

华中科技大学

课程实验报告

课程名称： 数据结构实验

专业班级： 信息安全 201901

学 号： U201911658

姓 名： 李欣宇

指导教师： 朱建新

报告日期： 2020 年 12 月 1 日

网络空间安全学院

目录

| | |
|------------------------------|-----------|
| 1 基于链式存储结构的线性表实现..... | 1 |
| 1.1 问题描述..... | 1 |
| 1.1.1 线性表的基本概念..... | 1 |
| 1.1.2 线性表的 ADT 定义..... | 1 |
| 1.2 系统设计..... | 4 |
| 1.2.1 数据类型及数据物理结构..... | 4 |
| 1.2.2 含菜单的演示系统的设计思想..... | 4 |
| 1.2.3 功能的算法思想与设计..... | 5 |
| 1.2.4 复杂度分析..... | 8 |
| 1.3 系统实现..... | 9 |
| 1.3.1 软硬件环境..... | 9 |
| 1.3.2 头文件及预定义常量说明..... | 9 |
| 1.3.3 系统菜单演示..... | 10 |
| 1.3.4 正常用例测试..... | 10 |
| 1.3.5 异常用例测试..... | 19 |
| 1.4 实验小结..... | 21 |
| 2 基于二叉链表的二叉树实现..... | 22 |
| 2.1 问题描述..... | 22 |
| 2.1.1 二叉树的基本概念..... | 22 |
| 2.1.2 二叉树的逻辑结构与基本运算..... | 22 |
| 2.2 系统设计..... | 28 |
| 2.2.1 数据类型及数据物理结构..... | 28 |
| 2.2.2 含菜单的演示系统的设计思想..... | 28 |
| 2.2.3 重要函数的设计思想..... | 29 |
| 2.3 系统实现..... | 32 |
| 2.3.1 软硬件环境..... | 32 |
| 2.3.2 头文件及预定义常量说明..... | 32 |

| | |
|-----------------------------|----|
| 2.3.3 系统菜单演示..... | 32 |
| 2.3.4 正常用例测试..... | 33 |
| 2.3.5 异常用例测试..... | 45 |
| 2.4 实验小结..... | 47 |
| 参考文献..... | 48 |
| 附录 A：基于链式存储结构的线性表实现源代码..... | 49 |
| 附录 B：基于二叉链表的二叉树实现源代码..... | 71 |

1 基于链式存储结构的线性表实现

1.1 问题描述

实验环境为 Win10 系统下的 Dev c++ 软件，采用顺序表的物理结构，构造一个具有菜单功能的演示系统，实现多线性表管理，包括初始化表、销毁表、清空表、判断表空、求表长、获得元素、查找元素、获得前驱后继、插入删除元素、遍历等 12 种基本功能，同时在此基础上进行功能扩展，实现两个一元稀疏多项式的加减运算。输入形式根据菜单的提示进行相关功能的代号输入与相关的参数输入，输出为对应功能函数执行结果与一些恰当的操作提示。

1.1.1 线性表的基本概念

线性表是具有相同特性元素的一个有限序列。该序列中所含元素个数是线性表的长度，用 n 表示，当 $n=0$ 时，线性表为空。线性表只有一个表头元素，只有一个表尾元素，表头元素没有前驱，表尾元素没有后继，除表头和表尾元素之外，其他元素只有一个直接前驱，也只有一个直接后继。线性表的存储结构分为两种：顺序存储结构和链式存储结构。

1.1.2 线性表的 ADT 定义

ADT List{

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

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

基本操作：

InitList(&L)

操作名称：初始化表

初始条件：线性表 L 不存在

操作结果：构造一个空的线性表

DestroyList(&L)

操作名称：销毁表

初始条件：线性表 L 存在

操作结果：销毁线性表 L

ClearList(&L)

操作名称：清空表

初始条件：线性表 L 存在

操作结果：将线性表 L 置为空表

ListEmpty(L)

操作名称：判断空表

初始条件：线性表 L 存在

操作结果：若 L 为空表，返回 TRUE，否则返回 FALSE

ListLength(L)

操作名称：求表长

初始条件：线性表 L 存在

操作结果：返回表中数据元素的个数

GetElem(L,i,&e)

操作名称：获取元素

初始条件：线性表 L 存在，且 $1 \leq i \leq \text{ListLength}(L)$

操作结果：用 e 返回 L 中第 i 个数据元素的值

LocateElem(L,e,compare())

操作名称：查找元素

初始条件：线性表 L 存在

操作结果：返回 L 中第一个值与 e 满足 compare 关系的元素在 L 中的位置，如果不存在返回-1

PriotElem(L,cur_e,pre_e)

操作名称：获得前驱

初始条件：线性表 L 存在

操作结果：若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e

返回它的前驱，否则操作失败，pre_e 无定义

NextElem(L,cur_e,next_e)

操作名称：获得后继

初始条件：线性表 L 存在

操作结果：若 cur_e 是 L 的数据元素，且不是最后一个，则用
next_e 返回它的后继，否则操作失败，next_e 无定义

ListInsert(&L,i,e)

操作名称：插入元素

初始条件：线性表 L 存在，且 $1 \leq i \leq \text{ListLength}(L)$

操作结果：将元素 e 插入到第 i 个位置之前

ListDelete(&L,i,&e)

操作名称：删除元素

初始条件：线性表 L 存在，且 $1 \leq i \leq \text{ListLength}(L)$

操作结果：将表中第 i 个元素，并用 e 返回该元素的值

ListTraverse(L,visit())

操作名称：遍历

初始条件：线性表 L 存在

操作结果：依次对 L 的每个数据元素调用函数 visit()

}ADT List

1.2 系统设计

1.2.1 数据类型及数据物理结构

数据类型：

```
typedef char ElemType;    //基本线性表元素类型
typedef double ElemType_D; //多项式线性表元素类型
```

数据物理结构：

```
LinkList array[11]; //数组实现多个线性表管理
typedef struct LNode{ //基于链表的线性表
    ElemType_D coe; //多项式系数（扩展）
    int expo; //多项式指数（扩展）
    ElemType data; //数据
    struct LNode* next; //指针
}LNode, *LinkList;
```

1.2.2 含菜单的演示系统的设计思想

将菜单写入 menu()函数中，菜单给予用户代码与功能的对应关系，便于用户输入代码进行功能选择。

功能选择使用一个 while 循环，其内使用 switch-case 结构对用户输入的代码进行相应功能函数实现，每一次功能实现后均使用 system("cls")进行清屏和 system("pause")实现菜单返回功能，直到用户输入 '0' 退出系统。

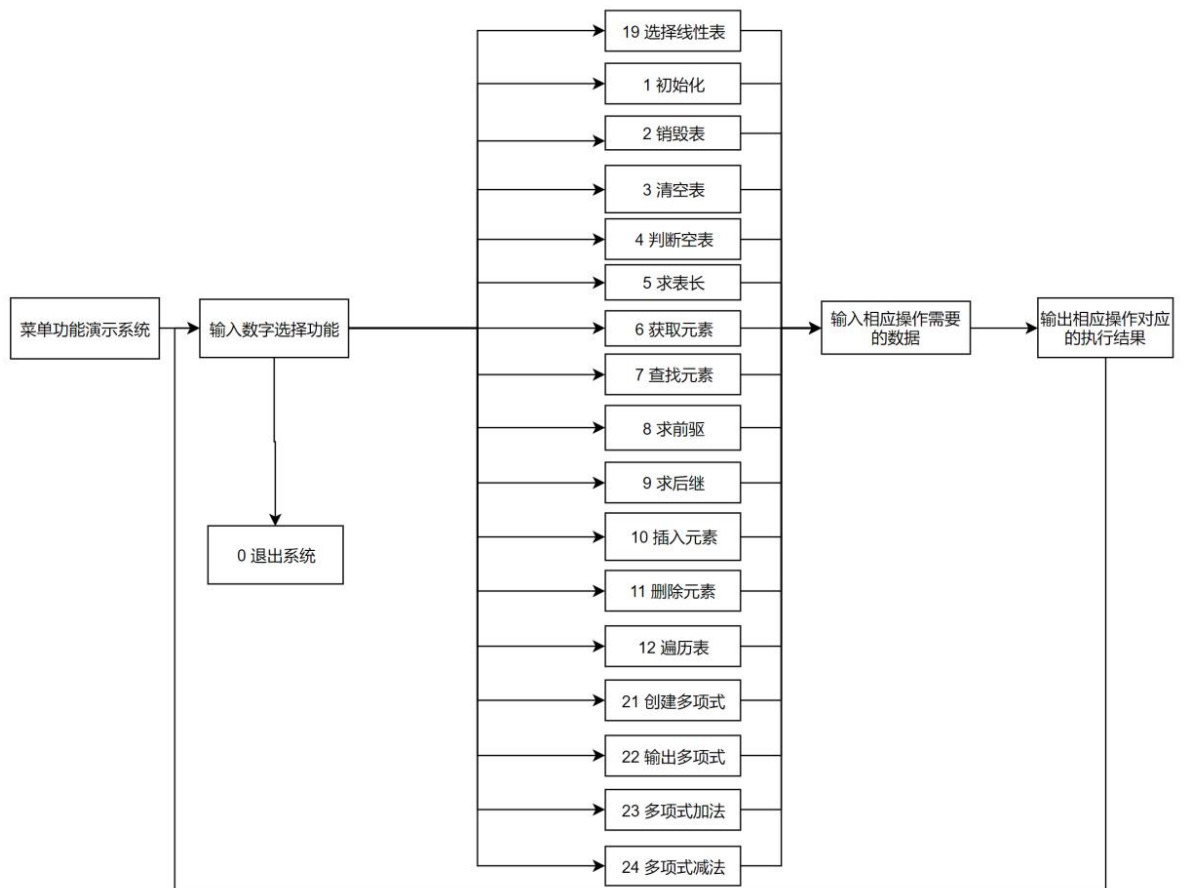


图 1-1 演示系统流程图

1.2.3 功能的算法思想与设计

基本功能：

1.初始化（InitList）

函数实现，传入指针 L，使用 malloc 开辟空间，开辟后判断是否溢出，无溢出则把 L->next 设置为 NULL，并把 L 存入如多线性表管理的数组中。

2.销毁表（DestroyList）

函数实现，传入表头指针 L，使用 while 循环判断 L 是否为空，不为空则把 p 赋值为 L->next，使用 free 释放当前 L 指向的存储空间，L=p。

3.清空表（ClearList）

思想与销毁表类似，唯一不同点在于，表头保留，不释放空间，把除表头外的后续节点全部释放。

4.判定空表（ListEmpty）

函数实现，传入表头指针 **L**，如果 **L** 为 **NULL** 则返回 **TRUE**，即为空表，否则返回 **FALSE**，表不为空。

5.求表长（ListLength）

函数实现，传入表头 **L**，使用 **i** 进行长度记录，先判断是否只有表头节点，若只有表头结点则返回 **0**，否则 **L** 指向第一个节点，进入 **while** 循环，如果 **L** 不为空，则 **i++**，**L** 指向下一个节点，直到 **L** 为空，返回 **i** 即为表长。

6.获得元素（GetElem）

函数实现，使用循环变量 **j** 和 **while** 循环，找到 **i** 的前一个元素，使用 **e** 返回第 **i** 个位置的元素。

7.查找元素（LocateElem）

函数实现，使用 **while** 循环，从第一个节点开始比较数据域是否与要查找的元素相同，只要找到相同的，则返回该位置，若找到表尾都未找到则返回 **ERROR**，表实未找到。

8.获得前驱（PriorElem）

函数实现，另外使用一个节点指针 **p** 记录前一个节点，当当前 **L** 指向的节点数据域与当前 **e** 相同时，返回 **p** 指向的数据域，即为前驱，并修改当前节点 **e**，需要特判是否是第一个元素，第一个元素没有前驱。

9.获得后继（NextElem）

函数实现，思想与获得前驱类似，不同的是用另一个节点指针 **p** 记录后一个节点，需要特判是否是最后一个元素，最后一个元素没有后继。

10.插入元素（ListInsert）

函数实现，使用 **while** 循环定位到 **i-1** 的位置，**malloc** 开辟空间，对该节点进行链接，前驱的 **next** 指向该节点，该节点的 **next** 指向原后继。

11.删除元素（ListDelete）

函数实现，使用 **while** 循环定位到 **i-1** 位置，将该节点的 **next** 指向第 **i+1** 个位置，使用 **e** 保存第 **i** 个节点的数据域，用于返回，使用 **free** 释放空间。

12.遍历表（ListTraverse）

函数实现，使用 **while** 循环，从表头开始，每个节点的数据域使用 **visit** 输出，输出后指向下一个节点，直到到达表尾。

19.设置表号 (SetListnum)

函数实现，对全局变量 num 进行赋值。

扩展功能：

21.建立多项式 (CreateList)

在原有节点结构体中扩展两个数据类型，coe 和 exp 用于保存系数和指数。

另设一个结构体存放系数和指数，并定义一个结构体数组，先将原始数据输入到该结构体数组，使用 sort 函数进行以指数为关键字的降序排序。

然后使用头插法将多项式每一项插入到线性表中，需要注意的是，在插入的时候需要进行合并同类项，使用一个 while 循环即可完成，判断当前项的指数是否与下一项相同，相同则系数合并，i+1，直到不相同，然后进行插入即可。

22.输出多项式 (ShowList)

函数实现，与线性表简单遍历不同的是，输出多项式需要对结果为 0、系数为 0、系数为 1、指数为 1、系数为正等多种情况进行判断，保证输出结果符合常规。这里另调用了两个函数，exp_print 进行指数输出，主要特判是否为 1，coe_print 进行系数的输出，特判①系数是否为正数，为正数时又是否是第一项，非第一项均要输出+，第一项不输出+，负数正常输出；②系数是否为 1，为 1 则不输出。

23.多项式加法 (AddList)

