广东开放大学计算机科学与技术专业

《数据结构》

课程实验方案

人工智能学院

2019/9/6

目录

一、课程基本信息

学院：人工智能学院

专业名称：计算机科学与技术

课程名称：数据结构

课程编码（课程 ID 号）：10090

课程性质（专业必修课 / 专业选修课）：专业必修课

适用专业：计算机科学与技术

先修课程：离散数学、C 语言程序设计

课程实验（实训）负责教师：李美满

课程总学分：4

课程总学时：72

课程实验（训）总学时：18

二、课程简介

本课程是面向计算机科学与技术专业开设的专核心课程。是学习其他软件开发与设计等方面课程的基础。主要内容包括：线性表、栈和队列、串、数组和广义表、树、图、查找算法和排序算法。数据结构研究数据的组织方式，内容丰富、学习量大，隐含在各部分内容中的方法和技术多，旨在让学生掌握计算机软件系统所必需的数据结构的算法。要求学生掌握贯穿全课程的动态链表存储结构，掌握算法设

计的动态性和抽象性。要求学生学会分析研究计算机加工的数据对象的特征，以便在实际应用中选择适当的数据结构、存储结构和相应算法，初步掌握算法的时间与空间性能分析技巧，并培养复杂程序设计的技能。

三、考核说明或要求

本课程共有 6 个实训，实验一：单链表的基本操作 20 分，实验二：栈和队列的应用 15 分，实验三：二叉树的基本算法 20 分，实验四：图的建立和搜索 15 分，实验五：顺序查找和折半查 15 分，实验六：综合性实验 15 分，满分 100 分。

四、实验名称：

实验一：单链表的基本操作

预计课时：4 学时

实训时长： 180(分钟)

