教师签名		王铁建	
验证性实验成绩			
1.线性表	2.栈和队列	3.二叉树	4.查找与排序

重庆交通大学

信息科学与工程学院

实验报告

实验名称 1.线性表实验

课程名称 数据结构

专业班级 电子信息类 3 班

学　　号 632107030319

姓　　名 吴术元

指导教师 王铁建

2023 年 5 月

一、 实验内容及算法原理

1.顺序线性表的建立、插入及删除。

2.链式线性表的建立、插入及删除。

1.建立含 n 个数据元素的顺序表并输出该表中各元素的值及顺序表的长度。

2.利用前面的实验先建立一个顺序表 L={21，23，14，5，56，17，31}，然后在第 i 个位置插入元素 68。

3.建立一个带头结点的单链表，结点的值域为整型数据。要求将用户输入的数据按尾插入法来建立相应单链表。

二、核心代码

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

#define ListSize 50
typedef int DataType;

//线性表的顺序存储方式
typedef struct {
    DataType data[ListSize];
    int l;
}SeqList;


//创建顺序线性表
void CreateList(SeqList* A, int n)
{
    int i;
    for (i = 0;i < n;i++)
    {
        scanf_s("%d", &(A->data[i]));
    }
    A->l = n;
}


//在顺序线性表中插入某个元素
void InsertList(SeqList* A, DataType x, int i)
```

```
{
    int j;
    if (i<1 || i>A->l)       //插入时的条件
    {
        printf("插入位置错误!\n");
        exit(0);
    }
    else
    {
        printf("插入成功!\n");
    }
    if (A->l >= ListSize)
    {
        printf("列表溢出!\n");
        exit(0);
    }
    for (j = A->l - 1;j >= i - 1;j--)
    {
        A->data[j + 1] = A->data[j];          //插入时，把各个元素向后移动后，然后在进行
插入
    }
    A->data[i - 1] = x;
    A->l++;
}


//输出线性表
void DisList(SeqList* L)
{
    int i;
    for (i = 0;i < L->l;i++)
        printf("%d ", L->data[i]);
    printf("\n");
}


void main()
{
    SeqList* A = (SeqList*)malloc(sizeof(SeqList));
    int a=0 ;
    printf("请输入顺序表长度:\n");
    scanf_s("%d", &a);
    printf("请输入整型元素:\n");
    CreateList(A, a);
```

```
    printf("输出SeqList的长度: \n");
    printf("长度=%d\n", A->l);
    printf("表内元素为");
    DisList(A);
    DataType x;
    printf("请输入需要插入的元素的位置!\n");
    int i;
    scanf_s("%d", &i);
    printf("请输入需要插入的元素!\n");
    scanf_s("%d", &x);
    InsertList(A, x, i);
    printf("长度=%d\n", A->l);
    printf("表内元素为");
    DisList(A);

}
```

链式结构线性表

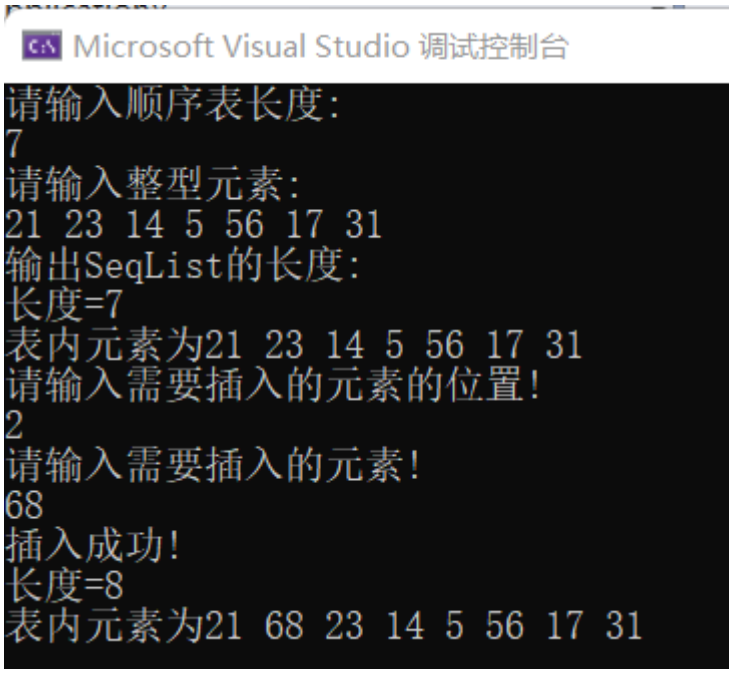
```
#include <stdlib.h>
#include <stdio.h>
struct Node {
    int value;
    Node* next;
};
Node* buildList(int arr[], int n)
{
    if (n == 0) return NULL;
    Node* head = new Node();
    head->value = arr[0];
    head->next = NULL;
    Node* p = head, * q;
    for (int i = 1; i < n; i++) {
        q = new Node();
        q->value = arr[i];
        q->next = NULL;
        p->next = q;
        p = p->next;
    }
    return head;
}
void printList(Node* head)
```

