

# 《数据结构与算法》课程设计指导书

## 一、适用专业及年级

本指导书适用于计算机科学与技术专业、软件工程专业二年级本科生。

## 二、目的与要求

数据结构课程设计的目的是通过对一个以数据结构与算法设计为核心的课题设计与实现过程的训练，培养学生综合运用数据结构以及程序设计、离散数学等相关课程的基础知识、能力和方法，系统学习和掌握问题建模、数据结构设计、算法设计与实现、测试等各环节的方法和能力。

本课程设计要求理解数据结构课程的作用和学习目标；理解数据结构的逻辑结构、存储结构、运算及其实现、算法分析等几个方面的关系；理解线性表、树、二叉树、图结构等结构的特性、基本算法及其实现，理解并掌握各种排序算法和查找方法的性能及其适用场合；能熟练运用一种程序设计语言实现相关的设计。测试数据的选择要具有充分性，设计报告应能完整表达设计与实现。

## 三、课程设计内容

### 第1题：集合的并、交和差运算

#### 【问题描述】

编制一个能演示执行集合的并、交和差运算的程序。

#### 【基本要求】

- (1) 集合的元素限定为小写字符['a'..'z']；
- (2) 演示程序以用户和计算机的对话方式执行。
- (3) 使用汉字显示。

#### 【选做内容】

- (1) 求集合的补集。

### 第2题：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 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)指示迷宫中的一个坐标，d 表示走到下一坐标的方向。如：对于下列数据的迷宫，输出的一条通路为：(1,1,1), (1,2,2), (2,2,2) (3,2,3), (3,1,2) .....

### 第 3 题：哈夫曼编码/译码器

#### 【问题描述】

设计一个利用哈夫曼算法的编码和译码系统，重复地显示并处理以下项目，直到选择退出为止。

#### 【基本要求】

(1) 将权值数据存放在数据文件(文件名为 data.txt，位于执行程序的当前目录中)

(2) 初始化：键盘输入字符集大小 n、n 个字符和 n 个权值，建立哈夫曼树；

(3) 编码：利用建好的哈夫曼树生成哈夫曼编码；

(4) 输出编码；

(5) 设字符集及频度如下表：

|    |     |    |    |    |    |     |    |    |    |    |    |    |    |   |
|----|-----|----|----|----|----|-----|----|----|----|----|----|----|----|---|
| 字符 | 空格  | A  | B  | C  | D  | E   | F  | G  | H  | I  | J  | K  | L  | M |
| 频度 | 186 | 64 | 13 | 22 | 32 | 103 | 21 | 15 | 47 | 57 | 15 | 32 | 20 |   |
| 字符 | N   | O  | P  | Q  | R  | S   | T  | U  | V  | W  | X  | Y  | Z  |   |
| 频度 | 57  | 63 | 15 | 1  | 48 | 51  | 80 | 23 | 8  | 8  | 1  | 16 | 1  |   |

#### 【进一步完成内容】

译码功能。

### 第 4 题：校园导游系统

#### 【问题描述】

设计一个校园导游程序，为来访的客人提供各种信息查询服务。

#### 【基本要求】

(1) 设计天津商业大学的校园平面图，所含景点不少于 10 个。以图中顶点表示校内各景点，存放景点名称、代号、简介等信息；以边表示路径，存放路径长度等相关信息。

(2) 为来访客人提供图中任意景点相关信息的查询。

(3) 为来访客人提供图中任意景点的问路查询，即查询任意两个景点之间的一条最短的简单路径。

### **第 5 题：哈希表的建立与查找**

#### **【问题描述】**

针对本班中的“人名”设计一个哈希表，使得平均查找长度不超过  $R$ ，完成相应的建表和查表程序。

#### **【基本要求】**

(1) 假设人名为中国人姓名的汉语拼音形式，待填入哈希表的人名共有 30 个，取平均查找长度的上限为  $R$ 。

(2) 哈希函数用除留余数法构造，处理冲突用线性探测再散列法。

### **第 6 题：内部排序算法比较**

#### **【问题描述】**

设计一个测试程序比较几种内部排序算法的关键字比较次数和移动次数以取得直观感受。

#### **【基本要求】**

(1) 对以下六种常用的内部排序算法进行比较：希尔排序、直接选择排序、快速排序、直接插入排序、堆排序、冒泡排序、。

(2) 待排序表的表长和数据可以由用户自己确定，也可以由随机数产生程序自动产生；至少要用五组不同的输入数据作比较；比较的指标为关键字的比较次数和关键字的移动次数（关键字的交换计为三次移动）。

(3) 最后要对结果作出简单分析。

## 附录：课程设计报告

### 第 1 题：集合的并、交和差运算

#### 1. 实验内容

集合的并、交和差运算

##### 【问题描述】

编制一个能演示执行集合的并、交和差运算的程序。

##### 【基本要求】

- (1) 集合的元素限定为小写字符['a'..'z'];
- (2) 演示程序以用户和计算机的对话方式执行。
- (3) 使用汉字显示。

##### 【选做内容】

- (1) 求集合的补集。

#### 2. 需求分析

(1) 本演示程序中，集合的元素限定为小写字母字符['a'..'z']，集合的输入形式为一个以“回车符”为结束标志的字符串，串中字符顺序不限，不能输入非法字符。

(2) 演示程序一用户和计算机的对话方式执行，日在计算机终端上显示“提示信息”，之后，有用户在键盘上输入演示程序中规定的运算命令；相应的输入数据和运算结果显示在其后。

(3) 程序执行的命令包括：

1) 构造集合 1; 2) 构造集合 2; 3) 求并集; 4) 求交集; 5) 求差集; 6 结束。

注释：“构造集合 1”和“构造集合 2”时，需要一字符串的形式键入集合元素。

### 3. 概要设计

#### 3.1 抽象数据类型定义

为了实现上述功能，应该有序链表表示集合。为此，需要两个抽象集合数据类型：有序表和集合；

（1）有序表的抽象数据类型定义为：

ADT OrderedList{

数据对象：  $D=\{a_i | a_i \in \text{CharSet}, i=1,2,\dots,n, n \geq 0\}$

数据关系：  $R1=\{ \langle a_{i-1}, a_i \rangle | a_{i-1}, a_i \in D, a_{i-1} < a_i, i=2,\dots,n \}$

基本操作：

Readdata(pointer head)

初始条件： head 是以 head 为头结点的空链表。

操作结果是：生成以 head 为头结点的非空链表。

pop(pointer head)

初始条件： head 是以 head 为头结点的非空链表。

操作结果：将以 head 为头结点的链表中数据逐个输出。

}ADT OrderedList

（2）集合的抽象数据类型定义为：

ADT Set{

数据对象：  $D=\{a_i | a_i \text{ 为小写英文字母且互不相同}, i=1,2,3,4,\dots,n, 0 \leq n \leq 26\}$

数据关系：  $R1=\{\}$ ;

基本操作：

bing(pointer head1, pointer head2, pointer head3)

初始条件：链表 head1, head2, head3 已存在

操作结果：生成一个由 head1, head2 的并集构成的集合 head3.

Jiao(pointer head1, pointer head2, pointer head3)

初始条件：链表 head1, head2, head3 已存在

操作结果：生成一个由 head1 和 head2 的交集构成的集合 head3。

```
Cha(pointer head1.pointer head2.pointer.head3)
```

初始条件：链表 head1.head2.head3 已存在

操作结果:生成一个由 head1 和 head2 的差集构成的集合 head3.

```
}ADT Set
```

### 3.2 模块划分

本程序包括四个模块：

- (1) 节点结构单元模块——定义有序表的节点结构；
- (2) 有序表单元模块——实现有序表的抽象数据类型；
- (3) 集合单元模块——实现集合获得抽象数据类型；
- (4) 主程序模块；

```
Void main(){
```

初始化：

```
do{
```

接受命令：

处理命令：

```
}while(“命令”!=”退出”);
```

```
}
```

## 4. 详细设计

### 4.1 数据类型的定义

```
typedef struct LNode// 定义结构体类型指针
```

```
{
```

```
    char data;
```

```
    struct LNode*next;
```

```
}*pointer;
```

```
void readdata(pointer head)//定义输入集合函数
```

```
{    pointer p;
```

```
    char tmp;
```

```
    scanf("%c",&tmp);
```

```

while(tmp!='\n')
{p=(pointer)malloc(sizeof(struct LNode));
p->data=tmp;
p->next=head->next;
head->next=p;
if(tmp>='a'&&tmp<='z')//判断 字符是否在(a,z)内
scanf("%c",&tmp);
}
}

```

## 4.2 主要模块的算法描述

### (1) 输入函数部分

void pop(pointer head)//定义输出集合函数

```

{ pointer p;
p=head->next;
while(p!=NULL)
{printf("%c",p->data);
p=p->next;
}
printf("\n");
}

```

### (2) 并集函数部分

void bing(pointer head1,pointer head2,pointer head3)//定义集合的并集函数

```

{pointer p1,p2,p3;
p1=head1->next;
while(p1!=NULL)
{
p3=(pointer)malloc(sizeof(struct LNode));
p3->data=p1->data;
p3->next=head3->next;
}
}

```

```

        head3->next=p3;
        p1=p1->next;
    }
    p2=head2->next;
    while(p2!=NULL)
    {p1=head1->next;
        while((p1!=NULL)&&(p1->data!=p2->data))
            p1=p1->next;
        if(p1==NULL)
        {
            p3=(pointer)malloc(sizeof(struct LNode));
            p3->data=p2->data;
            p3->next=head3->next;
            head3->next=p3;
        }
        p2=p2->next;
    }
}

```

### (3) 交集函数部分

void jiao(pointer head1,pointer head2,pointer head3)//定义集合的交集函数

```

{pointer p1,p2,p3;
p1=head1->next;
while(p1!=NULL)
{
    p2=head2->next;
    while((p2!=NULL)&&(p2->data!=p1->data))
        p2=p2->next;
    if((p2!=NULL)&&(p2->data==p1->data))
    {
        p3=(pointer)malloc(sizeof(struct LNode));

```



```

        p3->data=p1->data;
        p3->next=head3->next;
        head3->next=p3;
    }
    p1=p1->next;
}
}

```

#### (4) 差集函数部分

void cha(pointer head1,pointer head2,pointer head3)//定义函数的差集函数

```

{ pointer  p1,p2,p3;
p1=head1->next;
while(p1!=NULL)
{p2=head2->next;
while((p2!=NULL)&&(p2->data!=p1->data))
p2=p2->next;
if(p2==NULL)
{p3=(pointer)malloc(sizeof(struct LNode));
p3->data=p1->data;
p3->next=head3->next;
head3->next=p3;
}
p1=p1->next;
}
}

```

#### 4.3 函数之间的调用关系

```

void main()//主函数
{
    int x;

    printf("(输入 a 到 z 内的字符，按回车键结束，第一个集合大于第二

```

个集合)\n");

```
pointer head1,head2,head3;
```

```
head1=(pointer)malloc(sizeof(struct LNode));
```

```
head1->next=NULL;
```

```
head2=(pointer)malloc(sizeof(struct LNode));
```

```
head2->next=NULL;
```

```
head3=(pointer)malloc(sizeof(struct LNode));
```

```
head3->next=NULL;
```

```
printf("请输入集合函数 1: \n");
```

```
readdata(head1);//调用请输入集合函数
```

```
printf("请输入集合函数 2: \n");
```

```
readdata(head2);//调用输入集合函数
```

```
A:printf("1 并集    2 交集    3 差集    4 结束\n");
```

```
do{
```

```
    printf("请选择序号\n");
```

```
    scanf("%d",&x);
```

```
switch(x)
```

```
{
```

```
    case 1:
```

```
        printf("两个集合的并是\n");
```

```
        bing(head1,head2,head3);//调用并集函数
```

```
        pop(head3);
```

```
        head3->next=NULL;
```

```
        break;
```

```
    case 2:
```

```
        printf("两个集合的交是\n");
```

```
        jiao(head1,head2,head3);//调用交集函数
```

```
        pop(head3);
```

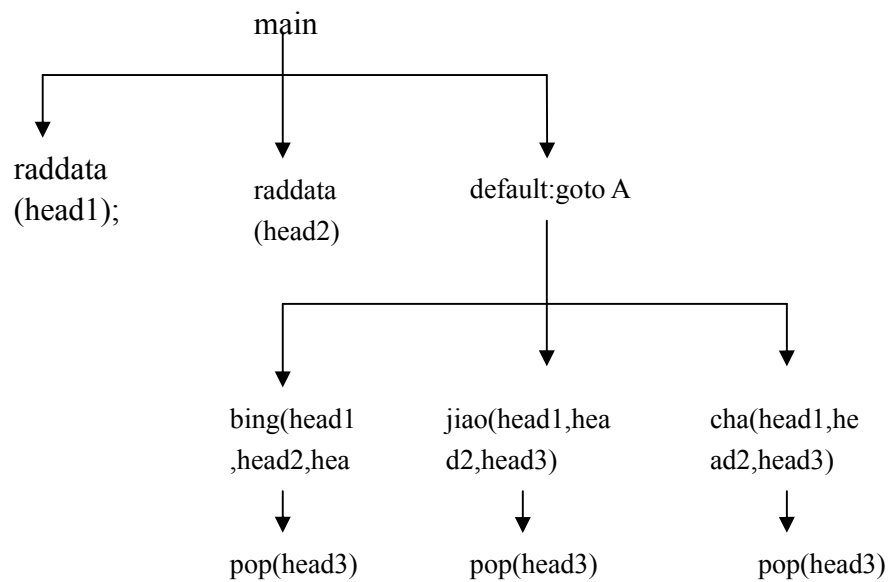
```
        head3->next=NULL;
```

```

        break;
    case 3:
        printf("两集合的差是\n");
        cha(head1,head2,head3);//调用差集函数
        pop(head3);
        head3->next=NULL;
        break;
    case 4:break;
    default:goto A;
}

```

4.4 函数的调用关系反映了程序的层次结构：



## 5. 程序运行说明与测试

### 5.1 用户手册

- (1) 本程序的运行环境为 DOC 操作系统，执行文件 sefr.exe.
- (2) 进入演示程序后即显示为本方式的用户界面：

The screenshot shows a DOS-style command window titled "F:\c程序设计\MSDev98\MyProjects\sefr\Debug\sefr.exe". The program prompts the user to "输入a到z内的字符，按回车键结束，第一个集合大于第二个集合" (Enter characters a-z, press Enter to end, the first set is greater than the second set). Below this, it asks for "两个集合之间的“并”，“交”，“差”" (Union, Intersection, Difference of two sets). The user has entered "asd" for the first set and "hsg" for the second set. The program then displays a menu: "1并集 2交集 3差集 4结束" (1 Union 2 Intersection 3 Difference 4 End). The user has selected "1" for the union operation. The program then displays the result: "两个集合的并是" (The union of the two sets is) followed by "ghasd". The user has selected "2" for the intersection operation. The program then displays the result: "两个集合的交是" (The intersection of the two sets is) followed by "sd". The user has selected "3" for the difference operation. The program then displays the result: "两集合的差是" (The difference of the two sets is) followed by "a". The user has selected "4" for the end operation. The program then displays "Press any key to continue".

Annotations in the image:

- 提示信息 (Information提示): Points to the initial instruction at the top.
- 显示集合1 (Display Set 1): Points to the input "asd".
- 显示集合2 (Display Set 2): Points to the input "hsg".
- 操作命令清单 (Operation Command List): Points to the menu "1并集 2交集 3差集 4结束".
- 显示结果集合 (Display Result Set): Points to the result "ghasd".
- 操作提示信息 (Operation Information提示): Points to the prompt "请选择序号" (Please select a number) after the intersection operation.

- (3) 进入：“构造集合 1”和“构造集合 2”的命令后，即提示键入集合元素串，结束符为“回车符”。
- (4) 接受其他命令后即执行相应的运算和显示相应结果。

## 5.2 测试数据：

- (1) set1="fag",set2="fat",
- (2) set1="123fag",set2="45fat";

## 5.3 测试结果及分析

- (1)  $Set1 \cup set2 = "fatg"$ ,  $set1 \cap set2 = "fa"$ ,  $set1 - set2 = "g"$ ;

```
C:\ "F:\c程序设计\MSDev98\MyProjects\ef\Debug\ef.exe"
<输入a到z内的字符，按回车键结束，第一个集合大于第二个集合>
*****

请输入集合1:
fag
请输入集合函数2:
fat
1并集  2交集  3差集  4结束
请选择序号
1
两个集合的并是
tfag
请选择序号
2
两个集合的交是
fa
请选择序号
3
两集合的差是
g
请选择序号
4
Press any key to continue
```

(2) 程序运行错误，无法继续运行，因为输入错误字符；

```
C:\ "F:\c程序设计\MSDev98\MyProjects\ef\Debug\ef.exe"
<输入a到z内的字符，按回车键结束，第一个集合大于第二个集合>
*****

请输入集合1:
123fag
```

## 6. 实验总结

在整个实验过程中，本组遇到的最大的困难是程序编码的实现，在这方面尤其凸显的问题就是编译程序后调试的问题。有些编译问题，在提示信息的提示下仍然不能看明白，通过各种修改不能编译正确，在这种情况下，我们组的解决办法是：将组能成员集中起来，注重研究，通过查找各方面的资料。如果仍然没有解决的话，去找其他组的同学们和老师寻求帮助。

在实验中我们的最大感悟就是编程需要巨大的毅力、耐力和细心度。在许多问题上，如果出现一点的错误的话，就很有可能成为这个程序的致命伤，导致程序的编译失败。还有组员之间的沟通协作同样是最重要的，正所谓集体力量大，得到了充分的体现。

### 附：源程序清单

```
#include<stdio.h>
#include<stdlib.h>
typedef struct LNode// 定义结构体类型指针
{
    char data;
    struct LNode*next;
}*pointer;
void readdata(pointer head)//定义输入集合函数
{ pointer p;
    char tmp;
    scanf("%c",&tmp);
    while(tmp!='\n')
    {p=(pointer)malloc(sizeof(struct LNode));
    p->data=tmp;
    p->next=head->next;
    head->next=p;
    if(tmp>='a'&&tmp<='z')//判断 字符是否在(a,z)内
        scanf("%c",&tmp);
    }
}
void pop(pointer head)//定义输出集合函数
{ pointer p;
    p=head->next;
    while(p!=NULL)
```

```

    {printf("%c",p->data);
    p=p->next;
    }
    printf("\n");
    }
    void bing(pointer head1,pointer head2,pointer head3)//定义集合的并集函数
    {pointer p1,p2,p3;
    p1=head1->next;
    while(p1!=NULL)
    {
        p3=(pointer)malloc(sizeof(struct LNode));
        p3->data=p1->data;
        p3->next=head3->next;
        head3->next=p3;
        p1=p1->next;
    }
    p2=head2->next;
    while(p2!=NULL)
    {p1=head1->next;
        while((p1!=NULL)&&(p1->data!=p2->data))
            p1=p1->next;
        if(p1==NULL)
        {
            p3=(pointer)malloc(sizeof(struct LNode));
            p3->data=p2->data;
            p3->next=head3->next;
            head3->next=p3;
        }
        p2=p2->next;
    }
    }
    void jiao(pointer head1,pointer head2,pointer head3)//定义集合的交集函数
    {pointer p1,p2,p3;
    p1=head1->next;
    while(p1!=NULL)
    {
        p2=head2->next;
        while((p2!=NULL)&&(p2->data!=p1->data))
            p2=p2->next;
        if((p2!=NULL)&&(p2->data==p1->data))

        {
            p3=(pointer)malloc(sizeof(struct LNode));
            p3->data=p1->data;

```

```

        p3->next=head3->next;
        head3->next=p3;
    }
    p1=p1->next;
}
}

void cha(pointer head1,pointer head2,pointer head3)//定义函数的差集函数
{ pointer  p1,p2,p3;
p1=head1->next;
while(p1!=NULL)
{p2=head2->next;
while((p2!=NULL)&&(p2->data!=p1->data))
p2=p2->next;
if(p2==NULL)
{p3=(pointer)malloc(sizeof(struct LNode));
p3->data=p1->data;
p3->next=head3->next;
head3->next=p3;
}
p1=p1->next;
}
}

void main()//主函数
{
int x;
printf("(输入 a 到 z 内的字符，按回车键结束，第一个集合大于第二个集合)\n");

printf("*****\n");
printf("\n");
    pointer head1,head2,head3;
    head1=(pointer)malloc(sizeof(struct LNode));
    head1->next=NULL;
    head2=(pointer)malloc(sizeof(struct LNode));
    head2->next=NULL;
    head3=(pointer)malloc(sizeof(struct LNode));
    head3->next=NULL;
    printf("请输入集合 1:  \n");
    readdata(head1);//调用请输入集合函数
    printf("请输入集合函数 2: \n");
    readdata(head2);//调用输入集合函数
    A:printf("1 并集    2 交集    3 差集    4 结束\n");

```



```

do{
    printf("请选择序号\n");
    scanf("%d",&x);
    switch(x)
    {
        case 1:
            printf("两个集合的并是\n");
            bing(head1,head2,head3);//调用并集函数
            pop(head3);
            head3->next=NULL;
            break;
        case 2:
            printf("两个集合的交是\n");
            jiao(head1,head2,head3);//调用交集函数
            pop(head3);
            head3->next=NULL;
            break;
        case 3:
            printf("两集合的差是\n");
            cha(head1,head2,head3);//调用差集函数
            pop(head3);
            head3->next=NULL;
            break;
        case 4:break;
        default:goto A;
    }while(x!=4);
}

```

## 第 2 题：迷宫问题

### 1. 实验内容

#### 【问题描述】

以一个  $m \times n$  的长方阵表示迷宫, 0 和 1 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:  $(i,j)$  指示迷宫中的一个坐标,  $d$  表示走到下一坐标的方向。如: 对于下列数据的迷宫, 输出的一条通路为:  $(1,1,1), (1,2,2), (2,2,2), (3,2,3), (3,1,2) \dots$

### 2. 需求分析

(1) 以二维数组  $\text{maze}[M+2][N+2]$  表示迷宫, 其中  $\text{maze}[i][0]$  和  $\text{maze}[i][N+1]$  ( $0 \leq i \leq N+1$ ) 以及  $\text{maze}[0][j]$  和  $\text{maze}[M+1][j]$  ( $0 \leq j \leq M+1$ ) 为外加的一圈围墙。数组中元素 0 表示障碍 1 表示可通过, 迷宫的大小可不限;

(2) 迷宫中的数据均由用户自由输入;

(3) 迷宫的出口和入口可由用户自由设定;

(4) 本程序只求出一条成功的通路;

(5) 程序执行的命令为:

① 创建迷宫      ② 求解迷宫      ③ 输出迷宫的解

### 3. 概要设计

#### 3.1 抽象数据类型定义

(1) 设定栈的抽象数据类型定义:

ADT Stack{

数据对象:  $D = \{a_i \mid a_i \in \text{CharSet}, i = 1, 2, \dots, n, n \geq 0\}$

数据关系:  $R_1 \{ \langle a(i-1) \rangle \mid a(i-1), a_i \in D, i = 2, 3 \dots n \}$

基本操作:

InitStack(&S)

操作结果: 构造一个空栈 S。

DestroyStack(&S)

初始结果: 栈 S 已存在。

操作结果: 销毁栈 S。

ClearStack(&S)

初始结果: 栈 S 已存在。

操作结果: 将 S 清为空栈。

StackLength(S)

初始结果: 栈 S 已存在。

操作结果: 返回栈 S 的长度

StackEmpty(S)

初始条件: 栈 S 已存在。

操作结果: 若 S 为空栈, 则返回 TRUE, 否则返回 FALSE

GetTop(S,&e)

初始条件: 栈 S 已存在。

操作结果: 若栈 S 不空, 则以 e 返回栈顶元素 e。

Push(&S,e)

初始条件: 栈 S 已存在。

操作结果: 在栈 S 的栈顶插入新的栈顶元素。

Pop(&S,&e)

