

# 数据结构课程设计指导书

计算机科学系

# 目 录

|  |    |
|--|----|
| 案例 1 基于链表的学生成绩管理系统.....                        | 1  |
| 1. 1 简介.....                                   | 1  |
| 1. 2 数据结构的定义.....                              | 1  |
| 1. 3 算法的实现步骤.....                              | 1  |
| 1. 4 程序的运行界面.....                              | 4  |
| 案例 2 基于哈夫曼树的文件压缩软件.....                        | 11 |
| 2. 1 简介.....                                   | 11 |
| 2. 2 数据结构的定义.....                              | 12 |
| 2. 3. 算法实现步骤.....                              | 13 |
| 2. 4. 程序的运行界面.....                             | 15 |
| 案例 3 家族树的创建与显示.....                            | 17 |
| 3. 1 简介 .....                                  | 17 |
| 3. 2 数据结构的定义.....                              | 17 |
| 3. 3 算法实现步骤.....                               | 17 |
| 3. 4 程序的运行界面.....                              | 20 |
| 案例 4 校园地图查询与导引软件.....                          | 25 |
| 4. 1 简介 .....                                  | 25 |
| 4. 2 数据结构的定义.....                              | 25 |
| 4. 3 算法实现步骤.....                               | 26 |
| 4. 4 程序的运行界面.....                              | 28 |
| 附录 1：如何在 C 语言中读写数据文件.....                      | 34 |
| 附录 2：如何在 Visual Studio 6.0 环境下创建和调试 C++程序..... | 37 |

# 案例 1 基于链表的学生成绩管理系统

## 1.1 简介

在有数据库的情况下，将数据存到数据库中是一个很好的选择。但是，在没有数据库的情况下，我们也可以用链表存储数据，并完成增加、删除、修改、查询等数据管理功能。本案例就是开发一个基于链表的学生信息管理系统。在这个系统中，学生信息保存在链表中，可以增加、删除、修改、查询，也可以把链表中的数据保存到一个硬盘文件中，便于下次运行程序时读取。此外，还要求对学生的学号和姓名建立索引，然后根据索引快速查找需要的信息，根据索引以数据递增的次序显示各个记录。这不但复习巩固了折半查找算法，而且对索引有了更直观的认识。

## 1.2 数据结构的定义

```
typedef struct studentStru
{
    int id;//学号
    char name[30];//姓名
    float score[3];//成绩
    struct studentStru *next;//指向下一个结点的指针
}StudentNode, *StudentLink;
typedef struct STUStru
{
    StudentLink Head; //指向表头的指针
    StudentLink Tail; //指向表尾的指针
    int count; //已保存的记录的数量
}STU;
StudentLink *IdIndex; //学号索引数组的起始地址
StudentLink *NameIndex; //姓名索引数组的起始地址
```

## 1.3 算法的实现步骤

### 1.3.1 链表的相关算法

```
//获得链表的第 i 个元素
Status GetElem(LinkList L, int i, ElemType &e)
{ /* L为带头结点的单链表的头指针。当第 i 个元素存在时，其值赋给 e 并返回OK，否则返回
ERROR */
    int j=1; /* j为计数器*/
    LinkList p=L->next; /* p指向第一个结点*/
    while(p&&j<i) /* 顺指针向后查找，直到p指向第 i 个元素或p为空*/
        p=p->next;
    if(j==i)
        e=p->data;
    else
        e=ERROR;
    return OK;
}
```

```

{
    p=p->next;
    j++;
}
if(!p||j>i) /* 第i个元素不存在*/
    return ERROR;
e=p->data; /* 取第i个元素*/
return OK;
}

//在链表中查找元素e
int LocateElem(LinkList L, ElemType e)
{
    int i=0;
    LinkList p=L->next;
    while(p)
    {
        i++;
        if(p->data==e) /* 找到这样的数据元素*/
            return i;
        p=p->next;
    }
    return 0;
}

// 在带头结点的单链线性表L中第i个位置之前插入元素e
Status ListInsert(LinkList L, int i, ElemType e)
{
    int j=0;
    LinkList p=L, s;
    while(p&&j<i-1) /* 寻找第i-1个结点*/
    {
        p=p->next;
        j++;
    }
    if(!p||j>i-1) /* i小于或者大于表长*/
        return ERROR;
    s=(LinkList)malloc(sizeof(struct LNode));
    s->data=e; /* 插入L中*/
    s->next=p->next;
    p->next=s;
    return OK;
}

//在带头结点的单链线性表L中，删除第i个元素，并由e返回其值
Status ListDelete(LinkList L, int i, ElemType &e)

```

```

{
    int j=0;
    LinkList p=L, q;
    while(p->next&&j<i-1) /* 寻找第i个结点, 并令p指向其前趋*/
    {
        p=p->next;
        j++;
    }
    if(!p->next||j>i-1) /* 删除位置不合理*/
        return ERROR;
    q=p->next; /* 删除并释放结点*/
    p->next=q->next;
    e=q->data;
    free(q);
    return OK;
}

//链表的遍历, 即依次打印输出链表的各个数据元素
Status ListTraverse(LinkList L)
{
    LinkList p=L->next;
    while(p)
    {
        printf("%d ", p->data);
        p=p->next;
    }
    printf("\n");
    return OK;
}

```

### 1.3.2 建立索引的原理

建立索引, 是指不对记录直接排序, 但是根据记录的某个字段对记录的编号进行排序并保存在索引中, 以便在查找记录时能够快速地找到记录的一种方法。

| 内存地址 | 学号    | 姓名     | 成绩  | next 指针 | 学号索引 | 姓名索引 |
|------|-------|--------|-----|---------|------|------|
| 2600 | 10001 | 王强强    | 100 | 3200    | 0    | 2600 |
| 3200 | 10008 | WangH  | 89  | 2580    | 1    | 2580 |
| 2580 | 10005 | ZhaoX  | 92  | 3620    | 2    | 3200 |
| 3620 | 10031 | QianM  | 73  | 4008    | 3    | 5200 |
| 4008 | 10018 | JiangV | 94  | 2100    | 4    | 2100 |
| 2100 | 10013 | WanH   | 69  | 8660    | 5    | 4008 |
| 8660 | 10019 | TianR  | 85  | 5200    | 6    | 8660 |
| 5200 | 10010 | ZuoU   | 77  | 0       | 7    | 3620 |

图 1-1 建立索引的原理

由于与索引对应的字段是递增排列的，所以，在索引上查找数据可以用折半查找的方法。折半查找的算法如下：

```
int Search_Bin ( SSTable ST, KeyType key ) {  
    low = 1;  high = ST.length;      // 置区间初值  
    while (low <= high) {  
        mid = (low + high) / 2;  
        if  (key == ST.elem[mid].key)  
            return mid;           // 找到待查元素  
        else  if (key < ST.elem[mid].key)  
            high = mid - 1;       // 继续在前半区间进行查找  
        else  low = mid + 1;     // 继续在后半区间进行查找  
    }  
    return 0;    // 顺序表中不存在待查元素  
}
```

## 1.4 程序的运行界面

主菜单

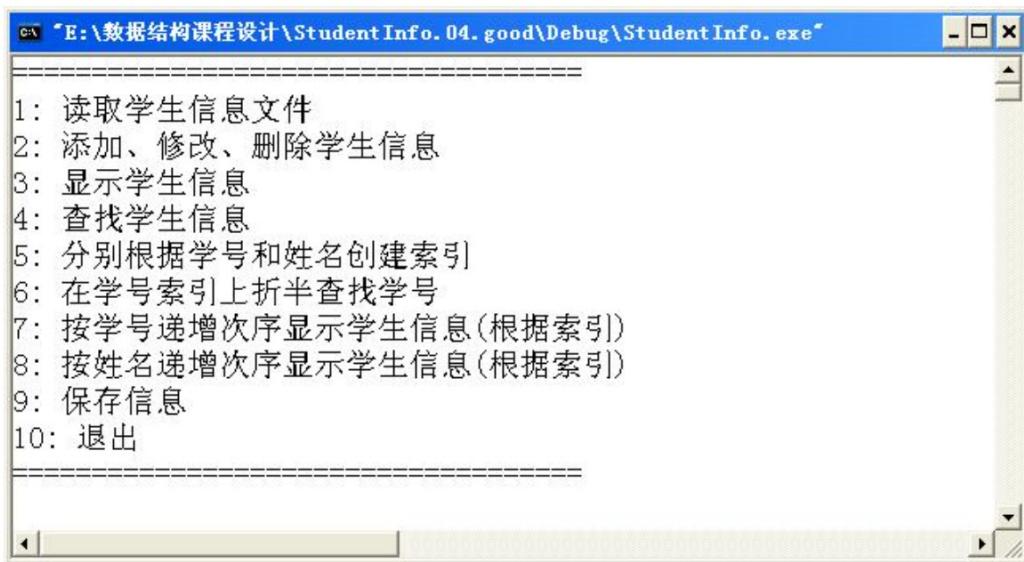


图 1-2 成绩主菜单

添加学生信息

```
CA "E:\数据结构课程设计\Student Info. 04. good\Debug\StudentInfo.exe"
10: 退出
=====
2
a: 添加学生信息
b: 删除学生信息
c: 修改学生姓名
d: 修改学生分数
a
请输入学号, 姓名和分数(学号=-1时退出):
10001 王刚 78
请输入学号, 姓名和分数(学号=-1时退出):
10002 孙小美 82
请输入学号, 姓名和分数(学号=-1时退出):
10008 WangH 89
请输入学号, 姓名和分数(学号=-1时退出):
10005 ZhaoX 92
请输入学号, 姓名和分数(学号=-1时退出):
-1 0 0
=====
1: 读取学生信息文件
```

图 1-3 添加学生信息界面

### 显示学生信息

```
CA "E:\数据结构课程设计\Student Info. 04. good\Debug\StudentInfo.exe"
10: 退出
=====
3
#####
#_#_#_# 学号:10001    姓名:王刚    分数: 78
#_#_#_# 学号:10002    姓名:孙小美   分数: 82
#_#_#_# 学号:10008    姓名:WangH    分数: 89
#_#_#_# 学号:10005    姓名:ZhaoX    分数: 92
#####
1: 读取学生信息文件
```

图 1-4 显示学生信息界面

### 删除学生信息

```
01 "E:\数据结构课程设计\Student Info. 04. good\Debug\Student Info. exe" -口x
10: 退出
=====
2
a: 添加学生信息
b: 删除学生信息
c: 修改学生姓名
d: 修改学生分数
b
请输入学号:
10002
=====
1: 读取学生信息文件
```

图 1-5 删除学生信息界面

修改学生姓名

```
01 "E:\数据结构课程设计\Student Info. 04. good\Debug\Student... -口x
10: 退出
=====
2
a: 添加学生信息
b: 删除学生信息
c: 修改学生姓名
d: 修改学生分数
c
请输入学号和姓名:
10001 王强强
=====
1: 读取学生信息文件
```