函数实现，使用 while 循环，条件为 L1 和 L2 均不为 NULL，则比较他们当前指向的节点的数据域，相等则系数相加插入 L3，大于则 L1 当前节点直接插入 L3，小于则 L2 当前节点直接插入 L3。while 循环结束后，把 L1 或 L2 剩余部分直接插入 L3 中。

24.多项式减法 (MinusList)

函数实现，思想与加法类似，不同的是，指向的指数 L1 小于 L2 时，要把 L2 指向节点的系数取负再插入 L3。

1.2.4 复杂度分析

| 函数 | 时间复杂度 | 函数 | 时间复杂度 |
|-------------|--------|--------------|--------|
| InitList | $O(1)$ | LocateElem | $O(N)$ |
| DestroyList | $O(N)$ | PriorElem | $O(N)$ |
| ClearList | $O(N)$ | NextElem | $O(N)$ |
| ListEmpty | $O(1)$ | ListInsert | $O(N)$ |
| ListLength | $O(N)$ | ListDelete | $O(N)$ |
| GetElem | $O(N)$ | ListTraverse | $O(N)$ |

表 1-1 基本功能函数时间复杂度

1.3 系统实现

1.3.1 软硬件环境

本此次实验在 win10 专业版下，采用 Dev c++ 编程软件进行编写与编译运行。

1.3.2 头文件及预定义常量说明

1.头文件

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<iostream>
#include<windows.h>
```

```
using namespace std;
```

2.预定义常量

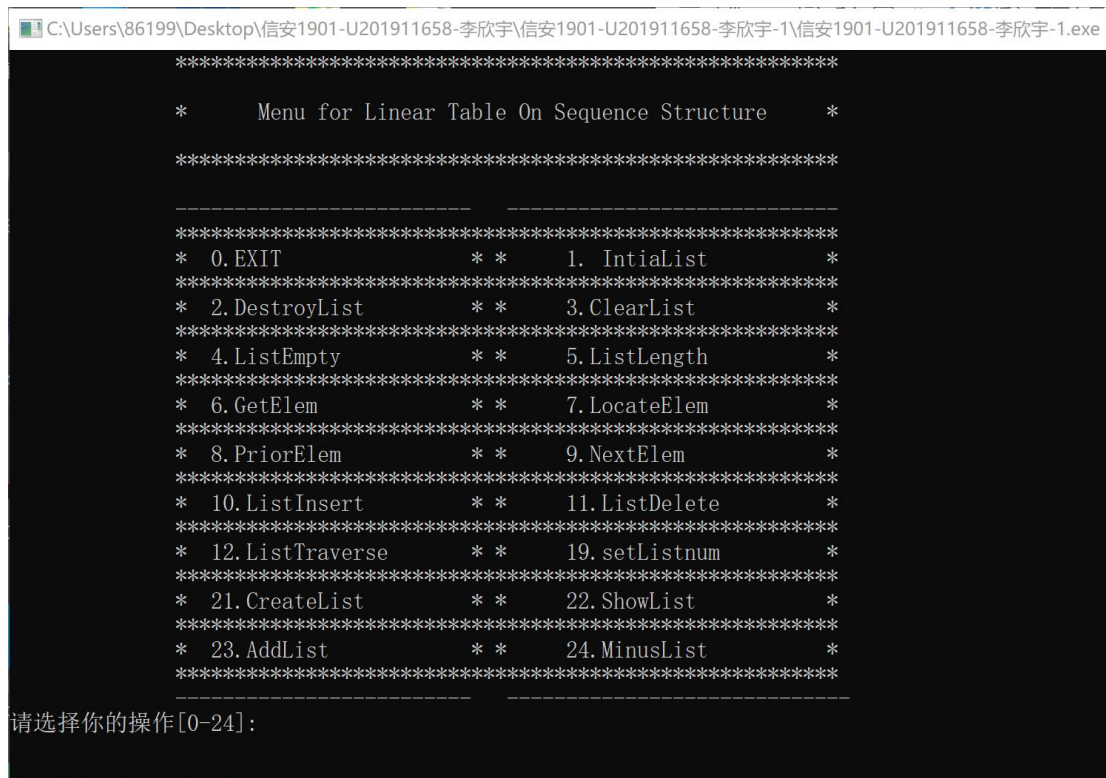
```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
```

3.类型表达式

```
typedef char ElemType; //元素类型
typedef int Status;
typedef double ElemType_D; //多项式元素类型
```

1.3.3 系统菜单演示

初始界面为菜单界面，并提示用户进行操作，在每次一个操作执行完毕后均会回到菜单界面



```

C:\Users\86199\Desktop\信安1901-U201911658-李欣宇\信安1901-U201911658-李欣宇-1\信安1901-U201911658-李欣宇-1.exe
*****
*      Menu for Linear Table On Sequence Structure      *
*****

-----

*****
* 0.EXIT          * *      1. IntialList          *
*****
* 2.DestroyList   * *      3.ClearList            *
*****
* 4.ListEmpty     * *      5.ListLength           *
*****
* 6.GetElem       * *      7.LocateElem           *
*****
* 8.PriorElem     * *      9.NextElem             *
*****
* 10.ListInsert   * *     11.ListDelete            *
*****
* 12.ListTraverse * *     19.setListnum            *
*****
* 21.CreateList   * *     22.ShowList              *
*****
* 23.AddList      * *     24.MinusList             *
*****

-----

请选择你的操作[0-24]:
    
```

图 1-2 系统初始菜单界面

1.3.4 正常用例测试

基本功能测试:

1. 19 号功能

输入: 19

1

功能: 选择编号为 1 的线性表

理论结果: 当前 num 为 1

实际输出:

```
请选择你的操作[0-24]:19
请输入您要设置的线性表编号: 1
设置成功!
请按任意键继续. . .
```

2. 1 号功能

输入: 1

功能: 初始化

理论结果: 实现对表 1 的初始化

实际输出:

```
请选择你的操作[0-24]:1
线性表创建成功!
请按任意键继续. . .
```

3. 10 号功能

输入: 10

1 H

10

2 U

10

3 T

10

3 S

功能: 插入元素

理论结果: 插入后为 HUST

实际输出:

```
请选择你的操作[0-24]:10
请输入您要插入元素的位置与元素（中间以空格隔开）: 1 H
插入成功!
请按任意键继续. . .
```

```
-----
请选择你的操作[0-24]:10
请输入您要插入元素的位置与元素（中间以空格隔开）: 2 U
插入成功!
请按任意键继续.
```

```
请选择你的操作[0-24]:10
请输入您要插入元素的位置与元素（中间以空格隔开）: 3 T
插入成功!
请按任意键继续. . .
```

```
请选择你的操作[0-24]:10
请输入您要插入元素的位置与元素（中间以空格隔开）: 3 S
插入成功!
请按任意键继续. . .
```

4. 12 号功能

输入: 12

功能: 遍历表

理论结果: 结果为 HUST

实际输出:

```
请选择你的操作[0-24]:12
该表遍历结果如下
HUST
请按任意键继续. . .
```

5. 4 号功能

输入: 4

功能: 判定表空

理论结果: 不为空

实际输出:

```
-----
请选择你的操作[0-24]:4
该线性表不为空
请按任意键继续. . .
```

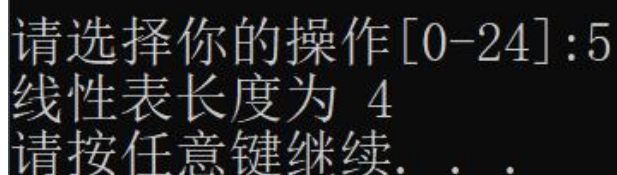
6. 5 号功能

输入： 5

功能：求表长

理论结果：表长为 4

实际输出：



请选择你的操作[0-24]:5
线性表长度为 4
请按任意键继续. . .

7. 6 号功能

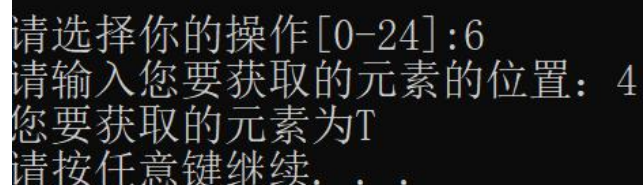
输入： 6

4

功能：获取位置为 4 的元素

理论结果：元素为 T

实际输出：



请选择你的操作[0-24]:6
请输入您要获取的元素的位置: 4
您要获取的元素为T
请按任意键继续. . .

8. 7 号功能

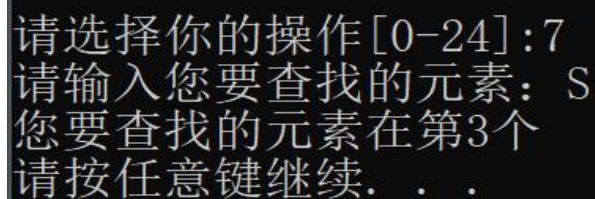
输入： 7

S

功能：查找元素 S 的位置

理论结果：位置为 3

实际输出：



请选择你的操作[0-24]:7
请输入您要查找的元素: S
您要查找的元素在第3个
请按任意键继续. . .

9. 8 号功能

输入： 8

8

功能：求两次 S 的前驱

理论结果：第一次为 U，第二次为 H

实际输出：

```
请选择你的操作[0-24]:8
当前元素的前驱是U
请按任意键继续. . .
```

```
请选择你的操作[0-24]:8
当前元素的前驱是H
请按任意键继续. . .
```

10. 9 号功能

输入： 9

功能：求后继

理论结果：后继为 U

实际输出：

```
请选择你的操作[0-24]:9
当前元素的后继是U
请按任意键继续. . .
```

11. 11 号功能

输入： 11

3

功能：删除位置 3 的元素

理论结果：删除元素 S

实际输出：

```
请选择你的操作[0-24]:11
请输入您要删除的元素的位置：3
您要删除的元素是S
元素删除成功！
请按任意键继续. . .
```

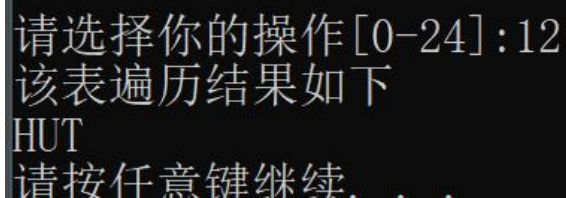
12. 12 号功能

输入： 12

功能：遍历输出

理论结果：HUT

实际输出：



```
请选择你的操作[0-24]:12
该表遍历结果如下
HUT
请按任意键继续...
```

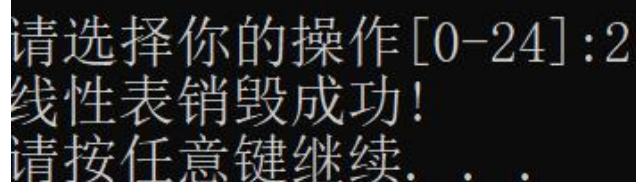
13. 2 号功能

输入： 2

功能：销毁表

理论结果：销毁表 1

实际输出：



```
请选择你的操作[0-24]:2
线性表销毁成功!
请按任意键继续...
```

扩展多项式功能测试：

1. 21 号功能

输入： 19

2

1

21

5

-1 4 1 5 6 9 -1 9 2 0

功能：选择表 2，初始化后，输入多项式

理论结果：多项式为 $5x^9 + x^5 - x^4 + 2$

实际输出：

```

请选择你的操作[0-24]:21
请输入您要输入的多项式的项数: 5
请依次输入多项式的系数与指数: -1 4 1 5 6 9 -1 9 2 0
多项式存储成功!
请按任意键继续. . .
    
```

2. 22 号功能

输入: 22

2

功能: 输出多项式

理论结果: 输出多项式 $2, 5x^9 + x^5 - x^4 + 2$

实际输出:

```

请选择你的操作[0-24]:22
请输入您要查看的多项式的编号:
2
您要查看的多项式如下:
5x^9+x^5-x^4+2
请按任意键继续. . .
    
```

3. 21 号功能

输入: 19

3

1

21

6

1 5 -6 9 4 8 3 4 3 3 1 1

功能: 选择表 2, 初始化后, 输入多项式

理论结果: 选择表 2, 初始化后, 输入多项式

实际输出:

```

请选择你的操作[0-24]:21
请输入您要输入的多项式的项数: 6
请依次输入多项式的系数与指数: 1 5 -6 9 4 8 3 4 3 3 1 1
多项式存储成功!
请按任意键继续. . .
    
```

4. 22 号功能

输入： 22

3

功能：输出多项式 3

理论结果：多项式 3 为 $-6x^9+4x^8+x^5+3x^4+3x^3+x$

实际输出：

```
请选择你的操作[0-24]:22
请输入您要查看的多项式的编号:
3
您要查看的多项式如下:
-6x^9+4x^8+x^5+3x^4+3x^3+x
请按任意键继续. . .
```

5. 23 号功能

输入： 19

4

1

23

2 3 4

功能：对 2 和 3 进行多项式加法，存到多项式 4 中

理论结果：结果为 $-6x^9+4x^8+x^5+3x^4+3x^3+x$

实际输出：

```
请选择你的操作[0-24]:23
请依次输入您要加法运算的两个多项式编号与存放结果多项式的编号:
2 3 4
加法运算结束!
请按任意键继续. . .
```

6. 22 号功能

输入： 22

4

功能：输出结果多项式 4

理论结果：结果为 $-6x^9+4x^8+x^5+3x^4+3x^3+x$

实际输出：

```

请选择你的操作[0-24]:22
请输入您要查看的多项式的编号:
4
您要查看的多项式如下:
-x^9+4x^8+2x^5+2x^4+3x^3+x+2
请按任意键继续. . .
    
```

7. 24 号功能

输入: 19

5

1

24

2 3 5

功能: 多项式 2 和 3 作减法, 存到多项式 5

理论结果: 结果为 $11x^9-4x^8-4x^4-3x^3-x+2$

实际输出:

```

请选择你的操作[0-24]:24
请依次输入您要作减法运算的两个多项式编号与存放结果多项式的编号:
2 3 5
减法运算结束!
请按任意键继续. . .
    
