```c
//>>>>>>>>>>>>>>>>>>>>>>>>>>引入头文件<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#include <stdio.h>        //使用了标准库函数
#include <stdlib.h>       //使用了动态内存分配函数
#include <string.h>       //使用了字符串处理函数strcpy

//>>>>>>>>>>>>>>>>>>>>>>>>自定义符号常量<<<<<<<<<<<<<<<<<<<<<<<<<<

#define OVERFLOW -2           //内存溢出错误常量
#define OK 1                  //表示操作正确的常量
#define ERROR 0               //表示操作错误的常量

//>>>>>>>>>>>>>>>>>>>>>>>>自定义数据类型<<<<<<<<<<<<<<<<<<<<<<<<<<

typedef int  Status;       //用typedef给int起个别名，也便于程序的维护

//----------------赫夫曼树和赫夫曼编码的存储表示-----------------
typedef struct{
    unsigned int weight;                    //权值
    unsigned int parent, lchild, rchild;   //双亲、左孩子，右孩子
}HTNode, * HuffmanTree;              //动态分配数组存储赫夫曼树

typedef char * * HuffmanCode;      //动态分配数组存储赫夫曼编码表


//作用：从HT指示的赫夫曼树的[1..i-1]范围的结点中选出权值最小的结点。
int min(HuffmanTree HT, int i){
    //j是临时变量，flag记录了权值最小的结点在HT中的下标
    int j, flag = 1;
    //设置k的值为一个结点的权值极难超越的最大值
    unsigned int k = 10000;

    //从赫夫曼树的所有结点中选出权值最小的结点并记录其在HT中的下标
    for(j = 1; j <= i; j++)
        if(HT[j].weight < k && HT[j].parent == 0){
            k = HT[j].weight, flag = j;}

    //循环结束后，k保存了赫夫曼树所有备选结点中权值的最小值
    //flag保存了权值最小结点在HT中的下标
    //此时被选中的权值最小的结点就是HT[flag]
    //修改被选中结点的双亲，防止这个结点再次被选中
    HT[flag].parent = 1;

    //返回被选中的权值最小的结点在HT中的下标
    return flag;
}//min

//作用：在HT[1..i-1]选择parent为0且weight最小的两个结点s1,s2，其中s1<s2
void Select(HuffmanTree HT, int i, int &s1, int &s2){  //选择函数
    int j;      //临时变量j
    //找出第一、二个权值最小结点在HT中的下标s1、s2
    s1 = min(HT, i);
    s2 = min(HT, i);
    //若s1和s2位置不对，则对调一下
    if(s1 > s2){
        j = s1; s1 = s2; s2 = j;}
}//Select


//作用：按照书上P149页格式打印HT的存储结构表
void PrintHT(HuffmanTree HT, int n){

    //n个字符构造的赫夫曼树共有2*n-1个结点
    int m = 2 * n - 1;
    printf("\n+----------------------------------------------------------------------+\n");
    printf("|                        赫夫曼树HT的存储结构                          |\n");
    printf("+----------+----------+----------+--------------+-------------+\n");
    printf("| 结点编号 |  weight  |  parent  |  leftchild   |  rightchild |\n");
    printf("+----------+----------+----------+--------------+-------------+\n");
    for(int i = 1; i <= m; i++){
        printf("|    %4d    |    %4d    |    %4d    |      %4d      |     %4d     |\n",
                i, HT[i].weight, HT[i].parent, HT[i].lchild, HT[i].rchild);
        printf("+----------+----------+----------+--------------+-------------+\n");
    }//for
}//PrintHT


//作用：w存放n个字符的权值（均>0），构造赫夫曼树 HT，并求出n个字符的赫夫曼编码 HC
Status HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, int *w, int n){

    //s1和s2保存选出的两个权值最小的结点在HT中的下标
    //i是临时变量，循环用
    //start指示了赫夫曼编码结束符在cd指示的求编码的临时空间中的位置
    //f是s1和s2的双亲在HT中的下标
```

```
85      int s1, s2, i, start, f;
86      char *cd;   //cd指向动态分配的求编码的临时空间
87
88      HuffmanTree p = NULL;       //p是工作指针，指向赫夫曼树中的结点
89
90      //检查权值个数参数n是否合法
91      if(n <= 1)
92          return ERROR;
93
94      //n个字符构造的赫夫曼树共有m = 2*n-1个结点