初始条件: 栈 S 已存在。

操作结果: 删除 S 的栈顶元素, 并以 e 返回其值。

StackTraverse(S,visit())

初始条件: 栈 S 已存在。

操作结果: 从栈底到栈顶依次对 S 中的每个元素调用 visit()

}ADT Stack

(2) 设定迷宫的抽象数据类型为:

ADT maze{

数据对象:  $D = \{a(i,j) \mid a(i,j) \in \{0, 1\}, 0 \leq i \leq m+1, 0 \leq j \leq n+1, m, n \leq 10\}$

数据关系:  $R = \{M, N\}$

$M = \{ \langle a(i-1,j), a(i,j) \rangle \mid a(i-1,j), a(i,j) \in D, i=1, 2, \dots, m+1, j=0, 1, \dots, n+1 \}$

$N = \{ \langle a(i,j-1), a(i,j) \rangle \mid a(i,j-1), a(i,j) \in D, i=0, 1, \dots, m+1, j=1, 2, \dots, n+1 \}$

基本操作:

InitMaze(&M,maze,m,n)

初始条件: 二维数组 maze[m+1][n+1] 已存在, 其中自第一行至第 m+1 行、每行中自第一列至第 n+1 列的元素已有值, 并且以值 0 表示通路, 以值 1 表示障碍。

操作结果: 构成迷宫的字符型数组, 以字符 0 表示通路, 以字符 1 障碍, 并在迷宫四周加上一圈障碍。

MazePath(&M)

初始条件: 迷宫 M 已被赋值。

操作结果: 若迷宫 M 中存在一条通路, 则按如下规定改变迷

宫 M 的状态：以数字 0 代表可通过，数字 1 代表不可通过。

## 3.2 模块划分

### (1) 主程序模块：

```
void main()
{
    int sto[M][N];
    struct mark start,end; //start,end 入口和出口的坐标
    int add[4][2]={ {0,1},{1,0},{0,-1},{-1,0}}; //行增量和列增量 方向依次为东西南北
    initmaze(sto); //建立迷宫
    printf("输入入口的横坐标,纵坐标[逗号隔开]\n");
    scanf("%d,%d",&start.x,&start.y);
    printf("输入出口的横坐标,纵坐标[逗号隔开]\n");
    scanf("%d,%d",&end.x,&end.y);
    MazePath(start,end,sto,add); //find path
    system("PAUSE");
}
```

### (2) 栈模块——实现栈抽象数据类型；

### (3) 迷宫模块——实现迷宫抽象数据类型，建立迷宫，求解迷宫的一条路径。

## 4. 详细设计

### 4.1 数据类型的定义

#### (1) 坐标位置，类型：

```
struct mark //定义迷宫内点的坐标类型
{
    int x;
    int y;
};
```

#### (2) 迷宫类型

```
void initmaze(int maze[M][N])
//按照用户输入的 M 行和 N 列的二维数组（元素值为 0 或 1）
//设置迷宫 maze 的初值，包括加上边缘一圈的值
void MazePath(struct mark start,struct mark end,int maze[M][N],int
diradd[4][2])
//求解迷宫 maze 中从入口 Start 到出口 end 的一条路径
//若存在，则返回 TRUE，否则返回 FALSE
```

#### (3) 栈类型

```
struct Element
{
    int x,y; //x 行,y 列
```

```

int d; //d 下一步的方向
};
typedef struct LStack //链栈
{
    Element elem;
    struct LStack *next;
}*PLStack;

```

## 4.2 主要模块的算法描述

### (1) 求迷宫路径的伪码算法

```

void MazePath(struct mark start, struct mark end, int maze[M][N], int
diradd[4][2])
{
    int i, j, d; int a, b;
    Element elem, e;
    PLStack S1, S2;
    InitStack(S1);
    InitStack(S2);
    maze[start.x][start.y]=2; //入口点作上标记
    elem.x=start.x;
    elem.y=start.y;
    elem.d=-1; //开始为-1
    Push(S1, elem);
    while(!StackEmpty(S1)) //栈不为空 有路径可走
    {
        Pop(S1, elem);
        i=elem.x;
        j=elem.y;
        d=elem.d+1; //下一个方向
        while(d<4) //试探东南西北各个方向
        {
            a=i+diradd[d][0];
            b=j+diradd[d][1];
            if(a==end.x && b==end.y && maze[a][b]==0) //如果到了出口
            {
                elem.x=i;
                elem.y=j;
                elem.d=d;
                Push(S1, elem);
                elem.x=a;
                elem.y=b;
                elem.d=886; //方向输出为-1 判断是否到了出口
                Push(S1, elem);
                printf("\n0=东 1=南 2=西 3=北 886 为则走出迷宫\n\n 通路为:(行坐标,列坐

```

```

    标,方向)\n");
    while(S1) //逆置序列 并输出迷宫路径序列
    {
        Pop(S1,e);
        Push(S2,e);
    }
    while(S2)
    {
        Pop(S2,e);
        printf("-->(%d,%d,%d)",e.x,e.y,e.d);
    }
    return; //跳出两层循环, 本来用 break,但发现出错, exit 又会结束程序, 选用
    return 还是不错的 o(∩_∩)o...
}
if(maze[a][b]==0) //找到可以前进的非出口的点
{
    maze[a][b]=2; //标记走过此点
    elem.x=i;
    elem.y=j;
    elem.d=d;
    Push(S1,elem); //当前位置入栈
    i=a; //下一点转化为当前点
    j=b;
    d=-1;
}
d++;
}
}
printf("没有找到可以走出此迷宫的路径\n");
}

```

(2) 建立迷宫的伪码算法

```

void initmaze(int maze[M][N])
{
    int i,j;
    int m,n; //迷宫行,列
    printf("请输入迷宫的行数 m=");
    scanf("%d",&m);
    printf("请输入迷宫的列数 n=");
    scanf("%d",&n);
    printf("\n 请输入迷宫的各行各列:\n 用空格隔开,0 代表路,1 代表墙\n",m,n);
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
            maze[i][j] = (int) (rand() % 2);
    //scanf("%d",&maze[i][j]);
}

```

```

printf("你建立的迷宫为 o(∩_∩)o...\n");
for(i=0;i<=m+1;i++) //加一圈围墙
{
    maze[i][0]=1;
    maze[i][n+1]=1;
}
for(j=0;j<=n+1;j++)
{
    maze[0][j]=1;
    maze[m+1][j]=1;
}
for(i=0;i<=m+1;i++) //输出迷宫
{
    for(j=0;j<=n+1;j++)
    printf("%d ",maze[i][j]);
    printf("\n");
}
}

```

(3) 主函数的伪码算法:

```

void main()
{
    int sto[M][N];
    struct mark start,end; //start,end 入口和出口的坐标
    int add[4][2]={0,1},{1,0},{0,-1},{-1,0}}; //行增量和列增量 方向依次为东西
    南北
    initmaze(sto); //建立迷宫
    printf("输入入口的横坐标,纵坐标[逗号隔开]\n");
    scanf("%d,%d",&start.x,&start.y);
    printf("输入出口的横坐标,纵坐标[逗号隔开]\n");
    scanf("%d,%d",&end.x,&end.y);
    MazePath(start,end,sto,add); //find path
    system("PAUSE");
}

```

(4) 栈的初始化、判空、进栈、和出栈等的伪码算法:

```

int InitStack(PLStack &S) //构造空栈
{
    S=NULL;
    return 1;
}
int StackEmpty(PLStack S) //判断栈是否为空
{
    if(S==NULL)
    return 1;
    else

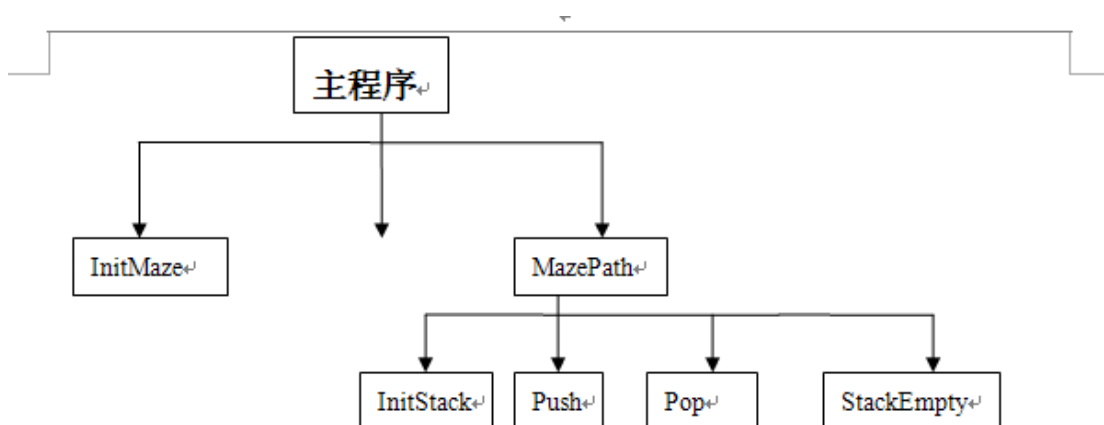
```

```

return 0;
}
int Push(PLStack &S, Element e)//压入新数据元素
{
    PLStack p;
    p=(PLStack)malloc(sizeof(LStack));
    p->elem=e;
    p->next=S;
    S=p;
    return 1;
}
int Pop(PLStack &S,Element &e) //栈顶元素出栈
{
    PLStack p;
    if(!StackEmpty(S))
    {
        e=S->elem;
        p=S;
        S=S->next;
        free(p);
        return 1;
    }
    else
        return 0;
}

```

### 4.3 函数之间的调用关系




## 5. 程序运行说明与测试

### 5.1 用户手册



- (1) 本程序的运行环境为 C 语言的运行环境，新建文件的后缀名：.cpp
- (2) 进入演示程序后，即显示以下的用户界面：



- ①、点击组建进行“编译”；
- ②、点击  运行程序；
- (3) 执行程序出现界面后，按照提示先迷宫的行数和列数回车，进一步输入迷宫内的数据‘1’代表墙，‘0’代表路径回车；输入迷宫的“入口”和“出口”回车即可现实路径，或者显示没有路径可走。

## 5.2 测试数据

- (1) 输入行数：3；输入列数：2

输入迷宫数据为：

```
0 0
0 0
0 0
```

入口位置：1 1

出口位置：3 2

- (2) 输入行数：3；输入列数：4

输入迷宫数据：

```
0 0 0 0
0 0 1 1
0 0 0 0
```

入口位置：1 1

出口位置：3 4

- (3) 输入行数：4；输入列数：9

输入迷宫数据：

```
0 0 0 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0
0 0 1 1 1 0 0 1 1
0 0 1 1 1 0 1 0 0
```

入口位置：1 1

出口位置：4 9

- (4) 输入行数：8；输入列数：9

输入迷宫数据：

```

0 0 1 0 0 0 1 0
0 0 1 0 0 0 1 0
0 0 0 0 1 1 0 1
0 1 1 1 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 1
0 1 1 1 1 0 0 1
1 1 0 0 0 1 0 1
1 1 0 0 0 0 0 0

```

入口位置: 1 1

出口位置: 8 9

### 5.3 测试结果及分析

(1)

```

"D:\C程序\fds\Debug\dsa.exe"
请输入迷宫的行数 m=3
请输入迷宫的列数 n=2

请输入迷宫的各行各列:
用空格隔开,0代表路,1代表墙
0 0
0 0
0 0
你建立的迷宫为o<┌┐┐>o...
1 1 1 1
1 0 0 1
1 0 0 1
1 0 0 1
1 1 1 1
输入入口的横坐标,纵坐标[逗号隔开]
1,1
输入出口的横坐标,纵坐标[逗号隔开]
3,2

0=东 1=南 2=西 3=北 886为则走出迷宫

通路为:<行坐标,列坐标,方向>
--><1,1,0>--><1,2,1>--><2,2,1>--><3,2,886>请按任意键继续. . .
Press any key to continue

```

(2)

```
"D:\C程序\fds\Debug\dsa.exe"
请输入迷宫的行数 m=3
请输入迷宫的列数 n=4

请输入迷宫的各行各列:
用空格隔开,0代表路,1代表墙
0 0 0 0
0 0 1 1
0 0 0 0
你建立的迷宫为o<┐└┐>o...
1 1 1 1 1 1
1 0 0 0 0 1
1 0 0 1 1 1
1 0 0 0 0 1
1 1 1 1 1 1
输入入口的横坐标,纵坐标[逗号隔开]
1,1
输入出口的横坐标,纵坐标[逗号隔开]
3,4

0=东 1=南 2=西 3=北 886为则走出迷宫

通路为:<行坐标,列坐标,方向>
--><1,1,0>--><1,2,1>--><2,2,1>--><3,2,0>--><3,3,0>--><3,4,886>请按任意键继续. .
Press any key to continue
```

(3)

```
"D:\C程序\fds\Debug\dsa.exe"
请输入迷宫的行数 m=4
请输入迷宫的列数 n=9

请输入迷宫的各行各列:
用空格隔开,0代表路,1代表墙
0 0 0 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0
0 0 1 1 1 0 0 1 1
0 0 1 1 1 0 1 0 0
你建立的迷宫为o<┐└┐>o...
1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1 0 0 1
1 0 1 0 0 0 1 0 0 0 1
1 0 0 1 1 1 0 0 1 1 1
1 0 0 1 1 1 0 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1
输入入口的横坐标,纵坐标[逗号隔开]
1,1
输入出口的横坐标,纵坐标[逗号隔开]
4,9
没有找到可以走出此迷宫的路径
请按任意键继续. . .
Press any key to continue
```

(4)

```
"D:\C程序\fds\Debug\dsa.exe"
请输入迷宫的行数 m=8
请输入迷宫的列数 n=9

请输入迷宫的各行各列:
用空格隔开,0代表路,1代表墙
0 0 1 0 0 0 1 0
0 0 1 0 0 0 1 0
0 0 0 0 1 1 0 1
0 1 1 1 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 1
0 1 1 1 1 0 0 1
1 1 0 0 0 1 0 1
1 1 0 0 0 0 0 0
你建立的迷宫为o<□_□>o...
1 1 1 1 1 1 1 1 1 1 1
1 0 0 1 0 0 0 1 0 0 1
1 0 1 0 0 0 1 0 0 0 1
1 0 0 1 1 0 1 0 1 1 1
1 1 0 0 1 0 0 0 0 1 1
1 0 0 0 0 0 1 0 0 0 1
1 1 0 1 0 1 1 1 1 0 1
1 0 1 1 1 0 0 0 1 0 1
1 1 1 1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1
输入入口的横坐标,纵坐标t逗号隔开t
1,1
输入出口的横坐标,纵坐标t逗号隔开t
8,9

0=东 1=南 2=西 3=北 886为则走出迷宫

通路为:<行坐标,列坐标,方向>
--><1,1,1>--><2,1,1>--><3,1,0>--><3,2,1>--><4,2,0>--><4,3,1>--><5,3,0>--><5,4,0>
--><5,5,3>--><4,5,0>--><4,6,0>--><4,7,0>--><4,8,1>--><5,8,0>--><5,9,1>--><6,9,1>
--><7,9,1>--><8,9,886>请按任意键继续. . .
Press any key to continue
```

## 6. 实验总结

- (1) 本次算法涉及迷宫的求解路径和迷宫的建立函数;建立迷宫函数时可自由确定迷宫的行数和列数,及迷宫内的数据也可自由输入;但在建立迷宫数据时一开始使用的是随机函数,但是用此方法生成的迷宫行数列数一致时迷宫也是一致,且找不着路径;
- (2) 本题中的主要算法: MazePath 和 initmaze 的时间复杂度为  $O(m*n)$ , 本题的空间复杂度异为  $O(m*n)$  (栈所占的最大空间)

## 附: 源程序清单

```
#include<stdio.h>
#include<stdlib.h>
#define M 15
#define N 15
struct mark //定义迷宫内点的坐标类型
```

```

{
int x;
int y;
};
struct Element //"恋"栈元素，嘿嘿。。
{
int x,y; //x 行,y 列
int d; //d 下一步的方向
};
typedef struct LStack //链栈
{
Element elem;
struct LStack *next;
}*PLStack;
/*****栈函数*****/
int InitStack(PLStack &S)//构造空栈
{
S=NULL;
return 1;
}
int StackEmpty(PLStack S)//判断栈是否为空
{
if(S==NULL)
return 1;
else
return 0;
}
int Push(PLStack &S, Element e)//压入新数据元素
{
PLStack p;
p=(PLStack)malloc(sizeof(LStack));
p->elem=e;
p->next=S;
S=p;
return 1;
}
int Pop(PLStack &S,Element &e) //栈顶元素出栈
{
PLStack p;
if(!StackEmpty(S))
{
e=S->elem;
p=S;
S=S->next;
}
}

```

```

free(p);
return 1;
}
else
return 0;
}
/*****求迷宫路径函数*****/
void MazePath(struct mark start,struct mark end,int maze[M][N],int diradd[4][2])
{
int i,j,d;int a,b;
Element elem,e;
PLStack S1, S2;
InitStack(S1);
InitStack(S2);
maze[start.x][start.y]=2; //入口点作上标记
elem.x=start.x;
elem.y=start.y;
elem.d=-1; //开始为-1
Push(S1,elem);
while(!StackEmpty(S1)) //栈不为空 有路径可走
{
Pop(S1,elem);
i=elem.x;
j=elem.y;
d=elem.d+1; //下一个方向
while(d<4) //试探东南西北各个方向
{
a=i+diradd[d][0];
b=j+diradd[d][1];
if(a==end.x && b==end.y && maze[a][b]==0) //如果到了出口
{
elem.x=i;
elem.y=j;
elem.d=d;
Push(S1,elem);
elem.x=a;
elem.y=b;
elem.d=886; //方向输出为-1 判断是否到了出口
Push(S1,elem);
printf("\n0=东 1=南 2=西 3=北 886 为则走出迷宫\n\n 通路为:(行坐标,列坐标,方向)\n");
while(S1) //逆置序列 并输出迷宫路径序列
{
Pop(S1,e);
Push(S2,e);
}
}
}
}

```

```

}
while(S2)
{
Pop(S2,e);
printf("-->(%d,%d,%d)",e.x,e.y,e.d);
}
return; //跳出两层循环，本来用 break,但发现出错，exit 又会结束程序，选用 return 还是不错的
滴 o(∩_∩)o...
}
if(maze[a][b]==0) //找到可以前进的非出口的点
{
maze[a][b]=2; //标记走过此点
elem.x=i;
elem.y=j;
elem.d=d;
Push(S1,elem); //当前位置入栈
i=a; //下一点转化为当前点
j=b;
d=-1;
}
d++;
}

printf("没有找到可以走出此迷宫的路径\n");
}
/*****建立迷宫*****/
void initmaze(int maze[M][N])
{
int i,j;
int m,n; //迷宫行,列
printf("请输入迷宫的行数 m=");
scanf("%d",&m);
printf("请输入迷宫的列数 n=");
scanf("%d",&n);
printf("\n 请输入迷宫的各行各列:\n 用空格隔开,0 代表路,1 代表墙\n",m,n);
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
maze[i][j] = (int) (rand() % 2);
//scanf("%d",&maze[i][j]);
printf("你建立的迷宫为 o(∩_∩)o...\n");
for(i=0;i<=m+1;i++) //加一圈围墙
{
maze[i][0]=1;
maze[i][n+1]=1;

```

```

    }
    for(j=0;j<=n+1;j++)
    {
        maze[0][j]=1;
        maze[m+1][j]=1;
    }
    for(i=0;i<=m+1;i++) //输出迷宫
    {
        for(j=0;j<=n+1;j++)
        printf("%d ",maze[i][j]);
        printf("\n");
    }

void main()
{
    int sto[M][N];
    struct mark start,end; //start,end 入口和出口的坐标
    int add[4][2]={0,1},{1,0},{0,-1},{-1,0}}; //行增量和列增量 方向依次为东西南北
    initmaze(sto); //建立迷宫
    printf("输入入口的横坐标,纵坐标[逗号隔开]\n");
    scanf("%d,%d",&start.x,&start.y);
    printf("输入出口的横坐标,纵坐标[逗号隔开]\n");
    scanf("%d,%d",&end.x,&end.y);
    MazePath(start,end,sto,add); //find path
    system("PAUSE");
}

```



## 第 3 题：哈夫曼编码/译码器

### 1. 实验内容

#### 【问题描述】

设计一个哈夫曼编码/译码系统。

#### 【基本要求】

系统应具备以下功能：

- (1) 识别信息：以文件或直接输入等方式对信息进行识别。
- (2) 显示编码规则：在屏幕上显示编码规则。
- (3) 编码：利用已建好的哈夫曼树对文本文件中的正文进行编码，然后将结果输出。
- (4) 译码：利用已建好的哈夫曼树对编码信息进行译码，在屏幕上显示结果。
- (5) 退出：结束程序。

### 2. 需求分析

#### (1) 程序的基本功能

本程序可以对任何大小的字符型文件进行 Huffman 编码，生成一个编码文件。并可以在程序运行结束后的任意时间对它解码还原生成字符文件。即：读出文件中待编码的字符，并实现 Huffman 编码，然后对 Huffman 编码生成的代码串进行译码，最后输出电文数字。

#### (2) 输入/输出形式

当进行编码时从原文件中提取。当进行译码时输入为编码文件名，从文件中读取 Huffman 编码树后输入字符文件的文件名。

#### (3) 测试数据要求

本程序中测试数据主要为字符型文件。涵盖数字，字母及特殊符号。

### 3. 概要设计

#### 3.1 抽象数据类型定义

ADT Haffman{

**数据对象 D:** D 是具有相同特性的数据元素的集合。

**数据关系 R:** 若 D 为空集, 则成为空树;

若 D 仅含一个数据元素, 则 R 为空集, 否则  $R = \{H\}$ , H 是如下二元关系;

- (1) 在 D 中存在惟一的称为根的数据元素 root, 它在关系 H 下无前驱;
- (2) 若  $D - \{root\} \neq \Phi$ , 则存在  $D - \{root\}$  的一个划分  $D_1, D_2, \dots, D_m (m > 0)$ , 对任意  $j \neq k (1 \leq j, k \leq m)$  有  $D_j \cap D_k = \Phi$ , 且对任意的  $i (1 \leq i \leq m)$ , 惟一存在数据元素  $x_i \in D_i$ , 有  $\langle root, x_i \rangle \in H$ ;
- (3) 对应于  $D - \{root\}$  的划分,  $H - \{\langle root, x_i \rangle, \dots, \langle root, x_m \rangle\}$  有惟一的一个划分  $H_1, H_2, \dots, H_m (m > 0)$ , 对任意  $j \neq k (1 \leq j, k \leq m)$  有  $H_j \cap H_k = \Phi$ , 且对任意  $i (1 \leq i \leq m)$ ,  $H_i$  是  $D_i$  上的二元关系,  $(D_i, \{H_i\})$  是一棵符合本定义棵树, 称为根 root 的子树。

**基本操作 P:**

Reading\_file(&Message);

初始条件: 文件存在。

操作结果: 从文件中读取信息。

Writing\_file(&Message);

初始条件: 文件存在。

操作结果: 将信息写进文件。

Total\_message(&Message, &Total);

初始条件: 文件存在。

操作结果: 统计信息中各字符的次数。

HaffmanTree(&Total, HNodeType);

初始条件: 文件存在, HNodeType 是数组。

操作结果: 构建哈夫曼树。

HaffmanCode(HNodeType, HCodetype, &Total);

初始条件: 文件存在, HNodeType, Hcodetype 是数组。

操作结果: 建立哈夫曼编码。

Writing\_HCode(HNodeType, HCodetype, &Total);

初始条件: 文件存在, HNodeType, Hcodetype 是数组。

操作结果: 将编码规则写进文件

Lock(&Message, HNodeType, HCodetype, &Total, &Locking);

初始条件: 文件存在, HNodeType, Hcodetype 是数组。

操作结果: 给文件信息加密编码

Writing\_lock(&Locking);