```

8. 22 号功能

输入: 22

5

功能: 输出结果多项式 5

理论结果: 结果为 $11x^9-4x^8-4x^4-3x^3-x+2$

实际输出:

```

请选择你的操作[0-24]:22
请输入您要查看的多项式的编号:
5
您要查看的多项式如下:
11x^9-4x^8-4x^4-3x^3-x+2
请按任意键继续. . .
    
```

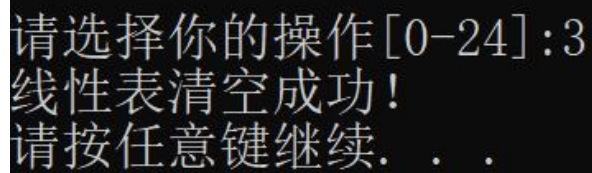
9.2 号功能

输入： 2

0

功能：清空表后退出

实际输出：



```
请选择你的操作[0-24]:3
线性表清空成功!
请按任意键继续. . .
```

1.3.5 异常用例测试

前期操作：表 1 中存储为 HUST

异常用例测试：

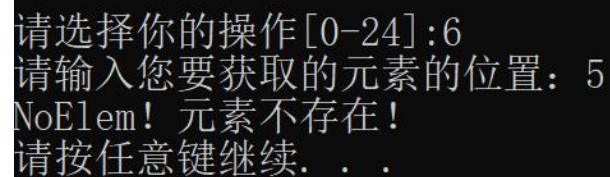
1. 获取的元素不存在

输入：

6

5

输出：



```
请选择你的操作[0-24]:6
请输入您要获取的元素的位置: 5
NoElem! 元素不存在!
请按任意键继续. . .
```

图 1-3 异常用例 1 输出结果

2. 查找的元素不存在

输入：

7

0

输出：

```
请选择你的操作[0-24]:7
请输入您要查找的元素: o
NoElem! 不存在该元素!
请按任意键继续. . .
```

图 1-4 异常用例 2 输出结果

3.前驱不存在

当前元素为 H（第一个元素）

输入:

8

输出:

```
请选择你的操作[0-24]:8
NoElem! 当前已经是第一个元素，不存在前驱!
请按任意键继续. . .
```

图 1-5 异常用例 3 输出结果

4.后继不存在

当前元素为 T（最后一个元素）

输入:

9

输出:

```
请选择你的操作[0-24]:9
NoElem! 当前已经是最后一个元素，不存在后继!
请按任意键继续. . .
```

图 1-6 异常用例 4 输出结果

1.4 实验小结

在实验过程中充分体会到了线性表的这一物理结构，也在写算法的过程中亲身体会了使用链表与数组的区别，两者本身各有优势也各有劣势，基于链表的线性表在插入与删除两方面更具有优势。

虽然只是使用了单链表，没有使用循环链表、双向链表，但是也在编写过程中考虑了这几者的使用方法，掌握了线性表的逻辑结构和物理结构的关系。

编写过程中思路较为清晰，编写也比较顺利，但在指针传参方面有一些小困难，比如说 双指针的使用，和取指针的地址等，加深了对线性表的概念、基本运算的理解。

在扩展功能中对一元多项式加减法的编写过程中，对编程的严谨性有了更深的认识，在多项式输出时，对格式的要求需要进行诸多特判，比如指数为 1，系数为 1 等。也进一步加深了对函数中，参数传递&与*的区别。

总而言之，虽然实验难度不大，但是在编写代码的过程中对细心的要求比较高，最重要的是学习与掌握程序框架的构建和通过键盘输入参数的方法，掌握了将各个基本运算功能模块组织在一个可执行系统中的方法。我也感受到，在调试代码的过程中对耐心的要求比较高，最终独立完成了一个较为完善的带菜单功能的系统也让我颇有成就感。

2 基于二叉链表的二叉树实现

2.1 问题描述

实验环境为 Win10 系统下的 Dev c++ 软件，采用二叉链表的物理结构，构造一个具有菜单功能的演示系统，实现多二叉树管理，包括初始化、创建、销毁、清空、判定空二叉树、求二叉树深度等 20 种基本功能，同时在此基础上进行功能扩展，实现哈夫曼树的编码与译码。输入形式根据菜单的提示进行相关功能的代号输入与相关的参数输入，输出为对应功能函数执行结果与一些恰当的操作提示。

2.1.1 二叉树的基本概念

二叉树是另一种树形结构，其特点是每个结点至多只有两颗子树（即二叉树中不存在度大于 2 的结点），并且，二叉树的子树有左右之分，其次序不能任意颠倒。

与树相似，二叉树也以递归的形式定义。二叉树是 $n (n \geq 0)$ 个结点的有限集合：

①或者为空二叉树，及 $n=0$ 。

②或者由一个根结点和两个互不相交的称为根的左子树和右子树组成。左子树和右子树又分别是一棵二叉树。

2.1.2 二叉树的逻辑结构与基本运算

抽象数据类型二叉树的定义如下：

ADT BiTree {

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 $D=\Phi$ ，则 $R=\Phi$ ，称 BinaryTree 为空二叉树；

若 $D \neq \Phi$, 则 $R = \{H\}$, H 是如下二元关系:

- (1) 在 D 中存在唯一的成为根的数据元素 $root$, 它在关系 H 中无前驱;
- (2) 若 $D - \{root\} \neq \Phi$, 则存在 $D - \{root\} = \{D_1, D_r\}$, 且 $D_1 \cap D_r = \Phi$;
- (3) 若 $D_1 \neq \Phi$, 则 D_1 中存在唯一的元素 X_1 , $\langle root, X_1 \rangle \in H$, 且存在 D_1 上的关系 H_1 包含于 H ; 若 $D_r \neq \Phi$, 则 D_r 中存在唯一的元素 X_r , $\langle root, X_r \rangle \in H$, 且存在 D_r 上的关系属于 H ;
- (4) $(D, \{H_1\})$ 是一棵符合本定义的二叉树, 称为根的左子树, $(D_r, \{H_r\})$ 是一棵符合本定义的二叉树, 称为根的右子树。

基本操作:

InitBiTree(T)

操作名称: 初始化二叉树

初始条件: 二叉树 T 不存在

操作结果: 构造空二叉树 T

DestroyBiTree(T)

操作名称: 销毁二叉树

初始条件: 二叉树 T 已存在

操作结果: 销毁二叉树 T

CreateBiTree(T, definition)

操作名称: 创建二叉树

初始条件: **definition** 给出二叉树 T 的定义

操作结果: 按 **definition** 构造二叉树 T

ClearBiTree(T)

操作名称：清空二叉树

初始条件：二叉树 T 存在

操作结果：将二叉树 T 清空

BiTreeEmpty(T)

操作名称：判定空二叉树

初始条件：二叉树 T 存在

操作结果：若 T 为空二叉树则返回 **TRUE**，否则返回 **FALSE**

BiTreeDepth(T)

操作名称：求二叉树深度

初始条件：二叉树 T 存在

操作结果：返回 T 的深度

Root(T)

操作名称：求二叉树的根

初始条件：二叉树 T 已存在

操作结果：返回 T 的根

Value(T, e)

操作名称：获得节点

初始条件：二叉树 T 已存在， e 是 T 中的某个结点

操作结果：返回 e 的值

Assign($T, \&e, value$)

操作名称：结点赋值

初始条件：二叉树 T 已存在， e 是 T 中的某个结点

操作结果：结点 e 赋值为 $value$

Parent(T, e)

操作名称：获得双亲

初始条件：二叉树 T 已存在， e 是 T 中的某个结点

操作结果：若 e 是 T 的非根结点，则返回它的双亲结点指针，否则返回 **NULL**

LeftChild(T, e)

操作名称：获得左孩子

初始条件：二叉树 T 存在， e 是 T 中某个节点

操作结果：返回 e 的左孩子结点指针若 e 无左孩子，则返回 **NULL**

RightChild(T, e)

操作名称：获得右孩子

初始条件：二叉树 T 已存在， e 是 T 中某个结点

操作结果：返回 e 的右孩子结点指针若 e 无右孩子，则返回 **NULL**

LeftSibling(T, e)

操作名称：获得左兄弟

初始条件：二叉树 T 存在， e 是 T 中某个结点

操作结果：返回 e 的左兄弟结点指针，若 e 是 T 的左孩子或者无左兄弟，则返回 **NULL**

RightSibling(T, e)

操作名称：获得右兄弟

初始条件：二叉树 T 存在， e 是 T 中某个结点

操作结果：返回 e 的右兄弟结点指针，若 e 是 T 的右孩子或者无右兄弟，则返回 `NULL`

`InsertChild(T,p,LR,c)`

操作名称：插入子树

初始条件：二叉树 T 存在， p 指向 T 中的某个结点， LR 为 0 或 1，非空二叉树 c 与 T 不相交且右子树为空

操作结果：根据 LR 为 0 或者 1，插入 c 为 T 中 p 所指结点的左或右子树， p 所指结点的原有左子树或右子树则为 c 的右子树

`DeleteChild(T,p,LR)`

操作名称：删除子树

初始条件：二叉树 T 存在， p 指向 T 中的某个结点， LR 为 0 或 1

操作结果：根据 LR 为 0 或者 1，删除 c 为 T 中 p 所指结点的左或右子树

`PreOrderTraverse(T,Visit())`

操作名称：先序遍历

初始条件：二叉树 T 存在，`Visit` 是对结点操作的应用函数

操作结果：先序遍历 t ，对每个结点调用函数 `Visit` 一次且一次，一旦调用失败，则操作失败

`InOrderTraverse(T,Visit)`

操作名称：中序遍历

初始条件：二叉树 T 存在，`Visit` 是对结点操作的应用函数

操作结果：中序遍历 t ，对每个结点调用函数 **Visit** 一次且一次，一旦调用失败，则操作失败

PostOrderTraverse(T,Visit)

操作名称：后序遍历

初始条件：二叉树 T 存在，**Visit** 是对结点操作的应用函数

操作结果：后序遍历 t ，对每个结点调用函数 **Visit** 一次且一次，一旦调用失败，则操作失败

LevelOrderTraverse(T,Visit)

操作名称：层序遍历

初始条件：二叉树 T 存在，**Visit** 是对结点操作的应用函数

操作结果：层序遍历 t ，对每个结点调用函数 **Visit** 一次且一次，一旦调用失败，则操作失败

}ADT BiTree

2.2 系统设计

2.2.1 数据类型及数据物理结构

数据类型：