图 1-6 修改学生姓名界面

修改学生分数

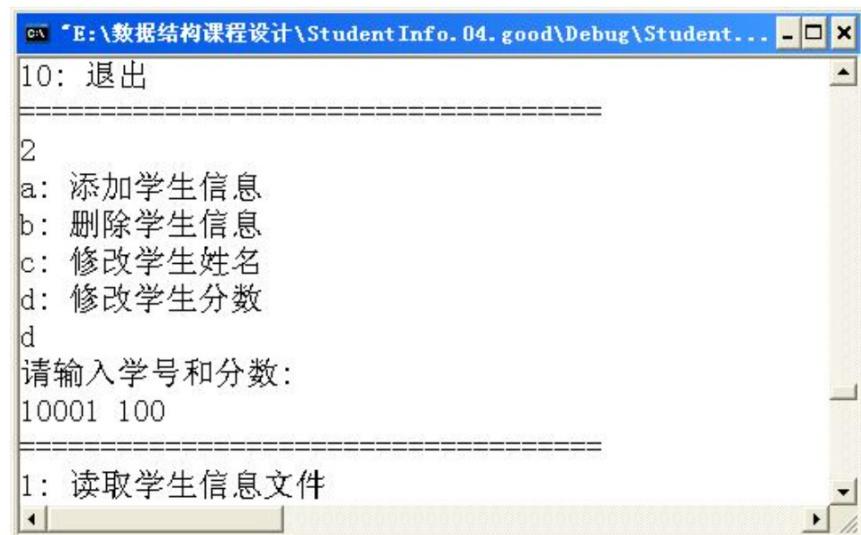


图 1-7 修改学生分数界面

查找学生信息

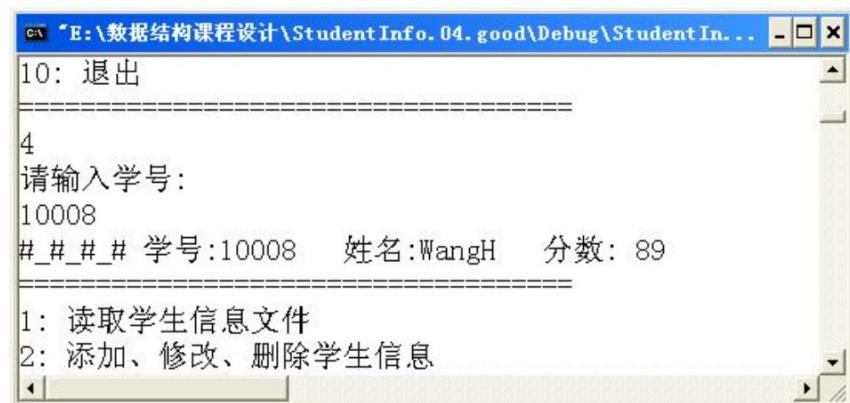


图 1-8 查找学生信息界面

假设已经保存的学生信息包括：

10001 王强强 100  
10008 WangH 89  
10005 ZhaoX 92  
10031 QianM 73  
10018 JiangV 94  
10013 WanH 69  
10019 TianR 85  
10010 ZuoU 77

在学号索引上折半查找学号的结果为：

```
E:\数据结构课程设计\Student Info. 04. good\Debug\Student Info... 10: 退出
=====
6
请输入学号:
10013
折半查找过程为:
10010 10018 10013
已找到, 共做了3次比较
=====
1: 读取学生信息文件
```

图 1-9 在学号索引上折半查找学号的结果

按学号递增次序显示学生信息(根据索引)

```
E:\数据结构课程设计\Student Info. 04. good\Debug\Student... 10: 退出
=====
7
#_#_#_# 学号:10001 姓名:王强强 分数:100
#_#_#_# 学号:10005 姓名:ZhaoX 分数: 92
#_#_#_# 学号:10008 姓名:WangH 分数: 89
#_#_#_# 学号:10010 姓名:ZuoU 分数: 77
#_#_#_# 学号:10013 姓名:WanH 分数: 69
#_#_#_# 学号:10018 姓名:JiangV 分数: 94
#_#_#_# 学号:10019 姓名:TianR 分数: 85
#_#_#_# 学号:10031 姓名:QianM 分数: 73
=====
1: 读取学生信息文件
```

图 1-10 按学号递增次序显示学生信息

按姓名递增次序显示学生信息(根据索引)

```
CD "E:\数据结构课程设计\StudentInfo_04.good\Debug\Student..." -口x  
10: 退出  
=====  
8  
#_#_#_# 学号:10018 姓名:JiangV 分数: 94  
#_#_#_# 学号:10031 姓名:QianM 分数: 73  
#_#_#_# 学号:10019 姓名:TianR 分数: 85  
#_#_#_# 学号:10013 姓名:WanH 分数: 69  
#_#_#_# 学号:10008 姓名:WangH 分数: 89  
#_#_#_# 学号:10005 姓名:ZhaoX 分数: 92  
#_#_#_# 学号:10010 姓名:ZuoU 分数: 77  
#_#_#_# 学号:10001 姓名:王强强 分数:100  
=====  
1: 读取学生信息文件
```

图 1-11 按姓名递增次序显示学生信息

在运行“保存信息”菜单后，得到的数据文件如下：

```
CD "E:\数据结构课程设计\StudentInfo_04.good\Debug\St... -口x  
10: 退出  
=====  
9  
请输入文件名(包括路径)  
c:\student.txt  
=====  
1: 读取学生信息文件
```

图 1-12 “保存信息”菜单界面

```
student.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
8  
10001 王强强 100.000000  
10008 WangH 89.000000  
10005 ZhaoX 92.000000  
10031 QianM 73.000000  
10018 JiangV 94.000000  
10013 WanH 69.000000  
10019 TianR 85.000000  
10010 ZuoU 77.000000
```

图 1-13 “保存信息”后得到的文件

在这个数据文件中，第一行的 8 代表总共有 8 条学生信息记录，后面的 8 行依次是各个学生的学号、姓名和成绩。

读取学生信息文件

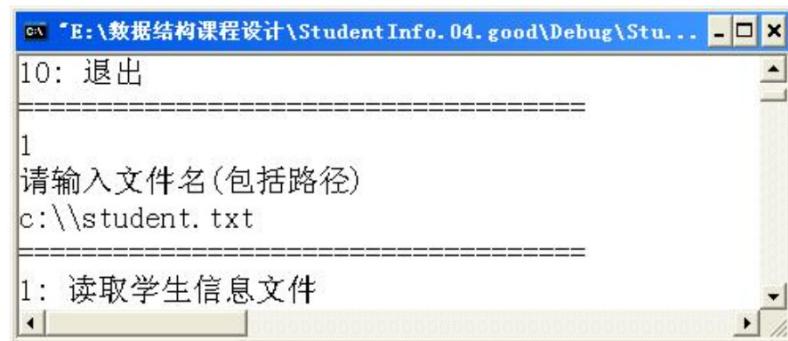


图 1-14 读取学生信息文件菜单

# 案例 2 基于哈夫曼树的文件压缩软件

## 2.1 简介

数据压缩分为两大类：无损压缩和有损压缩。无损压缩是指压缩后还能还原回原来的数据的压缩方法，有损压缩一般是针对图像、视频或音频进行的压缩，在压缩后图像、视频或音频的质量会有所下降，这种压缩一般不能再还原回原来的数据。本次课程设计主要研究无损压缩，不但设计压缩算法，还要设计解压缩算法。

数据压缩有许多不同的方法，其中一种重要的方法是利用了哈夫曼编码。用哈夫曼编码压缩的数据可以节约 20% 到 85% 的空间储蓄。这种算法是根据它的发明者 David Huffman 的名字命名的，他是一位信息理论学家和计算机科学家，他在 20 世纪 50 年代发明了此项技术。当压缩数据的时候，通常会把组成数据的字符转换成一些其他表现以节省空间。一种典型的压缩方法是把每个字符转换成二进制字符代码或者位字符串。例如，把字符 “a” 编码成 000，字符 “b” 编码成 001，而字符 “c” 则变成 010，如此等等。这种方法被称为是固定长度编码。

但是，还可以用一种更好的方法，就是使用可变长度编码。既然某些字符会多次用到，所以这些最频繁出现在字符串内的字符具有较短的编码，而具有较低出现频率的字符则拥有较长的编码。编码的过程就是依据某个字符的出现频率把位字符串赋值给该字符的过程。哈夫曼编码算法每次将最小的两个子树树根作为两个孩子重新生成一棵树，树根的权值为两个孩子的权值之和，重复这一过程，直至只有一个树根为止。把到达每个左孩子节点的路径设置为二进制字符 0，而把到达每个右孩子节点的路径设置为二进制字符 1。

例如：设存入 10000 个字符，出现的频率如下：

|        |        |
|--------|--------|
| A: 25% | B: 20% |
| C: 15% | D: 10% |
| E: 10% | F: 10% |
| G: 5%  | H: 5%  |

构造树，如下图所示，根据该哈夫曼树，得到各个字母的编码如下：

|          |
|----------|
| A: 01    |
| B: 11    |
| C: 001   |
| D: 100   |
| E: 101   |
| F: 0001  |
| G: 00000 |
| H: 00001 |

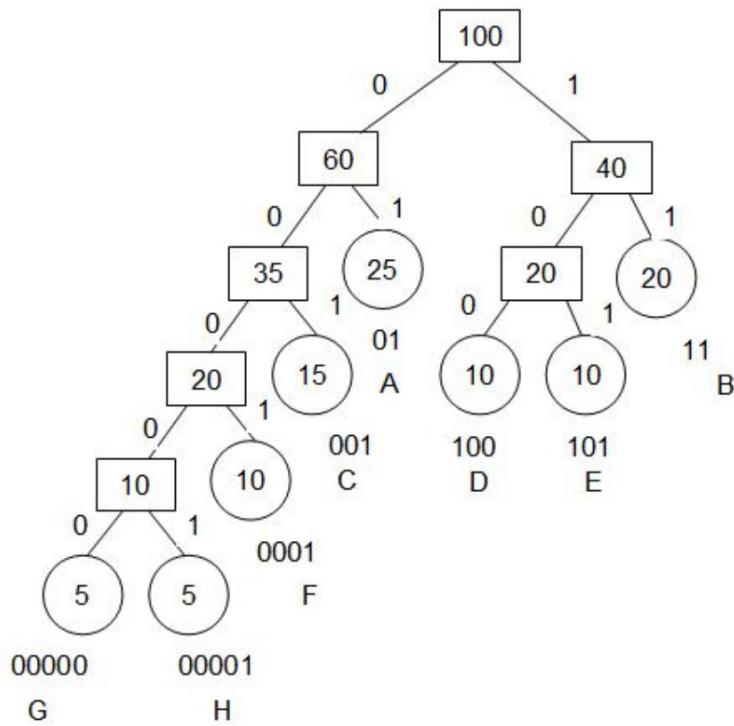


图 2-1 最优二叉树的一个例子

压缩后，总共需要的二进制位个数为：

$$\begin{aligned}
 & 2500*2 + 2000*2 + 1500*3 + 1000*3 + 1000*3 + 1000*4 + 500*5 + 500*5 \\
 & = 28500
 \end{aligned}$$

它的字节数为  $28500/8=3562$ ，与压缩前相比，长度只有原来的三分之一。

## 2.2 数据结构的定义

### 2.2.1 内存中的存储结构定义

```

typedef struct
{
    char ch; //字符名称
    int weight; //权值(字符出现的次数)
}charweight;

typedef struct hftnode
{
    int weight; //权值
    int parent, lchild, rchild; //双亲、左孩子、有孩子的存储位置(数组下标)
}HFMNode;

typedef struct hc
{
    char ch; //字符名称