初始条件: 文件存在。

操作结果: 将已编码信息写进文件。

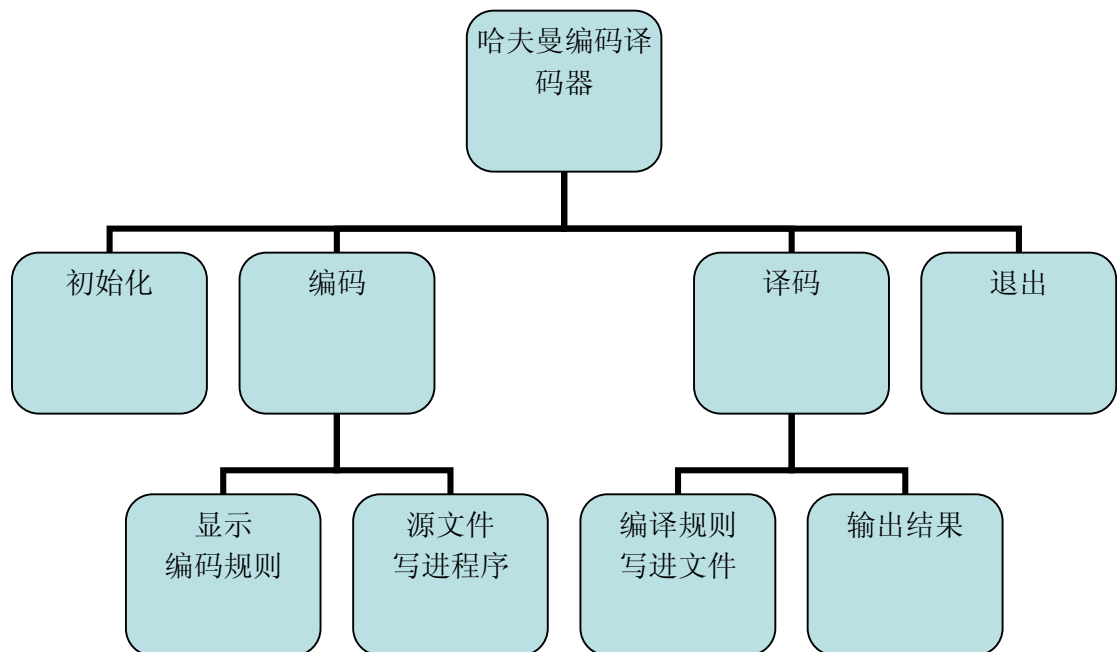
```
Writing_translate(&Locking,Hcodetype,HNode,HNode,&Total);
```

初始条件：文件存在，HNode,Hcodetype 是数组。

操作结果：将已编码信息翻译过来并写进文件}ADT Haffman

## 3.2 模块划分

### (1) 系统结构图（功能模块图）



### (2) 功能模块说明

- 1) .编码：从原文件中读取信息→建立哈弗曼树→对文本进行哈弗曼编码并输出编码→保存编码文件；
- 2) .译码：利用建立好的哈弗曼树进行译码→将译码输出→保存译码文件；
- 3) .退出：退出系统，返回源程序，程序运行结束。

## 4. 详细设计

### 4.1 数据类型的定义

```
typedef struct
{
    char data;//字符值
    int num;//某个值的字符出现的次数
}TotalNode;
//统计结点，包括字符种类和出现次数
typedef struct
{
```

```

    TotalNode tot[300]; //统计结点数组
    int num; //统计数组中含有的字符个数
}Total; //统计结构体，包括统计数组和字符种类数
typedef struct
{
    char mes[300]; //字符数组
    int num; //总字符数
}Message; //信息结构体，包括字符数组和总字符数
typedef struct{
    int locked[500]; //密码数组
    int num; //密码总数
}Locking; //哈夫曼编码后的密文信息
typedef struct
{
    char data; //字符
    int weight; //权值
    int parent; //双亲结点在数组 HuffNode[] 中的序号
    int lchild; //左孩子结点在数组 HuffNode[] 中的序号
    int rchild; //右孩子结点在数组 HuffNode[] 中的序号
}HNodeType; //哈夫曼树结点类型，包括左右孩子，权值和信息
typedef struct
{
    int bit[MAXBIT];
    int start;
}HCodetype; //哈夫曼编码结构体，包括编码数组和起始位

```

## 4.2 主要模块的算法描述

```

(1) 、int main()
{
    int i, j, choice, mark=0; //mark 标记文件信息是否读入到内存中
    int n;
    HNodeType HuffNode[500]; //保存哈夫曼树中各结点信息
    HCodetype HuffCode[300];
    Locking *locking;
    Total *total;
    Message *message;
    locking=new Locking;
    locking->num=0;
    total=new Total;
    total->num=0;
    message=new Message;
    message->num=0; //初始化变量
    while(1)
    {

```

```

        cout<<"*****";
        cout<<"*1: 从文件读取信息          2: 显示编码规则          3: 将
原文件信息写进文件          *";
        cout<<"*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将
译码后的信息写进文件          *";
        cout<<"*7          :          退          出
*****";
        ***"<<endl;
        cout<<"请输入操作代码: ";
        cin>>choice;
        switch(choice)
{
    case 1:
        reading_file(message);//从文件中读取信息
        for(n=0;n<message->num;n++)
            printf("%c", message->mes[n]);
        printf("\n");
        mark=1;
        break;
    case 2://显示编码规则
        if(mark==0) cout<<"请先从文件中读取信息!"<<endl;
        else
        {
            total_message(message, total);//统计信息中各字符的出现次数
            HaffmanTree(total, HuffNode);//构建哈夫曼树
            HaffmanCode(HuffNode, HuffCode, total);//建立哈夫曼编码
            for(i=0;i<total->num;i++)//显示编码规则
            {
                cout<<total->tot[i].data<<" ";
                for(j=HuffCode[i].start+1;j<total->num;j++)
                    cout<<HuffCode[i].bit[j];
                cout<<endl;
            }
        }
        break;
    case 3://将原文件信息写进文件
        if(mark==0) cout<<"请先从文件中读取信息!"<<endl;
        else
            writing_file(message);//将信息写进文件
        break;
    case 4://将编码规则写进文件
        if(mark==0) cout<<"请先从文件中读取信息!"<<endl;
        else

```

```

{
    total_message(message, total); //统计信息中各字符的出现次数
    HaffmanTree(total, HuffNode); //构建哈夫曼树
    HaffmanCode(HuffNode, HuffCode, total); //建立哈夫曼编码
    writing_HCode(HuffNode, HuffCode, total); //将编码规则写进文件
}

    break;
case 5: //将编码后的信息写进文件
    if(mark==0) cout<<"请先从文件中读取信息!"<<endl;
    else
    {
        total_message(message, total); //统计信息中各字符的出现次数
        HaffmanTree(total, HuffNode); //构建哈夫曼树
        HaffmanCode(HuffNode, HuffCode, total); //建立哈夫曼编码
        lock(message, HuffNode, HuffCode, total, locking); //给文件信息
加密编码
        writing_lock(locking); //将已编码信息写进文件
    }

    for(n=0; n<locking->num; n++) //写文件
        if(locking->locked[n]==-1)
            printf(" ");
        else
            cout<<locking->locked[n];
        printf("\n");
        break;
case 6: //将译码后的信息写进文件
    if(mark==0) cout<<"请先从文件中读取信息!"<<endl;
    else
    {
        total_message(message, total); //统计信息中各字符的出现次数
        HaffmanTree(total, HuffNode); //构建哈夫曼树
        HaffmanCode(HuffNode, HuffCode, total); //建立哈夫曼编码
        writing_translate(locking, HuffCode, HuffNode, total);
        //将已编码信息翻译过来并写进文件
    }

    reading_file1(message);
    for(n=0; n<message->num; n++) //写文件
        if(message->mes[n]==-1)
            printf(" ");
        else
            cout<<message->mes[n];
        printf("\n");
        break;
case 7:

```

```

        exit(1);
    default:
        cout<<"输入错误, 请重新选择"<<endl;
    }
}

for(i=0;i<locking->num;i++)
    if(locking->locked[i]==-1)cout<<" ";
    else
        cout<<locking->locked[i];
        cout<<endl;
for(i=0;i<total->num;i++)
    cout<<total->tot[i].data<<" "<<total->tot[i].num<<endl;
    for(i=0;i<2*(total->num)-1;i++)
        cout<<HuffNode[i].parent<<" ";
cout<<endl;
return 0;
}

(2)、void reading_file(Message *message)
{
/*打开 reading 文件, 失败则结束。不断读取字符并保存进 message 数组中,
直到遇到#结束, 记录字符总数*/
int i=0;
char ch;
ifstream infile("c:\\reading.txt", ios::in|ios::out);
if(!infile)//打开失败则结束
{
    cout<<"打开 c:\\reading.txt 文件失败"<<endl;
    exit(1);
}
else
    cout<<"读取文件成功"<<endl;
while(infile.get(ch) && ch!='#')//读取字符直到遇到#
{
    message->mes[i]=ch;
    i++;
}
message->num=i;//记录总字符数
infile.close();//关闭文件
} //从文件中读取信息

(3)、void reading_file1(Message *message)
{
    int i=0;
    char ch;
    ifstream infile("c:\\translate.txt", ios::in|ios::out);

```

```

        if(!infile)//打开失败则结束
        {
            cout<<"打开 c:\\translate.txt 文件失败"<<endl;
            exit(1);
        }
        else
            cout<<"读取文件成功"<<endl;
        while(infile.get(ch) && ch!='#')//读取字符直到遇到#
        {
            message->mes[i]=ch;
            i++;
        }
        message->num=i;//记录总字符数
        infile.close();//关闭文件
    }//从文件中读取信息
(4)、void writing_file(Message *message)//将信息写进文件
    { /*打开 writing 文件，失败则结束。将信息写进文件*/
        int i;
        ofstream outfile("c:\\writing.txt", ios::in|ios::out);
        if(!outfile)//打开失败则结束
        {
            cout<<"打开 c:\\writing.txt 文件失败"<<endl;
            exit(1);
        }
        for(i=0;i<message->num;i++)//写文件
            outfile.put(message->mes[i]);
        cout<<"信息写进文件成功"<<endl;
        outfile.close();//关闭文件
    }//将原信息写入文件
(5)、void total_message(Message *message, Total *total)
    { /*将 message 中的字符种类及出现次数统计保存到 total 数组中，重复字
      符用 mark 标记，
      否则新建字符种类。记录下字符种类的个数*/
        int i, j, mark;
        for(i=0;i<message->num;i++)//遍历 message 中的所有字符信息
        {
            if(message->mes[i]!=' ')//字符不为空格时
            {
                mark=0;
                for(j=0;j<total->num;j++)//在 total 中搜索当前字符
                    if(total->tot[j].data==message->mes[i])//搜索到，则
                        此字符次数加 1，mark 标志为 1
                {
                    total->tot[j].num++;
                }
            }
        }
    }

```



```

        mark=1;
        break;
    }
    if(mark==0)//未搜索到，新建字符种类，保存进 total 中，字
符类加 1
    {
        total->tot[total->num].data=message->mes[i];
        total->tot[total->num].num=1;
        total->num++;
    }
}
}
//统计信息中各字符的出现次数
(6)、void HaffmanTree(Total *total,HNodeType HuffNode[])
{ /*通过每次选取最小和次小两权值建立二叉树，最终构建成为哈夫曼树，
且左孩子权值比右孩子小*/
int i,j,min1,min2,x1,x2;
for(i=0;i<2*(total->num)-1;i++)//初始化哈夫曼树并存入叶子结点
权值和信息
{
    if(i<=total->num-1)HuffNode[i].data=total->tot[i].data;
    HuffNode[i].weight=total->tot[i].num;
    HuffNode[i].parent=-1;
    HuffNode[i].lchild=-1;
    HuffNode[i].rchild=-1;
}
for(i=0;i<total->num-1;i++)
{
    min1=min2=MAXVALUE;
    x1=x2=0;
    for(j=0;j<total->num+i;j++)//选取最小 x1 和次小 x2 的两权值
    {
        if(HuffNode[j].parent==-1&&HuffNode[j].weight<min1)
        {
            min2=min1;
            x2=x1;
            min1=HuffNode[j].weight;
            x1=j;
        }
        else
            if(HuffNode[j].parent==-1&&HuffNode[j].weight<min2)
                min2=HuffNode[j].weight;
        {
            x2=j;

```

```

        }
    }
    HuffNode[x1].parent=total->num+i;//修改亲人结点位置
    HuffNode[x2].parent=total->num+i;

    HuffNode[total->num+i].weight=HuffNode[x1].weight+HuffNode[x2].weight;
}
}
} //构建哈夫曼树
(7)、void HaffmanCode(HNodetype HuffNode[], HCodetype HuffCode[], Total
*total)
{
    /*在建立的哈夫曼树基础上，左分支为 0，右分支为 1，
    从叶结点向上直到根结点建立哈夫曼编码，并保存每个叶结点起始位*/
    int i, j, c, p;
    HCodetype cd;
    for(i=0; i<total->num; i++) //建立叶子结点个数的编码
    {
        cd.start=total->num-1; //起始位初始化
        p=HuffNode[i].parent;
        c=i;
        while(p!=-1) //从叶结点向上爬直到到达根结点
        {
            if(HuffNode[p].lchild==c) //左分支则为 0
                cd.bit[cd.start]=0;
            else
                cd.bit[cd.start]=1; //右分支则为 1
            cd.start--; //起始位向前移动
            c=p;
            p=HuffNode[p].parent;
        }
        for(j=cd.start+1; j<total->num; j++) //保存求出的每个叶结点编码和起始位
            HuffCode[i].bit[j]=cd.bit[j];
        HuffCode[i].start=cd.start;
    }
} //建立哈夫曼编码
(8) void writing_HCode(HNodetype HuffNode[], HCodetype HuffCode[], Total
*total)
{
    /*打开 HCode 文件，失败则结束。将信息写进文件*/
    int i, j;
    ofstream outfile("c:\\HCode.txt", ios::in|ios::out);

```

```

    if(!outfile)//打开失败则结束
    {
        cout<<"打开 c:\\HCode.txt 文件失败"<<endl;
        exit(1);
    }
    for(i=0;i<total->num;i++)//写文件
    {
        outfile.put(HuffNode[i].data);outfile<<" ";
        for(j=HuffCode[i].start+1;j<total->num;j++)
            outfile<<HuffCode[i].bit[j];
        outfile<<endl;
    }
    cout<<"编码规则写进文件成功"<<endl;
    outfile.close();//关闭文件
} //将编码规则写进文件
(9) 、 void lock(Message *message,HNodetype HuffNode[],HCodetype
HuffCode[],Total *total,Locking *locking)
{
    int i,j,m;
    for(i=0;i<message->num;i++)//遍历信息
    {if(message->mes[i]==' ')
    //碰到空格则以-1 形式保存进 locked 数组中
    {
        locking->locked[locking->num]=-1;
        locking->num++;
    }
    else
        for(j=0;j<total->num;j++)//搜索信息对应的编码
        {
            if(HuffNode[j].data==message->mes[i])//找到与信息对
应的编码则写进 locked 数组
                for(m=HuffCode[j].start+1;m<total->num;m++)
                {
                    locking->locked[locking->num]=HuffCode[j].bit[m];
                    locking->num++;//locked 数组总数加 1
                }
        }
    }
} //给文件信息加密编码
(10) 、 void writing_lock(Locking *locking)
{
    int i;
    ofstream outfile("c:\\locking.txt",ios::in|ios::out);

```

```

if(!outfile)//打开失败则结束
{
    cout<<"打开 c:\\locking.txt 文件失败"<<endl;
    exit(1);
}
for(i=0;i<locking->num;i++)//写文件
    if(locking->locked[i]==-1)
        outfile.put(' ');
    else
        outfile<<locking->locked[i];
cout<<"已编码信息写进文件成功"<<endl;
outfile.close();//关闭文件
} //将哈夫曼编码后的密文信息写进文件
( 11 ) 、 void writing_translate(Locking *locking,HCodetype
HuffCode[],HNodetype HuffNode[],Total *total)
{
    int i, j, mark[100], start[100], min, max;
    ofstream outfile("c:\\translate.txt", ios::in|ios::out); // 打开
文件
if(!outfile)//打开失败则结束
{
    cout<<"打开 c:\\translate.txt 文件失败"<<endl;
    exit(1);
}
for(i=0;i<total->num;i++)//start 数组初始化
{
    start[i]=HuffCode[i].start+1;
}
for(i=0;i<total->num;i++)//mark 数组初始化
{
    mark[i]=1;
}
min=0;
for(max=0;max<locking->num;max++)//写文件
{
    if(locking->locked[max]==-1)//碰到空格信息则直接输出空格
    {
        outfile.put(' ');
        min++;
    }
    else
    {
        for(i=min;i<=max;i++)//查找从 min 开始到 max 的编码对应的信息
        {

```

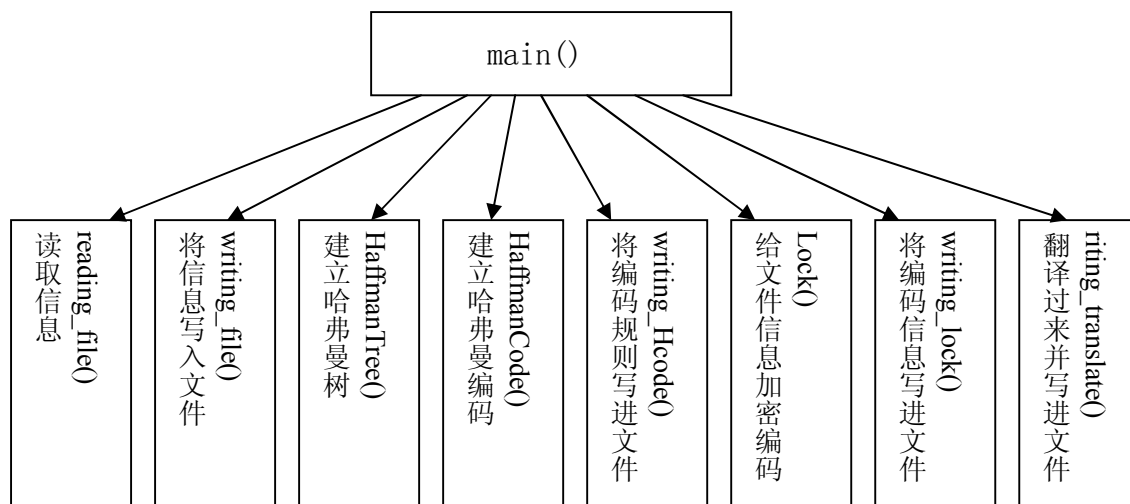
```

        for(j=0;j<total->num;j++)
        {
            if(start[j]==total->num+1)
                mark[j]=0;//对应编码比所给编码短则不在比较
            if(mark[j]==1&&locking->locked[i]==HuffCode[j].bit[start[j]])
                start[j]++;
            else
                mark[j]=0;
        }
    }
    for(i=0;i<total->num;i++)//找到对应信息，则写进文件
    {
        if(mark[i]==1&&start[i]==total->num)
        {
            outfile.put(HuffNode[i].data);
            break;
        }
    }
    if(i!=total->num)
        min=max+1;
    for(i=0;i<total->num;i++)//start 数组初始化
    {
        start[i]=HuffCode[i].start+1;
    }
    for(i=0;i<total->num;i++)//mark 数组初始化
    {
        mark[i]=1;
    }
}

cout<<"翻译信息写进文件成功"<<endl;
outfile.close();//关闭文件
} //将已编码信息翻译过来并写进文件

```

### 4.3 函数之间的调用关系



### 编码算法:

(1) 对输入的一段欲编码的字符串进行统计各个字符出现的次数，并它们转化为权值  $\{w_1, w_2, \dots, w_N\}$  构成  $n$  棵二叉树的集合  $F = \{T_1, T_2, \dots, T_n\}$  把它们保存到结构体数组  $HT[n]$  中, 其中  $\{T_i$  是按它们的 ASCII 码值先后排序。其中每棵二叉树  $T_i$  中只有一个带权为  $W_i$  的根结点的权值为其左、右子树上根结点的权值之和。

(2) 在  $HT[1..i]$  中选取两棵根结点的权值最小且没有被选过的树作为左右子树构造一棵新的二叉树，且置新的二叉树的根结点的权值为左、右子树上根结点的权值之和。

(3) 哈夫曼树已经建立后，从叶子到根逆向求每一个字符的哈夫曼编码。

### 译码算法:

译码的过程是分解电文中字符串，从根出发，按字符'0'，或'1'确定找左孩子或右孩子，直至叶子结点，便求的该子串相应字符并输出接着下一个字符。

## 5. 程序运行说明与测试

### 5.1 用户手册

注：该方法需建立文件，使用时需耐心。

1. 在源文件中输入待测信息并保存；
2. 根据屏幕提示，依次输入 1,2,3,4,5,6,7；

3. 输入结果显示在屏幕上，对应文件夹中也有相应信息；
4. 退出系统。

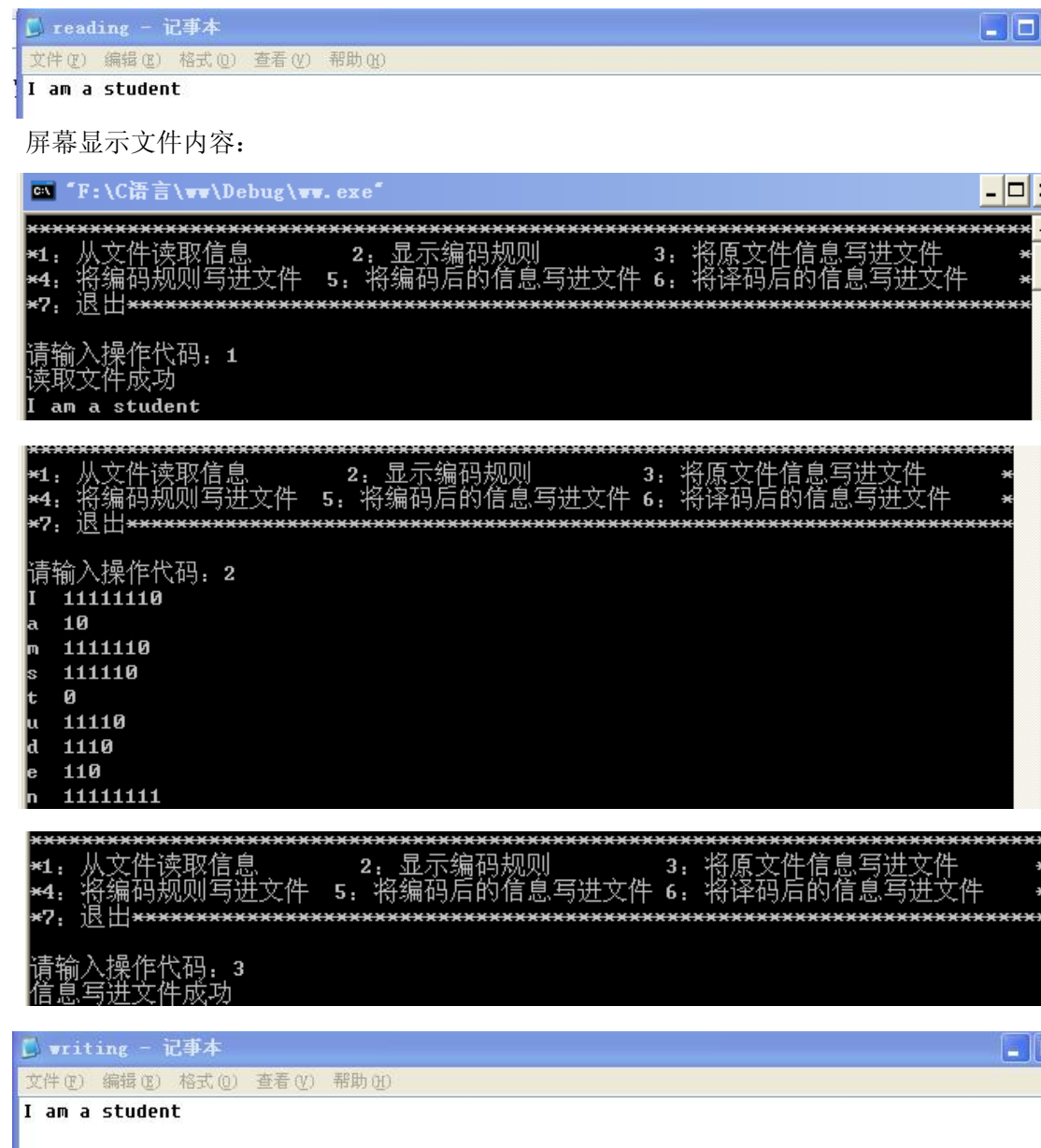
## 5.2 测试数据

数据一：I am a student； 数据二：12 3 456 7890；

数据三：# 12 %gh\*nj； 数据四：12 %gh\*nj；

## 5.3 测试结果及分析

测试结果 1:

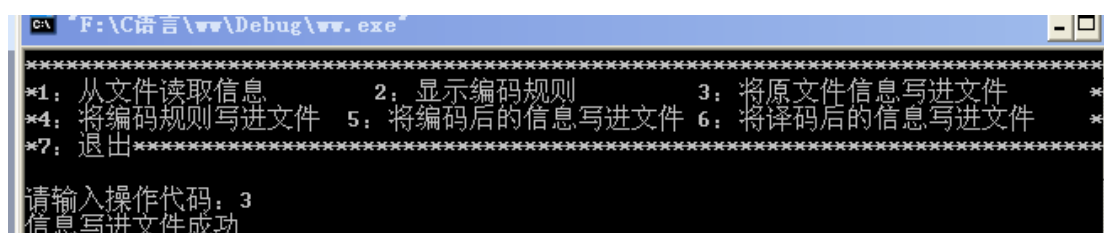
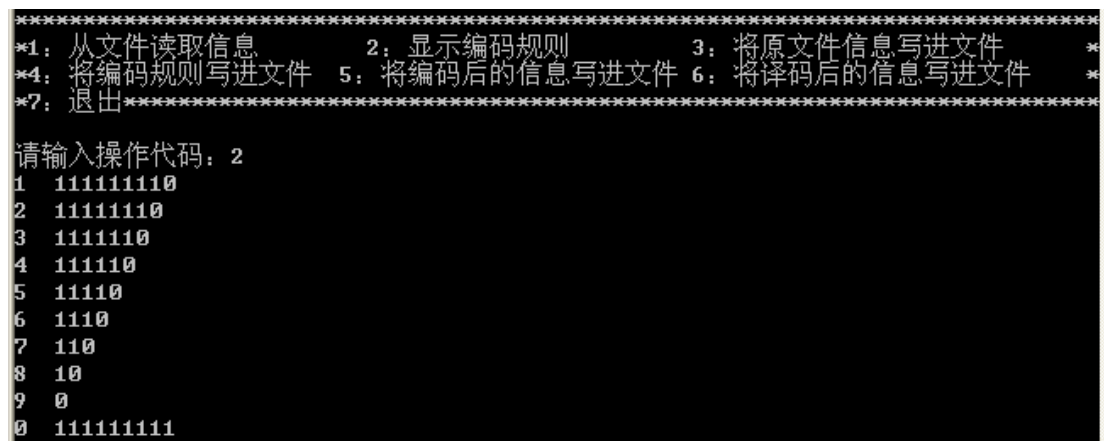
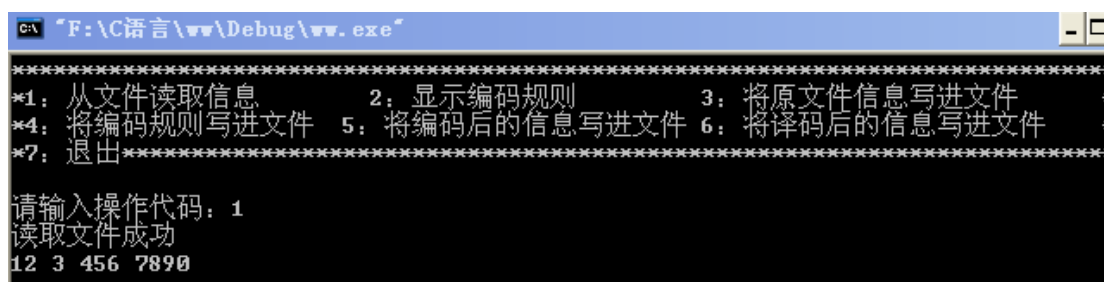




测试一旨在检测程序对字母的测试，结果如上图所示，截屏下方为对应文件夹情况。由于文件夹具有与屏幕显示同一性的特征，特仅在第一次测试结果中给出，后续测试结果仅给定初始源文件的截图。测试一结果证明改程序对输入的字母可以应用。

测试结果 2:





```
*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件  *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

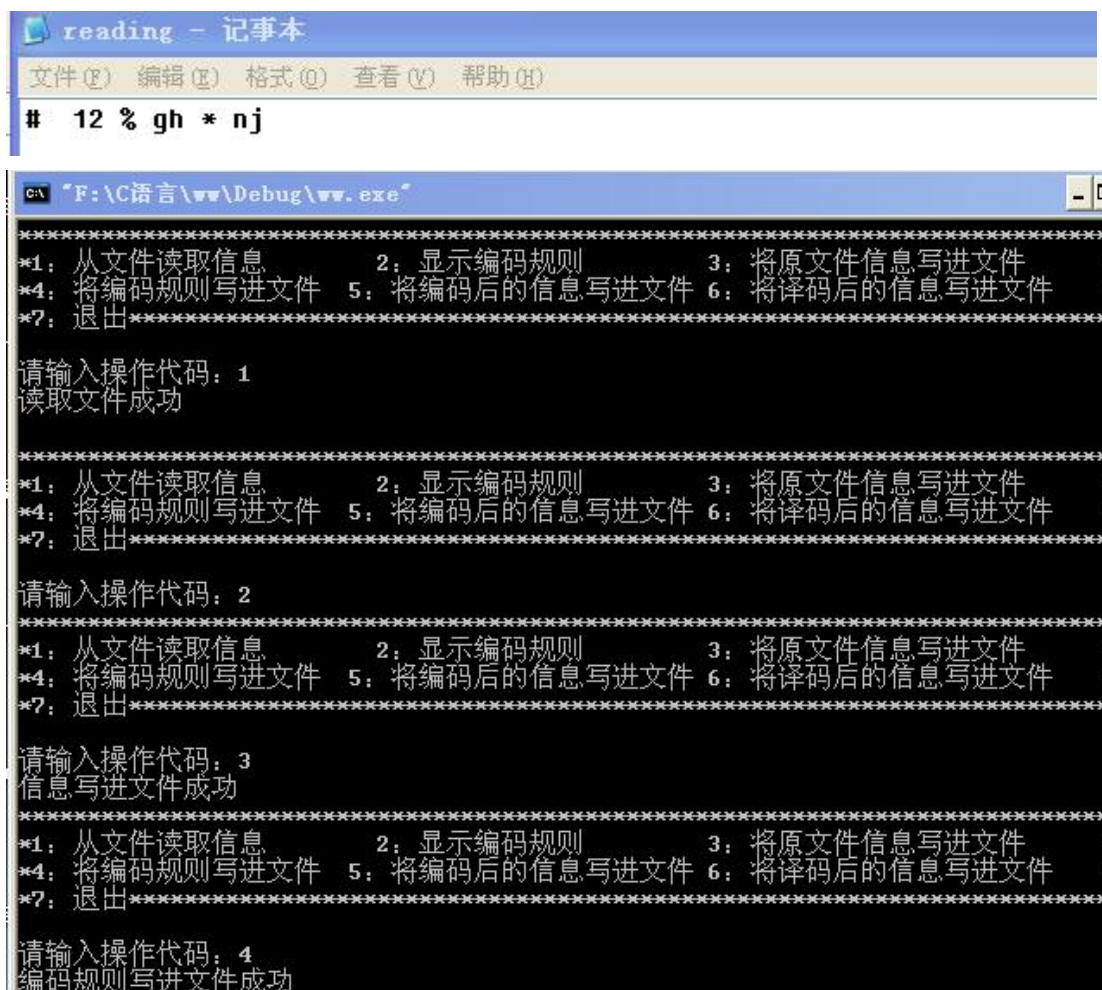
请输入操作代码: 6
翻译信息写进文件成功
读取文件成功
12 3 456 7890

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件  *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 7
Press any key to continue
```

测试二结果证明改程序对于数字信息适用。

测试结果 3:



```
reading - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

# 12 % gh * nj

C:\ "F:\C语言\ww\Debug\ww.exe"

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件  *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 1
读取文件成功

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件  *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 2

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件  *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 3
信息写进文件成功

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件  *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 4
编码规则写进文件成功
```

```
*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 5
已编码信息写进文件成功

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 6
翻译信息写进文件成功
读取文件成功

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 7
Press any key to continue
```

测试结果三屏幕无输出，其原因为源文件中信息一#打头，意味终止，程序虽运行，但无法对其后的信息进行测试，除源文件外，所有程序对应文件均为空。测试三证明#的作用。

测试结果 4:

```

C:\ "F:\C语言\ww\Debug\ww.exe"
*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 1
读取文件成功
12 % gh * nj
*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 2
1  1111110
2  111110
%  11110
g  1110
h  110
*  10
n  0
j  1111111
```

```
C:\ "F:\C语言\ww\Debug\ww.exe"

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 3
信息写进文件成功

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 4
编码规则写进文件成功

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 5
已编码信息写进文件成功
1111110111110 11110 1110110 10 01111111

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 6
翻译信息写进文件成功
读取文件成功
12 z gh * nj

*****
*1: 从文件读取信息      2: 显示编码规则      3: 将原文件信息写进文件      *
*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将译码后的信息写进文件 *
*7: 退出*****

请输入操作代码: 7
Press any key to continue
```

测试四证明程序对于特殊符号同样适用

## 6. 实验总结

- (1) 用户界面设计为“菜单”模式，使人们更加容易使用。
- (2) 源程序代码可以根据需要改变。
- (3) 在编程中使用了很不规范的编程方法，应用了一些临时变量来实现功能，而大量临时变量在代码中没有很好地进行命名。这给程序的阅读和维护带来了不方便。
- (4) 本程序仅能对源文件中的代码进行编译，具有小范围的局限性。
- (5) 本程序需建立多个文本文档，缺少灵活性，运行时需耐心，这是本程序最大的不足。

(6) 设计中得到了老师和同学的帮助，并参考了网络上的优秀论文和纸质文件，使程序设计能够较为顺利的进行下去。在此我衷心感谢我的老师同学和对以上资源的作者。

## 附：源程序清单

源程序：

```
#define MAXVALUE 1000//定义最大权值
#define MAXBIT 100//定义哈夫曼树中叶子结点个数

typedef struct
{
    char data;//字符值
    int num;//某个值的字符出现的次数
}TotalNode;
//统计结点，包括字符种类和出现次数
typedef struct
{
    TotalNode tot[300];//统计结点数组
    int num;//统计数组中含有的字符个数
}Total;
//统计结构体，包括统计数组和字符种类数
typedef struct
{
    char mes[300];//字符数组
    int num;//总字符数
}Message;
//信息结构体，包括字符数组和总字符数
typedef struct{
    int locked[500];//密码数组
    int num;//密码总数
}Locking;
//哈夫曼编码后的密文信息
typedef struct
{
    char data;//字符
    int weight;//权值
    int parent;//双亲结点在数组 HuffNode[] 中的序号
    int lchild;//左孩子结点在数组 HuffNode[] 中的序号
    int rchild;//右孩子结点在数组 HuffNode[] 中的序号
```

```

}HNodeType;
//哈夫曼树结点类型，包括左右孩子，权值和信息
typedef struct
{
    int bit[MAXBIT];
    int start;
}HCodeType;
//哈夫曼编码结构体，包括编码数组和起始位

void reading_file(Message *message); //从文件中读取信息
void reading_file1(Message *message); //从文件中读取信息
void writing_file(Message *message); //将信息写进文件
void total_message(Message *message, Total *total); //统计信息中各字符的次数
void HaffmanTree(Total *total, HNodeType HuffNode[]); //构建哈夫曼树
void HaffmanCode(HNodeType HuffNode[], HCodeType HuffCode[], Total *total); //建立哈夫曼编码
void writing_HCode(HNodeType HuffNode[], HCodeType HuffCode[], Total *total); //将编码规则写进文件
void lock(Message *message, HNodeType HuffNode[], HCodeType HuffCode[], Total *total, Locking *locking); //给文件信息加密编码
void writing_lock(Locking *locking); //将已编码信息写进文件
void writing_translate(Locking *locking, HCodeType HuffCode[], HNodeType HuffNode[], Total *total); //将已编码信息翻译过来并写进文件

#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    int i, j, choice, mark=0; //mark 标记文件信息是否读入到内存中
    int n;
    HNodeType HuffNode[500]; //保存哈夫曼树中各结点信息
    HCodeType HuffCode[300];
    Locking *locking;
    Total *total;
    Message *message;
    locking=new Locking;
    locking->num=0;
    total=new Total;
    total->num=0;
    message=new Message;
    message->num=0; //初始化变量

```

```

while(1)
{
    cout<<"*****";
    cout<<"*1: 从文件读取信息          2: 显示编码规则          3: 将
原文件信息写进文件          *";
    cout<<"*4: 将编码规则写进文件  5: 将编码后的信息写进文件 6: 将
译码后的信息写进文件          *";
    cout<<"*7          :          退          出
*****";
    ***"<<endl;
    cout<<"请输入操作代码: ";
    cin>>choice;
    switch(choice)
    {
        case 1:
            reading_file(message);//从文件中读取信息
            for(n=0;n<message->num;n++)
                printf("%c",message->mes[n]);
            printf("\n");
            mark=1;
            break;
        case 2://显示编码规则
            if(mark==0) cout<<"请先从文件中读取信息!"<<endl;
            else
            {
                total_message(message, total);//统计信息中各字符的出现次数
                HaffmanTree(total, HuffNode);//构建哈夫曼树
                HaffmanCode(HuffNode, HuffCode, total);//建立哈夫曼编码
                for(i=0;i<total->num;i++)//显示编码规则
                {
                    cout<<total->tot[i].data<<" ";
                    for(j=HuffCode[i].start+1;j<total->num;j++)
                        cout<<HuffCode[i].bit[j];
                    cout<<endl;
                }
            }
            break;
        case 3://将原文件信息写进文件
            if(mark==0) cout<<"请先从文件中读取信息!"<<endl;
            else
                writing_file(message);//将信息写进文件
            break;
        case 4://将编码规则写进文件

```

```

        if(mark==0)cout<<"请先从文件中读取信息!"<<endl;
        else
    {
        total_message(message, total); //统计信息中各字符的出现次数
        HaffmanTree(total, HuffNode); //构建哈夫曼树
        HaffmanCode(HuffNode, HuffCode, total); //建立哈夫曼编码
        writing_HCode(HuffNode, HuffCode, total); //将编码规则写进文件
    }

        break;
    case 5: //将编码后的信息写进文件
        if(mark==0)cout<<"请先从文件中读取信息!"<<endl;
        else
    {
        total_message(message, total); //统计信息中各字符的出现次数
        HaffmanTree(total, HuffNode); //构建哈夫曼树
        HaffmanCode(HuffNode, HuffCode, total); //建立哈夫曼编码
        lock(message, HuffNode, HuffCode, total, locking); //给文件信息
加密编码
        writing_lock(locking); //将已编码信息写进文件
    }

        for(n=0; n<locking->num; n++) //写文件
            if(locking->locked[n]==-1)
                printf(" ");
            else
                cout<<locking->locked[n];
            printf("\n");
            break;
    case 6: //将译码后的信息写进文件
        if(mark==0)cout<<"请先从文件中读取信息!"<<endl;
        else
    {
        total_message(message, total); //统计信息中各字符的出现次数
        HaffmanTree(total, HuffNode); //构建哈夫曼树
        HaffmanCode(HuffNode, HuffCode, total); //建立哈夫曼编码
        writing_translate(locking, HuffCode, HuffNode, total);
        //将已编码信息翻译过来并写进文件
    }

        reading_file1(message);
        for(n=0; n<message->num; n++) //写文件
            if(message->mes[n]==-1)
                printf(" ");
            else
                cout<<message->mes[n];
            printf("\n");

```



```

        break;
    case 7:
        exit(1);
    default:
        cout<<"输入错误, 请重新选择"<<endl;
    }
}

for(i=0;i<locking->num;i++)
    if(locking->locked[i]==-1)cout<<" ";
    else
        cout<<locking->locked[i];
    cout<<endl;
for(i=0;i<total->num;i++)
    cout<<total->tot[i].data<<" "<<total->tot[i].num<<endl;
    for(i=0;i<2*(total->num)-1;i++)
        cout<<HuffNode[i].parent<<" ";
cout<<endl;
return 0;
}

void reading_file(Message *message)
{
/*打开 reading 文件, 失败则结束。不断读取字符并保存进 message 数组中,
直到遇到#结束, 记录字符总数*/
    int i=0;
    char ch;
    ifstream infile("c:\\reading.txt", ios::in|ios::out);
    if(!infile)//打开失败则结束
    {
        cout<<"打开 c:\\reading.txt 文件失败"<<endl;
        exit(1);
    }
    else
        cout<<"读取文件成功"<<endl;
    while(infile.get(ch) && ch!='#')//读取字符直到遇到#
    {
        message->mes[i]=ch;
        i++;
    }
    message->num=i;//记录总字符数
    infile.close();//关闭文件
}

//从文件中读取信息
void reading_file1(Message *message)
{
    int i=0;

```

```

char ch;
ifstream infile("c:\\translate.txt", ios::in|ios::out);
if(!infile)//打开失败则结束
{
    cout<<"打开 c:\\translate.txt 文件失败"<<endl;
    exit(1);
}
else
    cout<<"读取文件成功"<<endl;
while(infile.get(ch) && ch!='#')//读取字符直到遇到#
{
    message->mes[i]=ch;
    i++;
}
message->num=i;//记录总字符数
infile.close();//关闭文件
} //从文件中读取信息
void writing_file(Message *message)//将信息写进文件
{ /*打开 writing 文件，失败则结束。将信息写进文件*/
    int i;
    ofstream outfile("c:\\writing.txt", ios::in|ios::out);
    if(!outfile)//打开失败则结束
    {
        cout<<"打开 c:\\writing.txt 文件失败"<<endl;
        exit(1);
    }
    for(i=0; i<message->num; i++)//写文件
        outfile.put(message->mes[i]);
    cout<<"信息写进文件成功"<<endl;
    outfile.close();//关闭文件
} //将原信息写入文件
void total_message(Message *message, Total *total)
{ /*将 message 中的字符种类及出现次数统计保存到 total 数组中，重复字
符用 mark 标记，
    否则新建字符种类。记录下字符种类的个数*/
    int i, j, mark;
    for(i=0; i<message->num; i++)//遍历 message 中的所有字符信息
    {
        if(message->mes[i]!=' ')//字符不为空格时
        {
            mark=0;
            for(j=0; j<total->num; j++)//在 total 中搜索当前字符
                if(total->tot[j].data==message->mes[i])//搜索到，则
                    此字符次数加 1, mark 标志为 1

```

```

        {
            total->tot[j].num++;
            mark=1;
            break;
        }
    if(mark==0)//未搜索到，新建字符种类，保存进 total 中，字
符类加 1
    {
        total->tot[total->num].data=message->mes[i];
        total->tot[total->num].num=1;
        total->num++;
    }
}
}
//统计信息中各字符的出现次数
void HaffmanTree(Total *total,HNodeType HuffNode[])
{ /*通过每次选取最小和次小两权值建立二叉树，最终构建成为哈夫曼树，
且左孩子权值比右孩子小*/
    int i,j,min1,min2,x1,x2;
    for(i=0;i<2*(total->num)-1;i++)//初始化哈夫曼树并存入叶子结点
权值和信
息
    {
        if(i<=total->num-1)HuffNode[i].data=total->tot[i].data;
        HuffNode[i].weight=total->tot[i].num;
        HuffNode[i].parent=-1;
        HuffNode[i].lchild=-1;
        HuffNode[i].rchild=-1;
    }
    for(i=0;i<total->num-1;i++)
    {
        min1=min2=MAXVALUE;
        x1=x2=0;
        for(j=0;j<total->num+i;j++)//选取最小 x1 和次小 x2 的两权值
        {
            if(HuffNode[j].parent==-1&&HuffNode[j].weight<min1)
            {
                min2=min1;
                x2=x1;
                min1=HuffNode[j].weight;
                x1=j;
            }
            else
                if(HuffNode[j].parent==-1&&HuffNode[j].weight<min2)
                    min2=HuffNode[j].weight;

```

```

        {
            x2=j;
        }
    }
    HuffNode[x1].parent=total->num+i;//修改亲人结点位置
    HuffNode[x2].parent=total->num+i;

    HuffNode[total->num+i].weight=HuffNode[x1].weight+HuffNode[x2].weight;

    HuffNode[total->num+i].lchild=x1;//左孩子比右孩子权值小
    HuffNode[total->num+i].rchild=x2;
}
} //构建哈夫曼树
void HaffmanCode(HNodetype HuffNode[], HCodetype HuffCode[], Total
*total)
{
    /*在建立的哈夫曼树基础上，左分支为 0，右分支为 1，
    从叶结点向上直到根结点建立哈夫曼编码，并保存每个叶结点起始位*/
    int i, j, c, p;
    HCodetype cd;
    for(i=0; i<total->num; i++) //建立叶子结点个数的编码
    {
        cd.start=total->num-1; //起始位初始化
        p=HuffNode[i].parent;
        c=i;
        while(p!=-1) //从叶结点向上爬直到到达根结点
        {
            if(HuffNode[p].lchild==c) //左分支则为 0
                cd.bit[cd.start]=0;
            else
                cd.bit[cd.start]=1; //右分支则为 1
            cd.start--; //起始位向前移动
            c=p;
            p=HuffNode[p].parent;
        }
        for(j=cd.start+1; j<total->num; j++)
            //保存求出的每个叶结点编码和起始位
            HuffCode[i].bit[j]=cd.bit[j];
            HuffCode[i].start=cd.start;
    }
} //建立哈夫曼编码
void writing_HCode(HNodetype HuffNode[], HCodetype HuffCode[], Total
*total)
{
    /*打开 HCode 文件，失败则结束。将信息写进文件*/

```

```

int i, j;
ofstream outfile("c:\\HCode.txt", ios::in|ios::out);
if(!outfile)//打开失败则结束
{
    cout<<"打开 c:\\HCode.txt 文件失败"<<endl;
    exit(1);
}
for(i=0;i<total->num;i++)//写文件
{
    outfile.put(HuffNode[i].data);outfile<<" ";
    for(j=HuffCode[i].start+1;j<total->num;j++)
        outfile<<HuffCode[i].bit[j];
    outfile<<endl;
}
cout<<"编码规则写进文件成功"<<endl;
outfile.close();//关闭文件
} //将编码规则写进文件
void lock(Message *message, HNodeType HuffNode[], HCodetype
HuffCode[], Total *total, Locking *locking)
{
    int i, j, m;
    for(i=0;i<message->num;i++)//遍历信息
    {if(message->mes[i]==' ')
//碰到空格则以-1 形式保存进 locked 数组中
    {
        locking->locked[locking->num]=-1;
        locking->num++;
    }
    else
        for(j=0;j<total->num;j++)//搜索信息对应的编码
        {
            if(HuffNode[j].data==message->mes[i])//找到与信息对
应的编码则写进 locked 数组
                for(m=HuffCode[j].start+1;m<total->num;m++)
                {
                    locking->locked[locking->num]=HuffCode[j].bit[m];
                    locking->num++;//locked 数组总数加 1
                }
        }
    }
} //给文件信息加密编码
void writing_lock(Locking *locking)
{