实验简介：以单链表形式创建一个学生表或图书表，并能实现相关的查找、插入和删除等算法。

实验目标：

1、掌握线性表的定义；

2、掌握线性表的基本操作，如建立、查找、插入和删除等。

实验内容：

定义一个包含学生信息（学号，姓名，成绩）的链表和顺序表，使其具有如下功能：

(1) 根据指定学生个数，逐个输入学生信息；

(2) 逐个显示学生表中所有学生的相关信息；

(3) 根据姓名进行查找，返回此学生的学号和成绩；

(4) 根据指定的位置可返回相应的学生信息(学号，姓名，成绩)；

(5) 给定一个学生信息，插入到表中指定的位置；

(6) 删除指定位置的学生记录；

(7) 统计表中学生个数。

实验所需基础：

实验环境：Visual C++

实验是否需要联网：否

实训步骤：

```
//1、教材单链表的基本操作的实现
#include<iostream>
#include<string>
#include<iomanip>
#include<fstream>
using namespace std;
#define OK 1
#define ERROR 0
#define OVERFLOW -2
typedef int Status; //Status 是函数返回值类型，其值是函数结果状态代码。
typedef int ElemType; //ElemType 为可定义的数据类型，此设为 int 类型

struct Book {
    string id;//ISBN
    string name;//书名
```

```
    double price;//定价
};
typedef struct LNode {
    Book data; //结点的数据域
    struct LNode *next; //结点的指针域
} LNode, *LinkList; //LinkList 为指向结构体 LNode 的指针类型

string head_1, head_2, head_3;
int length;

Status InitList_L(LinkList &L) { //算法 2.6  单链表的初始化
    //构造一个空的单链表 L
    L = new LNode; //生成新结点作为头结点，用头指针 L 指向头结点
    L->next = NULL; //头结点的指针域置空
    return OK;
}

Status GetElem_L(LinkList L, int i, Book &e) { //算法 2.7  单链表的取值
    //在带头结点的单链表 L 中查找第 i 个元素
    //用 e 返回 L 中第 i 个数据元素的值
    int j;
    LinkList p;
    p = L->next;
    j = 1; //初始化，p 指向第一个结点，j 为计数器
    while (j < i && p) { //顺链域向后扫描，直到 p 指向第 i 个元素或 p 为空
        p = p->next; //p 指向下一个结点
        ++j; //计数器 j 相应加 1
    }
    if (!p || j > i)
        return ERROR; //i 值不合法 i > n 或 i<=0
    e = p->data; //取第 i 个结点的数据域
    return OK;
} //GetElem_L

LNode *LocateElem_L(LinkList L, int e) { //算法 2.8  按值查找
    //在带头结点的单链表 L 中查找值为 e 的元素
    LinkList p;
    p = L->next;
    while (p && p->data.price != e)//顺链域向后扫描，直到 p 为空或 p 所指结点的
数据域等于 e
        p = p->next; //p 指向下一个结点
    return p; //查找成功返回值为 e 的结点地址 p，查找失败 p 为 NULL
} //LocateElem_L

Status ListInsert_L(LinkList &L, int i, Book &e) { //算法 2.9  单链表的插入
    //在带头结点的单链表 L 中第 i 个位置插入值为 e 的新结点
```

```
    int j;
    LinkList p, s;
    p = L;
    j = 0;
    while (p && j < i - 1) {
        p = p->next;
        ++j;
    }//查找第 i 个结点，p 指向该结点
    if (!p || j > i - 1)
        return ERROR; //i > n+1 或者 i < 1
    s = new LNode; //生成新结点*s
    s->data = e; //将结点*s 的数据域置为 e
    s->next = p->next; //将结点*s 的指针域指向结点 ai
    p->next = s; //将结点*p 的指针域指向结点*s
    ++length;
    return OK;
} //ListInsert_L

Status ListDelete_L(LinkList &L, int i) { //算法 2.9  单链表的删除
    //在带头结点的单链表 L 中，删除第 i 个位置
    LinkList p, q;
    int j;
    p = L;
    j = 0;
    while ((p->next) && (j < i - 1)) //查找第 i?1 个结点，p 指向该结点
    {
        p = p->next;
        ++j;
    }
    if (!(p->next) || (j > i - 1))
        return ERROR; //当 i>n 或 i<1 时，删除位置不合理
    q = p->next; //临时保存被删结点的地址以备释放
    p->next = q->next; //改变删除结点前驱结点的指针域
    delete q; //释放删除结点的空间
    --length;
    return OK;
} //ListDelete_L

void CreateList_H(LinkList &L, int n) { //算法 2.11  前插法创建单链表
    //逆位序输入 n 个元素的值，建立到头结点的单链表 L
    LinkList p;
    L = new LNode;
    L->next = NULL; //先建立一个带头结点的空链表
    length = 0;
    fstream file;
    file.open("book.txt");
```

```
    if (!file) {
        cout << "未找到相关文件，无法打开！" << endl;
        exit(ERROR);
    }
    file >> head_1 >> head_2 >> head_3;
    while (!file.eof()) {
        p = new LNode; //生成新结点*p
        file >> p->data.id >> p->data.name >> p->data.price; //输入元素值赋
给新结点*p 的数据域
        p->next = L->next;
        L->next = p; //将新结点*p 插入到头结点之后
        length++;//同时对链表长度进行统计
    }
    file.close();
} //CreateList_F

void CreateList_R(LinkList &L, int n) { //算法 2.12  后插法创建单链表
    //正位序输入 n 个元素的值，建立带表头结点的单链表 L
    LinkList p, r;
    L = new LNode;
    L->next = NULL; //先建立一个带头结点的空链表
    r = L; //尾指针 r 指向头结点
    length = 0;
    fstream file; //打开文件进行读写操作
    file.open("book.txt");
    if (!file) {
        cout << "未找到相关文件，无法打开！" << endl;
        exit(ERROR);
    }
    file >> head_1 >> head_2 >> head_3;
    while (!file.eof()) { //将文件中的信息运用后插法插入到链表中
        p = new LNode;//生成新结点
        file >> p->data.id >> p->data.name >> p->data.price;//输入元素值赋
给新结点*p 的数据域
        p->next = NULL;
        r->next = p;//将新结点*p 插入尾结点*r 之后
        r = p;//r 指向新的尾结点*p
        length++; //同时对链表长度进行统计
    }
    file.close();
} //CreateList_L

int main() {
    int a, n, choose;
    double price;
    Book e;
```

```
    LinkList L, p;
    cout << "1. 建立\n";
    cout << "2. 输入\n";
    cout << "3. 取值\n";
    cout << "4. 查找\n";
    cout << "5. 插入\n";
    cout << "6. 删除\n";
    cout << "7. 输出\n";
    cout << "0. 退出\n\n";

    choose = -1;
    while (choose != 0) {
        cout << "请选择:";
        cin >> choose;
        switch (choose) {
        case 1: //建立一个单链表
            if (InitList_L(L))
                cout << "成功建立链表!\n\n";
            break;
        case 2: //使用后插法创建单链表
            CreateList_R(L, length);
            cout << "输入 book.txt 信息完毕\n\n";
            break;
        case 3: //单链表的按序号取值
            cout << "请输入一个位置用来取值:";
            cin >> a;
            if (GetElem_L(L, a, e)) {
                cout << "查找成功\n";
                cout << "第" << a << "本图书的信息是：\n";
                cout << left << setw(15) << e.id << "\t" << left << setw(50)
                        << e.name << "\t" << left << setw(5) << e.price <<
endl
                        << endl;
            } else
                cout << "查找失败\n\n";
            break;
        case 4: //单链表的按值查找
            cout << "请输入所要查找价格:";
            cin >> price;
            if (LocateElem_L(L, price) != NULL) {
                cout << "查找成功\n";
                cout << "该价格对应的书名为：" << LocateElem_L(L,
price)->data.name
                        << endl << endl;
            } else
                cout << "查找失败! 定价" << price << " 没有找到\n\n";
```

```
                break;
            case 5: //单链表的插入
                cout << "请输入插入的位置和书的信息，包括：编号 书名 价格（用空
格隔开）:";
                cin >> a;
                cin >> e.id >> e.name >> e.price;
                if (ListInsert_L(L, a, e))
                    cout << "插入成功.\n\n";
                else
                    cout << "插入失败!\n\n";
                break;
            case 6: //单链表的删除
                cout << "请输入所要删除的书籍的位置:";
                cin >> a;
                if (ListDelete_L(L, a))
                    cout << "删除成功!\n\n";
                else
                    cout << "删除失败!\n\n";
                break;
            case 7: //单链表的输出
                cout << "当前图书系统信息（链表）读出:\n";
                p = L->next;
                while (p) {
                    cout << left << setw(15) << p->data.id << "\t" << left <<
setw(50) << p->data.name << "\t" << left << setw(5)
                                << p->data.price << endl;
                    p = p->next;
                }
                cout << endl;
                break;
            }
        }
        return 0;
}


//2、实验链表的参考源程序：
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
#include<string.h>
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#include<iostream>
using namespace std;
```

```
typedef int Status; // 定义函数返回值类型

typedef struct
{
    char no[10]; // 学号
    char name[20]; // 姓名
    double score; // 成绩
}student;

typedef student ElemType;

typedef struct LNode
{
    ElemType data; // 数据域
    struct LNode *next; //指针域
}LNode,*LinkList;

Status InitList(LinkList &L) // 构造空链表 L
{
    L=(struct LNode*)malloc(sizeof(struct LNode));
    L->next=NULL;
    return OK;
}

Status GetElem(LinkList L,int i,ElemType &e) // 访问链表，找到 i 位置的数据域，
返回给 e
{
    LinkList p;
    p=L->next;
    int j=1;
    while(p&&j<i)
    {
        p=p->next;
        ++j;
    }
    if(!p||j>i)   return ERROR;
    e=p->data;
    return OK;
}

Status Search(LNode L,char str[],LinkList &p) // 根据名字查找
{
    p=L.next;
    while(p)
    {
```

```
