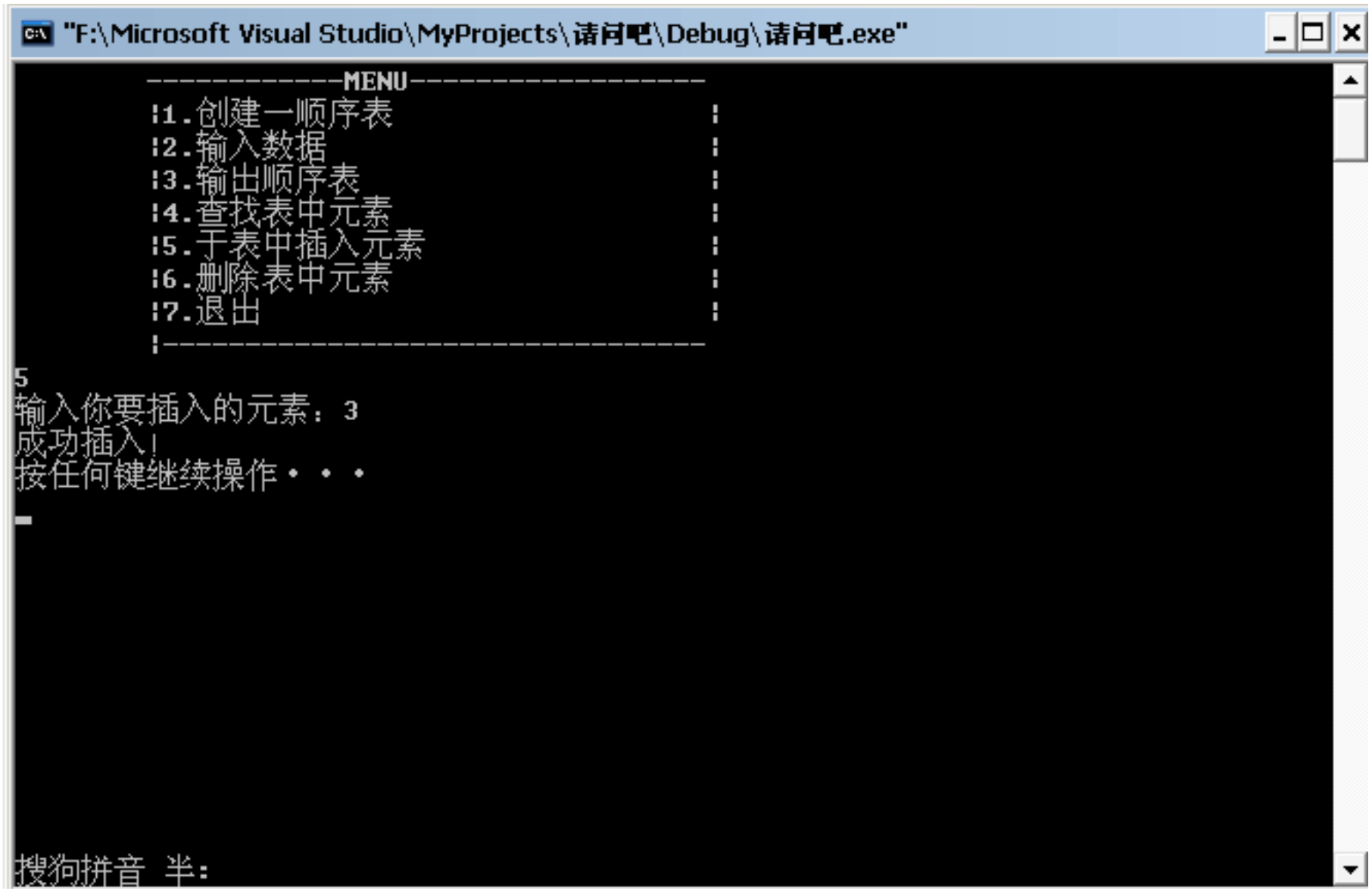
1. 创建顺序表；


"F:\Microsoft Visual Studio\MyProjects\请问吧\Debug\请问吧.exe"
----------MENU----------
!1.创建一顺序表
!2.输入数据
!3.输出顺序表
!4.查找表中元素
!5.于表中插入元素
!6.删除表中元素
!7.退出
!----------
1
初始化顺序表成功!
按任何键继续操作. . .