```
{
    Node* p = head;
    while (p != NULL) {
        printf("%d ", p->value);
        p = p->next;
    }
    printf("\n");
}
int main()
{
    int n, a[100];
    printf("输入节点的个数:");
    scanf_s("%d", &n);
    printf("输入节点的值:");
    for (int i = 0; i < n; i++)
        scanf_s("%d", &a[i]);
    Node* list = buildList(a, n);
    printList(list);
    //float x;
    //scanf("%f", &x);
}
```

三、实验结果及分析

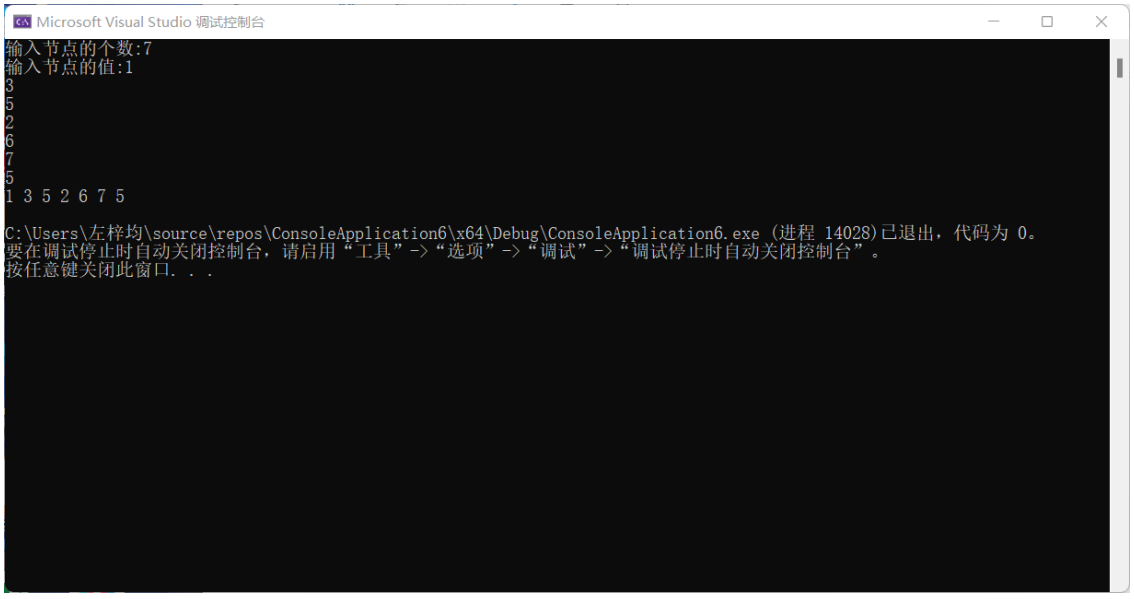
顺序结构线性表

实验结果：



链式结构线性表

实验结果：



实验分析：

对于线性链表和顺序表都属于线性表问题，但是线性链表的插入删除比顺序表要简单，方便。顺序表的查找比较方便，线性链表做元素寻找时，必须从头结点开始寻找。各有各的优缺点。

经过这次实验，对于线性表的顺序结构的相关代码已基本熟悉，算法知识得到了复习与巩固。在写代码的过程与调试中，在解决问题过程中，丰富了个人编程的经历和经验，提高了个人解决问题的能力。

重庆交通大学

信息科学与工程学院

实验报告

实验名称 2.栈和队列实验

课程名称 数据结构

专业班级 电子信息类 3 班

学　　号 632107030319

姓　　名 吴术元

指导教师 王铁建

2023 年 5 月

一、 实验内容及算法原理

1. 编写一个程序实现顺序栈的各种基本运算。

2. 实现队列的链式表示和实现。

二、核心代码

顺序栈

```
#define TRUE       1
#define FALSE      0
#define OK         1
#define ERROR      0
#define INFEASIBLE -1
#define OVERFLOW   -2

typedef int Status;
typedef int SElemType;

//----- 栈的顺序存储表示 -----
#define  STACK_INIT_SIZE  100
#define  STACKINCREMENT   10

typedef struct {
    SElemType* base;
    SElemType* top;
    int  stacksize;
} SqStack;

Status InitStack(SqStack& S);
Status DestroyStack(SqStack& S);
Status StackDisplay(SqStack& S);
Status GetTop(SqStack S, SElemType& e);
Status Push(SqStack& S, SElemType e);
Status Pop(SqStack& S, SElemType& e);
Status StackEmpty(SqStack S);

Status InitStack(SqStack& S) {//构造一个空栈S
    S.base = (SElemType*)malloc(STACK_INIT_SIZE * sizeof(SElemType));
    if (!S.base) exit(OVERFLOW); //存储分配失效
```

```
    S.top = S.base;
    S.stacksize = STACK_INIT_SIZE;
    return OK;
}//InitStack

Status DestroyStack(SqStack& S) {//销毁栈S
    if (S.base) free(S.base);
    S.top = S.base = NULL;
    return OK;
}//InitStack

Status StackDisplay(SqStack& S) {//显示栈S
    SElemType* p = S.base;
    int i = 0;
    if (S.base == S.top) {
        printf("堆栈已空!\n");
        return OK;
    }
    while (p < S.top)
        printf("[%d:%d]", ++i, *p++);
    printf("\n");
    return OK;
}//StackDisplay

Status GetTop(SqStack S, SElemType& e) {
    //若栈不空，则用e返回S的栈顶元素，
    //并返回OK；否则返回ERROR
    if (S.top == S.base) return ERROR;
    e = *(S.top - 1);
    return OK;
}//GetTop

Status Push(SqStack& S, SElemType e) {//插入元素e为新的栈顶元素
    if (S.top - S.base >= S.stacksize) {//栈满，追加存储空间
        S.base = (SElemType*)realloc(S.base,
            (S.stacksize + STACKINCREMENT) * sizeof(SElemType));
        if (!S.base) exit(OVERFLOW);//存储分配失败
        S.top = S.base + S.stacksize;
        S.stacksize += STACKINCREMENT;
    }
    *S.top++ = e;
    return OK;
}//Push
```

