



西安电子科技大学
XIDIAN UNIVERSITY

数据结构上机实验手册

西安电子科技大学软件学院 数据结构课程组

2014 年 2 月

实验目录

数据结构上机实验手册.....	1
实验 0 数组、指针和结构体.....	3
题目一 数据集合的表示及运算.....	3
题目二 约瑟夫问题.....	3
题目三 复数运算.....	4
实验一 链表的实现及运算.....	4
题目一 单链表基本运算.....	4
题目二 单链表上的排序运算.....	6
题目三 约瑟夫问题.....	6
题目四 一元多项式相加、减运算器.....	6
实验二 栈和队列的实现与应用.....	7
题目一 数制转换.....	7
题目二 括号匹配问题.....	8
题目三 停车场管理.....	9
题目四 迷宫问题.....	9
实验三 字符串运算.....	11
题目一 字符串运算.....	11
题目二 文学研究助手.....	12
实验四 二叉树的运算与应用.....	12
题目一 二叉树的遍历运算.....	12
题目二 哈夫曼编/译码器.....	13
实验五 查找方法.....	15
题目一 顺序查找、折半查找.....	15
题目二 二叉排序树的建立、查找、插入和删除运算.....	15
题目三 哈希表的设计和应用.....	15
实验六 常用的排序方法.....	16
题目一 简单排序算法.....	16
题目二 快速排序.....	16
实验七 图的遍历.....	17
题目一 深度优先遍历.....	17
题目二 广度优先遍历.....	19
附录 A 实验报告模板.....	20
附录 B 实验报告示例.....	20

实验 0 数组、指针和结构体

题目一 数据集合的表示及运算

【问题描述】

构造一个非递减数列，然后在该数列中新加入一个数，并保持该数列非递减有序的特性。

【基本要求】

用一维数组存储数列。

【实现提示】

1. 为简单起见，设数组元素为整型，用静态数组（即固定大小的数组）存储数列，定义如下：

```
#define ARRAYSIZE 10000
```


//先定义一个足够大的数组，并存入少量数据测试程序的逻辑是否正确

```
int data[ARRAYSIZE] = {10,20,30,40,50};
```



```
int n = 5; // 数列的长度用 n 表示，程序中应保证 n 不大于 ARRAYSIZE
```
2. 新加入的数从键盘输入，该数有以下几种可能性，在运行程序时都要进行测试：
 - （1）比数列中最小的数还要小；
 - （2）比数列中最大的数还要大；
 - （3）其他情况
3. 调用随机函数产生数列及要加入的新数，这就不能保证数列的非递减性质了，因此需要在程序中增加一个专门用来排序的函数；
- 4*. 用动态数组，即用动态内存申请函数申请内存块（作为数组的存储空间），借助指针访问数组元素，数组大小自行设定，例如：

```
#define DSIZE 100
```



```
int n = 5; //数列的长度用 n 表示，程序中应保证 n 不大于 DSIZE
```



```
int *ptr;
```



```
ptr = (int *)malloc(DSIZE * sizeof(int));
```


//calloc 也可以用来申请内存块
//与 malloc 不同的是，calloc 函数会将申请到的内存区域初始化为 0
//用法示例：ptr = (int *)calloc(DSIZE, sizeof(int));

```
if (!ptr) //申请失败，程序中止
```



```
exit(0);
```



```
for(i = 0; i < n; i++)
```



```
*(ptr+i) = rand(); //调用随机函数产生数据，*(ptr+i)等同于 ptr[i]
```

【测试数据】

自行设定。

题目二 约瑟夫问题

【问题描述】

约瑟夫问题是由公元 1 世纪时的犹太历史学家（也是军官及辩论家）弗拉维奥·约瑟夫斯提出的，他参加并记录了公元 66-70 年犹太人反抗罗马的起义。他和他的 40 个战友被罗马军队包围在洞

中。他们在讨论是自杀还是被俘，最终决定自杀，并以抽签的方式决定谁杀掉谁。约瑟夫斯和另外一个人是最后两个留下的人。约瑟夫斯说服了那个人，他们将向罗马军队投降，不再自杀。约瑟夫斯把他的存活归因于运气或天意，他不知道是哪一个。

在计算机编程的算法中，类似问题又称为**约瑟夫环**：n 个人（编号 1~n）围成一圈，从编号为 1 的人开始，依顺时针方向进行。从 1 开始报数，报到 m 的人退出，从下一个人开始，继续从 1 开始报数，报到 m 的人退出，重复上述过程，直到仅剩 1 人（称为胜利者）。按顺序输出出列者编号，以及胜利者的编号。

【基本要求】

用一维数组存储。

【测试数据】

m 和 n 的值自行设定。

题目三 复数运算

【问题描述】

复数是复变函数论、解析数论、傅里叶分析、分形、流体力学、相对论、量子力学等学科中最基础的对象和工具。设 $z_1=a+bi$ ， $z_2=c+di$ 是任意两个复数，则复数的四则运算为：

1. 相加 $z_3 = z_1 + z_2$: $z_3 = a+bi + c+di = (a+c) + (b+d)i$
2. 相减 $z_3 = z_1 - z_2$: $z_3 = a+bi - (c+di) = (a-c) + (b-d)i$
3. 相乘 $z_3 = z_1 * z_2$: $z_3 = (a+bi) * (c+di) = (ac-bd) + (bc+ad)i$
4. 相除 $z_3 = z_1 / z_2$: $z_3 = (a+bi) / (c+di) = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$