```
typedef char ElemType; //节点数据类型
typedef int Status;
```

数据物理结构：

```
typedef struct BNode{ //基于二叉链表的二叉树
    ElemType data; //数据域
    int weight; //哈夫曼树的节点权值（扩展后）
    struct BNode* lchild; //指向左孩子
    struct BNode* rchild; //指向兄弟
}BNode;
typedef BNode* BTree; //指针类型
```

2.2.2 含菜单的演示系统的设计思想

将菜单写入 menu()函数中，菜单给予用户代码与功能的对应关系，便于用户输入代码进行功能选择。

功能选择使用一个 while 循环，其内使用 switch-case 结构对用户输入的代码进行相应功能函数实现，每一次功能实现后均使用 system("cls")进行清屏和 system("pause")实现菜单返回功能，直到用户输入 '0' 退出系统。

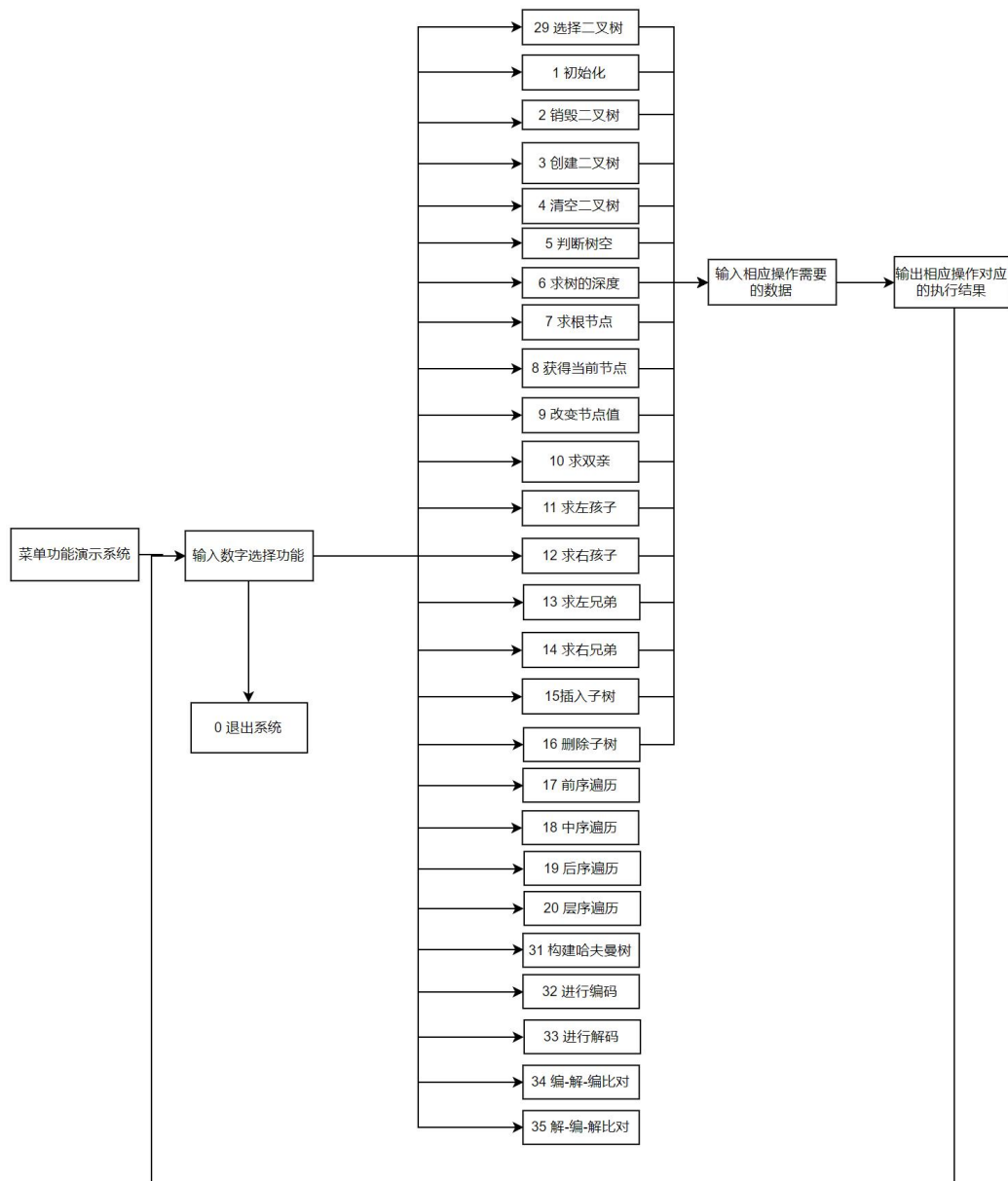


图 2-1 菜单演示系统流程图

2.2.3 重要函数的设计思想

1.求树的深度 BiTreeDepth

递归实现，为了求得树的深度，可以先求左右子树的深度，取二者较大者加 1 即是 树的深度，递归返回的条件是若节点为空，返回 0。

2.求左兄弟右兄弟 LeftSibling&RightSibling

队列实现，以寻找左兄弟为例，先把根节点加入到队列中，使用 while

循环，只要队列不为空，取队头元素指向的右孩子如果与当前节点相同，则把队头元素的左孩子返回，即为当前节点的左兄弟，如果为空，则说明没有左兄弟，或者如果队头元素的左孩子为当前节点，则对二叉树而言，必然没有左兄弟。如果前面条件均不成立，则把队头的左孩子右孩子加入队列，队头出队。

寻找右兄弟的思想类似。

3.插入子树 ChildInsert

以插入左子树为例，如果左子树本身存在，则将其放入要插入子树的右子树上，然后将待插入子树插入到左子树，如果左子树为空，直接插入，最后把子树在列表中删除。

插入右子树类似。

4.前序遍历（递归）PreOrderTraverse

递归实现，若二叉树为空,则返回空操作。否则先访问根节点,然后前序遍历左子树，再前序遍历右子树。

5.中序遍历（非递归）InOrderTraverse

从根结点开始，遍历左孩子同时压栈，当遍历结束，说明当前遍历的结点没有左孩子，从栈中取出来调用操作函数，然后访问该结点的右孩子，继续以上重复性的操作。

6.后序遍历 PostOrderTraverse

递归实现，若二叉树为空，则返回空操作，否则后序遍历访问左子树，然后后序遍历访问右子树，再访问根节点。

7.层序遍历 LevelOrderTraverse

初始时，根结点入队列。然后，while 循环判断队列不空时，弹出一个结点，访问它，并把它的所有孩子结点入队列。

8.创建哈夫曼树 CreateHTree

递归实现，如果当前字符为'^'，则当前指针为 NULL，否则，动态开辟新空间，将字符和权值加入到当前节点中，然后递归构建左子树，构建右子树。

9.编码 HTreeCode

先用一个二维字符数组将每个叶子节点的编码存起来，把输入进来的字符串从第一个字符开始，依次向后，直接取二维字符数组中对应的编码序列，加到 ans 字符串中去，最后，ans 即为编码后的结果。

10.解码 HTreeDecode

根据 code 字符串，是 0 找右孩子，是 1 找左孩子，直到找到一个叶子节点，把叶子节点的字符加到 ans 字符串中去，然后再从根节点开始根据 code 下一个字符找。重复上述操作，直到 code 到结尾。

11.解码与译码比对 CodeCheck&StringCheck

先将输入进来的 code 或 string 用另一个字符串 temp 存下来，然后进行对应的解码（编码）--编码（解码）操作得到 ans，将该结果与 temp 字符串比较，相同即为结果相同，不同则结果不同。

2.3 系统实现

2.3.1 软硬件环境

本此次实验在 win10 专业版下，采用 Dev c++ 编程软件进行编写与编译运行。

2.3.2 头文件及预定义常量说明

1.头文件

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<iostream>
#include<string.h>
#include<windows.h>
using namespace std;
```

2.预定义常量

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

2.3.3 系统菜单演示

初始界面为菜单界面，并提示用户进行操作，在每次一个操作执行完毕后均会回到菜单界面。

```

C:\Users\86199\Desktop\信安1901-U201911658-李欣宇-2.exe
*****
*      Menu for Function of Binary Tree      *
*****

-----
*****
* 0.EXIT          * * 1.IntiaBiTree          *
*****
* 2.DestroyBiTree * * 3.CreaeBiTree          *
*****
* 4.ClearBiTree   * * 5.BiTreeEmpty          *
*****
* 6.BiTreeDepth   * * 7.Root                  *
*****
* 8.Value          * * 9.Assign                *
*****
* 10.Parent        * * 11.LeftChild            *
*****
* 12.RightChild    * * 13.LeftSibling           *
*****
* 14.RightSibling  * * 15.InsertChild           *
*****
* 16.DeleteChild   * * 17.PreOrderTraverse     *
*****
* 18.InOrderTraverse * * 19.PostOrderTraverse  *
*****
* 20.LevelOrderTraverse * * 29.SetBTnum        *
*****
* 31.CreateHTree   * * 32.HTreeCode            *
*****
* 33.HTreeDecode   * * 34.StringCheck          *
*****
* 35.CodeCheck     * * *****              *
*****
-----
请选择你的操作[0-35]:

```

图 2-2 菜单演示系统初始界面

2.3.4 正常用例测试

基本功能测试:

1. 29 号功能

输入: 29

1

功能: 设置二叉树序号为 1

理论结果: Tnum 为 1

实际输出:

```

请选择你的操作[0-35]:29
请输入您要设置的二叉树的编号: 1
设置成功!
请按任意键继续. . .

```

2. 1 号功能

输入： 1

功能：初始化

理论结果：初始化二叉树

实际输出：

```
请选择你的操作[0-35]:1
二叉树初始化成功!
请按任意键继续. . .
```

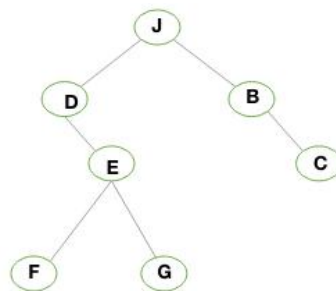
3. 3 号功能

输入： 3

JD^EF^^G^^B^C^^

功能：创建二叉树

理论结果：



实际输出：

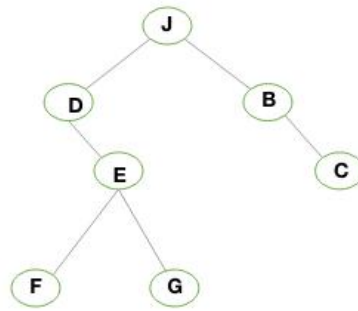
```
请选择你的操作[0-35]:3
请输入您要创建的二叉树的先序遍历序列:
JD^EF^^G^^B^C^^
二叉树创建成功!
请按任意键继续. . .
```

4. 17 号功能

输入： 17

功能：前序遍历

理论结果：前序遍历为 JDEFGBC



实际输出:

```

请选择你的操作[0-35]:17
前序遍历为: JDEFGBC
请按任意键继续. . .
    
```

5. 5 号功能

输入: 5

功能: 判断树空

理论结果: 不为空

实际输出:

```

请选择你的操作[0-35]:5
该树不为空
请按任意键继续. . .
    
```

6. 6 号功能

输入: 6

功能: 求树的深度

理论结果: 树的深度为 4

实际输出:

```

请选择你的操作[0-35]:6
树的深度为4
请按任意键继续. . .
    
```

7. 7 号功能

输入: 7

功能: 求树的根

理论结果: 根为 J

实际输出：

```
请选择你的操作[0-35]:7
根节点的值为J
请按任意键继续. . .
```

8. 8 号功能

输入： 8

功能：求当前节点

理论结果：当前节点为J

实际输出：

```
请选择你的操作[0-35]:8
当前节点的值为J
请按任意键继续. . .
```

9. 9 号功能

输入： 9

A

功能：改变当前节点的值

理论结果：前节点修改值为 A，即根结点修改为 A

实际输出：

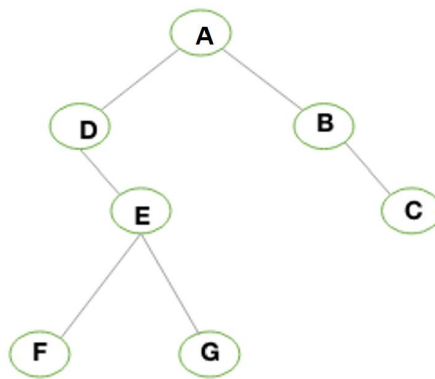
```
请选择你的操作[0-35]:9
请输入您要替换当前节点的值：A
当前节点的值替换成功！
请按任意键继续. . .
```

10. 20 号功能

输入： 20

功能：层序遍历

理论结果：层序遍历为 ADBECFG



实际输出:

```

-----
请选择你的操作[0-35]:20
层序遍历为: ADBECFG
请按任意键继续. . .
    
```

11. 11 号功能

输入: 11

功能: 找左孩子

理论结果: 求根结点的左孩子为 D

实际输出:

```

-----
请选择你的操作[0-35]:11
当前节点的左孩子是D
请按任意键继续. . .
    
```

12. 12 号功能

输入: 12

12

功能: 两次求右孩子

理论结果: 第一次为 E, 第二次为 G

实际输出:


```
请选择你的操作[0-35]:12
当前节点的右孩子是E
请按任意键继续. . .
```

```
-----
请选择你的操作[0-35]:12
当前节点的右孩子是G
请按任意键继续. . .
```

13. 13 号功能

输入: 13

功能: 求左兄弟

理论结果: G 的左兄弟为 F

实际输出:

```
-----
请选择你的操作[0-35]:13
当前节点的左兄弟是F
请按任意键继续. . .
```

14. 14 号功能

输入: 14

功能: 求右兄弟

理论结果: F 的右兄弟为 G

实际输出:

```
请选择你的操作[0-35]:14
当前节点的右兄弟是G
请按任意键继续. . .
```

```
-----
请选择你的操作[0-35]:13
当前节点的左兄弟是F
请按任意键继续. . .
```

15. 10 号功能

输入: 10

功能: 求父节点

理论结果：G 的父节点为 E

实际输出：

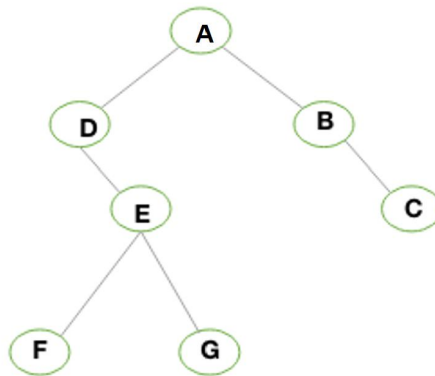
```
-----
请选择你的操作[0-35]:10
当前节点的父节点是E
请按任意键继续. . .
```

16. 17 号功能

输入： 17

功能：前序遍历

理论结果：先序遍历为 ADEFGBC



实际输出：

```
请选择你的操作[0-35]:17
前序遍历为：ADEFGBC
请按任意键继续. . .
```

17. 18 号功能

输入： 18

功能：中序遍历

理论结果：中序遍历为 DFEGABC

实际输出：

```
请选择你的操作[0-35]:18
中序遍历为：DFEGABC
请按任意键继续. . .
```

18. 19 号功能

输入： 19

功能：后序遍历

理论结果：后序遍历为 FGEDCBA

实际输出：

```

-----
请选择你的操作[0-35]:19
后序遍历为: FGEDCBA
请按任意键继续. . .
    
```

19. 29 & 1 & 3 号功能

输入： 29

2

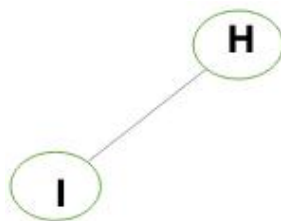
1

3

HI^^^

功能：创建第二棵二叉树

理论结果：



实际输出：

```

-----
请选择你的操作[0-35]:3
请输入您要创建的二叉树的先序遍历序列:
HI^^^
二叉树创建成功!
请按任意键继续. . .
    
```

20. 15 号功能

输入： 29

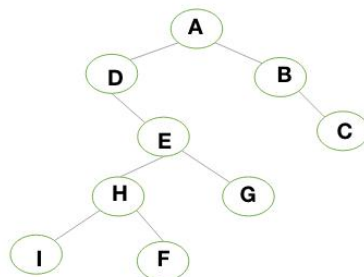
1

15

L 2

功能：插入子树

理论结果：



实际输出：

```

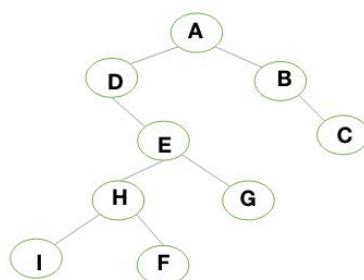
请选择你的操作[0-35]:15
L 2
子树插入成功!
请按任意键继续. . .
    
```