2. 输入数据：


"F:\Microsoft Visual Studio\MyProjects\请问吧\Debug\请问吧.exe"
----------MENU----------
!1.创建一顺序表
!2.输入数据
!3.输出顺序表
!4.查找表中元素
!5.于表中插入元素
!6.删除表中元素
!7.退出
!----------
2
输入数据后，按“0”结束输入
1
2
3
4
5
60
0
数据输入成功!
按任何键继续操作. . .
搜狗拼音 半:

3. 插入数据：



三、实验总结：

问题：

1. 刚开始接触数据结构时，完全不知道这门课程是学什么的，一脸茫然，通过反复看书，最后逐渐明白了其中的含义。
2. 有些 c 语言的函数和 c++语言函数不同。
3. 函数和指针的知识还不够好，不够牢固。

心得体会；

数据结构是一门重要的课程，万事开头难，刚开始学的时候是很难的，需要自己反复地思考和阅览书籍，才能解开一个个矛盾。C 语言是很基础的，数据结构要想学好，c 语言的最基本，特别是要学好里面的函数和指针知识，因为数据结构里面很多都是用到指针和函数的。

实验二 链表的操作及其应用

一、实验目的

了解单链表的基本概念、结构的定义及在单链表上的基本操作(插入、删除、查找以及线性表合并)，通过在 Turbo C 实现以上操作更好的了解书本上的内容并体会线性表的两种存储结构的区别。

二、实验内容

(1) 单链表的插入算法
(2) 单链表的删除算法
(3) 循环链表的插入和删除算法（选做）

源程序：

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>
typedef int ElemType;
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
typedef int Status;
typedef struct LNode
{
    ElemType data;
    struct LNode *next;
```

