

长江师范学院

计算机工程学院

综合课程设计文档

课 程 名 称 数据结构与算法综合课程设计

专 业 、 班 级 17 级计算机科学与技术(非师范)班

小 组 成 员 姓 名 及 学 号 : 胡令 (201706571325)

姓 名 及 学 号 : 杨柳 (201706571331)

指 导 教 师 皮晓炜

完 成 日 期 2018 年 7 月 6 日

目 录

1. 应用问题算法编程	- 1 -
1.1 问题一：一元多项式表示及相加.....	- 1 -
1.2 问题二：约瑟夫环问题处理.....	- 6 -
1.3 问题三：哈夫曼树的建立及哈夫曼编码	- 10 -
1.4 问题四：希尔排序.....	- 14 -
2. 课程设计总结.....	- 17 -

1. 应用问题算法编程

1.1 问题一：一元多项式表示及相加

1. 问题描述及分析

问题描述：用两个单链表分别表示两个一元多项式，在不另开销空间的情况下，实现两个多项式相加。

问题分析：两个多项式相加问题实质是将两个多项式中相同幂次的系数相加，不同的幂次的项合并到其中一个多项式中，并将这个多项式作为和多项式。

2. 功能模块及数据结构描述

功能模块：解决一元多项式相加问题的程序应包括：1.从键盘读入数据建立一个一元多项式的函数，2.打印输出一元多项式信息的函数，3.两个一元多项式相加的函数，三个功能函数。在主函数中按照一定的逻辑顺序调用这三个函数完成将两个一元多项式相加生成和多项式的问题。

把一元多项式每一项按升幂的顺序保存在单链表中，单链表的每一个结点包含三个域（系数域，指数域，下一个结点指针域）如图 1 所示：

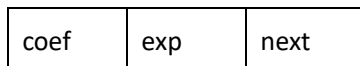


图 1 一元多项式单链表结点结构

结点结构定义如下：

```
typedef struct Polynode
```

```
{
```

```
    Int coef;           //系数
```

```
    Int exp;            //指数
```

```
    struct Polynode*next; //指针域：指向下一结点
```

```
}Polynode,*Polylist;
```

3. 详细设计及主要算法流程描述

从键盘读入数据建立一元多项式函数流程：先在主函数建立一个一元多项式链表指

针，以这个指针变量为实参调用一元多项式链表建立函数，然后通过键盘输入一组多项式的系数和指数，用尾插法建立一元多项式的链表，以输入系数 0 0 位结束标志，并约定建立多项式链表时，总是按指数从小到大的顺序排列。

两个一元多项式相加函数流程：1.指数相同项的对应系数相加，若和不为零，则构成和多项式的一项；2.指数不相同的项仍按升幂顺序复制到和多项式中。具体：逐步比较每一项的指数相同的指数项对应系数相加，如果不为零，就构成多项式的一项；指数不相同就按升幂插入一元多项式。

4. 组内分工情况

胡令：一元多项式加法函数，主函数，最后调试，报告中负责详细设计及主要算法流程描述及调试结果和问题。

杨柳：头文件及结构体类型，尾插法建立一元多项式的链表，报告中负责问题描述及分析，功能模块及数据结构类型。

5. 完整程序代码

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

/*一元多项式链式存储结点结构体声明*/
typedef struct Polynode{
    int coef;           //系数
    int exp;            //指数
    struct Polynode* next; //指针域：指向下一结点
}Polynode,*Polylist;

/*用尾插法建立一元多项式的链表*/
Polylist PolyCreate()
{
    Polynode* head,*rear,*s;
    int c,e;
    printf("输入多项式的系数和指数，使用英文逗号隔开，以 0, 0 结束：\n");
    head=(Polynode*)malloc(sizeof(Polynode)); //建立多项式的头结点
    rear=head; //rear 始终指向单链表的尾，便于尾插法建表
    scanf("%d,%d",&c,&e); //输入多项式的系数和指数项
    while(c!=0) //若 c=0，则代表多项式的输入结束
    {
        s=(Polynode *)malloc(sizeof(Polynode)); //申请新的结点
```