```

```

int start; //字符编码存储的起始为止，即从数组从 start 到 299 的各个字符为编码
char link[300];//用来存放 ch 的编码的字符数组
}HFMCode;

//哈夫曼树的叶子节点个数，即被压缩文件出现的不同字符数。
int HFMCodeLength;

charweight flist[256]; //存放被压缩文件各个字符出现的次数
HFMCode hfmcode[256]; //存放各个字符的 01 哈夫曼编码

```

### 2.2.2 压缩文件的结构定义

按照存储的先后次序，压缩文件的内容依次包括：

- (1) 压缩文件内部标识'Y' 和' S' (char 型， 2 字节)
- (2) 树叶结点个数(short int 型， 2 字节)
- (3) 各个树叶结点的 Huffman 编码，每个树叶结点的编码按上述格式存储：
  - a) 树叶字符  $L_i$  的名称(char 型， 1 字节)
  - b) 树叶字符  $L_i$  的编码 01 二进制位个数(unsigned char 型， 1 字节)
  - c) 树叶字符  $L_i$  的编码长度  $N_{i2}$ (unsigned char 型， 1 字节)
  - d) 树叶字符  $L_i$  的编码(char 型，  $N_i$  字节)
- (4) 8 位对齐补偿长度
- (5) 原始文件的 Huffman 编码长度  $N_s$ (int 类型， 4 字节)
- (6) 原始文件的 Huffman 编码(unsigned char 型，  $N_s$  字节)

## 2.3. 算法实现步骤

### 2.3.1 压缩算法实现步骤

- (1) 读取原始文件，统计各个字符出现的次数，把各个字符的名称和出现的次数放在结构体数据组 flist 中。把不同的字符数保存在 HFMCodeLength 中。
- (2) 把原始文件中出现的字符保存在 hfmcode[i].ch 中，把 hfmcode[i].start 置初值 299，把 hfmcode[i].link 数组初始化为全' 0' 的数组。
- (3) 用上述的哈夫曼树的算法，根据 flist 数组创建哈夫曼树，创建后的哈夫曼树保存在数组 hfmmtree 中。用上述的建哈夫曼编码的算法，创建哈夫曼编码，创建后的编码保存在数组 hfmcode 中。
- (4) 读取原始文件的各个字符，对每个字符
  - a) 在 hfmcode 中查找与之对应的哈夫曼编码
  - b) 把哈夫曼编码依次保存到字符数组 p 中，p 的每个字符，要么是' 0'，要么是' 1'。
- (5) 假设数组 p 中总共存储了 k 个字符，再增加 len\_duiqi 个字符，使字符总数为 8 的整数倍。
- (6) 创建压缩文件，将' Y' 和' S' 写入压缩文件，作为文件内部标识
- (7) 将树叶结点个数 cnum(short int 型)写入压缩文件

下面的(8)(9)(10)(11)循环执行 cnum 次

- (8) 写入树叶字符  $L_i$  的名称(char 型)
- (9) 写入树叶字符  $L_i$  的编码 01 二进制位个数(unsigned char 型)
- (10) 将树叶字符  $L_i$  的编码有 01 字节编码转换为 01 二进制编码, 然后  
写入 01 二进制编码的长度  $N_i$  (unsigned char 型)
- (11) 将转换后的二进制编码写入文件 ( $N_i$  字节)
- (12) 写入 8 位对齐补偿长度 len\_duiqi
- (12) 写入原始文件的 Huffman 编码长度  $N_s$  (int 类型)
- (13) 将数组 p 中的 01 字节编码转换为 01 二进制编码, 将该二进制编码写入文件

### 2.3.2 解压缩算法实现步骤

(1) 从压缩文件读取两个字节的内部标识

(2) 从压缩文件读取树叶结点个数 cnum

下面的(3)(4)(5)(6)循环执行 cnum 次

- (3) 读取树叶字符  $L_i$  的名称(char 型)
- (4) 读取树叶字符  $L_i$  的编码 01 二进制位个数(unsigned char 型)
- (5) 读取 01 二进制编码的长度  $N_i$
- (6) 读取  $N_i$  个字节到数组 hcode 中, 将 hcode 中的 01 二进制编码  
转换为 01 字节编码, 然后保存到 hfmcodes[i].link[] 中
- (7) 读取 8 位对齐补偿长度 len\_duiqi
- (8) 读取 Huffman 编码长度  $N_s$  (int 类型)
- (9) 读取  $N_s$  个字节到数组 str 中, 然后将 str 中的 01 二进制编码转换为 01  
字节编码。
- (10) 舍去 str 中的最后 len\_duiqi 个字符, 得到 01 字节编码的长度 len
- (11) 创建解压缩文件, 然后从前向后依次搜索 hfmcodes 中的各个编码,  
如果能搜到, 则把与之对应的字符写入解压缩文件。

### 2.3.3 算法的伪代码

以下为创建哈夫曼树的伪代码,

```
/*w 存放 n 个字符的权值(均>0), 构造哈夫曼树 HT, 并求出 n 个字符的哈夫曼编码 HC.*/
void HuffmanCoding(HFMNode *HT, HFMCode *HC, int *w, int n)
{
    if(n<=1) return;
    m=2*n-1;
    for(p=HT, i=0; i<n; ++i, ++p, ++w) *p={*w, 0, 0, 0};
    for(; i<m; ++i, ++p) *p={0, 0, 0, 0};
    //建哈夫曼树
    for(i=n+1; i<=m; ++i)
    {
        /*在 HT[1.. i-1]选择 parent 为 0 且 weight 最小的两个结点,
        其序号分别为 s1 和 s2.*/
        select(HT, i-1, s1, s2);
        HT[s1].parent=i; HT[s2].parent=i;
```

```

HT[i].lchild=s1;HT[i].rchild=s2;
HT[i].weight=HT[s1].weight+HT[s2].weight;
}

```

以下为创建哈夫曼编码的伪代码，

```

//-----从叶子到根逆向求每个字符的哈夫曼编码-----
//分配 n 个字符编码的头指针向量

```

```

for (i=0;i<n;++i)//逐个字符求哈夫曼编码
{
    HC[i].start=n-1;           //编码结束符位置
    //从叶子至根逆向求编码
    for (c=i, f=HT[i];f!=0;c=f, f=HT[f].parent)
        if (HT[f].lchild==c)
            HC[i].cd[--start] = "0";
        else
            HC[i].cd[--start] = "1";
}

```

## 2.4. 程序的运行界面

程序的主菜单：



图 2-2 程序的主菜单

压缩文件的运行界面：

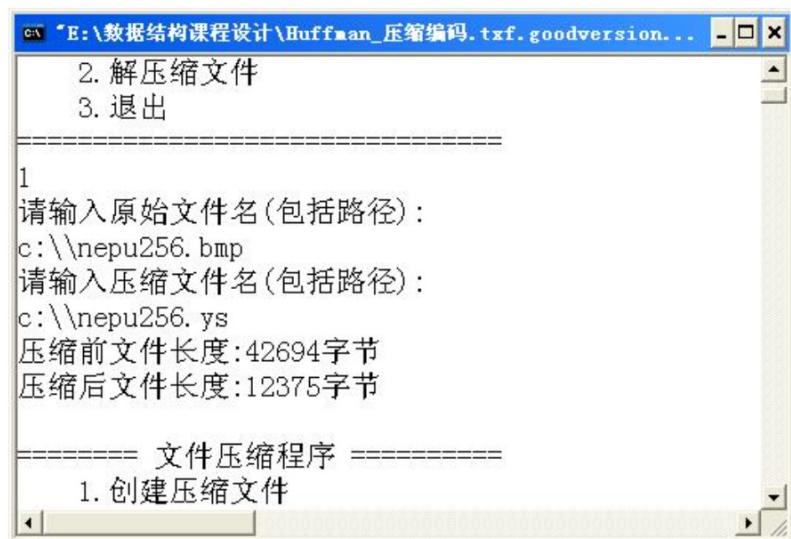


图 2-3 压缩文件的运行界面

解压缩文件的运行界面：

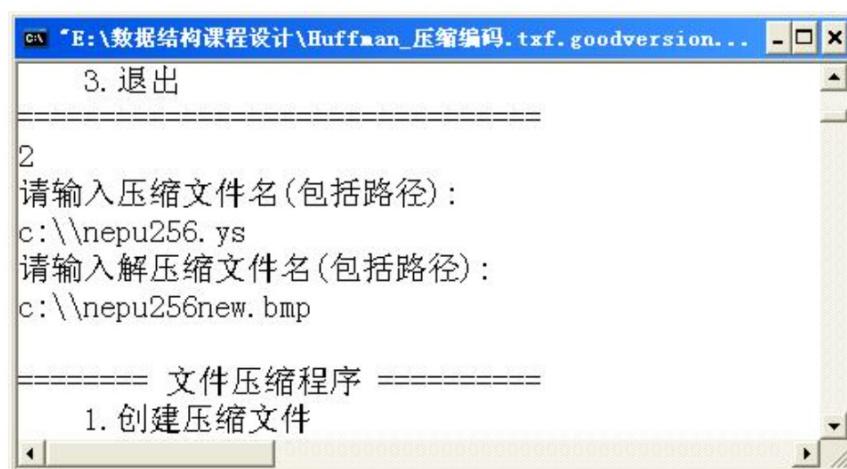


图 2-4 解压缩文件的运行界面

# 案例 3 家族树的创建与显示

## 3.1 简介

计算机或其它显示设备的显示模式可以分为文本模式和图形模式。在某些条件下，显示设备可能只支持文本模式，例如控制台应用程序、只支持文本显示模式的工控机等。在文本模式下显示树是一件比较困难的事情，因为树的层数是可变的，每个结点的孩子数量也都是可变的，而文本显示模式不能显示图形或图像，只能在若干固定的位置显示字符。编写家族树的创建与显示程序，即练习了数据结构中的树的相关算法，也具有一定的实际应用价值。

该程序涉及到多叉树的创建、多叉树的遍历、多叉树的插入、多叉树的删除、递归程序等数据结构相关内容。

## 3.2 数据结构的定义

```
typedef struct TreeStru
{
    char name[50];//结点名称
    struct TreeStru *child[20];//指向各个孩子的指针
    int numberofchild;//孩子的数量
    int level;//当前结点的层号
    int width;//以当前结点为根的子树的宽度
    int absolutePosition;//层内绝对开始位置，用于画竖线和输出结点
    int relativePosition;//相对于父亲的起始位置，用于画横线
}TreeNode,*Tree;
```

## 3.3 算法实现步骤

### 3.3.1 将结点添加到树中的算法

```
int AddToTree(Tree &T, char fname[50],char cname[50])
//T 为树根， fname 为父亲的名字， cname 为儿子的名字
{
    Tree childT;
    if(T==NULL)//如果目前是空树，则以 fname 为根，以 cname 为儿子创建树
    {
        T=(Tree)malloc(sizeof(TreeNode));//T 指向树根
```

```

T->numberofchild=0;
strcpy(T->name,fname);
childT=(Tree)malloc(sizeof(TreeNode));//childT 指向孩子
childT->numberofchild=0;
strcpy(childT->name,cname);
T->child[T->numberofchild++]=childT;
return 1;
}
else //目前树不空
{
    Tree k,s;
    int flag;
    k=NULL;
    SearchTree(T,fname,k);//在树中搜索 fname, 如果能搜到, 让 k 指向该结点
    if(k==NULL)
        return 2;
    flag=SearchChild(k,cname); //在 k 的各个孩子中搜索 cname
    if(flag==1) //k 的孩子中已有 cname, 不需要再添加
        return 3;
    s=(Tree)malloc(sizeof(TreeNode));
    s->numberofchild=0;
    strcpy(s->name,cname);
    k->child[k->numberofchild++]=s;
    return 1;
}
return 0;
}

