**数据结构课程设计**

班级：信管一班

姓名：高夕毓

学号：1513490428

日期：2017年5月26日

1. 各题目的编译、运行情况
2. 编译情况：所有项目均编译通过
3. 运行情况：1.一元多项式的表示和相加（链表、建立、相加、输出）、

2．表达式括号匹配检验（压栈、出栈）3.数制的转换（十进制到二进制转换）4.约瑟夫环（循环单链表）6.将两个有序线性表合并成一个有序线性表，并去掉重复元素。7.设有一个线性单链表，其结点值均为正整数，且按值从大到小链接。试写出一个算法，将该线性单链表分解成两个线性单链表，其中一个链表中的结点值均为奇数，而另一个链表中的结点值均为偶数，且这两个链表均按值从小到大链接。9．线性表综合题。10.二叉树应用运行正常。

5．表达式求解运行时如果计算中出现负数就无法运算，而且输入表达式的时候因为用的是getchar所以无法读出两位数以上的数字进行计算。

8．已知二叉树的后序遍历和中序遍历序列，构造对应的二叉树，并非递归前序遍历该二叉树。运行如果是数字数组可以，字符数组无法实现。

11．飞机票预订系统。无法有效添加飞机节点，需要全部重置。并且最大乘客数只有25。

1. 课程设计内容描述
2. 一元多项式的表示和相加（链表、建立、相加、输出）

<1>解题思路：先建立一个结构体，结构体中包含系数与指数的数据域以及next的指针域。将每个结构体定义为一个节点，在通过指针域链接来建立链表。表示部分，通过从头结点遍历至尾节点的数据域来确定多项式。而相加部分则先利用冒泡排序以指数为基准由小到大将多项式排序，然后由小到大同时遍历两个链表，相同指数则系数相加插入新的链表，不同指数则指数小的先插入链表，大的后插入。最后将剩余节点全部插入新的链表，即完成相加。

<2>函数调用图：

开始

相加放入L3

冒泡排序

表示L1，L2

建立链表L1，L2，L3

结束程序

表示L3

各函数功能:

int createPoly(Poly \*L) 创建多项式链表

int printPolynoimal(Poly \*L) 通过从头结点遍历至尾节点输出链表

int sortPolynoimal(Poly \*L) 对节点进行冒泡排序，由小至大。

int addPolynoimal(Poly \*L1, Poly \*L2, Poly \*L3) 将L1链表和L2链表相加组合成L3链表

int main() 主函数,用来流程控制，建立L1，L2，L3链表

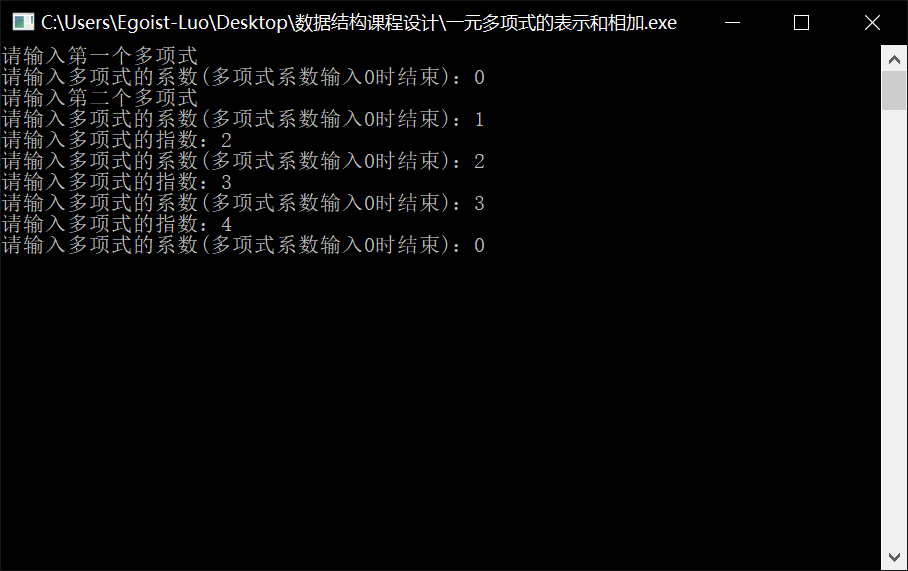
<3>三组测试数据

1)原始数据：L1：F(x) = 0

L2：F(x) = 1x^2+2x^3+3x^4

预期结果：L3: F(x) = 1x^2+2x^3+3x^4

实际结果：



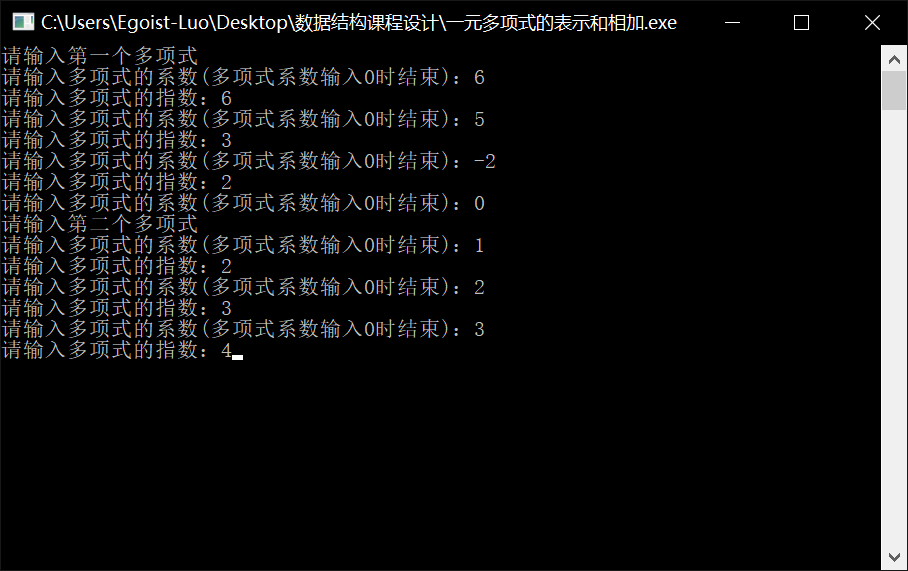


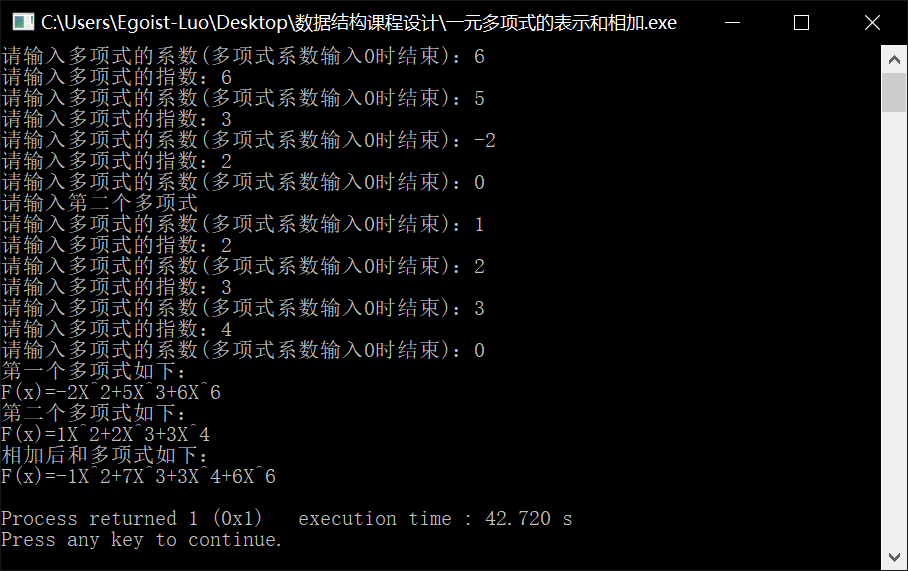
2)原始数据：L1：F(x) = 6x^6+5X^3-2x^2

L2：F(x) = 1x^2+2x^3+3x^4

预期结果：L3: F(x) = -x^2+7x^3+3x^4+6x^6

实际结果：



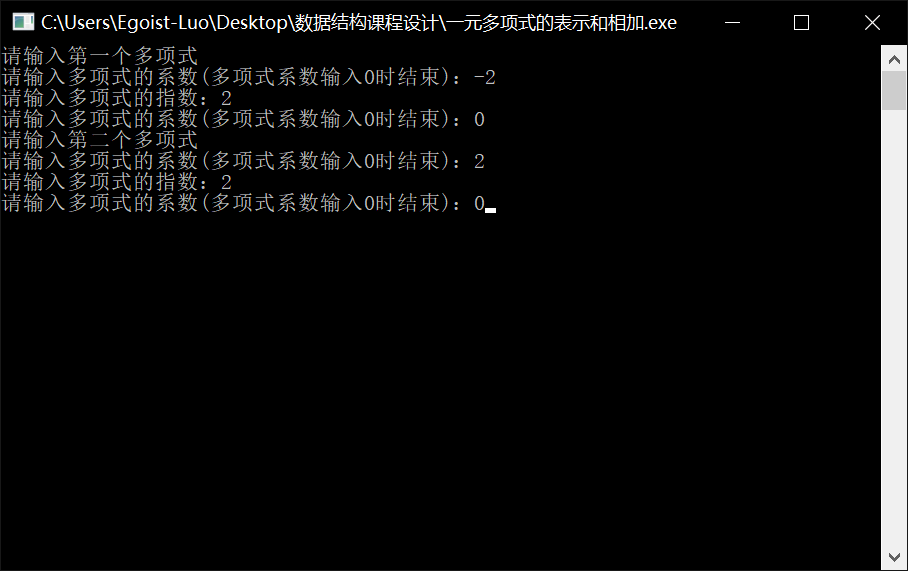


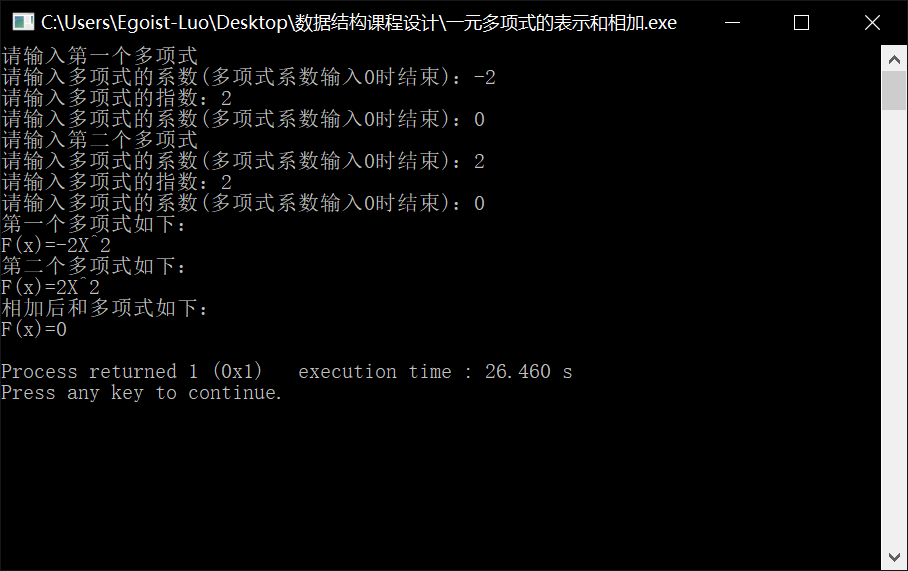
3)原始数据：L1：F(x) = -2x^2

L2：F(x) = 2x^2

预期结果：L3: F(x) = 0

实际结果：





<4>含注释的源程序：

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

typedef struct {

int cof; //系数

int exp; //指数

struct Node \*next; /\*指向下一项的指针\*/

}Poly;

int createPoly(Poly \*L) { /\*创建多项式链表\*/

int x, y; /\*x表示系数，y表示指数\*/

Poly \*q; /\*定义新节点\*/

do {

printf("请输入多项式的系数(多项式系数输入0时结束)：");

scanf("%d", &x);

if (x == 0) { /\*如果系数为0就将指数填入0\*/

y = 0;

} else {

printf("请输入多项式的指数：");

scanf("%d", &y);

}

q = (Poly\*)malloc(sizeof(Poly));

q -> cof = x;

q -> exp = y;

L -> next = q;

L = q;

}while(L -> cof != 0); /\*系数为0就跳出\*/

L -> next = NULL; /\*尾标识为NULL\*/

return 1;

}

int printPolynoimal(Poly \*L) { /\*遍历链表并且输出\*/

Poly \*head;

head = L;

head = head -> next;

if (head -> cof == 0) { /\*如果第一个节点系数是0就说明整个链表为空\*/

printf("F(x)=0\n");

return;

} else {

printf("F(x)=%dX^%d", head -> cof, head -> exp);

head = head -> next;

}

while (head -> cof != 0) {

if (head -> cof > 0) {

printf("+%dX^%d", head -> cof, head -> exp);

head = head -> next;

} else {

printf("%dX^%d", head -> cof, head -> exp);

head = head -> next;

}

}

printf("\n");

return 1;

}

int sortPolynoimal(Poly \*L) { /\*对节点进行排序，仅交换数据\*/

int i, j, tempCof, tempExp, length = 0;

Poly \*head, \*point; /\*辅助指针point帮助指向head的下一个节点\*/

head = L;

head = head -> next;

while (head -> next != NULL) { /\*计算链表长度\*/

length++;

head = head -> next;

}

for (i = 0; i < length - 1; i++) { /\*冒泡排序\*/

head = L; /\*将head复位\*/

head = head -> next;

for(j = i; j < length - 1; j++) {

point = head -> next;

if(head -> exp > point -> exp) {

tempCof = head -> cof;

tempExp = head -> exp;

head -> cof = point -> cof;

head -> exp = point -> exp;

point -> cof = tempCof;

point -> exp = tempExp;

}

head = head -> next;

}

}

return 1;

}

int addPolynoimal(Poly \*L1, Poly \*L2, Poly \*L3) {

int x = 0; /\*x为和多项式每一项的系数\*/

Poly \*headL1, \*headL2, \*headL3; /\*定义和多项式的指针和节点\*/

headL1 = L1;

headL2 = L2;

headL1 = headL1 -> next;

headL2 = headL2 -> next;

while (headL1 -> cof && headL2 -> cof) {

if (headL1 -> exp == headL2 -> exp) { /\*两指数相同时，系数相加\*/

x = headL1 -> cof + headL2 -> cof;

if (x) { /\*系数和不为零就插入到和多项式\*/

headL3 = (Poly\*)malloc(sizeof(Poly));

headL3 -> cof = x;

headL3 -> exp = headL1 -> exp;

L3 -> next = headL3;

L3 = headL3;

headL1 = headL1 -> next;

headL2 = headL2 -> next;

} else { /\*系数和为零就向后移一个节点\*/

headL1 = headL1 -> next;

headL2 = headL2 -> next;

}

} else if (headL1 -> exp < headL2 -> exp){ /\*两指数不相同时，将小的节点插入L3\*/

headL3 = (Poly\*)malloc(sizeof(Poly));

headL3 -> cof = headL1 -> cof;

headL3 -> exp = headL1 -> exp;

L3 -> next = headL3;

L3 = headL3;

headL1 = headL1 -> next;

} else {

headL3 = (Poly\*)malloc(sizeof(Poly));

headL3 -> cof = headL2 -> cof;

headL3 -> exp = headL2 -> exp;

L3 -> next = headL3;

L3 = headL3;

headL2 = headL2 -> next;

}

}

while (headL1 -> cof != 0) { /\*如果L1有剩余即当前节点系数不为0，则将剩下的全部插入L3\*/

headL3 = (Poly\*)malloc(sizeof(Poly));

headL3 -> cof = headL1 -> cof;

headL3 -> exp = headL1 -> exp;

L3 -> next = headL3;

L3 = headL3;

headL1 = headL1 -> next;

}

while (headL2 -> cof != 0) { /\*如果L2有剩余即当前节点系数不为0，则将剩下的全部插入L3\*/

headL3 = (Poly\*)malloc(sizeof(Poly));

headL3 -> cof = headL2 -> cof;

headL3 -> exp = headL2 -> exp;

L3 -> next = headL3;

L3 = headL3;

headL2 = headL2 -> next;

}

headL3 = (Poly\*)malloc(sizeof(Poly)); /\*制造出cof值为0的节点，插入到链表末尾作为标志节点\*/

headL3 -> cof = 0;

L3 -> next = headL3;

return;

}

int main() {

Poly \*L1, \*L2, \*L3;

L1 = (Poly\*)malloc(sizeof(Poly));

L2 = (Poly\*)malloc(sizeof(Poly));

L3 = (Poly\*)malloc(sizeof(Poly));

printf("请输入第一个多项式\n");

createPoly(&L1);

printf("请输入第二个多项式\n");

createPoly(&L2);

sortPolynoimal(&L1);

sortPolynoimal(&L2);

printf("第一个多项式如下：\n");

printPolynoimal(&L1);

printf("第二个多项式如下：\n");

printPolynoimal(&L2);

printf("相加后和多项式如下：\n");

addPolynoimal(&L1, &L2, &L3);

printPolynoimal(&L3);

return 1;

}

<5>编译与运行情况：编译正常通过，运行正常。

(2)表达式括号匹配检验（压栈、出栈）

<1>解题思路：构造包含顶指针，底指针和增量的结构体。以此建立一个空栈。然后依次读入输入的字符，存放至栈中。最后从栈中依次读出字符。分别设置三种括号的标志。当右括号读出时标志加一，当，左括号读出时标志减一。如果表达式括号匹配，则三种标志位都等于0，如果括号不匹配则标志位不等于0。最后输出结果。

<2>函数调用图：

开始

建立空栈S

是否是回车

入栈

输入字符

不是

是

出栈

是否是括号

得出结论

计算计数器

不是

是

不是

是否出栈完

计数器加/减1

是

结束

各函数功能:

int InitStack (SqStack \*S) 构造一个空栈

int Push (SqStack \*S, char e) 让e入栈

int Pop (SqStack \*S, char \*e) 让e出栈

int isEmpty (SqStack \*S) 判断栈是否为空，空返回0，非空返回1

int match (SqStack \*S, int counter) 判断括号是否匹配，通过计数器计算前后括号数量。如果三个计数器都为0，括号匹配。

int main() 流程控制，并且建立栈S。

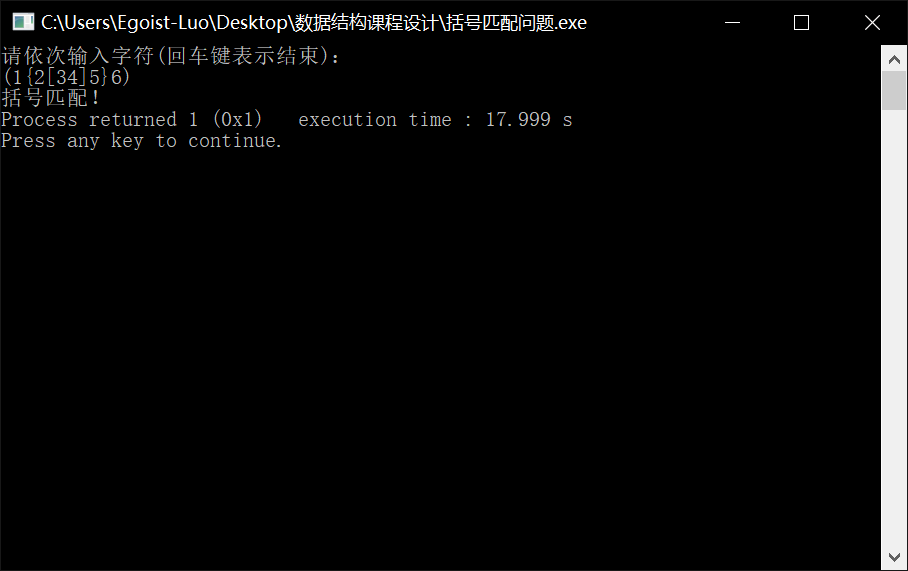
<3>三组测试数据

1. 原始数据：(1{2[34]5}6)