```
s->coef=c;
s->exp=e;
rear->next=s;          //在当前表尾做插入
rear=s;
scanf("%d,%d",&c,&e);
}
rear->next=NULL;        //将表的最后一个结点的 next 置 NULL，以示表结束
return(head);
}

/*一元多项式加法*/
void PolyAdd(Polylist head1,Polylist head2)
{
    PolyNode *p,*q,*last,*t;    //t:用来释放内存
    int coef_sum;                //系数和
    p=head1->next;               //p 指向多项式 1 的第一个结点
    q=head2->next;               //q 指向多项式 2 的第一个结点
    last=head1;                  //last 指向和多项式的最后一个结点
    while(p!=NULL&&q!=NULL) //当其中一个多项式扫描完成后退出循环
    {
        if(p->exp<q->exp)
            /*将指数小的多项式链入到 last 后面,链入: p*/
            {
                last->next=p;
                last=p;
                p=p->next;
            }//if
        else if(p->exp==q->exp)
            /*指数相等*/
            {
                coef_sum=p->coef+q->coef;
                if(coef_sum!=0)    //系数之和不等于 0
                    /*修改 p 所指结点系数，链入到 last 之后，释放 q 所指结点*/
                    {
                        p->coef=coef_sum;
                        last->next=p;
                        last=p;
                        p=p->next;
                        t=q;
                        q=q->next;
                        free(t);
                    }//if
                else
                    /*系数等于 0，释放这两个结点*/
```

```
        {
            t=p;
            p=p->next;
            free(t);
            t=q;
            q=q->next;
            free(t);
        }//else
    }//else if
    else
    /*指数小的多项式链入到 last 后面，链入: q*/
    {
        last->next=q;
        last=q;
        q=q->next;
    }//else
}//while
if(p!=NULL)        //如果此时其中一个多项式还有剩余，链入 last 之后
    last->next=p;
else
    last->next=q;
}

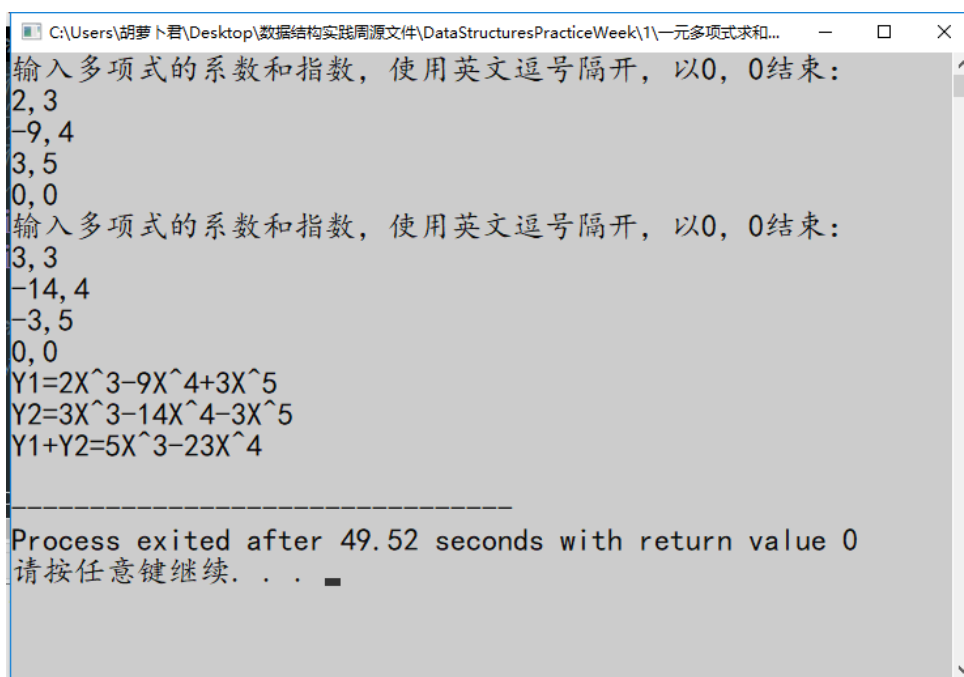
/*打印输出链表*/
void showList(Polylist head)
{
    Polylist p;
    p=head->next;
    while(p)
    {
        if(p!=head->next&& p->coef>0)
            printf("+");
        printf("%dX^%d",p->coef,p->exp);
        p=p->next;
    }
    printf("\n");
}

int main()
{
    Polylist head1=NULL,head2=NULL;
    /*建立多项式*/
    head1=PolyCreate();
    head2=PolyCreate();
}
```