95      int m = 2 * n - 1;
96      //申请赫夫曼树结点占用的内存空间,0号单元不用
97      if(!(HT = (HuffmanTree)malloc((m + 1) * sizeof(HTNode))))
98          exit(OVERFLOW);
99
100     //初始化HT内部各个分量的值
101     for(p = HT + 1, i = 1; i <= n; ++i, ++p, ++w)
102         *p = {*w,0,0,0};
103
104     for(; i <= m; i++, p++)
105         *p = {0,0,0,0};
106
107     //开始构造赫夫曼树
108     for(i = n + 1; i <= m; ++i){
109         //在HT[1..i-1]选择parent为0且weight最小的两个结点，其序号分别为是s1和s2
110         Select(HT, i - 1, s1, s2);
111         HT[s1].parent = i;
112         HT[s2].parent = i;
113         HT[i].lchild = s1;
114         HT[i].rchild = s2;
115
116         HT[i].weight = HT[s1].weight + HT[s2].weight;
117     }//for
118
119     //申请赫夫曼编码占用的内存空间
120     if(!(HC = (HuffmanCode)malloc((n + 1) * sizeof(char *))))
121         exit(OVERFLOW);
122
123     //申请求编码的工作空间
124     if(!(cd = (char *)malloc(n * sizeof(char))))
125         exit(OVERFLOW);
126     //编码结束符—字符串结束标志\0
127     cd[n-1] = '\0';
128
129     //逐个字符求赫夫曼编码
130     for(int i = 1; i <= n; ++i){
131         start = n - 1;  //编码结束符位置
132         //从叶子到根逆向求编码
133         for(int c = i, f = HT[i].parent; f != 0; c = f, f = HT[f].parent) {
134             if(HT[f].lchild == c)       //叶子结点根结点的左孩子
135                 cd[--start] = '0';
136             else                        //叶子结点根结点的右孩子
137                 cd[--start] = '1';
138         }//for
139
140         //申请一段内存空间用于保存生成的赫夫曼编码
141         if(!(HC[i] = (char *)malloc((n - start) * sizeof(char))))
142             exit(OVERFLOW);
143
144         //将临时工作空间cd中拷贝计算出的赫夫曼编码到HC对应的位置上
145         strcpy(HC[i], &cd[start]);
146     }
147     free(cd);   //释放临时工作空间cd
148     return OK;
149 }//HuffmanCoding
150
151 //----------------------主函数----------------------
152 int main() {
153
154     //指向赫夫曼树的指针
155     HuffmanTree HT;
156
157     //指向存储赫夫曼编码存储区域的指针
158     HuffmanCode HC;
159
160     //n是输入权值的个数
161     //w是指向权值存储区域的指针
162     int *w, n;
163     printf("->请输入所要建立赫夫曼树权值的个数(>1): ");
164     scanf("%d", &n);
165
166     //根据输入权值的个数申请保存权值的存储空间
167     if(!(w =  (int *)malloc(n * sizeof(int)))) {
168         exit(OVERFLOW);
```

```
169         }//if
170
171         //从键盘接收n个权值并保存到w指示的内存区域中
172         printf("->请依次输入%d个权值（整型，用空格隔开）：\n",n);
173         for(int i = 0; i <= n - 1; i++) {
174             scanf("%d", w + i);
175         }//for
176         for(int t=0; t<n; t++)
177             printf("%d\t",*(w+t));
178
179         //调用算法得出赫夫曼编码
180         HuffmanCoding(HT, HC, w, n);
181
182         PrintHT(HT, n);
183
184         //输出赫夫曼编码
185         printf("->您建立的赫夫曼树对应的赫夫曼编码如下：\n");
186         for(int i = 1; i <= n; i++) {
187             puts(HC[i]);
188         }//for
189
190         //释放掉存放权值的内存空间
191         free(w);
192         w = NULL;
193
194         //释放掉赫夫曼树HT占用的内存空间
195         free(HT);
196         HT = NULL;
197
198         //释放赫夫曼编码HC占用的内存空间
199         free(HC);
200         HC = NULL;
201
202         return 0;
203     }//main
204
```