21. 20 号功能

输入： 20

功能：层序遍历

理论结果：层序遍历为 ADBECHGIF



实际输出：

```

请选择你的操作[0-35]:20
层序遍历为: ADBECHGIF
请按任意键继续. . .
    
```

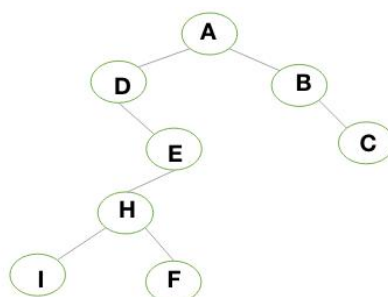
22. 16 号功能

输入： 16

R

功能：删除子树

理论结果：删除当前节点 E 的右子树



实际输出：

```

请选择你的操作[0-35]:16
R
子树删除成功!
请按任意键继续. . .
    
```

23.2 号功能

输入： 2

0

功能：销毁二叉树并退出

实际输出：

```

请选择你的操作[0-35]:2
二叉树销毁成功!
请按任意键继续. . .
    
```

扩展功能测试：

1.31 号功能

输入： 29

1

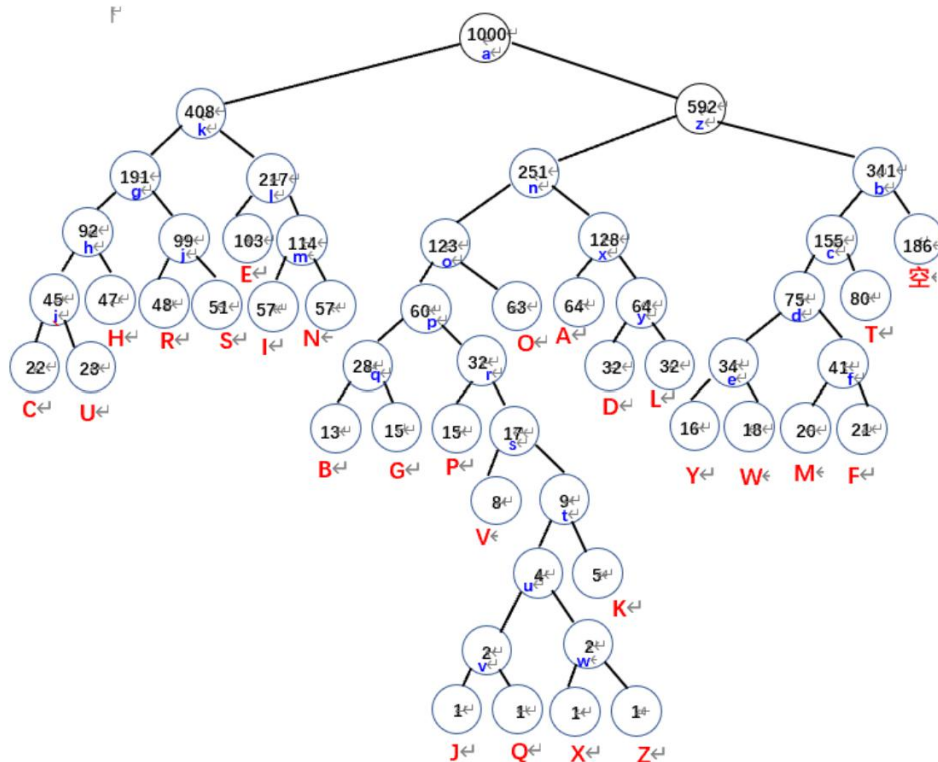
1

31

akghiC^^U^^H^^jR^^S^^IE^^mI^^N^^znopqB^^G^^rP^^sV^^tuvJ^^Q^^
^wX^ ^Z^^K^^O^^xA^^yD^^L^^bcdeY^^W^^fM^^F^^T^^ ^^

功能：按照先序遍历构建哈夫曼树

理想结果：创建哈夫曼树



实际输出：

```
请选择你的操作[0-35]:31
请输入您要创建的哈夫曼树的序列:
akghiC^^U^^H^^jR^^S^^IE^^mI^^N^^znopqB^^G^^rP^^sV^^tuvJ^^Q^^wX^^Z^^K^^O
^^xA^^yD^^L^^bcdeY^^W^^fM^^F^^T^^
哈夫曼树创建成功!
请按任意键继续. . .
```

2.20 号功能

输入：20

功能：层序遍历

实际输出：

```
请选择你的操作[0-35]:20
层序遍历为: akzglnbhjEmoxc iHRSINpOAYdTCUqrDLefBGPsYWMFVtuKvwJ
请按任意键继续. . .
```

3.32 号功能

输入： 32

HI IT

功能：进行编码

理想结果：编码为 0001011011101101101

实际输出：

```
请选择你的操作[0-35]:32
请输入您要进行编码的字符串 :
HI IT
编码的结果为0001011011101101101
请按任意键继续. . .
```

4.33 号功能

输入： 33

功能：进行解码

理想结果：解码结果为： THIS PROGRAM IS MY FAVORITE

实际输出：

```
请选择你的操作[0-35]:33
请输入您要进行解码的字符串 :
1101000101100011111100010001010011000010010101011001011101100011111
110010110000111110011101010001101001001001101101010
解码结果为 : THIS PROGRAM IS MY FAVORITE
请按任意键继续. . .
```

5.34 号功能

输入： 34

功能：编-解-编匹配

理想结果：结果匹配

实际输出：

```
请选择你的操作[0-35]:34
THIS PROGRAM IS MY FAVORITE
编码-解码-编码结果匹配
请按任意键继续. . .
```

6.35 号功能

输入： 35

功能：解-编-解匹配

理想结果：结果匹配

实际输出：

```
请选择你的操作[0-35]:35
110100010110001111110001000101001100001001010101100101110110001111111001011000011
1110011101010001101001001001101101010
解码-编码-解码结果匹配
请按任意键继续. . .
```

2.3.5 异常用例测试

异常用例测试：

当前二叉树的序号为 1，创建的二叉树前序遍历为 $HI^{^^^}$ ，当前节点为根节点

1.双亲不存在

输入：

10

输出：

```
请选择你的操作[0-35]:10
当前节点已经是根节点，没有双亲！
请按任意键继续. . .
```

图 2-2 异常用例 1 测试结果

2.右孩子不存在

输入：

12

输出：

```
请选择你的操作[0-35]:12
当前节点没有右孩子
请按任意键继续. . .
```

图 2-3 异常用例 2 测试结果

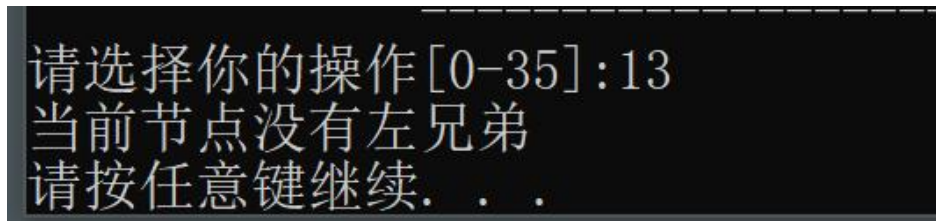
3.左兄弟不存在

输入：

11（切换当前节点为 I）

13

输出：



```
请选择你的操作[0-35]:13
当前节点没有左兄弟
请按任意键继续. . .
```

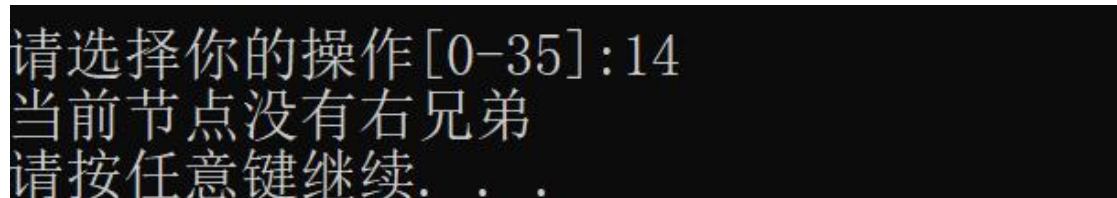
图 2-3 异常用例 3 测试结果

4.右兄弟不存在

输入：

14

输出：



```
请选择你的操作[0-35]:14
当前节点没有右兄弟
请按任意键继续. . .
```

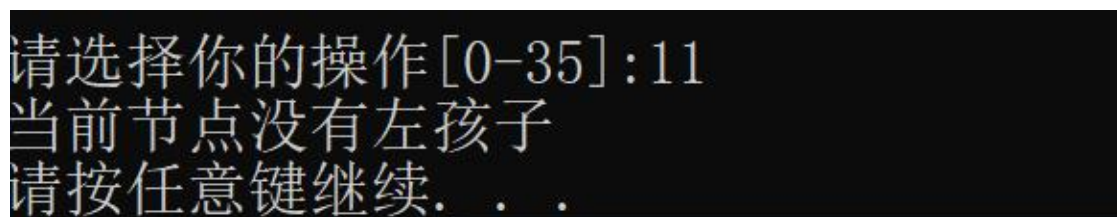
图 2-4 异常用例 4 测试结果

5.左孩子不存在

输入：

11

输出：



```
请选择你的操作[0-35]:11
当前节点没有左孩子
请按任意键继续. . .
```

图 2-5 异常用例 5 测试结果

2.4 实验小结

通过本次实验加深了对二叉树的概念、对二叉树的基本运算的理解，熟练掌握了二叉树的逻辑结构与物理结构的关系，并通过编程掌握了二叉树基本运算的实现方法，尤其是建树和遍历。

在基本功能实现中，通过树的遍历和找左右兄弟，加深了递归的使用与栈和队列的使用。

在扩展功能中，了解了哈夫曼树的构成和生成方法，如何译码、编码。也感知到模块程序设计在大程序设计使用中的普遍性，这次实验就是最好的证明，通过模块程序设计，使程序的可读可写性明显增强。

这次实验比第一次实验难度加大了许多，但也正因为难度的提升让我学到了更多东西，虽然还没有进行实际的应用，只是编写了基本的操作，但是我也在查询资料过程中感受到树这一数据结构的强大和美妙之处，相信这次实验也会帮助我在将使用这一数据结构时更加熟练。

参考文献

- [1] Robert L K, Alexander J R. 数据结构与程序设计——C++语言描述, 高等教育出版社, 2001
- [2] 严蔚敏.数据结构 C 语言版.清华大学出版社, 2007
- [3] 殷人昆. 数据结构（用面向对象方法与 C++描述）, 清华大学出版社, 1999
- [4] 王广芳. 数据结构算法与应用-C++语言描述, 机械工业出版社, 2006 [5] 管纪文, 刘大有. 数据结构, 高等教育出版社, 1985

附录 A：基于链式存储结构的线性表实现源代码

```
/*
作者：李欣宇
班级：信安 1901 班
时间：2020 年 10 月 14 日
*/

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<iostream>

#include<windows.h>

using namespace std;

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

typedef char ElemType; //元素类型
typedef int Status;
typedef double ElemType_D; //多项式元素类型

typedef struct LNode{
    ElemType_D coe; //系数
    int expo; //指数
    ElemType data; //基本数据
    struct LNode* next;
```

```

}LNode;    //节点结构体

struct ARRAY_S{
    double co; //系数
    int exp; //指数
}sz[100];    //多项式结构体数组
typedef LNode* LinkList; //节点指针类型
int Length;    //长度
int num;    //全局变量，用于指示当前操作的线性表的标号
LinkList
array[11]={NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL};// 多表
管理数组，存储表头指针

void menu(void);    //菜单函数
void exp_print(int exp); //多项式输出中指数格式调整函数
void coe_print(ElemType_D coe,int flag); //多项式输出中系数格式调整函数
void setListnum(int n); //19 设置表编码
void visit (ElemType x); //visit 遍历函数
double abs_D(double x); //浮点数的 abs 函数
Status Compare (ElemType a,ElemType b); //compare 函数
Status InitList(LinkList L); //1 初始化表
void DestroyList(LinkList *L); //2 销毁表
Status ClearList(LinkList L); //3 清空表
Status ListEmpty(LinkList L); //4 判断空表
int ListLength(LinkList L); //5 求表长
Status GetElem(LinkList L,int i, ElemType *e); //6 获取元素
Status LocateElem(LinkList L, ElemType e, Status(Compare)()); //7 查找元素
Status PriorElem(LinkList L, ElemType cur_e, ElemType *pre_e); //8 获得前
驱
Status NextElem(LinkList L, ElemType cur_e, ElemType *next_e); //9 获得后

```

继

```
Status ListInsert(LinkList L, int i, ElemType e);    //10 插入元素
Status ListDelete(LinkList L, int i, ElemType *e);    //11 删除元素
Status ListTraverse(LinkList L, void(visit)());    //12 遍历
Status CreatList(int n);    //21 创建多项式线性表
Status ShowList(int num);    //22 输出多项式
Status AddList(int num1, int num2, int num3);    //23 多项式加法
Status MinusList(int num1, int num2, int num3);    //24 多项式减法
```

```
void exp_print(int exp ){    //输出多项式中的 x^
    if (exp==1){
        printf("x");
    }
    else if (exp!=0) cout<<"x^"<<exp;
}
```

void coe_print(ElemType_D coe,int flag){ //输出多项式中的系数，主要调整前面的正负与特判系数为 1 和-1 的情况

```
int flag2 = 0; //flag2 指示 1 为 - 0 为+
if (flag){    //flag=1 为第一个系数不为 0 的元素
    if (coe<0) flag2= 1;
    coe = abs_D(coe);
    if (flag2) cout<<"-";
    if (coe !=1 ) cout<<coe;

}
else {
    if (coe <0) flag2 =1;
    coe =abs_D(coe);
    if (flag2) cout << "-"; else cout << "+";
    if (coe !=1) cout <<coe;
}
```

```

    }
}

int cmp(const void *a, const void *b){    //sort 函数的 cmp
    return (*(ARRAY_S *)a).exp > (*(ARRAY_S *)b).exp ? 1 : -1;
}

double abs_D(double x){    //double 类型的 abs 函数
    if (x>=0) return x; else return (-x);
}

void setListnum(int n){    //设置表号
    num = n;    //num 为全局变量
}

void visit (ElemType x){    //visit 函数
    printf("%c",x);    //访问输出
}

Status Compare (ElemType a,ElemType b){    //compare 函数，用于查找比较
    if (a==b) return OK; else return ERROR;
}

Status InitList(LinkList L){    //初始化表
    (L) = (LinkList)malloc(sizeof(LNode));    //动态开辟空间
    if(!(L)) return OVERFLOW;    //判断溢出
    (L)->next = NULL;    //指示 NULL
    array[num] = L;    //存入多表管理数组中
    return OK;
}

void DestroyList(LinkList L){    //销毁表
    LinkList p = L;    //表头
    while (p){    //只要不为空
        p = (L)->next;    //指向下一个节点
    }
}