```
Status Pop(SqStack& S, SElemType& e) {
    //若栈不为空，则删除S的栈顶元素，
    //用e返回其值，并返回OK；否则返回ERROR
    if (S.top == S.base) return ERROR;
    e = *--S.top;
    return OK;
}//Pop

Status StackEmpty(SqStack S) {
    //若S为空栈，则返回TRUE，否则返回FALSE。
    if (S.top == S.base) return TRUE;
    else return FALSE;
}// StackEmpty

void main() {
    SqStack St;
    Status temp;
    int flag = 1, ch;
    int e;
    InitStack(St);       //初始化堆栈St
    while (flag) {
        printf("请选择:\n");
        printf("1.显示栈中所有元素     \n");
        printf("2.入栈                 \n");
        printf("3.出栈                 \n");
        printf("4.取栈顶元素           \n");
        printf("5.退出程序             \n");
        scanf_s("%d", &ch);
        switch (ch) {
        case 1:
            StackDisplay(St);
            break;
        case 2:
            printf("请输入要入栈的元素(一个整数):");
            scanf_s("%d", &e);      //输入要入栈的元素
            temp = Push(St, e);      //入栈
            if (temp != OK) printf("堆栈已满!入栈失败!\n");
            else {
                printf("成功入栈!\n");  //成功入栈
                StackDisplay(St);
            }
            break;
        case 3:
            temp = Pop(St, e);  //出栈
```

```
            if (temp == ERROR) printf("堆栈已空!\n");
            else {
                printf("成功出栈一个元素:%d\n", e); //成功出栈
                StackDisplay(St);
            }
            break;
        case 4:
            temp = GetTop(St, e); //取得栈顶元素
            if (temp == ERROR) printf("堆栈已空!\n");
            else printf("栈顶元素是:%d\n", e); //显示栈顶元素
            break;
        default:
            flag = 0;
            printf("程序结束，按任意键退出!\n");
            getchar();

        }
    }
    DestroyStack(St);
}
```

链队列

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#define TRUE       1
#define FALSE      0
#define OK         1
#define ERROR      0
#define INFEASIBLE -1
#define OVERFLOW   -2

//Status 是函数的类型,其值是函数结果状态代码
typedef int Status;
//ElemType 是顺序表数据元素类型,此程序定义为int型
typedef int QElemType;

//-----单链队列--队列的链式存储结构-----
typedef struct QNode {       //定义结点结构
    QElemType     data;     //数据域
    struct QNode* next;     //指针域
```

```
}QNode, * QueuePtr;
typedef struct linkqueue {  //定义队列结构
    QueuePtr front;         //队头指针
    QueuePtr rear;          //队尾指针
}LinkQueue;

Status InitLinkQueue(LinkQueue&);           //初始化一个队列
Status DestroyLinkQueue(LinkQueue&);        //销毁一个队列
Status EnLinkQueue(LinkQueue&, QElemType);  //将一个元素入队列
Status DeLinkQueue(LinkQueue&, QElemType&);//将一个元素出队列
Status DisplayLinkQueue(LinkQueue);          //显示队列中所有元素

void main() {
    LinkQueue LQ;
    QElemType e;
    int flag = 1, ch, len;
    Status temp;
    printf("本程序实现链式结构队列的操作。\n");
    printf("可以进行入队列、出队列等操作。\n");
    InitLinkQueue(LQ);       //初始化队列
    while (flag) {
        printf("请选择:\n");
        printf("1.显示队列所有元素\n");
        printf("2.入队列\n");
        printf("3.出队列\n");
        printf("4.退出程序\n");
        scanf_s("%d", &ch);
        switch (ch) {
        case 1:DisplayLinkQueue(LQ);   //显示队列中所有元素
            break;
        case 2:printf("请输入要人队的元素(一个整数):");
            scanf_s("%d", &e);   //输入要入队列的字符
            EnLinkQueue(LQ, e);//入队列
            DisplayLinkQueue(LQ);
            break;
        case 3:temp = DeLinkQueue(LQ, e);  //出队列
            if (temp == OK) {
                printf("出队一个元素:%d\n", e);
                DisplayLinkQueue(LQ);
            }
            else printf("队列为空!\n");
            break;

        default:flag = 0;
```