预期结果：匹配

实际结果：



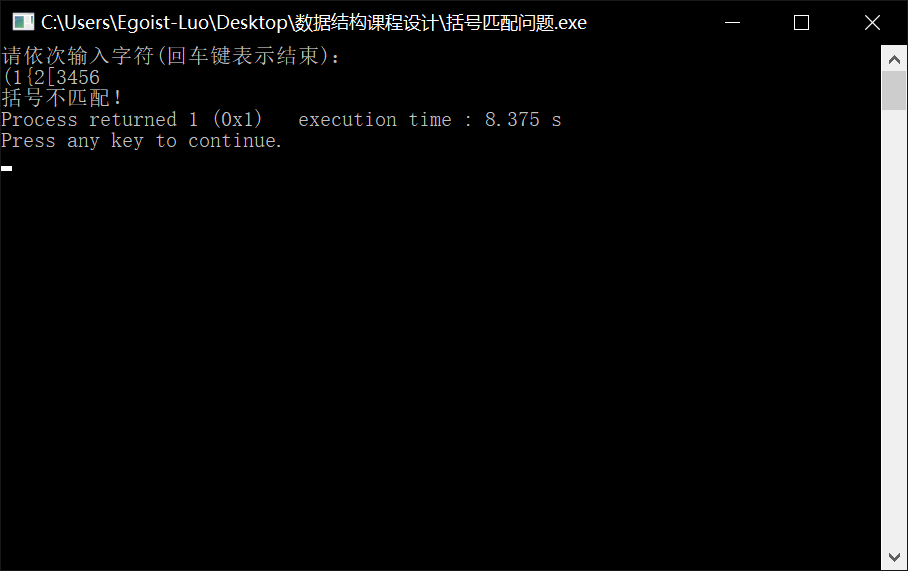


2)原始数据：(1{2[3456

预期结果：不匹配

实际结果：



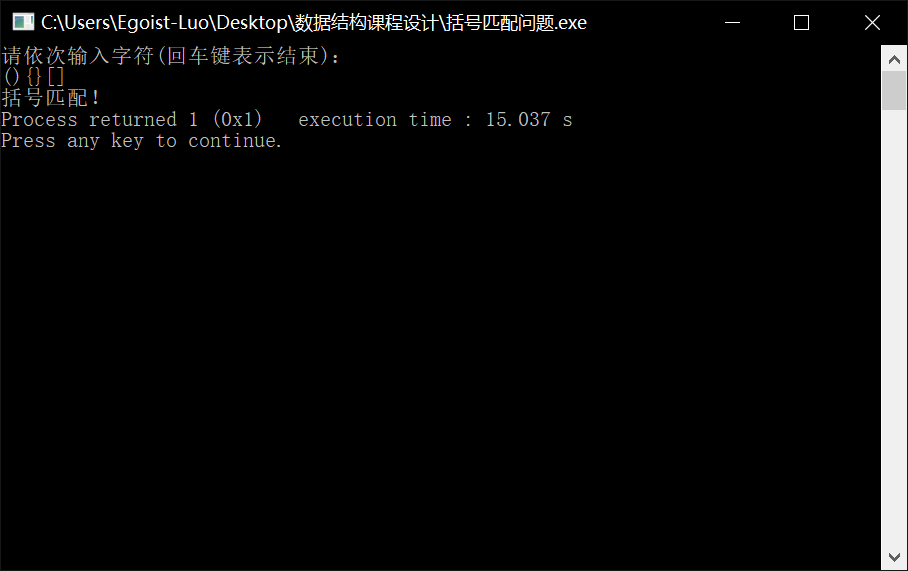


3)原始数据：(){}[]

预期结果：匹配

实际结果：





<4>含注释的源程序：

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#define STACK\_INIT\_SIZE 100 /\*存储空间初始分配量\*/

#define STACKINCREMENT 20 /\*存储空间分配增量\*/

typedef struct {

char \*top;

char \*base;

int stacksize;

}SqStack;

int InitStack (SqStack \*S) { /\*构造一个空栈\*/

S -> base = (char\*)malloc(STACK\_INIT\_SIZE\*sizeof(char));

S -> top = S -> base;

S -> stacksize = STACK\_INIT\_SIZE;

return 1;

}

int Push (SqStack \*S, char e) { /\*插入元素e为新的栈顶元素\*/

if (S -> top - S -> base >= S -> stacksize) {

S -> base = (char\*)realloc(S -> base, (S -> stacksize + STACKINCREMENT)\*sizeof(char));

if (!S -> base) { /\*栈溢出\*/

return -1;

}

S -> top = S -> base + S -> stacksize;

S -> stacksize += STACKINCREMENT;

}

\*(S -> top) = e; /\*先赋值在将指针加1\*/

S -> top ++;

return 1;

}

int Pop (SqStack \*S, char \*e) { /\*退栈\*/

if (S -> top == S -> base) { /\*栈空\*/

return 0;

} else {

S -> top --; /\*先将指针减1在出栈\*/

\*e = \*(S -> top);

}

return 1;

}

int isEmpty (SqStack \*S) { /\*判断栈是否为空\*/

if (S -> top == S -> base) {

return 0;

} else {

return 1;

}

}

int match (SqStack \*S, int counter) { /\*判断括号是否匹配\*/

int i, flag1 = 0, flag2 = 0, flag3 = 0; /\*标志位flag1,2,3, 记录三种括号()[]{}的次数\*/

char e;

if (isEmpty(S) == 0) { /\*判断栈是否为空\*/

printf("输入的字符为空！");

} else { /\*不为空就弹栈\*/

for (i = 0; i < counter; i++) {

Pop(S, &e);

switch (e) { /\*判断括号是否匹配\*/

case '(': flag1--;break;

case ')': flag1++;break;

case '[': flag2--;break;

case ']': flag2++;break;

case '{': flag3--;break;

case '}': flag3++;break;

default: break;

}

}

if ((flag1 == 0 && flag2 == 0) && (flag3 == 0 && (counter != 0))) {

printf("括号匹配！");

} else {

printf("括号不匹配！");

}

}

return 1;

}

int main() {

int counter; /\*定义计数器，记录入栈次数\*/

char c = 0;

SqStack \*S;

InitStack(&S);

printf("请依次输入字符(回车键表示结束)：\n");

while (c != '\n') { /\*当输入的字符是回车时,停止输入\*/

c = getchar();

if (c != '\n') { /\*当输入的字符不是回车时,将其放入栈\*/

Push(&S, c);

counter++;

}

}

match(&S, counter);

return 1;

}

<5>编译与运行情况：编译正常通过，运行正常。

(3)数制的转换（十进制到二进制转换）

<1>解题思路: 构造包含顶指针，底指针和增量的结构体。以此建立一个空栈。然后输入数据，通过除法和求模运算分别求出整数商和余数，将余数放入栈，用商作为数据，循环上一步的的运算直到商为0，最后将数据出栈，就是转换为二进制的结果。

<2>函数调用图:

开始

建立空栈S

输入数据

将余数入栈

做除法，求模运算

不是

商是否为0

是

依次出栈

输出

结束

各函数功能：

int InitStack (SqStack \*S) 构造一个空栈

int Push (SqStack \*S, char e) 让e入栈

int Pop (SqStack \*S, char \*e) 让e出栈

int Convert (SqStack \*S, int data) 转换函数，将data除2和取模运算，即分别计算出商和余数。接下来继续计算，以商作为data，重复运算。

int main() 流程控制，并且建立栈S。

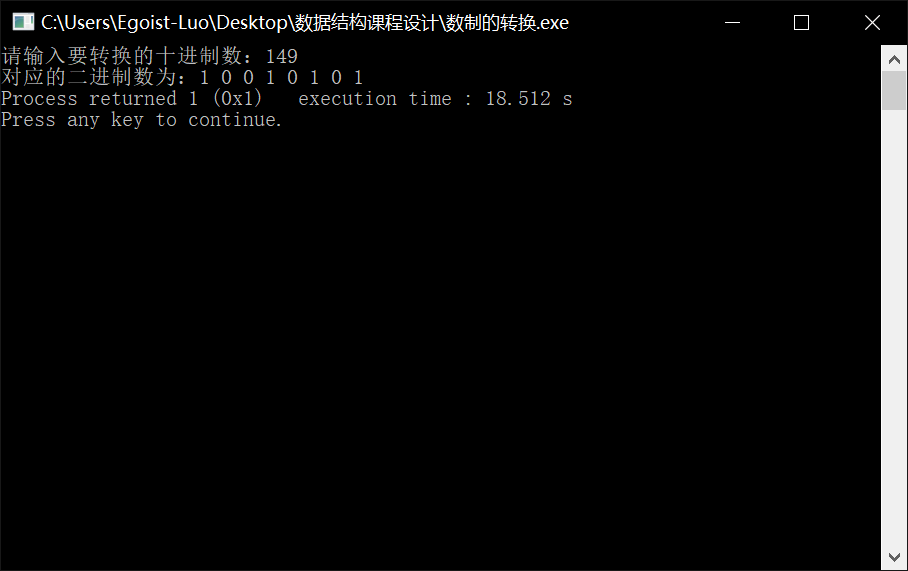
<3>三组测试数据

1)原始数据：149

预期结果：1001 0101

实际结果：



2)原始数据：0

预期结果：0

实际结果：



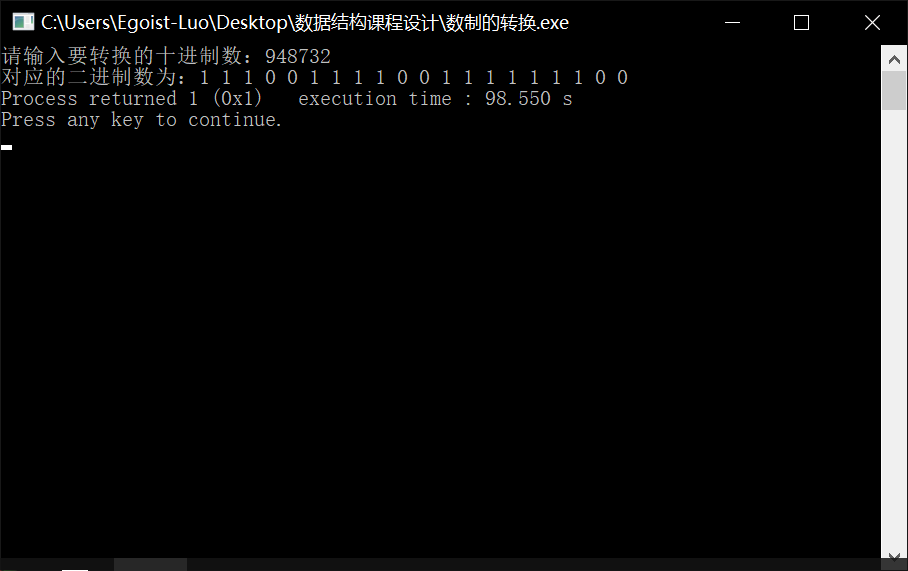


3)原始数据：948732

预期结果：1110 0111 1001 1111 1100

实际结果：





<5>编译与运行情况：编译正常通过，运行正常。

(4)约瑟夫环（循环单链表）

<1>解题思路：先定义包含一个数字域以及next的指针域的结构体。然后输入节点总数创建链表，最后将链表尾指针指向第一个数据节点使之闭合成为约瑟夫环。接下来输入开始节点和间距。如果间距为1就是依次输出即可。如果间距大于一就相隔间距输出，并且输出后将断环闭合。最后直到约瑟夫环中没有节点即结束。

<2>函数调用图：

开始

创建链表

闭合约瑟夫环

判断间距是否为1

是

不是

间隔输出

连续输出

判断约瑟夫环中是否空

闭环

不是

是

结束

各函数功能:

int create(Node \*L) 创建链表

int process(Node \*L) 实现约瑟夫环，先进行闭环操作，然后根据开始节点和间距进行输出节点操作。

int main() 流程控制，并且建立列表L

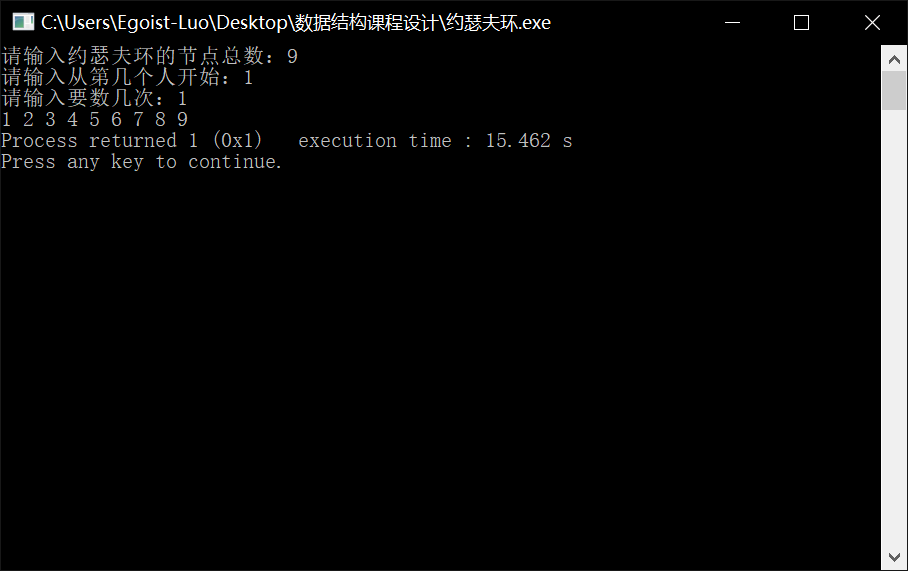
<3>三组测试数据

1)原始数据：节点总数9即123456789 起始1，间隔1

预期结果：123456789

实际结果：

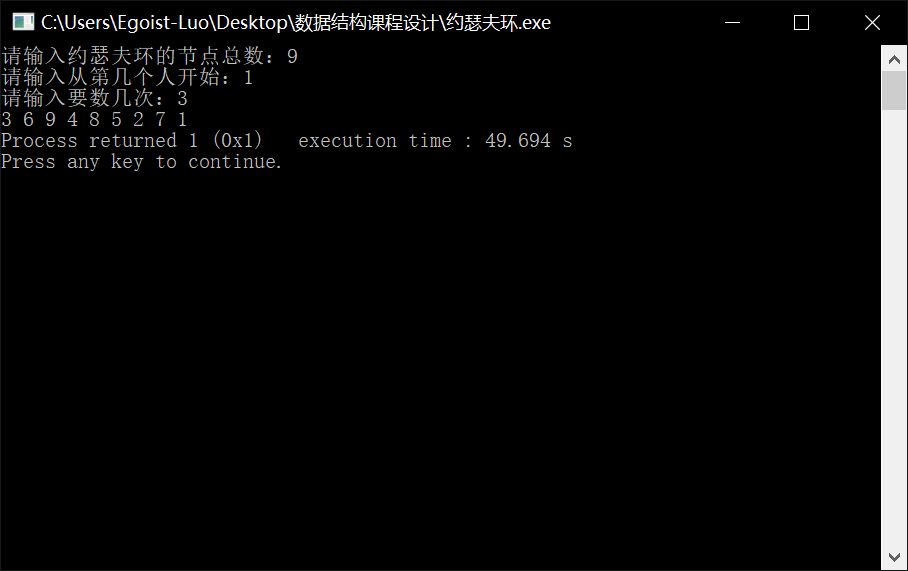


2)原始数据：节点总数9即123456789 起始1，间隔3

预期结果：369485271

实际结果：



3)原始数据：节点总数9即123456789 起始6，间隔4

预期结果：648521376

实际结果：





<4>含注释的源程序

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef struct {

int num;

struct Node \*next;

}Node;

int create(Node \*L) { /\*创建约瑟夫环\*/

int i; /\*计数器\*/

int num; /\*每个节点中的数字及约瑟夫环的节点总数\*/

Node \*q; /\*定义新节点\*/

printf("请输入约瑟夫环的节点总数：");

scanf("%d", &num);

for (i = 1; i <= num; i++) { /\*建立数据节点\*/

q = (Node\*)malloc(sizeof(Node));

q -> num = i;

L -> next = q;

L = q;

}

L -> next = NULL; /\*尾标识为NULL\*/

return 1;

}

int process(Node \*L) { /\*实现约瑟夫环\*/

int m, n, i; /\*定义从第m个人开始，数n次\*/

Node \*head, \*point; /\*定义头结点，point节点(用以删除节点)\*/

head = L;

head = head -> next; /\*将头结点和L移到有数值的第一个节点\*/

L = head;

while (head -> next != NULL) { /\*遍历到最后一个节点\*/

head = head -> next;

}

head -> next = L; /\*闭环\*/

printf("请输入从第几个人开始：");

scanf("%d", &m);

printf("请输入要数几次：");

scanf("%d", &n);

for (i = 1; i < m; i++) { /\*将指针移到第m个人处\*/

L = L -> next;

}

if (n == 1) { /\*如果第m个人数1次，就依次出列\*/

while(L -> next != L) { /\*如果最后只有L指向的节点自己形成环，就跳出\*/

head = L; /\*L自身就是要输出的节点\*/

while(head -> next != L) { /\*将head指向L的前节点\*/

head = head -> next;

}

printf("%d ", L -> num);

point = L; /\*将point指针指向已经输出的节点\*/

L = L -> next;

head -> next = L; /\*将断环闭合\*/

free(point);

}

printf("%d", L -> num);

free(L);

return 1;

} else { /\*如果第m个人数n次\*/

while(L -> next != L) {

for (i = 1; i < n; i++) { /\*将L指向要输出的节点\*/

L = L -> next;

}

while(head -> next != L) {

head = head -> next;

}

printf("%d ", L -> num);

point = L;

L = L -> next;

head -> next = L;

free(point);

}

printf("%d", L -> num);

free(L);

return 1;

}

}

int main() {

Node \*Circle;

Circle = (Node\*)malloc(sizeof(Node));

create(&Circle);

process(&Circle);

}

<5>编译与运行情况：编译正常通过，运行正常。

(5)表达式求解

<1>解题思路: 构造包含顶指针，底指针和增量的结构体。然后分别构造一个只包含运算符的栈（OPTR）和只包含数字的栈（OPND）。之后依次读入所输入的表达式。判断是不是数字，如果是数字就将数字放入数字栈（OPND）。如果不是即运算符，让运算符栈栈顶元素和读入的运算符进行比较。如果优先级小于将读入的运算符入栈，优先级相等的就让栈顶元素出栈，优先级的大于的就让栈顶元素弹栈，并且连续两次让数字栈弹栈，得到一个运算符和两个数字，进行计算，得到的结果放入数字栈。循环以上过程直到读入的表达式字符为#为止。最后将数字栈出栈，即得到结果。