```
/*输出多项式 1、2*/  
printf("Y1=");  
showList(head1);  
printf("Y2=");  
showList(head2);  
/*加法运算 */  
PolyAdd(head1,head2);  
/*输出和多项式*/  
printf("Y1+Y2=");  
showList(head1);  
return 0;  
}
```

6. 调试过程和数据测试

程序运行后，分别输入数据建立两个一元多项式的单链表存储，利用一元多项式相加算法完成相加运算后，将和多项式信息打印输出，运行结果如下图所示：



```
C:\Users\胡萝卜君\Desktop\数据结构实践周源文件\DataStructuresPracticeWeek\1\一元多项式求和...  
输入多项式的系数和指数，使用英文逗号隔开，以0, 0结束：  
2, 3  
-9, 4  
3, 5  
0, 0  
输入多项式的系数和指数，使用英文逗号隔开，以0, 0结束：  
3, 3  
-14, 4  
-3, 5  
0, 0  
Y1=2X^3-9X^4+3X^5  
Y2=3X^3-14X^4-3X^5  
Y1+Y2=5X^3-23X^4  
  
-----  
Process exited after 49.52 seconds with return value 0  
请按任意键继续. . .
```

7. 调试过程中遇到的主要问题，及解决办法

当两个一元多项式的系数互为相反数时，应释放的是 t 指针所指空间，而不是 p, q 指针所指空间。

8. 总结及体会

解决问题一运用了线性表中链式储存方式，在不开销新空间的情况下，完成一元多

项式的相加，主要注意到释放的是哪一个指针指向的空间，以及要注意指向和多项式的指针后移问题。

1.2 问题二：约瑟夫环问题处理

1. 问题描述及分析

问题描述：用循环队列实现约瑟夫环处理。若干人循环报数，凡报到 3 的倍数退出，最后剩下的一个人是第几人。

问题分析：要淘汰 3 的倍数的数，并求得剩下的数的序号实质是用一个计数器循环记录 1~3 次，当计数器加到 3 时，循环队列中 front 指向的数出队，而计数器记录 1 或 2 时，入队 front 指向的数并出队，循环此操作至队列剩一个数。

2. 功能模块及数据结构描述

功能模块：解决该问题的程序应包括：1.从键盘读入数据并入队，2.调用淘汰函数让三及三的倍数的序号出队（调用出队函数），3.调用函数获取队头元素。

在主函数中按照一定逻辑顺序调用三个函数。

循环队列类型定义如下：

```
typedef struct
{
    int element[MAX];    //数组表示循环队列
    int front;           //头指针
    int rear;            //尾指针
    int num;             //队列元素个数(人数)
}SeqQueue;
```

3. 详细设计及主要算法流程描述

循环队列的建立函数的流程：在主函数建立循环队列和输入人数，以队列地址和人数为实参调用建立函数，根据输入的人数初始化队列元素个数并输入序号。

循环报数找 3 的倍数函数的流程：以创建好的循环队列地址和数值 3 为实参调用该函数，报数为 1 和 2 的队列元素先入队到队列末尾，再出队列，报数为 3 的队列元素进行出队列操

作，直到队列元素个数为 1 的时候，结束报数，对头元素即为获胜者。

4. 组内分工情况

胡令：淘汰函数，主函数，最后调试，报告中负责详细设计及主要算法流程描述及调试结果和调试中遇到的问题。

杨柳：头文件及结构体类型，建立循环队列，报告中负责问题描述及分析，功能模块及数据结构类型。

5. 完整程序代码

```
#include<stdio.h>
#define MAX 50
#define FALSE 0
#define TRUE 1

//循环队列的类型定义
typedef struct
{
    int element[MAX];    //队列的元素空间
    int front;           //头指针指示器
    int rear;            //尾指针指示器
    int num;             //队列元素个数
}SeqQueue;

