```
typedef struct HNode *Heap; /* 堆的类型定义 */
struct HNode {
    ElementType *Data; /* 存储元素的数组 */
    int Size;          /* 堆中当前元素个数 */
    int Capacity;      /* 堆的最大容量 */
};
typedef Heap MaxHeap; /* 最大堆 */
typedef Heap MinHeap; /* 最小堆 */

#define MAXDATA 1000  /* 该值应根据具体情况定义为大于堆中所有可能元素的值 */

MaxHeap CreateHeap( int MaxSize )
{ /* 创建容量为MaxSize的空的最大堆 */

    MaxHeap H = (MaxHeap)malloc(sizeof(struct HNode));
    H->Data = (ElementType *)malloc((MaxSize+1)*sizeof(ElementType));
    H->Size = 0;
    H->Capacity = MaxSize;
    H->Data[0] = MAXDATA; /* 定义"哨兵"为大于堆中所有可能元素的值*/

    return H;
}

bool IsFull( MaxHeap H )
{
    return (H->Size == H->Capacity);
}

bool Insert( MaxHeap H, ElementType X )
{ /* 将元素X插入最大堆H，其中H->Data[0]已经定义为哨兵 */
    int i;

    if ( IsFull(H) ) {
        printf("最大堆已满");
        return false;
    }
    i = ++H->Size; /* i指向插入后堆中的最后一个元素的位置 */
    for ( ; H->Data[i/2] < X; i/=2 )
        H->Data[i] = H->Data[i/2]; /* 上滤X */
    H->Data[i] = X; /* 将X插入 */
    return true;
}

#define ERROR -1 /* 错误标识应根据具体情况定义为堆中不可能出现的元素值 */

bool IsEmpty( MaxHeap H )
{
    return (H->Size == 0);
}

ElementType DeleteMax( MaxHeap H )
{ /* 从最大堆H中取出键值为最大的元素，并删除一个结点 */
    int Parent, Child;
    ElementType MaxItem, X;

    if ( IsEmpty(H) ) {
        printf("最大堆已为空");
        return ERROR;
    }

    MaxItem = H->Data[1]; /* 取出根结点存放的最大值 */
    /* 用最大堆中最后一个元素从根结点开始向上过滤下层结点 */
    X = H->Data[H->Size--]; /* 注意当前堆的规模要减小 */
    for( Parent=1; Parent*2<=H->Size; Parent=Child ) {
        Child = Parent * 2;
        if( (Child!=H->Size) && (H->Data[Child]<H->Data[Child+1]) )
            Child++;  /* Child指向左右子结点的较大者 */
        if( X >= H->Data[Child] ) break; /* 找到了合适位置 */
        else  /* 下滤X */
            H->Data[Parent] = H->Data[Child];
    }
    H->Data[Parent] = X;

    return MaxItem;
}

/*------------ 建造最大堆 ------------*/
void PercDown( MaxHeap H, int p )
{ /* 下滤：将H中以H->Data[p]为根的子堆调整为最大堆 */
    int Parent, Child;
    ElementType X;
```

```
    X = H->Data[p]; /* 取出根结点存放的值 */
    for( Parent=p; Parent*2<=H->Size; Parent=Child ) {
        Child = Parent * 2;
        if( (Child!=H->Size) && (H->Data[Child]<H->Data[Child+1]) )
            Child++;  /* Child指向左右子结点的较大者 */
        if( X >= H->Data[Child] ) break; /* 找到了合适位置 */
        else  /* 下滤X */
            H->Data[Parent] = H->Data[Child];
    }
    H->Data[Parent] = X;
}

void BuildHeap( MaxHeap H )
{ /* 调整H->Data[]中的元素，使满足最大堆的有序性  */
  /* 这里假设所有H->Size个元素已经存在H->Data[]中 */

    int i;

    /* 从最后一个结点的父节点开始，到根结点1 */
    for( i = H->Size/2; i>0; i-- )
        PercDown( H, i );
}
```