<2>函数调用图

开始

创建运算符栈和数字栈，并初始化

读入表达式

不是

是

判断是否为数字

小于

放入数字栈

判断优先级

大于

等于

出栈

出栈

入栈

计算

入栈

判断是否结束读入

不是

是

数字栈出栈

结束

各函数功能:

char InitStack (SqStack \*S) 构造空栈

char Push (SqStack \*S, char e) 让e入栈

char Pop (SqStack \*S, char \*e) 让e出栈

char getTop(SqStack S) 返回栈顶第一个元素

char Precede(char a, char b) 比较两个运算符的优先级,a，b中存放待比较的运算符,'>'表示a>b,'0'表示不可能出现的比较，这是通过二维数组建立比较表实现的。

int in(char c, char OP[8]) 判断输入的某个字符是否是运算符

char Operate(char a, char n, char b) 四则运算计算

char evaluateExpression(SqStack OPTR, SqStack OPND) 表达式计算，流程控制

int main() 主函数，建立OPTR与OPND栈

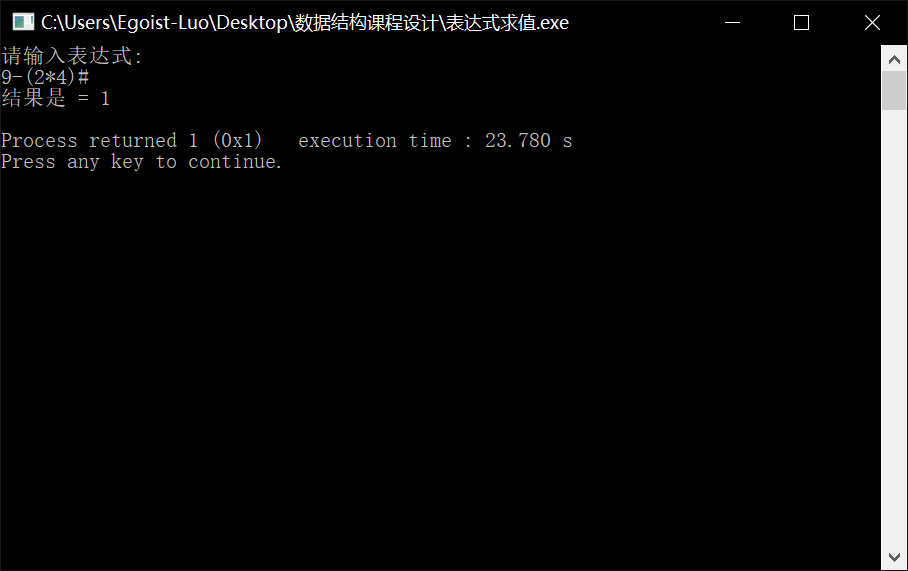
<3>三组测试数据

1)原始数据：9-(2\*4)#

预期结果：1

实际结果：

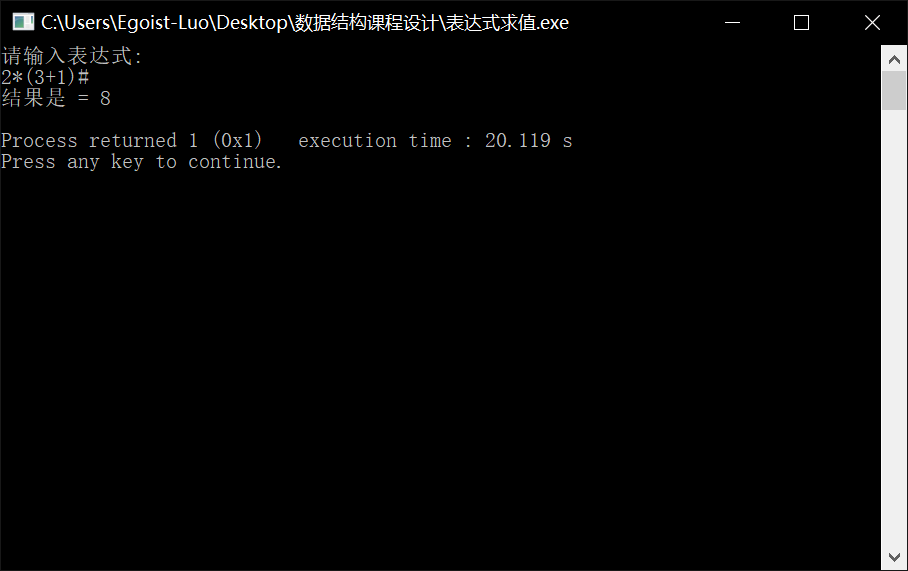


2)原始数据：2\*(3+1)#

预期结果：8

实际结果：

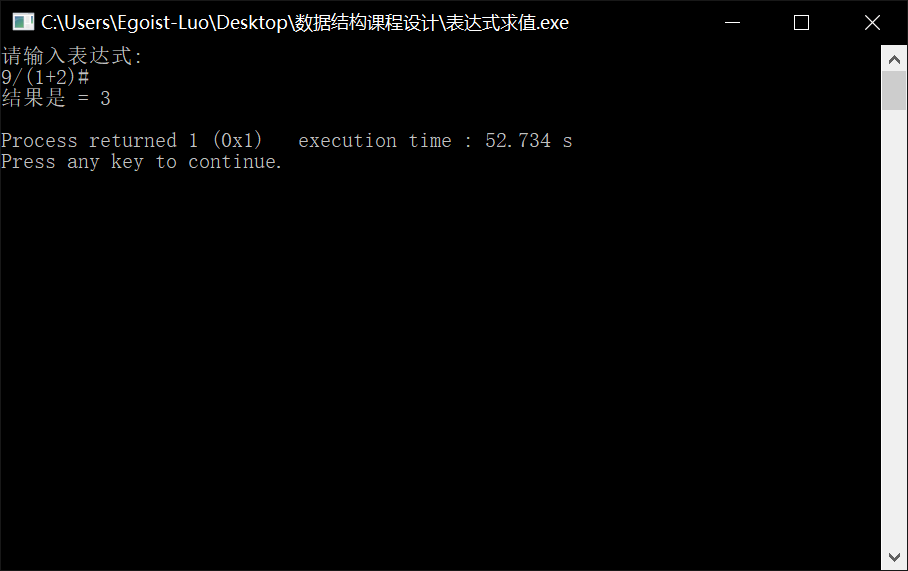


3)原始数据：9/(1+2)#

预期结果：3

实际结果：





<4>含注释的源程序

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#define STACK\_INIT\_SIZE 100 /\*存储空间初始分配量\*/

#define STACKINCREMENT 10 /\*存储空间分配增量\*/

typedef struct {

char \*top;

char \*base;

int stacksize;

}SqStack;

int InitStack (SqStack \*S) { /\*构造一个空栈\*/

S -> base = (char\*)malloc(STACK\_INIT\_SIZE\*sizeof(char));

S -> top = S -> base;

S -> stacksize = STACK\_INIT\_SIZE;

return 1;

}

int Push (SqStack \*S, char e) { /\*插入元素e为新的栈顶元素\*/

if (S -> top - S -> base >= S -> stacksize) {

S -> base = (char\*)realloc(S -> base, (S -> stacksize + STACKINCREMENT)\*sizeof(char));

if (!S -> base) { /\*栈溢出\*/

return -1;

}

S -> top = S -> base + S -> stacksize;

S -> stacksize += STACKINCREMENT;

}

\*(S -> top) = e; /\*先赋值在将指针加1\*/

S -> top ++;

return 1;

}

int Pop (SqStack \*S, char \*e) { /\*退栈\*/

if (S -> top == S -> base) { /\*栈空\*/

return 0;

} else {

S -> top --; /\*先将指针减1在出栈\*/

\*e = \*(S -> top);

}

return 1;

}

char getTop(SqStack S){ /\*返回栈顶第一个元素\*/

char e;

if(S.top == S.base)

return 0;

e = \*(S.top - 1);

return e;

}

char Precede(char a, char b) { /\*比较两个运算符的优先级,a，b中存放待比较的运算符,

'>'表示a>b,'0'表示不可能出现的比较 \*/

int i,j;

char pre[][7]={ /\*运算符之间的优先级制作成一张表格\*/

{'>','>','<','<','<','>','>'},

{'>','>','<','<','<','>','>'},

{'>','>','>','>','<','>','>'},

{'>','>','>','>','<','>','>'},

{'<','<','<','<','<','=','0'},

{'>','>','>','>','0','>','>'},

{'<','<','<','<','<','0','='}};

switch(a) {

case '+': i = 0; break;

case '-': i = 1; break;

case '\*': i = 2; break;

case '/': i = 3; break;

case '(': i = 4; break;

case ')': i = 5; break;

case '#': i = 6; break;

}

switch(b) {

case '+': j = 0; break;

case '-': j = 1; break;

case '\*': j = 2; break;

case '/': j = 3; break;

case '(': j = 4; break;

case ')': j = 5; break;

case '#': j = 6; break;

}

return pre[i][j];

}

int in(char c, char OP[8]) { /\*判断输入的某个字符是否是运算符\*/

int i;

for(i = 0; OP[i] != '\0'; i++) {

if(OP[i] == c) {

return 1;

}

}

return 0;

}

char Operate(char a, char n, char b) { /\*四则运算计算\*/

int i, j, result;

char back;

i = a - '0'; /\*将char转换为int\*/

j = b - '0';

switch(n) {

case '+': result = i + j; break;

case '-': result = i - j; break;

case '\*': result = i \* j; break;

case '/': result = i / j; break;

}

back = result + '0';

return back;

}

char evaluateExpression(SqStack OPTR, SqStack OPND) {

char a, b, c, x, n, temp;

temp = '#';

Push(&OPTR, temp);

char OP[8] = {'+','-','\*','/','(',')','#','\0'};

c = getchar(); /\*输入字符\*/

while(c != '#' || getTop(OPTR) != '#') { /\*判断是不是运算符\*/

if(!in(c, OP)) { /\*不是运算符则进栈\*/

Push(&OPND, c);

c = getchar();

} else { /\*是运算符就和前面的比较\*/

x = getTop(OPTR);

switch(Precede(x, c)) {

case '<': /\*栈顶元素优先级低\*/

Push(&OPTR, c);

c = getchar();

break;

case '=': /\*脱括号并接受下一字符\*/

Pop(&OPTR, &x);

c = getchar();

break;

case '>': /\*退栈并将运算结果入栈\*/

Pop(&OPTR, &n);

Pop(&OPND, &b);

Pop(&OPND, &a);

Push(&OPND, Operate(a, n, b));

break;

}

}

}

return getTop(OPND);

}

int main(){

char c;

SqStack OPTR; /\*OPTR为运算符栈，OPND为数字栈\*/

SqStack OPND;

InitStack(&OPTR);

InitStack(&OPND);

printf("请输入表达式:\n");

c = evaluateExpression(OPTR, OPND);

printf("结果是 = %c\n",c);

return 1;

}

<5>编译与运行情况：编译正常通过。运行时如果计算中出现负数就无法运算，而且输入表达式的时候因为用的是getchar所以无法读出两位数以上的数字进行计算，这是该程序缺陷的地方。

(6) 将两个有序线性表合并成一个有序线性表，并去掉重复元素。

<1>解题思路：先建立一个结构体，结构体中包含数据域以及next的指针域。将每个结构体定义为一个节点，再通过指针域链接来建立链表。选择升序或者降序排序，分别输入节点数据，建立L1，L2的链表。然后根据选择升序或者降序调用不同的合并函数。如果是升序则数据小的先插入链表，大的后插入，如果是降序则大的先插入链表，小的后插入。最后输出新链表L3。