//初始化循环队列
void InitQueue(SeqQueue *Q)
{
    //将*Q 初始化为一个空的循环队列
    Q->front=Q->rear=0;
    Q->num=0;
}

//循环队列入队操作
int EnterQueue(SeqQueue *Q,int x)
{
    if((Q->rear+1)%MAX==Q->front)    //尾指针加 1 追上头指针，标志队列已经满了
        return(FALSE);
    Q->element[Q->rear]=x;
    Q->rear=(Q->rear+1)%MAX;          //重新设置队尾指针
    return(TRUE);
}
```

```
//循环队列出队
int DeleteQueue(SeqQueue *Q,int *x)    //删除队列的队头元素，用 x 返回其值
{
    if(Q->front==Q->rear)                //队列为空
        return(FALSE);
    *x=Q->element[Q->front];
    Q->front=(Q->front+1)%MAX;            //重新设置队头指针
    return(TRUE);                        //操作成功
}

//获取对头元素
int GetHead(SeqQueue *Q,int *x)
{
    if(Q->front==Q->rear)                //队列为空
        return(FALSE);
    else
    {
        *x=Q->element[Q->front];
        return(TRUE);
    }
}

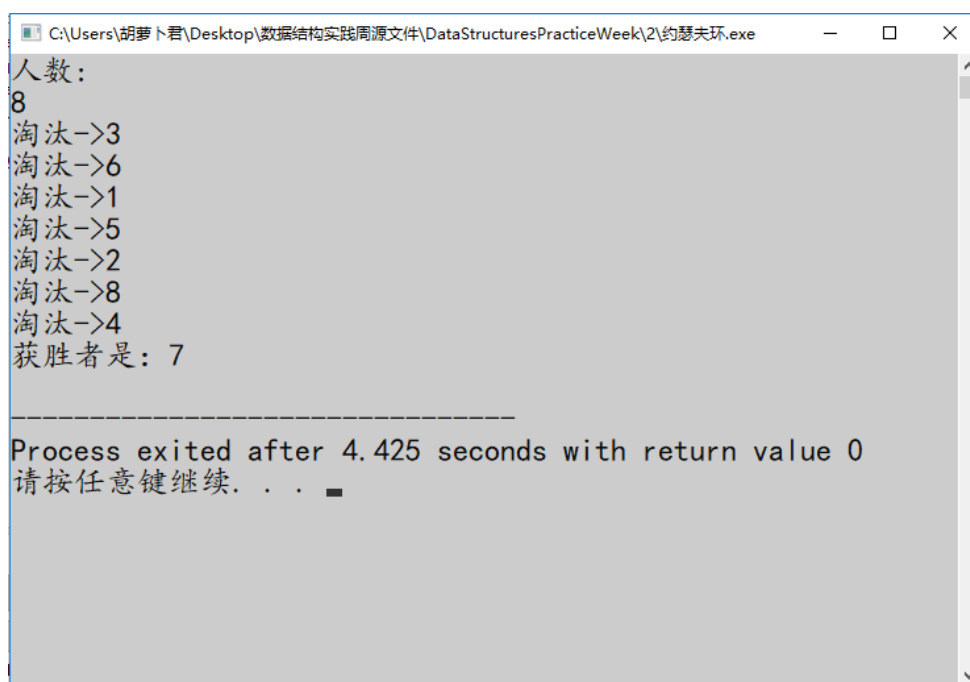
/*淘汰函数*/
void OutPeople(SeqQueue *Q,int n)
{
    int count;
    while(1)
    {
        count=1;
        while(count<n)
        {
            EnterQueue(Q,Q->element[Q->front]);    //非 3 的倍数入队
            DeleteQueue(Q,&Q->element[Q->front]);    //出队
            count++;
        }
        printf("淘汰->%d\n",Q->element[Q->front]);
        DeleteQueue(Q,&Q->element[Q->front]);        //淘汰队头
        Q->num--;    //人数减少 1
        if(Q->num==1)break;    //判断是否只有一个队列元素(循环退出条件)
    }
}

int main()
{
```

```
SeqQueue Q;
InitQueue(&Q);
int i,number,winner;
printf("人数:\n");
scanf("%d",&number);
getchar();
for(i=0;i<number;i++)          //入队列
    EnterQueue(&Q,i+1);
Q.num=number;                  //初始化为总人数
OutPeople(&Q,3);
GetHead(&Q,&winner);          //获取队头元素，即胜者
printf("获胜者是: %d\n",winner);
return 0;
}
```

6. 调试过程和数据测试

程序运行后，输入人数，利用调用函数实现约瑟夫环问题后输出，运行结果如下图所示：