```
}LNode,*LinkList;
Status CreateList(LinkList &L,int n)//      创建链表
{
   L=(LinkList)malloc(sizeof(LNode));//  生成带头结点的链表
   LinkList head,p;
   int i;
   if(!L)
   {
                  创建失败!
        return FALSE;
   }
   L->next=NULL;
   head=L;
   for(i=0;i<n;i++)
   {
      p=(LinkList)malloc((sizeof(LNode)));// 生成新结点
              请输入结点值:
                                  输入新结点值
      p->next=NULL;// 尾插法，插入新结点
      head->next=p;
      head=p;
   }
    return TRUE;
}
Status GetElem_L(LinkList L,int i,ElemType &e)
{
    int j=1;
    LinkList p;
    p=L->next;// 使 p 指向第一个结点
    while(p&&j<i)
    {
       p=p->next; // 后移
      ++j;
    }
    if(!p||j>i)// 第 i 个元素不存在
    {
                  取元素失败!
        return ERROR;
    }
    e=p->data;// 取第 i 个元素
    return OK;
}
Status ListInsert_L(LinkList &L,int i,ElemType e)//     在带头结点的单链表 L 中的第 i 个位置之前插入元素 e
```

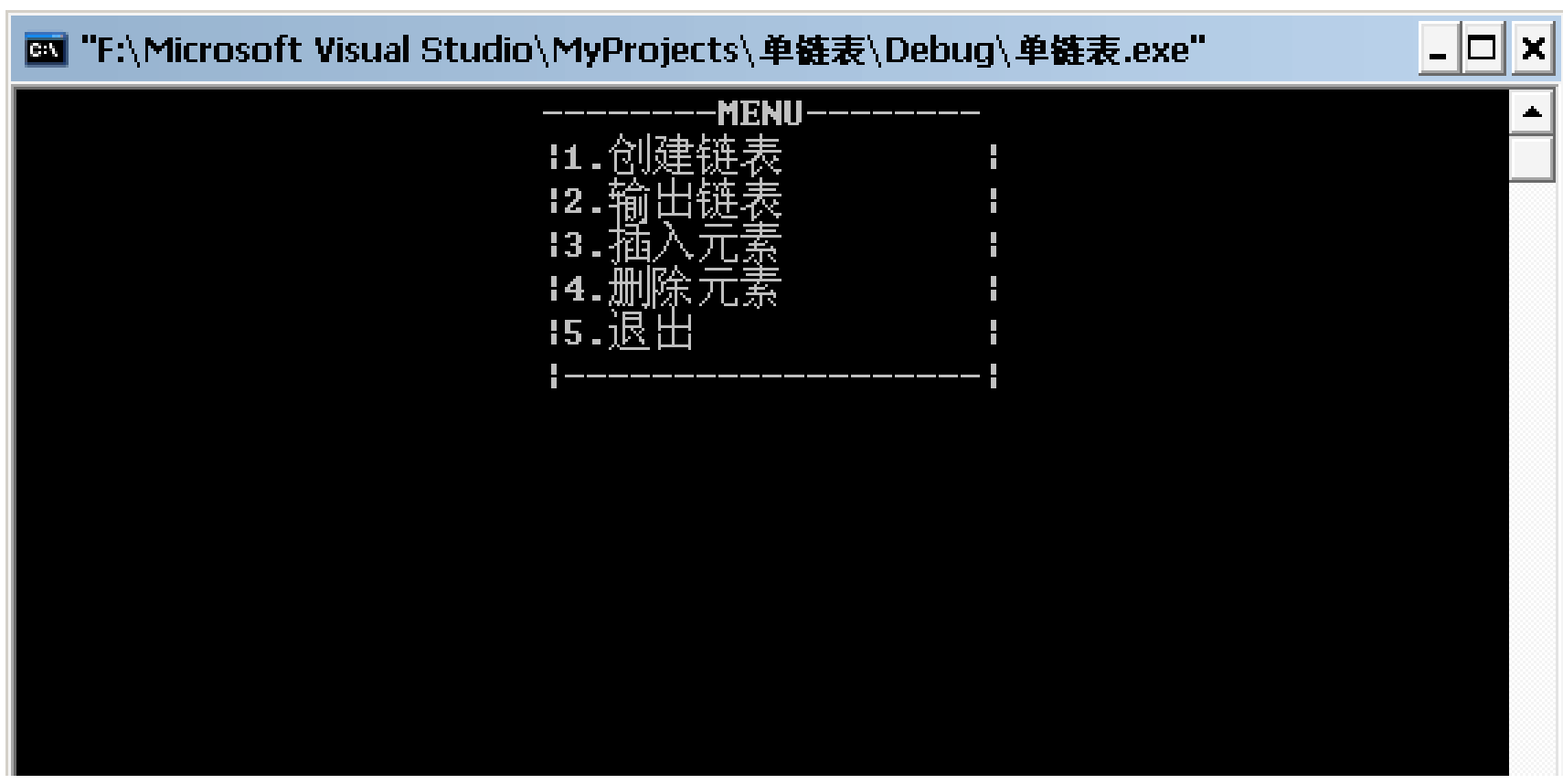
```
 LinkList p,s;
 p=L;
  int j=0;
while(p&&j<i-1)// 寻找第 i-i 个结点
{
    p=p->next;
    ++j;
}
if(!p||j>i-1)
{
                插入失败!
    return ERROR;
        }
    s=(LinkList)malloc(sizeof(LNode));// 生成新结点指向要插入的元素
    s->data=e;
    s->next=p->next;
    p->next=s;
    return OK;
}
Status ListDelete_L(LinkList L,int i,ElemType &e)//    在带头结点的单链表 L 中，删除第
i 个元素，并由 e 返回其值
{
   LinkList p,q;
   int j=0;
   p=L;
   while(p->next&&j<i-1)// 寻找第 i 个结点，并令 p 指向其前驱
   {
        p=p->next;
        ++j;
   }
   if(!p||j>i-1)
   {
                删除元素失败!
        return ERROR;
   }/删除位置不合理
   q=p->next;              /删除并释放结点
   p->next=q->next;
   e=q->data;
   free(q);
   return OK;
}
Status OutputList(LinkList L)
{
```