```

```

        free(L); //释放空间
        (L) = p;    // 指向下一个 (p)
    }
}

Status ClearList(LinkList L){ //清空表
    LinkList ptemp, p;
    if(!L) return ERROR; //表已经为空则返回 ERROR
    ptemp = L->next; //指向第一个节点
    while(ptemp){ //只要不为 null
        p = ptemp->next; //指向下一个节点
        free(ptemp); //释放空间
        ptemp = p;
    }
    L->next = NULL; //表头的 next 赋值为 NULL
    return OK;
}

Status ListEmpty(LinkList L){ //判断表空
    if(L != NULL && L->next != NULL)
        return FALSE; //不为空返回 FALSE
    else return TRUE; //为空返回 TRUE
}

int ListLength(LinkList L){ //求表长
    LinkList p;
    int i = 0; //使用 i 计数
    if (L){
        i = 0; //赋初值 0
        p = L->next; //指向第一个节点
    }
}

```



```

        while (p){ //直到 p 为空，到表尾
            i++; //长度加 1
            p = p->next;
        }
    }
    return i; //返回表长
}

```

```

Status GetElem(LinkList L,int i, ElemType *e){ //获得元素
    int j = 1; //使用 j 计数
    LinkList p = L->next; //指向第一个元素
    if (i>ListLength(L)) return ERROR; //判断 i 是否合法
    while (p && j<i){ //循环条件为 p 不为空且 j<i
        j++;
        p = p->next;
    }
    *e = p->data; //把获得的元素用 e 返回
    return OK;
}

```

```

int LocateElem(LinkList L, ElemType e,
Status(Compare)(ElemType,ElemType)){ //查找元素
    int i;
    int t;
    LinkList p;
    if (L){ //表不为空
        i = 0; //变量初值
        p = L->next; //指向第一个节点
        while (p){ //只要当前节点不为 null

```

```

        i++;    //位置加 1

        if (!Compare(e,p->data)) p = p->next; else return i;    //查找到返回
位置 i ， 当前 不是要查找的元素则指向下一个节点
    }

}

else return ERROR; //到达表尾且没有查找到则返回 ERROR

if(!p) return ERROR;

}

Status PriorElem(LinkList L, ElemType cur_e, ElemType *pre_e){//获得前驱

    LinkList p, temp;

    p = L->next;    //第一个元素

    if (p->data != cur_e){    //当前元素不是的第一个元素 ， 如果是第一个元
素不执行 if， 直接返回 ERROR

        while (p->next){

            temp = p->next; //使用 temp 获取 下一个元素

            if (temp->data == cur_e){    //下一个元素是 cur_e 则当前元素即为
cur_e 的前驱

                *pre_e = p->data;    //使用 pre_e 返回前驱

                return OK;

            }

            p = temp;

        }

    }

    return ERROR;    // 第一个元素的时候返回 ERROR

}

Status NextElem(LinkList L, ElemType cur_e, ElemType *next_e){    //获得后继

    LinkList p, temp;

    if (L){

        p = L->next;

```

```

        while (p && p->next){
            temp = p->next; //使用 temp 获得下一个元素
            if (p->data == cur_e){ // 当前元素是 cur_e 则下一个元素即为
cur_e 的后继
                *next_e = temp->data; //使用 next_e 返回后继
                return OK;
            }
            p = temp;
        }
    }
    return ERROR; //最后一个元素的时候则返回 ERROR
}

```

Status ListInsert(LinkList L, int i, ElemType e){ //插入元素

```

    LinkList p = L, temp;
    int j = 0; //使用 j 找 i 的前一个位置
    while (p && j < i-1){ //查找到 i 的前一个位置
        p = p->next;
        j++;
    }

    temp = (LinkList)malloc(sizeof(LNode)); //开辟空间
    if (!temp) return OVERFLOW; //防止溢出
    temp->data = e; //赋值
    temp->next = p->next; //链接进表
    p->next = temp; //链接进表
    return OK;
}

```

Status ListDelete(LinkList L, int i, ElemType *e){ //删除元素

```

    LinkList temp, pre = L;

```

```

int j = 1;  //j 找 i 的前一个位置
while (pre->next && j<i){
    pre = pre ->next;
    j++;
}
if (!pre->next || j>i) return ERROR;
temp = pre->next;  //重新链接表
pre->next = temp->next;  //重新链接表
*e = temp->data;  //使用 e 返回删除的元素
free(temp);  //释放空间
return OK;
}

Status ListTraverse(LinkList L, void(visit)(ElemType)){  //遍历表
    LinkList p;
    if (!L) return ERROR;  //表为空则返回 ERROR
    else p = L->next;  //指向第一个元素
    while (p){  //只要没到表尾
        visit(p->data);  //输出元素
        p = p->next;  //指向下一个
    }
    printf("\n");
    return OK;
}

Status CreatList(int n,int numm){  //创建多项式表

    LinkList p = array[numm], temp;
    int i=0,j=0;

    qsort(sz,100,sizeof(sz[0]),cmp);//先以指数为关键字进行升序排序

```

```

while(i<n){
    temp = (LinkList)malloc(sizeof(LNode)); //开辟空间
    if (!temp) return OVERFLOW; //防止溢出
    while(sz[i].exp==sz[i+1].exp){ //进行合并同类项
        sz[i+1].co+=sz[i].co;
        i++;
    }
    temp->coe = sz[i].co;//插入表中系数
    temp->expo = sz[i].exp;//插入表中指数
    temp->next = p->next; // 头插法插入
    p->next = temp;
    i++;
}

return OK;
}

Status ShowList(int num){ //输出多项式
    LinkList p=array[num];
    printf("您要查看的多项式如下： \n");
    int flag_1=1, flag_2 = 0;
    if (!p->next){ //运算结果为 0 的输出
        printf("0\n");
        return OK;
    }
    p=p->next;
    while((p->coe ==0)&& p->next){ //如果系数为 0，则该项直接跳过
        p=p->next;
    }
    if (!p->next && p->coe ==0){ //如果第一项系数为 0 且没有后项，输出 0

```

退出

```
        printf("0\n");
        return OK;
    }
    coe_print(p->coe,1); //进入 coe_print 输出函数,进行系数输出,此时 flag
```

为 1, 指示这是第一项的输出

```
    exp_print(p->expo); //进入 exp_print 输出函数, 进行指数输出
    p=p->next; //下一项
    while (p){
```

```
        while ((p->coe==0)&& p->next){
            p=p->next; //同样判断系数是否为 0
        }
```

```
        if (!p->next && p->coe ==0) break; //同样判断是否系数为 0 且没有
```

后项

```
        coe_print(p->coe,0); //输出系数, flag=0, 指示这不是第一项输出
        exp_print(p->expo); //输出指数
        p=p->next; //下一项
    }
    printf("\n");
    return OK;
}
```

Status ListInsert_2(LinkList L,int i,ElemType_D co, int exp){ //进行指数和系数的插入算法, 用于加减法运算中的结果插入

```
    LinkList p=L,temp;
    int j = 0;
    while (p && j < i-1){
        p = p->next;
        j++;
    }
```

```

temp = (LinkedList)malloc(sizeof(LNode));
if (!temp) return OVERFLOW;
temp->coe = co; //以上均与 ListInsert 函数相同, 不同点在这两个数据域
需要进行赋值
temp->expo = exp;
temp->next = p->next;
p->next = temp;
return OK;
}

Status AddList(int num1, int num2, int num3){ //多项式加法
    LinkedList L1=array[num1]->next,L2=array[num2]->next,L3=array[num3];
    int len_l3=0; //指示结果多项式长度
    ClearList(L3); //清空 L3
    while ((L1) &&(L2)){ //如果 L1 和 L2 都不为空

        if (L1->expo == L2->expo){ //如果当前 L1 和 L2 指示的指数相同, 则
对系数进行加法
            ListInsert_2(array[num3],len_l3+1,L1->coe+L2->coe,L1->expo); //
运算后插入到 L3 中
            len_l3++; //L3 长度加 1
            L1 = L1->next; //L1 和 L2 均指向下一个节点
            L2 = L2->next;
        }
        else if (L1->expo > L2->expo){ //如果当前 L1 指向的大于 L2 指示的指
数, 则把 L1 指示的插入 L3 中, 并 L1 后移
            ListInsert_2(array[num3],len_l3+1,L1->coe,L1->expo);
            len_l3++; //L3 长度加 1
            L1 = L1->next; //L1 指向下一个节点
        }
    }
}

```

else if(L1->expo < L2->expo){//如果当前 L2 指向的大于 L1 指示的指数，
则把 L2 指示的插入 L3 中，并 L2 后移

```
ListInsert_2(array[num3],len_l3+1,L2->coe,L2->expo);
```

```
len_l3++; //L3 长度加 1
```

```
L2= L2->next;
```

```
}
```

```
}
```

```
if (!L1){ //L1 为空
```

```
while (L2){ //把 L2 中剩余的都插入到 L3 中
```

```
ListInsert_2(array[num3],len_l3+1,L2->coe,L2->expo);
```

```
len_l3++;//L3 长度加 1
```

```
L2 = L2->next; //L2 指向下一个节点
```

```
}
```

```
}
```

```
if (!L2){ //L2 为空
```

```
while (L1){ //把 L1 中剩余的都插入到 L3 中
```

```
ListInsert_2(array[num3],len_l3+1,L1->coe,L1->expo);
```

```
len_l3++;
```

```
L1 = L1->next;
```

```
}
```

```
}
```

```
return OK;
```

```
}
```

```
Status MinusList(int num1, int num2, int num3){ //多项式减法
```

```
LinkList L1=array[num1]->next,L2=array[num2]->next,L3=array[num3];
```

```
int len_l3=0;
```

```
ClearList(L3);
```

```
while ((L1) &&(L2)){
```


if (L1->expo == L2->expo){//如果当前 L1 和 L2 指示的指数相同，则对系数进行减法

ListInsert_2(array[num3],len_l3+1,(L1->coe)-(L2->coe),L1->expo);

len_l3++;

L1 = L1->next;

L2 = L2->next;

}

else if (L1->expo > L2->expo){ //如果当前 L1 指向的大于 L2 指示的指数，则把 L1 指示的插入 L3 中，并把 L1 后移

ListInsert_2(array[num3],len_l3+1,L1->coe,L1->expo);

len_l3++;

L1= L1->next;

}

else if(L1->expo < L2->expo){//如果当前 L2 指向的大于 L1 指示的指数，则把 L2 指示系数取反的插入 L3 中，并把 L2 后移

ListInsert_2(array[num3],len_l3+1,-L2->coe,L2->expo);

len_l3++;

L2= L2->next;

}

}

if (!L1){//L1 为空

while (L2){ //把 L2 中剩余的都插入到 L3 中

ListInsert_2(array[num3],len_l3+1,-L2->coe,L2->expo);

len_l3++;

L2 = L2->next;

}

}

```

if (!L2){ //L2 为空
    while (L1){//把 L1 中剩余的都插入到 L3 中
        ListInsert_2(array[num3],len_l3+1,L1->coe,L1->expo);
        len_l3++;
        L1 = L1->next;
    }

}

return OK;
}

void menu(){ //menu 输出
    system("cls");

    printf("*****\n\n");
    printf("*      Menu for Linear Table On Sequence Structure      * \n \n");
    printf("***** \n\n");
    printf("----- \n");
    printf("*****\n");
    printf(" *   0.EXIT           * *   1. InitiaList           * \n");
    printf("*****\n");
    printf(" *   2.DestroyList    * *   3.ClearList           *  \n");
    printf("*****\n");
    printf(" *   4.ListEmpty      * *   5.ListLength           * \n");
    printf("*****\n");
    printf(" *   6.GetElem        * *   7.LocateElem           * \n");
    printf("*****\n");
    printf(" *   8.PriorElem      * *   9.NextElem            * \n");
    printf("*****\n");
    printf(" *  10.ListInsert     * *  11.ListDelete           * \n");
    printf("*****\n");
    printf(" *  12.ListTraverse   * *  19.setListnum           * \n");

```

```

printf("*****\n");
printf("*   21.CreateList      * *   22.ShowList      *\n");
printf("*****\n");
printf("*   23.AddList          * *   24.MinusList      *\n");
printf("*****\n");
printf("-----\n");
printf("请选择你的操作[0-24]:");
}

int main (){
    int xx,y,num1,num2,num3,i;
    ElemType ee,cure;
    LinkList L = NULL;
    menu();
    scanf("%d",&xx);
    while (xx!=0){
        switch(xx){
            case 0:{
                int i=1;
                while (array[i]!= NULL &&i<11){
                    DestroyList(array[i]);
                    i++;
                }
                break;
            }
            case 19:{
                printf("请输入您要设置的线性表编号: ");
                scanf("%d",&y);
                setListnum(y);
                printf("设置成功! \n");
                system("pause");
            }
        }
    }
}

```

```

        menu();

        break;
    }
case 1:{
    InitList(L);
    printf("线性表创建成功! \n");
    system("pause");
    menu();
    break;
}
case 2:{
    DestroyList(array[num]);
    printf("线性表销毁成功! \n");
    system("pause");
    menu();
    break;
}
case 3:{
    ClearList(array[num]);
    printf("线性表清空成功! \n");
    system("pause");
    menu();
    break;
}
case 4:{
    if (ListEmpty(array[num]) == TRUE) printf("该线性表为空 \n"); else
printf("该线性表不为空\n");
    system("pause");
    menu();
    break;
}

```

```

        }
    case 5:{
        Length = ListLength(array[num]);
        printf("线性表长度为 %d\n",Length);
        system("pause");
        menu();
        break;
    }
    case 6:{
        printf("请输入您要获取的元素的位置: ");
        scanf("%d",&y);
        y = GetElem(array[num],y,&cure);
        if (y!=ERROR) printf("您要获取的元素为 %c\n",cure); else
printf("NoElem! 元素不存在! \n");
        system("pause");
        menu();
        break;
    }
    case 7:{
        getchar();
        printf("请输入您要查找的元素: ");
        scanf("%c",&ee);
        getchar();
        y = LocateElem(array[num],ee,Compare);
        if (y!=ERROR) printf("您要查找的元素在第%d个\n",y); else
printf("NoElem! 不存在该元素! \n");
        system("pause");
        menu();
        break;
    }