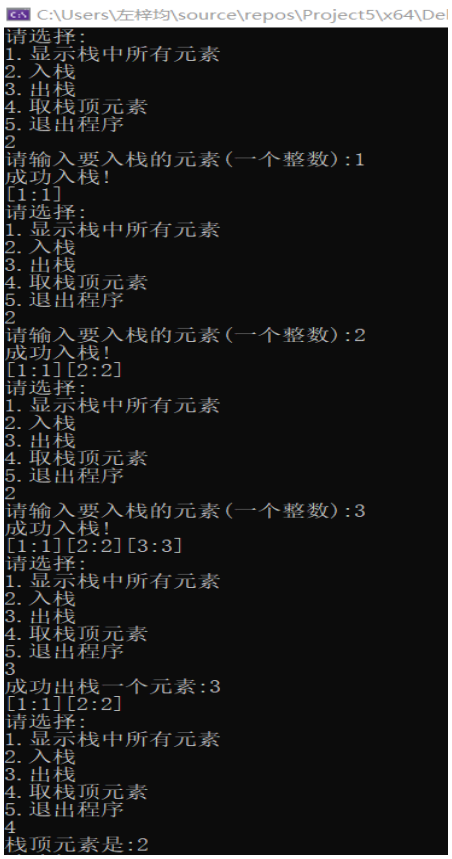
```
            printf("程序运行结束，按任意键退出!\n");
            getchar();
        }
    }
}

Status InitLinkQueue(LinkQueue& Q) {//队列初始化
    Q.front = Q.rear = (QueuePtr)malloc(sizeof(QNode)); //生成一个头结点，并把首尾指针指向头结点
    Q.front->next = NULL;
    return OK;
}
Status DestroyLinkQueue(LinkQueue& Q) {//销毁一个队列
    QueuePtr p;
    QElemType e;
    while (Q.front != Q.rear)
        DeLinkQueue(Q, e);
    free(Q.front);
    Q.front = Q.rear = NULL;
    return OK;
}



Status EnLinkQueue(LinkQueue& Q, QElemType e) {//入队列
    QueuePtr p;
    p = (QueuePtr)malloc(sizeof(QNode));//生成一个新结点
    p->data = e;                        //赋值
    p->next = NULL;
    Q.rear->next = p;                   //插入至队列尾
    Q.rear = p;                         //修改队尾指针
    return OK;
}

Status DeLinkQueue(LinkQueue& Q, QElemType& e) {//出队列
    QueuePtr p;
    if (Q.front == Q.rear) return ERROR;     //判断队列是否已空，已空返回ERROR
    p = Q.front->next;                      //p指向队列中第一个元素
    e = p->data;                            //取得该元素值
    Q.front->next = p->next;                //修改队首指针
    if (Q.rear == p) Q.rear = Q.front;          //若队列已空，把队尾指针指向头结点
    return OK;                             //成功出队列，返回OK
}

Status DisplayLinkQueue(LinkQueue Q) {//显示队列中所有元素
```

```
    QueuePtr p;
    int i = 0;
    p = Q.front->next;
    if (p == NULL) printf("队列为空!\n");//队列为空
    else {
        while (p) {        //否则显示队列中所有元素
            printf("[%d:%d]", ++i, p->data);
            p = p->next;
        }
        printf("\n");
    }
    return OK;
}
```

三、实验结果及分析

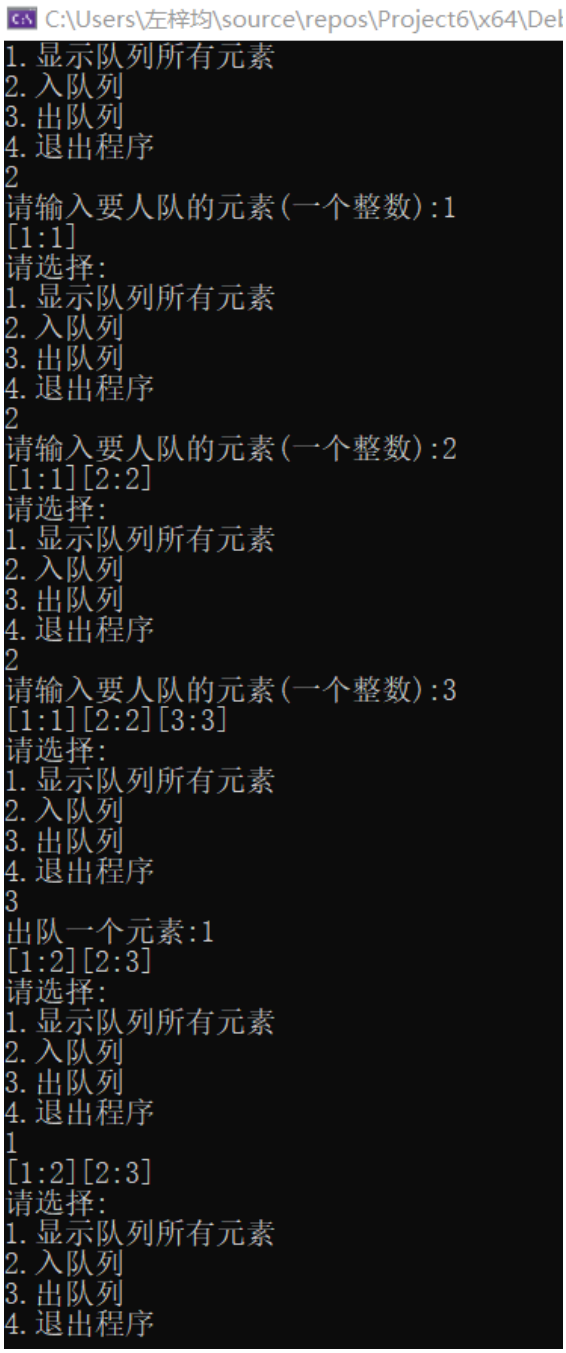
顺序栈

实验结果：



链队列

实验结果：



实验分析：

栈和队列是两种常用的数据结构，栈和队列是操作受限的线性表，栈和队列的数据元素具有单一的前驱和后继的线性关系；栈和队列又是两种重要的抽象数据类型。

栈是限定在表尾进行插入和删除操作的线性表允许插入和删除的一端为栈顶，另一端为栈底，出栈元素只能是栈顶元素，后进先出，相邻元素具有前驱与

后继关系。

队列是只允许在一端进行插入操作，在另一端进行删除操作的线性表。允许插入的一端为队尾，允许删除的一端为队头，先进先出，相邻元素具有前驱与后继关系。

重庆交通大学

信息科学与工程学院

实验报告

实验名称 3.二叉树实验

课程名称 数据结构

专业班级 电子信息类 3 班

学　　号 632107030319

姓　　名 吴术元

指导教师 王铁建

2023 年 5 月

一、 实验内容及算法原理

1. 练习二叉树的建立与存储

2. 练习二叉树的遍历

二、核心代码

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

#define TRUE        1
#define FALSE       0
#define OK          1
#define ERROR       0
#define INFEASIBLE -1
#define OVERFLOW    -2

