```
int main(){
     char* c1;
     c1=(char*)malloc(5*sizeof(char));
     char** c2;
     c2=(char**)malloc(5*sizeof(char*));
     c1[2]='1';
     c1[3]='0';
     c1[4]='1';
     c2[0]=(char*)malloc(3*sizeof(char));
     strcpy(c2[0],&c1[2]);

      printf("%s",c2[0]);

}



#include<stdlib.h>
#include<stdio.h>
#include<string.h>
//哈夫曼树结点结构
typedef struct {
    int weight;//结点权重
    int parent, left, right;//父结点、左孩子、右孩子在数组中的位置下标
}HTNode, *HuffmanTree;
//动态二维数组，存储哈夫曼编码
typedef char ** HuffmanCode;

//HT数组中存放的哈夫曼树，end表示HT数组中存放结点的最终位置，s1和s2传递的是HT
数组中权重值最小的两个结点在数组中的位置
void Select(HuffmanTree HT, int end, int *s1, int *s2)
{
    int min1, min2;
    //遍历数组初始下标为 1
    int i = 1;
    //找到还没构建树的结点
    while(HT[i].parent != 0 && i <= end){
        i++;
    }
    min1 = HT[i].weight;
    *s1 = i;

    i++;
    while(HT[i].parent != 0 && i <= end){
        i++;
    }
```

```
    //对找到的两个结点比较大小，min2为大的，min1为小的
    if(HT[i].weight < min1){
        min2 = min1;
        *s2 = *s1;
        min1 = HT[i].weight;
        *s1 = i;
    }else{
        min2 = HT[i].weight;
        *s2 = i;
    }
    //两个结点和后续的所有未构建成树的结点做比较
    for(int j=i+1; j <= end; j++)
    {
        //如果有父结点，直接跳过，进行下一个
        if(HT[j].parent != 0){
            continue;
        }
        //如果比最小的还小，将min2=min1，min1赋值新的结点的下标
        if(HT[j].weight < min1){
            min2 = min1;
            min1 = HT[j].weight;
            *s2 = *s1;
            *s1 = j;
        }
        //如果介于两者之间，min2赋值为新的结点的位置下标
        else if(HT[j].weight >= min1 && HT[j].weight < min2){
            min2 = HT[j].weight;
            *s2 = j;
        }
    }
}

//HT为地址传递的存储哈夫曼树的数组，w为存储结点权重值的数组，n为结点个数
void CreateHuffmanTree(HuffmanTree *HT, int *w, int n)
{
    if(n<=1) return; // 如果只有一个编码就相当于0
    int m = 2*n-1; // 哈夫曼树总节点数，n就是叶子结点
    *HT = (HuffmanTree) malloc((m+1) * sizeof(HTNode)); // 0号位置不用
    HuffmanTree p = *HT;
    // 初始化哈夫曼树中的所有结点
    for(int i = 1; i <= n; i++)
    {
        (p+i)->weight = *(w+i-1);
        (p+i)->parent = 0;
        (p+i)->left = 0;
        (p+i)->right = 0;
    }
```

```
    //从树组的下标 n+1 开始初始化哈夫曼树中除叶子结点外的结点
    for(int i = n+1; i <= m; i++)
    {
        (p+i)->weight = 0;
        (p+i)->parent = 0;
        (p+i)->left = 0;
        (p+i)->right = 0;
    }
    //构建哈夫曼树
    for(int i = n+1; i <= m; i++)
    {
        int s1, s2;
        Select(*HT, i-1, &s1, &s2);
        (*HT)[s1].parent = (*HT)[s2].parent = i;
        (*HT)[i].left = s1;
        (*HT)[i].right = s2;
        (*HT)[i].weight = (*HT)[s1].weight + (*HT)[s2].weight;
    }
}
//HT为哈夫曼树，HC为存储结点哈夫曼编码的二维动态数组，n为结点的个数
void HuffmanCoding(HuffmanTree HT, HuffmanCode *HC,int n){
    *HC = (HuffmanCode) malloc((n+1) * sizeof(char *));
    char *cd = (char *)malloc(n*sizeof(char)); //存放结点哈夫曼编码的字符串数组
    cd[n-1] = '\0';//字符串结束符

    for(int i=1; i<=n; i++){
        //从叶子结点出发，得到的哈夫曼编码是逆序的，需要在字符串数组中逆序存放
        int start = n-1;
        //当前结点在数组中的位置
        int c = i;
        //当前结点的父结点在数组中的位置
        int j = HT[i].parent;
        // 一直寻找到根结点
        while(j != 0){
            // 如果该结点是父结点的左孩子则对应路径编码为0，否则为右孩子编码为1
            if(HT[j].left == c)
                cd[--start] = '0';
            else
                cd[--start] = '1';
            //以父结点为孩子结点，继续朝树根的方向遍历
            c = j;
            j = HT[j].parent;
        }
        //跳出循环后，cd数组中从下标 start 开始，存放的就是该结点的哈夫曼编码
        (*HC)[i] = (char *)malloc((n-start)*sizeof(char));
        strcpy((*HC)[i], &cd[start]);
    }
```

```
    //使用malloc申请的cd动态数组需要手动释放
    free(cd);
}
//打印哈夫曼编码的函数
void PrintHuffmanCode(HuffmanCode htable,int *w,int n)
{
    printf("Huffman code : \n");
    for(int i = 1; i <= n; i++)
        printf("%d code = %s\n",w[i-1], htable[i]);
}
int main(void)
{
    int w[5] = {2, 8, 7, 6, 5};
    int n = 5;
    HuffmanTree htree;
    HuffmanCode htable;
    CreateHuffmanTree(&htree, w, n);
    HuffmanCoding(htree, &htable, n);
    PrintHuffmanCode(htable,w, n);
    return 0;
}
```