```
    LinkList p;
    for(p=L->next;p!=NULL;p=p->next)
    {

    }

    return OK;
}
main()
{
   char ch;
   int n,i;
   int e;
   LinkList L;
   while(1)
   {


                    创建链表
                    输出链表
                    插入元素
                    删除元素
                    退出

   ch=getchar();
   if(ch=='5')
       break;
   switch(ch)
   {
                            请输入你要创建的结点数:

                        每输入一个值后按 ENTER 继续
    CreateList(L,n);
                创建成功!
    break;
            单链表的值是:
    OutputList(L);
   getch();
   break;
            链表是:
    OutputList(L);
                请输入你要在第几个元素前插入元素:

                请输入你要插入的元素值:
```

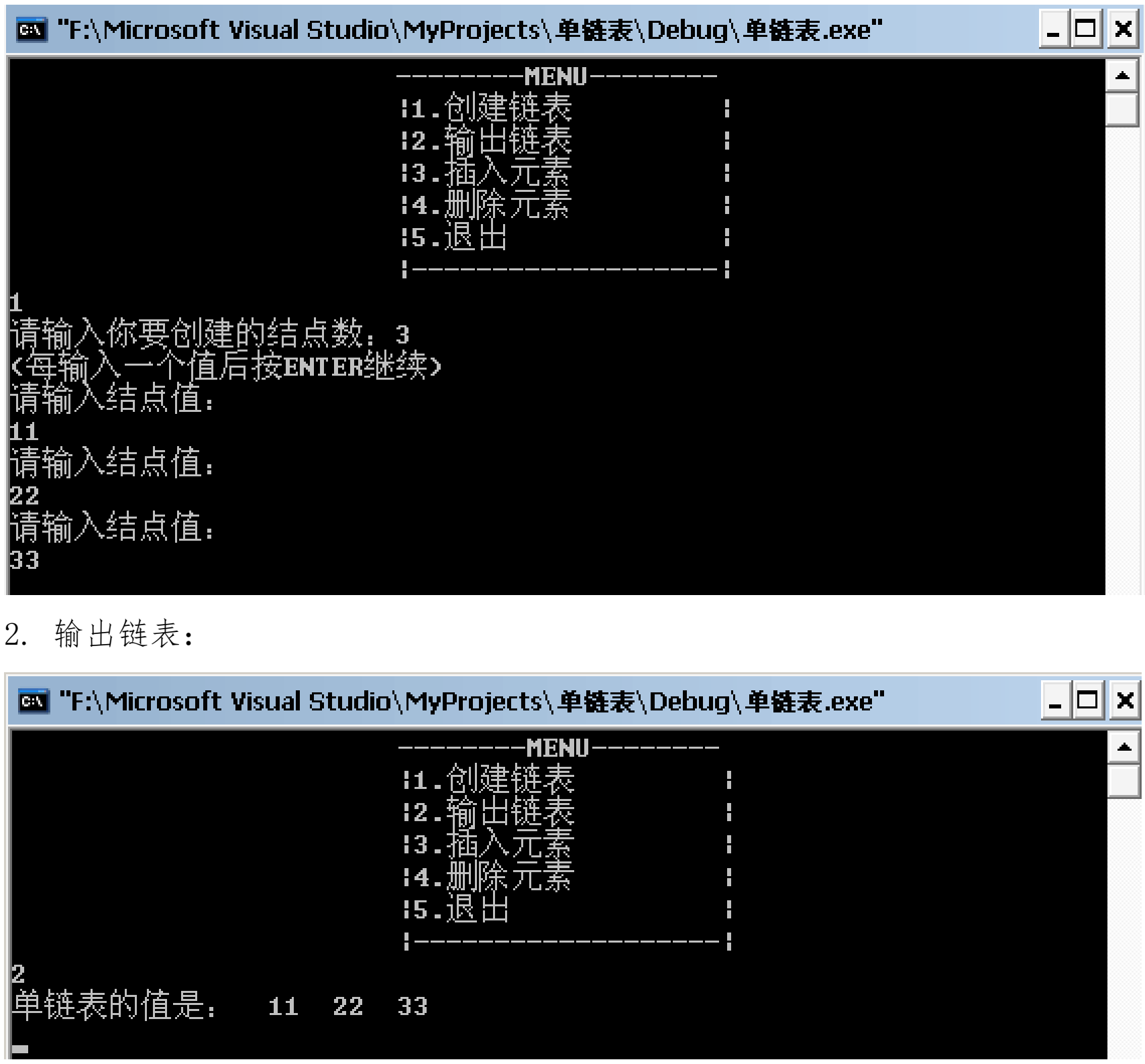
```
 ListInsert_L(L,i,e);
            插入成功!
            新单链表是:
   OutputList(L);
getch();
break;
          链表是:
 OutputList(L);
            你要删除第几个元素:

 ListDelete_L(L,i,e);
            删除成功!
                        新单链表是:
                    OutputList(L);
                  getch();
                  break;
          default:break;
    }
    }
    getch();
  return OK;
}
```

运行截图:

主菜单:

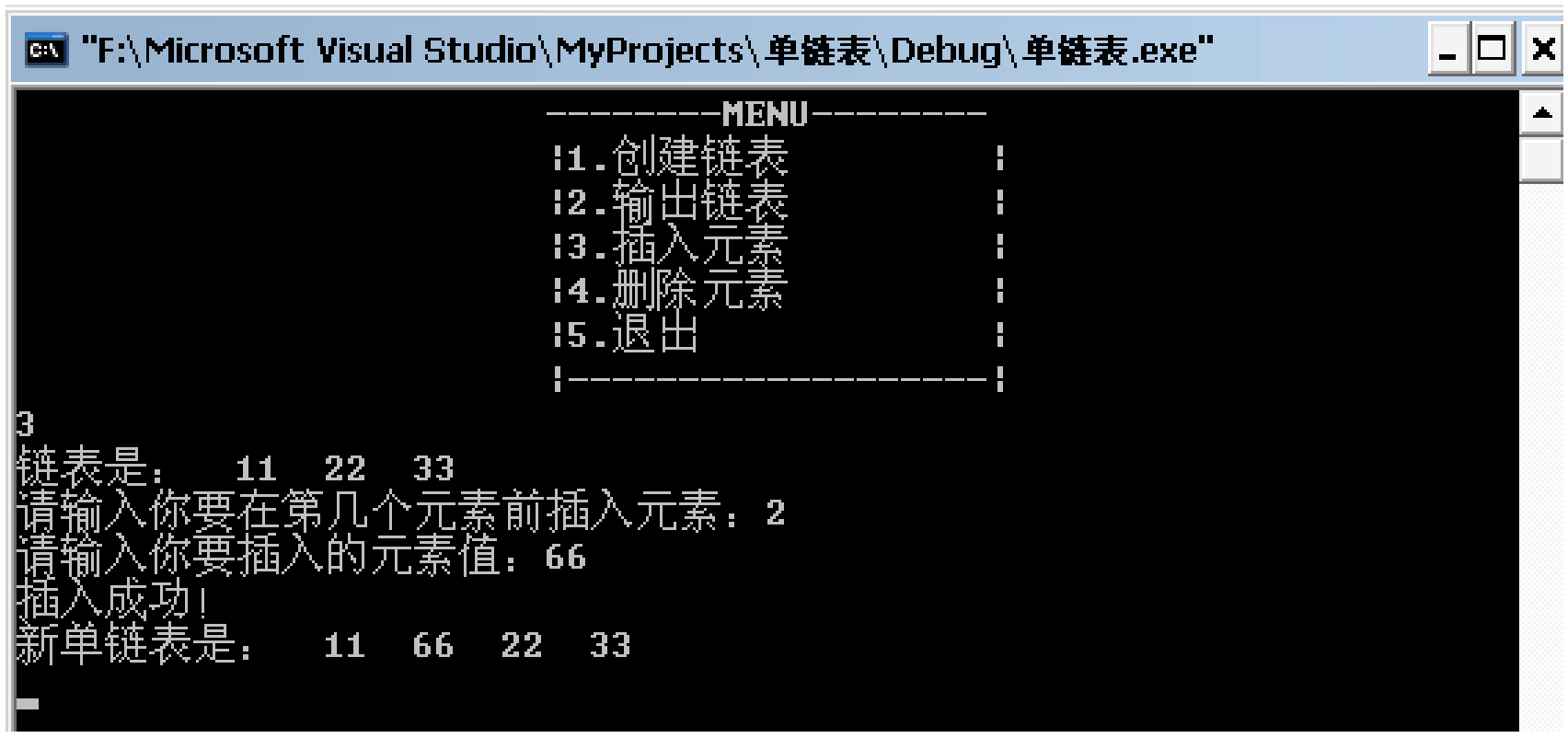


1. 创建链表:



2. 输出链表：

3. 插入元素：



4. 删除链表：

三、实验总结：

问题：

1. 创建链表时，运用前插法，使得最后输出时是反过来输出的；
2. 在调用函数 ListInsert_L(L,i,e); 时忘记写输入 e 的语句(就是要插入的元素)，使得在最后输出了一个负数或者说是乱码；

解决方法：1.通过看书和上网学到了尾插法，就是说尾插法的作用是使得最后输出时的数值是按照输入的顺序输出的；

3. 第二个问题的解决方法是最后加上了 这个语句，这样才不会出现乱码，输出如我所愿；

心得体会：通过对实验一的学习后，我对数据结构有了更深刻的认识，使得我在做实验的过程中速度加快了很多，掌握了各种函数的应用，能够更深刻地了解单链表的操作及其应用。

实验三 栈的操作及其应用

一、实验目的

了解栈的概念、栈的特性、在两种存储结构上如何实现栈的基本操作以及栈在程序设计中的应用。通过在 Turbo C中实现顺序栈的插入和删除加深理解顺序栈的意义。

二、实验内容

(1) 顺序栈的进栈、出栈算法

(2) 链式栈的进栈、出栈算法（选做）

源程序 code：

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
```

```
#define chushi 100
#define zengliang 10
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
typedef int Status;
typedef struct
{
      selemtype *base;
      selemtype *top;
      int stacksize;
}sqstack;
Status initstack(sqstack &s)      /初始化栈，构造一个空栈;
{
   s.base=(selemtype *)malloc(chushi* sizeof(selemtype));
   if(!s.base)
   {
                存储分配失败
        return FALSE;
   }
   s.top=s.base;
   s.stacksize=chushi;
   return TRUE;
}
Status gettop(sqstack s,selemtype &e)//   取栈顶元素
{
      if(s.top==s.base)
      {
                栈为空
          return FALSE;
      }
      e=*(s.top-1);
      return TRUE;
}
Status push(sqstack &s,selemtype e)//   插入元素    进栈
{
      if(s.top-s.base>=s.stacksize)// 栈满
      {
          s.base=(selemtype *)realloc(s.base,(s.stacksize+zengliang)*sizeof(selemtype));
          if(!s.base)
          {
```

```
                    存储分配失败!
                return FALSE;
            }
            s.top=s.base+s.stacksize;
            s.stacksize+=zengliang;
        }
        *s.top++=e;
        return TRUE;
}
Status pop(sqstack &s,selemtype &e)//   删除元素    出栈
{