//Status 是函数的类型,其值是函数结果状态代码
typedef int Status;
//TElemType 是二叉树数据元素类型,此程序定义为char型
typedef char TElemType;

//-----二叉树的二叉链表存储表示-----
typedef struct  BiTNode {      //定义二叉树结点结构
    TElemType  data;                //数据域
    struct BiTNode* lchild, * rchild; //左右孩子指针域
}BiTNode, * BiTree;

Status CreateBiTree(BiTree& T);//生成一个二叉树(可用两种方法输入)
Status CreateBiTreeInPreOrderResult(BiTree& T);//生成一个二叉树(先序遍历结果输入)
Status CreateBiTreeInBracket(BiTree& T);//生成一个二叉树(嵌套括号法输入)
Status PrintElement(BiTree t);
Status PreOrderTraverse(BiTree T, Status(*Visit)(BiTree t)); //先序递归遍历二叉树
Status InOrderTraverse(BiTree T, Status(*Visit)(BiTree t));  //中序递归遍历二叉树
Status PostOrderTraverse(BiTree T, Status(*Visit)(BiTree t));//后序递归遍历二叉树

char* pstr;
```

```
Status CreateBiTree(BiTree& T) {//生成一个二叉树(可用两种方法输入)
    int i, len, choice = 0;
    char str[200];
    printf("请选择建立二叉树的方法:\n");
    printf("1.按先序遍历的结果输入二叉树\n");
    printf("2.按嵌套括号表示法输入二叉树\n");
    do {
        gets_s(str);
        choice = atoi(str);
    } while (choice < 1 || choice>2);
    if (choice == 1) {
        printf("请输入先序遍历二叉树的结果,程序据此建立二叉树。\n");
        printf("对于叶子结点以空格表示。\n");
        printf("例如:abc__de_g__f___(回车), 建立二叉树\n");
        pstr = gets_s(str);
        len = strlen(str);
        for (i = len;i < 180;++i)
            str[i] = ' ';
        str[i] = 0;
        CreateBiTreeInPreOrderResult(T);//初始化二叉树
    }
    else {
        printf("请输入嵌套括号表示法表示的二叉树,程序据此建立二叉树。\n");
        printf("例如:(a(b(c,d(e(,g),f))))(回车), 建立二叉树\n");
        pstr = gets_s(str);
        CreateBiTreeInBracket(T);       //初始化二叉树
    }
    return OK;
}

Status CreateBiTreeInPreOrderResult(BiTree& T) {
    //根据存放在字符串*str中的先序遍历二叉树的结果,生成链接存储的二叉树。
    //(若某结点无左孩子或右孩子,则以空格表示其"孩子")。
    if (!(*pstr) || *pstr == ' ') {
        T = NULL;
        pstr++;
    }
    else {
        T = (BiTNode*)malloc(sizeof(BiTNode)); //生成一个新结点
        if (!T) exit(OVERFLOW);
        T->data = *(pstr++);
        CreateBiTreeInPreOrderResult(T->lchild);  //生成左子树
        CreateBiTreeInPreOrderResult(T->rchild);  //生成右子树
    }
```

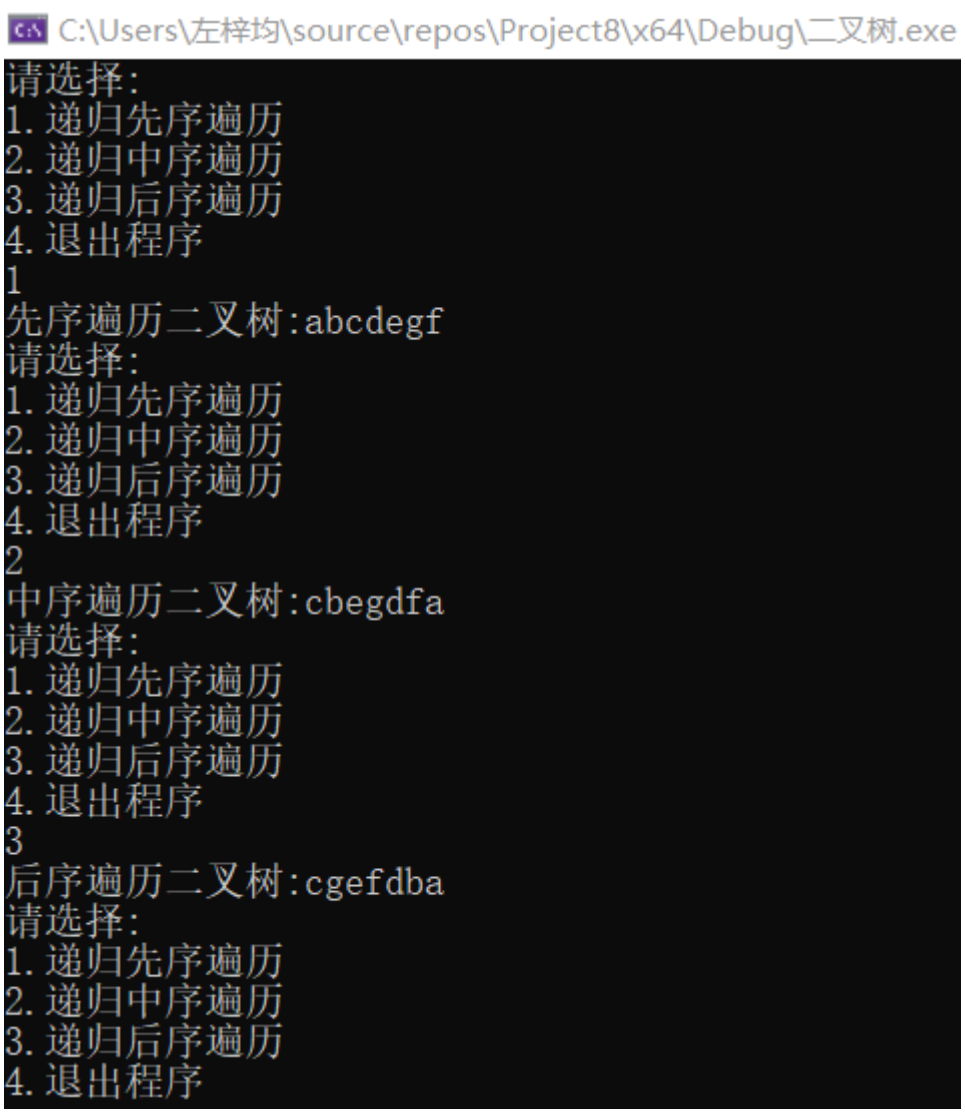
```
    return OK;
}