```

```

int i;
ofstream outfile("c:\\locking.txt", ios::in|ios::out);
if(!outfile)//打开失败则结束
{
    cout<<"打开 c:\\locking.txt 文件失败"<<endl;
    exit(1);
}
for(i=0;i<locking->num;i++)//写文件
    if(locking->locked[i]==-1)
        outfile.put(' ');
    else
        outfile<<locking->locked[i];
cout<<"已编码信息写进文件成功"<<endl;
outfile.close();//关闭文件
} //将哈夫曼编码后的密文信息写进文件
void writing_translate(Locking *locking, HCodetype
HuffCode[], HNodeType HuffNode[], Total *total)
{
    int i, j, mark[100], start[100], min, max;
    ofstream outfile("c:\\translate.txt", ios::in|ios::out); // 打开
文件
    if(!outfile)//打开失败则结束
    {
        cout<<"打开 c:\\translate.txt 文件失败"<<endl;
        exit(1);
    }
    for(i=0;i<total->num;i++)//start 数组初始化
    {
        start[i]=HuffCode[i].start+1;
    }
    for(i=0;i<total->num;i++)//mark 数组初始化
    {
        mark[i]=1;
    }
    min=0;
    for(max=0;max<locking->num;max++)//写文件
    {
        if(locking->locked[max]==-1)//碰到空格信息则直接输出空格
        {
            outfile.put(' ');
            min++;
        }
        else
        {

```

```

        for(i=min;i<=max;i++)//查找从min开始到max的编码对应的信息
        {
            for(j=0;j<total->num;j++)
            {
                if(start[j]==total->num+1)
                    mark[j]=0;//对应编码比所给编码短则不在比较
                if(mark[j]==1&&locking->locked[i]==HuffCode[j].bit[start[j]])
                    start[j]++;
                else
                    mark[j]=0;
            }
        }
        for(i=0;i<total->num;i++)//找到对应信息，则写进文件
        {
            if(mark[i]==1&&start[i]==total->num)
            {
                outfile.put(HuffNode[i].data);
                break;
            }
        }
        if(i!=total->num)
            min=max+1;
        for(i=0;i<total->num;i++)//start 数组初始化
        {
            start[i]=HuffCode[i].start+1;
        }
        for(i=0;i<total->num;i++)//mark 数组初始化
        {
            mark[i]=1;
        }
    }
    cout<<"翻译信息写进文件成功"<<endl;
    outfile.close();//关闭文件
} //将已编码信息翻译过来并写进文件

```

## 第 4 题：校园导游系统

### 1. 实验内容

#### 【问题描述】

设计一个校园导游程序，为来访的客人提供各种信息查询服务。

#### 【基本要求】

- (1) 设计天津商业大学的校园平面图，所含景点不少于 10 个。以图中顶点表示校内各景点，存放景点名称、代号、简介等信息；以边表示路径，存放路径长度等相关信息。
- (2) 为来访客人提供图中任意景点相关信息的查询。
- (3) 为来访客人提供图中任意景点的问路查询，即查询任意两个景点之间的一条最短的简单路径。
- (4) 使用汉字显示。

#### 【选做内容】

- (1) 系统功能的完善；
- (2) 提供求任意两个景点之间的所有路径的功能；
- (3) 提供校园图中多个景点的最佳访问路线查询，即求途经这多个景点的最佳（短）路径。

### 2. 需求分析

为方便游客迅速查到自己想要去的地方，我们设计了天津商业大学的校园平面图，所含景点不少于 10 个。以图中顶点表示校内各景点，存放景点名称、代号、简介等信息；以边表示路径，存放路径长度等相关信息。为来访客人提供图中任意景点相关信息的查询。为来访客人提供图中任意景点的问路查询，即查询任意两个景点之间的一条最短的距离。



### 3. 概要设计

#### 3.1 本文采用的抽象数据类型

ADT Graph{

数据对象 V: V 是具有相同特性数据元素的集合, 称为顶点集。

数据关系:  $R=\{VR\}$

$\{VR\}=\{(v,w)|v,w\in V,(v,w)\text{表示 } v \text{ 和 } w \text{ 之间的存在路径}\}$

基本操作 P:

introduce(int num)

初始条件: num 为 1——13 的数字;

操作结果: 输出景点编号为 num 的景点简介;

shortestdistance(int i, int j)

初始条件: i 为游客当前所在景点的编号, j 为游客想要去的景点的编号;

操作结果: 输出两景点间的最短路径;

}ADT Graph

#### 3.2 模块划分

- (1) 景点信息查询
- (2) 两景点的最短距离
- (3) 两个景点之间的路径

### 4. 详细设计

#### 4.1 数据类型的定义

int top[14]={0}; //存放顶点信息

int cost[14][14]; //边的值

int path[14][14]; //经过的顶点

int final[14]={0}; //结点信息

int D[14]; //最短距离

#### 4.2 主要模块的算法描述

void print() //景点列表

```

{   printf("      欢迎您来到天津商业大学\n");
    printf("      *****\n");
    printf("      祝您旅途愉快\n");
    printf("以下是您可能要前往的地方\n");
    printf("1   东门   \n");
    printf("2   桃李园   \n");
    printf("3   西区    \n");
    printf("4   南区    \n");
    printf("5   大食堂   \n");
    printf("6   实验楼   \n");
    printf("7   信息交流中心   \n");
    printf("8   FIU     \n");
    printf("9   瑞德厦   \n");
    printf("10  图书馆   \n");
    printf("11  游泳馆   \n");
    printf("12  实验楼   \n");
    printf("13  操场     \n");
    printf("功能 1.景点查询请输入 i\n");
    printf("功能 2.查询最短路径请输入 s\n");
    printf("功能 3.退出系统请输入 e\n");
    printf("请输入您的选择:");
}

void introduce() // 景点介绍
{
    int a;
    printf("请输入景点编号:");
    scanf("%d",&a);
    getchar();
    printf("\n");
    while(a<1||a>13)

```

```

{
    printf("ERROR! 请输入数字 1 到 13:\n\n");
    scanf("%d",&a);
}
switch(a)
{
case 1:
    printf("1:东门 天津商业大学大门\n\n");
break;
case 2:
    printf("2:桃李园 学生住宿的地方\n\n");
break;
case 3:
    printf("3:西区 学生住宿的地方\n\n");
break;
case 4:
    printf("4:南区 学生住宿的地方，有食堂和超市等\n\n");
break;
case 5:
    printf("5:大食堂 学生吃饭的地方\n\n");
break;
case 6:
    printf("6:实验楼 学生开展各种实验的地方\n\n");
break;
case 7:
    printf("7:信息交流中心 学生可以上机的地方以及教师的办公地点\n\n");
break;
case 8:

```

```

        printf("8:FIU    学生上课的地方\n\n");
break;

        case 9:
            printf("9:瑞德厦    学生可以吃饭的地方，较大食堂贵\n\n\n");
break;

        case 10:
            printf("10:图书馆    内有许多藏书，学生可以自习，也有电子阅览室\n\n");
break;

        case 11:
            printf("11:游泳馆    可以游泳，上体育课\n\n");
break;

        case 12:
            printf("12:重点实验楼    学生可以做重点实验\n\n");
break;

        case 13:
            printf("13:操场        学生活动锻炼的好去处\n\n");
break;
    }
    printf("/n");
}

void main()
{
    Int  k;
    print();
    scanf("%d",&k);
    while((k!=1)&&(k!=2)&&(k!=3))
    {
        printf("ERROR 请输入 1 或 2 或 3\n");
        scanf("%d",&k);
    }
}

```

```

    }
    switch(k)
    {
    case 1:
        printf("进入景点查询:\n");
        introduce();
        break;
    case 2:
        printf("进入最短路径查询:\n");
        shortestdistance();
        break;
    case 3:
        exit(0);
    }
}

void shortestdistance() // 求最短路径
{
    int i,v,w,v0,j;
    int min;
    int top[14]={0};
    int cost[14][14];
    int path[14][14];
    int final[14]={0};
    int D[14];
    for(i=0;i<14;i++)
        for(j=0;j<14;j++)
            cost[i][j]=Init_Length;
    cost[1][3]=cost[3][1]=10;
    cost[3][5]=cost[5][3]=40;
    cost[1][7]=cost[7][1]=10;

```

```

cost[3][7]=cost[7][3]=30;
cost[2][7]=cost[7][2]=20;
cost[2][6]=cost[6][2]=10;
cost[4][6]=cost[6][4]=10;
cost[4][13]=cost[13][4]=10;
cost[6][12]=cost[12][6]=20;
cost[12][8]=cost[8][12]=10;
cost[8][9]=cost[9][8]=10;
cost[6][9]=cost[9][6]=15;
cost[10][9]=cost[9][10]=10;
cost[6][10]=cost[10][6]=20;
cost[9][10]=cost[10][9]=10;
cost[9][11]=cost[11][9]=10;

printf("请输入您现在所在的位置:\n");
scanf("%d",&v0);
while(v0>13||v0<1)
{
    printf("ERROR!请重新输入编号从 1 到 13 的数\n");
    scanf("%d",&v0);
}
for(i=1;i<14;i++)
    for(j=1;j<14;j++)
        path[i][j]=0;
for(v=1;v<14;v++)
{
    D[v]=cost[v0][v];
    if(D[v]<Init_Length)
    {
        path[v][++(top[v])]=v0;
        path[v][++(top[v])]=v;
    }
}

```

```

    }
}
D[v0]=0;final[v0]=1;
    for(i=2;i<14;++i)
    {
        min=Init_Length;
        for(w=1;w<14;++w)
        {
            if((final[w]==0)&&(D[w]<min))
            {
                v=w;min=D[w];
            }
        }
        final[v]=1;
        for(w=1;w<14;++w)
        {
            if((final[w]==0)&&(min+cost[v][w]<D[w]))
            {
                D[w]=min+cost[v][w];
                for(j=1;j<14;j++)
                path[w][j]=path[v][j];
                top[w]=top[v]+1;
                path[w][(top[w])]=w;
            }
        }
    }

    printf("请输入你要去的地方:\n");
    scanf("%d",&w);
    printf("\n");
    while(w>13||w<1)

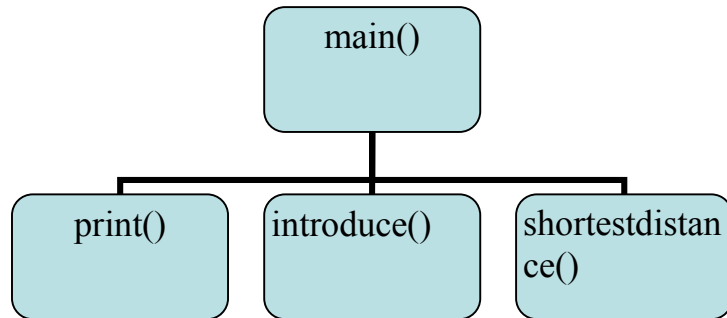
```

```

{
    printf("ERROR!输入错误,请重新输入编号从 1 到 13\n");
    scanf("%d",&w);
}
printf("最短路径为:\n");
for(i=1;path[w][i]!=0;i++)
    printf("-->%d",path[w][i]);
printf("\n");
printf("最短路径的长度为: %d\n",D[w]);
}

```

#### 4.3 函数之间的调用关系



## 5. 程序运行说明与测试

### 5.1 用户手册

程序运行后，屏幕显示开始界面景点列表，并提供功能 1、查询景点信息 2、查询景点间的最短浏览路径 3、退出系统。此时用户可根据自己的选择选择自己要查询的项目。选择 1 后界面将会展现出景点编号及简介。选择 2 后，任意选取两个景点，系统将会给出两景点间的最短路径。选择 3，退出系统。

### 5.2 测试数据

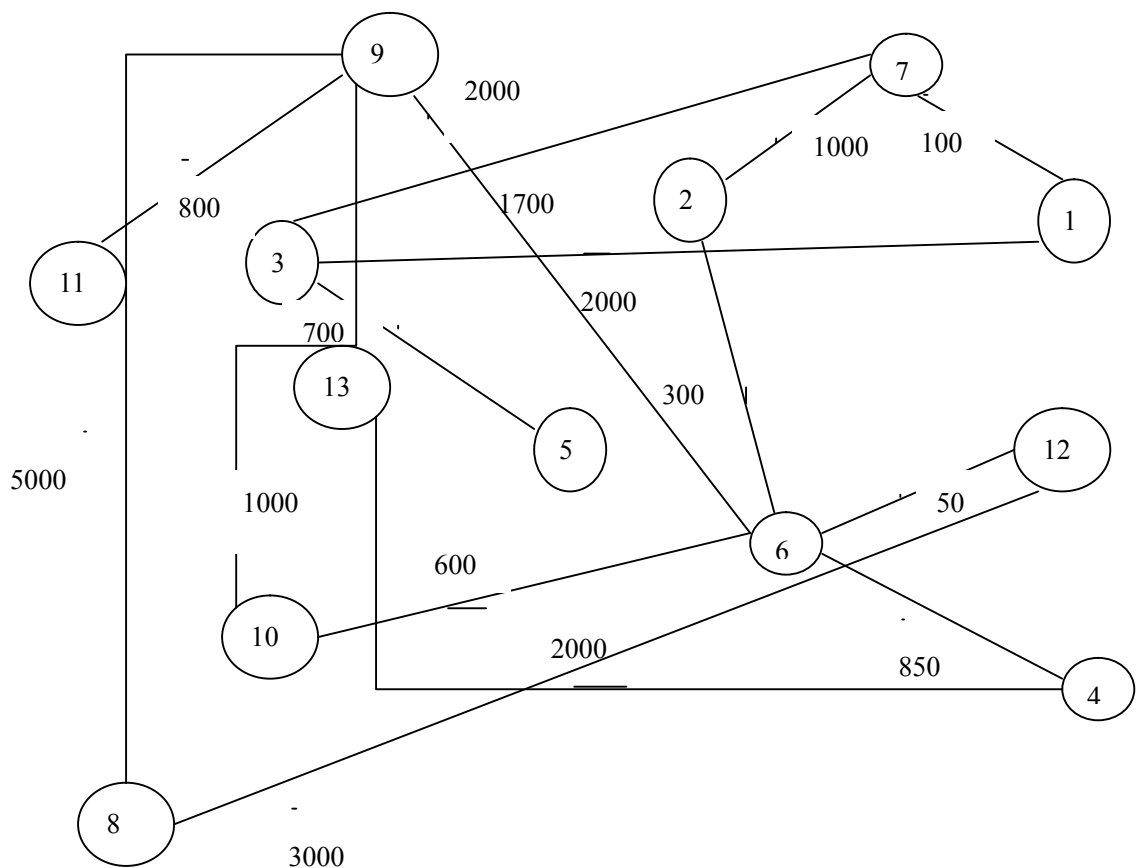
校园景点平面图：



```

欢迎您来到天津商业大学(Welcome to Tianjin University of Commerce)
*****
祝您旅途愉快(Wish you a pleasant trip!)
以下为天津商业大学的主要景点(The following are the main view spots)
1 东门 <The gate of east>
2 学生公寓 <Students Apartment >
3 西区 <West Area>
4 南区 <South Area>
5 大食堂 <Main Restaurant >
6 实验楼 < Laboratory Building >
7 信息交流中心
8 FIU
9 瑞德厦 <RuiDe Building>
10 图书馆 <Library>
11 游泳馆 <Swimming Pool >
12 重点实验楼 <Key Laboratory Building>
13 操场 <Playground>
景点查询请输入 1<Please input number one if you want to search the view spots. >
查询最短路径请输入 2<Please input number two if you want to search the shortest
way of two spots.>
退出系统请输入 3<Please input number three if you want to exit the system.>
请输入您的选择:<Please input your choice>

```



(1)选择 1 查询景点，比如查询 4 号景点的简介

```
"D:\C程序\gao\Debug\gao. exe"
2 学生公寓 <Students Apartment >
3 西区 <West Area>
4 南区 <South Area>
5 大食堂 <Main Restaurant >
6 实验楼 < Laboratory Building >
7 信息交流中心
8 FIU
9 瑞德厦<RuiDe Building>
10 图书馆<Library>
11 游泳馆 <Swimming Pool >
12 重点实验楼<Key Laboratory Building>
13 操场 <Playground>
景点查询请输入 1<Please input number one if you want to search the view spots. >
查询最短路径请输入 2<Please input number two if you want to search the shortest
way of two spots.>
退出系统请输入 3<Please input number three if you want to exit the system.>
请输入您的选择:<Please input your choice>1
进入景点查询:<The spots seraching system.>
请输入您想要查找的景点编号:<Please input the number of the spot which you want t
o search.>4
4:南区 学生住宿的地方, 有食堂和超市等<Students live here with resturant and su
permarket.>
Press any key to continue
```

(2)查询景点间的最短路径比如查询景点四号和八号的最短路径

```
"D:\C程序\gao\Debug\gao. exe"
5 大食堂 <Main Restaurant >
6 实验楼 < Laboratory Building >
7 信息交流中心
8 FIU
9 瑞德厦<RuiDe Building>
10 图书馆<Library>
11 游泳馆 <Swimming Pool >
12 重点实验楼<Key Laboratory Building>
13 操场 <Playground>
景点查询请输入 1<Please input number one if you want to search the view spots. >
查询最短路径请输入 2<Please input number two if you want to search the shortest
way of two spots.>
退出系统请输入 3<Please input number three if you want to exit the system.>
请输入您的选择:<Please input your choice>2
进入最短路径查询:<The shortest way searching system.>
请输入您现在所在的位置:<Please input the place you are now.>
4
请输入你要去的地方:<Please input the place you want to go.>
8
最短路径为<The shortest way is:>:
46128
最短路径的长度为<The length of the shortest way is:>: 3900
Press any key to continue
```

(3)选择 3 退出系统

```
"D:\C程序\gao\Debug\gao.exe"
欢迎您来到天津商业大学(Welcome to Tianjin University of Commerce)
*****
祝您旅途愉快(Wish you a pleasant trip!)
以下为天津商业大学的主要景点(The following are the main view spots)
1 东门 <The gate of east>
2 学生公寓 <Students Apartment >
3 西区 <West Area>
4 南区 <South Area>
5 大食堂 <Main Restaurant >
6 实验楼 < Laboratory Building >
7 信息交流中心
8 FIU
9 瑞德厦<RuiDe Building>
10 图书馆<Library>
11 游泳馆 <Swimming Pool >
12 重点实验楼<Key Laboratory Building>
13 操场 <Playground>
景点查询请输入 1<Please input number one if you want to search the view spots.>
查询最短路径请输入 2<Please input number two if you want to search the shortest
way of two spots.>
退出系统请输入 3<Please input number three if you want to exit the system.>
请输入您的选择:<Please input your choice>3
Press any key to continue
```

## 6. 实验总结

在设计中，查找最短路径比较困难，借助了大量的辅助资料，设计的程序仅仅给出了最短路径，没有给出最短路径经过的点。在程序中加入了英文解释，对外国游客且不懂中文有很大帮助。

## 附：源程序清单

```
#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#define Max 20

#define Init_Length 10000

void shortestdistance();

void print()//景点列表

{
```

```

printf("      欢迎您来到天津商业大学(Welcome to Tianjin University of
Commerce)\n");

printf("      *****\n");

printf("      祝您旅途愉快(Wish you a pleasant trip!)\n");

printf("以下为天津商业大学的主要景点(The following are the main view
spots)\n");

printf("1 东门 (The gate of east) \n");
printf("2 学生公寓 (Students Apartment ) \n");
printf("3 西区 (West Area) \n");
printf("4 南区 (South Area) \n");
printf("5 大食堂 (Main Restaurant ) \n");
printf("6 实验楼 ( Laboratory Building ) \n");
printf("7 信息交流中心 \n");
printf("8 FIU \n");
printf("9 瑞德厦(RuiDe Building) \n");
printf("10 图书馆(Library) \n");
printf("11 游泳馆 (Swimming Pool ) \n");
printf("12 重点实验楼(Key Laboratory Building) \n");
printf("13 操场 (Playground) \n");

printf("景点查询请输入 1(Please input number one if you want to search
the view spots. )\n");

printf("查询最短路径请输入 2(Please input number two if you want to
search the shortest way of two spots.)\n");

printf("退出系统请输入 3(Please input number three if you want to exit the
system.)\n");

printf("请输入您的选择:(Please input your choice)");
}

void introduce() //景点简介
{
    int a;

```

```
printf("请输入您想要查找的景点编号:(Please input the number of the spot  
which you want to search.)");  
scanf("%d",&a);  
while(a<1||a>13)  
{  
    printf("输入错误! 请输入数字 1 到 13(Error!Please input number from one  
to Thirteen)\n");  
    scanf("%d",&a);  
}  
switch(a)  
{  
case 1:  
    printf("1: 东门 天津商业大学大门(The gate of Tianjin University of  
Commerce)\n\n");  
break;  
case 2:  
    printf("2:学生公寓 学生住宿的地方(Students live here)\n\n");  
break;  
case 3:  
    printf("3:西区 学生住宿的地方(Students live here)\n\n");  
break;  
case 4:  
    printf("4:南区 学生住宿的地方, 有食堂和超市等(Students live here with  
resturant and supermarket.)\n\n");  
break;  
case 5:  
    printf("5:大食堂 学生吃饭的地方(Have meals in here)\n\n");  
break;  
case 6:
```

```

printf("6:实验楼  学生开展各种实验的地方(A place for students do some
experiments )\n\n");
break;

case 7:
printf("7:信息交流中心  学生可以上机的地方以及教师的办公地点
(Teachers do their work here.)\n");
break;

case 8:
printf("8:FIU  学生上课的地方(A place for students to have classes.)\n\n");
break;

case 9:
printf("9:瑞德厦  学生可以吃饭的地方，较大食堂贵(A resturant)\n\n\n");
break;

case 10:
printf("10:图书馆  内有许多藏书，学生可以自习，也有电子阅览室\n\n");
break;

case 11:
printf("11:游泳馆  可以游泳，上体育课(Swimming in here.)\n\n");
break;

case 12:
printf("12:重点实验楼  学生可以做重点实验 (Do
importantexperiments )\n\n");
break;

case 13:
printf("13:操场  学生活动锻炼的好去处(A place for students to do
exercises.)\n\n");
break;

}

printf("/n");
}

```

```

void main()
{
    int k;
    print();
    scanf("%d",&k);
    while((k!=1)&&(k!=2)&&(k!=3))
    {
        printf("输入错误！请输入 1 或 2 或 3(Error!Please input number one or
number two or number three.)\n");
        scanf("%d",&k);
    }
    switch(k)
    {
        case 1:
            printf("进入景点查询:(The spots seraching system.)\n");
            introduce();//调用景点简介算法
            break;
        case 2:
            printf("进入最短路径查询:(The shortest way searching system.)\n");
            shortestdistance();//调用最短路径算法
            break;
        case 3:
            exit(0);
    }
}

void shortestdistance() //求最短路径
{
    //查询最短路径
    int i,v,w,v0,j;
    int min;
    int top[14]={0};

```

```

int cost[14][14];
int path[14][14];
int final[14]={0};
int D[14];
for(i=0;i<14;i++)
for(j=0;j<14;j++)
cost[i][j]=Init_Length;
cost[1][3]=cost[3][1]=2000;
cost[3][5]=cost[5][3]=700;
cost[1][7]=cost[7][1]=100;
cost[3][7]=cost[7][3]=1700
cost[2][7]=cost[7][2]=1000;
cost[2][6]=cost[6][2]=300;
cost[4][6]=cost[6][4]=850;
cost[4][13]=cost[13][4]=2000;
cost[6][12]=cost[12][6]=50;
cost[12][8]=cost[8][12]=3000;
cost[8][9]=cost[9][8]=5000;
cost[6][9]=cost[9][6]=2000;
cost[10][9]=cost[9][10]=1000;
cost[6][10]=cost[10][6]=600;
cost[9][11]=cost[11][9]=800;

printf("请输入您现在所在的位置:(Please input the place you are
now.)\n");

scanf("%d",&v0);
while(v0>13||v0<1)
{
printf("输入错误!请重新输入编号从 1 到 13 的数(Error!Please input
number from one to Thirteen)\n");

scanf("%d",&v0);