```

```

case 8:{
    cure = ee;
    y = PriorElem(array[num],cure,&ee);
    if (y!=ERROR) printf("当前元素的前驱是 %c\n",ee); else
printf("NoElem! 当前已经是第一个元素, 不存在前驱! \n");
    system("pause");
    menu();
    break;
}
case 9:{
    cure = ee;
    y = NextElem(array[num],cure,&ee);
    if (y!=ERROR) printf("当前元素的后继是 %c\n",ee); else
printf("NoElem! 当前已经是最后一个元素, 不存在后继! \n");
    system("pause");
    menu();
    break;
}
case 10:{
    printf("请输入您要插入元素的位置与元素（中间以空格隔开）：
");

    scanf("%d %c",&y,&cure);
    getchar();
    ListInsert(array[num],y,cure);
    printf("插入成功! \n");
    system("pause");
    menu();
    break;
}
case 11:{

```

```

printf("请输入您要删除的元素的位置: ");
scanf("%d",&y);
ListDelete(array[num],y,&cure);
printf("您要删除的元素是%c\n",cure);
printf("元素删除成功! \n");
system("pause");
menu();
break;
    }
case 12:{
    printf("该表遍历结果如下\n");
    ListTraverse(array[num],visit);
    system("pause");
    menu();
    break;
    }
case 21:{
    printf("请输入您要输入的多项式的项数: ");
    scanf("%d",&y);
    printf("请依次输入多项式的系数与指数: ");
    for (i=0;i<=99;i++){
        sz[i].co = 0;
        sz[i].exp = 32765;
    }
    for (i=0; i<y;i++){
        scanf("%lf %d",&sz[i].co,&sz[i].exp);
    }
    CreatList(y,num);
    printf("多项式存储成功!\n");
    system("pause");

```

```

        menu();

        break;
    }
case 22:{
    printf("请输入您要查看的多项式的编号： \n");
    scanf("%d",&y);
    ShowList(y);
    system("pause");
    menu();
    break;
}
case 23:{
    printf("请依次输入您要进加法运算的两个多项式编号与存放
结果多项式的编号： \n");
    scanf("%d%d%d",&num1,&num2,&num3);
    AddList(num1,num2,num3);
    printf("加法运算结束！ \n");
    system("pause");
    menu();
    break;
}
case 24:{
    printf("请依次输入您要进减法运算的两个多项式编号与存放
结果多项式的编号： \n");
    scanf("%d%d%d",&num1,&num2,&num3);
    MinusList(num1,num2,num3);
    printf("减法运算结束！ \n");
    system("pause");
    menu();
    break;
}

```



```
        }  
  
    }  
    scanf("%d",&xx);  
}  
return 0;  
}
```

附录 B：基于链表的二叉树实现源代码

```
/*
姓名:李欣宇
班级:信安 1901 班
学号: U201911658
日期:2020 年 11 月 5 日
*/

#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<iostream>
#include<string.h>
#include<windows.h>

using namespace std;

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define DestroyBiTree ClearBiTree    //销毁二叉树与清空二叉树操作相同

typedef char ElemType; //节点数据类型
typedef int Status;
typedef struct BNode{
    ElemType data;
    int weight;
    struct BNode* lchild; //指向左孩子
```

```

    struct BNode* rchild; //指向兄弟
}BNode;

typedef BNode* BTree;

//p_e 数组存放第 i 棵二叉树的当前节点的指针，array 数组用于多二叉树管理
BTree
p_e[11]={NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},array[11]
={NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL}; //数组管理多二
叉树

ElemType str[2000];

int ii=0,jj=0,k=0;

int num=0;

//W 数组用于构建哈夫曼树，是频数
int
W[100]={1000,408,191,92,45,22,28,47,99,48,51,217,103,114,57,57,592,251,123,60,
28,13,15,32,15,17,8,9,4,2,1,1,2,1,1,5,63,128,64,64,32,32,341,155,75,34,16,18,41,20,
21,80,186};

char ans[201],anscode[201];

int code_[31][201]={};

int a[100];

void menu(void);

void Visit(ElemType e);

void setBTnum(int n); //29 设置当前二叉树序号

void InitBiTree(BTree *T); //1 初始化

Status CreateBiTree(BTree *T); //3 创建二叉树

void ClearBiTree(BTree *T); //4 清空二叉树

Status BiTreeEmpty(BTree T); //5 判断空二叉树

Status BiTreeDepth(BTree T); //6 求二叉树深度

ElemType Root(BTree T); //7 获得根节点

```

```

BTree Order_Bi(BTree T,ElemType e); //返回指向 e 节点的指针
Status Value(BTree T,ElemType *e); //8 获得当前节点的值???
Status Assign(BTree T,ElemType e,ElemType value); //9 结点赋值
ElemType Parent(BTree T,ElemType e); //10 获得双亲
ElemType GetChild(BTree T,ElemType e,int order); //11&&12 用于找孩子
ElemType LeftChild(BTree T,ElemType e); //11 左孩子
ElemType RightChild(BTree T,ElemType e); //12 右孩子
ElemType LeftSibling(BTree T,ElemType e); //13 左兄弟
ElemType RightSibling(BTree T,ElemType e); //(13) //14 右兄弟
Status InsertChild(BTree T,BTree p,char LR,int tnum); //15 插入子树
Status DeleteChild(BTree T,BTree p,char LR); //16 删除子树
void PreOrderTraverse(BTree T,void(Visit)()); //17 前序遍历
void InOrderTraverse(BTree T,void(Visit)()); //18 中序遍历
void PostOrderTraverse(BTree T,void(Visit)()); //19 后序遍历
void LevelOrderTraverse(BTree T,void(Visit)()); //20 层序遍历
Status CreateHTree(BTree F,BTree *T,char string1); //31 创建哈夫曼树
void HTreeCode(char string1[]); //32 进行编码
void HTreeDecode(char code[]); //33 进行译码
Status StringCheck(char string1[]); //34 编码-译码-编码 检查
Status CodeCheck(char code[]); //35 译码-编码-译码 检查

void Visit(ElemType e){ //遍历时输出
    printf("%c",e);
}

void setBTnum(int n){ //29 设置树序号
    num = n;
}

void InitBiTree(BTree *T){ // 初始化二叉树
    *T = NULL;

```

```

}

Status CreateBiTree(BTree *T,char str[]){ //创建二叉树 ， 使用前序遍历序列

    char ch;

    ch = str[ii++];

    if (ch == '^') //如果 ch 是^,则当前节点为 NULL

        *T = NULL;

    else {

        *T = (BTree)malloc(sizeof(BNode)); //开辟一个节点空间

        if (jj==0) array[num]= *T; //如果是根节点， 则把该指针存入多二叉树数组
        管理中， 便于后续访问

        jj++; //这里的 jj 是全局变量， 在主函数中调用该函数时进行赋初值 0 ，
        作用是判断当前节点是不是根节点

        if(!(*T)) return  OVERFLOW; //防止溢出

        (*T)->data = ch; //对节点数据域进行赋值

        CreateBiTree(&(*T)->lchild,str); //递归创建左子树

        CreateBiTree(&(*T)->rchild,str); //递归创建右子树

    }

    return OK;

}

void ClearBiTree(BTree *T){ //清空二叉树， 与销毁二叉树操作相同

    if (*T){

        if ((*T)->lchild) ClearBiTree(&(*T)->lchild); //只要左子树不为空， 递归销
        毁左子树

        if ((*T)->rchild) ClearBiTree(&(*T)->rchild); // 只要右子树不为空， 递归
        销毁右子树

        free(*T); //释放空间

        *T = NULL;

    }

}

```

```

Status BiTreeEmpty(BTree T){    //判断空二叉树

    return T==NULL ? TRUE :FALSE; //根节点为空则返回 TRUE

}

Status BiTreeDepth(BTree T){ //求树的深度

    if (T == NULL) return 0; //递归出口，当前节点为 NULL

    else return max(1+BiTreeDepth(T->lchild),1+BiTreeDepth(T->rchild)); //递归
求深度

}

ElemType Root(BTree T){ //求树的根节点

    if (T){

        p_e[num] = T;    //如果多二叉树数组中指示的根节点存在，则返回根节
点的数据域

        return T->data;

    }

    else

        return '^'; //不存在则为空

}

Status Value(BTree T,ElemType *e){ //获得当前节点

    *e = p_e[num]->data; //使用 e 返回

    return OK;

}

Status Assign(BTree T,ElemType *e,ElemType value){//节点赋值

    *e = value;

    return OK;

}

```

```

ElemType Parent(BTree T,ElemType e){ //10 获得双亲
    int i=0,j=0;

    BTree Q[100]={}; //使用队列

    if (T){ //如果根节点不为空，把根节点加入队列
        Q[j++] = T;
    }

    if (Q[i]==p_e[num]) return '^'; //如果恰好根节点与当前节点相同，则说明没有双亲，已经是根节点，返回 NULL

    while (i<j){
        if (Q[i]){ //只要队列当前的不为空
            if(Q[i]->lchild){ //如果左孩子
                if(Q[i]->lchild->data == p_e[num]->data){ //且左孩子的值恰好与当前节点的值相同，则说明队列当前节点为其父节点
                    p_e[num] = Q[i];
                    return Q[i]->data; //返回父节点
                }
            }
            if (Q[i]->rchild){ //如果右孩子存在
                if(Q[i]->rchild->data == p_e[num]->data){ // 且右孩子的值恰好与当前节点的值相同，则说明队列当前节点为其父节点
                    p_e[num] = Q[i];
                    return Q[i]->data; //返回父节点
                }
            }
            Q[j++] = Q[i]->lchild; //把左孩子加入队列
            Q[j++] = Q[i]->rchild; //把右孩子加入队列
        }
        i++; //后移一个
    }
}

```

```
}
```

```
ElemType LeftChild(BTree T,ElemType *e){//11 找左孩子
    if (p_e[num]->lchild) { //只要当前节点的左孩子村子
        p_e[num]=p_e[num]->lchild; //把当前节点修改为其左孩子
        return p_e[num]->data; //返回左孩子
    }
    return '^'; //没有左孩子，返回 NULL
}
```

```
ElemType RightChild(BTree T,ElemType e){// 找右孩子，思想同上
    if(p_e[num]->rchild){
        p_e[num] = p_e[num]->rchild;
        return p_e[num]->data;
    }
    return '^';
}
```

```
ElemType LeftSibling(BTree T,ElemType e) //找左兄弟
{
    int i=0,j=0;
    BTree Q[100]; //使用队列
    if (T){
        Q[j++] = T; //加入根节点
    }
    if(Q[i]->data==p_e[num]->data) return '^'; //如果根节点与当前节点相同，则无
兄弟，返回 NULL
    while (i<j){
```



```

        if (Q[i]){ //如果队列不空
            if(Q[i]->rchild==p_e[num]) { //如果队列元素的右孩子与当前节点相同
                if (Q[i]->lchild) { //查看是否又左孩子
                    p_e[num] = Q[i]->lchild; //如果有左孩子，则查找成功
                    return p_e[num]->data; //返回其左兄弟
                }
                else return '^'; //如果没有左孩子，说明没有左兄弟
            }

            if(Q[i]->lchild==p_e[num]) return '^'; //如果要查找的节点与队列当前元素的左孩子相等，则其必然没有左兄弟，返回 NULL

            Q[j++] = Q[i]->lchild; //左孩子加入队列
            Q[j++] = Q[i]->rchild; //右孩子加入队列
        }
        i++; //后移一个
    }

}

ElemType RightSibling(BTree T,ElemType e){ //找右兄弟，思想同上
    int i=0,j=0;
    BTree Q[100];
    if (T){
        Q[j++] = T;
    }

    if(Q[i]->data==p_e[num]->data) return '^'; //如果根节点与当前节点相同，则无兄弟，返回 NULL

    while (i<j){
        if (Q[i]){
            if(Q[i]->lchild==p_e[num]) {
                if (Q[i]->rchild) {

```

```

        p_e[num] = Q[i]->rchild;
        return p_e[num]->data;
    }
    else return '^';
}

if(Q[i]->rchild==p_e[num]) return '^';
    Q[j++] = Q[i]->lchild;
    Q[j++]=Q[i]->rchild;
}

i++;
}
}

Status InsertChild(BTree T,BTree p,char LR,int tnum){ //插入子树
    BTree tp=p_e[num],temp=NULL,childtree=array[tnum];
    switch (LR){
        case 'L':{
            if (p->lchild){ //如果左子树本身存在，则将其放入要插入的子树的
                //右子树上
                temp = p->lchild;
                p_e[num]->lchild = array[tnum];
                childtree->rchild = temp;
                array[tnum]= NULL; //将子树在列表中删除
            }
            else { //如果左子树为空，则直接插入
                p_e[num]->lchild = array[tnum];
                array[tnum] = NULL; //将子树在列表中删除
            }
            break;
        }
    }
}

```

```

        case 'R':{          //思想同上
            if (p->rchild){
                temp = p->rchild;
                p_e[num]->rchild = array[tnum];
                childtree->rchild = temp;
                array[tnum]= NULL;
            }
            else {
                p_e[num]->rchild = array[tnum];
                array[tnum] = NULL;
            }
            break;
        }
    }
}

Status DeleteChild(BTree T,BTree p,char LR)//16  删除子树
{
    switch(LR){
        case 'L':{
            ClearBiTree(&p->lchild); //清空左子树
            break;
        }
        case 'R':{
            ClearBiTree(&p->rchild); //清空右子树
            break;
        }
    }
}
}