【基本要求】

1. 用结构体类型描述复数数据；
2. 定义四个函数分别实现复数的四则运算；
3. 实部和虚部为 double 型，输出时小数点后保留 2 位。

【测试数据】

自行设定。

实验一 链表的实现及运算

题目一 单链表基本运算

【问题描述】

设计并实现线性表的单链表存储和运算。

【基本要求】

实现单链表的插入、删除和遍历运算，每种操作用一个函数实现。

插入操作：将一个新元素插入表中指定序号的位置。

删除操作：将指定序号的元素从表中删除。

遍历操作：从表头按次序输入所有元素的值，若是空表，则输出信息“empty list!”。

【实现提示】

1. 程序运行时，首先在 main 函数中创建空的、带头结点的单链表。然后多次调用实现插入操作

的函数（每次都把元素在序号 1 位置上插入），将元素依次插入表中，最后调用实现遍历操作的函数输出所有元素。之后再多次调用实现删除操作的函数将表还原为空表（每次都删除第 1 个元素，每删除一个元素后，将表中剩余元素都输出一次）。

2. 单链表结点类型定义：

```
typedef int ElemType; //为简化起见，元素类型定义为整型
typedef struct Node{
    ElemType data;
    struct Node *next;
}Node, *LinkList;
```

3. 为了简化指针参数传递，使用 c++ 的引用参数，存储源程序时注意后缀名应为 **cpp**。（其他实验题目在此处的处理相同）

4. 创建链表时，可以多次调用插入函数（插入函数的功能是在链表的第 i 个元素结点前插入一个新结点），通过不断地在链表中增加结点来实现，也可以定义一个专门创建链表的函数，调用一次该函数就完成链表中所有结点的创建（当然，此后需要在链表中插入一个新结点时调用前述的插入函数即可）。

例如，用头插法创建长度为 n 的单链表的函数定义如下：

```
Status CreateList_L(LinkList &L, int n)
{    //用表头插入法逆序建立长度为 n 的、带头结点的单链表
    int i;
    LinkList p;
    L = (LinkList) malloc(sizeof(Node));
    if (!L)
        return INFEASIBLE;
    L->next = NULL;    //建立仅有头结点的单链表

    for(i=n; i>0; --i) {    //输入 n 个整数，逐个插入单链表中
        p = (LinkList) malloc(sizeof(Node));    //生成新结点
        if (!p)
            return INFEASIBLE;
        scanf("%d", &p->data);    //从键盘输入元素值，存入新结点中
        p->next = L->next;    L->next = p;    //新结点插入到表头
    }
    return OK;
}
```

【测试数据】

输入数据：1 2 3 4 5 0（为 0 时结束，0 不存入链表）

第一次输出：5 4 3 2 1

第二次输出：4 3 2 1

第三次输出：3 2 1

第四次输出：2 1

第五次输出：1

第六次输出：empty list!

题目二 单链表上的排序运算

【问题描述】

创建单链表，用冒泡排序方法（或其他排序方法）实现非递减排序。

【基本要求】

创建单链表，遍历单链表并输出元素序列，将单链表中的元素按非递减顺序排序，再次遍历单链表并输出元素序列。

【实现提示】

分别定义函数实现创建单链表、遍历单链表并输出和排序过程。程序运行时，首先在 `main` 函数中调用创建单链表的函数，然后调用遍历单链表（包含对元素的输出）的函数，再调用排序函数，最后再调用遍历单链表（包含对元素的输出）的函数观察是否实现了排序运算。

【测试数据】

自行设计，输入的数据序列应该是无序的。

题目三 约瑟夫问题

【问题描述】

编号为 $1, 2, \dots, n$ 的 n 个人按顺时针方向围坐一圈，每人持有一个密码（正整数）。现在给定一个随机数 $m > 0$ ，从编号为 1 的人开始，按顺时针方向 1 开始顺序报数，报到 m 时停止。报 m 的人出圈，同时留下他的密码作为新的 m 值，从他在顺时针方向上的下一个人开始，重新从 1 开始报数，如此下去，直至所有的人全部出列为止。

【基本要求】

利用单向循环链表存储结构模拟此过程，按照出列的顺序打印输出每个人的编号。

【测试数据】

M 的初始值为 20； n 等于 7，7 个人的密码依次为：3，1，7，2，4，8，4。
输出为：6，1，4，7，2，3，5

【实现提示】

程序运行时，首先要求用户指定初始报数上限值，然后读取各人的密码。可设 $n \leq 30$ 。此题所用的循环链表中不需要“头结点”，请注意空表和非空表的界限。

【选作内容】

用顺序存储结构实现该题目。

题目四 一元多项式相加、减运算器

【问题描述】

设计一个一元稀疏多项式简单计算器。

【基本要求】

一元稀疏多项式简单计算器的基本功能：

(1) 输入并建立多项式的单向链表存储；

(2) 输出多项式，形式为一个整数序列： $n, c_1, e_1, c_2, e_2, \dots, c_n, e_n$ ，其中， n 是多项式的项数， c_i 、 e_i 分别是第 i 项的系数和指数，序列按指数降序排列；

- (3) 多项式 $A(x)$ 和 $B(x)$ 相加得到多项式 $C(x)$ ，输出 $C(x)$ ；
 (4) 多项式 $A(x)$ 和 $B(x)$ 相减得到多项式 $D(x)$ ，输出 $D(x)$ 。

【测试数据】

- (1) $A(x)=2x+5x^8-3.1x^{11}$ $B(x)=7-5x^8+11x^9$ $C(x)=-3.1x^{11}+11x^9+2x+7$
 (2) $A(x)=1+x+x^2+x^3+x^4+x^5$ $B(x)=-x^3-x^4$ $C(x)=1+x+x^2+x^5$
 (3) $A(x)=x+x^3$ $B(x)=-x-x^3$ $C(x)=0$
 (4) $A(x)=x+x^{100}$ $B(x)=x^{100}+x^{200}$ $C(x)=x+2x^{100}+x^{200}$
 (5) $A(x)=x+x^2+x^3$ $B(x)=0$ $C(x)=x+x^2+x^3$
 (6) $A(x)=6x^{-3}-x+4.4x^2-1.2x^9$ $B(x)=-6x^{-3}+5.4x^2-x^2+7.8x^{15}$
 $D(x)=-7.8x^{15}-1.2x^9+12x^{-3}-x$