            if(strcmp(p->data.name,str)==0)
                return OK;
            p=p->next;
        }
        return ERROR;
}

Status ListInsert(LinkList L,int i,ElemType e) // 在 i 个位置插入某个学生的信息
{
        LinkList p,s;
        p=L;
        int j=0;
        while(p&&j<i-1)
        {
            p=p->next;
            ++j;
        }
        if(!p||j>i-1)       return ERROR;
        s=(struct LNode*)malloc(sizeof(LNode));
        s->data=e;
        s->next=p->next;
        p->next=s;
        return OK;
}

Status ListDelete(LinkList p,int i) // 删除 i 位置的学生信息
{
        int j=0;
        while((p->next)&&(j<i-1))
        {
            p=p->next;
            ++j;
        }
        if(!(p->next)||(j>i-1))     return ERROR;
        LinkList q;
        q=p->next;
        p->next=q->next;
        delete q;
        return OK;
}

void Input(ElemType *e)
{
        cout<<"姓名：";
            cin>>e->name;
        cout<<"学号：";
```

```
        cin>>e->no;
    cout<<"成绩：";
        cin>>e->score;
    cout<<"完成输入\n\n";
}

void Output(ElemType *e)
{
    printf("姓名:%-20s\n 学号:%-10s\n 成绩:%-10.2lf\n\n",
e->name,e->no,e->score);
}

int main()
{
    LNode L;
    LinkList p;
    ElemType a,b,c,d;
    cout<<"------------10.2.34 版  -----------\n";
    cout<<"1. 构造顺序表\n";
    cout<<"2. 录入指定人数的学生信息\n";
    cout<<"3. 显示学生表中的所有信息\n";
    cout<<"4. 根据姓名查找该学生，并返回学号和成绩\n";
    cout<<"5. 根据某指定位置显示该学生信息\n";
    cout<<"6. 在指定位置插入学生信息\n";
    cout<<"7. 在指定位置删除学生信息\n";
    cout<<"8. 统计学生个数\n";
    cout<<"0. 退出\n";
    cout<<"------------------------\n";
    int n,choose=-1;
    while(choose!=0)
    {
        puts("请输入你要选择的功能前的序号:");
        cin>>choose ;
        if(choose==0)
            break;
        else if (choose==1)
        {
            if(InitList(p))
                cout<<"建立顺序表成功\n";
            else
                cout<<"建立顺序表失败\n";

        }

        else if (choose==2)
        {
```

```
        cout<<"将要输入学生的人数：";
        cin>>n;
        for(int i=1;i<=n;i++)
        {
                printf("第%d 个学生:\n",i);
            Input(&a);
            ListInsert(&L,i,a);
        }

}


else if (choose==3)
{
        for(int i=1;i<=n;i++)
        {
            GetElem(&L,i,b);
            Output(&b);
        }

}
else if (choose==4)
{
        char s[20];
        cout<<"请输入要查找的学生姓名:";
        cin>>s;
        if(Search(L,s,p))
            Output(&(p->data));
        else
            cout<<"对不起，查无此人";
        puts("");

}
else if (choose==5)
{
        cout<<"请输入要查询的位置:";
        int id1;
        cin>>id1;
        GetElem(&L,id1,c);
        Output(&c);

}
else if (choose==6)
{
        cout<<"请输入要插入的位置:";
        int id2;
```

```
            cin>>id2;
            cout<<"请输入学生信息:\n";
            Input(&d);
            if(ListInsert(&L,id2,d))
            {
                n++;
                cout<<"插入成功"; ;
                puts("");
            }
            else
            {
                cout<<"插入失败";
                puts("");
            }

        }
        else if (choose==7)
        {
            cout<<"请输入要删除的位置:";
            int id3;
            cin>>id3;
            if(ListDelete(&L,id3))
            {
                n--;
                cout<<"删除成功";
                puts("");
            }
            else
            {
                cout<<"删除失败";
                puts("");
            }

        }
        else if (choose==8)
                {
                    cout<<"已录入的学生个数为:"<<n<<endl;
                    break;
            }

    }
    cout<<"\n\n 谢谢您的使用，请按任意键退出\n\n\n";
    system("pause");
    return 0;
}
```

```
//顺序表的参考源程序：
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
#include<string.h>
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXSIZE 100

typedef int Status; // 定义函数返回值类型

typedef struct
{
    char no[10]; // 学号
    char name[20]; // 姓名
    int score; // 成绩
}student;

typedef student ElemType;

typedef struct
{
    ElemType *elem; // 存储空间的基地址
    int length; // 当前长度
}SqList;

Status InitList(SqList *L) // 构造空的顺序表 L
{
    L->elem=(ElemType *)malloc(sizeof(ElemType)*MAXSIZE);
    if(!L->elem)  exit(OVERFLOW);
    L->length=0;
    return OK;
}

ElemType GetElem(SqList &L,int i) // 访问顺序表，找到 i 位置，返回给 e
{
    return L.elem[i];
}

int Search(SqList &L,char str[]) // 根据名字查找，返回该同学在顺序表中的编号
{
    for(int i=1;i<=L.length;i++)
    {
        if(strcmp(L.elem[i].name,str)==0)
```

```
                return i;
        }
        return 0;
}

Status ListInsert(SqList &L,int i,ElemType e) // 在 i 位置插入某个学生的信息
{
        if((i<1)||(i>L.length+1)) return ERROR;
        if(L.length==MAXSIZE)     return ERROR;
        for(int j=L.length;j>=i;j--)
        {
                L.elem[j+1]=L.elem[j];
        }
        L.elem[i]=e;
        ++L.length;
        return OK;
}

Status ListDelete(SqList &L,int i) // 在顺序表中删除 i 位置的学生信息
{
        if((i<1)||(i>L.length))     return ERROR;
        for(int j=i;j<=L.length;j++)
        {
                L.elem[j]=L.elem[j+1];
        }
        --L.length;
        return OK;
}

void Input(ElemType *e)
{
        printf("姓名:");    scanf("%s",e->name);
        printf("学号:");    scanf("%s",e->no);
        printf("成绩:");    scanf("%d",&e->score);
        printf("输入完成\n\n");
}

void Output(ElemType *e)
{
        printf("姓名:%-20s\n 学号:%-10s\n 成
绩:%-10.2d\n\n",e->name,e->no,e->score);
}

int main()
{
        SqList L;
```

```
ElemType a,b,c,d;
printf("------------10.2.34 版-----------\n");
puts("1. 构造顺序表");
puts("2. 录入指定人数的学生信息");
puts("3. 显示学生表中的所有信息");
puts("4. 根据姓名查找该学生，并返回学号和成绩");
puts("5. 根据某指定位置显示该学生信息");
puts("6. 在指定位置插入学生信息");
puts("7. 在指定位置删除学生信息");
puts("8. 统计学生个数");
puts("0. 退出");
printf("------------------------\n");
int x,choose;
while(1)
{
    puts("请输入你要选择的功能前的序号:");
    scanf("%d",&choose);
    if(choose==0)    break;
    switch(choose)
    {
        case 1:
            if(InitList(&L))
                printf("成功建立顺序表\n\n");
            else
                printf("顺序表建立失败\n\n");
            break;
        case 2:
            printf("请输入要录入学生的人数（小于 100）:");
            scanf("%d",&x);
            for(int i=1;i<=x;i++)
            {
                printf("第%d 个学生:\n",i);
                Input(&L.elem[i]);
            }
            L.length=x;
            puts("");
            break;
        case 3:
            for(int i=1;i<=x;i++)
            {
                a=GetElem(L,i);
                Output(&a);
            }
            break;
        case 4:
            char s[20];
```

```
        printf("请输入要查找的学生姓名:");
        scanf("%s",s);
        if(Search(L,s))
            Output(&L.elem[Search(L,s)]);
        else
            puts("对不起，查无此人");
        puts("");
        break;
    case 5:
        printf("请输入要查询的位置:");
        int id1;
        scanf("%d",&id1);
        b=GetElem(L,id1);
        Output(&b);
        break;
    case 6:
        printf ("请输入要插入的位置:");
        int id2;
        scanf("%d",&id2);
        printf("请输入学生信息:\n");
        Input(&c);
        if(ListInsert(L,id2,c))
        {
            x++;
            puts("插入成功");
            puts("");
        }
        else
        {
            puts("插入失败");
            puts("");
        }
        break;
    case 7:
        printf("请输入要删除的位置:");
        int id3;
        scanf("%d",&id3);
        if(ListDelete(L,id3))
        {
            x--;
            puts("删除成功");
            puts("");
        }
        else
        {
            puts("删除失败");
```

```
                    puts("");
                }
                break;
            case 8:
                printf("已录入的学生个数为:%d\n\n",L.length);
                break;
        }
    }
    printf("\n\n 谢谢您的使用，请按任意键退出\n\n\n");
    system("pause");
    return 0;
}
```