Status CreateBiTreeInBracket(BiTree& T) {
    //根据嵌套括号表示法的字符串*str生成链接存储的二叉树
    //例如:*pstr="(a(b(c),d(e(,f),g)))"
    BiTree stack[100], p{};
    int top = 0, k;  //top为栈指针,k指定是左还是右孩子;
    T = NULL;
    while (*pstr) {
        switch (*pstr) {
        case '(':stack[top++] = p;k = 1;break;  //左结点,其父结点为*p
        case ')':top--;             break;
        case ',':               k = 2;break;  //右结点,其父结点为*p
        case ' ':                   break;
        default:
            p = (BiTree)malloc(sizeof(BiTNode));
            p->data = *pstr;
            p->lchild = p->rchild = NULL;
            if (!T) T = p;                      //根结点
            else {
                switch (k) {
                case 1:stack[top - 1]->lchild = p;break;
                case 2:stack[top - 1]->rchild = p;break;
                }
            }
        }
        pstr++;
    }
    return OK;
}

Status PrintElement(BiTree t) {
    printf("%c", t->data);  //显示结点数据域
    return OK;
}

Status PreOrderTraverse(BiTree T, Status(*Visit)(BiTree t)) {//先序
    if (T) {
        if ((*Visit)(T))     //访问结点
            if (PreOrderTraverse(T->lchild, Visit))   //遍历左子树
                if (PreOrderTraverse(T->rchild, Visit))   //遍历右子树
                    return OK;
        return ERROR;
```

```
    }
    else return OK;
}

Status InOrderTraverse(BiTree T, Status(*Visit)(BiTree t)) {//中序
    if (T) {
        if (InOrderTraverse(T->lchild, Visit))   //遍历左子树
            if ((*Visit)(T))     //访问结点
                if (InOrderTraverse(T->rchild, Visit))   //遍历右子树
                    return OK;
        return ERROR;
    }
    else return OK;
}

Status PostOrderTraverse(BiTree T, Status(*Visit)(BiTree e)) {//后序
    if (T) {
        if (PostOrderTraverse(T->lchild, PrintElement))   //遍历左子树
            if (PostOrderTraverse(T->rchild, PrintElement))   //遍历右子树
                if ((*Visit)(T))     //访问结点
                    return OK;
        return ERROR;
    }
    else return OK;
}


void main() {
    BiTree T;
    char ch, j;
    char str[200];
    int choice, flag = 1, len, i;
    Status temp;
    printf("本程序实现二叉树的操作:\n");
    printf("可以进行建立二叉树，递归先序、中序、后序遍历等操作。\n");

    CreateBiTree(T);

    while (flag) {
        printf("请选择: \n");
        printf("1.递归先序遍历\n");
        printf("2.递归中序遍历\n");
        printf("3.递归后序遍历\n");;
        printf("4.退出程序\n");
```

```
        scanf_s("%d", &choice);
        switch (choice) {
        case 1:
            if (T) {
                printf("先序遍历二叉树:");
                PreOrderTraverse(T, PrintElement); //先序递归遍历二叉树
                printf("\n");
            }
            else
                printf("二叉树为空!\n");
            break;
        case 2:
            if (T) {
                printf("中序遍历二叉树:");
                InOrderTraverse(T, PrintElement);  //中序递归遍历二叉树
                printf("\n");
            }
            else
                printf("二叉树为空!\n");
            break;
        case 3:
            if (T) {
                printf("后序遍历二叉树:");
                PostOrderTraverse(T, PrintElement);//后序递归遍历二叉树
                printf("\n");
            }
            else
                printf("二叉树为空!\n");
            break;
        default:
            flag = 0;
            printf("程序运行结束，按任意键退出!\n");
            getchar();
        }
    }