```

### 3.3.2 在树中搜索一个结点的算法

```

void SearchTree(Tree T, char fname[50], Tree &k)
{
    if(T!=NULL)
    {
        if(strcmp(T->name,fname)==0)
        {
            k=T;
            return;
        }
        int i;
        for(i=0;i<T->numberofchild;i++)
            SearchTree(T->child[i],fname,k);
    }
}

```

### 3.3.3 树的显示

为了在文本模式下显示树，需要预先计算出每个结点的层号、以当前结点为根的子树的宽度、相对于父亲的起始位置和层内绝对开始位置。

例如，对于下面的树，**a,b,c,d,e,f,g,h,i** 的上述参数值如下表所示。

表 3-1 孩子与父亲的位置关系

| 结点名称 | 孩子数量 | 结点的层号 | 以当前结点为根的子树的宽度 | 相对于父亲的起始位置 | 层内绝对开始位置 |
|------|------|-------|---------------|------------|----------|
| a    | 3    | 1     | 6             | 0          | 0        |
| b    | 0    | 2     | 1             | 0          | 0        |
| c    | 2    | 2     | 4             | 1          | 1        |
| d    | 0    | 2     | 1             | 5          | 5        |
| e    | 0    | 3     | 1             | 0          | 1        |
| f    | 3    | 3     | 3             | 1          | 2        |
| g    | 0    | 4     | 1             | 0          | 2        |
| h    | 0    | 4     | 1             | 1          | 3        |
| i    | 0    | 4     | 1             | 2          | 4        |

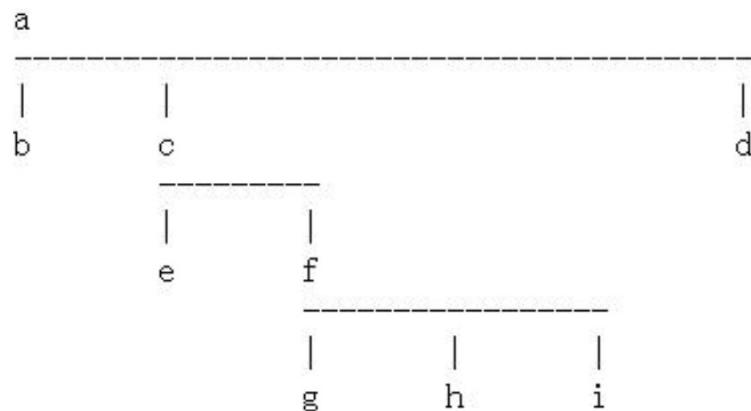


图 3-1 显示的家族树

获得各个结点的层号的算法为：

```

void GetLevel(Tree TreeRoot, int level)
{
    如果 TreeRoot 不空
    {
        TreeRoot->level=level;
        for(i=0;i<TreeRoot->numberofchild;i++)
            GetLevel(TreeRoot->child[i],level+1);
    }
}
  
```

获得树的宽度可以由一个递归程序得到，递归的次序按照后根遍历次序，即先求各个子

树的宽度，各个子树的宽度之和即为当前的树的宽度。

在得到各个子树的宽度后，可以计算各个结点的相对于父亲的起始位置，它可以由递归程序得到，但是要按先根的次序，先指定树根的开始位置为 0，然后计算它的各个孩子的相对于父亲的起始位置，再以这些孩子为根，继续计算它的孩子的相对于父亲的起始位置。

在得到各个结点的相对于父亲的起始位置之后，可以计算各个结点的层内绝对开始位置，它可以由递归程序得到，但是要按先根的次序，先指定树根的绝对开始位置为 0，然后计算它的各个孩子的绝对开始位置，再以这些孩子为根，继续计算它的孩子的绝对开始位置。

在得到各个结点的上述数据后，根据结点的层号可以知道该结点在第几行显示，根据结点的层内绝对开始位置可以知道该结点和'|'在第几列开始显示，根据相对于父亲的起始位置，可以知道向左画多少个'-'。在结点名称、'|'和'-'都显示出来以后，家族树的绘制任务即结束。

### 3.4 程序的运行界面

主菜单

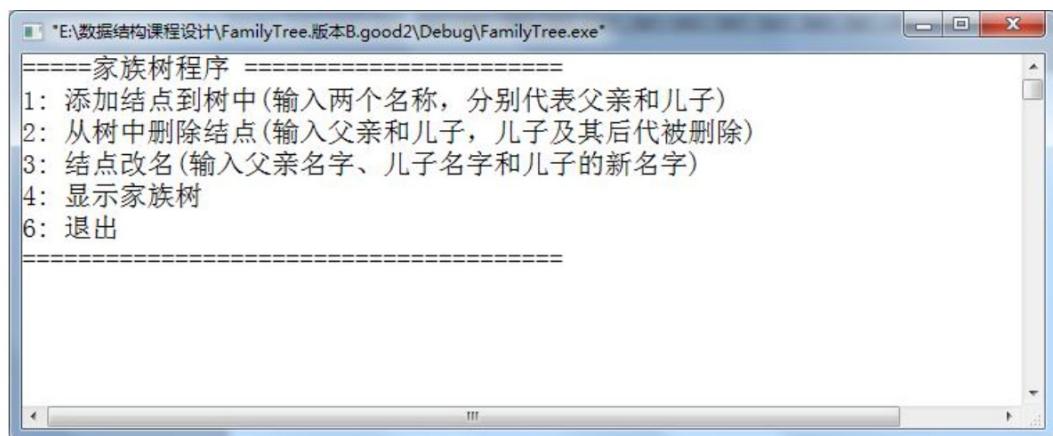


图 3-2 主菜单

菜单：添加结点到树中

```
E:\数据结构课程设计\FamilyTree.版本B.good2\Debug\FamilyTree.exe"
6: 退出
=====
1
input father and son, @ @ for end:
a b1
input father and son, @ @ for end:
a c1
input father and son, @ @ for end:
a d1
input father and son, @ @ for end:
c1 Tom
input father and son, @ @ for end:
@ @
=====家族树程序 =====
1: 添加结点到树中(输入两个名称, 分别代表父亲和儿子)
2: 从树中删除结点(输入父亲和儿子, 儿子及其后代被删除)
3: 结点改名(输入父亲名字、儿子名字和儿子的新名字)
```

图 3-3 菜单：添加结点到树中

菜单：显示家族树

```
E:\数据结构课程设计\FamilyTree.版本B.good2\Debug\FamilyTree.exe"
6: 退出
=====
4
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
a
-----
|   |   |
b1  c1  d1
-
|
Tom

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
the tree is saved to c:\FamilyTree.txt
=====家族树程序 =====
1: 添加结点到树中(输入两个名称, 分别代表父亲和儿子)
```