实验二：栈和队列的应用

预计课时：4 学时

实训时长： 180(分钟)

实验简介：借助栈或队列来解决某些实际应用问题

实验目标：

1、掌握栈的定义及实现；

2、掌握利用栈求解 Hanoi 塔问题；

3、链栈实现数制的转换。

实验内容：

1、Hanoi 塔问题

规则:

(1) 每次只能移动一个圆盘

(2) 圆盘可以插在 A,B 和 C 中的任一塔座上

(3) 任何时刻不可将较大圆盘压在较小圆盘之上

n = 1，则直接从 A 移到 C。否则

(1)用 C 柱做过渡，将 A 的(n-1)个移到 B

(2)将 A 最后一个直接移到 C

(3)用 A 做过渡，将 B 的 (n-1) 个移到 C

2、数制的转换

(1) 初始化一个空栈 S。

(2)当十进制数 N 非零时，循环执行以下操作：

把 N 与 8 求余得到的八进制数压入栈 S；

N 更新为 N 与 8 的商。

(3)当栈 S 非空时，循环执行以下操作：

弹出栈顶元素 e；

输出 e。

实验所需基础：

实验环境：Visual C++

实验是否需要联网：否

实训步骤：

```
/***顺序栈的实现***/

#include<iostream>
#include<fstream>
using namespace std;

//顺序栈定义
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXSIZE   100//顺序栈存储空间的初始分配量
typedef int Status;
typedef char SElemType;

typedef struct {
 SElemType *base;//栈底指针
 SElemType *top;//栈顶指针
 int stacksize;//栈可用的最大容量
} SqStack;

//算法 3.1   顺序栈的初始化
Status InitStack(SqStack &S) {
 //构造一个空栈 S
 S.base = new SElemType[MAXSIZE];//为顺序栈动态分配一个最大容量为
MAXSIZE 的数组空间
 if (!S.base)
       exit(OVERFLOW); //存储分配失败
 S.top = S.base; //top 初始为 base，空栈
 S.stacksize = MAXSIZE; //stacksize 置为栈的最大容量 MAXSIZE
 return OK;
}
```

```
//算法 3.2　顺序栈的入栈
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)
       return ERROR; //栈满
 *(S.top++) = e; //元素 e 压入栈顶，栈顶指针加 1
 return OK;
}
//算法 3.3　顺序栈的出栈
Status Pop(SqStack &S, SElemType &e) {
 //删除 S 的栈顶元素，用 e 返回其值
 if (S.base == S.top)
       return ERROR;//栈空
 e = *(--S.top); //栈顶指针减 1，将栈顶元素赋给 e
 return OK;
}
//算法 3.4　取顺序栈的栈顶元素
char GetTop(SqStack S) {//返回 S 的栈顶元素，不修改栈顶指针
 if (S.top != S.base) //栈非空
       return *(S.top - 1); //返回栈顶元素的值，栈顶指针不变
}

int main() {
 SqStack s;
 int choose, flag = 0;
 SElemType j, e, t;
 cout << "1.初始化\n";
 cout << "2.入栈\n";
 cout << "3.读栈顶元素\n";
 cout << "4.出栈\n";
 cout << "0.退出\n\n";

 choose = -1;
 while (choose != 0) {
       cout << "请选择:";
       cin >> choose;
       switch (choose) {
       case 1://算法 3.1　顺序栈的初始化
             if (InitStack(s)) {
                   flag = 1;
                   cout << "成功对栈进行初始化\n\n";
             } else
                   cout << "初始化栈失败\n\n";
             break;
       case 2: {//算法 3.2　顺序栈的入栈
             fstream file;
```

```
            file.open("SqStack.txt");//
            if (!file) {
                cout << "错误！未找到文件！\n\n" << endl;
                exit(ERROR);
            }
            if (flag) {
                flag = 1;
                cout << "进栈元素依次为：\n";
                while (!file.eof()) {
                    file >> j;
                    if (file.fail())
                        break;
                    else {
                        Push(s, j);
                        cout << j << "    ";
                    }
                }
                cout << endl << endl;
            } else
                cout << "栈未建立，请重新选择\n\n";
            file.close();
        }
            break;
        case 3://算法 3.3   顺序栈的出栈
            if(flag != -1 && flag != 0)
                cout << "栈顶元素为：\n" << GetTop(s) << endl << endl;
            else
                cout << "栈中无元素，请重新选择\n" << endl;
            break;
        case 4://算法 3.4   取顺序栈的栈顶元素
            cout << "依次弹出的栈顶元素为:\n";
            while (Pop(s, t)){
                flag = -1;
                cout << t << "    ";
            }
            cout << endl << endl;
            break;
        }
    }
    return 0;
}

/***Hanoi 塔问题***/

#include"3.10.h"
```

```
int main() {
 int n;
 char a, b, c;
 a = '1';
 b = '2';
 c = '3';
 cout << "请输入初始第一个柱子上的圆盘个数：" << endl;
 cin >> n;
 cout << "将第一个柱子上的圆盘全部移动到第三个柱子上的过程为：" << endl;
 Hanoi(n, a, b, c);
 return 0;
}

//算法 3.10　Hanoi 塔问题的递归算法
#include<iostream>
using namespace std;

int m = 0;//（m 是初值为 0 的全局变量，对搬动计数）
void move(char A, int n, char C) //  搬动操作
{
 cout << "第" << ++m << "步," << "将编号为" << n << "的圆盘从第" << A
<< "个柱子上移到第" << C
            << "个柱子上" << endl;
}

void Hanoi(int n, char A, char B, char C) {//将塔座 A 上的 n 个圆盘按规则搬到 C
上，B 做辅助塔
 if (n == 1)
       move(A, 1, C); //将编号为 1 的圆盘从 A 移到 C
 else {
       Hanoi(n - 1, A, C, B); //将 A 上编号为 1 至 n-1 的圆盘移到 B，C 做辅助塔
       move(A, n, C);//将编号为 n 的圆盘从 A 移到 C
       Hanoi(n - 1, B, A, C); //将 B 上编号为 1 至 n-1 的圆盘移到 C，A 做辅助塔
 }
}


//算法 3.20　数制的转换(链栈实现)
void conversion(int N) {//对于任意一个非负十进制数，打印输出与其等值的八进
制数
 int e;
 LinkStack S;
 InitStack(S); //初始化空栈 S
 while (N) //当 N 非零时,循环
 {
       Push(S, N % 8); //把 N 与 8 求余得到的八进制数压入栈 S
```

```
        N = N / 8; //N 更新为 N 与 8 的商
    }
    while (!StackEmpty(S)) //当栈 S 非空时，循环
    {
        Pop(S, e); //弹出栈顶元素 e
        cout << e; //输出 e
    }
}
int main() {//对于输入的任意一个非负十进制数，打印输出与其等值的八进制数
    int n, e;
    cout << "输入一个非负十进制数:" << endl;
    cin >> n;

    conversion(n);
    return 0;
}
```

实验三：二叉树的基本算法

预计课时：4 学时

实训时长： 180(分钟)

实验简介：利用二叉链表方法建立二叉树，实现二叉树的前、中、后序三种遍历算法，并运用遍历算法实现二叉树的其他操作，如计算二叉树结点个数、叶子结点个数、二叉树的高度等。