}
```

三、实验结果及分析

实验结果：



实验分析：(1).树本身是一种非线性结构，在二叉树的基本操作的算法中多次利用到递归思想，二叉树本身的定义即为递归定义，调用自身定义左右孩子指针；后继的遍历等算法过程中也多次利用到递归；

(2).三种遍历算法的递归算法不同之处仅在于访问根结点和遍历左、右子树的先后关系，需要注意的是，任何一棵二叉树的叶子结点在先序，中序，后序遍历中，其访问的相对次序不变，包括双序遍历仅是在先序遍历的基础上，在访问右子树之前再次访问根结点；

(3).利用栈实现先序，中序遍历时，需要注意在定义栈的存储结构时，其头指针，尾指针的数据类型为指向根结点的指针的指针，队列实现层序遍历同理；

重庆交通大学

信息科学与工程学院

实验报告

实验名称 4.查找与排序实验

课程名称 数据结构

专业班级 电子信息类 3 班

学　　号 632107030319

姓　　名 吴术元

指导教师 王铁建

2023 年 5 月

一、 实验内容及算法原理

设计一个读入一串整数，然后构造二叉排序树，进行查找。

统计成绩

给出 n 个学生的考试成绩表，每条信息由姓名和分数组成，试设计一个算法：

（1） 按分数高低次序，打印出每个学生在考试中获得的名次，分数相同的为同一名次；

（2） 按名次列出每个学生的姓名与分数。

二、核心代码

查找

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
#define ENDKEY 0

typedef int KeyType;

typedef struct  node
{
    KeyType  key; /*关键字的值*/
    struct node* lchild, * rchild;/*左右指针*/
}BSTNode, * BSTree;

int InsertBST(BSTree* bst, KeyType key)
/*若在二叉排序树中不存在关键字等于key的元素，插入该元素*/
{
    BSTree s;
    if (*bst == NULL)
    {
        s = (BSTree)malloc(sizeof(BSTNode));
        s->key = key;
        s->lchild = NULL;
        s->rchild = NULL;
        *bst = s;
    }
    else if (key < (*bst)->key)
```

```
        InsertBST(&((*bst)->lchild), key);
    else if (key > (*bst)->key)
        InsertBST(&((*bst)->rchild), key);
    return 0;
    //请完成本函数的功能
}

void  CreateBST(BSTree* bst)
/*从键盘输入元素的值，创建相应的二叉排序树*/
{
    KeyType key;
    *bst = NULL;
    scanf_s("%d", &key);
    while (key != ENDKEY)
    {
        InsertBST(bst, key);
        scanf_s("%d", &key);
    }
    //请完成本函数的功能
}

void  InOrder(BSTree bst)
/*中序遍历二叉树，root为指向二叉树(或某一子树)根结点的指针*/
{
    if (bst != NULL)
    {
        InOrder(bst->lchild);   /*中序遍历左子树*/
        printf("%d->", bst->key);        /*访问根结点*/
        InOrder(bst->rchild);   /*中序遍历右子树*/
    }
}

BSTree  SearchBST(BSTree bst, KeyType key)
/*在根指针bst所指二叉排序树中，递归查找某关键字等于key的元素，若查找成功，返回指向该元素
结点指针，否则返回空指针*/
{
    if (!bst)
        return NULL;
    else if (bst->key == key)
        return bst;
    else if (bst->key > key)
        return SearchBST(bst->lchild, key);
    else
        return SearchBST(bst->rchild, key);
```

```
    //请完成本函数的功能
}


void main()
{
    BSTree T, p;
    int keyword, temp;
    char ch, j = 'y';
    T = NULL;
    while (j != 'n')
    {
        printf("1.创建二叉排序树\n");
        printf("2.显示排序的数据\n");
        printf("3.查找数据\n");
        printf("4.程序结束，退出\n");
        scanf_s(" %c", &ch); //输入操作选项
        switch (ch)
        {
        case '1':
            printf("请输入数据，以0作为数据输入结束。\n");
            CreateBST(&T);
            break;
        case '2':
            if (!T) printf("二叉排序树中没有数据。\n");
            else { InOrder(T);printf("\b\b  \n"); }
            break;
        case '3':
            printf("输入待查找的数据值:\n");
            scanf_s("%d", &keyword); //输入要查找元素的关键字
            p = SearchBST(T, keyword);
            if (!p) printf("%d 没有找到。\n", keyword); //没有找到
            else printf("%d 查找成功。\n", keyword); //成功找到
            break;

        default:
            j = 'n';
        }
    }
    printf("程序结束!\nPress any key to shut off the window!\n");
    getchar();
    getchar();
}
```