图 3-4 菜单：显示家族树

继续添加一些结点后，树变为：

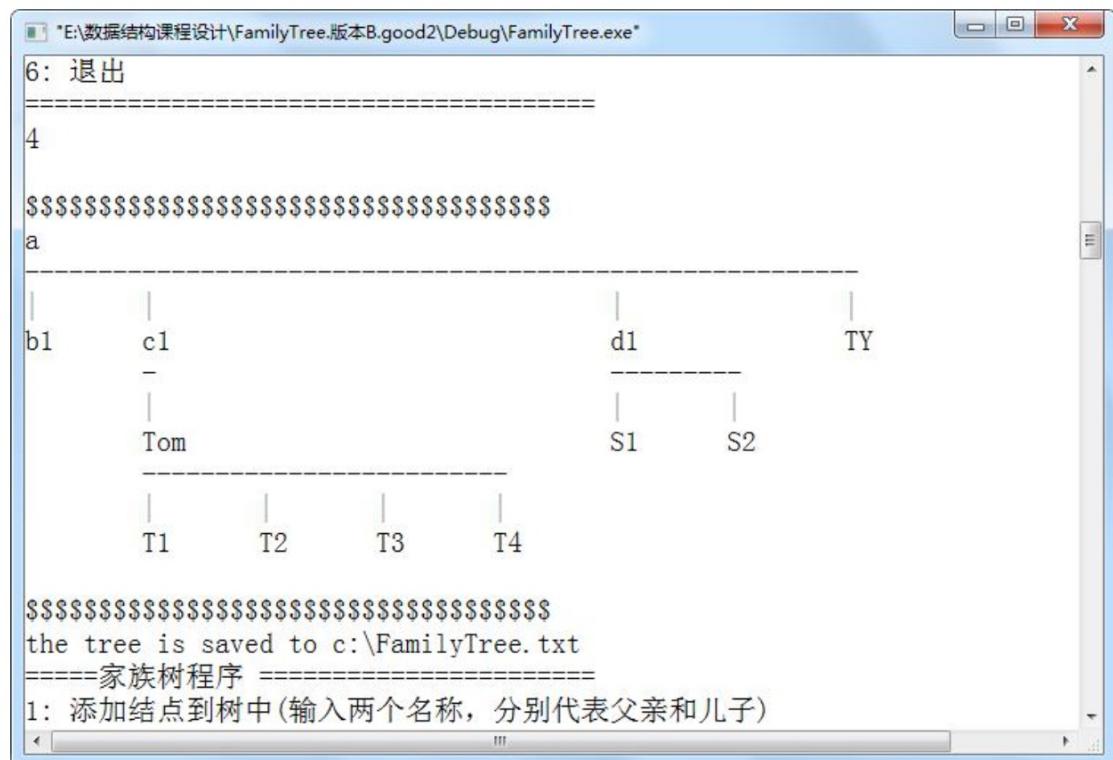


图 3-5 继续添加结点得到的家族树

再继续添加结点，树变为：

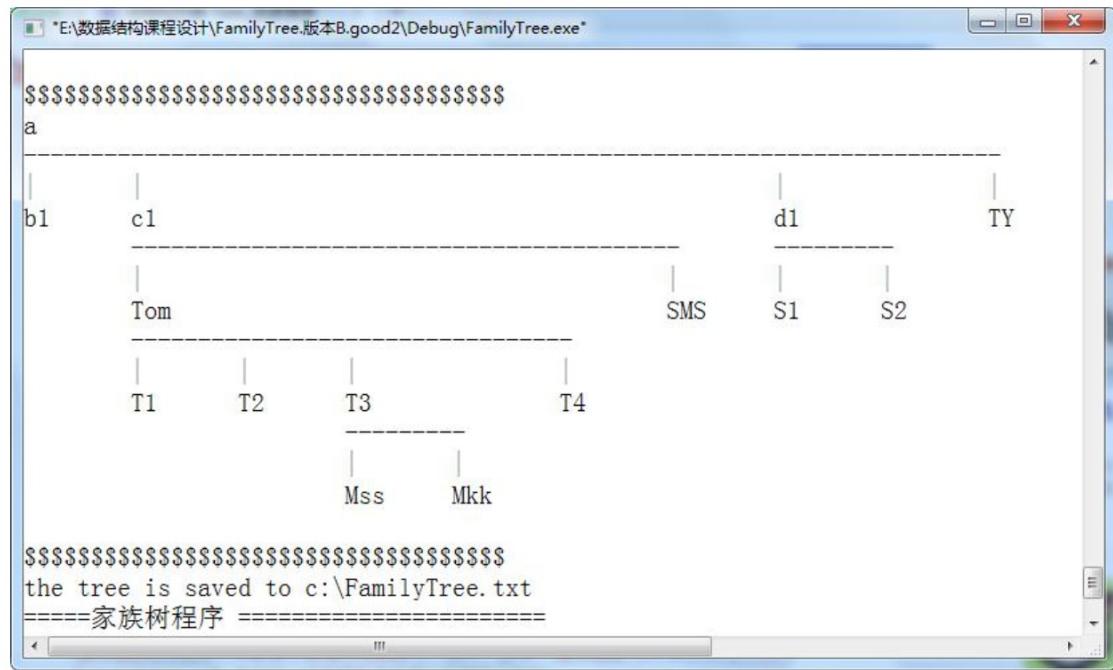


图 3-6 再继续添加结点得到的家族树

菜单：从树中删除结点

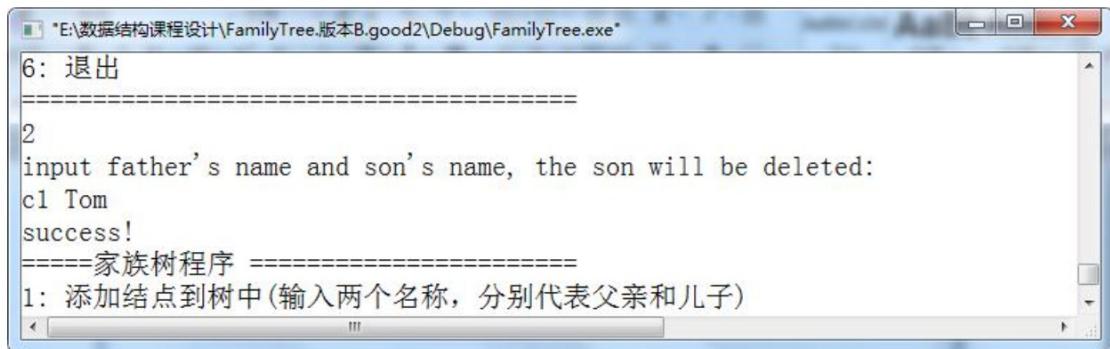


图 3-7 菜单：从树中删除结点

删除 Tom 后，树变为：

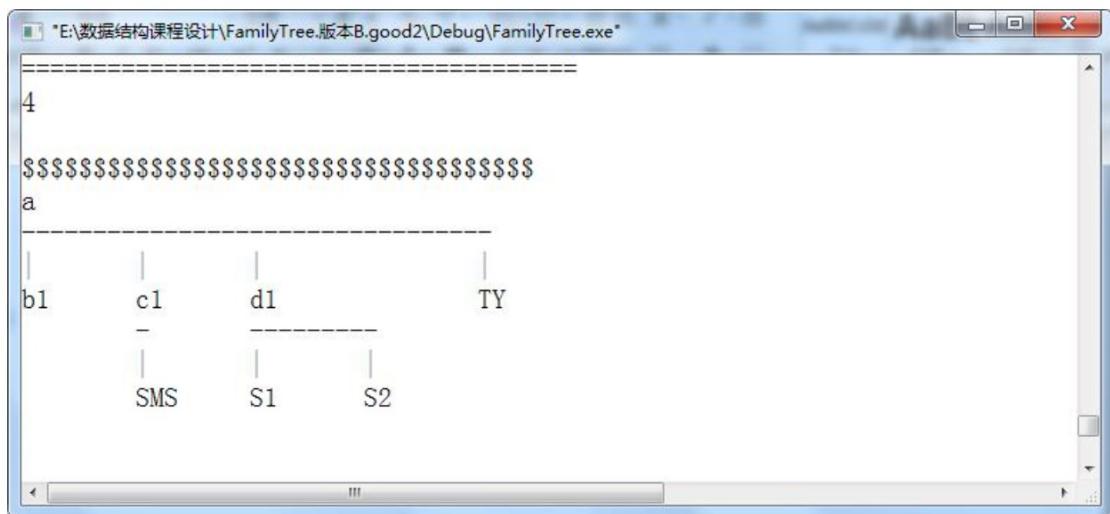


图 3-8 删除 Tom 之后的家族树

菜单：结点改名

改名时要求输入父亲的名字、儿子的名字、儿子的新名字

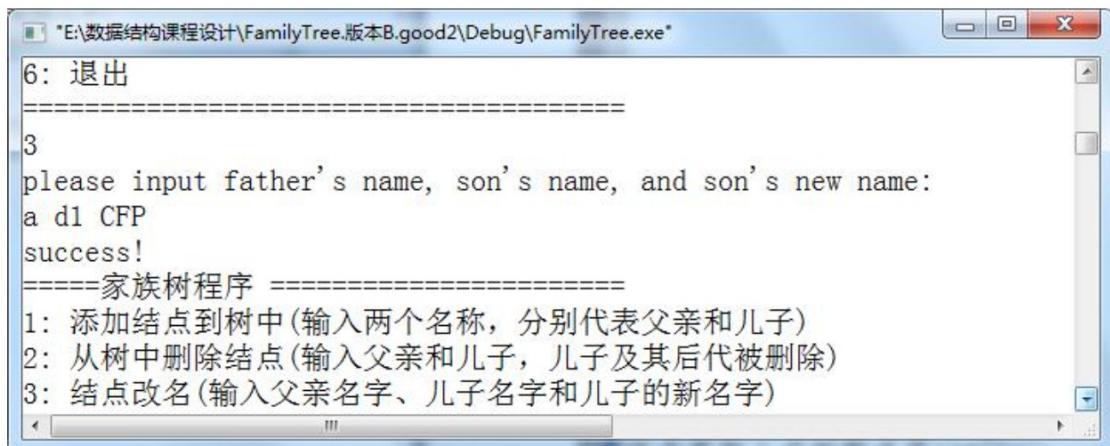


图 3-9 结点改名菜单

结点改名后，树变为：

```
E:\数据结构课程设计\FamilyTree.版本B.good2\Debug\FamilyTree.exe"
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
a
-----
|       |       |
b1      c1      CFP           TY
|       -       -----
|       |       |
SMS     s1      s2

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
the tree is saved to c:\FamilyTree.txt
=====家族树程序 =====
```

图 3-10 结点改名的结果

在程序退出时，会生成文件 c:\FamilyTree.txt，它的内容为：



图 3-11 程序退出时生成的文件

# 案例 4 校园地图查询与导引软件

## 4.1 简介

设计一个校园交通查询导引系统，能让游客查询学校的全部地点，能够给出各个地点离当前地点的距离，能够给出从一个地点到另外一个地点的最近距离和路径。同时，该软件还具有常用的数据维护功能，例如调研地图信息文件、保存地图、地点添加、道路添加、道路修改等功能。

该软件用到数据结构课程的图、排序、字符串等章节内容。

## 4.2 数据结构的定义

### 4.2.1 内存中的数据结构定义

```
#define GRAPH_TYPE 1    //定义为 1 时是无向图，定义为 2 时是有向图
#define MAX_VERTEX_NUM 30 //定义结点数的最大值
#define MAX_NAME_LENGTH 40 //结点名称最多包含的字符数
#define INFINITY 6503148 //定义无穷大

typedef struct ArcCell { // 弧的定义
    float distance;
    int direction; // 路径前进方向相对于 x 轴的夹角，360 度为 1 周
    char info[MAX_NAME_LENGTH]; // 该弧相关信息的指针
} ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];

typedef struct { // 图的定义
    char vexs[MAX_VERTEX_NUM][MAX_NAME_LENGTH]; // 顶点的信息
    AdjMatrix arcs; // 弧的信息
    int vexnum, arcnum; // 顶点数，弧数
} Graph;
```

### 4.2.2 地图数据文件的结构定义

地图的信息需要保存到一个数据文件中，下图为一个数据文件的例子，在这个例子中，第一行的 7 代表有 7 个地点，接下来的 7 行为 7 个地点的名称。接下来的 12 代表这个图共有 12 条边，最后的 12 行分别代表这 12 条边，每行的各个数据分别代表起点名称、终点名称、边的长度，边(道路)名称和边(道路)与 x 轴的夹角。这个数据文件可以通过程序的“保存地图”菜单得到，也可以用记事本编辑，然后通过程序的“调用地图信息文件”菜单读取到程序中。

```

7
v0
v1
v2
v3
v4
v5
v6
12
v0 v1 400 铁人路 45
v0 v2 600 创业路 0
v0 v3 600 逸夫路 315
v1 v2 100 雷锋路 315
v1 v4 700 中山路 0
v2 v4 600 学府路 45
v2 v5 400 卡尔加里路 315
v3 v2 200 西雅图路 45
v3 v5 500 花果山路 0
v4 v6 600 水帘洞路 315
v5 v4 100 曼谷路 90
v5 v6 800 敬业路 45

```

图 4-1 地图信息文件的一个例子

## 4.3 算法实现步骤

### 4.3.1 狄克斯特拉(Dijkstra)算法

基本思想是：设  $G=(V,E)$  是一个带权有向图，把图中顶点集合  $V$  分成两组：

第一组为已求出最短路径的顶点集合(用  $S$  表示,初始时  $S$  中只有一个源点,以后每求得一条最短路径  $v_0 \cdots v_k$ ,就将  $v_k$  加入到集合  $S$  中,直到全部顶点都加入到  $S$  中,算法就结束了)

第二组为其余未确定最短路径的顶点集合(用  $U$  表示)。

按最短路径长度的递增次序依次把第二组的顶点加入  $S$  中。在加入的过程中,总保持从源点  $v_0$  到  $S$  中各顶点的最短路径长度不大于从源点  $v_0$  到  $U$  中任何顶点的最短路径长度。

此外,每个顶点对应一个距离, $S$  中的顶点的距离就是从  $v_0$  到此顶点的最短路径长度, $U$  中的顶点的距离从  $v_0$  到此顶点只包括  $S$  中的顶点为中间顶点的当前最短路径长度。

狄克斯特拉算法的具体步骤如下：

(1) 初始时, $S$  只包含源点,即  $S=\{v_0\}$ , $v_0$  的距离为 0。 $U$  包含除  $v_0$  外的其他顶点, $U$  中顶点  $u$  距离为边上的权(若  $v_0$  与  $u$  有边  $\langle v_0, u \rangle$ )或  $\infty$ 。

(2) 从  $U$  中选取一个距离  $v_0$  最小的顶点  $v$ ,把  $v$  加入  $S$  中(该选定的距离就是  $v_0$  到  $v$  的最短路径长度)。

(3) 以  $v$  为新考虑的中间点,修改  $U$  中各顶点的距离：若从源点  $v_0$  到顶点  $w$  ( $w \in U$ ) 的距离(经过顶点  $v$ )比原来距离(不经过顶点  $v$ )短,则修改顶点  $w$  的距离值,修改后的距离值为顶点  $v$  的距离加上边  $\langle v, w \rangle$  上的权。