实验目标：

1．掌握二叉树的定义；

2．掌握二叉树的基本操作，如二叉树的建立、遍历、结点个数统计、树的深度计算等

实验内容：

用递归的方法实现以下算法：

1、以二叉链表表示二叉树，建立一棵二叉树；

2、输出二叉树的中序遍历结果；

3、计算二叉树的深度；

4、统计二叉树的结点个数。

实验所需基础：

实验环境：Visual C++

实验是否需要联网：否

实训步骤：

```
//算法 5.3 先序遍历的的顺序建立二叉链表
#include<iostream>
using namespace std;

//二叉树的二叉链表存储表示
typedef struct BiNode
{
char data;                          //结点数据域
struct BiNode *lchild,*rchild;      //左右孩子指针
}BiTNode,*BiTree;

void CreateBiTree(BiTree &T)
{
//按先序次序输入二叉树中结点的值（一个字符），创建二叉链表表示的二叉树 T
char ch;
cin >> ch;
if(ch=='#')   T=NULL;               //递归结束，建空树
else{
 T=new BiTNode;
 T->data=ch;                        //生成根结点
 CreateBiTree(T->lchild);     //递归创建左子树
 CreateBiTree(T->rchild);     //递归创建右子树
}                                   //else
}                                       //CreateBiTree

//用算法 5.1 中序遍历的递归算法
void InOrderTraverse(BiTree T)
{
//中序遍历二叉树 T 的递归算法
if(T){
 InOrderTraverse(T->lchild);
 cout << T->data;
 InOrderTraverse(T->rchild);
}
}
void main()
{
BiTree tree;
cout<<"请输入建立二叉链表的序列：\n";
CreateBiTree(tree);
cout<<"所建立的二叉链表中序序列：\n";
InOrderTraverse(tree);
cout<<endl;
}
//算法 5.5 计算二叉树的深度
#include<iostream>
```

```
using namespace std;

//二叉树的二叉链表存储表示
typedef struct BiNode
{
char data;                          //结点数据域
struct BiNode *lchild,*rchild;      //左右孩子指针
}BiTNode,*BiTree;

//用算法 5.3 建立二叉链表
void CreateBiTree(BiTree &T)
{
//按先序次序输入二叉树中结点的值（一个字符），创建二叉链表表示的二叉树 T
char ch;
cin >> ch;
if(ch=='#')   T=NULL;               //递归结束，建空树
else{
 T=new BiTNode;
 T->data=ch;                        //生成根结点
 CreateBiTree(T->lchild);     //递归创建左子树
 CreateBiTree(T->rchild);     //递归创建右子树
}                                   //else
}                                       //CreateBiTree

int Depth(BiTree T)
{
int m,n;
if(T == NULL ) return 0;            //如果是空树，深度为 0，递归结束
else
{
 m=Depth(T->lchild);                //递归计算左子树的深度记为 m
 n=Depth(T->rchild);                //递归计算右子树的深度记为 n
 if(m>n) return(m+1);          //二叉树的深度为 m 与 n 的较大者加 1
 else return (n+1);
}
}

void main()
{
BiTree tree;
cout<<"请输入建立二叉链表的序列：\n";
CreateBiTree(tree);
cout<<"数的深度为："<<Depth(tree)<<endl;

}
//算法 5.6  统计二叉树中结点的个数
```

```
#include<iostream>
using namespace std;

//二叉树的二叉链表存储表示
typedef struct BiNode
{
char data;                          //结点数据域
struct BiNode *lchild,*rchild;      //左右孩子指针
}BiTNode,*BiTree;

//用算法 5.3 建立二叉链表
void CreateBiTree(BiTree &T)
{
//按先序次序输入二叉树中结点的值（一个字符），创建二叉链表表示的二叉树 T
char ch;
cin >> ch;
if(ch=='#')   T=NULL;               //递归结束，建空树
else{
 T=new BiTNode;
 T->data=ch;                        //生成根结点
 CreateBiTree(T->lchild);     //递归创建左子树
 CreateBiTree(T->rchild);     //递归创建右子树
}                                   //else
}                                       //CreateBiTree

int NodeCount(BiTree T)
{
      if(T==NULL) return 0;        // 如果是空树，则结点个数为 0，递归结束
      else return NodeCount(T->lchild)+ NodeCount(T->rchild) +1;
      //否则结点个数为左子树的结点个数+右子树的结点个数+1
}

void main()
{
BiTree tree;
cout<<"请输入建立二叉链表的序列：\n";
CreateBiTree(tree);
cout<<"结点个数为："<<NodeCount(tree)<<endl;

}
```

实验四：图的建立和搜索

预计课时：2 学时

实训时长： 90(分钟)