【实现提示】

用带头结点的单链表存储多项式，多项式的项数可放入头结点中。

实验二 栈和队列的实现与应用

题目一 数制转换

【问题描述】

十进制数 N 和其它 d 进制数的转换是计算机实现计算的基本问题，其解决方法很多，其中一个简单的算法是基于下列原理：

$N = (N \text{ div } d) * d + N \text{ mod } d$ 。其中： div 为整除运算， mod 为求余运算。

例如， $43_{10} = 101011_2$

被除数	除数	商	余数
43	2	21	1
21	2	10	1
10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

从上述运算过程可见：①从低位到高位顺序产生二进制数的各个位数；②与打印输出由高位到低位的顺序相反。

因此，将计算过程中得到的二进制数的各位顺序进栈，再按出栈序列打印输出，即可得到与输入对应的二进制数。

【基本要求】

写一个程序，利用栈将一个十进制数转换为二进制数的形式输出，其中，栈的基本运算应自行实

现，不能使用开发工具中提供的栈类型。

题目二 括号匹配问题

【问题描述】

设一个表达式中可以包含三种括号：“(”和“)”，“[”和“]”，“{”和“}”，并且这三种括号可以按照任意的次序嵌套使用，考查表达式中的括号是否匹配。

【基本要求】

写一个程序，判断给定表达式中的括号是否匹配。

【测试数据】

有多个表达式，每个表达式（不超过 100 个字符）占一行。例如，

[(d+f)*{}2]

[(2+3))

()

[4(6)7)9

【实现提示】

自左至右扫描表达式，若遇左括号，则将左括号入栈，若遇右括号，则将其与栈顶的左括号进行匹配，若配对，则栈顶的左括号出栈，否则出现括号不匹配错误。

程序运行时，对输入每个表达式，判断其括号匹配情况。若其中的括号是匹配的，则输出“yes”，否则输出“no”。

【算法】

```
Status CheckBrackets(char *expr)
//表达式作为字符串保存在 expr 中，括号完全匹配时返回 OK，否则返回 ERROR
    InitStack(S);                // 构造一个空栈，用于暂存左括号;
    for(i = 0; expr[i]!='\0'; i++) {    // 逐个考查表达式中的字符，忽略非括号字符
        if (expr[i]=='(' || expr[i]=='[' || expr[i]=='{')
            push(S, expr[i]);        // 遇到左括号时，令左括号入栈
        else
            if (expr[i]==')' || expr[i]==']' || expr[i]=='}') { // 遇到右括号时，判断匹配情况
                if (StackEmpty(S))    // 遇到右括号时栈为空，即不存在与之匹配的左括号
                    return ERROR;
                else {                // 栈中有左括号
                    Gettop(S,c);      // 读取栈顶的左括号（不出栈）
                    switch (expr[i]) {
                        case ')': if (c!='(') // 不匹配：当前括号是“）”，栈顶不是“（”
                                    return ERROR;
                                    break;    // 当前的“）”与栈顶的“（”相匹配
                        case ']': if (c!='[') // 不匹配：当前括号是“】”，栈顶不是“[”
                                    return ERROR;
                                    break;    // 当前的“】”与栈顶的“[”相匹配
                        case '}': if (c!='{') // 不匹配：当前括号是“}”，栈顶不是“{”
                                    return ERROR;
                                    break;
                    }
                }
            }
    }
```



```

        return ERROR;
    break;    // 当前的“}”与栈顶的“{”相匹配
} // switch
pop(S);      // 栈顶的左括号出栈
} // if-else
} // if
} // for
if (StackEmpty(S)) return OK; // 已到表达式结尾且栈空，说明所有括号都匹配
return ERROR; // 表达式中的左括号多（留在栈中），右括号少，括号不完全匹配
}

```

题目三 停车场管理

【问题描述】

设停车场是一个可停放 n 辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满 n 辆汽车，则后来的汽车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入。当停车场内某辆车要离开时，在它之后进入的车辆必须先退出车场为它让路，待该辆车开出大门外，其他车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。试为停车场编制按上述要求进行管理的模拟程序。

【基本要求】

以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入的输入数据序列进行模拟管理。每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码以及到达或离去的时刻。对每一组输入数据进行操作后的输出信息为：若是车辆到达，则输出汽车在停车场内或便道上的停车位置，若是车辆离去，则输出汽车在停车场内停留的时间和应交纳的费用（在便道上停留的时间不收费）。栈以顺序结构实现，队列以链表结构实现。

【测试数据】

设 $n=2$ ，输入数据为：('A',1,5), ('A',2,10), ('D',1,15), ('A',3,20), ('A',4,25), ('A',5,30), ('D',2,35), ('D',4,40), ('E',0,0)。其中：'A'表示到达（Arrival）；'D'表示离去（Departure）；'E'表示输入结束（End）。

【实现提示】

需另设一个栈，临时停放为给要离去的汽车让路而从停车场退出来的汽车，也用顺序存储结构实现。输入数据按到达或离去的时刻有序。栈中的每个元素表示一辆汽车，包含两个数据项：汽车的牌照号码和进入停车场的时刻。