```



```

    }
    for(i=1;i<14;i++)
    for(j=1;j<14;j++)
    path[i][j]=0;
for(v=1;v<14;v++)
{
    D[v]=cost[v0][v];
    if(D[v]<Init_Length)
    {
        path[v][++(top[v])]=v0;
        path[v][++(top[v])]=v;
    }
}
D[v0]=0;final[v0]=1;
for(i=2;i<14;++i)
{
    min=Init_Length;
    for(w=1;w<14;++w)
    {
        if((final[w]==0)&&(D[w]<min))
        {
            v=w;min=D[w];
        }
    }
    final[v]=1;
    for(w=1;w<14;++w)
    {
        if((final[w]==0)&&(min+cost[v][w]<D[w]))
        {
            D[w]=min+cost[v][w];
            for(j=1;j<14;j++)
            path[w][j]=path[v][j];

```

```

        top[w]=top[v]+1;
        path[w][(top[w])]=w;
    }
}
}

printf("请输入你要去的地方:(Please input the place you want to
go.)\n");

scanf("%d",&w);
printf("\n");
while(w>13||w<1)
{
    printf("输入错误,请重新输入编号从 1 到 13(Error!Please input number
from one to Thirteen)\n");
    scanf("%d",&w);
}

printf("最短路径为(The shortest way is:):\n");
for(i=1;path[w][i]!=0;i++)
    printf("%d",path[w][i]);
printf("\n");
printf("最短路径的长度为(The length of the shortest way is:): %d\n",D[w]);
}

```

## 第 6 题：哈希表的建立与查找

### 1. 实验内容

针对本班中的“人名”设计一个哈希表，使得平均查找长度不超过 2，完成相应的建表和查表程序。假设人名为中国人姓名的汉语拼音形式，待填入哈希表的人名共有 30 个，取平均查找长度的上限为 2。哈希函数用除留余数法构造，用伪随机探测再散列法处理冲突。

### 2. 需求分析

(1) 针对某个集体中的人名设计一个哈希表，使得平均查找长度不超过 2，完成相应的建立和查表程序。

(2) 人名为汉语拼音形式，最长不超过 18 个字符(如：庄双双 zhuangshuangshuang)。

(3) 假设待填入哈希表的人名有 30 个，平均查找长度为 2。哈希表用除留余数法构造，用伪随机探测再散列法处理冲突。

(4) 在输入人名过程中能自动识别非法输入，并给与非法输入的反馈信息要求重新输入。

(5):输入非法数据来检验

如:12345,zhuang shuangshuang,\$%&^&\*等等

|                    |      |
|--------------------|------|
| 查找 输入:zhengzhuowan | 查找成功 |
|--------------------|------|

|                       |     |
|-----------------------|-----|
| 输入:zhuangshuangshuang | 无记录 |
|-----------------------|-----|

|                |      |
|----------------|------|
| 输入:zhangyiying | 查找成功 |
|----------------|------|

|             |     |
|-------------|-----|
| 输入:zhanglei | 无记录 |
|-------------|-----|

(在输入时也输入非法数据来检验)

要完成如下要求：设计哈希表实现姓名查找。实现本程序需要解决以下几个问题：

- ◆用除留余数建立哈希函数。
- ◆如何利用伪随机探测再散列法处理冲突。
- ◆如何实现用哈希法查找并显示用户信息。
- ◆如何查找并显示给定用户的记录。

### 3. 概要设计

#### 3.1 抽象数据类型定义:

ADT HashList\_T

```
{  
    数据对象:D={ai| ai ∈ ElemSet, {不多于 18 个字符的字符串}0≤i<18};  
    数据关系:R={< ai-1, ai >| ai-1, ai ∈ D, 0≤i<18 }
```

基本操作:

void InitNameList()

操作结果:初始化一个哈希表

void createHashList()

操作结果:创建一个哈希表

void FindList()

前置条件:哈希表已经建立,s 非空

后置条件:查找所输入的人名在不在哈希表中

void Display()

前置条件: 哈希表已经建立,s 非空

后置条件:显示创立的哈希表

};

#### 3.2 模块划分

本程序的模块可分为两个大模块:

主程序模块和哈希表模块,

其中哈希表模块又可分为初始化模块,建立模块,查找模块,显示模块,

### 4. 详细设计

#### 4.1 数据类型的定义

```
typedef struct  
{    char *py;        //名字的拼音  
    int k;            //拼音所对应的整数  
}NAME;  
  
NAME NameList[HASH_LENGTH];    //全局变量 NAME
```

```

typedef struct    //哈希表
{
    char *py;    //名字的拼音
    int k;        //拼音所对应的整数
    int si;       //查找长度
}HASH;

HASH HashList[HASH_LENGTH];    //全局变量 HASH

```

## 4.2 主要模块的算法描述

初始化(创立)模块算法:

```

void InitNameList()
{
    char *f;
    int r, s0, i;
for (i=0; i<HASH_LENGTH; i++)
{
    NameList[i].py = new char[64];
    NameList[i].py[0] = 0;
}
    for(i=0;i<NAME_NO;i++)
{
    s0=0;
    f=NameList[i].py;
    for(r=0;*(f+r)!='\0';r++)

        s0=*(f+r)+s0;
    NameList[i].k=s0;
}
}

```

建立哈希表模块的算法:

```

void CreateHashList()
{
int i;
for(i=0; i<HASH_LENGTH;i++)
{
    HashList[i].py=new char[64];
    HashList[i].py[0] = 0;
    HashList[i].k=0;
    HashList[i].si=0;
}
for(i=0;i<HASH_LENGTH;i++)

```

```

{
    int sum=0;
    int adr=(NameList[i].k)%M;
    int d=adr;
    if(HashList[adr].si==0)    //如果不冲突
    {
        HashList[adr].k=NameList[i].k;
        HashList[adr].py=NameList[i].py;
        HashList[adr].si=1;
    }
    else    //冲突
    {
        while (HashList[d].k!=0)
        {
            d=(d+NameList[i].k%10+1)%M;
            sum=sum+1;
        };
        HashList[d].k=NameList[i].k;
        HashList[d].py=NameList[i].py;
        HashList[d].si=sum+1;
    }
}
}
}

```

查找哈希表模块的算法:

```

void FindList()
{
    string name;
    int s0=0, r, sum=1, adr, d;
    cout<<"请输入姓名的拼音:"<<endl;
    cin>>name;;
    for(r=0;r<20;r++)
        s0+=name[r];
    adr=s0%M;
    d=adr;
    if(HashList[adr].k==s0)
        cout<<"姓名:"<<HashList[d].py<<" "<<"关键字:"<<s0<<" "<<"查找长度为:
1"<<endl;
    else if (HashList[adr].k==0)
        cout<<"无此记录!"<<endl;
    else
    {
        int g=0;
        while(g==0)

```

```

{
    d=(d+s0%10+1)%M;
    sum=sum+1;
    if (HashList[d].k==0)
    {
        cout<<"无此记录!"<<endl;
        g=1;
    }
    if (HashList[d].k==s0)
    {
        cout<<"姓名:"<<HashList[d].py<<" "<<"关键字:"<<s0<<" "<<"查找长度
为:"<<sum<<endl;
        g=1;
    }
};
}
}

```

显示哈希表模块的算法:

```

void    Display()
{
    int i;
    float average=0;

    cout<<"\n 地址\t 关键字\t\t 搜索长度\tH(key)\t 姓名\n";
    for(i=0; i<50; i++)
    {
        cout<<i<<" ";
        cout<<"\t"<<HashList[i].k<<" ";
        cout<<"\t\t"<<HashList[i].si<<" ";
        cout<<"\t\t"<<(HashList[i].k%M)<<" ";
        cout<<"\t "<<HashList[i].py<<" ";
        cout<<"\n";
    }

    for(i=0; i<HASH_LENGTH; i++)
        average+=HashList[i].si;
    average/=NAME_NO;
    cout<<"平均查找长度: ASL("<<NAME_NO<<")="<<average<<endl;
}

```

主程序算法:

```

int main()
{

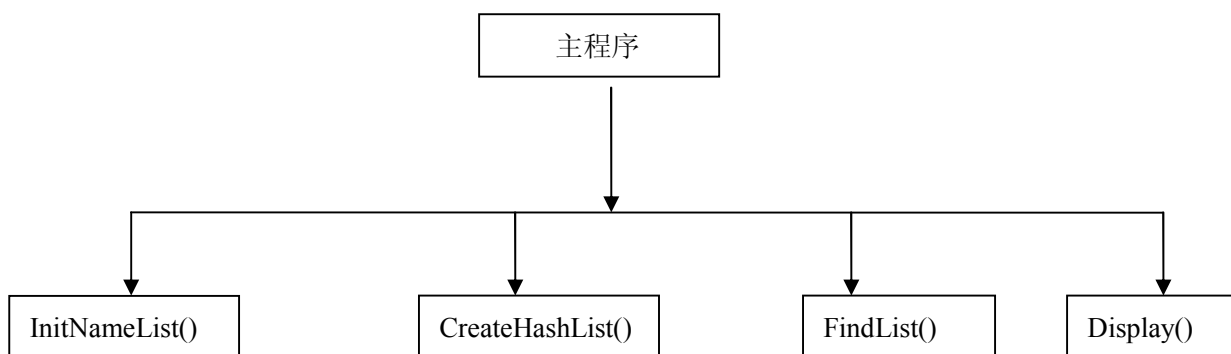
```

```

char x;
InitNameList();
CreateHashList ();
cout<<"d. 显示哈希表 f. 查找 任意键退出    请选择: "<<endl;
while(cin>>x)
{
    if(x=='d')
    {
        Display();
        cout<<endl;
    }
    else if(x=='f')
    {
        FindList();
        cout<<endl;
    }
    else break;
}
for (int i=0; i<HASH_LENGTH; i++)
{
    free(NameList[i].py);
    free(HashList[i].py);
}
return 0;
}

```

### 4.3 函数之间的调用关系





## 5. 程序运行说明与测试

### 5.1 用户手册

程序运行后自动生成哈希表

(1)欲显示哈希表数据,按”d”,回车键

(2)欲查找表中数据,按”f”,输入所要查找的内容,按回车键.

### 5.2 测试数据

合法数据汇总:

wanglu, lizhen, zhangjinyang, wangminqi, cailankai, lijie, zhoujie,  
zhaoxiaotian, zhenzhuowan, zoudeqiang, zhangyiying, qiuliuying,  
sunchunyan, gaofeifei, zhouxun, lvyuanhao, wangqian, tangrong,  
wangcaiyan, makun, wangjiaqi, zhaojianan, lvxuepeng, zhangyahui, xiasiyu,  
lixin, liuqianning, zhangtingting, cuijianxiong

查找合法数据:zhengzhuowan

测试非法数据:0125

### 5.3 测试结果及分析

显示哈希表:

|    |      |   |    |               |
|----|------|---|----|---------------|
| 21 | 0    | 0 | 0  |               |
| 22 | 0    | 0 | 0  |               |
| 23 | 1104 | 1 | 23 | qiuliuying    |
| 24 | 1058 | 1 | 24 | wangcaiyan    |
| 25 | 540  | 3 | 23 | makun         |
| 26 | 1295 | 1 | 26 | zhaoxiaotian  |
| 27 | 0    | 0 | 0  |               |
| 28 | 1201 | 2 | 26 | zhangyiying   |
| 29 | 780  | 2 | 28 | xiasiyu       |
| 30 | 0    | 0 | 0  |               |
| 31 | 971  | 1 | 31 | wangminqi     |
| 32 | 925  | 1 | 32 | cailankai     |
| 33 | 0    | 0 | 0  |               |
| 34 | 927  | 1 | 34 | gaofeifei     |
| 35 | 1059 | 2 | 25 | zhaojianan    |
| 36 | 0    | 0 | 0  |               |
| 37 | 1288 | 3 | 19 | cuijianxiong  |
| 38 | 0    | 0 | 0  |               |
| 39 | 650  | 1 | 39 | lizhen        |
| 40 | 548  | 2 | 31 | lixin         |
| 41 | 1404 | 1 | 41 | zhangtingting |
| 42 | 0    | 0 | 0  |               |
| 43 | 654  | 1 | 43 | wanglu        |
| 44 | 0    | 0 | 0  |               |
| 45 | 1079 | 1 | 45 | zoudeqiang    |
| 46 | 1080 | 1 | 46 | zhangyahui    |
| 47 | 0    | 0 | 0  |               |
| 48 | 0    | 0 | 0  |               |
| 49 | 0    | 0 | 0  |               |

平均查找长度: ASL<30>=1.43333

其中平均查找长度为 1.4333 < 2, 满足题意。

查找合法数据:



```
C:\Program Files\Microsoft Visual Studio\MyProjects\hash\Debu
d. 显示哈希表 f. 查找 任意键退出 请选择:
f
请输入姓名的拼音:
zhengzhuowan
姓名:zhengzhuowan 关键字:1320 查找长度为: 1
```

测试非法数据:



```
C:\Program Files\Microsoft Visual Studio\
d. 显示哈希表 f. 查找 任意键退出 请选择:
f
请输入姓名的拼音:
0125
无此记录!
```

## 6. 实验总结

本次作业只有一个核心算法就是构造散列函数的算法,在调试的时候发现 string 的问题(系统自带的),输入的人名不应该含有空格字符,否则会出现逻辑错误,这是程序的一个问题,如果要修改成 char[]型处理方法类似就没改.在调试其他代码时候没有出现问题,比较顺利的调试成功.

有些函数不写系统会自己生成,为了避免出错自己写了上去只是声名并没有定义,如果用了编译器会报错.

经验体会:借助 DEBUG 调试器和数据观察窗口,可以加快找到程序中的错误,采用软件工程的方法将程序划分层次结构使得代码设计时思路清晰,实现时调试顺利,得到了一次良好的程序设计训练.

## 附：源程序清单

hashList.h 文件

/\*\*\*\*\*\*

针对某个集体中的人名设计一个哈希表，使得平均查找长度不超过 R  
假设待填入哈希表的人名有 30 个，平均查找长度为 2。  
哈希表用除留余数法构造，用伪随机探测再散列法处理冲突。

\*\*\*\*\*

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

```
#define HASH_LENGTH 50          //哈希表的长度
#define M 47                    //随机数
#define NAME_NO 30             //人名的个数
```

```
typedef struct
{   char *py;    //名字的拼音
    int k;       //拼音所对应的整数
}NAME;
```

```
NAME NameList[HASH_LENGTH];    //全局变量 NAME
```

```
typedef struct    //哈希表
{   char *py;    //名字的拼音
    int k;       //拼音所对应的整数
    int si;      //查找长度
}HASH;
```

```
HASH HashList[HASH_LENGTH];    //全局变量 HASH
```

```
void InitNameList() //姓名（结构体数组）初始化
```

```
{   char *f;
    int r,s0,i;
for (i=0; i<HASH_LENGTH; i++)
{
    NameList[i].py = new char[64];
    NameList[i].py[0] = 0;
```

```

}
strcpy(NameList[0].py, "wanglu");
    strcpy(NameList[1].py, "lizhen");
    strcpy(NameList[2].py, "zhangjinyang");
    strcpy(NameList[3].py, "wangminqi");
    strcpy(NameList[4].py, "cailankai");
    strcpy(NameList[5].py, "lijie");
    strcpy(NameList[6].py, "zhoujie");
    strcpy(NameList[7].py, "zhaoxiaotian");
    strcpy(NameList[8].py, "zhenzhuowan");
    strcpy(NameList[9].py, "zoudeqiang");
    strcpy(NameList[10].py, "zhangyiying");
    strcpy(NameList[11].py, "qiuliuying");
    strcpy(NameList[12].py, "sunchunyan");
    strcpy(NameList[13].py, "gaofefei");
    strcpy(NameList[14].py, "zhouxun");
    strcpy(NameList[15].py, "lvyuanhao");
    strcpy(NameList[16].py, "wangqian");
    strcpy(NameList[17].py, "tangrong");
    strcpy(NameList[18].py, "wangcaiyan");
    strcpy(NameList[19].py, "makun");
    strcpy(NameList[20].py, "wangjiaqi");
    strcpy(NameList[21].py, "zhaojianan");
    strcpy(NameList[22].py, "lvxuepeng");
    strcpy(NameList[23].py, "zhangyahui");
    strcpy(NameList[24].py, "xiasiyu");
    strcpy(NameList[25].py, "lixin");
    strcpy(NameList[26].py, "liuqianning");
    strcpy(NameList[27].py, "zhangtingting");
    strcpy(NameList[28].py, "cuijianxiong");

```

```

        for(i=0;i<NAME_NO;i++)
    {
        s0=0;
        f=NameList[i].py;
        for(r=0;*(f+r)!='\0';r++)

            s0=*(f+r)+s0;
        NameList[i].k=s0;
    }
}
void CreateHashList() //建立哈希表
{

```

```

int i;
for(i=0; i<HASH_LENGTH;i++)
{
    HashList[i].py=new char[64];
    HashList[i].py[0] = 0;
    HashList[i].k=0;
    HashList[i].si=0;
}
for(i=0;i<HASH_LENGTH;i++)
{
    int sum=0;
    int adr=(NameList[i].k)%M;
    //哈希函数
    int d=adr;
    if(HashList[adr].si==0)    //如果不冲突
    {
        HashList[adr].k=NameList[i].k;
        HashList[adr].py=NameList[i].py;
        HashList[adr].si=1;
    }
    else    //冲突
    {
        while (HashList[d].k!=0)
        {
            d=(d+NameList[i].k%10+1)%M;
            sum=sum+1;
        };
        HashList[d].k=NameList[i].k;
        HashList[d].py=NameList[i].py;
        HashList[d].si=sum+1;
    }
}
}
}
void FindList() //查找
{
    string name;
    int s0=0,r,sum=1,adr,d;
    cout<<"请输入姓名的拼音:"<<endl;
    cin>>name;;
    for(r=0;r<20;r++)    //求出姓名的拼音所对应的整数(关键字)
        s0+=name[r];
    adr=s0%M;    //使用哈希函数
    d=adr;
    if(HashList[adr].k==s0)    //分 3 种情况进行判断

```

```

        cout<<"姓名:"<<HashList[d].py<<" "<<"关键字:"<<s0<<" "<<"查找长度为:
1"<<endl;
        else if (HashList[adr].k==0)
        cout<<"无此记录!"<<endl;
        else
        {
            int g=0;
            while(g==0)
            {
                d=(d+s0%10+1)%M;          //伪随机探测再散列法处理冲突
                sum=sum+1;
                if(HashList[d].k==0)
                {
                    cout<<"无此记录!"<<endl;
                    g=1;
                }
                if(HashList[d].k==s0)
                {
                    cout<<"姓名:"<<HashList[d].py<<" "<<"关键字:"<<s0<<" "<<"查找长度
为:"<<sum<<endl;
                    g=1;
                }
            }
        }
    }
}
void    Display() // 显示哈希表
{
    int i;
    float average=0;

    cout<<"\n 地址\t 关键字\t\t 搜索长度\tH(key)\t 姓名\n"; //显示的格式
    for(i=0; i<50; i++)
    {
        cout<<i<<" ";
        cout<<"\t"<<HashList[i].k<<" ";
        cout<<"\t\t"<<HashList[i].si<<" ";
        cout<<"\t\t"<<(HashList[i].k%M)<<" ";
        cout<<"\t "<<HashList[i].py<<" ";
        cout<<"\n";
    }

    for(i=0;i<HASH_LENGTH;i++)
        average+=HashList[i].si;
    average/=NAME_NO;
    cout<<"平均查找长度: ASL("<<NAME_NO<<")="<<average<<endl;

```

```

}
int main()
{
char x;
InitNameList();
CreateHashList ();
cout<<"d. 显示哈希表 f. 查找 任意键退出    请选择: "<<endl;
while(cin>>x)
{
    if(x=='d')
    {
        Display();
        cout<<endl;
    }
    else if(x=='f')
    {
        FindList();
        cout<<endl;
    }
    else break;
}
for (int i=0; i<HASH_LENGTH; i++)
{
    free(NameList[i].py);
    free(HashList[i].py);
}
return 0;
}

```

## 第 6 题：内部排序算法比较

### 1. 实验内容

#### 【问题描述】

设计一个测试程序比较几种内部排序算法的关键字比较次数和移动次数以取得直观感受。

#### 【基本要求】

(1) 对以下六种常用的内部排序算法进行比较：希尔排序、直接选择排序、快速排序、直接插入排序、堆排序、冒泡排序、。

(2) 待排序表的表长和数据可以由用户自己确定，也可以由随机数产生程序自动产生；至少要用五组不同的输入数据作比较；比较的指标为关键字的比较次数和关键字的移动次数（关键字的交换计为三次移动）。

(3) 最后要对结果作出简单分析。

(4) 使用汉字显示。

#### 【选做内容】

(1) 对不同表长进行比较；

(2) 验证各算法的稳定性；

(3) 各种比较指标值的列表，用饼图或条形图进行表示；

### 2. 需求分析

(1) 本演示程序对以下 6 种常用的内部排序算法进行实测比较：起泡排序，直接插入排序，简单选择排序，快速排序，希尔排序，堆排序。

(2) 待排序表的元素的关键字为整数。用电脑随机产生的不同的数据做测试。比较的指标为有关关键字参加的比较次数，关键字的交换次数和关键字的花费时间（关键字交换记为 3 次移动）。

(3) 演示程序以用户和计算机的对话方式执行，在计算机终端上显示提示信息，用户对随机数组进行排序，并输出比较指标值，通过采用不同的最大



值来比较 6 种排序算法的比较次数，交换次数和花费时间

(4) 最后对结果作出简单分析，包括对各数组数取得出结果波动给予解释。

### 3. 概要设计

#### 3.1 抽象数据类型定义

(1) 可排序表的抽象数据类型定义：

ADT OrderableList{

数据对象：  $D=\{a_i \mid a_i \in \text{IntegerSet}, i=1, 2, \dots, n, n \geq 0\}$

数据关系：  $R1=\{\langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D, i=2, \dots, n\}$

基本操作：

InitList(n)

操作结果：构造一个长度为 n，元素值依次为 1, 2, ..., n 的有序表。

RandomizeList(d, isInverseOrder)

操作结果：首先根据 isInverseOrder 为 True 或 False, 将表置为逆序或正序，然后将表进行 d( $0 \leq d \leq 8$ ) 级随机打乱。d 为 0 时表不打乱，d 越大，打乱程度越高。

RecallList()

操作结果：恢复最后一次用 RandomizeList 随机打乱得到的可排序表。

ListLength()

操作结果：返回可排序表的长度。

ListEmpty()

操作结果：若可排序表为空表，则返回 Ture，否则返回 False。

BubbleSort(&c, &s)

操作结果：进行起泡排序，返回关键字比较次数 c 和移动次数 s。

InsertSort(&c, &s)

操作结果：进行插入排序，返回关键字比较次数 c 和移动次数 s。

SelectSort (&c, &s)

操作结果：进行选择排序，返回关键字比较次数 c 和移动次数 s。

QuickSort(&c, &s)

操作结果：进行快速排序，返回关键字比较次数  $c$  和移动次数  $s$ 。

ShellSort(long &c, long &s)

操作结果：进行希尔排序，返回关键字比较次数  $c$  和移动次数  $s$ 。

HeapSort (&c, &s)

操作结果：进行堆排序，返回关键字比较次数  $c$  和移动次数  $s$ 。

ListTraverse(visit())

操作结果：依次对  $L$  中的每个元素调用函数  $visit()$ 。

}ADT OrderableList

### 3.2 模块划分

本程序包含两个模块：

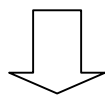
#### 1) 主程序模块

```
void main() {  
    初始化;  
    do {  
        接受命令;  
        处理命令;  
    } while(“命令” != “退出”);  
}
```