   if(s.top==s.base)
    {
                栈为空!
        return FALSE;
    }
   e=*--s.top;
   return TRUE;
}
stacklength(sqstack &s) //  初始条件：栈 S 已存在     操作结果：返回 S 的元素个数，即
栈的长度
{
      return s.top-s.base;
}

 Status output_stack(sqstack s)
{
       selemtype *p;
      int i;
      p=s.base;
      if(s.top==s.base)
      {
                栈不存在
          return FALSE;
      }
      for(i=0;i<stacklength(s);i++)
      {

      }

      return TRUE;
}
Status clearstack(sqstack s)
{
```

```
    s.top=s.base;     //s.top == s.base  作为顺序栈空的标记
     return OK;
}

main()
{
     sqstack s;
    char ch;
    selemtype e;
    while(1)
    {
                    初始化顺序栈
                    输入栈的元素
                    输出顺序栈
                    进栈
                    出栈
                    清空顺序栈
                     退出

ch=getchar();
if(ch=='7')
     break;
switch(ch)
{
   case '1':initstack(s) ;
                          初始化成功
                         按任何键继续••
            getch();
            break;
                 顺序栈是:
output_stack(s);
                    请逐个输入数据)□你要输入的元素是:

push(s,e);
                 成功输入数据
                 顺序栈是:
output_stack(s);
                 按任何键继续••
getch();
break;
                 顺序栈的值是:
 output_stack(s);
```

```
            按任何键继续 ••
getch();
break;
          顺序栈的值是:
  output_stack(s);
            进栈的元素:

  push(s,e);
          进栈成功
             顺序栈的值是:
  output_stack(s);
          按任何键继续 ••
getch();
                    break;
                          顺序栈的值是:
                     output_stack(s);
                     pop(s,e);
                           出栈成功
                              现在顺序栈的值是:
                     output_stack(s);
                           按任何键继续 ••
                    getch();
                    break;
          case '6': clearstack(s);
                               清空完毕
                              按任何键继续 ••
                      break;
      }
      }
          return OK;
}
```