题目四 迷宫问题

【问题描述】

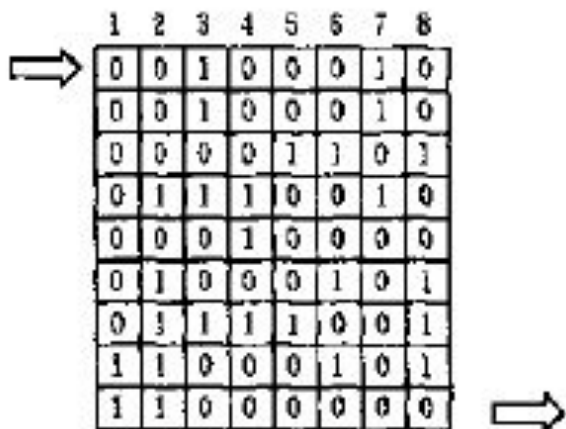
以一个 $m \times n$ 的长方阵表示迷宫，0 和 1 分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

【基本要求】

首先实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫的非递归程序。求得的通路以三元组 (i, j, d) 的形式输出，其中 (i, j) 指示迷宫中的一个位置（行号和列号）， d 表示走到下一位置的方向（对于迷宫中任一位置，均有下、右、上、左四个方向来走出下一个位置，这四个方向可分别编号为 1, 2, 3, 4）。例如，对于下面测试数据给出的迷宫，输出的一条通路为： $(1,1,1)$, $(2,1,1)$, $(3,1,1)$, $(4,1,1)$, $(5,1,2)$, $(5,2,2)$, $(5,3,1)$, ...。

【测试数据】

迷宫如下图所示：左上角 $(1,1)$ 为入口，右下角 $(8,9)$ 为出口。



	1	2	3	4	5	6	7	8
1	0	0	1	0	0	0	1	0
2	0	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	1
4	0	1	1	1	0	0	1	0
5	0	0	0	1	0	0	0	0
6	0	1	0	0	0	1	0	1
7	0	1	1	1	1	0	0	1
8	1	1	0	0	0	1	0	1
9	1	1	0	0	0	0	0	0

【实现提示】

计算机解迷宫通常用的是“穷举求解”方法，即从入口出发，顺着某一个方向进行探索，若能走通，则继续往前走；否则沿着原路退回，换一个方向继续探索，直至出口位置，求得一条通路。假如所有可能的通路都探索到而未能到达出口，则所设定的迷宫没有通路。

可以用二维数组存储迷宫数据，用数组元素的下标 $[rowNo][colNo]$ 表示坐标。

通常设定入口点的下标为 $(1,1)$ ，出口点的下标为 (n,n) 。为处理方便起见，可在迷宫的四周加一圈障碍（由行号为 0 和 $n+1$ 、列号为 0 和 $n+1$ 的所有数组元素构成）。

例如，上图所示的迷宫可如下表示：

```
#define N 8
char maze[N+2][N+2]; //此处 char 作为小整数类型使用
int rowNo,colNo;
for(rowNo = 0; rowNo < N+2; rowNo++) //迷宫位置全部初始化为有障碍
    for(colNo = 0; colNo < N+2; colNo++)
        maze[rowNo][colNo] = 1;
```

然后通过键盘输入或读取提前保存在文件中的迷宫数据设置迷宫的状态，以及出口和入口位置。

实验三 字符串运算

题目一 字符串运算

【问题描述】

字符串是非数值数据处理中常见的运算对象。串的运算有很多，下面介绍部分基本运算：

1. 求串长 StrLength(s)

操作条件：串 s 存在

操作结果：求出串 s 的长度。

2. 串赋值 StrAssign(s1,s2)

操作条件：s1 是一个串变量，s2 或者是一个串常量，或者是一个串变量（通常 s2 是一个串常量时称为串赋值，是一个串变量称为串拷贝）。

操作结果：将 s2 的串值赋值给 s1，s1 原来的值被覆盖掉。

3. 连接操作：StrConcat(s1,s2,s) 或 StrConcat(s1,s2)

操作条件：串 s1,s2 存在。

操作结果：两个串的连接就是将一个串的串值紧接着放在另一个串的后面，连接成一个串。前者是产生新串 s，s1 和 s2 不改变，后者是在 s1 的后面连接 s2 的串值，s1 改变，s2 不改变。

例如：s1=" he "，s2=" bei "，前者操作结果是 s=" he bei "；后者操作结果是 s1=" he bei "。

4. 求子串 SubStr(s,i,len):

操作条件：串 s 存在， $1 \leq i \leq \text{StrLength}(s)$ ， $0 \leq \text{len} \leq \text{StrLength}(s)-i+1$ 。

操作结果：返回从串 s 的第 i 个字符开始的长度为 len 的子串。len=0 得到的是空串。

例如：SubStr(" abcdefghi ",3,4)= " cdef "

5. 串比较 StrCmp(s1,s2)

操作条件：串 s1,s2 存在。

操作结果：若 $s1=s2$ ，操作返回值为 0；若 $s1<s2$ ，返回一个负整数；若 $s1>s2$ ，返回一个正整数。

6. 子串定位 StrIndex(s,t): 找子串 t 在主串 s 中首次出现的位置

操作条件：串 s,t 存在。

操作结果：若 $t \in s$ ，则操作返回 t 在 s 中首次出现的位置，否则返回值为-1。

如：StrIndex(" abcdebda ", " bc ")=2

StrIndex(" abcdebda ", " ba ")=-1

7. 串插入 StrInsert(s,i,t)

操作条件：串 s,t 存在， $1 \leq i \leq \text{StrLength}(s)+1$ 。

操作结果：将串 t 插入到串 s 的第 i 个字符位置上，s 的串值发生改变。

8. 串删除 StrDelete(s,i,len)

操作条件：串 s 存在， $1 \leq i \leq \text{StrLength}(s)$ ， $0 \leq \text{len} \leq \text{StrLength}(s)-i+1$ 。