#### 2) 可排序表单元模块——实现可排序表的抽象数据类型；

### 3.3 各模块之间的调用关系如下：

主程序模块



可排序表单元模块

## 4. 详细设计

### 4.1 数据类型的定义

```

#define MAXSIZE 5000

#define TRUE 1

#define FALSE 0

typedef int BOOL;

typedef struct{
    int key;
} RedType;

typedef struct LinkList{
    RedType r[MAXSIZE+1];
    int Length;
} LinkList;

int RandArray[MAXSIZE+1];

```

## 4.2 主要模块的算法描述

//希尔排序

```

void ShellInsert(LinkList *L, int dk, int *CmpNum, int *ChgNum){
    int i, j;
    RedType Temp;
    for (i = dk; i < L->Length; i++){
        if (LT(L->r[i].key, L->r[i - dk].key, CmpNum)){
            memcpy(&Temp, &L->r[i], sizeof(RedType));
            for (j = i - dk; j >= 0 && LT(Temp.key, L->r[j].key, CmpNum); j -= dk){
                (*ChgNum)++;
                memcpy(&L->r[j + dk], &L->r[j], sizeof(RedType));
            }
            memcpy(&L->r[j + dk], &Temp, sizeof(RedType));
        }
    }
}

void ShellSort(LinkList *L, int dlta[], int t, int *CmpNum, int *ChgNum){

```

```

    int k;
    for (k = 0; k < t; k++)
        ShellInsert(L, dlta[k], CmpNum, ChgNum);
}

//快速排序
int Partition(LinkList *L, int low, int high, int *CmpNum, int *ChgNum){
    RedType Temp;
    int PivotKey;
    memcpy(&Temp, &L->r[low], sizeof(RedType));
    PivotKey = L->r[low].key;
    while (low < high){
        while (low < high && L->r[high].key >= PivotKey){
            high--;
            (*CmpNum)++;
        }
        (*ChgNum)++;
        memcpy(&L->r[low], &L->r[high], sizeof(RedType));
        while (low < high && L->r[low].key <= PivotKey){
            low++;
            (*CmpNum)++;
        }
        (*ChgNum)++;
        memcpy(&L->r[high], &L->r[low], sizeof(RedType));
    }
    memcpy(&L->r[low], &Temp, sizeof(RedType));
    return low;
}

void QSort(LinkList *L, int low, int high, int *CmpNum, int *ChgNum){
    int PivotLoc = 0;

```

```

    if (low < high){
        PivotLoc = Partition(L, low, high, CmpNum, ChgNum);
        QSort(L, low, PivotLoc - 1, CmpNum, ChgNum);
        QSort(L, PivotLoc + 1, high, CmpNum, ChgNum);
    }
}

void QuickSort(LinkList *L, int *CmpNum, int *ChgNum){
    QSort(L, 0, L->Length - 1, CmpNum, ChgNum);
}

```

//堆排序

```

void HeapAdjust(LinkList *L, int s, int m, int *CmpNum, int *ChgNum){
    RedType Temp;
    int j = 0;
    s++;
    memcpy(&Temp, &L->r[s - 1], sizeof(RedType));
    for (j = 2 * s; j <= m; j *= 2){
        if (j < m && LT(L->r[j - 1].key, L->r[j].key, CmpNum))
            ++j;
        if (!LT(Temp.key, L->r[j - 1].key, CmpNum))
            break;
        (*ChgNum)++;
        memcpy(&L->r[s - 1], &L->r[j - 1], sizeof(RedType));
        s = j;
    }
    memcpy(&L->r[s - 1], &Temp, sizeof(RedType));
}

void HeapSort(LinkList *L, int *CmpNum, int *ChgNum){
    int i = 0;
    RedType Temp;

```

```

for (i = L->Length / 2 - 1; i >= 0; i--)
    HeapAdjust(L, i, L->Length, CmpNum, ChgNum);
for (i = L->Length; i > 1; i--) {
    memcpy(&Temp, &L->r[0], sizeof(RedType));
    (*ChgNum)++;
    memcpy(&L->r[0], &L->r[i - 1], sizeof(RedType));
    memcpy(&L->r[i - 1], &Temp, sizeof(RedType));
    HeapAdjust(L, 0, i - 1, CmpNum, ChgNum);
}
}

```

//冒泡排序

```

void BubbleSort(LinkList *L, int *CmpNum, int *ChgNum){
    int i, j;
    RedType temp;
    for (i = 1; i <= MAXSIZE; i++){
        for (j = 1; j <= MAXSIZE - i; j++){
            if (!LT(L->r[j].key, L->r[j + 1].key, CmpNum)){
                (*ChgNum)++;
                memcpy(&temp, &L->r[j], sizeof(RedType));
                memcpy(&L->r[j], &L->r[j + 1], sizeof(RedType));
                memcpy(&L->r[j + 1], &temp, sizeof(RedType));
            }
        }
    }
}

```

//选择排序

```

int SelectMinKey(LinkList *L, int k, int *CmpNum){
    int Min = k;
    for (; k < L->Length; k++){

```

```

        if (!LT(L->r[Min].key, L->r[k].key, CmpNum))
            Min = k;
    }
    return Min;
}

void SelSort(LinkList *L, int *CmpNum, int *ChgNum){
    int i, j;
    RedType temp;
    for (i = 0; i < L->Length; i++){
        j = SelectMinKey(L, i, CmpNum);
        if (i != j){
            (*ChgNum)++;
            memcpy(&temp, &L->r[i], sizeof(RedType));
            memcpy(&L->r[i], &L->r[j], sizeof(RedType));
            memcpy(&L->r[j], &temp, sizeof(RedType));
        }
    }
}

void RandomNum(){
    int i;
    srand(2000);
    for (i = 1; i <= MAXSIZE; i++){
        RandArray[i] = (int)rand();
    }
}

void InitLinkList(LinkList *L){
    int i;
    memset(L, 0, sizeof(LinkList));
    RandomNum();
    for (i = 1; i <= MAXSIZE; i++){
        L->r[i].key = RandArray[i];
    }
}

```

```
    L->Length = i;
}
```

```
bool LT(int i, int j, int *CmpNum){
    (*CmpNum)++;
    if (i < j)
        return TRUE;
    else
        return FALSE;
}
```

```
void Display(LinkList *L){
    FILE *f;
    int i;
    if ((f = fopen("SortRes.txt", "w")) == NULL){
        printf("can't open file\n");
        exit(0);
    }
    for (i = 0; i < L->Length; i++)
        fprintf(f, "%d\n", L->r[i].key);
    fclose(f);
}
```

```
void main() {
    int i, j;
    int select = 0;
    int dlta[3] = {7, 3, 1};
    int Indata[1] = {1};
    int CmpNum[8], ChgNum[8];
```



```

int SpendTime = 0;
int TempTime;
LinkList L;
InitLinkList(&L);
memset(CmpNum, 0, sizeof(CmpNum));
memset(ChgNum, 0, sizeof(ChgNum));
do{
    SelectSort();
    for (i = 0; i < MAXSIZE; i++)
        L.r[i].key = RandArray[i];           //随机数列复位
    scanf("%d", &select);
    switch (select){
        case 1:
            printf("\n 插入排序:\n");
            TempTime = (int)GetTickCount();
            ShellSort(&L, Indata, 1, &CmpNum[select], &ChgNum[select]);
            SpendTime = (int)GetTickCount() - TempTime;
            for(i=1;i<=MAXSIZE;i++){
                printf("%5d ", L.r[i].key);
                if(++j%10==0)printf("\n");
            }

            printf("\n\nCompareNumber=%d\tChageNumber=%d\tTimeSpent=%dms\n",
CmpNum[select], ChgNum[select], SpendTime);

            break;
        case 2:
            printf("\n 希尔排序:\n");
            TempTime = (int)GetTickCount();
            ShellSort(&L, dlta, 3, &CmpNum[select], &ChgNum[select]);

```

```

        SpendTime = (int)GetTickCount() - TempTime;
    for(i=1;i<=MAXSIZE;i++) {
        printf("%5d ", L.r[i].key);
        if(++j%10==0)printf("\n");
    }

    printf("\n\nCompareNumber=%d\tChageNumber=%d\tTimeSpent=%dms\n",
    CmpNum[select], ChgNum[select], SpendTime);

    break;
case 3:
    printf("\n 快速排序:\n");
    TempTime = (int)GetTickCount();
    QuickSort(&L, &CmpNum[select], &ChgNum[select]);
    SpendTime = (int)GetTickCount() - TempTime;
    for(i=1;i<=MAXSIZE;i++) {
        printf("%5d ", L.r[i].key);
        if(++j%10==0)printf("\n");
    }

    printf("\n\nCompareNumber=%d\tChageNumber=%d\tTimeSpent=%dms\n",
    CmpNum[select], ChgNum[select], SpendTime);

    break;
case 4:
    printf("\n 堆排序:\n");
    TempTime = (int)GetTickCount();
    HeapSort(&L, &CmpNum[select], &ChgNum[select]);
    SpendTime = (int)GetTickCount() - TempTime;
    for(i=1;i<=MAXSIZE;i++) {
        printf("%5d ", L.r[i].key);

```

```

        if(++j%10==0)printf("\n");
    }

    printf("\n\nCompareNumber=%d\tChageNumber=%d\t\tTimeSpent=%dms\n",
    , CmpNum[select], ChgNum[select], SpendTime);
    break;
case 5:
    printf("\n 冒泡排序:\n");
    TempTime = (int)GetTickCount();
    BubbleSort(&L, &CmpNum[select], &ChgNum[select]);
    SpendTime = (int)GetTickCount() - TempTime;
    for(i=1;i<=MAXSIZE;i++) {
        printf("%5d ", L.r[i].key);
        if(++j%10==0)printf("\n");
    }

    printf("\n\nCompareNumber=%d\tChageNumber=%d\t\tTimeSpent=%dms\n",
    CmpNum[select], ChgNum[select], SpendTime);
    break;
case 6:
    printf("\n 选择排序:\n");
    TempTime = (int)GetTickCount();
    SelSort(&L, &CmpNum[select], &ChgNum[select]);
    SpendTime = (int)GetTickCount() - TempTime;
    for(i=1;i<=MAXSIZE;i++) {
        printf("%5d ", L.r[i].key);
        if(++j%10==0)printf("\n");
    }

```

```

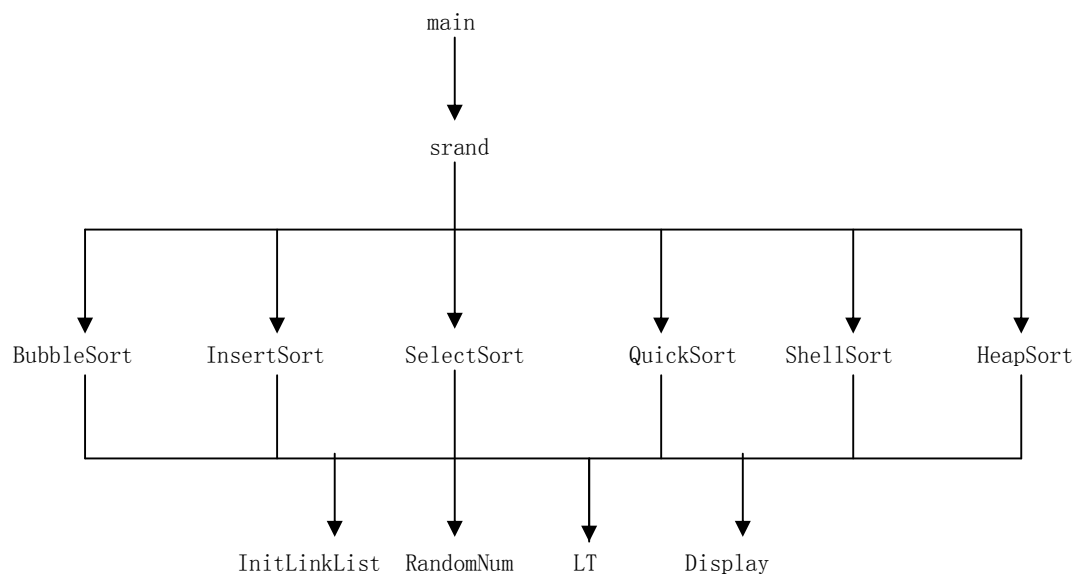
printf("\n\nCompareNumber=%d\tChageNumber=%d\tTimeSpent=%dms\n",
CmpNum[select],ChgNum[select], SpendTime);

    break;
case 7:
    AllAbove (&L, CmpNum, ChgNum);
    break;
}
}

while (select != 8 );
Display (&L);
}

```

#### 4.3 函数之间的调用关系



## 5. 程序运行说明与测试

### 5.1 用户手册和测试数据

用户手册：

- (1) 本程序运行环境为 DOS 操作系统，执行文件为：SortDemo。Exe。
- (2) 进入演示程序后，即显示文本方式用户界面：



有 8 种操作可供选择。在每种选择下，所排列数据由系统随机生成。

输入 **1** 回车，即得插入排序的排序结果及其关键字比较次数和移动次数及时间

输入 **2** 回车，即得希尔排序的排序结果及其关键字比较次数和移动次数及时间

输入 **3** 回车，即得快速排序的排序结果及其关键字比较次数和移动次数及时间

输入 **4** 回车，即得堆排序的排序结果及其关键字比较次数和移动次数及时间

输入 **5** 回车，即得冒泡排序的排序结果及其关键字比较次数和移动次数及时间

输入 **6**，即得选择排序的排序结果及其关键字比较次数和移动次数及时间

输入 **7** 回车，即得以上所有排序的排序结果及其关键字比较次数和移动次数及时间

输入 **8** 回车，即退出该程序，明显看到比输入 7 的时候多了一句

Press any key to continue

运行结果:

输入 1

```
C:\Program Files\Microsoft Visual Studio\MyProjects\xxf\Debug\xxf.exe
32046 32054 32055 32062 32067 32080 32087 32141 32169 32171
32173 32176 32181 32186 32189 32192 32193 32193 32200 32210
32228 32249 32254 32255 32259 32261 32263 32263 32264 32265
32273 32281 32293 32305 32305 32305 32309 32320 32326 32334
32337 32347 32367 32371 32377 32381 32382 32398 32399 32406
32409 32410 32412 32421 32422 32433 32437 32450 32452 32454
32456 32461 32475 32483 32485 32489 32492 32497 32500 32522
32531 32531 32534 32534 32540 32542 32552 32565 32565 32566
32585 32589 32591 32603 32604 32605 32608 32609 32625 32625
32628 32632 32640 32647 32649 32652 32661 32663 32674 32689
32692 32700 32707 32710 32713 32717 32724 32724 32724 32725
32726 32728 32729 32730 32731 32738 32738 32740 32743 32754

比较次数=6176620      交换次数=6166631      花费时间=297ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序

Please Select the Operate:
```

输入 2

```
C:\Program Files\Microsoft Visual Studio\MyProjects\xxf\Debug\xxf.exe
32054 32055 32062 32067 32080 32087 32141 32169 32171 32173
32176 32181 32186 32189 32192 32193 32193 32200 32210 32228
32249 32254 32255 32259 32261 32263 32263 32264 32265 32273
32281 32293 32305 32305 32305 32309 32320 32326 32334 32337
32347 32367 32371 32377 32381 32382 32398 32399 32406 32409
32410 32412 32421 32422 32433 32437 32450 32452 32454 32456
32461 32475 32483 32485 32489 32492 32497 32500 32522 32531
32531 32534 32534 32540 32542 32552 32565 32565 32566 32585
32589 32591 32603 32604 32605 32608 32609 32625 32625 32628
32632 32640 32647 32649 32652 32661 32663 32674 32689 32692
32700 32707 32710 32713 32717 32724 32724 32724 32725 32726
32728 32729 32730 32731 32738 32738 32740 32743 32754 32754

比较次数=963698      交换次数=937286      花费时间=46ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序

Please Select the Operate:
```

### 输入 3

```
C:\ "C:\Program Files\Microsoft Visual Studio\MyProjects\sss\Debug\sss.exe"
32452 32454 32454 32456 32457 32461 32475 32475 32483 32485
32485 32489 32492 32495 32497 32500 32506 32509 32518 32518
32522 32531 32531 32532 32534 32534 32535 32537 32540 32542
32542 32543 32546 32552 32556 32557 32565 32565 32566 32566
32574 32585 32587 32589 32591 32597 32599 32602 32603 32604
32605 32608 32608 32609 32610 32611 32617 32620 32621 32624
32625 32625 32628 32630 32632 32634 32640 32641 32643 32647
32649 32652 32654 32658 32659 32661 32663 32673 32673 32674
32678 32683 32688 32689 32692 32695 32700 32700 32707 32709
32710 32713 32717 32724 32724 32724 32725 32726 32728 32729
32730 32731 32734 32738 32738 32740 32743 32744 32754 32761

比较次数=170876 交换次数=62216 花费时间=0ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序

Please Select the Operate:_
微软拼音 半:
```

### 输入 4

```
C:\ "C:\Program Files\Microsoft Visual Studio\MyProjects\xxf\Debug\xxf.exe"
32054 32055 32062 32067 32080 32087 32141 32169 32171 32173
32176 32181 32186 32189 32192 32193 32193 32200 32210 32228
32249 32254 32255 32259 32261 32263 32263 32264 32265 32273
32281 32293 32305 32305 32305 32309 32320 32326 32334 32337
32347 32367 32371 32377 32381 32382 32398 32399 32406 32409
32410 32412 32421 32422 32433 32437 32450 32452 32454 32456
32461 32475 32483 32485 32489 32492 32497 32500 32522 32531
32531 32534 32534 32540 32542 32552 32565 32565 32566 32585
32589 32591 32603 32604 32605 32608 32609 32625 32625 32628
32632 32640 32647 32649 32652 32661 32663 32674 32689 32692
32700 32707 32710 32713 32717 32724 32724 32724 32725 32726
32728 32729 32730 32731 32738 32738 32740 32743 32754 32754

比较次数=107743 交换次数=57175 花费时间=16ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序

Please Select the Operate:
```

## 输入 5

```
C:\Program Files\Microsoft Visual Studio\MyProjects\xxf\Debug\xxf.exe
32054 32055 32062 32067 32080 32087 32141 32169 32171 32173
32176 32181 32186 32189 32192 32193 32193 32200 32210 32228
32249 32254 32255 32259 32261 32263 32263 32264 32265 32273
32281 32293 32305 32305 32305 32309 32320 32326 32334 32337
32347 32367 32371 32377 32381 32382 32398 32399 32406 32409
32410 32412 32421 32422 32433 32437 32450 32452 32454 32456
32461 32475 32483 32485 32489 32492 32497 32500 32522 32531
32531 32534 32534 32540 32542 32552 32565 32565 32566 32585
32589 32591 32603 32604 32605 32608 32609 32625 32625 32628
32632 32640 32647 32649 32652 32661 32663 32674 32689 32692
32700 32707 32710 32713 32717 32724 32724 32724 32725 32726
32728 32729 32730 32731 32738 32738 32740 32743 32754 32754

比较次数=12497500      交换次数=6463970      花费时间=610ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序

Please Select the Operate:
```

## 输入 6

```
C:\Program Files\Microsoft Visual Studio\MyProjects\xxf\Debug\xxf.exe
32054 32055 32062 32067 32080 32087 32141 32169 32171 32173
32176 32181 32186 32189 32192 32193 32193 32200 32210 32228
32249 32254 32255 32259 32261 32263 32263 32264 32265 32273
32281 32293 32305 32305 32305 32309 32320 32326 32334 32337
32347 32367 32371 32377 32381 32382 32398 32399 32406 32409
32410 32412 32421 32422 32433 32437 32450 32452 32454 32456
32461 32475 32483 32485 32489 32492 32497 32500 32522 32531
32531 32534 32534 32540 32542 32552 32565 32565 32566 32585
32589 32591 32603 32604 32605 32608 32609 32625 32625 32628
32632 32640 32647 32649 32652 32661 32663 32674 32689 32692
32700 32707 32710 32713 32717 32724 32724 32724 32725 32726
32728 32729 32730 32731 32738 32738 32740 32743 32754 32754

比较次数=12507501      交换次数=4991      花费时间=390ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序

Please Select the Operate:
```



输入 7

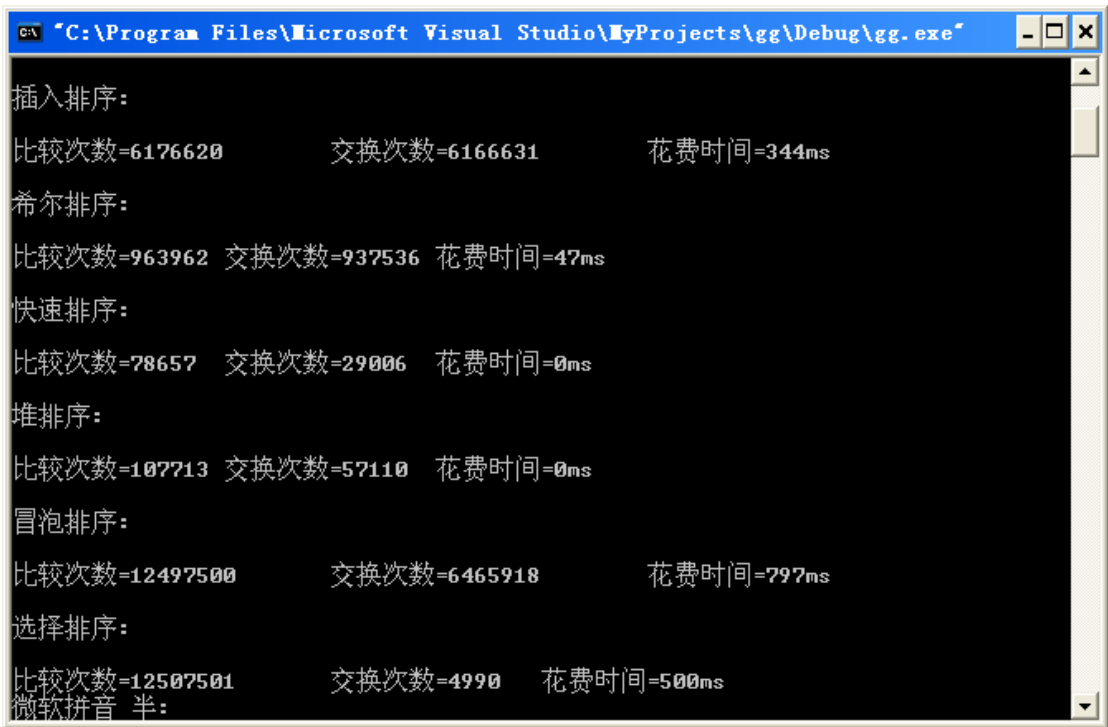
```
C:\Program Files\Microsoft Visual Studio\MyProjects\xxf\Debug\xxf.exe
快速排序:
比较次数=1042355      交换次数=966292  花费时间=0ms
堆排序:
比较次数=187183  交换次数=86058  花费时间=0ms
冒泡排序:
比较次数=12605243      交换次数=6523093      花费时间=531ms
选择排序:
比较次数=25005001      交换次数=6468960      花费时间=313ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序
Please Select the Operate:
```

输入 8

```
C:\Program Files\Microsoft Visual Studio\MyProjects\xxf\Debug\xxf.exe
比较次数=1042355      交换次数=966292  花费时间=0ms
堆排序:
比较次数=187183  交换次数=86058  花费时间=0ms
冒泡排序:
比较次数=12605243      交换次数=6523093      花费时间=531ms
选择排序:
比较次数=25005001      交换次数=6468960      花费时间=313ms
1. 插入排序
2. 希尔排序
3. 快速排序
4. 堆排序
5. 冒泡排序
6. 选择排序
7. 以上所有排序方式
8. 退出程序
Please Select the Operate:8
Press any key to continue
```