```
C:\Users\胡萝卜君\Desktop\数据结构实践周源文件\DataStructuresPracticeWeek\2\约瑟夫环.exe
人数:
8
淘汰->3
淘汰->6
淘汰->1
淘汰->5
淘汰->2
淘汰->8
淘汰->4
获胜者是: 7

-----
Process exited after 4.425 seconds with return value 0
请按任意键继续. . .
```

7. 调试过程中遇到的主要问题，及解决办法

循环退出条件应该使用 $Q \rightarrow \text{num} == 1$ ，而不使用 $(Q \rightarrow \text{rear} - Q \rightarrow \text{front} + M) \% M == 1$ 。

8. 总结及体会

解决这个问题主要需要一个计数器来淘汰 3 及其 3 的倍数,还有让不是 3 的倍数的数出队后应即时入队。这个问题开始感觉挺难的,知道该怎么做后发现没有想象中那么难,平时学习不要只学基本知识,要多做题,特别是应用问题,把知识运用起来。

1.3 问题三：哈夫曼树的建立及哈夫曼编码

1. 问题描述及分析

问题描述：根据字符出现的频率创建哈夫曼树，并得出各字符的编码。

问题分析：该问题实质是创建哈夫曼树，再求哈夫曼数编码。

2. 功能模块及数据结构描述

功能模块：解决该问题程序包括：1.从键盘读入数据调用函数创建哈夫曼树，2.调用函数求哈夫曼编码，3.在主函数中按照一定逻辑顺序调用两个函数完成创建哈夫曼树及求哈夫曼编码问题。

用静态三叉链表实现的哈夫曼树类型定义如下：

```
#define N 20

#define M 2*N-1

typedef struct
{
    int weight;        //权值
    int parent;        //双亲结点下标
    int left;          //左孩子结点下标
    int right;         //右孩子结点下标
}tree,hfmtree[m+1];
```

3. 详细设计及主要算法流程描述

初始化函数的流程：在主函数定义结构体指针，以结构体指针为实参调用该函数，给结构体中的数据赋初值。

创建哈夫曼树的函数的流程：以结构体指针为实参调用该函数；在函数中输入一串字符并统计字符的权值；找出权值最小的两棵树，组成新树，重复操作直到只剩一棵树，然后返回字符的个数。

对字符进行哈夫曼编码函数的流程：以结构体指针为实参调用该函数；从字符所在

的叶子节点回到根节点，左子树就赋‘0’，右子树就赋‘1’；

输出函数：以结构体指针为实参调用该函数；输出结构体中的数据。

4. 组内分工情况

胡令：头文件及结构体类型，创建哈夫曼树，最后调试，报告中负责详细设计及主要算法流程描述及调试结果和问题。

杨柳：求哈夫曼树的哈夫曼编码，主函数，报告中负责问题描述及分析，功能模块及数据结构类型。

5. 完整程序代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 20
#define m 2*N-1

typedef char *hfmcode[N+1];

typedef struct
{
    int weight;          //权值
    int parent;          //双亲结点下标
    int left;            //左孩子结点下标
    int right;           //右孩子结点下标
}tree,hfmtree[m+1];

hfmtree ht;

/*在 1 到 i-1 中找出最小的两个结点权值最小的下标保存到 s1,s2*/
void select(hfmtree ht,int n,int &s1,int &s2)
{
    int i,j=0,x=0;
    int min1=999,min2=999;
    for(i=1;i<=n;i++)
    {
        if(min1>ht[i].weight&&ht[i].parent==0)
        {min1=ht[i].parent;
        j=i;}
    }
    s1=j;
```

```
for(i=1;i<=n;i++)
{
    if(min2>=ht[i].weight&&ht[i].parent==0&&i!=j)
    {min2=ht[i].parent;
    x=i;}
}
s2=x;
}
```