<2>函数调用图：

开始

建立链表L1,L2

选择升降序

升序合并

降序合并

输出

结束

各函数功能：

int createList (LNode \*L) 建立线性表

int mergeListSL(LNode \*L1, LNode \*L2, LNode \*L3) 将L1，L2合并成L3，升序并输出

int mergeListLS(LNode \*L1, LNode \*L2, LNode \*L3) 将L1，L2合并成L3，

降序并输出

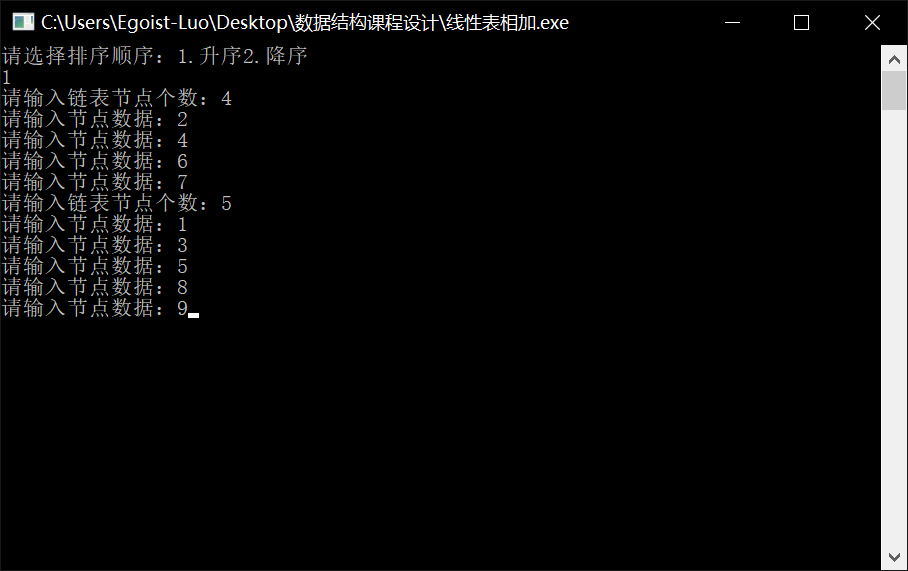
int main() 主函数，流程控制。

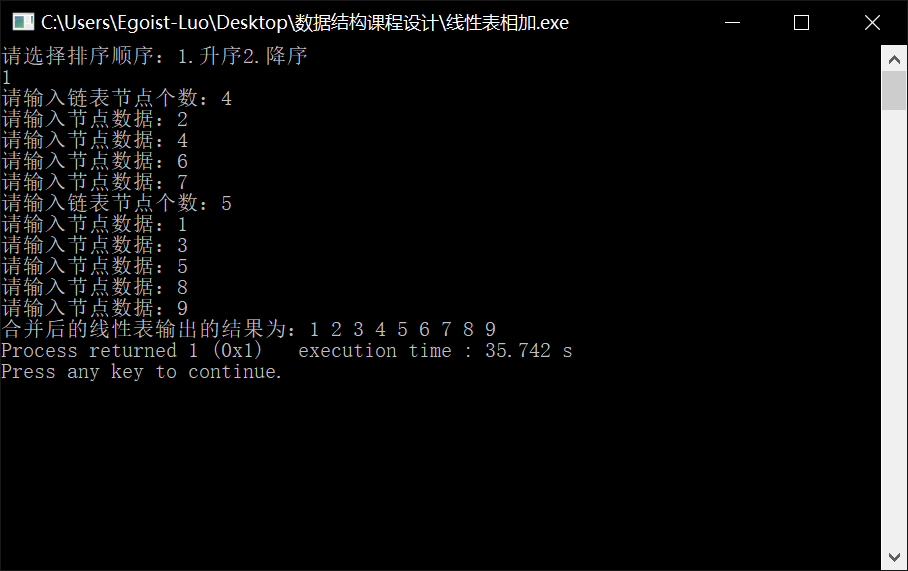
<3>三组测试数据

1)原始数据：升序（1） L1：2467 L2：13589

预期结果：123456789

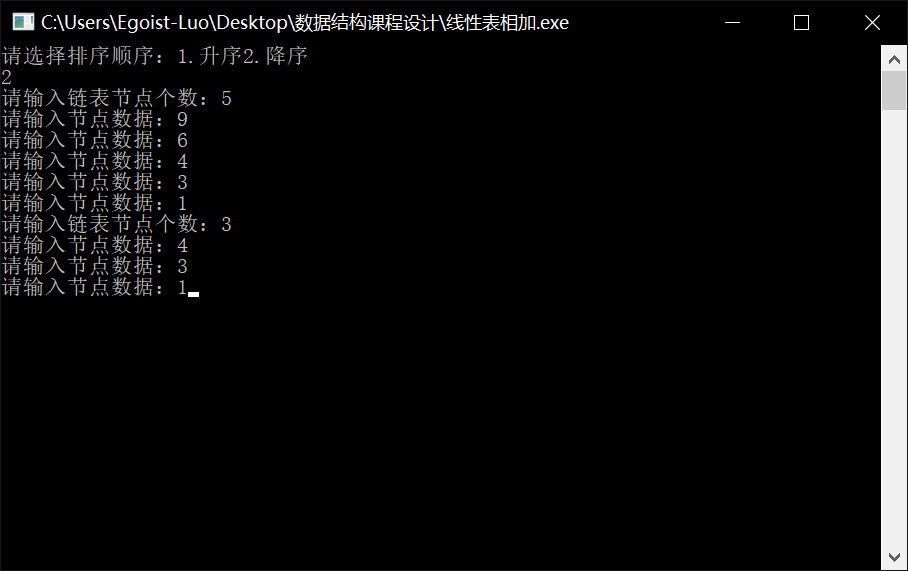
实际结果：

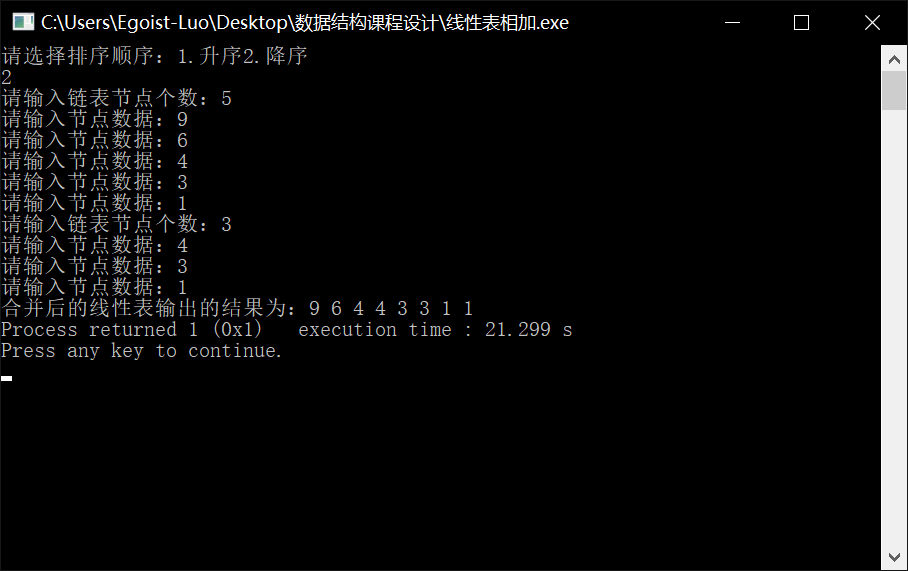


2)原始数据：降序（2） L1：96431 L2：431

预期结果：96443311

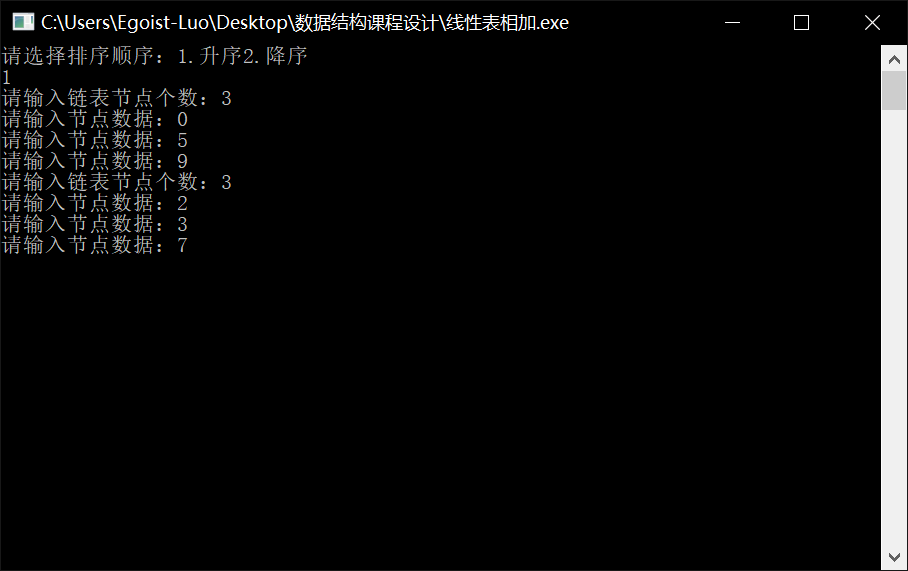
实际结果：

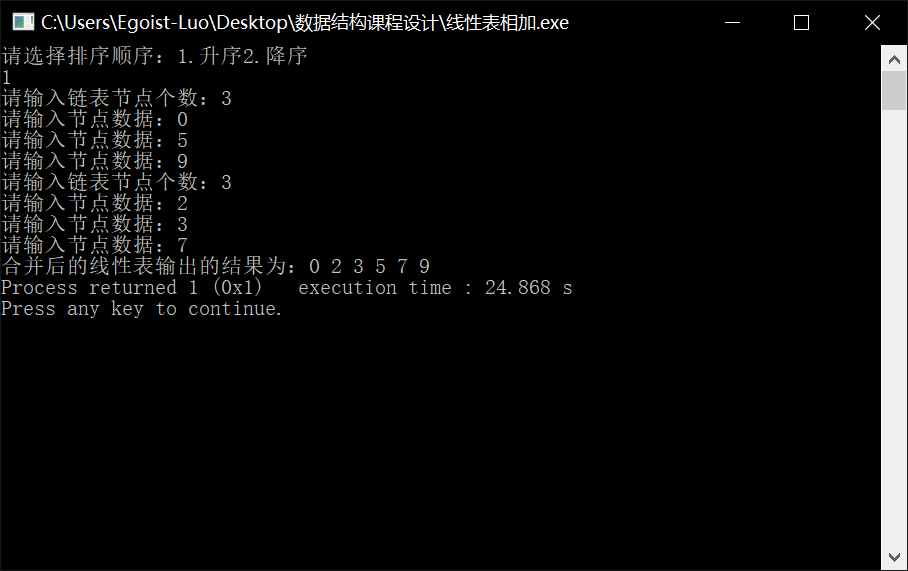


3)原始数据：升序（1） L1：059 L2：237

预期结果：023579

实际结果：





<4>含注释的源程序

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef struct {

int data;

struct Node \*next;

}LNode;

int createList (LNode \*L) { /\*建立线性表\*/

LNode \*q;

int m, n, i; /\*节点个数为m，节点数据为n\*/

printf("请输入链表节点个数：");

scanf("%d", &m);

for (i = 0; i < m; i++) {

q = (LNode\*)malloc(sizeof(LNode));

printf("请输入节点数据：");

scanf("%d", &n);

q -> data = n;

L -> next = q;

L = q;

}

L -> next = NULL; /\*尾标识为NULL\*/

return 1;

}

int mergeListSL(LNode \*L1, LNode \*L2, LNode \*L3) { /\*将L1，L2合并成L3，升序\*/

LNode \*p1, \*p2, \*p3;

p1 = L1;

p2 = L2;

p3 = L3;

p1 = p1 -> next;

p2 = p2 -> next;

while (p1 != NULL && p2 != NULL) {

if (p1 -> data <= p2 -> data) { /\*如果p1的data小于p2的data，就将p1插入L3\*/

p3 -> next = p1;

p3 = p1;

p1 = p1 -> next;

} else { /\*如果p2的data小于p1的data，就将p2插入L3\*/

p3 -> next = p2;

p3 = p2;

p2 = p2 -> next;

}

}

p3 -> next = (p1 != NULL)? p1: p2; /\*如果p1还有节点，就将p1剩余节点全部插入L3，p2同理\*/

p3 = L3; /\*指针回到L3开头\*/

p3 = p3 -> next;

printf("合并后的线性表输出的结果为：");

while (p3 != NULL) { /\*将L3线性表显示出来\*/

printf("%d ", p3 -> data);

p3 = p3 -> next;

}

return 1;

}

int mergeListLS(LNode \*L1, LNode \*L2, LNode \*L3) { /\*将L1，L2合并成L3，降序\*/

LNode \*p1, \*p2, \*p3;

p1 = L1;

p2 = L2;

p3 = L3;

p1 = p1 -> next;

p2 = p2 -> next;

while (p1 != NULL && p2 != NULL) {

if (p1 -> data >= p2 -> data) { /\*如果p1的data大于p2的data，就将p1插入L3\*/

p3 -> next = p1;

p3 = p1;

p1 = p1 -> next;

} else { /\*如果p2的data大于p1的data，就将p2插入L3\*/

p3 -> next = p2;

p3 = p2;

p2 = p2 -> next;

}

}

p3 -> next = (p1 != NULL)? p1: p2; /\*如果p1还有节点，就将p1剩余节点全部插入L3，p2同理\*/

p3 = L3; /\*指针回到L3开头\*/

p3 = p3 -> next;

printf("合并后的线性表输出的结果为：");

while (p3 != NULL) { /\*将L3线性表显示出来\*/

printf("%d ", p3 -> data);

p3 = p3 -> next;

}

return 1;

}

int main() {

LNode \*L1, \*L2, \*L3;

int ch;

L1 = (LNode\*)malloc(sizeof(LNode));

L2 = (LNode\*)malloc(sizeof(LNode));

L3 = (LNode\*)malloc(sizeof(LNode));

printf("请选择排序顺序：");

printf("1.升序");

printf("2.降序\n");

scanf("%d", &ch);

createList(&L1);

createList(&L2);

switch(ch) {

case 1:mergeListSL(&L1, &L2, &L3);break;

case 2:mergeListLS(&L1, &L2, &L3);break;

default: printf("输入错误");break;

}

return 1;

}

<5>编译与运行情况：编译正常通过，运行正常。

(7)设有一个线性单链表，其结点值均为正整数，且按值从大到小链接。试写出一个算法，将该线性单链表分解成两个线性单链表，其中一个链表中的结点值均为奇数，而另一个链表中的结点值均为偶数，且这两个链表均按值从小到大链接。

<1>解题思路： 先建立一个结构体，结构体中包含数据域以及next的指针域。然后将原链表先通过冒泡排序倒置，再遍历倒置后的链表。如果数据是奇数的放入新的链表L2，如果是偶数放入新的链表L3，直到遍历完成。就成功拆分了原链表。

<2>函数调用图：

开始

建立初始链表L1

遍历

冒泡排序

将节点放入L2

是

不是

数据域是否为奇数

将节点放入L3

打印链表

结束

各函数功能：

int createList (LNode \*L) 建立线性表

int print(LNode \*L) 打印数据

int sortList(LNode \*L) 对链表进行冒泡排序

int splitList(LNode \*L1, LNode \*L2, LNode \*L3) 将原线性表分开成两个线性表

int main() 主函数，流程控制

<3>三组测试数据

1)原始数据：987654321

预期结果：L1：13579 L2：2468

实际结果：



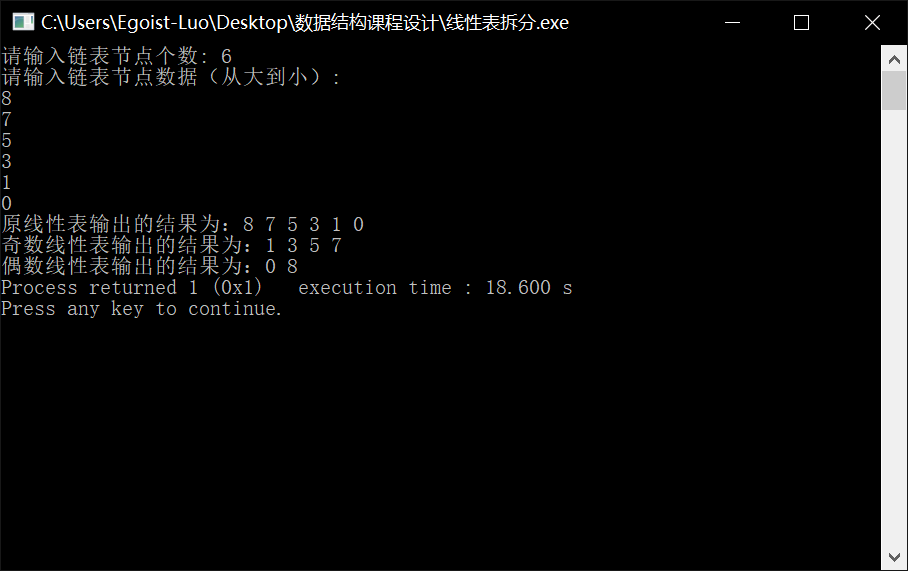


2)原始数据：875310

预期结果：L1：1357 L2：08

实际结果：





3)原始数据：8754210

预期结果：L1：157 L2：0248

实际结果：





<4>含注释的源程序：

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef struct {

int data;

struct Node \*next;

}LNode;

int createList (LNode \*L) { /\*建立线性表\*/

LNode \*q;

int m, n, i; /\*节点个数为m，节点数据为n\*/

printf("请输入链表节点个数: ");

scanf("%d", &m);

printf("请输入链表节点数据（从大到小）:\n");

for (i = 0; i < m; i++) {

q = (LNode\*)malloc(sizeof(LNode));

scanf("%d", &n);

q -> data = n;

L -> next = q;

L = q;

}

L -> next = NULL; /\*尾标识为NULL\*/

return 1;

}

int print(LNode \*L) { /\*打印数据\*/

LNode \*q;

q = L; /\*指针回到开头\*/

q = q -> next;

if (q == NULL) { /\*如果链表不存在数据就显示无数据\*/

printf("无数据！");

} else {

while (q != NULL) { /\*将排序后的线性表显示出来\*/

printf("%d ", q -> data);

q = q -> next;

}

}

return 1;

}

int sortList(LNode \*L) { /\*对节点进行排序，仅交换数据\*/

int i, j, tempData;

LNode \*head, \*point; /\*辅助指针point帮助指向head的下一个节点\*/

head = L;

head = head -> next;

int length = 0;

if (head != NULL) { /\*如果链表不存在数据就跳过排序\*/

while (head -> next != NULL) { /\*计算链表长度\*/

head = head -> next;

length++;

}

for (i = 0; i < length; i++) { /\*冒泡排序\*/

head = L; /\*将head复位\*/

head = head -> next;

for(j = i; j < length; j++) {

point = head -> next;

if (head -> data >= point -> data) {

tempData = head -> data;

head -> data = point -> data;

point -> data = tempData;

}

head = head -> next;

}

}

}

return 1;

}

int splitList(LNode \*L1, LNode \*L2, LNode \*L3) { /\*将原线性表分开成两个线性表\*/

LNode \*p1, \*p2, \*p3;

int m; /\*m为余数\*/

p1 = L1;

p2 = L2;

p3 = L3;

p1 = p1 -> next;

while (p1 != NULL) {

m = (p1 -> data) % 2;

if (m == 1) { /\*当前指针指向的数据为奇数时\*/

p2 -> next = p1;

p2 = p2 -> next;

p1 = p1 -> next;

} else if (m == 0) { /\*当前指针指向的数据为偶数时\*/

p3 -> next = p1;

p3 = p3 -> next;

p1 = p1 -> next;

} else {

break;

}

}

p2 -> next = NULL;

p3 -> next = NULL;

return 1;

}

int main() {

LNode \*L1, \*L2, \*L3, \*p;

L1 = (LNode\*)malloc(sizeof(LNode));

L2 = (LNode\*)malloc(sizeof(LNode));

L3 = (LNode\*)malloc(sizeof(LNode));

createList(&L1);

printf("原线性表输出的结果为：");

print(&L1);

splitList(&L1, &L2, &L3);

sortList(&L2);

sortList(&L3);

printf("\n奇数线性表输出的结果为：");

print(&L2);

printf("\n偶数线性表输出的结果为：");

print(&L3);

}

<5>编译和运行情况：编译与运行情况：编译正常通过，运行正常。

(8)已知二叉树的后序遍历和中序遍历序列，构造对应的二叉树，并非递归前序遍历该二叉树。

<1>解题思路：先建立一个结构体，结构体中包含数据域以及左孩子和右孩子的指针域。然后首先输入中序遍历和后序遍历的数组，再定义四个变量：il,ir,pl,pr即中序遍历的左右端点和后序遍历的左右端点。然后调用子函数，

分配空间，将后序遍历的右端点数据放入数据域。之后处理左右孩子，若左孩子存在则递归调用子函数，右孩子同理。最后将二叉树用非递归先序遍历输出。

<2>函数调用图：

开始

建立树节点

输入树的数据

输入中序，后序遍历

是

左孩子是否存在

否

右孩子是否存在

是

否

非递归先序遍历

结束

各函数功能：

void buildTree(int inOrder[], int postOrder[], int il, int ir, int pl, int pr, BitTree T) 根据中序和后序构造二叉树

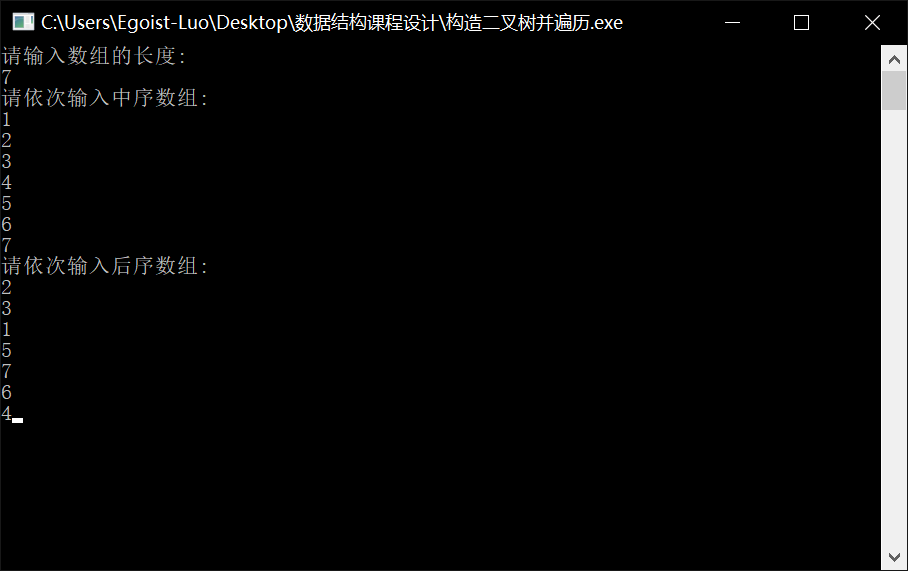
int main() 主函数，流程控制，输入中序遍历和后序遍历的数组。

<3>三组测试数据

1)原始数据：中序：1234567 后序2315764

预期结果：先序：4132657

实际结果：

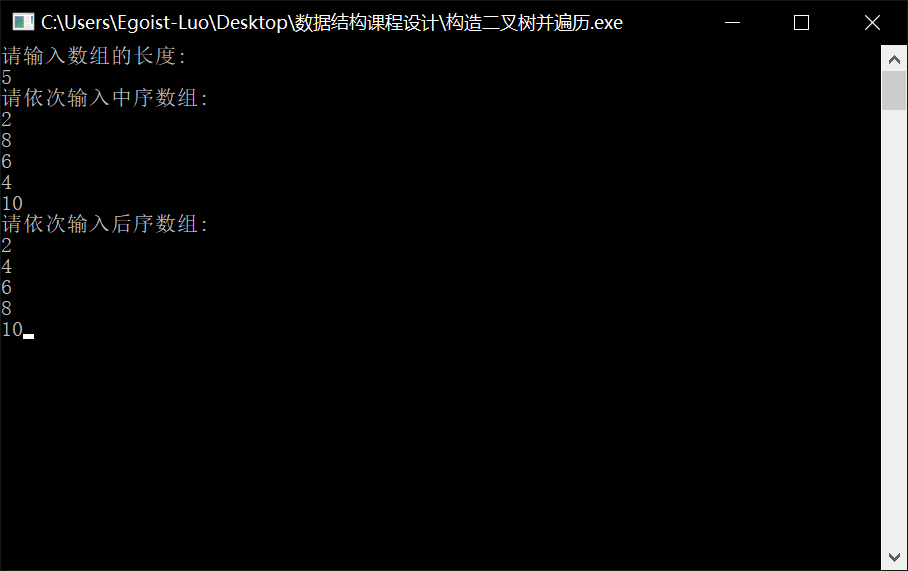




2)原始数据：中序：2 8 6 4 10后序：2 4 6 8 10

预期结果：先序：10 8 2 6 4

实际结果：

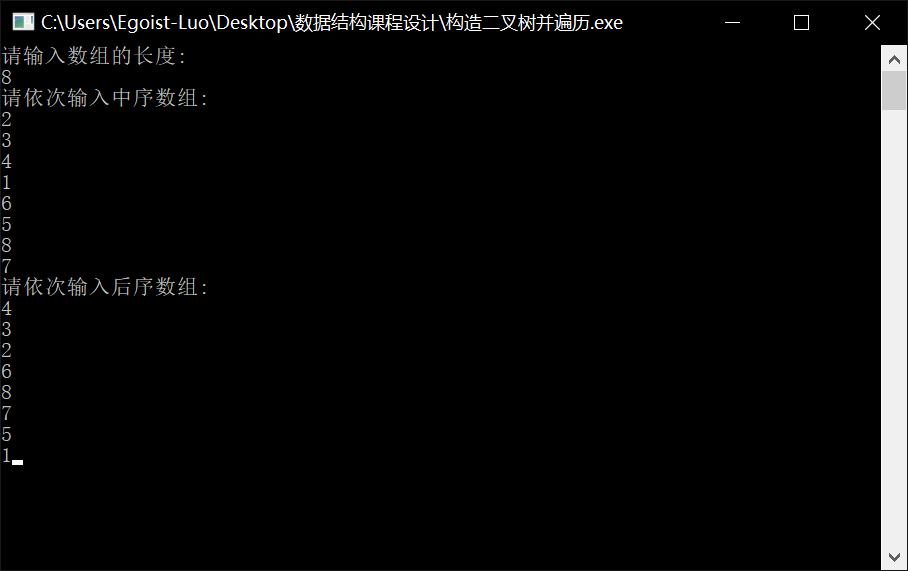


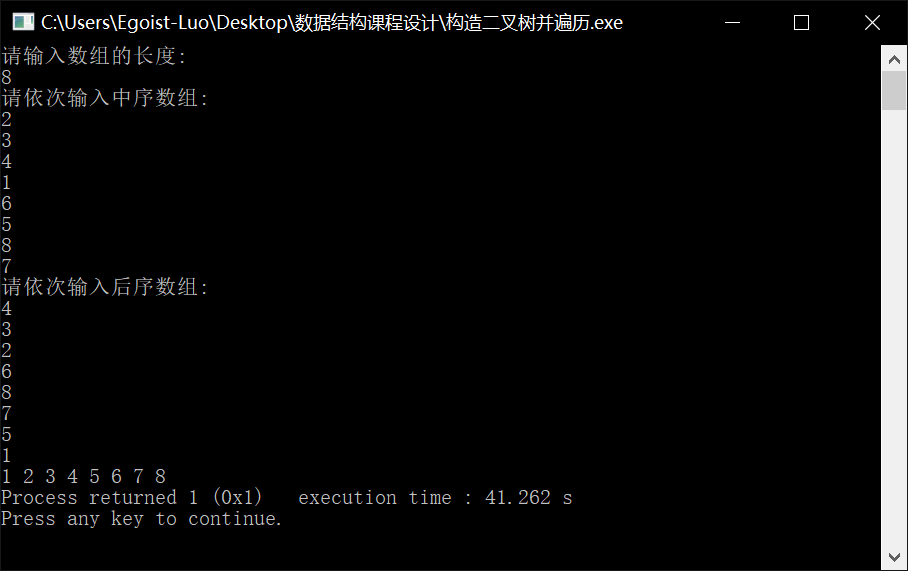


2)原始数据：中序：23416587后序：43268751

预期结果：先序：1 2 3 4 5 6 7 8

实际结果：





<4>含注释的源程序：

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef struct BitTree {

int data; /\*定义数据\*/

struct BitTree \*lChild, \*rChild; /\*定义左右孩子指针\*/

}BitNode, \*BitTree;

void buildTree(int inOrder[], int postOrder[], int il, int ir, int pl, int pr, BitTree T) { /\*根据中序和后序构造二叉树\*/

