```
/*
哈夫曼树与哈夫曼编码

带权路径长度
    设二叉树有 n 个叶子结点，每个叶子结点带有权值 wk
    从根结点到每个叶子结点的长度为 lk
    所有叶子结点的 wklk 之和记为 WPL（带权路径长度）

哈夫曼树（最优二叉树）
    WPL最小的二叉树

哈夫曼树的构造
    每次把权值最小的两棵二叉树合并

哈夫曼树的特点
    没有度为 1 的结点
    n 个叶子结点的哈夫曼树共有 2n-1 个结点
    哈夫曼树的任意非叶结点左右子树交换之后仍是哈夫曼树
    同一组权值可能存在不同构的哈夫曼树

哈夫曼编码
    给定一段字符串，对字符进行编码，使该字符串的编码存储空间最少
    使用前缀码避免二义性，任何字符的编码都不是另一字符编码的前缀
*/

#include<iostream>
#include<malloc.h>
#define MaxSize 1000
#define MinData -1000
using namespace std;

int A[] = {13, 1, 45, 7, 20, 4, 19, 13, 40, 33, 38}; // 预先定义好一组权值
int A_length = 11; // 定义其长度

typedef struct TreeNode *HuffmanTree;
struct TreeNode{        // 哈夫曼树
    int weight;         // 权值
    HuffmanTree Left;   // 左子树
    HuffmanTree right; // 右子树
};

typedef struct HeapStruct *MinHeap;
struct HeapStruct{      // 使用最小堆存放哈夫曼树
    HuffmanTree *data; // 存值的数组
    int size;           // 堆的当前大小
    int capacity;       // 最大容量
};

MinHeap create(); // 初始化堆
HuffmanTree Create(); // 初始化哈夫曼树
void sort(MinHeap H, int i); // 调整子最小堆
void adjust(MinHeap H); // 调整最小堆
void BuildMinHeap(MinHeap H); // 建堆
HuffmanTree Delete(MinHeap H); // 删除最小堆元素
void Insert(MinHeap H,HuffmanTree Huff); // 插入最小堆元素
void PreOrderTraversal(HuffmanTree Huff); // 先序遍历
HuffmanTree Huffman(MinHeap H); // 哈夫曼树的构建

// 初始化哈夫曼树
HuffmanTree Create()
{
    HuffmanTree Huff;
    Huff = (HuffmanTree)malloc(sizeof(struct TreeNode));
    Huff->weight = 0;
    Huff->Left = NULL;
    Huff->right = NULL;
    return Huff;
}

// 初始化堆
MinHeap create()
{
    MinHeap H;
```

```
    HuffmanTree Huff;
    H = (MinHeap)malloc(sizeof(struct HeapStruct));
    H->data = (HuffmanTree *)malloc((MaxSize+1)*sizeof(struct TreeNode));
    H->capacity = MaxSize;
    H->size = 0;
    // 给堆置哨兵
    Huff = Create();
    Huff->weight = MinData;
    H->data[0] = Huff;
    return H;
}

// 调整子最小堆
void sort(MinHeap H, int i)
{
    int parent, child;
    int tmp = H->data[i]->weight; // 取出当前"根结点"值
    for(parent=i; parent*2<=H->size; parent=child)
    {
        child = 2*parent;
        if((child!=H->size) && (H->data[child+1]->weight<H->data[child]->weight))
            child++;
        if(tmp <= H->data[child]->weight)
            break;
        else
            H->data[parent] = H->data[child];
    }
    H->data[parent]->weight = tmp;
}

// 调整最小堆
void adjust(MinHeap H)
{
    for(int i=H->size/2; i>0; i--)
        sort(H, i); // 每个"子最小堆"调整
}

// 建堆
void BuildMinHeap(MinHeap H)
{
    // 将权值读入堆中
    HuffmanTree Huff;
    for(int i=0; i<A_length; i++)
    {
        Huff = Create();
        Huff->weight = A[i];
        H->data[++H->size] = Huff;
    }
    // 调整堆
    adjust(H);
}

// 删除最小堆元素
HuffmanTree Delete(MinHeap H)
{
    int parent, child;
    HuffmanTree T = H->data[1]; // 取出根结点的哈夫曼树
    HuffmanTree tmp = H->data[H->size--]; // 取出最后一个结点哈夫曼树的权值
    for(parent=1; parent*2<=H->size; parent=child)
    {
        child = 2*parent;
        if((child!=H->size) && (H->data[child+1]->weight<H->data[child]->weight))
            child++;
        if(tmp->weight <= H->data[child]->weight)
            break;
        else
            H->data[parent] = H->data[child];
    }
    H->data[parent] = tmp;
    // 构造一个 HuffmanTree 结点 T，附上刚才取出来的权值，返回该结点
    return T;
}

```

```
147  // 插入一个哈夫曼树
148  void Insert(MinHeap H,HuffmanTree Huff)
149  {
150      int weight = Huff->weight; // 取出权值
151      int i = ++H->size;
152      for(; H->data[i/2]->weight > weight; i/=2)
153          H->data[i] = H->data[i/2];
154      H->data[i] = Huff;
155  }
156
157  //遍历
158  void PreOrderTraversal(HuffmanTree Huff)
159  {
160      if(Huff)
161      {
162          cout<<Huff->weight<<" ";
163          PreOrderTraversal(Huff->Left);
164          PreOrderTraversal(Huff->right);
165      }
166  }
167
168  // 哈夫曼树的构造，时间复杂度O(nlogn)
169  HuffmanTree Huffman(MinHeap H)
170  {
171      HuffmanTree T;
172      BuildMinHeap(H); // 建堆，堆中存放了所有二叉树结点
173      int times = H->size;
174      // 做 times-1 次合并
175      for(int i=1; i<times; i++)
176      {
177          T = (HuffmanTree)malloc(sizeof(struct TreeNode));
178          T->Left = Delete(H);  // 从堆中删除一个结点，作为新 T 的左子结点
179          T->right = Delete(H); // 从堆中删除一个结点，作为新 T 的右子结点
180          T->weight = T->Left->weight + T->right->weight; // 重新计算权值
181          Insert(H, T); // 再加进堆中
182      }
183      T = Delete(H);
184      return T;
185  }
186
187  int main()
188  {
189      MinHeap H;
190      HuffmanTree Huff;
191      H = create();
192      Huff = Huffman(H);
193      PreOrderTraversal(Huff);
194      return 0;
195  }
196
```