(4) 重复步骤(2)和(3)直到所有顶点都包含在  $S$  中。

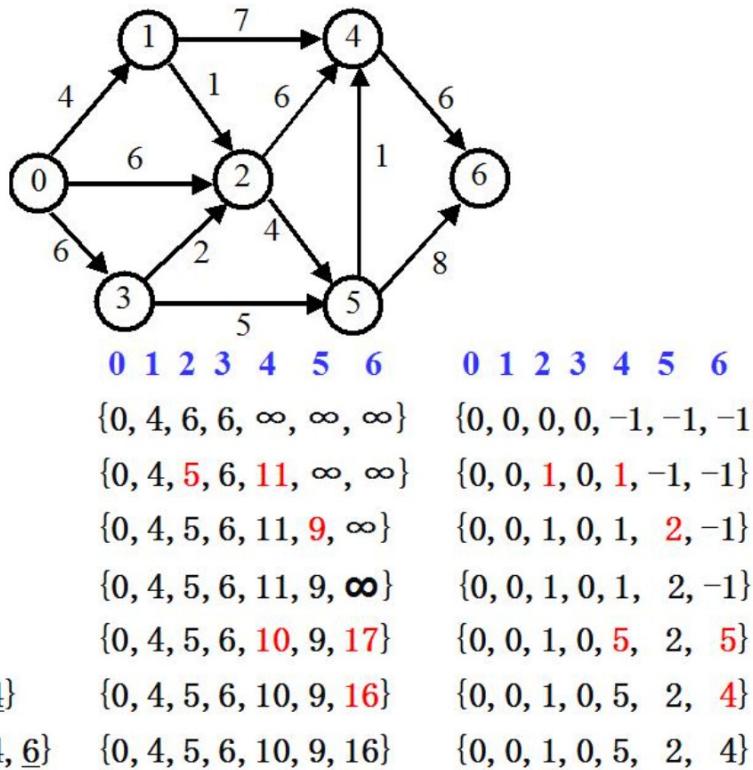


图 4-2 狄克斯特拉算法的计算过程

```

void ShortestPath_DIJ(Graph G, int v0,int P[], int D[])
{
    for(v=0;v<G.vexnum;++v)
    {
        final[v]=FALSE;
        D[v]=G.arcs[v0][v];
        if(D[v]<INFINITY)
            P[v]=v0;
        else
            P[v]=-1;
    }
    D[v0]=0;
    final[v0]=TRUE;
    for(i=1;i<G.vexnum;++i)
    {
        min=INFINITY;
        for(w=0;w<G.vexnum;++w)
            if(!final[w] && D[w]<min)
            {
                v=w;
                min=D[w];
            }
    }
}

```

```

final[v]=TRUE;
for(w=0;w<G.vexnum;++w)
{
    if(!final[w] && (min+G.arcs[v][w]<D[w]))
    {
        D[w]=min+G.arcs[v][w];
        P[w]=v;
    }
}

```

#### 4.3.2 前进方向的判定

在路径导引时，除了指出走那条路，还要指出是向前走、向左走还是向右走。为此，需存储每条道路的方向。道路方向按角度制存储，范围为从 0 到 359 度，认为 360 度和 0 度是相同的方向。将图中的道路与 x 轴夹角(假设有一个虚拟的 x 轴)定义为道路的方向，另外，道路的方向是和前进方向有关的，同一条道路，两个前进方向时道路方向刚好相差 180 度。假设从 A 点出发到达 B 点，再从 B 点到达 C 点，则计算 BC 与 AB 的角度差值，并且规定前进方向为：

|    |                        |
|----|------------------------|
| 向前 | 角度差值小于 45 度，或者大于 315 度 |
| 向左 | 角度差值大于 45 度，且小于 180 度  |
| 向右 | 其它情况                   |

下面是一个具体的例子，假设 AB 的方向为 50 度，BC 的方向为 315 度，则  $315-50=265$  度，所以 A 到达 B 后，应向右走。反之，如果从 C 出发，经过 B 到达 A，则 CB 的方向为  $315+180=495$  度，以 360 为模，得到 135 度，BA 的方向为  $50+180=230$  度， $230-135=95$  度，所以，C 到达 B 后，应向左。

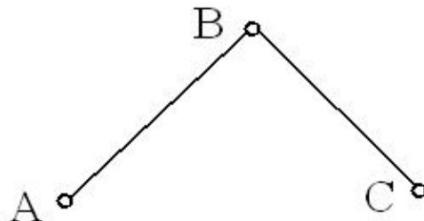


图 4-3 旋转角度计算的一个例子

## 4.4 程序的运行界面

主菜单：

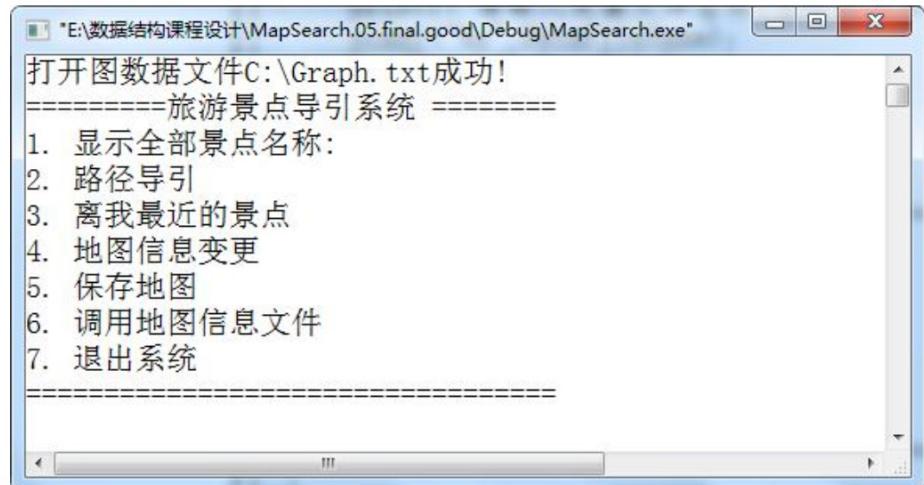


图 4-4 程序主菜单

显示全部景点名称的运行结果:

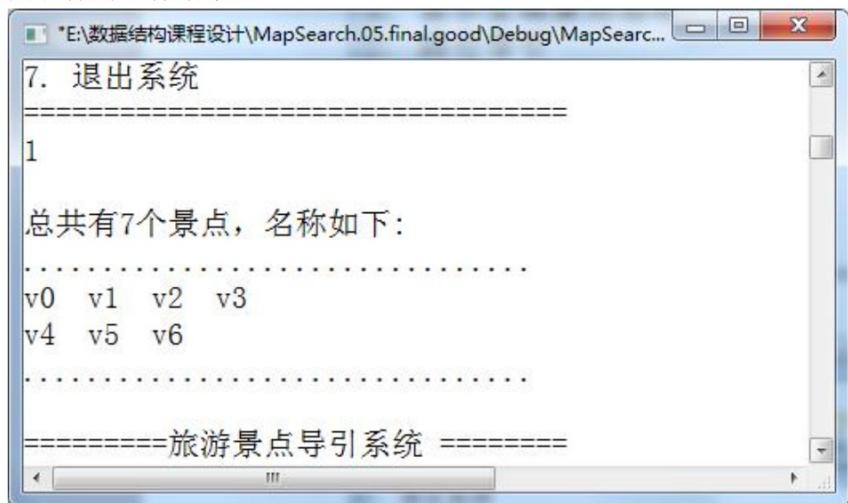


图 4-5 显示全部景点名称的运行结果

路径导引的运行结果，注意该运行结果与第 3 小节算法实现步骤中的图是相对应的：

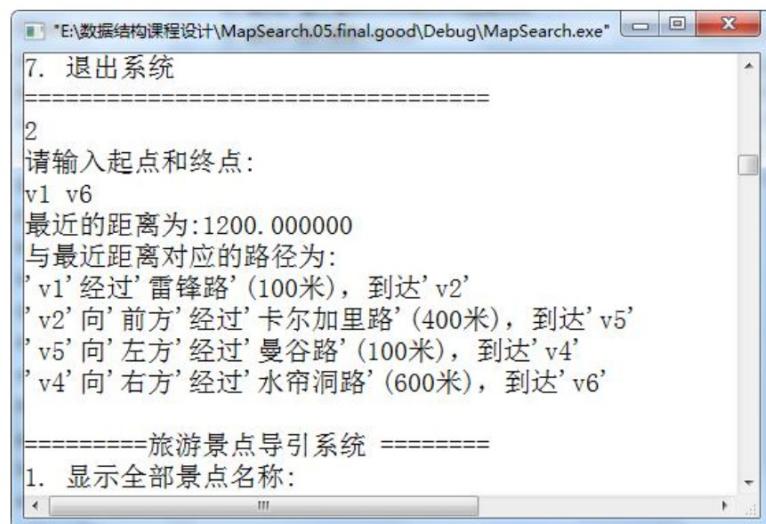


图 4-6 路径导引的运行结果

“离我最近的景点”的运行结果：



```
E:\数据结构课程设计\MapSearch.05.final.good\Debug\MapSearch.exe
=====
3
请输入你现在位置的编号:
1:v0 2:v1 3:v2 4:v3
5:v4 6:v5 7:v6
4
v2, 距离当前位置的距离为200
v1, 距离当前位置的距离为300
v5, 距离当前位置的距离为500
v4, 距离当前位置的距离为600
v0, 距离当前位置的距离为600
v6, 距离当前位置的距离为1200
=====旅游景点导引系统 =====
1. 显示全部景点名称:
```

图 4-7 “离我最近的景点”的运行结果

“地图信息变更—增加景点”的运行结果



```
E:\数据结构课程设计\MapSearch.05.final.good\Debug\MapSearch.exe
7. 退出系统
=====
4
请选择:
----1. 增加景点
----2. 修改景点名称
----3. 增加道路
----4. 修改道路
----5. 删除道路
1
请输入景点名称: TuShuGuan
=====旅游景点导引系统 =====
1. 显示全部景点名称:
```

图 4-8 “地图信息变更—增加景点”的运行结果

“地图信息变更—修改景点名称”的运行结果



图 4-9 “地图信息变更—修改景点名称”的运行结果

“地图信息变更—增加道路”的运行结果

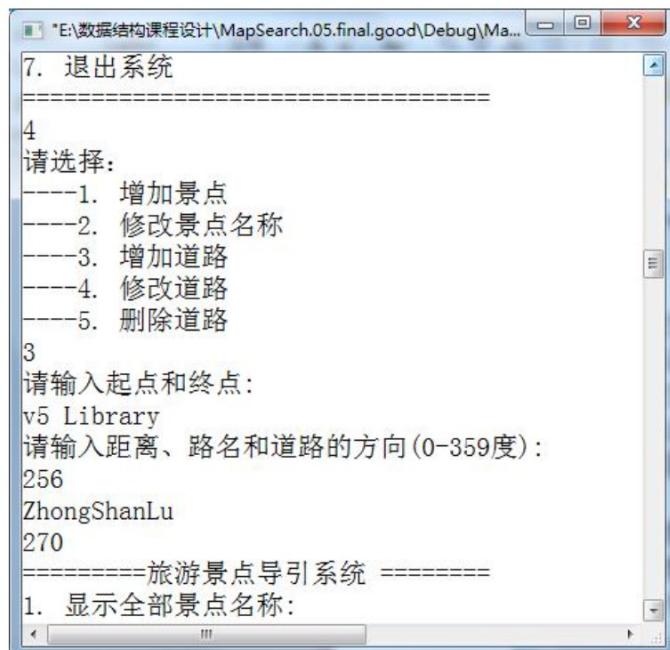


图 4-10 “地图信息变更—增加道路”的运行结果

保存地图的运行结果

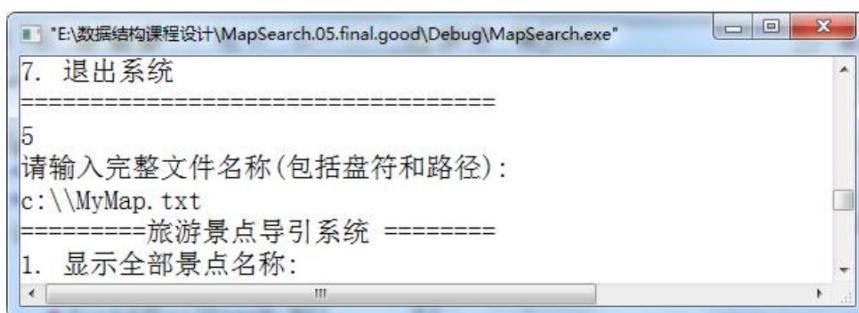


图 4-11 保存地图的运行结果

“地图信息变更—修改道路”的运行结果

```
7. 退出系统
=====
4
请选择:
----1. 增加景点
----2. 修改景点名称
----3. 增加道路
----4. 修改道路
----5. 删除道路
4
请输入起点和终点:
v5 Library
原来的距离、路名和道路方向(0-359度)为:
256 ZhongShanLu 270
请输入新的距离、路名和道路的方向(0-359度):
278 ZhongShanLu 270
=====旅游景点导引系统 =====
```