T = (BitTree)malloc(sizeof(BitNode));

T -> data = postOrder[pr]; /\*后序序列中最后一个元素为二叉树的根节点\*/

int m = il;

while(inOrder[m] != postOrder[pr]) { /\*查找根节点在中序序列的位置\*/

m++;

}

if (m == il) { /\*左子树的中序序列为空\*/

T -> lChild = NULL;

} else {

buildTree(inOrder, postOrder, il, m - 1, pl, pl + m -1 - il, T -> lChild); /\*确定左子树\*/

}

if (m == ir) { /\*右子树的中序序列为空\*/

T -> rChild = NULL;

} else {

buildTree(inOrder, postOrder, m + 1, ir, pr - ir + m, pr - 1, T -> rChild); /\*确定右子树\*/

}

}

int preOrder(BitTree T) { /\*非递归先序遍历\*/

BitTree stack[100]; /\*基于栈进行遍历\*/

int front = 0, rear = 0;

BitNode \*p;

if(T != NULL) {

stack[rear] = T; /\*将根节点入栈\*/

rear = (rear + 1) % 100;

}

while (front != rear) {

p = stack[front];

front = (front + 1) % 100;

printf("%5d", p -> data); /\*输出节点\*/

if(p -> lChild != NULL) { /\*左子树不为空就输出左子树\*/

stack[rear] = p -> lChild; /\*左子树入栈\*/

rear = (rear + 1) % 100;

}

if(p -> rChild != NULL) { /\*最后是右子树\*/

stack[rear] = p -> rChild; /\*右子树入栈\*/

rear = (rear + 1) % 100;

}

}

return;

}

int main() {

int i, il, ir, pl, pr; /\*il和ir为中序序列的左右端点, pl和pr为后序序列的左右端点\*/

int len;

BitNode \*T; /\*树的根节点\*/

printf("请输入数组的长度:\n");

scanf("%d", &len);

int inOrder[len]; /\*中序的数组\*/

printf("请依次输入中序数组:\n");

for(i = 0; i < len; i++) {

scanf("%d", &(inOrder[i]));

}

int postOrder[len]; /\*后序的数组\*/

printf("请依次输入后序数组:\n");

for(i = 0; i < len; i++) {

scanf("%d", &(postOrder[i]));

}

il = 0;

ir = len - 1;

pl = 0;

pr = len - 1;

buildTree(inOrder, postOrder, il, ir, pl, pr, &T);

preOrder(&T);

return 1;

}

<5>编译和运行情况：编译正常通过，运行如果是数字数组可以，字符数组无法实现。

(9)线性表综合题

（1）按照输入的顺序建立顺序表

（2）对顺序表进行排序（直接插入、冒泡、选择、快速、合并）

（3）按照由大到小的顺序建立一个单链表

（4）链表逆置

（5）将顺序表和链表合并成一个有序表。

（6）结果输出

<1>解题思路：通过建立一个数组和一个结构体,数组用以保存顺序表，而在结构体内建立数据域和指针域用以保存链表。首先设定序表长度并要求输入数据，建立输出顺序表函数，在通过关于顺序表的五种排序的子函数选择其一进行排序，最后通过输出函数输出；再建立链表，使数据输入链表且由大到小输出，并设计链表逆置函数；最后再写出函数将已有的顺序表和链表合并为一张有序表并输出。而在主函数中通过switch函数来选择所需要的步骤。

<2>函数调用图：

建立链表

建立顺序表

开始

逆置链表

快速排序

直接选择

直接插入

冒泡排序

归并排序

合并顺序表和链表

结束

输出

各函数功能：

int createList (int arr[]) 建立顺序表

int createList (int arr[]) 建立顺序表

int sort(int arr[]) 排序列表，流程控制，并且建立快速排序和归并排序所需数组

int insertSort(int arr[]) 直接插入排序

int bubbleSort(int arr[]) 冒泡排序

int selectionSort(int arr[]) 直接选择排序

int quickSort(int arr[], int M, int N) 快速排序

int merge(int r[], int r2[], int S, int M, int N) 归并排序递归

int mergeSort(int r[], int r2[], int S, int N) 归并排序

int createNode(Node \*L) 建立链表

int sortNode(Node \*L) 对链表节点进行倒序

int combine(int arr[], Node \*L2) 结合两张表

int main() 主函数，建立链表节点，流程控制

<3>三组测试数据

1)原始数据：顺序表：642189，使用直接插入排序 链表97654

预期结果：12445667899

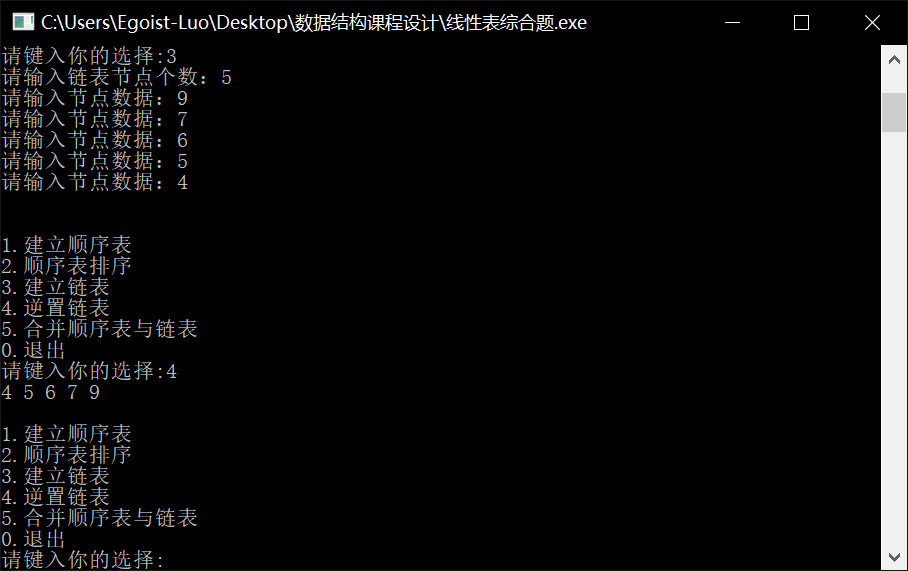
实际结果：













2)原始数据：顺序表：47213，使用冒泡排序 链表521

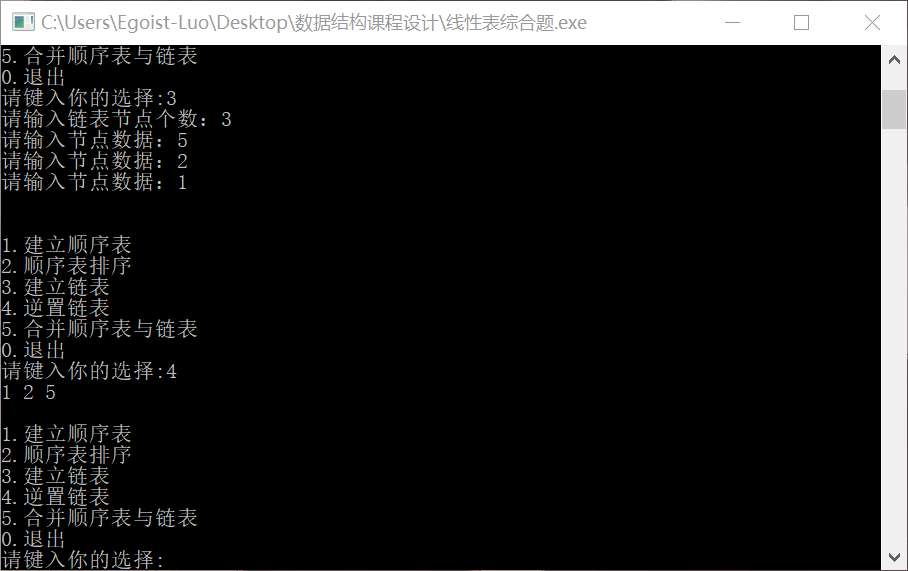
预期结果：11223457

实际结果：











3)原始数据：顺序表：8962，使用归并排序 链表65321

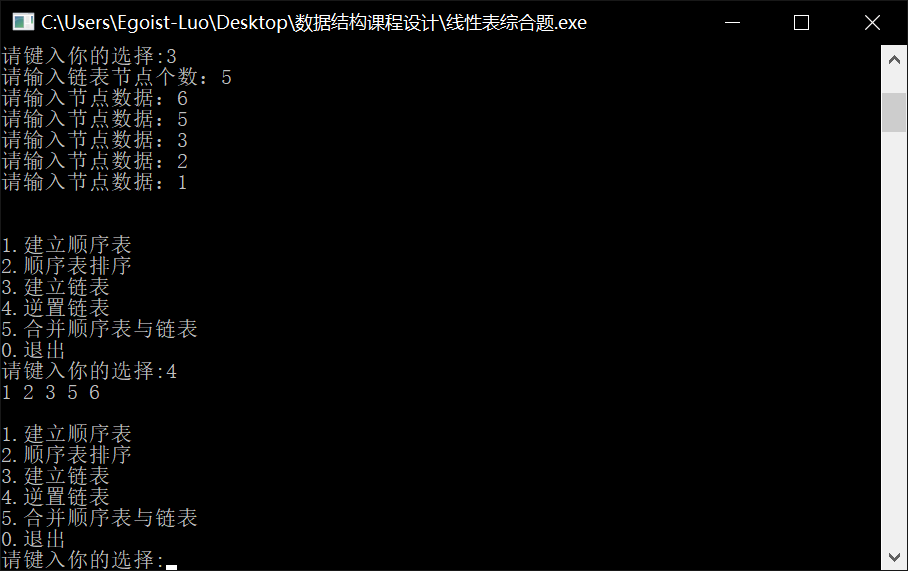
预期结果：12235689

实际结果：











<4>含注释的源程序：

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef struct { /\*链表节点\*/

int data;

struct Node \*next;

}Node;

int array[100]; /\*线性表用数组存储\*/

int count; /\*顺序表计数器\*/

int createList (int arr[]) { /\*建立顺序表\*/

int i;

arr[0] = 0;

printf("请输入顺序表长度：");

scanf("%d", &count);

printf("请输入顺序表节点数据:\n");

for (i = 1; i <= count; i++) {

scanf("%d", &arr[i]);

}

return 1;

}

int printList(int arr[]) { /\*打印顺序表\*/

int i;

for (i = 1; i <= count; i++) { /\*打印排序结果\*/

printf("%d ", arr[i]);

}

printf("\n");

return 1;

}

int sort(int arr[]) { /\*排序列表\*/

int M = 1, N = count, i, ch;/\*建立快速，归并排序所需数组\*/

int r1[N + 1], r2[N + 1];

for (i = 1; i <= N; i++) {

r1[i - 1] = arr[i];

}

system("CLS");

printf("1.直接插入排序\n");

printf("2.冒泡排序\n");

printf("3.直接选择排序\n");

printf("4.快速排序\n");

printf("5.归并排序\n");

printf("请输入你的选择：");

scanf("%d", &ch);

switch(ch) {

case 1:insertSort(arr);break;

case 2:bubbleSort(arr);break;

case 3:selectionSort(arr);break;

case 4:quickSort(arr, M, N);break;

case 5:mergeSort(r1, r2, 0, N - 1);

for (i = 1; i <= N; i++) {

arr[i] = r1[i - 1];

}break;

}

}

int insertSort(int arr[]) { /\*直接插入排序\*/

int i, j;

for (i = 2; i <= count; i++) {

arr[0] = arr[i]; /\*array[0]作为监督哨\*/

j = i - 1;

while (arr[0] < arr[j]) { /\*从后向前找到第一个关键字不小于arr[0]的记录\*/

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = arr[0];

}

return 1;

}

int bubbleSort(int arr[]) { /\*冒泡排序\*/

int i, j, temp;

for (i = 1; i <= count; i++) {

for (j = 1; j <= count - i; j++) {

if (arr[j] > arr[j + 1]) {

temp = arr[j + 1];

arr[j + 1] = arr[j];

arr[j] = temp;

}

}

}

return 1;

}

int selectionSort(int arr[]) { /\*直接选择排序\*/

int i, j, temp, m;

for (i = 1; i <= count; i++) {

m = i;

for (j = i + 1; j <= count; j++) { /\*找到最小的数字\*/

if(arr[j] < arr[m]) {

m = j;

}

if (i != m) {

temp = arr[i];

arr[i] = arr[m];

arr[m] = temp;

}

}

}

}

int quickSort(int arr[], int M, int N) { /\*快速排序\*/

int i, j, x;

i = M, j = N;

x = arr[i];

do {

while ((arr[j] >= x) && (j > i)) {

j--;

}

if (i < j) {

arr[i] = arr[j];

i++;

while (arr[j] <= x && (i < j)) {

i++;

}

if (i < j) {

arr[j] = arr[i];

j--;

}

}

}while (i != j);

arr[i] = x;

i++;

j--;

if (M < j) {

quickSort(arr, M ,j);

}

if (i < N) {

quickSort(arr, i, N);

}

}

int merge(int r[], int r2[], int S, int M, int N) {

int i,k,j;

i = S, k = S, j = M + 1; /\*从S开始\*/

while ((i != M + 1) && (j != N + 1)) { /\*当两个表都有内容未排完时\*/

if (r[i] > r[j]) {

r2[k] = r[j];

j++;

} else {

r2[k] = r [i];

i++;

}

k++;

}

while (i != M + 1) { /\*将剩下的全部放入\*/

r2[k++] = r[i++];

}

while (j != N + 1) {

r2[k++] = r[j++];

}

for (i = S; i <= N; i++) {

r[i] = r2[i];

}

}

int mergeSort(int r[], int r2[], int S, int N) { /\*归并排序\*/

int M;

if (S < N) {

M = (S + N)/2;

mergeSort(r, r2, S, M);

mergeSort(r, r2, M + 1, N);

merge(r, r2, S, M, N);

}

}

int createNode(Node \*L) { /\*建立链表\*/

Node \*q;

int m, n, i; /\*节点个数为m，节点数据为n\*/

printf("请输入链表节点个数：");

scanf("%d", &m);

for (i = 0; i < m; i++) {

q = (Node\*)malloc(sizeof(Node));

printf("请输入节点数据：");

scanf("%d", &n);

q -> data = n;

L -> next = q;

L = q;

}

L -> next = NULL; /\*尾标识为NULL\*/

printf("\n");

return 1;

}

int sortNode(Node \*L) { /\*对链表节点进行排序，仅交换数据\*/

int i, j, tempData;

Node \*head, \*point; /\*辅助指针point帮助指向head的下一个节点\*/

head = L;

head = head -> next;

int length = 0;

while (head -> next != NULL) { /\*计算链表长度\*/

head = head -> next;

length++;

}

for (i = 0; i < length; i++) { /\*冒泡排序\*/

head = L; /\*将head复位\*/

head = head -> next;

for(j = i; j < length; j++) {

point = head -> next;

if(head -> data >= point -> data) {

tempData = head -> data;

head -> data = point -> data;

point -> data = tempData;

}

head = head -> next;

}

}

head = L; /\*指针回到开头\*/

head = head -> next;

while (head != NULL) { /\*将排序后的线性表显示出来\*/

printf("%d ", head -> data);

head = head -> next;

}

printf("\n");

return 1;

}

int combine(int arr[], Node \*L2) { /\*结合两张表\*/

int a[100];

int i, j, temp;

for (i = 1; i <= count; i++) {

a[i - 1] = arr[i];

}

for (i = 1; i <= count; i++) {

;

}

while (L2 != NULL) {

a[count + 1] = L2 -> data;

count++;

}

for (i = 0; i <= count; i++) {

for (j = 0; j <= count - i; j++) {

if (a[j] > a[j + 1]) {

temp = a[j + 1];

a[j + 1] = a[j];

a[j] = temp;

}

}

}

for (i = 0; i <= count; i++) {

printf("%d ", a[i]);

}

}

int main() {

int choose;

Node \*L2;

L2 = (Node\*)malloc(sizeof(Node));

while(1) {

printf("1.建立顺序表\n");

printf("2.顺序表排序\n");

printf("3.建立链表\n");

printf("4.逆置链表\n");

printf("5.合并顺序表与链表\n");

printf("0.退出\n");

printf("请键入你的选择:");

scanf("%d", &choose);

switch(choose) {

case 1:createList(array);printList(array);break;

case 2:sort(array);printList(array);break;

case 3:createNode(L2);break;

case 4:sortNode(L2);break;

case 5:combine(array, &L2);break;

case 0:exit(0);

}

printf("\n");