操作结果：删除串 s 中从第 i 个字符开始的长度为 len 的子串，s 的串值改变。

9. 串替换 StrRep(s,t,r)

操作条件：串 s,t,r 存在，t 不为空。

操作结果：用串 r 替换串 s 中出现的所有与串 t 相等的非重叠的子串，s 的串值改变。

以上是串的几个基本操作。其中前 5 个操作是最为基本的，它们不能用其他的操作来合成，因此通常将这 5 个基本操作称为最小操作集。

【基本要求】

编程实现上述串的基本运算（不能用 C 编译环境中提供的串操作库函数），每种运算用一个函数实现。

串的存储空间采用以下两种方案：

- （1）固定大小的数组（串的赋值、连接和串替换运算要考虑超长情况）
- （2）动态申请

题目二 文学研究助手

【问题描述】

文学研究人员经常需要统计某篇英文小说中某些词语出现的次数，试写一个程序完成该统计要求。

【基本要求】

英文小说存于一个文本文件中（有多行），需要统计的单词由键盘输入。程序运行结束后输出该关键字在文中出现的总次数以及出现该关键字的行号和在该行中出现的次数。

【实现提示】

为简化起见，设单词不跨行。

由于文件可能很长，因此，不要试图将文件中所有内容全部读入后才开始统计。逐行读入文本文件的内容，每读入一行，就在该行中统计一遍指定单词的出现次数。

可以假设每行字符个数不超过 120，行号从 1 开始计数。

实验四 二叉树的运算与应用

题目一 二叉树的遍历运算

【问题描述】

二叉树是一种重要的树结构，遍历运算是其基本运算。

【基本要求】

在二叉树的二叉链表存储结构基础上，实现二叉树的以下运算：

- （1）创建二叉树的二叉树链表表示；
- （2）实现二叉树的先序遍历运算，输出先序遍历序列；
- （3）实现二叉树的中序遍历运算，输出中序遍历序列；
- （4）实现二叉树的后序遍历运算，输出后序遍历序列。

【实现提示】

应根据一个输入序列建立二叉树的二叉链表。建立二叉树的算法如下：

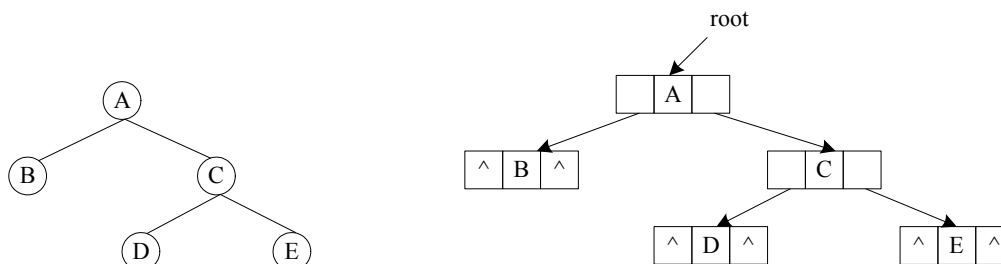
```
Status CreateBiTree(BiTree &T)
{
    char ch;
```

```

scanf("%c",&ch);
if (ch == '.')    T = NULL;
else {
    if (!(T = (BiTree)malloc(sizeof(BiTNode))))
        exit(OVERFLOW);
    T->data = ch;
    CreateBiTree(T->Lchild);
    CreateBiTree(T->Rchild);
}
return OK;
}

```

例如，调用 CreateBiTree(root)时输入序列 “AB..CD..E..”，则建立如下所示的二叉树：



另外，也可以考虑用顺序存储方式存储二叉树。

题目二 哈夫曼编/译码器

【问题描述】

利用哈夫曼编码进行通信可以大大提高对信道的利用率，缩短信息传输时间，降低传输成本。但是，这要求在发送端通过一个编码系统对待传输数据预先编码，在接收端对接收的数据进行译码（复原），对于双工信道（可以双向传输信息的信道），每端都需要一个完整的编/译码系统。试为这样的信息收发站写一个哈夫曼码的编/译码系统。

【基本要求】

一个完整的哈夫曼编/译码系统应具有以下功能：

(1) I: 初始化 (Initialization)。从终端（键盘）读入字符集的大小 n ， n 个字符及 n 个权值，建立哈夫曼树并将其存于文件 hfmTree 中。

(2) E: 编码 (Encoding)。利用已经建好的哈夫曼树，为每个叶子结点编码，并对文本文件 ToBeTran 中的正文进行编码，然后将结果存入文件 CodeFile 中。

(3) D: 译码 (Decoding)。利用已经建好的哈夫曼树，对文本文件 CodeFile 中的代码进行翻译，结果存入 TextFile 中。

(4) V: 比较文件 ToBeTran 和 TextFile 的内容是否相同，若不同，则编译码过程中有错误。

【测试数据】

(1) 8 个字符及其权值

a	b	c	d	e	f	g	h
0.05	0.29	0.07	0.08	0.14	0.23	0.03	0.11

(2) 空格和 26 个英文字母及其权值

	A	B	C	D	E	F	G	H
186	4	13	22	32	103	21	15	47
I	J	K	L	M	N	O	P	Q
57	1	5	32	20	57	63	15	1
R	S	T	U	V	W	X	Y	Z
48	51	80	23	8	18	1	16	1

【实现提示】

实现上述的哈夫曼编/译码系统时，先不考虑文件操作（建立文件、读写文件等），而是将形成的哈夫曼树存储在用 `HuffmanTree` 定义的数组中，将各叶子结点的编码保存在用 `HuffmanCode` 定义的字符数组中，需要编码的信息原文由终端（键盘）输入后存入一个字符数组 `A`，编码后的 01 序列存入另一个字符数组 `B`，最后再翻译 `B` 中的 01 串（译码），结果在终端（显示器屏）显示。