图 4-12 “地图信息变更—修改道路”的运行结果

“地图信息变更—删除道路”的运行结果

```
7. 退出系统
=====
4
请选择:
----1. 增加景点
----2. 修改景点名称
----3. 增加道路
----4. 修改道路
----5. 删除道路
5
请输入起点和终点:
v0 v9
v9不存在!
=====旅游景点导引系统 =====
```

图 4-13 “地图信息变更—删除道路”的运行结果

“调用地图信息文件”的运行结果

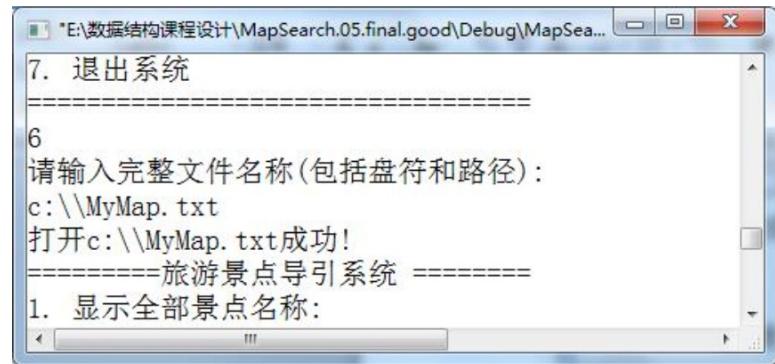


图 4-14 “调用地图信息文件”的运行结果

# 附录 1：如何在 C 语言中读写数据文件

读写文件操作是算法实现过程中经常用到的操作。文件的操作包括打开、关闭、读写、文件指针定位等操作。

`fopen`: 打开文件

定义函数 `FILE * fopen(const char * path, const char * mode)`;

`path` 为文件名及其路径

`mode` 为读写模式，有如下常见模式：

`r` 打开只读的文本文件，该文件必须存在。

`r+` 打开可读写的文本文件，该文件必须存在。

`w` 打开只写的文本文件，若文件存在则文件长度清为 0，即该文件内容会消失。若文件不存在则建立该文件。

`w+` 打开可读写的文本文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。

`rb, rb+, w, wb+` 这四种读写方式与上述类似，但是操作的对象是二进制文件，而不是文本文件。

`fclose`: 关闭文件

定义函数 `int fclose(FILE * stream)`;

函数说明 `fclose()` 用来关闭先前 `fopen()` 打开的文件。此动作会让缓冲区内的数据写入文件中，并释放系统所提供的文件资源。

返回值：若关文件动作成功则返回 0，否则返回值为非 0。

`fprintf`: 传送格式化输出到一个文件中，用于向文本文件写入数据

表头文件：`#include<stdio.h>`

函数原型：`int fprintf(FILE *stream, char *format[, argument,...])`;

`FILE*` 一个 `FILE` 型的指针

`char*` 格式化输入函数，和 `printf` 里的格式一样

返回值：成功时返回转换的字节数，失败时返回一个负数

`fscanf`: 从一个流中执行格式化输入

函数原型：`int fscanf(FILE *stream, char *format[, argument...])`;

`FILE*` 一个 `FILE` 型的指针

`char*` 格式化输出函数，和 `scanf` 里的格式一样

返回值：成功时返回转换的字节数，失败时返回一个负数

`feof`: 检查文件流是否读到了文件尾

定义函数 `int feof(FILE * stream)`;

函数说明 `feof()` 用来侦测是否读取到了文件尾，参数 `stream` 为 `fopen()` 所返回的文件指针。

返回值：如果已到文件尾则返回非零值，其他情况返回 0。

典型用法：`while(!feof(fp)) {.....}`

`fwrite`: 将数据写至文件流，用于向二进制文件写入数据，写入的数据可以用 `fread` 函数读取。

定义函数 `size_t fwrite(const void * ptr, size_t size, size_t nmemb, FILE * stream)`;

函数说明 `fwrite()` 用来将数据写入文件流中。参数 `stream` 为已打开的文件指针，参数 `ptr` 指向欲写入的数据地址，总共写入的字符数以参数 `size*nmemb` 来决定。

返回值 返回实际写入的 `nmemb` 数目。

`fread`: 从文件流读入数据，用于从二进制文件读入数据，该文件可以由 `fwrite` 函数写入。

定义函数 `size_t fread(void * ptr, size_t size, size_t nmemb, FILE * stream);`

函数说明 `fread()` 用于从二进制文件读入数据。参数 `stream` 为已打开的文件指针，读取到的数据放到参数 `ptr` 指向的数据地址，总共写入的字符数以参数 `size*nmemb` 来决定。

返回值 返回实际写入的 `nmemb` 数目。

例子 1：文本文件的读和写

```
#include "stdio.h"
```

```
void main()
{
    FILE *fp;//定义文件指针
    int data1;
    float data2;
    data1=3;
    data2=(float)1.8;

    fp=fopen("c:\\mydata.txt", "w");//以写入的方式打开文本文件
    if(fp==NULL)//判断文件打开是否成功
    {
        printf("open file error!\n");
        return;
    }
    fprintf(fp, "%d ", data1);//以文本形式向文件写入数据
    fprintf(fp, "%f ", data2);
    fclose(fp);//关闭文件

    fp=fopen("c:\\mydata.txt", "r");//以只读的方式打开文本文件
    if(fp==NULL)//判断文件打开是否成功
    {
        printf("open file error!\n");
        return;
    }
    fscanf(fp, "%d", &data1);//从文件中读取数据
    fscanf(fp, "%f", &data2);
    fclose(fp);//关闭文件
    printf("%d\n", data1);//验证结果
    printf("%f\n", data2);//验证结果
}
```

例子 2：二进制文件的读和写

```
#include "stdio.h"
```

```
void main()
{
    FILE *fp;
    int data1;
    float data2;
    data1=3;
    data2=(float)1.8;

    fp=fopen("c:\\mydata.txt", "wb");//以写入的方式打开二进制文件
    if(fp==NULL)//判断文件打开是否成功
    {
        printf("open file error!\n");
        return;
    }
    fwrite(&data1, sizeof(int), 1, fp);//以二进制形式向文件写入数据
    fwrite(&data2, sizeof(float), 1, fp);
    fclose(fp);//关闭文件

    fp=fopen("c:\\mydata.txt", "rb");//以只读的方式打开二进制文件
    if(fp==NULL)//判断文件打开是否成功
    {
        printf("open file error!\n");
        return;
    }
    fread(&data1, sizeof(int), 1, fp);//从二进制文件读取数据
    fread(&data2, sizeof(float), 1, fp);
    fclose(fp);//关闭文件
    printf("%d\n", data1);//验证结果
    printf("%f\n", data2);//验证结果
}
```

## 附录 2：如何在 Visual Studio 6.0 环境下 创建和调试 C++程序

算法竞赛入门课程的实验都可以在 Visual Studio 6.0 环境下完成。熟悉和掌握 Visual Studio 6.0 环境的使用是完成后续的实验的前提条件。

### 1. 创建一个 C++控制台项目

创建一个 C++控制台项目可以按照如下步骤进行：

(1) 点“File”菜单下的“New”菜单，出现如图 1 所示的窗口。先在左侧的列表中选择“Win32 Console Application”选项，然后在窗口右侧 Project name 文本框中添加要创建的项目的名称。点“OK”按钮。