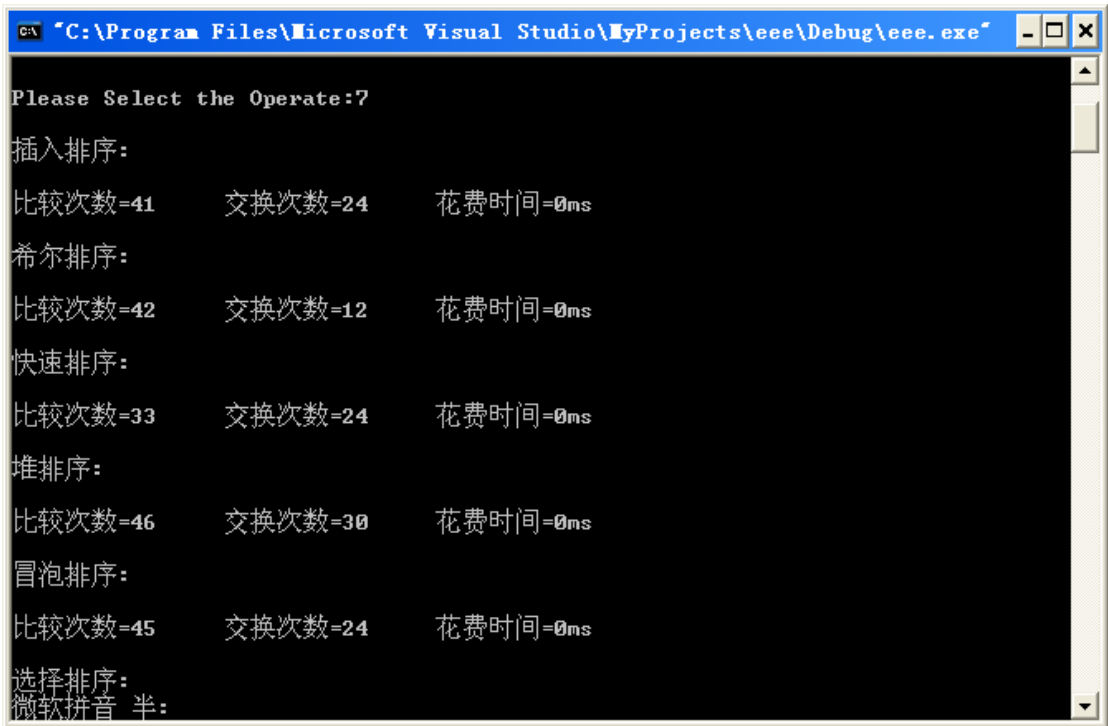
测试结果及分析

当把 MAXSIZE 由 10000 改为 5000 时候，输入 7 回车，运行结果如下：



```
C:\Program Files\Microsoft Visual Studio\MyProjects\gg\Debug\gg.exe
插入排序:
比较次数=6176620      交换次数=6166631      花费时间=344ms
希尔排序:
比较次数=963962  交换次数=937536  花费时间=47ms
快速排序:
比较次数=78657   交换次数=29006   花费时间=0ms
堆排序:
比较次数=107713  交换次数=57110   花费时间=0ms
冒泡排序:
比较次数=12497500  交换次数=6465918  花费时间=797ms
选择排序:
比较次数=12507501  交换次数=4990     花费时间=500ms
微软拼音 半:
```

当把 MAXSIZE 的值由 10000 改为 10，输入 7，运行结果为：



```
C:\Program Files\Microsoft Visual Studio\MyProjects\eee\Debug\eee.exe
Please Select the Operate:7
插入排序:
比较次数=41      交换次数=24      花费时间=0ms
希尔排序:
比较次数=42      交换次数=12      花费时间=0ms
快速排序:
比较次数=33      交换次数=24      花费时间=0ms
堆排序:
比较次数=46      交换次数=30      花费时间=0ms
冒泡排序:
比较次数=45      交换次数=24      花费时间=0ms
选择排序:
微软拼音 半:
```

## 6. 实验总结

对于课程要求的内容总结如下表：

|      |                    |                   |                   |                  |                                  |                  |
|------|--------------------|-------------------|-------------------|------------------|----------------------------------|------------------|
| 测试   | 插入排序               | 希尔排序              | 快速排序              | 堆排序              | 冒泡排序                             | 选择排序             |
| 比较次数 | 第三多<br>越乱（逆）<br>越多 | 少<br>乱否差异<br>小    | 少<br>乱否差异<br>小    | 稍多<br>乱否差异<br>很小 | 最多<br>越乱（逆）<br>越多<br>越乱（否）<br>越多 | 第二多<br>与乱否无<br>关 |
| 移动次数 | 第二多<br>越乱（逆）<br>越多 | 约为快速<br>排序的两<br>倍 | 第二少<br>乱否差异<br>较小 | 稍多<br>乱否差异<br>很小 | 最多<br>越乱（逆）<br>越多                | 最少<br>正和逆序<br>少  |

在程序设计过程中，主要遇到的困难在于利用数组编写有关各种内部排序算法的程序，各种排序算法的主要内容也是难点之一，还有再第七点，也就是将各种程序的比较次数，移动次数，还有所需要的时间进行比较的算法上面，也用着一定的难度，这个程序比较长，在主函数中，如何调用各种算法，以及对于如何产生随机数字的设计上都应该注意的。

本程序跟老师要求的程序中，增加了一定的选做内容，如：

#### (1) 根据比较数字的多少来进行比较

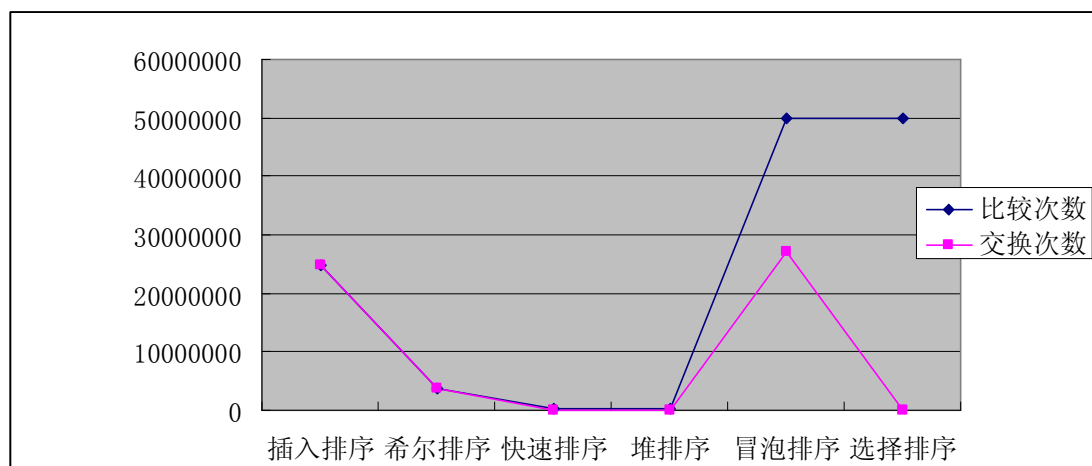
除了程序中的最大值 5000 以外，我们还先后使用了比 5000 小很多倍的 10 和大 2 倍的 10000，在使用 10 作为最大数时候，由于数字小，比较次数跟移动次数很容易就找到差距，但是花费的时间都是 0ms，说明不了问题，而在采取 10000 的时候，

各种数据的数字量非常大，但是时间上很显然就能完整的进行比较，在下结论之前总是要经过多次的对比，找出共同规律，才有说服力。（但是将数据最大值变为 100000 的时候，就很长时间不能出结果，看来计算机硬件水平还有待提高）

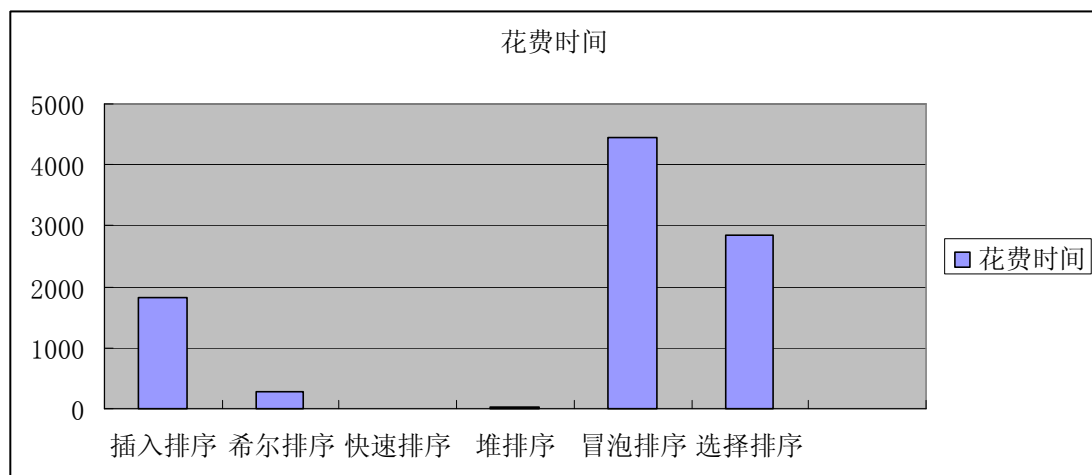
(2) 除了采用不同的数字进行测试外，还多增加了题目中没有要求的每种排序算法所花费的时间的比较，比较结果如下表

|      |                      |                      |  |                      |                     |                     |
|------|----------------------|----------------------|--|----------------------|---------------------|---------------------|
| 测试   | 插入排序                 | 希尔排序                 | 快速排序                                     | 堆排序                  | 冒泡排序                | 选择排序                |
| 花费时间 | 比较多，<br>排在第三<br>多的位置 | 比较少，<br>排在第四<br>多的位置 | 最少的，<br>10000 个<br>数字用时<br>少的位置<br>是 0ms | 比较少，<br>排在第二<br>少的位置 | 最多，用<br>时是最长<br>的一个 | 很多，排<br>在第二多<br>的位置 |

(3) 六种排序算法的比较次数和交换次数的折线图：（单位：次）



花费时间的柱形图：（单位：ms）



## 附：源程序清单

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

```

#include <string.h>
#include <time.h>
#include <windows.h>
#include <winbase.h>
#define MAXSIZE 10000
#define TRUE 1
#define FALSE 0
typedef int BOOL;
typedef struct{
    int key;
} RedType;
typedef struct LinkList{
    RedType r[MAXSIZE+1];
    int Length;
} LinkList;
int RandArray[MAXSIZE+1];

void RandomNum(){
    int i;
    srand(2000);
    for (i = 1; i <= MAXSIZE; i++)
        RandArray[i] = (int)rand();
}

void InitLinkList(LinkList *L){
    int i;
    memset(L, 0, sizeof(LinkList));
    RandomNum();
    for (i = 1; i <= MAXSIZE; i++)
        L->r[i].key = RandArray[i];
    L->Length = i;
}

bool LT(int i, int j, int *CmpNum){
    (*CmpNum)++;
    if (i < j)
        return TRUE;
    else
        return FALSE;
}

void Display(LinkList *L){
    FILE *f;
    int i;

```

```

if ((f = fopen("SortRes.txt", "w")) == NULL){
    printf("can't open file\n");
    exit(0);
}
for (i = 0; i < L->Length; i++)
    fprintf(f, "%d\n", L->r[i].key);
fclose(f);
}

```

//希尔排序

```

void ShellInsert(LinkList *L, int dk, int *CmpNum, int *ChgNum){
    int i, j;
    RedType Temp;
    for (i = dk; i < L->Length; i++){
        if (LT(L->r[i].key, L->r[i - dk].key, CmpNum)){
            memcpy(&Temp, &L->r[i], sizeof(RedType));
            for (j = i - dk; j >= 0 && LT(Temp.key, L->r[j].key, CmpNum); j -= dk){
                (*ChgNum)++;
                memcpy(&L->r[j + dk], &L->r[j], sizeof(RedType));
            }
            memcpy(&L->r[j + dk], &Temp, sizeof(RedType));
        }
    }
}

void ShellSort(LinkList *L, int dlta[], int t, int *CmpNum, int *ChgNum){
    int k;
    for (k = 0; k < t; k++)
        ShellInsert(L, dlta[k], CmpNum, ChgNum);
}

```

//快速排序

```

int Partition(LinkList *L, int low, int high, int *CmpNum, int *ChgNum){
    RedType Temp;
    int PivotKey;
    memcpy(&Temp, &L->r[low], sizeof(RedType));
    PivotKey = L->r[low].key;
    while (low < high){
        while (low < high && L->r[high].key >= PivotKey){
            high--;
            (*CmpNum)++;
        }
        (*ChgNum)++;
        memcpy(&L->r[low], &L->r[high], sizeof(RedType));
        while (low < high && L->r[low].key <= PivotKey){

```

```

        low++;
        (*CmpNum)++;
    }
    (*ChgNum)++;
    memcpy(&L->r[high], &L->r[low], sizeof(RedType));
}
memcpy(&L->r[low], &Temp, sizeof(RedType));
return low;
}
void QSort(LinkList *L, int low, int high, int *CmpNum, int *ChgNum){
    int PivotLoc = 0;
    if (low < high){
        PivotLoc = Partition(L, low, high, CmpNum, ChgNum);
        QSort(L, low, PivotLoc - 1, CmpNum, ChgNum);
        QSort(L, PivotLoc + 1, high, CmpNum, ChgNum);
    }
}
void QuickSort(LinkList *L, int *CmpNum, int *ChgNum){
    QSort(L, 0, L->Length - 1, CmpNum, ChgNum);
}

```

//堆排序

```

void HeapAdjust(LinkList *L, int s, int m, int *CmpNum, int *ChgNum){
    RedType Temp;
    int j = 0;
    s++;
    memcpy(&Temp, &L->r[s - 1], sizeof(RedType));
    for (j = 2 * s; j <= m; j *= 2){
        if (j < m && LT(L->r[j - 1].key, L->r[j].key, CmpNum))
            ++j;
        if (!LT(Temp.key, L->r[j - 1].key, CmpNum))
            break;
        (*ChgNum)++;
        memcpy(&L->r[s - 1], &L->r[j - 1], sizeof(RedType));
        s = j;
    }
    memcpy(&L->r[s - 1], &Temp, sizeof(RedType));
}
void HeapSort(LinkList *L, int *CmpNum, int *ChgNum){
    int i = 0;
    RedType Temp;
    for (i = L->Length / 2 - 1; i >= 0; i--){
        HeapAdjust(L, i, L->Length, CmpNum, ChgNum);
    }
    for (i = L->Length; i > 1; i--){

```

```

        memcpy(&Temp, &L->r[0], sizeof(RedType));
        (*ChgNum)++;
        memcpy(&L->r[0], &L->r[i - 1], sizeof(RedType));
        memcpy(&L->r[i - 1], &Temp, sizeof(RedType));
        HeapAdjust(L, 0, i - 1, CmpNum, ChgNum);
    }
}

```

//冒泡排序

```

void BubbleSort(LinkList *L, int *CmpNum, int *ChgNum){
    int i, j;
    RedType temp;
    for (i = 1; i <= MAXSIZE; i++){
        for (j = 1; j <= MAXSIZE - i; j++){
            if (!LT(L->r[j].key, L->r[j + 1].key, CmpNum)){
                (*ChgNum)++;
                memcpy(&temp, &L->r[j], sizeof(RedType));
                memcpy(&L->r[j], &L->r[j + 1], sizeof(RedType));
                memcpy(&L->r[j + 1], &temp, sizeof(RedType));
            }
        }
    }
}

```

//选择排序

```

int SelectMinKey(LinkList *L, int k, int *CmpNum){
    int Min = k;
    for (; k < L->Length; k++){
        if (!LT(L->r[Min].key, L->r[k].key, CmpNum))
            Min = k;
    }
    return Min;
}

void SelSort(LinkList *L, int *CmpNum, int *ChgNum){
    int i, j;
    RedType temp;
    for (i = 0; i < L->Length; i++){
        j = SelectMinKey(L, i, CmpNum);
        if (i != j){
            (*ChgNum)++;
            memcpy(&temp, &L->r[i], sizeof(RedType));
            memcpy(&L->r[i], &L->r[j], sizeof(RedType));
            memcpy(&L->r[j], &temp, sizeof(RedType));
        }
    }
}

```



```
    }  
}
```

```
void SelectSort(){  
    printf("1. 插入排序\n");  
    printf("2. 希尔排序\n");  
    printf("3. 快速排序\n");  
    printf("4. 堆排序\n");  
    printf("5. 冒泡排序\n");  
    printf("6. 选择排序\n");  
    printf("7. 以上所有排序方式\n");  
    printf("8. 退出程序\n\n");  
    printf("Please Select the Operate:");  
}
```

```
void AllAbove(LinkList *L, int *CmpNum, int *ChgNum){  
    int TempTime, i,j;  
    int SpendTime;  
    int dlta[3] = {  
        7, 3, 1  
    };  
    int Indata[1] = {  
        1  
    };  
    for (i = 1; i <= MAXSIZE; i++)  
        L->r[i].key = RandArray[i];           //随机数列复位(插入排序)  
    printf("\n 插入排序:\n");  
    TempTime = (int)GetTickCount();  
    ShellSort(L, Indata, 1, &CmpNum[0], &ChgNum[0]);  
    SpendTime = (int)GetTickCount() - TempTime;  
    printf("\n 比较次数=%d\t 交换次数=%d\t 花费时间=%dms\n", CmpNum[0],  
ChgNum[0],SpendTime);  
    for (i = 1; i <= MAXSIZE; i++)  
        L->r[i].key = RandArray[i];           //随机数列复位(希尔排序)  
    printf("\n 希尔排序:\n");  
    TempTime = (int)GetTickCount();  
    ShellSort(L, dlta, 3, &CmpNum[1], &ChgNum[1]);  
    SpendTime = (int)GetTickCount() - TempTime;  
    printf("\n 比较次数=%d\t 交换次数=%d\t 花费时间=%dms\n", CmpNum[1],  
ChgNum[1],SpendTime);  
    for (i = 1; i <= MAXSIZE; i++)  
        L->r[i].key = RandArray[i];           //随机数列复位(快速排序)  
    printf("\n 快速排序:\n");  
    TempTime = (int)GetTickCount();
```

```

QuickSort(L, &CmpNum[2], &ChgNum[2]);
SpendTime = (int)GetTickCount() - TempTime;
printf("\n 比较次数=%d\t 交换次数=%d\t 花费时间=%dms\n", CmpNum[2],
ChgNum[2],SpendTime);
for (i = 1; i <= MAXSIZE; i++)
    L->r[i].key = RandArray[i];                //随机数列复位(堆排序)
printf("\n 堆排序:\n");
TempTime = (int)GetTickCount();
HeapSort(L, &CmpNum[3], &ChgNum[3]);
SpendTime = (int)GetTickCount() - TempTime;
printf("\n 比较次数=%d\t 交换次数=%d\t 花费时间=%dms\n", CmpNum[3],
ChgNum[3],SpendTime);
for (i = 1; i <= MAXSIZE; i++)
    L->r[i].key = RandArray[i];                //随机数列复位 (冒泡排序)
printf("\n 冒泡排序:\n");
TempTime = (int)GetTickCount();
BubbleSort(L, &CmpNum[4], &ChgNum[4]);
SpendTime = (int)GetTickCount() - TempTime;
printf("\n 比较次数=%d\t 交换次数=%d\t 花费时间=%dms\n", CmpNum[4],
ChgNum[4],SpendTime);
for (i = 1; i <= MAXSIZE; i++)
    L->r[i].key = RandArray[i];                //随机数列复位(选择排序)
printf("\n 选择排序:\n");
TempTime = (int)GetTickCount();
SelSort(L, &CmpNum[5], &ChgNum[5]);
SpendTime = (int)GetTickCount() - TempTime;
printf("\n 比较次数=%d\t 交换次数=%d\t 花费时间=%dms\n", CmpNum[5],
ChgNum[5],SpendTime);
}
void main(){
    int i,j;
    int select = 0;
    int dlta[3] = {7, 3, 1};
    int Indata[1] = {1};
    int CmpNum[8], ChgNum[8];
    int SpendTime = 0;
    int TempTime;
    LinkList L;
    InitLinkList(&L);
    memset(CmpNum, 0, sizeof(CmpNum));
    memset(ChgNum, 0, sizeof(ChgNum));
    do{
        SelectSort();
        for (i = 0; i < MAXSIZE; i++)

```

```

    L.r[i].key = RandArray[i];                //随机数列复位
scanf("%d", &select);
switch (select){                             //调用各种算法
    case 1:
        printf("\n 插入排序:\n");
        TempTime = (int)GetTickCount();
        ShellSort(&L, Indata, 1, &CmpNum[select], &ChgNum[select]);
        SpendTime = (int)GetTickCount() - TempTime;
        for(i=1;i<=MAXSIZE;i++){
            printf("%5d ",L.r[i].key);
            if(++j%10==0)printf("\n");
        }
        printf("\n\n 比较次数=%d\t 交换次数=%d\t\t 花费时间=%dms\n",
CmpNum[select],ChgNum[select], SpendTime);
        break;
    case 2:
        printf("\n 希尔排序:\n");
        TempTime = (int)GetTickCount();
        ShellSort(&L, dlta, 3, &CmpNum[select], &ChgNum[select]);
        SpendTime = (int)GetTickCount() - TempTime;
        for(i=1;i<=MAXSIZE;i++){
            printf("%5d ",L.r[i].key);
            if(++j%10==0)printf("\n");
        }
        printf("\n\n 比较次数=%d\t 交换次数=%d\t\t 花费时间=%dms\n",
CmpNum[select],ChgNum[select], SpendTime);
        break;
    case 3:
        printf("\n 快速排序:\n");
        TempTime = (int)GetTickCount();
        QuickSort(&L, &CmpNum[select], &ChgNum[select]);
        SpendTime = (int)GetTickCount() - TempTime;
        for(i=1;i<=MAXSIZE;i++){
            printf("%5d ",L.r[i].key);
            if(++j%10==0)printf("\n");
        }
        printf("\n\n 比较次数=%d\t 交换次数=%d\t\t 花费时间=%dms\n",
CmpNum[select],ChgNum[select], SpendTime);
        break;
    case 4:
        printf("\n 堆排序:\n");
        TempTime = (int)GetTickCount();
        HeapSort(&L, &CmpNum[select], &ChgNum[select]);
        SpendTime = (int)GetTickCount() - TempTime;

```

```

        for(i=1;i<=MAXSIZE;i++){
            printf("%5d ",L.r[i].key);
            if(++j%10==0)printf("\n");
        }
        printf("\n\n 比较次数=%d\t 交换次数=%d\t\t 花费时间
        =%dms\n",CmpNum[select], ChgNum[select], SpendTime);
        break;
    case 5:
        printf("\n 冒泡排序:\n");
        TempTime = (int)GetTickCount();
        BubbleSort(&L, &CmpNum[select], &ChgNum[select]);
        SpendTime = (int)GetTickCount() - TempTime;
        for(i=1;i<=MAXSIZE;i++){
            printf("%5d ",L.r[i].key);
            if(++j%10==0)printf("\n");
        }

        printf("\n\n 比较次数=%d\t 交换次数=%d\t\t 花费时间=%dms\n",
        CmpNum[select],ChgNum[select], SpendTime);
        break;
    case 6:
        printf("\n 选择排序:\n");
        TempTime = (int)GetTickCount();
        SelSort(&L, &CmpNum[select], &ChgNum[select]);
        SpendTime = (int)GetTickCount() - TempTime;
        for(i=1;i<=MAXSIZE;i++){
            printf("%5d ",L.r[i].key);
            if(++j%10==0)printf("\n");
        }
        printf("\n\n 比较次数=%d\t 交换次数=%d\t\t 花费时间=%dms\n",
        CmpNum[select],ChgNum[select], SpendTime);
        break;
    case 7:
        AllAbove(&L, CmpNum, ChgNum);
        break;

}
while (select != 8 );
Display(&L);
}

```