若采用文件保存信息，则以测试数据 1 为例，需要以下几个文件：

（1）文件 `hfmTree` 用于保存哈夫曼树，其内容示例如下（其中的数据来自课堂示例，每行表示树中的一个结点，结点中的权值可以不保存）：

```
a 0.05 9 0 0
b 0.29 14 0 0
c 0.07 10 0 0
d 0.08 10 0 0
e 0.14 12 0 0
f 0.23 13 0 0
g 0.03 9 0 0
h 0.11 11 0 0
  0.08 11 1 7
  0.15 12 3 4
  0.19 13 9 8
  0.29 14 5 10
  0.42 15 11 6
  0.58 15 2 12
  1.00 0 13 14
```

（2）文件 `ToBeTran` 中存储需要编码的信息原文，其内容示例如下（其中的数据来自课堂示例）：

```
bbfehagd
```

（3）文件 `CodeFile` 保存对 `ToBeTran` 的编码结果，其内容如下所示：

```
101001110001000100001111
```

（4）文件 `TextFile` 保存对 `CodeFile` 的翻译结果，其内容如下所示：

```
bbfehagd
```

注意：以上各文件中的内容都可以另行设计，不一定与这里给出的例子相同。将 `hfmTree` 中的内容读取出来后，应依次保存在下标从 1 开始的数组元素中。

实验五 查找方法

题目一 顺序查找、折半查找

【问题描述】

顺序查找是一种简单通用的查找方法。与顺序查找方法相比，折半查找的效率较高，但是它要求查找表采用顺序存储结构且元素排列有序。

【基本要求】

用顺序存储结构表示查找表，实现以下运算要求：

- (1) 建立一个整数数据文件 `datafile`;
- (2) 从文件 `datafile` 读取数据，并导入一维数组中;
- (3) 用顺序查找方法查找指定元素（由键盘输入），并显示查找结果;
- (4) 先对数组中的元素进行排序，分别用递归和非递归两种方式实现折半查找方法。

【测试数据】

用随机函数产生测试数据。

【实现提示】

题目二 二叉排序树的建立、查找、插入和删除运算

【问题描述】

二叉排序树是一种动态树表，可用于关键字检索，在二叉排序树中进行查找的过程和二分查找类似。

【基本要求】

在采用二叉链表存储结构的基础上，实现二叉排序树的以下运算：

- (1) 实现在二叉排序树上查找指定关键字的运算;
- (2) 实现在二叉排序树上插入一个关键字的运算;
- (3) 实现在二叉排序树上删除指定关键字的运算;
- (4) 实现二叉排序树的中序遍历运算，输出中序遍历序列;
- (5) 从空的二叉排序树开始，根据一个关键字序列，调用实现插入运算的函数，建立二叉排序树。

【测试数据】

使用以下两组测试数据建立二叉排序树，或自行设计测试数据。

- (1) 5 4 2 1 7 3 6
- (2) 1 2 3 4 5 6 7

题目三 哈希表的设计和应用

【问题描述】

哈希表可用于实现高效的数据存储和查找。

【基本要求】

针对记录集合设计一个哈希表，完成相应的建表和查表程序。

(1) 假定每个记录有下列数据项：电话号码（长度不超过 11 的数字串）、用户名（长度不超过 20 个字符的字符串）、地址（长度不超过 50 个字符的字符串）

(2) 从数据文件 `data.txt`（预先建立）中读入各项记录，并逐一插入到哈希表中，分别用以电话号码和姓名为关键码设计相应的哈希函数；

(3) 以姓名为关键字时，采用链地址法解决冲突构造哈希表；以电话号码为关键字时，采用线性探查法解决冲突构造哈希表；

(4) 待查找的记录从键盘输入，应显示查找结果。

【测试数据】

自行设计。

实验六 常用的排序方法

题目一 简单排序方法

【问题描述】

简单排序算法主要包括冒泡排序、简单选择排序和直接插入排序，它们都是时间复杂度为 $O(n^2)$ 的排序方法，需要熟练掌握。

【基本要求】

用随机函数产生 10000（或更多）个整数（或浮点数），保存在文件（`intfile.dat` / `realfile.dat`）中，然后将文件中的所有整数（或浮点数）读入一个数组 `A`。

(1) 用冒泡法对数组 `A` 排序；

(2) 用简单选择排序方法对数组 `A` 排序；

(3) 用直接插入排序法对数组 `A` 排序；

将上述排序算法分别用函数实现，观察每种排序过程中元素的比较次数、交换（或移动）次数，以及排序过程所消耗的时间（以 `s` 或 `ms` 为单位）。

题目二 快速排序

【问题描述】

快速排序算法是由 C.A.R. Hoare 在 1961 年发明的一种内排序算法。一般来说，快速排序算法要显著的快于其他的 $O(n \log n)$ 算法，其最坏情况发生的概率也可以通过改进设计（三者取中选取枢轴元素、分区较小时改用直接插入排序等）尽量减小。

快速排序算法的具体实现有多种，数据结构课程及教材中给出的算法如下：


```

void QSort(SqList &L, int low, int high) {
    //对 L.r[low]~L.r[high]中的元素序列进行快速排序
    if (low < high) {
        pivotloc = Partition(L,low,high);    //一趟划分，确定枢轴元素位置
        QSort(L, low, pivotloc - 1);    // 左子序列进行快速排序
        QSort(L, pivotloc+1, high);    // 右子序列进行快速排序
    }//if
} //QSort

int Partition(SqList &L,int low,int high) {
    L.r[0] = L.r[low]; pivotkey = L.r[0].key;    //元素移动
    i = low; j = high;
    while (i<j) {
        while (i<j && L.r[j].key >= pivotkey) j--;
        L.r[i] = L.r[j];    // 元素移动
        while (i<j && L.r[i].key <= pivotkey) i++;
        L.r[j] = L.r[i];    // 元素移动
    }
    L.r[i] = L.r[0];    // 元素移动
    return i;    // 返回枢轴元素最后确定的位置
} //Partition

