```c
#include<stdio.h>
#include<stdlib.h>

//Declaration of heap
struct heap{
    int *array;
    int count;
    int capacity;
    int heap_type;
};

//Creating heap
struct heap *CreateHeap(int capacity,int heap_type){
    struct heap* h=(struct heap*)malloc(sizeof(struct heap));
    if(h==NULL)
       printf("Memory error");
       return;

    h->heap_type=heap_type;
    h->count=0;
    h->capacity=capacity;
    h->array=(int*)malloc(sizeof(int)* h->capacity);
    if(h->array==NULL){
        printf("Memory error");
        return;
   }
  return h;
}

//Parent of node
int parent(struct heap* h,int i){
    if(i<=0||i>=h->count)
       return -1;
    return (i-1)/2;
}

//Children of a node
int leftchild(struct heap* h,int i){
    int left = 2*i+1;
    if(left>=h->count)
       return -1;
    return left;
}

int rightchild(struct heap* h,int i){
    int right = 2*i+2;
    if(right>=h->count)
       return -1;
    return left;
}

//Heapifying an element at location i
void percolateDown(struct heap* h,int i){
    int l,r,max,temp;
    l=leftchild(h,i);
    r=rightchild(h,i);
    if(l!=-1 && h->array[l]>h->array[i])
        max=l;
    else
        max=i;
    if(r!=-1 && h->array[l]>h->array[max])
        max=r;
    if(max!=i){
        temp=h->array[i];
        h->array[i]=h->array[max];
        h->array[max]=temp;
    }
    percolateDown(h,max);
}

//Deleting an element
int deletemax(struct heap* h){
    int data;
    if(h->count==0)
       return -1;
    data=h->array[0];
    //Replacing root with last element
    h->array[0]=h->array[count-1];
    //Reducing the heap size
    h->count--;
    percolateDown(h,0);
    return data;
}

//Inserting an element
//• Increase the heap size to hold new item
//• Keep the new element at the end of the heap (tree)
//• Heapify the element from bottom to top (root) i.e heapi

int insert(struct heap* h,int i){
    int i;
    if(h->count==h->capacity)
      resizeheap(h);
    h->count++;
    i=h->count-1;
    while(i>=0 && data>h->array[(i-1)/2]){
        h->array[i]=h->array[(i-1)/2];
        i=(i-1)/2;
    }
    h->array[i]=data;
}

void resizeheap(struct heap* h){
    int* array_old=h->array;
    h->array=(int*)malloc(sizeof(int)* h->capacity*2);
    if(h->array==NULL)
      printf("Memory error");
      rteurn;
    for(int i=0;i<h->capacity,i++){
        h->array[i]=array_old;
    h->capacity*=2;
    free(array_old);
    }
}

//Destroying heap
void destroyheap(struct heap* h){
```

```c
    if(h==NULL)
      return;
    free(h->array);
    free(h);
    h=NULL;
}

//Build an entire heap from a given list of keys
//(Heapify an array)
void buildsize(struct heap* h,int A[],int n){
    if(h==NULL)
      return;

    while(n>h->capacity)
      resizeheap(h);
    //Inserting one by one
    for(int i=0;i<n;i++){
        h->array[i]=A[i];
    h->count=n;
    for(int i=(n-1)/2;i>=0;i++)
      percolateDown(h,i);
    }
}

//Do heapsort algo in algo analysis part(Appli of heap ADT)

//Find max element in a min heap

int findmaxinminheap(struct heap* h){
    int max=-1;
    for(i=(h->count+1)/2;i<h->count;i++)
       if(h->array[i]>max)
         max=h->array[i];

}

//Deleting the ith indexed element in minheap
int Delete(struct heap* h,int i){
    int key;
    if(n<i){
      printf("Wrong position");
      return;}
    key=h->array[i];
    h->array[i]=h->array[h->count-1];
    h->count--;
    percolateDown(h,i);
    return key;
}
```