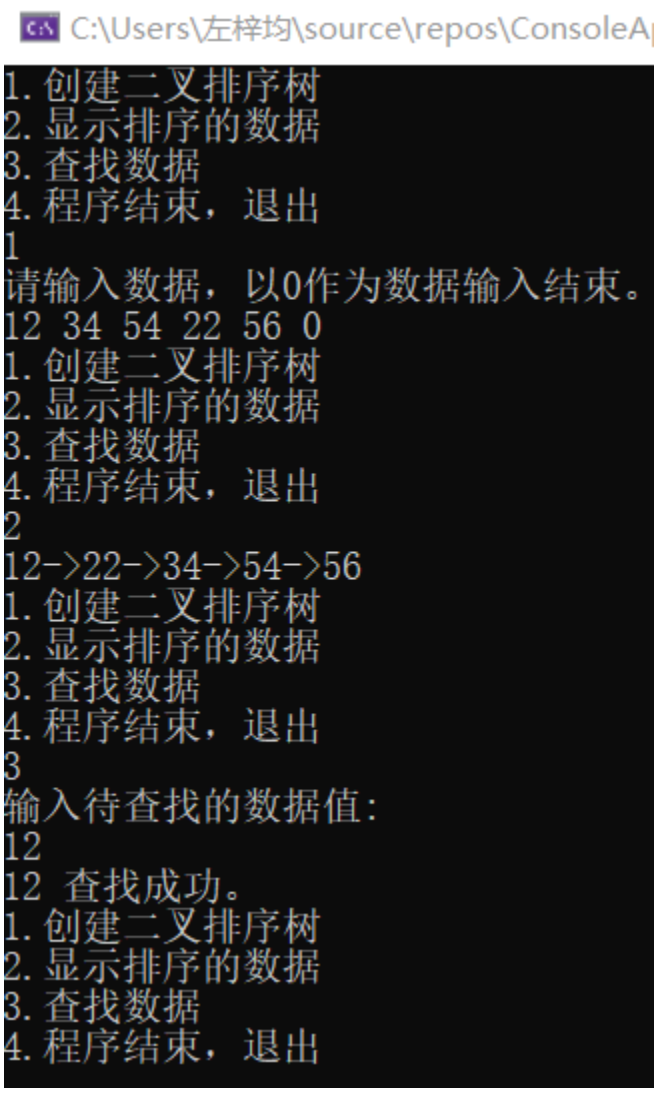
排序

```
#include<stdio.h>
#include<stdlib.h>
#include <conio.h>
#define n 30
struct student
{
    char name[8];
    int score;
} R[n];
int main()
{
    int num, i, j, max, temp;
    printf("\n请输入学生成绩: \n");
    for (i = 0; i < n; i++)
    {
        printf("姓名:");
        scanf_s("%s", R[i].name);
        printf("成绩:");
        scanf_s("%d", &(R[i].score));
    }
    num = 1;
    for (i = 0; i < n; i++)
    {
        max = i;
        for (j = i + 1; j < n; j++)
            if (R[j].score > R[max].score)
                max = j;
        if (max != i)
        {
            temp = R[max].score;
            R[max] = R[i];
            R[i].score = temp;
        }
        if ((i > 0) && (R[i].score < R[i - 1].score))
            num = num + 1;
        printf("%4d%s%4d", num, R[i].name, R[i].score);
    }
    getchar();

}
```

三、实验结果及分析

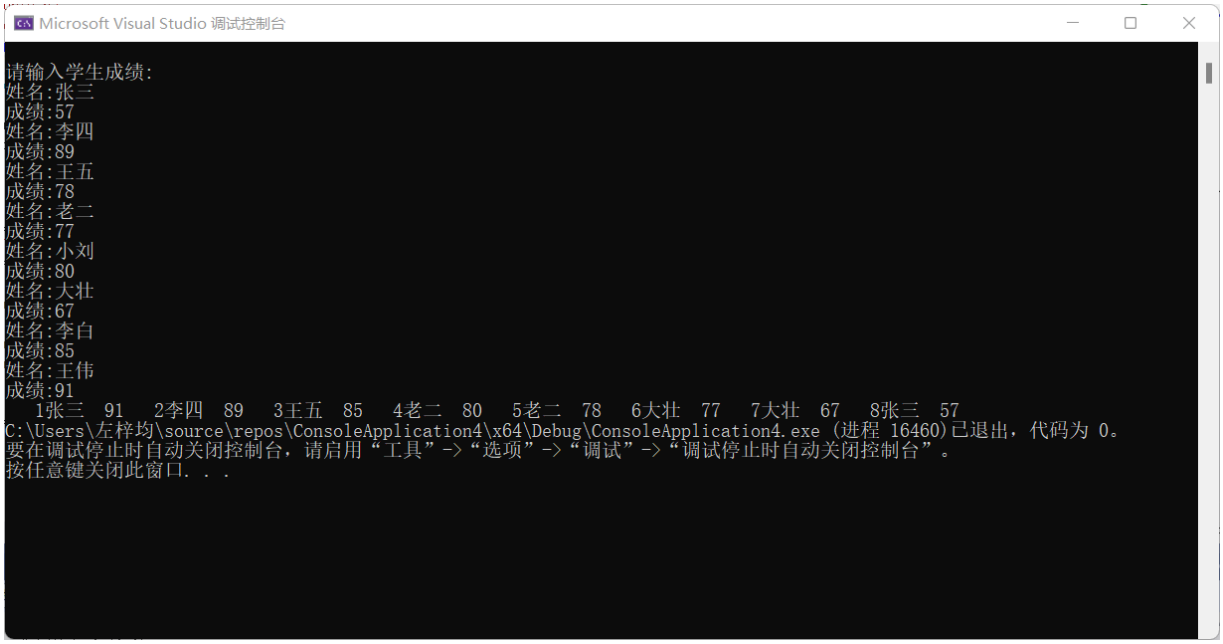
查找

实验结果：



排序

实验结果：



实验分析：

插入排序、选择排序、冒泡排序等实现简单，并且在待排序的元素总量较小的时候与快速排序、归并排序等速度相差不大，但是当待排序元素总量较大时，快速排序和二路归并排序的优势就非常明显了。所以在元素总量较小时，各种排序方法的差距不会太大，但当元素总量大时，为了提高效率，还是应该选择实现过程较复杂的但效率高的算法。

查找算法：在顺序查找、折半查找、二叉查找树查找中算法中，顺序查找是效率最低的；对于折半查找、二叉查找树查找来说，这两种查找算法的时间复杂度在一般情况下是相同的，但二叉查找树是链式结构，插入和删除都只需要修改指针，这是非常方便的。