}

}

<5>编译与运行情况：编译正常通过，运行正常。

(10)二叉树应用

（1）建立一棵二叉树。

（2）前序、中序、层次非递归遍历该二叉树。

（3）判断该二叉树是否为二叉排序树。

（4）如果是二叉排序树，进行结点的插入或删除。

（5）输出结果。

<1>解题思路：首先设计一个结构体，确定需要输入的数据类型，再设计一个结构体，用来存放左右孩子的指针。输入数据建立一个二叉树，首先输入左子树的数据，以‘0’以表示最后的数据作为叶子结点，再输入右子树，并以同样的方式结尾，构成二叉树。接下来进行二叉树的非递归的先序、中序、层次遍历。然后判断该树是否为二叉排序树，则先判断是否是空树，是则不是二叉排序树，不是则递归调用并且遍历左子树，检查左子树是否符合二叉排序树，一旦发现有数据大于根节点，则不是二叉排序树；若没有，则遍历右子树，检查右子树是否符合二叉排序树特征，有发现数据小于根节点，则不是二叉排序树，如果两者都不是，此二叉树就是二叉排序树。在判断为二叉排序树后，调用查找函数，插入数据，以及删除数据。

<2>函数调用图：

开始

判断是否为二叉树

非递归后序遍历

非递归中序遍历

非递归先序遍历

建立二叉树

是

插入数据

删除数据

结束

输出二叉树

各函数功能：

int createTree(BitTree \*T) 创建二叉树

int preOrder(BitTree T) 非递归先序遍历

int preOrder(BitTree T) 非递归先序遍历

int levelPrint(BitTree T) 非递归层次遍历

int judge(BitTree T) 判断是否为二叉树

int searchBst(BitTree T, BitTree G, Elemtype key, BitTree \*p)查找节点

int insertBst(BitTree T, Elemtype key) 二叉排序树插入

int deleteBst(BitTree \*T, int key) 删除二叉排序树

int Delete(BitTree \*p) 删除节点

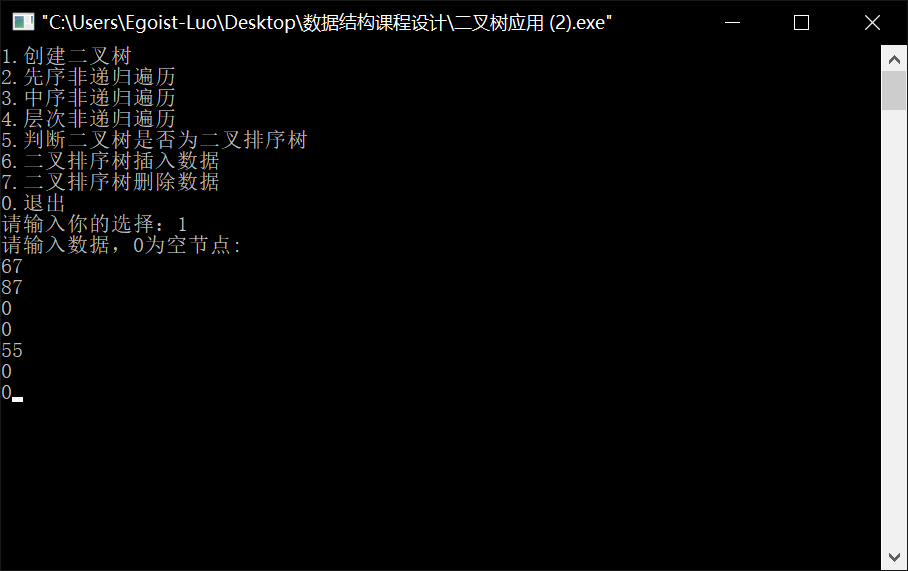
int main() 主函数，流程控制

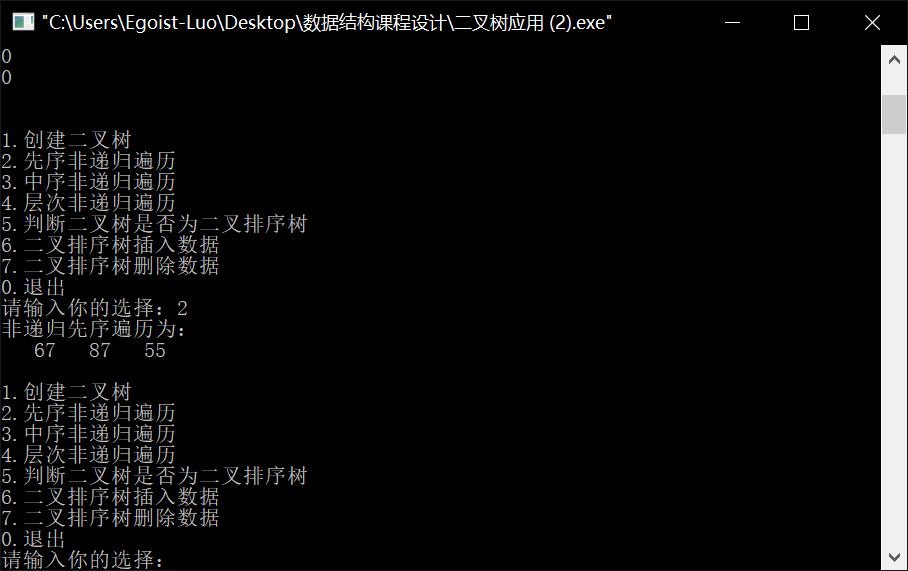
<3>三组测试数据

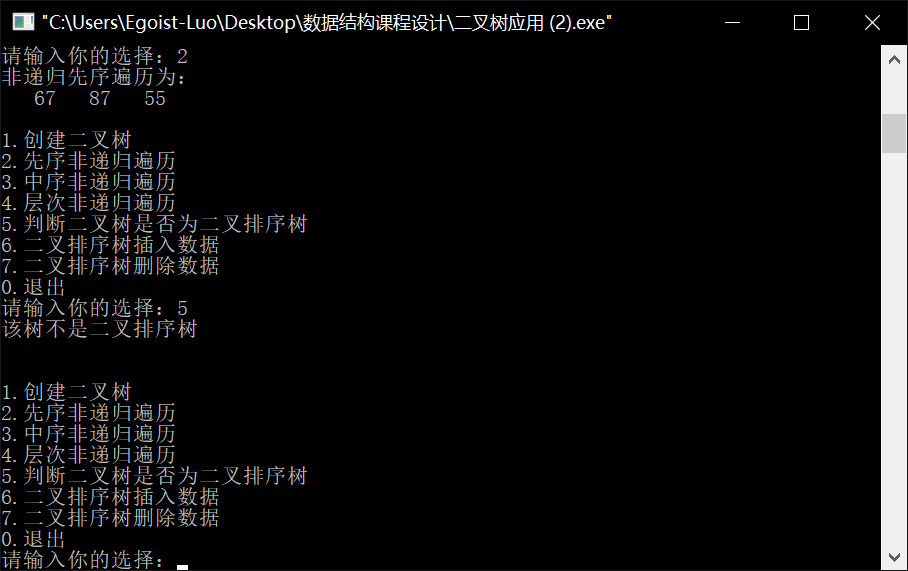
1)原始数据：67 87 0 0 55 0 0 非递归先序遍历

预期结果：不是二叉排序树

实际结果：





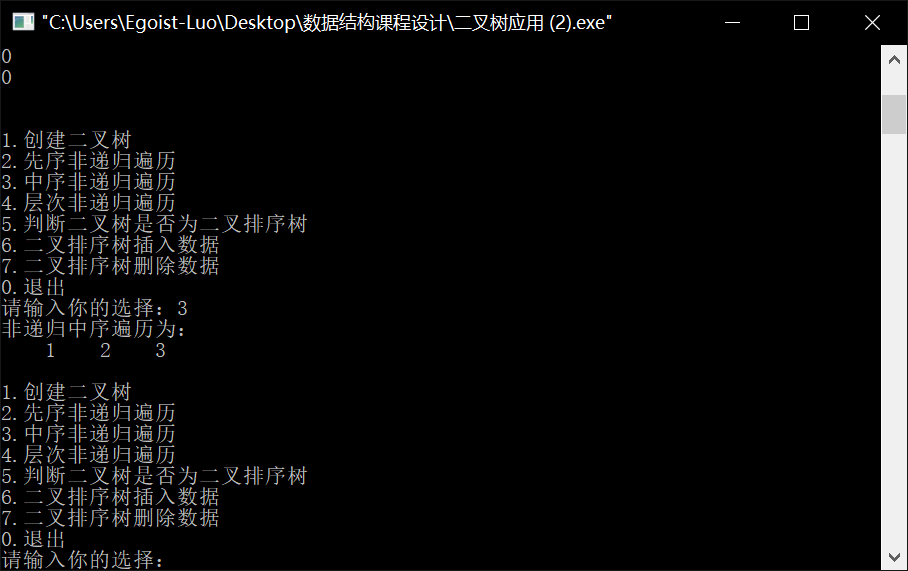


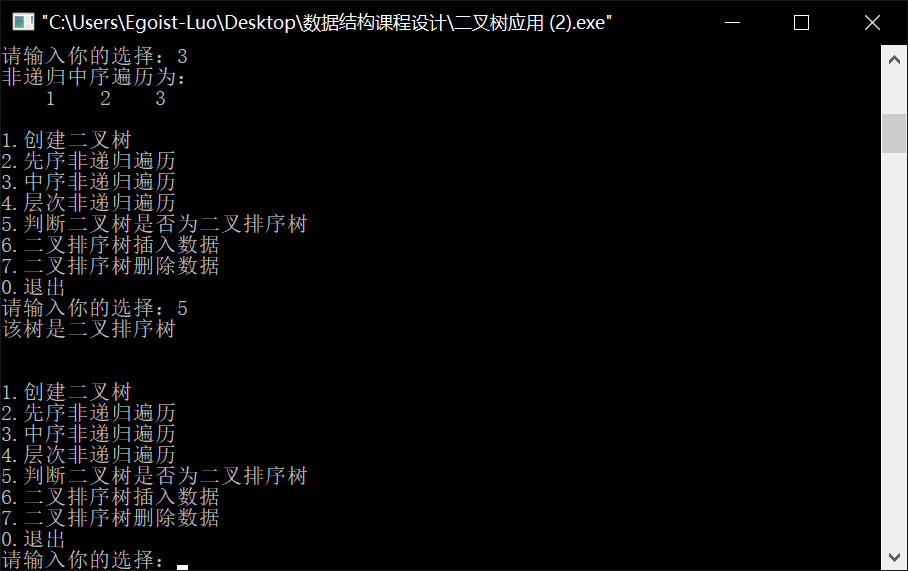
2)原始数据：2 1 0 0 3 0 0 非递归中序遍历 删除3，插入6

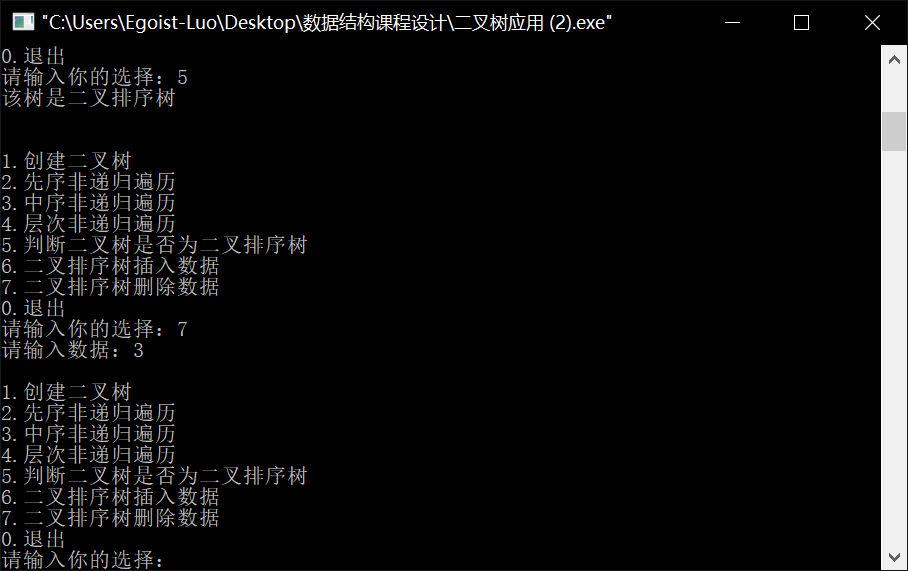
预期结果：是二叉排序树

实际结果：

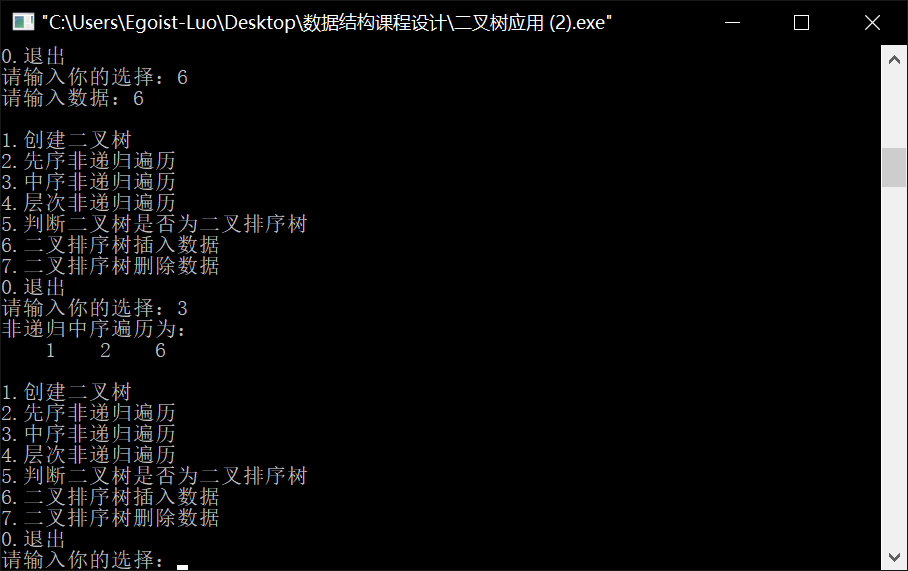








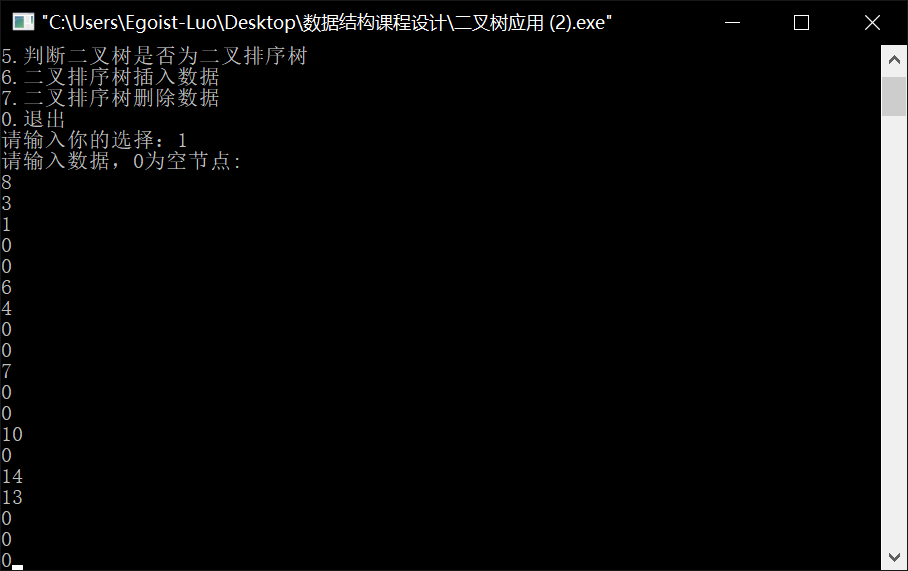


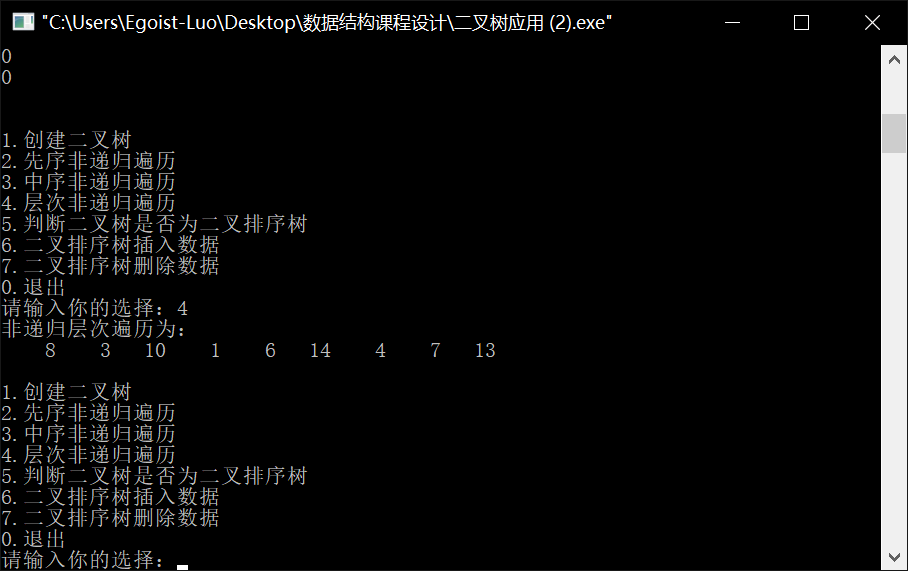


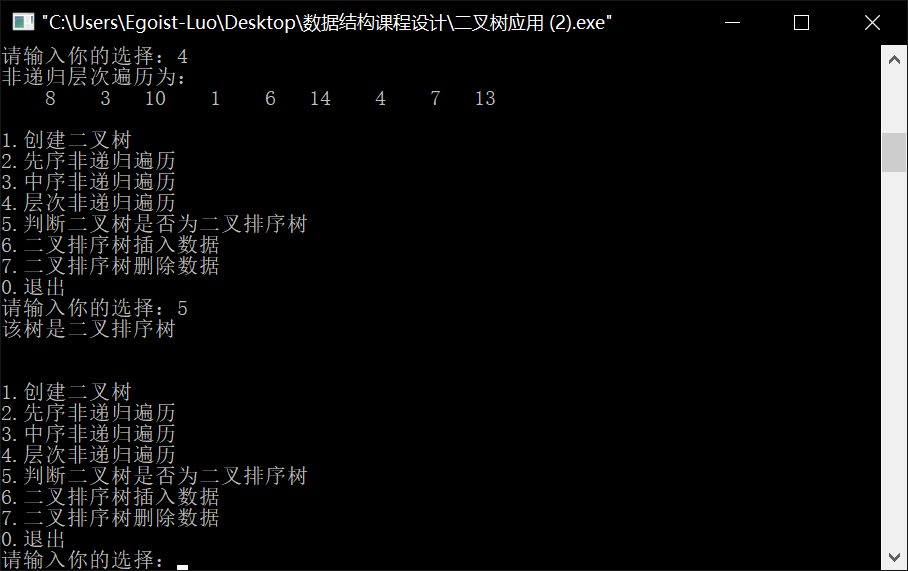
2)原始数据：8 3 1 0 0 6 4 0 0 7 0 0 10 0 14 13 0 0 0 非递归层次遍历 删除6，插入11