```

```

void PreOrderTraverse(BTree T,void(Visit)(ElemType)){//前序遍历 递归
    if(T){
        Visit(T->data); //根节点
        PreOrderTraverse(T->lchild,Visit); //左孩子
        PreOrderTraverse(T->rchild,Visit);//右孩子
    }
}

Status InOrderTraverse(BTree T,void(Visit)(ElemType)){//中序遍历 非递归
    BTree S[100]; //使用栈
    int top = 0; //栈顶
    do{
        while(T){ //只要当前节点不为空
            if(top == 100) return OVERFLOW; //栈溢出
            S[top++] =T; //根加入栈中
            T= T->lchild; //找左孩子（因中序遍历是 根 左 右 ）
        }
        if (top){ //如果栈不为空
            T = S[--top]; //栈顶元素弹出栈
            Visit(T->data);
            T = T->rchild; //找右孩子
        }
    }while(top || T); //只要节点不为空或栈不为空
    return OK;
}

void PostOrderTraverse(BTree T,void(Visit)(ElemType))//19 后续遍历
{
    if(T){
        PostOrderTraverse(T->lchild,Visit); //左
        PostOrderTraverse(T->rchild,Visit); //右
    }
}

```

```

        Visit(T->data); //根
    }
}

void LevelOrderTraverse(BTree T,void(Visit)(ElemType)){ //层序遍历
    int i=0,j=0;
    BTree Q[100]={}; //使用队列
    if (T){
        Q[j++] = T; //根节点加入队列中
    }
    while (i<j){
        if (Q[i]){
            Visit(Q[i]->data); //输出
            Q[j++] = Q[i]->lchild; //加左孩子
            Q[j++] = Q[i]->rchild; //加右孩子 即把下一层的加入
        }
        i++;
    }
    printf("\n");
}

Status CreateHTree(BTree F,BTree *T,char string1[]) //31 创建哈夫曼树
{
    char ch;
    ch = string1[ii++]; //取字符
    if (ch == '^'){

        *T= NULL; //为空则当前指针值置为 NULL
    }
    else {
        *T = (BTree)malloc(sizeof(BNode)); //开辟空间
    }
}

```

if (jj==0) array[num]= *T; //根节点加入数组，便于后续使用，jj 为全局变量

jj++;

(*T)->data = ch; //加入字符

(*T)->weight = W[k++]; //加入权值

CreateHTree(*T,&(*T)->lchild,string1); //构建左子树

CreateHTree(*T,&(*T)->rchild,string1); //构建右子树

}

return OK;

}

void HTCode(BTree T,int len){ //所有叶子节点编码，存入 code_二维数组（每一行对应一个字母的编码，以-1 结尾，共 27 行），便于后续编码直接使用

int temp;

if(!T) return;

int i;

if (!T->lchild && !T->rchild){ //只要左子树和右子树有一个不为空

for (int i=0;i<len;i++){

if (T->data == ' ') temp = 0; else temp=(int)(T->data)-64;

code_[temp][i]=a[i]; //temp 对应 26 个字母分别为 1-27，空格为

0

}

}

else {

a[len] = 0; //赋值，用于 0，1 的编码

HTCode(T->lchild,len+1); //找左子树

a[len] = 1;

```

        HTCode(T->rchild, len+1);    //找右子树
    }
}

void HTreeCode(char string1[]) //32 编码，存储在 anscode 数组里
{
    int k = 0;
    int length = strlen(string1);    //获得字符串长度
    int temp;
    for (int i = 0; i < length; i++) {    //i 指示字符，从第一个到最后一个
        int j = 0; //j 指示 code_
        temp = string1[i] - 64;    //转换，便于使用 code_ 数组
        if (temp < 0) temp = 0;    //如果为空格，则 temp 小于 0，赋值为 0
        while (code_[temp][j] != -1) {    //只要没有到最后（code_ 初值为 -1，当编
            码结束后的所有均为 -1）
                anscode[k++] = code_[temp][j] + 48;    //把当前 code_ 对应的编码加入
            anscode 中
                j++;
            }
        }
    }

}

void HTreeDecode(char code[]) //33 译码
{
    int k = 0;
    char ch;
    int len = strlen(code), i = 0;
    while (i < len) {
        BTree p = array[num];
        while (p) {

```

```

        if(!p->lchild && !p->rchild){ //先加入当前节点
            ans[k++]=p->data; //结果存入 ans 字符数组
            break;
        }
        else {
            if(((int)(code[i]))-48) p=p->rchild,i++; //如果为 1，则找右子树，
            如果为 0，则找左子树
            else p=p->lchild,i++;
        }
    }
}
}

```

}

Status StringCheck(char string1[])//34 编码-解码-比对

```

{
    //基本思想 即为编码-解码-编码得到的字符串与原串比较
    HTreeCode(string1);
    char temps[201];
    for (int i=0;i<201;i++) temps[i]=anscode[i];
    for (int kk=0;kk<200;kk++) ans[kk]='\0';
    HTreeDecode(temps);
    if (!strcmp(ans,string1)) return OK;
    else return ERROR;
}

```

}

Status CodeCheck(char code[])//35 解码-编码-比对

```

{
    //基本思想 即为解码-编码-解码得到的字符串与原串比较
    for (int kk=0;kk<200;kk++) ans[kk]='\0';
}

```



```

HTreeDecode(code);

HTreeCode(ans);

if(!strcmp(anscode,code)) return OK;

    else return ERROR;

}

void menu(){

    system("cls");

    printf("*****\n\n");
    printf(" *           Menu for Function of Binary Tree           * \n\n");
    printf("*****\n\n");
    printf(" ----- \n");
    printf("***** \n");
    printf(" *    0.EXIT          * *    1.IntiaBiTree          * \n");
    printf("***** \n");
    printf(" *    2.DestroyBiTree * *    3.CreateBiTree         * \n");
    printf("***** \n");
    printf(" *    4.ClearBiTree   * *    5.BiTreeEmpty          * \n");
    printf("*****\n");
    printf(" *    6.BiTreeDepth   * *    7.Root                 * \n");
    printf("*****\n");
    printf(" *    8.Value          * *    9.Assign               * \n");
    printf("*****\n");
    printf(" *   10.Parent         * *   11.LeftChild            * \n");
    printf("***** \n");
    printf(" *   12.RightChild     * *   13.LeftSibling           * \n");
    printf("***** \n");
    printf(" *   14.RightSibling   * *   15.InsertChild          * \n");
    printf("***** \n");
    printf(" *   16.DeleteChild    * *   17.PreOrderTraverse     * \n");

```

```

printf("***** \n");
printf("* 18.InOrderTraverse * * 19.PostOrderTraverse * \n");
printf("***** \n");
printf("* 20.LevelOrderTraverse * * 29.SetBTnum * \n");
printf("***** \n");
printf("* 31.CreateHTree * * 32.HTreeCode * \n");
printf("***** \n");
printf("* 33.HTreeDecode * * 34.StringCheck * \n");
printf("***** \n");
printf("* 35.CodeCheck * * * * * * \n");
printf("***** \n");
printf("----- \n");
printf("请选择你的操作[0-35]:");
}

int main(){
    int xx,y,num1,num2,num3;
    ElemType ee,cure;
    BTree T=NULL;
    menu();
    scanf("%d",&xx);
    while(xx!=0){
        switch(xx){
            case 0:{
                int i=0;
                while(array[i]!=NULL){

                    DestroyBiTree(&array[i]);

                    i++;
                }
                break;
            }
        }
    }
}

```

```

}

case 29:{    // 设置树的序号

    printf("请输入您要设置的二叉树的编号: ");
    scanf("%d",&y);
    setBTnum(y);
    printf("设置成功! \n");
    system("pause");
    menu();

    break;
}

case 1:{    //初始化

    printf("二叉树初始化成功! \n");
    InitBiTree(&T);
    system("pause");
    menu();

    break;
}

case 2:{    //销毁二叉树

    DestroyBiTree(&array[num]);
    printf("二叉树销毁成功! \n");
    system("pause");
    menu();

    break;
}

case 3:{    //构建树

    printf("请输入您要创建的二叉树的先序遍历序列: \n");
    ii=jj=0; //两个全局变量在函数中使用, 进行赋初值
    getchar();
    scanf("%s",str);

    CreateBiTree(&array[num],str);

```

```

        printf("二叉树创建成功!\n");
        system("pause");
        menu();
        break;
    }
    case 4:{ //清空树

        ClearBiTree(&array[num]);
        printf("二叉树清空成功!\n");
        system("pause");
        menu();
        break;
    }
    case 5:{ //判断树空
        if(BiTreeEmpty(array[num])) printf("该树为空\n"); else printf("该树
不为空\n");
        system("pause");
        menu();
        break;
    }
    case 6:{ //树的深度
        y = BiTreeDepth(array[num]);
        printf("树的深度为%d\n",y);
        system("pause");
        menu();

        break;
    }
    case 7:{ //根节点
        cure = Root(array[num]);

```

```

        printf("根节点的值为%c\n",cure);
        system("pause");
        menu();
        break;
    }
    case 8:{ //获得当前节点，对该程序而言设定初始为获得根节点为
当前节点
        Value(array[num],&ee);
        printf("当前节点的值为%c\n",ee);
        system("pause");
        menu();
        break;
    }
    case 9:{ // 改变当前节点的值
        printf("请输入您要替换当前节点的值: ");
        getchar();
        scanf("%c\n",&ee);
        Assign(array[num],&(p_e[num]->data),ee);
        printf("当前节点的值替换成功! \n");
        system("pause");
        menu();
        break;
    }
    case 10:{ //求双亲节点
        cure = Parent(array[num],ee);
        if (cure == '^') printf ("当前节点已经是根节点，没有双亲!\n"); else
printf("当前节点的父节点是%c\n",cure);
        // printf("当前节点父节点是%c\n",cure);
        system("pause");
        menu();
    }
}

```

```

        break;
    }
    case 11:{ //左孩子
        cure = LeftChild(array[num],&ee);
        if (cure == '^') printf ("当前节点没有左孩子\n"); else    printf("当前
节点的左孩子是%c\n",cure);
        //printf("当前节点的左孩子是%c\n",cure);
        system("pause");
        menu();
        break;
    }
    case 12:{ //右孩子
        cure = RightChild(array[num],ee);
        if (cure == '^') printf ("当前节点没有右孩子\n"); else    printf("当前
节点的右孩子是%c\n",cure);
        // printf("当前节点的右孩子是%c\n",cure);
        system("pause");
        menu();
        break;
    }
    case 13:{ //左兄弟
        cure = LeftSibling(array[num],ee);
        if (cure == '^') printf ("当前节点没有左兄弟\n"); else    printf("当前
节点的左兄弟是%c\n",cure);
        system("pause");
        menu();
        break;
    }
    case 14:{ //右兄弟
        cure = RightSibling(array[num],ee);

```

```

        if (cure == '^') printf ("当前节点没有右兄弟\n"); else    printf("当前
节点的右兄弟是%c\n",cure);

        system("pause");

        menu();

        break;
    }

    case 15:{ //插入子树

        printf ("请输入您要插入的当前节点的左子树还是右子树(L or R)
和要插入的子树的编号（以空格隔开）\n");

        //getchar();

        scanf("%c %d\n",&cure,&y);

        InsertChild(array[num],p_e[num],cure,y);

        printf ("子树插入成功！ \n");

        system("pause");

        menu();

        break;
    }

    case 16:{ //删除子树

        printf ("请输入您删除当前节点的左子树还是右子树(L or R)\n");

        //getchar();

        scanf("%c\n",&cure);

        DeleteChild(array[num],p_e[num],cure);

        printf ("子树删除成功！ \n");

        system("pause");

        menu();

        break;
    }

    case 17:{ //前序遍历

        printf ("前序遍历为： ");

        PreOrderTraverse(array[num],Visit);
    }

```

```

        printf("\n");
        system("pause");
        menu();
        break;
    }
    case 18:{ //中序遍历
        printf ("中序遍历为: ");
        InOrderTraverse(array[num],Visit);
        printf("\n");
        system("pause");
        menu();
        break;
    }
    case 19:{ //后序遍历
        printf ("后序遍历为: ");
        PostOrderTraverse(array[num],Visit);
        printf("\n");
        system("pause");
        menu();
        break;
    }
    case 20:{ //层序遍历
        printf ("层序遍历为: ");
        LevelOrderTraverse(array[num],Visit);

        system("pause");
        menu();

        break;
    }

```



```

case 31:{ //构建哈夫曼树

    printf("请输入您要创建的哈夫曼树的序列： \n");

    getchar();

    ii=jj=k=0;

    scanf("%s",str);

    for (int i=0;i<30;i++)
        for (int j=0;j<200;j++)
            code_[i][j]=-1;

    CreateHTree(NULL,&array[num],str);

    HTCode(array[num],0);

    printf("哈夫曼树创建成功！ \n");

    system("pause");

    menu();

    break;

}

case 32:{ //进行编码

    printf("请输入您要进行编码的字符串： \n");

    getchar();

    scanf("%s",str);

    //cout<<str<<endl;

    //getchar();

    HTreeCode(str);

    printf("编码的结果为");

    cout<<anscode<<endl;

    system("pause");

    menu();

    break;

}

case 33:{ //进行解码

```

```

        printf("请输入您要进行解码的字串 : \n");
        getchar();
        scanf("%[^\\n]",str);
        //getchar();
        HTreeDecode(str);
        printf("解码结果为 : ");
        cout<<ans<<endl;
        system("pause");
        menu();
        break;
    }
    case 34:{ //编码-解码-编码 比对

        for (int kk=0;kk<200;kk++) ans[kk]='\0';
        getchar();
        scanf("%[^\\n]",str);
        if(StringCheck(str)) printf(" 编码 - 解码 - 编码结果匹配\n"); else
printf("编码-解码-编码不匹配\n");
        system("pause");
        menu();
        break;
    }
    case 35:{ //解码-编码-编码 比对
        getchar();
        char strr[200];
        scanf("%s",str);
        if(CodeCheck(str)) printf(" 解码 - 编码 - 解码结果匹配\n"); else
printf("解码-编码-解码不匹配\n");
        system("pause");
        menu();
    }
}

```

```
        break;
    }

}

scanf("%d",&xx);

}

return 0;

}
```