```

【基本要求】

用随机函数产生 10000（或更多）个整数（或浮点数），保存在文件（intfile.dat / realfile.dat）中，然后将文件中的所有整数（或浮点数）读入一个数组 A。

用快速排序算法对上述数组 A 中的数据进行排序，输出排序过程中元素的比较次数、交换（移动）次数，以及排序过程所消耗的时间（以 s 或 ms 为单位）。

实验七 图的遍历

题目一 深度优先遍历

【问题描述】

邻接矩阵和邻接表是图（网）的两种基本存储结构。

深度优先遍历图是图结构上的一种基本运算，其主要操作为查找顶点的邻接顶点。对于 n 个顶点、 e 条边的图，若图采用邻接矩阵存储，则遍历运算的时间复杂度为 $O(n^2)$ ；若采用邻接表存储，则遍历运算的时间复杂度为 $O(n + e)$ 。

【基本要求】

从键盘输入图的信息，包括顶点数、边数以及各条边。

1. 用邻接矩阵存储图，输出深度优先遍历的顶点序列；
2. 用邻接表存储图，输出深度优先遍历的顶点序列。

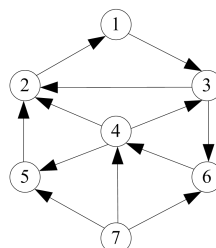
【测试数据】

可采用多种方式输入数据。

1. 输入数据的第一行有两个整数，分别表示顶点数 n 和边数 e ，接下来 n 行、每行 n 个整数表示图的邻接矩阵。（顶点编号从 1 开始）

例 1

```
7 12
0 0 1 0 0 0 0
1 0 0 0 0 0 0
0 1 0 0 0 1 0
0 1 1 0 1 0 0
0 1 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 1 1 1 0
```

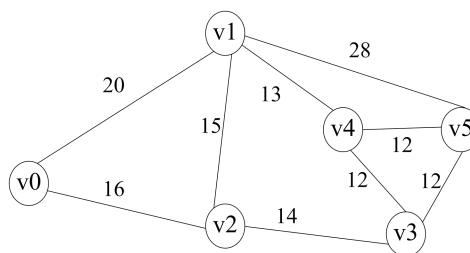


输出如下所示：1 3 2 6 4 5 7

2. 输入数据的第一行有两个整数，分别表示顶点数 n 和边数 e ，接下来分 e 行、每行 3 个整数的方式给出图中的边的信息，每行上的三个整数分别表示一条边关联的第一个顶点（起始顶点）、第二个顶点（终止顶点）、边上的权值。

例 2

```
6 9
0 1 20
0 2 16
1 4 13
1 2 15
1 5 28
5 4 12
4 3 12
2 3 14
5 3 12
```



邻接矩阵存储类型定义：

```
#define MaxVnum 50 // 图的顶点数上限
typedef int AdjMatrix[MaxVnum][MaxVnum]; // 可根据权值的类型将 int 替换为 double
typedef struct { // 图的类型定义
    int vexnum, arcnum; // 图的实际顶点数、边数
    AdjMatrix arcs; // 邻接矩阵
}Graph;
```

邻接表存储类型定义：

```
#define MaxVnum 50 // 图的顶点数上限
typedef char VertexType; // 如果结点的名字较长，可在 VertexType 后面加上[128]
typedef struct ArcNode{ // 表结点
    int adjvex;
```

```

        double weight;
        struct ArcNode *nextarc;
    }ArcNode;

typedef struct {                // 头结点
    VertexType data;
    ArcNode *firstarc;
}AdjList[MaxVnum];

typedef struct {                // 图的类型定义
    int vexnum,arcnum;          // 实际的顶点数、边数
    AdjList vertices;
}Graph;

```

题目二 广度优先遍历

【问题描述】

广度优先遍历图是图结构上的一种基本运算，与深度优先遍历图相同，其主要操作为查找顶点的邻接顶点。

【基本要求】

从键盘输入图的信息，包括顶点数、边数以及各条边。

1. 用邻接矩阵存储图，输出广度优先遍历的顶点序列；
2. 用邻接表存储图，输出广度优先遍历的顶点序列。

【测试数据】

同上。

附录 A 实验报告模板

数据结构实验报告

学号-姓名		实验时间	年 月 日
诚信声明	手写，签名		
实验题目与实验目的（或要求）			
实验过程中遇到的主要问题			
实验小结			
数据结构 （自定义数据类型）	给出自定义数据类型即可		
主要算法 （或算法说明）	可用多种方式描述，常见的有： 自然语言； 程序流程图； 类程序语言代码（包含注释）； **注意：不要仅仅展示源程序代码		

附录 B 实验报告示例

例 1:

数据结构实验报告

学号-姓名	13091*** - ***	实验时间	2010 年 3 月 25 日
诚信声明	本实验及实验报告所写内容为本人所作，没有抄袭。 签 名		
实验题目与实验目的（或要求）	熟悉链表存储结构及运算。 题目 2：约瑟夫环问题：利用单向循环链表存储结构模拟此过程，按照出列的顺序印出个人的编号。 题目 3：一元多项式相加、相减		