预期结果：是二叉排序树

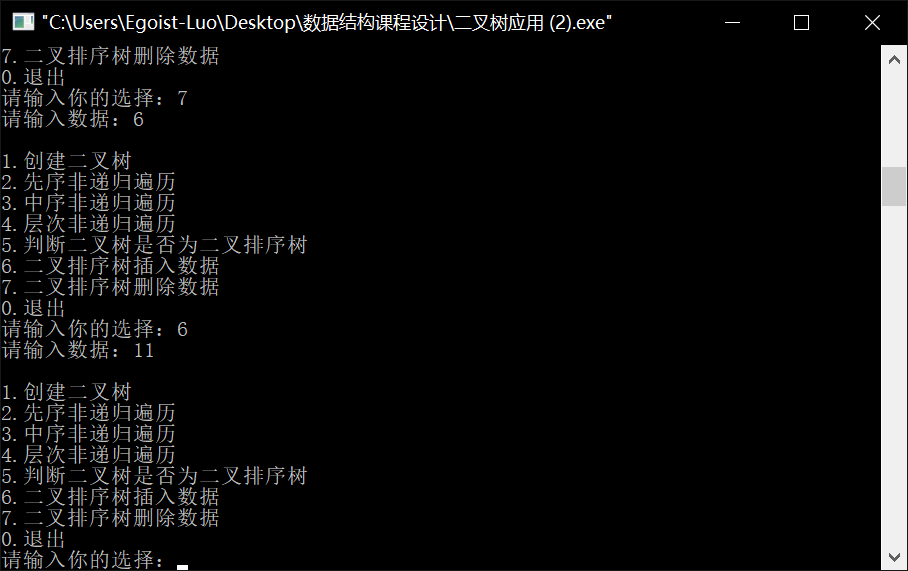
实际结果：

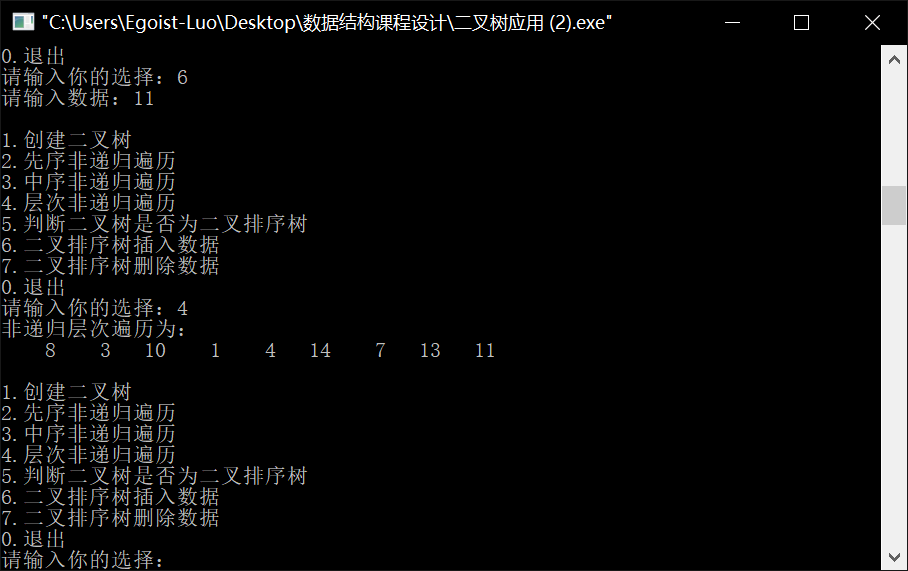












<4>含注释的源程序：

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<stdbool.h>

typedef int Elemtype;

typedef struct {

Elemtype e;

}Elem;

typedef struct BitTree {

Elem data; /\*定义数据\*/

struct BitTree \*lChild, \*rChild; /\*定义左右孩子指针\*/

}BitNode, \*BitTree;

BitNode \*T = NULL;

int createTree(BitTree \*T) { /\*创建二叉树\*/

Elemtype ch;

scanf("%d", &ch);

if(ch == 0) { /\*0表示空树\*/

\*T = NULL;

} else {

\*T = (BitTree)malloc(sizeof(BitNode));

if (!T) {

exit(1);

}

(\*T) -> data.e = ch; /\*生成根节点\*/

createTree(&(\*T) -> lChild); /\*构造左子树\*/

createTree(&(\*T) -> rChild); /\*构造右子树\*/

}

return 1;

}

int preOrder(BitTree T) { /\*非递归先序遍历\*/

BitTree stack[100]; /\*基于栈进行遍历\*/

int front = 0, rear = 0;

BitNode \*p;

if(T != NULL) {

stack[rear] = T; /\*将根节点入栈\*/

rear = (rear + 1) % 100;

}

while (front != rear) {

p = stack[front];

front = (front + 1) % 100;

printf("%5d", p -> data); /\*输出节点\*/

if(p -> lChild != NULL) { /\*左子树不为空就输出左子树\*/

stack[rear] = p -> lChild; /\*左子树入栈\*/

rear = (rear + 1) % 100;

}

if(p -> rChild != NULL) { /\*最后是右子树\*/

stack[rear] = p -> rChild; /\*右子树入栈\*/

rear = (rear + 1) % 100;

}

}

return;

}

int inOrder(BitTree T) { /\*非递归中序遍历\*/

BitNode \*stack[100];

int top;

BitNode \*p;

top = 0;

p = T;

while(p != NULL || top > 0) {

while(p != NULL) { /\*遍历到最左端的节点\*/

stack[top++] = p;

p = p -> lChild;

}

if(top > 0) { /\*出栈，选择当前节点的右子树\*/

p = stack[--top];

printf("%5d", p -> data);

p = p -> rChild;

}

}

}

int levelPrint(BitTree T) { /\*非递归层次遍历\*/

BitTree queue[100];

BitNode \*p;

int front = -1, rear = -1;

rear++;

queue[rear] = T;

while(front != rear) {

front = (front + 1) % 100;

p = queue[front];

printf("%5d", p -> data);

if(p -> lChild != NULL) {

rear = (rear + 1) % 100;

queue[rear] = p -> lChild;

}

if(p -> rChild != NULL) {

rear = (rear + 1) % 100;

queue[rear] = p -> rChild;

}

}

}

int judge(BitTree T) { /\*判断是否为二叉树\*/

if(!T) { /\*返回1代表是二叉排序树\*/

return 1;

} else if(!(T -> lChild)&&!(T -> rChild)) { /\*左右子树都没有\*/

return 1;

} else if((T -> lChild)&&!(T -> rChild)) { /\*只有左子树\*/

if((T->lChild->data.e)>(T->data.e)) {

return 0; /\*返回0代表无二叉树\*/

} else {

return(judge(T -> lChild)); /\*继续寻找子节点\*/

}

}

else if(!(T -> lChild)&&(T -> rChild)) { /\*只有右子树\*/

if((T->rChild -> data.e) < (T -> data.e)) {

return 0;

} else {

return(judge(T -> rChild));

}

} else { /\*如果左右子树都存在\*/

if((T -> lChild -> data.e) > (T -> data.e)||(T -> rChild -> data.e) < (T -> data.e)) {

return 0;

} else {

return (judge(T -> lChild) && judge(T -> rChild));

}

}

}

int searchBst(BitTree T, BitTree G, Elemtype key, BitTree \*p) { /\*查找节点\*/

if(!T) { /\*判断原树是否存在\*/

\*p = G;

return 0;

} else if(key == T -> data.e) {

\*p = T;

return 1;

}

else if(key < T -> data.e) {

return searchBst(T -> lChild, T, key, p);

}

else {

return searchBst(T -> rChild, T, key, p);

}

return 1;

}

int insertBst(BitTree T, Elemtype key) { /\*二叉排序树插入\*/

BitTree p, N;

if(searchBst(T, NULL, key, &p)) {

return 0;

} else {

N = (BitTree)malloc(sizeof(BitNode));

N -> data.e = key;

N -> lChild = NULL;

N -> rChild = NULL;

if(!p) {

T = N;

} else if(key < p -> data.e) {

p -> lChild = N;

} else {

p -> rChild = N;

}

}

return 1;

}

int deleteBst(BitTree \*T, int key) { /\*删除二叉排序树\*/

if(!(\*T)) {

return 0;

} else {

if(key == (\*T) -> data.e) {

Delete(T); /\*找到指定节点，删除\*/

} else if(key < (\*T) -> data.e) {

return deleteBst(&(\*T)->lChild, key);

} else {

return deleteBst(&(\*T)->rChild, key);

}

}

}

int Delete(BitTree \*p) { /\*删除节点\*/

BitTree q, N;

if(!(\*p) -> lChild && !(\*p) -> rChild) {

\*p = NULL;

} else if(!(\*p) -> lChild) {

q = \*p;

free(q);

} else if(!(\*p) -> rChild) {

q = \*p;

free(q);

} else {

q = \*p;

N = (\*p) -> lChild;

while(N -> rChild) {

q = N;

N = N -> rChild;

}

(\*p) -> data.e = N -> data.e;

if(q!=\*p) {

q -> rChild = N -> lChild;

} else {

q -> lChild = N -> lChild;

}

free(N);

}

return 1;

}

int main() {

int ch = 0, flag = 0, key, m;

flag = judge(T);

while(1) {

printf("1.创建二叉树\n");

printf("2.先序非递归遍历\n");

printf("3.中序非递归遍历\n");

printf("4.层次非递归遍历\n");

printf("5.判断二叉树是否为二叉排序树\n");

printf("6.二叉排序树插入数据\n");

printf("7.二叉排序树删除数据\n");

printf("0.退出\n");

printf("请输入你的选择：");

scanf("%d", &ch);

switch(ch) {

case 1:printf("请输入数据，0为空节点:\n");

createTree(&T);printf("\n");break;

case 2:printf("非递归先序遍历为：\n");

preOrder(T);printf("\n");break;

case 3:printf("非递归中序遍历为：\n");

inOrder(T);printf("\n");break;

case 4:printf("非递归层次遍历为：\n");

levelPrint(T);printf("\n");break;

case 5:flag = judge(T);

if(flag)

printf("该树是二叉排序树\n");

else

printf("该树不是二叉排序树\n");

printf("\n");break;

case 6:printf("请输入数据：");

scanf("%d",&key);

insertBst(T, key);break;

case 7:printf("请输入数据：");

scanf("%d",&m);

deleteBst(&T, m);break;

case 0:exit(0);

}

printf("\n");

}

return 1;

}

<5>编译和运行情况：编译正常通过，运行正常。

(11)飞机票预订系统

<1>解题思路：本题需要综合使用数据结构的知识。以此，将航班数据设计成链表形式即定义结构体，其中包含飞机序号，登机口作为数据域，next作为指针域，将此结构体称为Node。将乘客信息设计成特殊的结构体，结构体中包含乘客姓名，性别，目的地，舱位，座位号和身份证，并且用数组包含每个乘客的信息。再设计一个结构体，其中包含刚刚的数组，以及乘客总人数，将此结构体称为Sqlist。而函数执行时，首先将Sqlist初始化。然后进入switch选择，通过选择来执行不同的函数。首先创建航班链表，通过判断输入的飞机序号是不是等于0来判断是否输入完毕。然后输入乘客数据，输入所在飞机号和该飞机乘客容量，使用for循环依次将数组赋值。之后可以通过飞机链表和乘客数据分别查询航班的信息和乘客的信息。并且可以删除飞机信息。最后是将所输入的信息保存成txt格式的文件，以及可以从txt格式的文件中读出数据进行处理。

<2>函数调用图：

开始

设置乘客数据

设置航班数据

从文件中读取

初始化sqlist，node

查询航班

删除航班信息

查询乘客数据

保存数据

结束

各函数功能：

int createPlane(Node \*L) 创建飞机链表

int deleteNode(Node \*L) 删除飞机节点

int initSqList (sqList \*S) 容器初始化

int createSqList (Node \*L, sqList \*S) 构造乘客容器

int createCustomer(int i) 创建新乘客

int searchPlane(Node \*L) 搜寻航班

int searchPlane(Node \*L) 搜寻航班

int reserve(Node \*L, sqList \*S) 保存数据

int read(Node \*L, sqList\*S) 读取数据

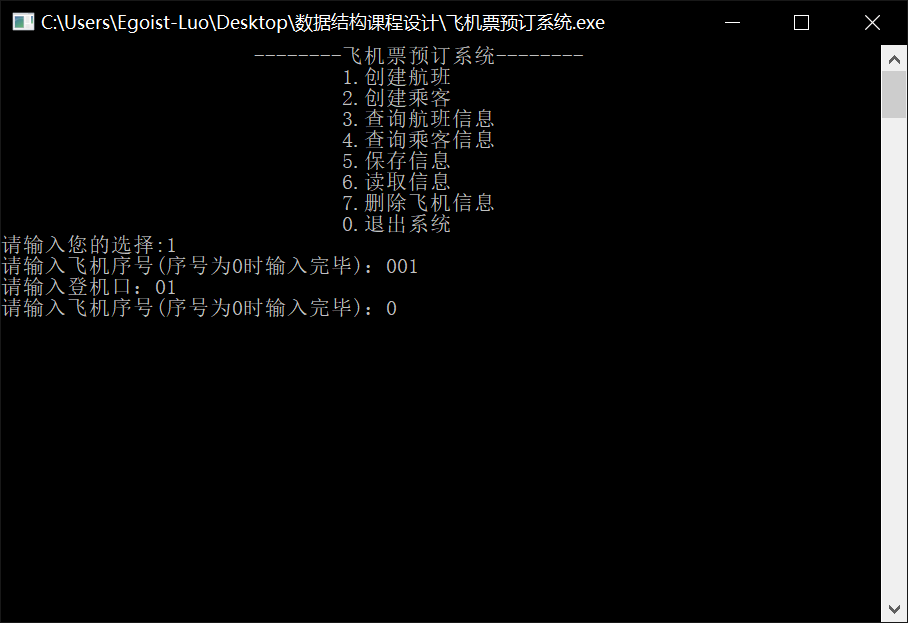
int main() 主函数，流程控制

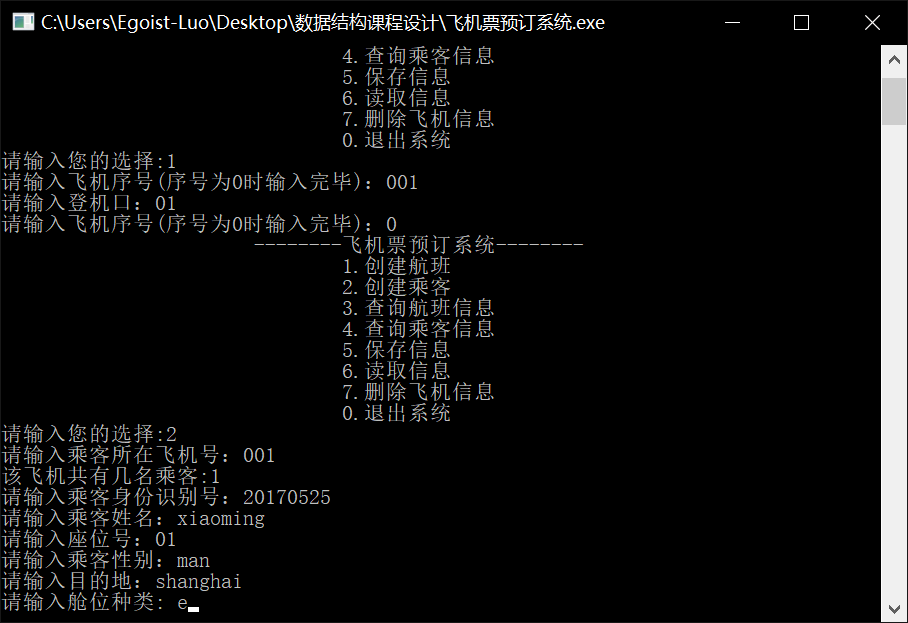
<3>三组测试数据

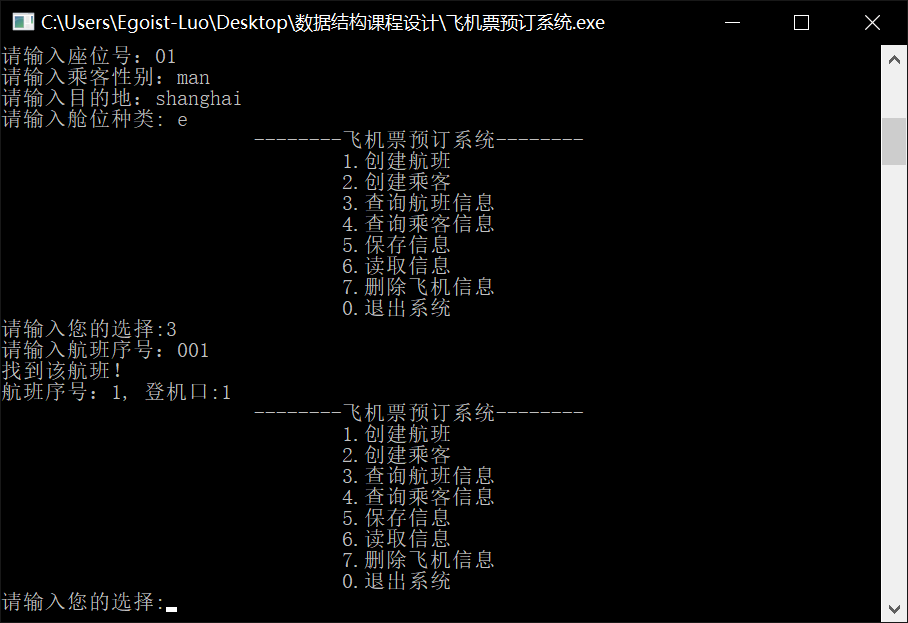
1)原始数据： 航班：飞机序号001，登机口：01；乘客：共有1名乘客，所在飞机：001，身份识别号：20170525，姓名：xiaoming，性别：man，座位号：01，目的地：shanghai，舱位：e（economy）