实验简介：使用邻接矩阵或邻接表表示法存储一个图，实现图的深度优先搜索和广度优先搜索的算法。

实验目标：

1、掌握图的邻接矩阵的存储定义；

2、掌握图的深度优先搜索和广度优先搜索算法的实现。

实验内容：

1、图的邻接矩阵的存储定义；

2、深度优先搜索算法；

3、广度优先搜索算法。

实验所需基础：

实验环境：Visual C++

实验是否需要联网：否

实训步骤：

```
//算法 6.3  深度优先搜索遍历连通图的递归算法

#include <iostream>
using namespace std;

#define MVNum 100                       //最大顶点数
typedef char VerTexType;                //假设顶点的数据类型为字符型
typedef int ArcType;                    //假设边的权值类型为整型

typedef struct{
    VerTexType vexs[MVNum];                     //顶点表
    ArcType arcs[MVNum][MVNum];                 //邻接矩阵
```

```
    int vexnum,arcnum;                              //图的当前点数和边数
}Graph;

bool visited[MVNum];                            //访问标志数组，其初值为"false"
int FirstAdjVex(Graph G , int v);               //返回 v 的第一个邻接点
int NextAdjVex(Graph G , int v , int w);        //返回 v 相对于 w 的下一个邻接点

int LocateVex(Graph G , VerTexType v){
    //确定点 v 在 G 中的位置
    for(int i = 0; i < G.vexnum; ++i)
        if(G.vexs[i] == v)
            return i;
        return -1;
}//LocateVex

void CreateUDN(Graph &G){
    //采用邻接矩阵表示法，创建无向网 G
    int i , j , k;
    cout <<"请输入总顶点数，总边数，以空格隔开:";
    cin >> G.vexnum >> G.arcnum;                //输入总顶点数，总边数
    cout << endl;

    cout << "输入点的名称，如 a：" << endl;

    for(i = 0; i < G.vexnum; ++i){
        cout << "请输入第" << (i+1) << "个点的名称:";
        cin >> G.vexs[i];                               //依次输入点的信息
    }
    cout << endl;

    for(i = 0; i < G.vexnum; ++i)                       //初始化邻接矩阵 ,边的权值
均置为极大值 MaxInt
        for(j = 0; j < G.vexnum; ++j)
            G.arcs[i][j] = 0;
    cout << "输入边依附的顶点，如：a b" << endl;
    for(k = 0; k < G.arcnum;++k){                       //构造邻接矩阵
        VerTexType v1 , v2;
        cout << "请输入第" << (k + 1) << "条边依附的顶点:";
        cin >> v1 >> v2;                            //输入一条边依附的顶点及权值
        i = LocateVex(G, v1);   j = LocateVex(G, v2);   //确定 v1 和 v2 在 G 中的位
置，即顶点数组的下标
        G.arcs[j][i] = G.arcs[i][j] = 1;                //置<v1, v2>的对称边<v2,
v1>的权值为 w
    }//for
}//CreateUDN
```

```
void DFS(Graph G, int v){                    //从第 v 个顶点出发递归地深度优先遍历图 G
    cout << G.vexs[v] << "    ";  visited[v] = true;      //访问第 v 个顶点，并置访问标志数组相应分量值为 true
    int w;
    for(w = FirstAdjVex(G, v);  w >= 0;  w = NextAdjVex(G, v, w))
        //依次检查 v 的所有邻接点 w ，FirstAdjVex(G, v)表示 v 的第一个邻接点
        //NextAdjVex(G, v, w)表示 v 相对于 w 的下一个邻接点，w≥0 表示存在邻接点
        if(!visited[w]) DFS(G, w);                  //对 v 的尚未访问的邻接顶点 w 递归调用 DFS
}//DFS

int FirstAdjVex(Graph G , int v){
    int i;
    for(i = 0 ; i < G.vexnum ; ++i){
        if(G.arcs[v][i] == 1 && visited[i] == false)
            return i;
    }
    return -1;
}//FirstAdjVex

int NextAdjVex(Graph G , int v , int w){
    int i;
    for(i = w ; i < G.vexnum ; ++i){
        if(G.arcs[v][i] == 1 && visited[i] == false)
            return i;
    }
    return -1;
}//NextAdjVex

int main(){
    cout << "**********算法 6.3  深度优先搜索遍历连通图的递归算法**********" << endl << endl;
    Graph G;
    CreateUDN(G);
    cout << endl;
    cout << "无向连通图 G 创建完成！" << endl << endl;

    cout << "请输入遍历连通图的起始点：";
    VerTexType c;
    cin >> c;

    int i;
    for(i = 0 ; i < G.vexnum ; ++i){
        if(c == G.vexs[i])
            break;
```

```
    }
    cout << endl;
    while(i >= G.vexnum){
        cout << "该点不存在，请重新输入！" << endl;
        cout << "请输入遍历连通图的起始点：";
        cin >> c;
        for(i = 0 ; i < G.vexnum ; ++i){
            if(c == G.vexs[i])
                break;
        }
    }
    cout << "深度优先搜索遍历连通图结果：" << endl;
    DFS(G , i);

    cout <<endl;
    return 0;
}//main
//算法 6.4   深度优先搜索遍历非连通图

#include <iostream>
using namespace std;

#define MVNum 100                       //最大顶点数
typedef char VerTexType;                //假设顶点的数据类型为字符型
typedef int ArcType;                    //假设边的权值类型为整型

//-------------图的邻接矩阵-----------------
typedef struct{
    VerTexType vexs[MVNum];             //顶点表
    ArcType arcs[MVNum][MVNum];         //邻接矩阵
    int vexnum,arcnum;                  //图的当前点数和边数
}Graph;

bool visited[MVNum];                    //访问标志数组，其初值为"false"
int FirstAdjVex(Graph G , int v);       //返回 v 的第一个邻接点
int NextAdjVex(Graph G , int v , int w);//返回 v 相对于 w 的下一个邻接点

int LocateVex(Graph G , VerTexType v){
    //确定点 v 在 G 中的位置
    for(int i = 0; i < G.vexnum; ++i)
        if(G.vexs[i] == v)
            return i;
        return -1;
}//LocateVex

void CreateUDN(Graph &G){
```

```
    //采用邻接矩阵表示法，创建无向网 G
    int i , j , k;
    cout <<"请输入总顶点数，总边数，以空格隔开:";
    cin >> G.vexnum >> G.arcnum;                    //输入总顶点数，总边数
    cout << endl;

    cout << "输入点的名称，如 a" << endl;
    for(i = 0; i < G.vexnum; ++i){
        cout << "请输入第" << (i+1) << "个点的名称:";
        cin >> G.vexs[i];                           //依次输入点的信息
    }
    cout << endl;
    for(i = 0; i < G.vexnum; ++i)                   //初始化邻接矩阵，
边的权值均置为极大值 MaxInt
        for(j = 0; j < G.vexnum; ++j)
            G.arcs[i][j] = 0;
    cout << "输入边依附的顶点，如 a b" << endl;
    for(k = 0; k < G.arcnum;++k){                   //构造邻接矩阵
        VerTexType v1 , v2;
        cout << "请输入第" << (k + 1) << "条边依附的顶点:";
        cin >> v1 >> v2;                            //输入一条边依附的顶点及权值
        i = LocateVex(G, v1);   j = LocateVex(G, v2);          //确定 v1 和 v2 在 G
中的位置，即顶点数组的下标
        G.arcs[j][i] = G.arcs[i][j] = 1;            //置<v1, v2>的对称边<v2, v1>
的权值为 w
    }//for
}//CreateUDN

void DFS(Graph G, int v){
    //从第 v 个顶点出发递归地深度优先遍历图 G
    cout << G.vexs[v] << "      ";   visited[v] = true;      //访问第 v 个顶点，并置
访问标志数组相应分量值为 true
    int w;
    for(w = FirstAdjVex(G, v);   w >= 0;   w = NextAdjVex(G, v, w))
        //依次检查 v 的所有邻接点 w ，FirstAdjVex(G, v)表示 v 的第一个邻接点
        //NextAdjVex(G, v, w)表示 v 相对于 w 的下一个邻接点，w≥0 表示存在邻接
点
        if(!visited[w]) DFS(G, w);                  //对 v 的尚未访问的邻接顶点 w 递归调
用 DFS
}//DFS

void DFSTraverse(Graph G){
    //对非连通图 G 做深度优先遍历
    int v;
    for(v = 0; v < G.vexnum; ++v)   visited[v] = false;    //访问标志数组初始化
    for(v = 0; v < G.vexnum; ++v)                   //循环调用算法 6.3
```

```
            if(!visited[v])    DFS(G, v);                    //对尚未访问的顶点调用 DFS
//算法 6.5    采用邻接矩阵表示图的深度优先搜索遍历

#include <iostream>
using namespace std;

#define MVNum 100                                   //最大顶点数
typedef char VerTexType;                           //假设顶点的数据类型为字符型
typedef int ArcType;                               //假设边的权值类型为整型

//------------图的邻接矩阵-----------------
typedef struct{
    VerTexType vexs[MVNum];                        //顶点表
    ArcType arcs[MVNum][MVNum];                    //邻接矩阵
    int vexnum,arcnum;                             //图的当前点数和边数
}Graph;

bool visited[MVNum];                               //访问标志数组，其初值为"false"
int FirstAdjVex(Graph G , int v);                  //返回 v 的第一个邻接点
int NextAdjVex(Graph G , int v , int w);           //返回 v 相对于 w 的下一个邻接点

int LocateVex(Graph G , VerTexType v){
    //确定点 v 在 G 中的位置
    for(int i = 0; i < G.vexnum; ++i)
        if(G.vexs[i] == v)
            return i;
        return -1;
}//LocateVex

void CreateUDN(Graph &G){
    //采用邻接矩阵表示法，创建无向网 G
    int i , j , k;
    cout <<"请输入总顶点数，总边数，以空格隔开:";
    cin >> G.vexnum >> G.arcnum;                   //输入总顶点数，总边数
    cout << endl;

    cout << "输入点的名称，如 a" <<endl;

    for(i = 0; i < G.vexnum; ++i){
        cout << "请输入第" << (i+1) << "个点的名称:";
        cin >> G.vexs[i];                                   //依次输入点的信息
    }
    cout << endl;

    for(i = 0; i < G.vexnum; ++i)                           //初始化邻接矩阵，边的
权值均置为极大值 MaxInt
```

```
        for(j = 0; j < G.vexnum; ++j)
            G.arcs[i][j] = 0;
    cout << "输入边依附的顶点，如 a b" << endl;
    for(k = 0; k < G.arcnum;++k){                          //构造邻接矩阵
        VerTexType v1 , v2;
        cout << "请输入第" << (k + 1) << "条边依附的顶点:";
        cin >> v1 >> v2;                                   //输入一条边依附的顶点及权值
        i = LocateVex(G, v1);   j = LocateVex(G, v2);      //确定 v1 和 v2 在 G 中的
位置，即顶点数组的下标
        G.arcs[j][i] = G.arcs[i][j] = 1;           //置<v1, v2>的对称边<v2, v1>的权
值为 w
    }//for
}//CreateUDN

void DFS(Graph G, int v){
    //图 G 为邻接矩阵类型
    int w;
    cout << G.vexs[v] << "      ";   visited[v] = true;          //访问第 v 个顶点，
并置访问标志数组相应分量值为 true
    for(w = 0; w < G.vexnum; w++)                  //依次检查邻接矩阵 v 所在的行
        if((G.arcs[v][w] != 0)&& (!visited[w]))   DFS(G, w); //G.arcs[v][w]!=0 表示
w 是 v 的邻接点，如果 w 未访问，则递归调用 DFS
}//DFS

int FirstAdjVex(Graph G , int v){
    //返回 v 的第一个邻接点
    int i;
    for(i = 0 ; i < G.vexnum ; ++i){
        if(G.arcs[v][i] == 1 && visited[i] == false)
            return i;
    }
    return -1;
}//FirstAdjVex

int NextAdjVex(Graph G , int v , int w){
    //返回 v 相对于 w 的下一个邻接点
    int i;
    for(i = w ; i < G.vexnum ; ++i){
        if(G.arcs[v][i] == 1 && visited[i] == false)
            return i;
    }
    return -1;
}//NextAdjVex

int main(){
    cout << "**********算法 6.5   采用邻接矩阵表示图的深度优先搜索遍历
```

```
**************" << endl << endl;
    Graph G;
    CreateUDN(G);
    cout << endl;
    cout << "无向图 G 创建完成！" << endl << endl;

    cout << "请输入遍历无向图 G 的起始点：";
    VerTexType c;
    cin >> c;

    int i;
    for(i = 0 ; i < G.vexnum ; ++i){
        if(c == G.vexs[i])
            break;
    }
    cout << endl;
    while(i >= G.vexnum){
        cout << "该点不存在，请重新输入！" << endl;
        cout << "请输入遍历连通图的起始点：";
        cin >> c;
        for(i = 0 ; i < G.vexnum ; ++i){
            if(c == G.vexs[i])
                break;
        }
    }
    cout << "深度优先搜索遍历无向图 G 结果：" << endl;
    DFS(G , i);

    cout <<endl;
    return 0;
}//main

//算法 6.6　采用邻接表表示图的深度优先搜索遍历

#include <iostream>
using namespace std;

#define MVNum 100                       //最大顶点数
typedef char VerTexType;                //假设顶点的数据类型为字符型

//-------------图的邻接表----------------------
typedef struct ArcNode{                 //边结点
    int adjvex;                         //该边所指向的顶点的位置
    struct ArcNode *nextarc;            //指向下一条边的指针
}ArcNode;
```

```
typedef struct VNode{
    VerTexType data;                        //顶点信息
    ArcNode *firstarc;                      //指向第一条依附该顶点的边的指针
}VNode, AdjList[MVNum];                     //AdjList 表示邻接表类型

typedef struct{
    AdjList vertices;                       //邻接表
    int vexnum, arcnum;                     //图的当前顶点数和边数
}ALGraph;

bool visited[MVNum];                        //访问标志数组，其初值为"false"

int LocateVex(ALGraph G , VerTexType v){
    //确定点 v 在 G 中的位置
    for(int i = 0; i < G.vexnum; ++i)
        if(G.vertices[i].data == v)
            return i;
        return -1;
}//LocateVex

void CreateUDG(ALGraph &G){
    //采用邻接表表示法，创建无向图 G
    int i , k;

    cout <<"请输入总顶点数，总边数，以空格隔开:";
    cin >> G.vexnum >> G.arcnum;            //输入总顶点数，总边数
    cout << endl;

    cout << "输入点的名称，如 a" << endl;

    for(i = 0; i < G.vexnum; ++i){          //输入各点，构造表头结点表
        cout << "请输入第" << (i+1) << "个点的名称:";
        cin >> G.vertices[i].data;          //输入顶点值
        G.vertices[i].firstarc=NULL;        //初始化表头结点的指针域为 NULL
    }//for
    cout << endl;

    cout << "输入边依附的顶点，如 a b" << endl;

    for(k = 0; k < G.arcnum;++k){           //输入各边，构造邻接表
        VerTexType v1 , v2;
        int i , j;
        cout << "请输入第" << (k + 1) << "条边依附的顶点:";
        cin >> v1 >> v2;                    //输入一条边依附的两个顶点
        i = LocateVex(G, v1);   j = LocateVex(G, v2);
        //确定 v1 和 v2 在 G 中位置，即顶点在 G.vertices 中的序号
```

```
        ArcNode *p1=new ArcNode;              //生成一个新的边结点*p1
        p1->adjvex=j;                         //邻接点序号为 j
        p1->nextarc= G.vertices[i].firstarc;  G.vertices[i].firstarc=p1;
        //将新结点*p1 插入顶点 vi 的边表头部

        ArcNode *p2=new ArcNode;              //生成另一个对称的新的边结点*p2
        p2->adjvex=i;                         //邻接点序号为 i
        p2->nextarc= G.vertices[j].firstarc;  G.vertices[j].firstarc=p2;
        //将新结点*p2 插入顶点 vj 的边表头部
    }//for
}//CreateUDG

void DFS(ALGraph G, int v){                   //图 G 为邻接表类型
    cout << G.vertices[v].data << "    ";
    visited[v] = true;          //访问第 v 个顶点 ,并置访问标志数组相应分量值为 true
    ArcNode *p = G.vertices[v].firstarc;      //p 指向 v 的边链表的第一个边结点
    while(p != NULL){                         //边结点非空
        int w = p->adjvex;                    //表示 w 是 v 的邻接点
        if(!visited[w])  DFS(G, w);           //如果 w 未访问，则递归调用 DFS
        p = p->nextarc;                       //p 指向下一个边结点
    }
}//DFS

int main(){
    cout << "**********算法 6.6  采用邻接表表示图的深度优先搜索遍历
**********" << endl << endl;
    ALGraph G;
    CreateUDG(G);
    cout << endl;
    cout << "无向连通图 G 创建完成！" << endl << endl;

    cout << "请输入遍历连通图的起始点：";
    VerTexType c;
    cin >> c;

    int i;
    for(i = 0 ; i < G.vexnum ; ++i){
        if(c == G.vertices[i].data)
            break;
    }
    cout << endl;
    while(i >= G.vexnum){
        cout << "该点不存在，请重新输入！" << endl;
        cout << "请输入遍历连通图的起始点：";
        cin >> c;
```

```
        for(i = 0 ; i < G.vexnum ; ++i){
            if(c == G.vertices[i].data)
                break;
        }
    }

    cout << "深度优先搜索遍历图结果：" << endl;
    DFS(G , i);

    cout <<endl;
    return 0;
}//main
```