实验过程中遇到的主要问题	<ol style="list-style-type: none"> 1. While (p 或 p->next) 语句中的两条件易混淆。 2. 循环单链表的创建不熟悉。 3. 空表与非空表的界限分辨不清。 4. 一元多项式的相加中没有充分利用已有的结点,使得代码过于复杂。 5. 对链表的操作细节没有充分理解。 6. 对链表没有形成完整的知识体系。
实验小结	<p>本次上机练习单链表的算法的实现。主要练习单链表和循环点链表的创建、插入、删除、遍历等。</p> <p>当然,本次上机也练习用单链表的数据结构来解决一些数学问题。</p> <p>线性表是一种很重要的数据结构,它包含了很多算法,这些算法很多都是一些基础性的算法,对学习后面的内容有很大的作用。通过这次上机,我对线性表有了很大了解,加深了对指针、结构体的理解,对数据结构这门课有了初步了解。通过实验我体会到了数据结构这门课的实用性以及数据结构与上学期学的 C 语言有很大不同。对于上面所说的目前还存在的主要问题,我需要深化对线性表的理解,需要调整自己的学习思维,加强逻辑思考能力,还需要进一步修改调试优化代码,学习更多优秀的算法,进一步丰富自己的知识。</p>
数据结构 (自定义数据类型)	<p>题目2:</p> <pre>typedef int Status; typedef struct node{ int code; int cipher; struct node *next; }Node,*Linklist;</pre> <p>题目 3:</p> <pre>typedef struct Node{ int c;//系数 int e;//指数 struct Node *next; }Node,*Linklist;</pre>
主要算法 (或算法说明)	<p>题目2:</p> <p>先用尾插法创建一个结点数为 n 的单循环链表 (create_linklist), 返回值为尾结点的指针; 再定义一个删除函数来实现游戏过程 (delete_linklist), 然后定义一个输出链表中结点信息的函数 (print_linklist)。然后用主函数调用这 2 个函数。</p> <pre>Status Delete_linklist(Linklist tail, unsigned int n, unsigned int m) {//tail 是循环链表表尾指针, n 是游戏人数, m 是初始密码 if (n<1) return ERROR; k = 1; // 用于游戏过程中计数, 每次都从 1 开始</pre>

	<pre> m = m % n ? m % n : n; // 消除重复计数 p = tail -> next; //p 指向所计数的结点 pre = tail; // pre 总是指向 p 的前驱结点 while (n > 1) { // 圈中多于 1 个人时继续游戏 if (k == m) { // 数到需要出圈的人（结点）时进行删除处理*/ // p 指向的结点为需要删除的结点 printf(p->No); //输出出圈者的编码 pre -> next = p -> next; //删除 p 所指结点 n--; //游戏人数减 1 m = p -> Pwd % n ? p -> Pwd % n : n; //取新的游戏密码 free(p); p = pre -> next; //继续游戏，重新开始计数 k = 1; } else { //未数到出圈者时继续计数 k++; pre = p; p = p->next; } } /*while*/ printf(p->No); //输出最后一个出圈者的编号 free(p); return OK; } /*playing*/ </pre> <p>题目 3: 定义一个建表函数，一个加法函数，一个减法函数，一个输出函数； 然后两次调用建表函数建立多项式单链表，调用加减法函数创建结果链表，再调用输出函数输出结果。</p>
--	--

例 2:

数据结构实验报告

学号-姓名	13091*** - ***	实验时间	2010 年 3 月 25 日
诚信声明	本实验及实验报告所写内容为本人所作, 没有抄袭。 签 名		
实验题目与实验目的（或要求）	题目 3：一元多项式相加减运算 输入并建立多项式的单向链表存储； 输出多项式； 多项式相加减。		
实验过程中遇到的主要问题	1. 目前所写的代码没有实现异常处理的情况。 2. 输入函数 scanf 的使用出了问题, 千万注意&；		

	3. 循环部分的循环次数存在问题； 4. 注意单链表插入和删除时指针的变动； 5. 函数的参数是否应该使用引用（引用是第一次接触，应特别注意）； 6. 不熟悉函数 malloc 的使用方式；
实验小结	<p>本次上机主要是单链表的学习和使用。单链表是动态存储，比使用数组方便，节省空间。要掌握单链表的存储，删除，添加等的方法。</p> <p>同一个问题有很多种解决的方法，这时就要考虑算法问题，尽量使用最优算法，减少运算次数，提高程序的执行速度，为以后开发大型软件打好基础。</p>
数据结构 (自定义数据类型)	<pre>typedef struct term{ float coef; int expn; struct term*next; }term,*Linklist;</pre>
主要算法 (或算法说明)	<p>1. int CreatPolyn(Linklist&L,int m) L 是含有头结点的链表头指针，用头插法创建 m 项多项式链表表示。</p> <p>2. int PrintPolyn(Linklist L) L 为含头结点的链表的头指针，先遍历链表算出多项式的项数，然后输出多项式的项数和各项的系数、指数。</p> <p>3. int Polyn(Linklist a,Linklist b,Linklist&c) 多项式用含有头结点的单链表表示，a 是表示第一个多项式的链表的头指针，b 是表示第二个多项式的链表的头指针，将两个多项式相加，结果链表的头指针为 c，同时销毁链表 a 和 b。</p> <p>4. 测试函数：main() 在 main 函数中分别输入两个多项式的项数，首先创建头结点，然后调用 CreatPolyn 创建多项式的链表表示，最后调用 Polyn 将两个多项式相加，并输出和多项式。</p> <p>5. 函数间的调用关系</p> <pre> graph TD main((main)) --> CP1((CreatPolyn)) main --> CP2((CreatPolyn)) main --> P((Polyn)) CP1 --> PP1((PrintPolyn)) CP2 --> PP2((PrintPolyn)) P --> PP3((PrintPolyn)) </pre>