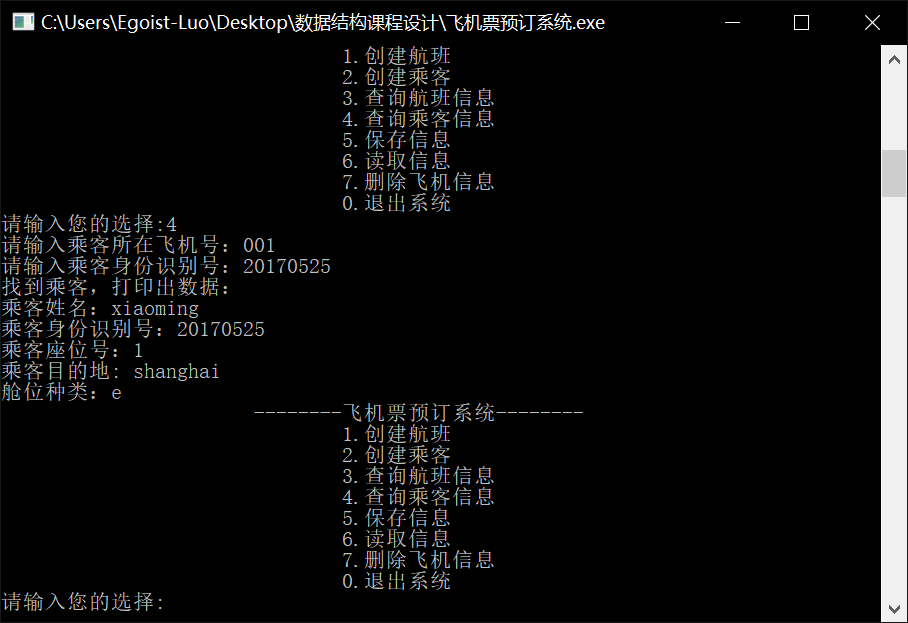
预期结果： 成功显示航班信息和乘客信息

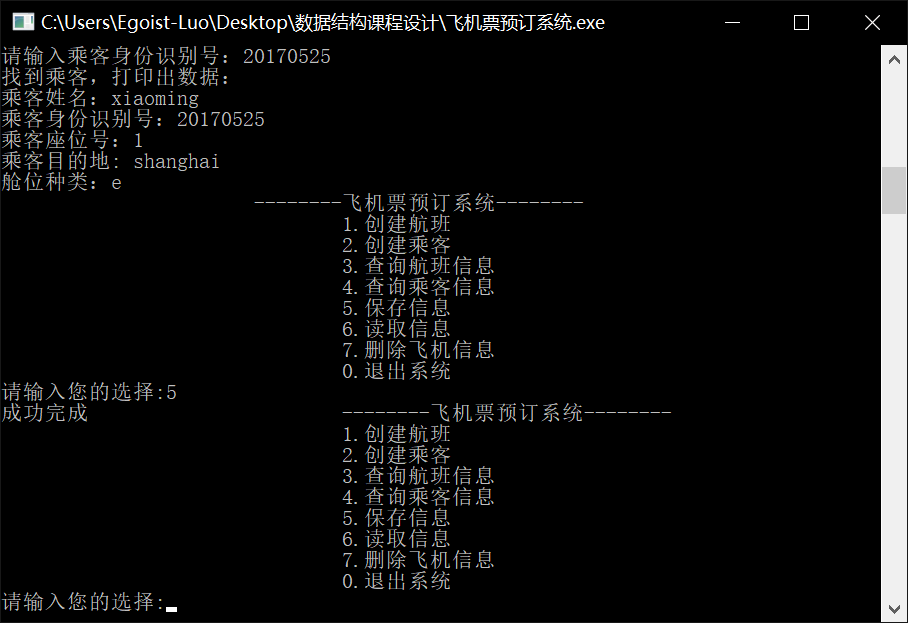
实际结果：

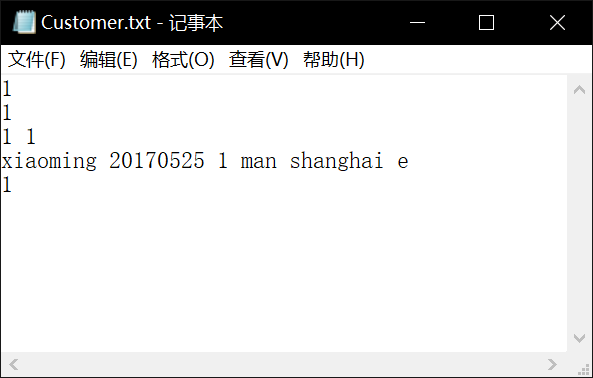








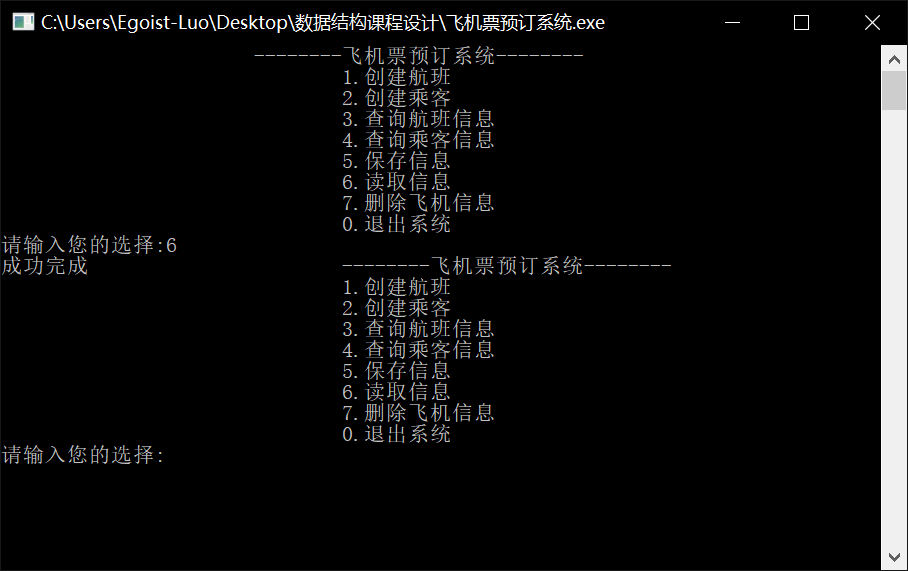


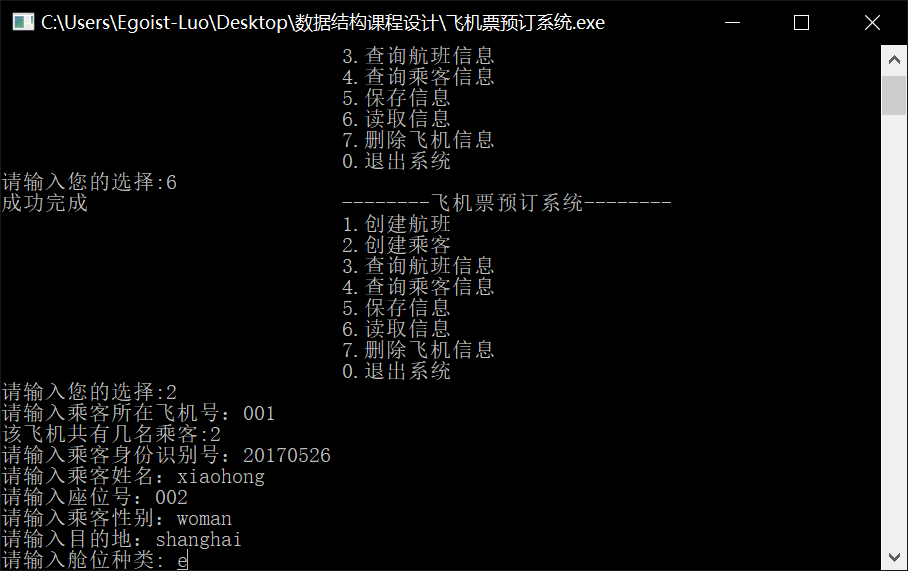


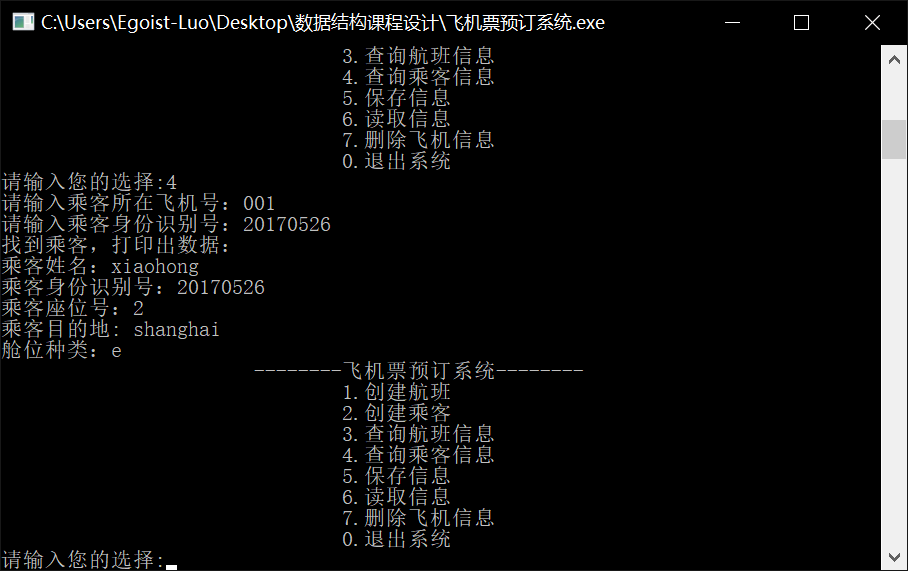
2)原始数据： 在原始数据1上增加乘客：所在飞机：001，身份识别号：20170526，姓名：xiaohong，性别：woman，座位号：02，目的地：shanghai，舱位：e（economy）

预期结果： 成功添加乘客信息

实际结果：



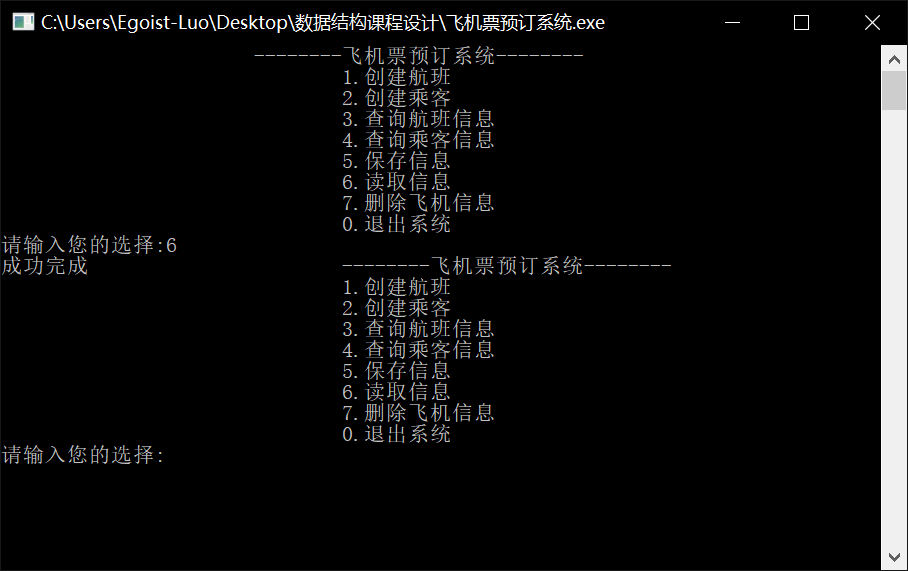




3)原始数据： 在原始数据1上删除航班信息；

预期结果： 找不到001号航班，报错

实际结果：





<4>含注释的源程序：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <malloc.h>

typedef struct customer { /\*乘客结构体\*/

char name[20];

int id;

int seat;

char sex[10];

char goal[10];

char type[1];

}Customer;

typedef struct list { /\*将乘客规划为数组\*/

Customer \*array[25]; /\*最大存放25人\*/

int length;

}sqList;

typedef struct node { /\*将飞机规划为链表\*/

int number;

int input;

struct node \*next;

}Node;

int temp1 = 0; /\*航班计数器\*/

int temp2 = 0; /\*人数计数器\*/

int createPlane(Node \*L) { /\*创建飞机链表\*/

int x, y;

do {

printf("请输入飞机序号(序号为0时输入完毕)：");

scanf("%d", &x);

if (x != 0) {

printf("请输入登机口：");

scanf("%d", &y);

Node \*q; /\*定义新节点\*/

q = (Node\*)malloc(sizeof(Node));

temp1++;

q -> number = x;

q -> input = y;

L -> next = q;

L = q;

}

}while(x != 0); /\*飞机序号为0时结束\*/

L -> next = NULL;

return 1;

}

int deleteNode(Node \*L) { /\*删除飞机节点\*/

int num;

Node \*q;

q = L;

Node \*r = q;

q = q -> next;

printf("请输入要删除的飞机序号（警告该飞机下的乘客全部删除！）：");

scanf("%d", &num);

while(q != NULL) {

if (q -> number == num) {

r -> next = q -> next;

free(q);

temp1--;

return 1;

} else {

q = q -> next;

r = r -> next;

}

}

printf("未找到该飞机！");

return 0;

}

int initSqList (sqList \*S) { /\*容器初始化\*/

S -> length = 0;

return 1;

}

int createSqList (Node \*L, sqList \*S) { /\*构造乘客容器\*/

int x, y, i;

Node \*q;

q = L;

q = q -> next;

printf("请输入乘客所在飞机号：");

scanf("%d", &x);

while (q != NULL) {

if (q -> number == x) {

printf("该飞机共有几名乘客:");

scanf("%d", &y);

for (i = temp2 + 1; i <= y; i++) {

S -> array[i] = createCustomer(i);

S -> length ++;

temp2++;

}

return S;

} else {

q = q -> next;

}

}

printf("未找到该飞机序号！");

return 0;

}

int createCustomer(int i) { /\*创建新乘客\*/

Customer \*q;

q = (Customer\*)malloc(sizeof(Customer));

printf("请输入乘客身份识别号：");

scanf("%d", &(q -> id));

printf("请输入乘客姓名：");

scanf("%s", &(q -> name));

printf("请输入座位号：");

scanf("%d", &(q -> seat));

printf("请输入乘客性别：");

scanf("%s", &(q-> sex));

printf("请输入目的地：");

scanf("%s", &(q -> goal));

printf("请输入舱位种类: ");

scanf("%s", &(q -> type));

return q;

}

int searchPlane(Node \*L) { /\*搜寻航班\*/

int x, y;

Node \*q;

q = L;

printf("请输入航班序号：");

scanf("%d", &x);

while (q != NULL) {

if (x == q -> number) {

printf("找到该航班！\n");

printf("航班序号：%d, 登机口:%d\n", q -> number, q -> input);

return 1;

} else {

q = q -> next;

}

}

printf("未找到该航班\n");

return 0;

}

int searchCustomer(Node \*L, sqList \*S) { /\*搜寻乘客\*/

int x, y, i;

char s;

Node \*q;

q = L;

q = q -> next;

printf("请输入乘客所在飞机号："); /\*检测飞机是否存在\*/

scanf("%d", &x);

while(q != NULL) {

if (q -> number == x) {

break;

} else {

q = q -> next;

}

}

if (q == NULL) {

printf("未找到该飞机\n");

return 0;

}

printf("请输入乘客身份识别号：");

scanf("%d", &y);

for (i = 1; i <= temp2; i++) {

if (y == S -> array[i] -> id) {

printf("找到乘客，打印出数据：\n");

printf("乘客姓名：%s\n", S -> array[i] -> name);

printf("乘客身份识别号：%d\n", S -> array[i] -> id);

printf("乘客座位号：%d\n", S -> array[i] -> seat);

printf("乘客目的地: %s\n", S -> array[i] -> goal);

printf("舱位种类：%s\n", S -> array[i] -> type);

return 1;

}

}

printf("未找到该乘客\n");

return 0;

}

int reserve(Node \*L, sqList \*S) { /\*保存\*/

int i;

L = L -> next;

FILE \*fp = NULL;

if ((fp = fopen("Customer.txt", "w")) == NULL) {

printf("不能打开文件！");

return 0;

}

fprintf(fp, "%d\n", temp1);

fprintf(fp, "%d\n", temp2);

while (L != NULL) { /\*保存飞机节点\*/

fprintf(fp,"%d %d\n", L -> number, L -> input);

L = L -> next;

}

for (i = 1; i <= temp2; i++) { /\*保存乘客节点\*/

fprintf(fp, "%s %d %d %s %s %s\n", S -> array[i] -> name, S -> array[i] -> id, S -> array[i] -> seat,

S -> array[i] -> sex, S -> array[i] -> goal, S -> array[i] -> type);

}

fprintf(fp, "%d\n", S -> length);

fclose(fp);

printf("成功完成");

return 1;

}

int read(Node \*L, sqList\*S) { /\*读取\*/

FILE \*fp = NULL;

int i = 0;

Node \*q;

if ((fp = fopen("Customer.txt", "r")) == NULL) {

printf("不能打开文件！");

return 0;

}

fscanf(fp, "%d\n", &temp1);

fscanf(fp, "%d\n", &temp2);

for (i = 0; i < temp1; i++) { /\*读取飞机数据\*/

q = (Node\*)malloc(sizeof(Node));

fscanf(fp, "%d %d\n", &q -> number, &q -> input);

L -> next = q;

L = q;

}

for (i = 1; i <= temp2; i++); {

Customer \*temp;

temp = (Customer\*)malloc(sizeof(Customer));

fscanf(fp,"%s %d %d %s %s %s\n", &temp -> name, &temp -> id, &temp -> seat, &temp -> sex, &temp -> goal,

&temp -> type);

S -> array[i] = temp;

}

fscanf(fp ,"%d\n", &S -> length);

fclose(fp);

printf("成功完成");

return 1;

}

int main() {

Node \*L;

sqList \*S;

L = (Node\*)malloc(sizeof(Node));

S = (sqList\*)malloc(sizeof(sqList));

initSqList(&S);

int ch;

while (1) {

printf(" --------飞机票预订系统--------\n");

printf(" 1.创建航班\n");

printf(" 2.创建乘客\n");

printf(" 3.查询航班信息\n");

printf(" 4.查询乘客信息\n");

printf(" 5.保存信息\n");

printf(" 6.读取信息\n");

printf(" 7.删除飞机信息\n");

printf(" 0.退出系统\n");

printf("请输入您的选择:");

scanf("%d", &ch);

switch (ch) {

case 1:createPlane(&L);break;

case 2:S = createSqList(&L, &S);break;

case 3:searchPlane(&L);break;

case 4:searchCustomer(&L, S);break;

case 5:reserve(&L, S);break;

case 6:read(&L, &S);break;

case 7:deleteNode(&L);break;

case 0:exit(0);

}

}

}

<5>编译和运行情况：编译正常通过，运行正常。缺点：无法有效添加飞机节点，需要全部重置。并且最大乘客数只有25。

3．短学期小结

设计基础题以及应用题1、4、6、7，这些题主要是通过简单链表来实现需求的。这些题目的练习让我加深了对链表结构的理解，对于课上所讲的删除链表节点的操作更加熟记于心。而设计基础题以及应用题2、3、5，这些题目主要是应用栈的的练习，考察的是进栈和出栈的操作，栈和链表的结构不一样，栈是连续形成的，而链表是通过next指针域所连接的。最后，应用题8是进行二叉树的操作，我在这道题上卡了不少时间，这道题难点在于要用到递归的方式来取得先序遍历。

之后是两道综合题，综合题顿时题目量就上了，要求不难但是任务多，导致代码量增加。这两道题更加是针对基础的C语言的操作实践。在其中遇到了不少的难点，也更加考验我的细心和耐心。在这其中我学会了调试的小技巧。

最后是一道扩展题。这道题要求是把实际的，抽象的需求转化为实际的目标。刚刚开始我使用了不合适的数据结构导致自己调试困难，任务也没有完成，并且占用空间大。经过参考我选择了数组作为主要的数据结构进行使用，最后达到了满意的效果。

4．参考书目

<1>书名：数据结构（C语言版）

作者：严蔚敏 吴伟民

出版社：清华大学出版社

<2>书名：C语言程序设计

作者：谭浩强

出版社：清华大学出版社

<3>书名：数据结构习题分析与解答（C语言版）

作者：袁和金 牛为华 李宗民 刘军 王翠如

出版社：中国电力出版社