/*构建哈夫曼树*/

void create(hfmtree ht,int w[],int n)

```
{
    int i,M;
    int s1,s2;
    M=2*n-1;
    for(i=1;i<=n;i++)
    {
        ht[i].weight=w[i];
        ht[i].parent=0;
        ht[i].left=0;
        ht[i].right=0;
    }

    for(i=n+1;i<=M;i++)
    {
        ht[i].weight=0;
        ht[i].parent=0;
        ht[i].left=0;
        ht[i].right=0;
    }

    for(i=n+1;i<=M;i++)
    {
        select(ht,i-1,s1,s2);
        ht[s1].parent=i;ht[s2].parent=i;
        ht[i].left=s1;ht[i].right=s2;
        ht[i].weight=ht[s1].weight+ht[s2].weight;
    }
}
```

/*哈夫曼编码*/

void hfmbianma(hfmtree ht,hfrcode hc,int n)

```
{
    int i,c,start,p;
```

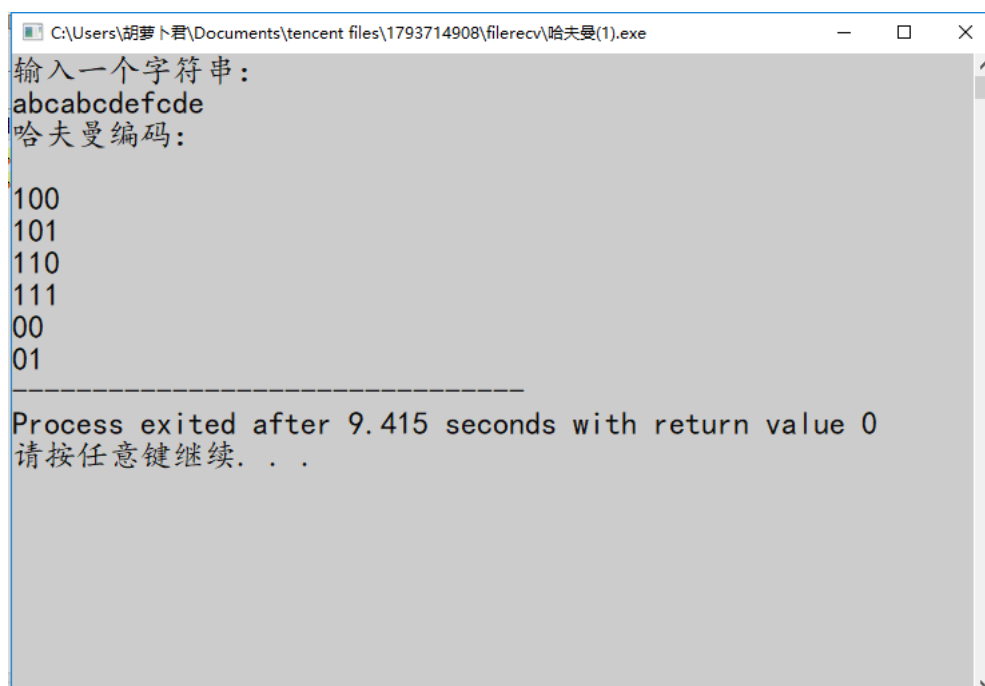
```
char *cd;
cd=(char *)malloc(n*sizeof(char));
cd[n-1]='\0';
for(i=1;i<=n;i++)
{
    start=n-1;
    c=i;
    p=ht[i].parent;
    while(p!=0)
    {
        --start;
        if(ht[p].left==c)
            cd[start]='0';
        else
            cd[start]='1';
        c=p;
        p=ht[p].parent;
    }
    hc[i]=(char *)malloc((n-start)*sizeof(char));
    strcpy(hc[i],&cd[start]);
}
free(cd);
}

int main()
{
    char c;
    hfmtree ht;
    int zifu[26]={0};
    int p[N];
    int i,n=1,y;
    printf("输入一个字符串: \n");
    scanf("%c",&c);
    for(;c!='\n';)
    {
        scanf("%c",&c);
        zifu[c-'a']++;
    }
    for(i=0;i<26;i++)
        if(zifu[i]!=0)
            p[n++]=zifu[i];
    y=n-1;
    create(ht,p,y);
    hfmtree hc;
```

```
hfmajianma(ht, hc, n);  
printf("哈夫曼编码: \n");  
for(i=1; i<=y; i++)  
    printf("\n%s ", hc[i]);  
return 0;  
}
```