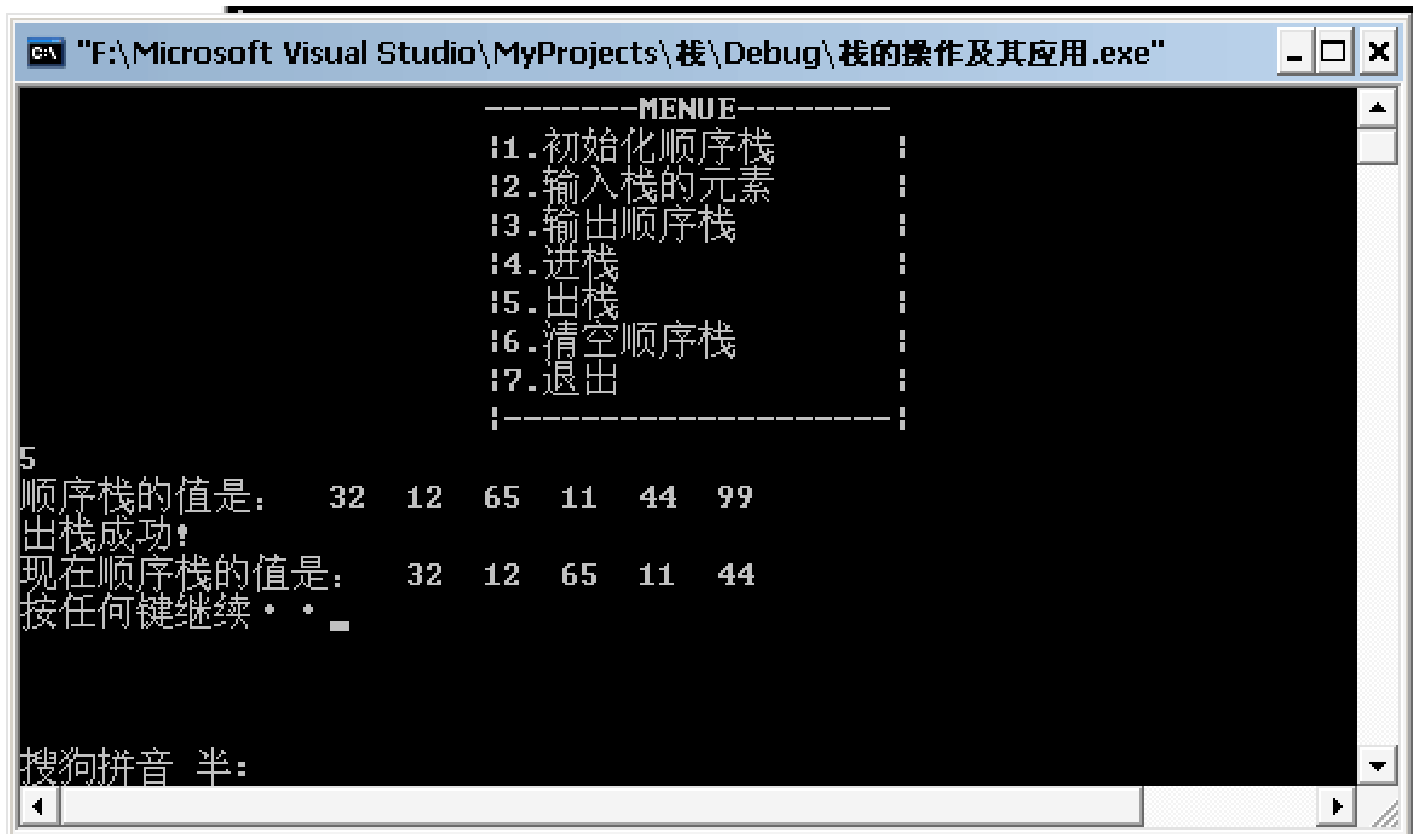
运行截图:

主菜单:

1. 初始化顺序表：

2.进栈：

3.出栈：



三、实验总结：

问题：

（1）：在进栈和输入的时，忘记了判断栈是否为满栈；在输出和出栈时，忘记判断栈是否为空。

（2）：有时运用了太多的 getch（）函数，使得程序运行的有点慢；

实验心得：

（1）：掌握了栈这种抽象数据类型的特点，并能在相应的应用任务中正确选用它；

总的来说，栈是操作受限的线性表，是限定仅在表尾进行插入或删除操作的线性表。因此，对栈来说，表尾端有其特殊含义，称为栈顶（top），相应地，表头端称为栈底（botton ）；

栈又称为后进先出（Last In First Out ）的线性表,简称 LIFO结构，因为它的修改是按后进先出的原则进行的。

（2）： 加上这个实验，我已经学了线性表（顺序表，单链表）和栈，知道它们都是线性表，而且对以后的学习有很大的作用，可以说这是学习以后知识的总要基础；

实验四 队的操作及其应用

一.实验目的：

了解队列的概念、队列的特性、在两种存储结构上如何实现队列的基本操作以及队列在程序设计中的应用。通过在 Turbo C 中实现队列的插入和删除加深理解链队列和循环队列的意义。

二、实验内容：

(1) 链队列的进队和出队算法

源程序 code:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
typedef int Status;
typedef int QElemType;
typedef struct QNode   //        结点类型
{
  QElemType data;
  struct QNode *next;
}QNode,*QueuePtr;
typedef struct
{
  QueuePtr front;//        队头指针
  QueuePtr rear;//        对尾指针
}LinkQueue;
Status InitQueue(LinkQueue &Q)//      初始化链队列     构造一个空队列
{
 Q.front=Q.rear=(QueuePtr)malloc(sizeof(QNode));
 if(!Q.front)
   {
                存储分配失败
     exit(OVERFLOW);
   }
 Q.front->next=NULL;
 return OK;
}
Status EnQueue(LinkQueue &Q,QElemType e)//     在队尾插入元素e
{
     QNode *p;
     p=(QueuePtr)malloc(sizeof(QNode));
     if(!p)
      {
                存储分配失败
       exit(OVERFLOW);
      }
     p->data=e;
     p->next=NULL;
```

```
        Q.rear->next=p;
        Q.rear=p;
        return OK;
}
Status DeQueue(LinkQueue Q,QElemType &e)//     在队头删除元素
{
      QNode *p;
  if(Q.front==Q.rear)
    {
                    队列为空
        return ERROR;
    }
  p=Q.front->next;
  e=p->data;
  Q.front->next=p->next;
  if(Q.rear==p)
        Q.rear=Q.front;//       如果被删的是最后一个元素，则为指针丢失，因此为为指针
重新赋值（指向头结点）
  free(p);
  return OK;
}
Status OutputQueue(LinkQueue Q)//      输出元素
{
      QNode *p;
   if(Q.front==Q.rear)
     {
                    队列为空
        return ERROR;
     }
   for(p=Q.front->next;p!=NULL;p=p->next)
       {
        }

    return OK;
}
Status DestroyQueue(LinkQueue &Q)//      销毁队列
{
  while(Q.front)
     {
        Q.rear=Q.front->next;
        free(Q.front);
        Q.front=Q.rear;
     }
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/575200021140011312