实验五：顺序查找和折半查找

预计课时：2 学时

实训时长： 90(分钟)

实验简介：对顺序表实现顺序查找和折半查找，比较查找效率。

实验目标：

熟悉顺序查找和折半查找算法，掌握其时间复杂度。

实验内容：

1、顺序查找;

2、折半查找。

实验所需基础：

实验环境：Visual C++

实验是否需要联网：否

实训步骤：

```
//算法 7.1　顺序查找
#include<iostream>
using namespace std;
#define MAXSIZE 100
#define OK 1;

typedef struct{
int key;//关键字域
}ElemType;

typedef struct{
ElemType *R;
int length;
}SSTable;

int InitList_SSTable(SSTable &L)
{
L.R=new ElemType[MAXSIZE];
if (!L.R)
{
 cout<<"初始化错误";
 return 0;
}
L.length=0;
return OK;
}

int Insert_SSTable(SSTable &L)
{
int j=0;
for(int i=0;i<MAXSIZE;i++)
{
 L.R[i].key=j;
 L.length++;
 j++;
}
return 1;
}

int Search_Seq(SSTable ST, int key){
      //在顺序表 ST 中顺序查找其关键字等于 key 的数据元素。若找到，则函数值
为
      //该元素在表中的位置，否则为 0
        for (int i=ST.length; i>=1; --i)
                  if (ST.R[i].key==key) return i;          //从后往前找
        return 0;
```

```
    }// Search_Seq

void Show_End(int result,int testkey)
{
if(result==0)
 cout<<"未找到"<<testkey<<endl;
else
 cout<<"找到"<<testkey<<"位置为"<<result<<endl;
return;
}
void main()
{
SSTable ST;
InitList_SSTable(ST);
Insert_SSTable(ST);
int testkey1=7,testkey2=200;
int result;
result=Search_Seq(ST, testkey1);
Show_End(result,testkey1);
result=Search_Seq(ST, testkey2);
Show_End(result,testkey2);
}

//算法 7.2　设置监视哨的顺序查找
#include<iostream>
using namespace std;
#define MAXSIZE 100
#define OK 1;

typedef struct{
int key;//关键字域
}ElemType;

typedef struct{
ElemType *R;
int length;
}SSTable;

int InitList_SSTable(SSTable &L)
{
L.R=new ElemType[MAXSIZE];
if (!L.R)
{
 cout<<"初始化错误";
 return 0;
}
```

```
L.length=0;
return OK;
}

int Insert_SSTable(SSTable &L)
{
int j=1;//空出 ST.R[0]的位置
for(int i=1;i<MAXSIZE;i++)
{
 L.R[i].key=j;
 L.length++;
 j++;
}
return 1;
}

int Search_Seq(SSTable ST, int key){
   //在顺序表 ST 中顺序查找其关键字等于 key 的数据元素。若找到，则函数值为
        //该元素在表中的位置，否则为 0
       ST.R[0].key = key;                                          // “哨兵”
       for(int i = ST.length; ST.R[i].key!=key; --i)   ;          //从后往前找
       return i;
}// Search_Seq
void Show_End(int result,int testkey)
{
if(result==0)
 cout<<"未找到"<<testkey<<endl;
else
 cout<<"找到"<<testkey<<"位置为"<<result<<endl;
return;
}
void main()
{
SSTable ST;
InitList_SSTable(ST);
Insert_SSTable(ST);
int testkey1=7,testkey2=200;
int result;
result=Search_Seq(ST, testkey1);
Show_End(result,testkey1);
result=Search_Seq(ST, testkey2);
Show_End(result,testkey2);
}

//算法 7.3  折半查找
#include<iostream>
```

```
using namespace std;
#define MAXSIZE 100
#define OK 1;

typedef struct{
int key;//关键字域
}ElemType;

typedef struct{
ElemType *R;
int length;
}SSTable;

int InitList_SSTable(SSTable &L)
{
L.R=new ElemType[MAXSIZE];
if (!L.R)
{
 cout<<"初始化错误";
 return 0;
}
L.length=0;
return OK;
}

int Insert_SSTable(SSTable &L)
{
int j=1;
for(int i=1;i<MAXSIZE;i++)
{
 L.R[i].key=j;
 L.length++;
 j++;
}
return 1;
}

int Search_Bin(SSTable ST,int key) {
    // 在有序表 ST 中折半查找其关键字等于 key 的数据元素。若找到，则函数值
为
    // 该元素在表中的位置，否则为 0
    int low=1,high=ST.length;                              //置查找区间初值
    int   mid;
    while(low<=high) {
    mid=(low+high) / 2;
        if (key==ST.R[mid].key)   return mid;              //找到待查元素
```

```
            else if (key<ST.R[mid].key)    high = mid -1;          //继续在前一子表进
行查找
            else    low =mid +1;                                    //继续在后一子
表进行查找
        }//while
        return 0;                                               //表中不存在待查元素
    }// Search_Bin

    void Show_End(int result,int testkey)
    {
    if(result==0)
     cout<<"未找到"<<testkey<<endl;
    else
     cout<<"找到"<<testkey<<"位置为"<<result<<endl;
    return;
    }

    void main()
    {
    SSTable ST;
    InitList_SSTable(ST);
    Insert_SSTable(ST);
    int testkey1=7,testkey2=200;
    int result;
    result=Search_Bin(ST, testkey1);
    Show_End(result,testkey1);
    result=Search_Bin(ST, testkey2);
    Show_End(result,testkey2);
    }
```