6. 调试过程和数据测试

程序运行后，程序根据输入字符串进行哈夫曼树的创建和编码，运行结果如下图所示：



```
C:\Users\胡萝卜君\Documents\tencent files\1793714908\filerecv\哈夫曼(1).exe  
输入一个字符串:  
abcabcdefcde  
哈夫曼编码:  
  
100  
101  
110  
111  
00  
01  
  
-----  
Process exited after 9.415 seconds with return value 0  
请按任意键继续. . .
```

7. 调试过程中遇到的主要问题，及解决办法

建立哈夫曼树过程中注意数组下标的相关操作，例如 0 号空间的使用情况。

8. 总结及体会

解决该问题难点在于创建哈夫曼树算法中写 select 函数，写这个函数有参考其他同学的写法，其他部分注意基本语法操作即可。

1.4 问题四：希尔排序

1. 问题描述及分析

问题描述：实现希尔排序

问题分析：结合教材所讲的希尔排序算法操作

2. 功能模块及数据结构描述

功能模块：解决希尔排序问题的程序包括：1.从键盘中读入数据存入数组中，2.求出增量数组，3.调用函数排序

在主函数中按一定逻辑顺序调用函数完成希尔排序。数据结构类型为整型数组。

3. 详细设计及主要算法流程描述

先将待排序记录序列分割成若干个“较稀疏”子序列，分别进行直接插入排序。经过上述粗略调整，整个序列中的记录已经基本有序，最后再对全部记录进行一次直接插入排序。

首先选定记录间的距离为 $d_i(i=1)$ ，在整个待排序记录序列中，将所有记录为 d_1 的记录分成一组，进行组内直接插入排序。

然后取 $i=i+1$ ；记录间的距离为 $d_i(d_i < d_{i-1})$ ，在整个待排序记录序列中，将所有间隔为 d_i 的记录分成一组，进行组内直接插入排序。重复此操作，直到记录间的距离 $d_i=1$ ，此时整个只有一个子序列，对该序列进行直接插入排序，完成整个过程。