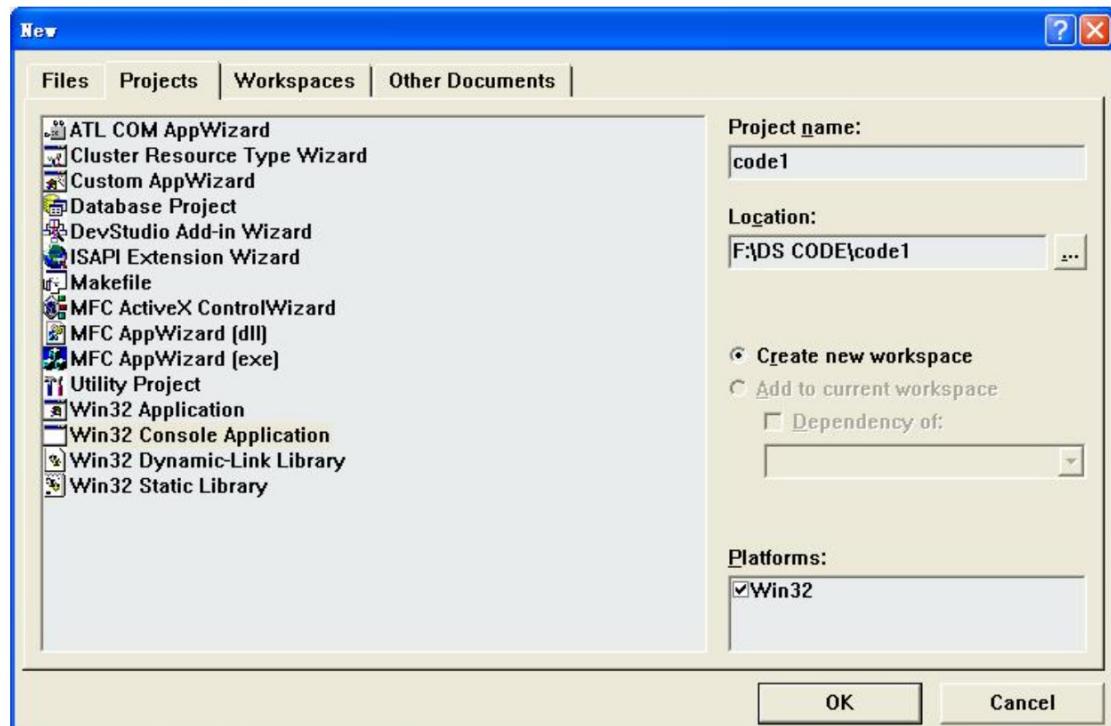


图 1 选择项目类型与填写项目名称

(2) 如图 2 所示，在弹出的窗口中选择 A “Hello, World!” application。

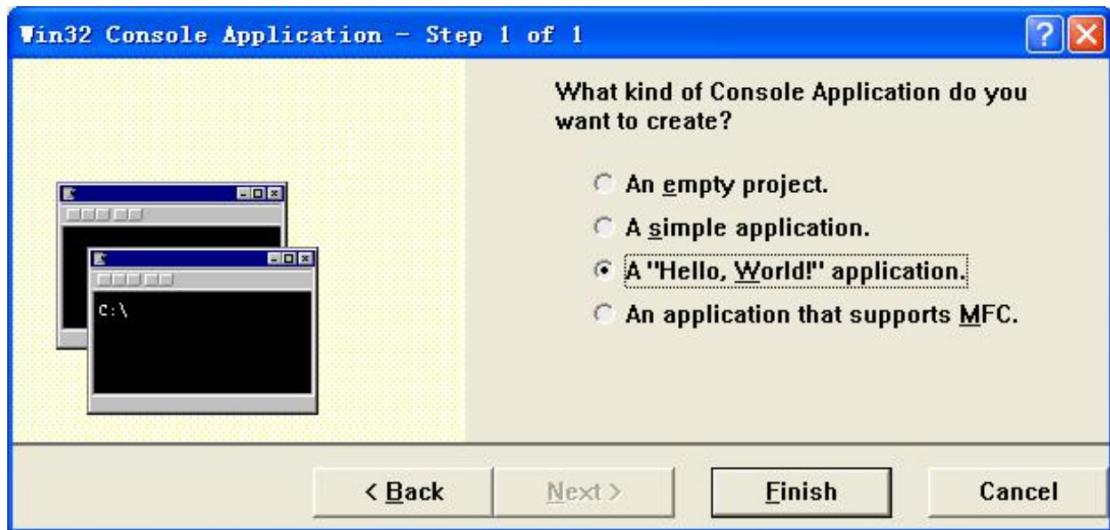


图 2 选择控制台应用程序的类型

(3) 如图 3 所示，在创建后的窗口的左侧，点左侧的“+”，再点“Globals”左侧的“+”，最后点“main”，得到如图 4 所示的窗口。

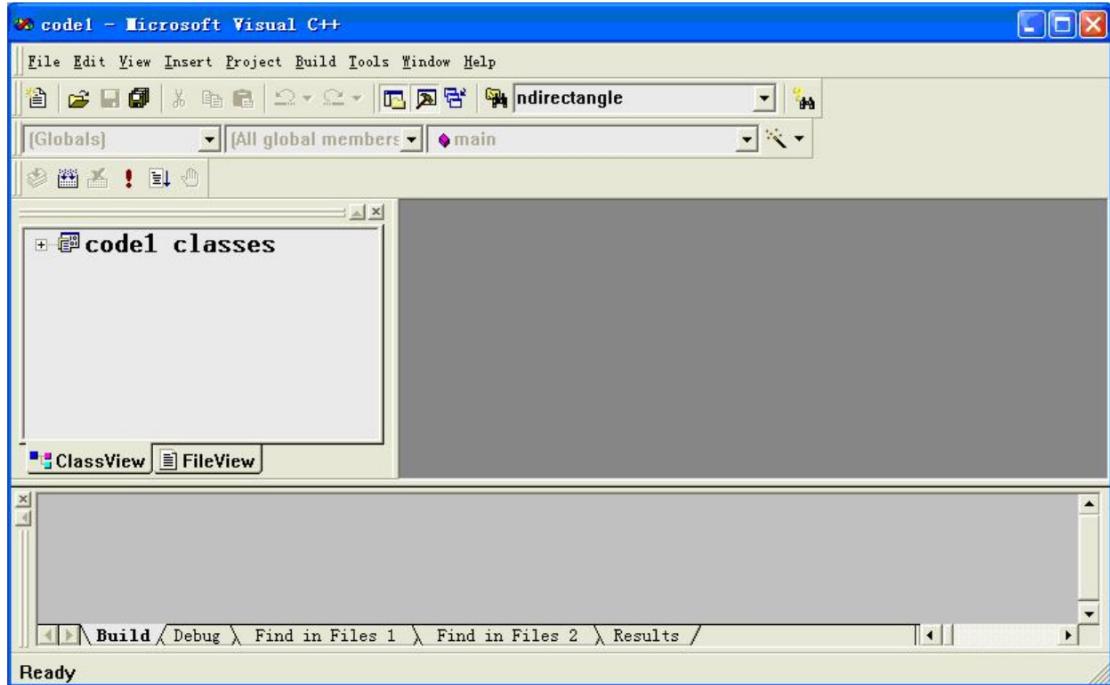


图 3 创建成功的控制台应用程序

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "code1 - Microsoft Visual C++ - [code1.cpp]". The menu bar includes File, Edit, View, Insert, Project, Build, Tools, Window, Help. The toolbar has icons for file operations like Open, Save, and Build. The status bar at the bottom shows "Ready".

The left pane is the ClassView, showing a tree structure under "code1 classes" with a "Globals" node containing "main(int argc, char\* argv[])". The right pane is the main code editor, displaying the following C++ code:

```
// code1.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"

int main(int argc, char* argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

图 4 显示 main() 函数

(4) 如图 5 所示，接下来在右侧的窗口中添加程序代码，得到一个简单的程序。然后点“Build”菜单下的“Rebulid All”，生成可执行文件。

The screenshot shows the Microsoft Visual Studio IDE interface, similar to Figure 4. The title bar reads "code1 - Microsoft Visual C++ - [code1.cpp]". The menu bar includes File, Edit, View, Insert, Project, Build, Tools, Window, Help. The toolbar has icons for file operations like Open, Save, and Build. The status bar at the bottom shows "Ln 11, Col 11".

The left pane is the ClassView, showing a tree structure under "code1 classes" with a "Globals" node containing "fun(int x, int y)" and "main()". The right pane is the main code editor, displaying the following C++ code:

```
// code1.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
int fun(int x, int y)
{
    int val;
    val=x + y;
    return val;
}

int main()
{
    int v;
    v=fun(4, 5);
    printf("the result is: %d\n",v);
    return 0;
}
```

图 5 改写后得到的一个简单的程序

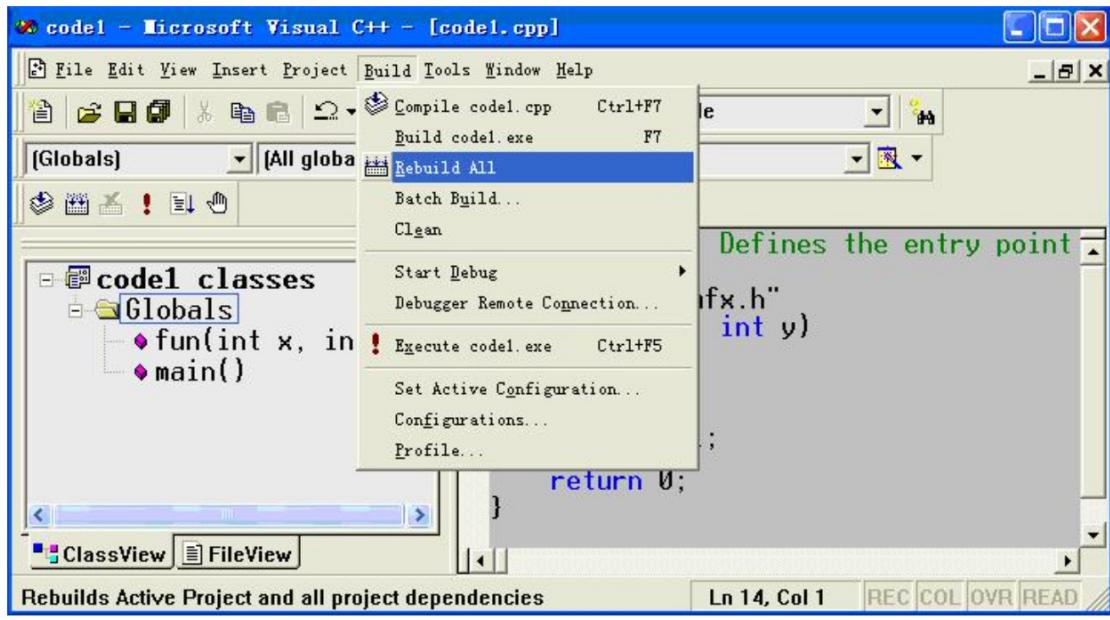


图 6 编译程序

## 2. 程序的调试

调试程序程序包括设置断点，调试执行，变量查看等。

### (1) 设置断点

把光标放在需要设置断点的行，然后按”F9”功能键，即可设置断点。如图 7 所示，在某行被设置断点后，该行的左侧会出现一个棕色的圆点。

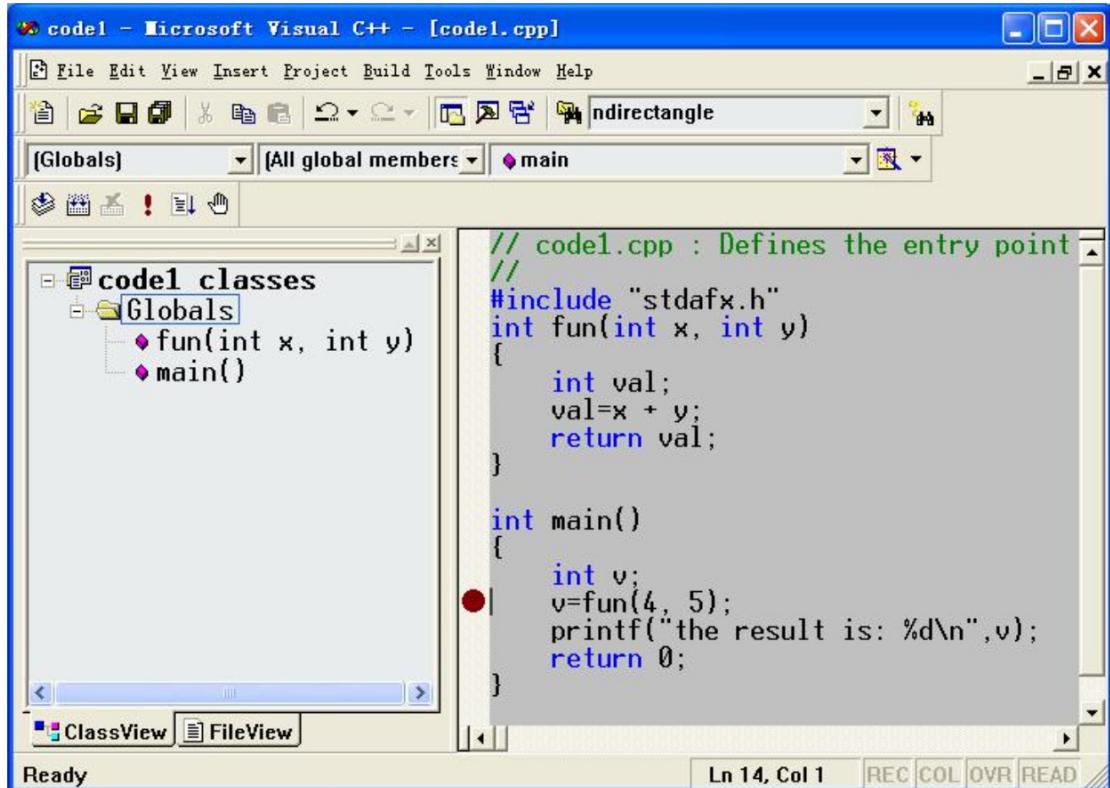


图 7 设置断点

### (2) 调试执行

按功能键”F5”可以运行程序至光标处。接下来有如下几种执行方式：