实验六：综合性实验

预计课时：2 学时

实训时长： 90(分钟)

实验简介：选取一个合适的数据结构存储数据，能对数据进行插入、删除，用不同查找算法进行查找、用不同的排序算法进行排序等。

实验目标：

综合运用所学数据结构知识，提高解决实际问题的能力。

实验内容：

设计并实现一个学生管理系统，即定义一个包含学生信息（学号，姓名，成绩）的顺序表，可以不考虑重名的情况，系统至少包含以下功能：

1、根据指定学生个数，逐个输入学生信息；

2、逐个显示学生表中所有学生的相关信息；

3、给定一个学生信息，插入到表中指定的位置；

4、删除指定位置的学生记录；

5、统计表中学生个数；

6、利用直接插入排序或者折半插入排序按照姓名进行排序；

7、利用快速排序按照学号进行排序；

8、根据姓名进行折半查找，要求使用递归算法实现，成功返回此学生的学号和成绩；

9、根据学号进行折半查找，要求使用非递归算法实现，成功返回此学生的姓名和成绩。

实验所需基础：

实验环境：Visual C++

实验是否需要联网：否

实训步骤：

```
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <cmath>
#include <cstdlib>
using namespace std;
#define MAXSIZE 100//假设的表的最大长度;
typedef struct{
    int no;    //8位学号
    char name[20]; //姓名
    int price;       //成绩
}Student,ElemType;
typedef   struct {
  Student    *elem=NULL;       //指向数据元素的基地址
  int   length;          //线性表的当前长度
 }SqList;

SqList L;//声明一个数据表

void InitList()
{//构造一个空的顺序表
    L.elem=new ElemType[MAXSIZE];
    if(!L.elem) exit(0);//存储分配失败退出
    L.length=0;
}
int menu_select()//选择菜单函数
{
    char s[3];
    int c;
    printf("\n          ***************主菜单**************\n");
    printf("          *     1. 录入新记录                 *\n");
    printf("          *     2. 浏览显示所有记录           *\n");
    printf("          *     3. 插入记录                   *\n");
    printf("          *     4. 删除记录                   *\n");
    printf("          *     5. 统计记录                   *\n");
    printf("          *     6. 按姓名顺序显示所有记录     *\n");
```

```
        printf("              *      7. 按学号顺序显示所有记录      *\n");
        printf("              *      8. 通过姓名查找记录            *\n");
        printf("              *      9. 通过学号查找记录            *\n");
        printf("              *      10. 退出                       *\n");
        printf("              ************************************\n\n");

        do
        {
            printf("              请选择操作(1-10):");
            scanf("%s",s);
            c=atoi(s);
        }while(c<0||c>10); /*选择项不在~10之间重输*/
        return(c); /*返回选择项，主程序根据该数调用相应的函数*/
}
/*数据表取值*/
void Create()
{//新建学生数据
    int n;
    cout<<"根据指定学生个数，逐个输入学生信息:"<<endl;
    cout<<"学生个数 n=";
    cin>>n;
    if(n<1||L.length+n>MAXSIZE){
        cout<<"创建失败"<<endl;
        return ;
    }
    for(int i=0;i<n;i++)//逐个添加数据
    {
        L.length++;//每添加一个,则更新一次长度。
        cout<<"第"<<i+1<<"个学生数据"<<endl<<"姓名:";
        cin>>L.elem[L.length].name;
        cout<<"学号:";
        cin>>L.elem[L.length].no;
        cout<<"成绩:";
        cin>>L.elem[L.length].price;
    }
    cout<<"***操作成功***"<<endl;
}
void ShowAllDate()
{
    if(L.length<1)
    {
        cout<<endl<<"      很遗憾，空表中没有任何记录可供显示!"<<endl;
        return ;
    }
    cout<<"************ STUDENT ************"<<endl;
    cout<<"   编号       姓名       学号       成绩"<<endl;
```

```
	cout<<"--------------------------------"<<endl;
	for(int i=1;i<=L.length;i++)
	{
		cout<<"		"<<i<<"			"<<L.elem[i].name<<"
"<<L.elem[i].no<<"		"<<L.elem[i].price<<endl;
	}
	cout<<"********************************"<<endl;
	cout<<"***操作成功***"<<endl;
}
/*插入数据*/
bool InsertByID()
{//在顺序表 L 中第 i 个位置插入新的元素 e,i 值的合法范围三1<=i<=L.length+1
	int i;
	ElemType e;
	cout<<"给定一个学生信息，插入到表中指定的位置"<<endl;
	ShowAllDate();
	cout<<"请输入插入的位置:";
	cin>>i;
	if((i<1)||(i>L.length+1)){cout<<"位置不合法"; return false;}//i 值不合法
	if(L.length==MAXSIZE) {cout<<"存储空间已满";return false;}//存储空间已满
	cout<<"请输入插入的数据:"<<endl;
	cout<<"姓名:";
	cin>>e.name;
	cout<<"学号:";
	cin>>e.no;
	cout<<"成绩:";
	cin>>e.price;
	for(int j=L.length;j>=i;j--){
		L.elem[j+1]=L.elem[j];
	}
	L.elem[i]=e;
	++L.length;//更新记录数
	cout<<"***操作成功***"<<endl;
	return true;
}
bool DeleteByID()
{//在顺序表 L 中删除第 i 个元素,i 值的合法范围三1<=i<=L.length
	int i;
	cout<<"删除指定位置的学生记录"<<endl;
	cout<<"位置 i=";
	cin>>i;
	if((i<1)||(i>L.length)){
		cout<<"位置不合法";
		return false;
	}
	for(int j=i;j<=L.length-1;j++){
```

```
            L.elem[j]=L.elem[j+1];
        }
        --L.length;
        cout<<"***操作成功***"<<endl;
        return true;
}
/*利用直接插入排序或者折半插入排序按照姓名进行排序；*/
int flag=0;//2表示数据已经更新为姓名排序方案

void InsertSort(ElemType *a);
void BInsertSort(ElemType *a);
//姓名排序操作函数
void SelectSort()
{
    if(L.length<1){
        cout<<endl<<"      很遗憾，空表中没有任何记录可供显示!"<<endl;
        return ;
    }
    cout<<"在按姓名排序之前请先选择排序方法:1.直接插入排序   2.折半插入排序
"<<endl;
    cout<<"请选择序号1-2(默认选择1):";
    int n;
    cin>>n;
    cout<<"请确认记录是否按排序方式保存 :1.直接显示排序记录  2.保存排序方案并
显示记录"<<endl;
    cout<<"请选择序号1-2(默认选择1):";
    cin>>flag;
    if(flag!=2){//按姓名排序方式直接显示记录
        ElemType a[MAXSIZE];//排序专用临时数组
        //把数据拷贝至排序数组中
        for(int i=1;i<=L.length;i++) a[i]=L.elem[i];
        if(n==2)
            BInsertSort(a);
        else
            InsertSort(a);
        //因为默认选择序号1，用户只能操作一次选择，选择错误即默认选择序号1
        cout<<"************ STUDENT ************"<<endl;
        cout<<"  编号       姓名       学号       成绩"<<endl;
        cout<<"--------------------------------"<<endl;
        for(int i=1;i<=L.length;i++)
        {
            cout<<"    "<<i<<"          "<<a[i].name<<"          "<<a[i].no<<"
"<<a[i].price<<endl;
        }
        cout<<"********************************"<<endl;
        cout<<"***操作成功***"<<endl;
```

```
            flag=1;//防止用户输入非1非2数值
        }else{//按姓名排序方式改变记录排序方案并显示记录
            if(n==2)
                BInsertSort(L.elem);
            else
                InsertSort(L.elem);
            //因为默认选择序号1，用户只能操作一次选择，选择错误即默认选择序号1
            cout<<"************ STUDENT ************"<<endl;
            cout<<"  编号       姓名       学号       成绩"<<endl;
            cout<<"--------------------------------"<<endl;
            for(int i=1;i<=L.length;i++)
            {
                cout<<"          "<<i<<"                    "<<L.elem[i].name<<"
"<<L.elem[i].no<<"          "<<L.elem[i].price<<endl;
            }
            cout<<"*********************************"<<endl;
            cout<<"***操作成功***"<<endl;
        }
}
//此处为直接插入排序。对姓名排序
void InsertSort(ElemType a[])
{
    for(int i=2;i<=L.length;i++)
        if(strcmp(a[i].name,a[i-1].name)<0)
        {
            a[0]=a[i];
            a[i]=a[i-1];
            int j;
            for(j=i-2;strcmp(a[0].name,a[j].name)<0;j--)
                a[j+1]=a[j];
            a[j+1]=a[0];
        }
}
//此处为折半插入排序。对姓名排序
void BInsertSort(ElemType a[])
{
    for(int i=2;i<=L.length;i++)
    {
        a[0]=a[i];
        int low=1,high=i-1;
        while(low<=high)
        {
            int m=(low+high)/2;
            if(strcmp(a[0].name,a[m].name)<0) high=m-1;
            else low=m+1;
        }
```

```
            for(int j=i-1;j>=high+1;--j) a[j+1]=a[j];
            a[high+1]=a[0];
        }
}
/*利用快速排序按照学号进行排序*/
int Partition(ElemType a[],int low,int high)
{//对顺序表 a 中 low..high 进行一趟排序，返回枢轴位置
        a[0]=a[low];
        int pivotkey=a[low].no;
        while(low<high)
        {
            while(low<high &&a[high].no>=pivotkey) --high;
            a[low]=a[high];
            while(low<high&&a[low].no<=pivotkey) ++low;
            a[high]=a[low];
        }
        a[low]=a[0];
        return low;
}
//快速排序，对学号进行排序
void QSort(ElemType a[],int low,int high)
{//调用前置初值:low=1;high=L.length;
//对顺序表 a 中子序列 low..high 做快速排序
        if(low<high){
            int pivotloc=Partition(a,low,high);
            QSort(a,low,pivotloc-1);
            QSort(a,pivotloc+1,high);
        }
}
//学号排序操作函数
void QuickSort()
{
        if(L.length<1){
            cout<<endl<<"       很遗憾，空表中没有任何记录可供显示!"<<endl;
            return ;
        }
        cout<<"请确认记录是否按排序方式保存 :1.直接显示排序记录  2.保存排序方案并
显示记录"<<endl;
        cout<<"请选择序号1-2(默认选择1):";
        cin>>flag;
        //把数据拷贝至排序数组中
        if(flag!=2){//不改变原数据情况下排序并输出结果
            ElemType a[MAXSIZE];//排序专用临时数组
            for(int i=1;i<=L.length;i++) a[i]=L.elem[i];
            //因为默认选择序号1，用户只能操作一次选择，选择错误即默认选择序号1
            QSort(a,1,L.length);
```

```
            cout<<"************ STUDENT ************"<<endl;
            cout<<"  编号      姓名      学号      成绩"<<endl;
            cout<<"--------------------------------"<<endl;
            for(int i=1;i<=L.length;i++)
            {
                cout<<"    "<<i<<"          "<<a[i].name<<"          "<<a[i].no<<"
"<<a[i].price<<endl;
            }
            cout<<"**********************************"<<endl;
            cout<<"***操作成功***"<<endl;
            flag=1;
        }else{
            QSort(L.elem,1,L.length);
            cout<<"************ STUDENT ************"<<endl;
            cout<<"  编号      姓名      学号      成绩"<<endl;
            cout<<"--------------------------------"<<endl;
            for(int i=1;i<=L.length;i++)
            {
                cout<<"          "<<i<<"                    "<<L.elem[i].name<<"
"<<L.elem[i].no<<"          "<<L.elem[i].price<<endl;
            }
            cout<<"**********************************"<<endl;
            cout<<"***操作成功***"<<endl;
            flag=3;//在程序内部，2是代表姓名排序后的数据，3代表学号排序结果
        }
}
/*根据姓名进行折半查找，要求使用递归算法实现，成功返回此学生的学号和成绩；*/
//折半查找,递归方法
int Search_Bin(ElemType a[],char *key,int low,int high)
{
    if(low>high)
        return 0;
    else{
        int mid=(low+high)/2;
        if(strcmp(key,a[mid].name)==0) return mid;
        else if(strcmp(key,a[mid].name)<0) Search_Bin(a,key,low,mid-1);
        else Search_Bin(a,key,mid+1,high);
    }
}
//折半查找操作函数
void BinSearchByName()
{
    if(L.length<1){
        cout<<endl<<"      很遗憾，空表中没有任何记录可供显示!"<<endl;
        return ;
    }
```

```
    char s[20];
    cout<<"根据姓名进行查找，返回此学生的学号和成绩"<<endl;
    cout<<"请输入学生姓名:";
    cin>>s;
    if(flag!=2){
        cout<<"检测到您还未对数据库数据进行姓名排序。";
        cout<<"现在有两种方案 :1.不改变原数据情况下查找  2.给原数据按姓名排序
后再查找"<<endl;
        cout<<"请选择序号1-2(默认选择1):";
        cin>>flag;
        if(flag!=2)//等于直接跳出
        {//不改变原数据情况下查找
            ElemType a[MAXSIZE];//排序专用临时数组
            for(int i=1;i<=L.length;i++) a[i]=L.elem[i];
            //因为默认选择序号1，用户只能操作一次选择，选择错误即默认选择序
号1
            BInsertSort(a);
            int pos=Search_Bin(a,s,1,L.length);
            if(pos){//查找到数据后
                cout<<"************ STUDENT ************"<<endl;
                cout<<"   姓名       学号       成绩"<<endl;
                cout<<"--------------------------------"<<endl;
                cout<<"       "<<a[pos].name<<"                "<<a[pos].no<<"
"<<a[pos].price<<endl;
                cout<<"********************************"<<endl;
            }else{
                cout<<"没有查找到姓名："<<s<<"的记录";
            }
            flag=1;
        }else{
            BInsertSort(L.elem);
        }
    }
    if(flag==2){
        int pos=Search_Bin(L.elem,s,1,L.length);
        if(pos){//查找到数据后
            cout<<"************ STUDENT ************"<<endl;
            cout<<"   编号       姓名       学号       成绩"<<endl;
            cout<<"--------------------------------"<<endl;
            cout<<"         "<<pos<<"                  "<<L.elem[pos].name<<"
"<<L.elem[pos].no<<"          "<<L.elem[pos].price<<endl;
            cout<<"********************************"<<endl;
        }else{
            cout<<"没有查找到姓名："<<s<<"的记录";
        }
    }
```

```
}
/*根据学号进行折半查找，要求使用非递归算法实现，成功返回此学生的姓名和成绩。
*/
//折半查找,非递归方法
int Non_Search_Bin(ElemType a[],int key)
{
	int low=1,high=L.length;
	while(low<=high)
	{
		int mid=(low+high)/2;
		if(key==a[mid].no) return mid;
		else if(key<a[mid].no) high=mid-1;
		else low=mid+1;
	}
	return 0;
}
void BinSearchByID()
{
	if(L.length<1){
		cout<<endl<<"	很遗憾，空表中没有任何记录可供显示!"<<endl;
		return ;
	}
	int s;
	cout<<"根据学号进行查找，返回此学生的姓名和成绩"<<endl;
	cout<<"请输入学生学号:";
	cin>>s;
	if(flag!=3){
		cout<<"检测到您还未对数据库数据进行学号排序。";
		cout<<"现在有两种方案 :1.不改变原数据情况下查找  2.给原数据按学号排序
后再查找"<<endl;
		cout<<"请选择序号1-2(默认选择1):";
		cin>>flag;
		if(flag!=2)//等于时直接跳出
		{//不改变原数据情况下查找
			ElemType a[MAXSIZE];//排序专用临时数组
			for(int i=1;i<=L.length;i++) a[i]=L.elem[i];
		 //因为默认选择序号1，用户只能操作一次选择，选择错误即默认选择序号1
			QSort(a,1,L.length);
			int pos=Non_Search_Bin(a,s);
			if(pos){//查找到数据后
				cout<<"************ STUDENT ************"<<endl;
				cout<<"  姓名	学号	成绩"<<endl;
				cout<<"--------------------------------"<<endl;
				cout<<"	"<<a[pos].name<<"		"<<a[pos].no<<"
"<<a[pos].price<<endl;
				cout<<"********************************"<<endl;
```

```
            }else{
                cout<<"没有查找到姓名："<<s<<"的记录";
            }
            flag=1;
        }else{
            QSort(L.elem,1,L.length);
            flag=3;
        }
    }
    if(flag==3){
        int pos=Non_Search_Bin(L.elem,s);
        if(pos){//查找到数据后
            cout<<"************ STUDENT ************"<<endl;
            cout<<"  编号      姓名      学号      成绩"<<endl;
            cout<<"--------------------------------"<<endl;
            cout<<"        "<<pos<<"                  "<<L.elem[pos].name<<"
"<<L.elem[pos].no<<"          "<<L.elem[pos].price<<endl;
            cout<<"********************************"<<endl;
        }else{
            cout<<"没有查找到姓名："<<s<<"的记录";
        }
    }
}
int main()
{
    system("color 3e");                 /*清屏*/
    InitList();
    for(;;)                     /*无限循环*/
    {
        switch(menu_select())                   /*调用主菜单函数，返回值整数作开
关语句的条件*/
        {
            case 1: Create();break;             //新建记录
            case 2: ShowAllDate();break;    //显示全部记录
            case 3: InsertByID();break;     //插入记录
            case 4: DeleteByID();break;     //通过 ID 删除记录
            case 5: cout<<"当前存档学生个数为:"<<L.length;break;    //显示表
中学生个数
            case 6: SelectSort();break;//利用直接插入排序或者折半插入排序按照
姓名进行排序；
            case 7: QuickSort();break;//利用快速排序按照学号进行排序；
            case 8: BinSearchByName();break;//根据姓名进行折半查找，要求使用
递归算法实现，成功返回此学生的学号和成绩；
            case 9: BinSearchByID();break;//根据学号进行折半查找，要求使用非递
归算法实现，成功返回此学生的姓名和成绩。
            case 10: exit(0);                       //程序结束*/
```

```
            }
        }
        return 0;
}
```