4. 组内分工情况

胡令：求增量数组部分，最后调试，报告中负责详细设计及主要算法流程描述及调试结果和问题。

杨柳：头文件及结构体类型，希尔排序函数，报告中负责问题描述及分析，功能模块及数据结构类型。

5. 完整程序代码

```
#include<stdio.h>
#define MAX 20

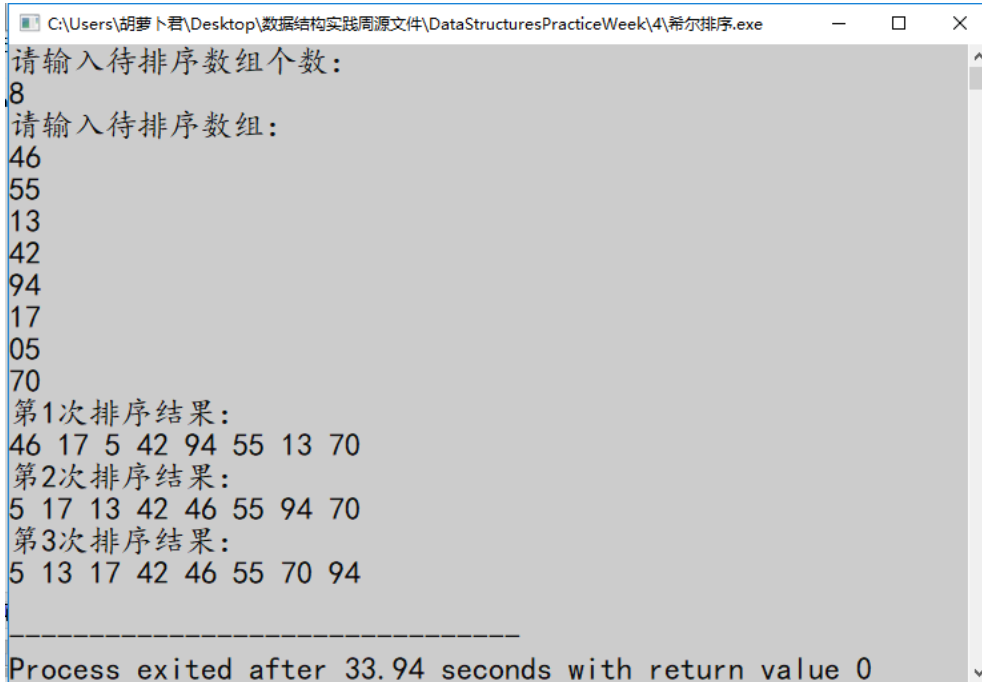
/*对记录数组 r 做一趟希尔插入排序，length 为数组的长度，delta 为增量 */
void ShellInsert(int r[],int length,int d)
{
    int i,j;
    for(i=1+d;i<=length;i++)                //1+delta 为第一个子序列的第二个元素的下标
        if(r[i]<r[i-d])
        {
            r[0]=r[i];                        //备份 r[i](不做监视哨)
            for(j=i-d;j>0&& r[0]<r[j];j-=d)
            {
```

```
        r[j+d]=r[j];
    }
    r[j+d]=r[0];
}

int main()
{
    int length,r[MAX],i,j,delta[MAX],count=0;
    printf("请输入待排序数组个数: \n");
    scanf("%d",&length);
    printf("请输入待排序数组: \n");
    for(i=1;i<=length;i++)
        scanf("%d",&r[i]);
    for(i=length;i>1;count++)
    {
        i=i/2;
        delta[count]=i;
    }
    for(i=0;i<count;i++)
    {
        ShellInsert(r,length,delta[i]);           //循环调用 ShellInsert 进行希尔排序
        printf("第%d 次排序结果: \n",i+1);
        for(j=1;j<=length;j++)                     //输出每次排序的结果
            printf("%d ",r[j]);
        printf("\n");
    }
    return 0;
}
```

6. 调试过程和数据测试

程序运行后，输入排序元素个数，然后依次输入要排序的元素，程序显示每次排序的结果如下：



```
C:\Users\胡萝卜君\Desktop\数据结构实践周源文件\DataStructuresPracticeWeek\4\希尔排序.exe
请输入待排序数组个数:
8
请输入待排序数组:
46
55
13
42
94
17
05
70
第1次排序结果:
46 17 5 42 94 55 13 70
第2次排序结果:
5 17 13 42 46 55 94 70
第3次排序结果:
5 13 17 42 46 55 70 94
-----
Process exited after 33.94 seconds with return value 0
```

7. 调试过程中遇到的主要问题，及解决办法

数组 0 号元素地址作为监视哨使用，用于备份 $r[i]$ 。

8. 总结及体会

解决该问题主要理解教材上的希尔排序算法，以及求出增量数组，求增量数组时注意计数器的使用。

2. 课程设计总结

《数据结构与算法》是大学计算机编程基础课程之一，数据结构和算法是计算机求解问题过程的两大基石，数据结构是计算机科学研究的基本课题，数据结构又是算法研究的基础。

通过这次实践周的程序设计，提高了我们的编程能力，我们不仅学会了根据问题使用适合的数据结构，还学会了使用一些简便且实用的算法。除此之外，我们还学会单步跟踪、调试自己的程序，加深了对线性表特别是链表知识的理解和掌握，并能够运用相关知识来解决一些具体的问题，如一元多项式相加等。这让我们进一步理解了线性表的逻辑结构和存储结构，进一步提高使用理论知识指导解决实际问题的能力。让我们通过实践来更加深入的掌握了哈夫曼编码原理，以及其生成方法，理解数据编码压缩和译码输出编码的实现。本周的实

训也让我更加深刻的理解到很多东西需要自己实际操作一遍才能更好的去理解期原理,通过实践更能学习透彻理论知识。本周的实训真的让我们获益匪浅,每完成一个程序编写,我自己也是成就感满满的!