

```

1: #include<stdio.h>
2: #include<stdlib.h>
3:
4: //Declaration of heap
5: struct heap{
6:     int *array;
7:     int count;
8:     int capacity;
9:     int heap_type;
10: };
11:
12: //Creating heap
13: struct heap *CreateHeap(int capacity,int heap_type){
14:     struct heap* h=(struct heap*)malloc(sizeof(struct heap));
15:     if(h==NULL)
16:         printf("Memory error");
17:     return;
18:
19:     h->heap_type=heap_type;
20:     h->count=0;
21:     h->capacity=capacity;
22:     h->array=(int*)malloc(sizeof(int)* h->capacity);
23:     if(h->array==NULL){
24:         printf("Memory error");
25:         return;
26:     }
27:     return h;
28: }
29:
30: //Parent of node
31: int parent(struct heap* h,int i){
32:     if(i<=0||i>=h->count)
33:         return -1;
34:     return (i-1)/2;
35: }
36:
37: //Children of a node
38: int leftchild(struct heap* h,int i){
39:     int left = 2*i+1;

```

```

40:     if(left>=h->count)
41:         return -1;
42:     return left;
43: }
44:
45: int rightchild(struct heap* h,int i){
46:     int right = 2*i+2;
47:     if(right>=h->count)
48:         return -1;
49:     return left;
50: }
51:
52: //Heapifying an element at location i
53: void percolateDown(struct heap* h,int i){
54:     int l,r,max,temp;
55:     l=leftchild(h,i);
56:     r=rightchild(h,i);
57:     if(l!=-1 && h->array[l]>h->array[i])
58:         max=l;
59:     else
60:         max=i;
61:     if(r!=-1 && h->array[r]>h->array[max])
62:         max=r;
63:     if(max!=i){
64:         temp=h->array[i];
65:         h->array[i]=h->array[max];
66:         h->array[max]=temp;
67:     }
68:     percolateDown(h,max);
69: }
70:
71: //Deleting an element
72: int deletemax(struct heap* h){
73:     int data;
74:     if(h->count==0)
75:         return -1;
76:     data=h->array[0];
77:     //Replacing root with last element
78:     h->array[0]=h->array[count-1];

```

```

79:      //Reducing the heap size
80:      h->count--;
81:      percolateDown(h,0);
82:      return data;
83:  }
84:
85:  //Inserting an element
86:  //• Increase the heap size to hold new item
87:  //• Keep the new element at the end of the heap (tree)
88:  //• Heapify the element from bottom to top (root) i.e heapi
89:
90:  int insert(struct heap* h,int i){
91:      int i;
92:      if(h->count==h->capacity)
93:          resizeheap(h);
94:      h->count++;
95:      i=h->count-1;
96:      while(i>=0 && data>h->array[(i-1)/2]){
97:          h->array[i]=h->array[(i-1)/2];
98:          i=(i-1)/2;
99:      }
100:      h->array[i]=data;
101:  }
102:
103:  void resizeheap(struct heap* h){
104:      int* array_old=h->array;
105:      h->array=(int*)malloc(sizeof(int)* h->capacity*2);
106:      if(h->array==NULL)
107:          printf("Memory error");
108:      return;
109:      for(int i=0;i<h->capacity,i++){
110:          h->array[i]=array_old;
111:          h->capacity*=2;
112:          free(array_old);
113:      }
114:  }
115:
116:  //Destroying heap
117:  void destroyheap(struct heap* h){

```

```

118:     if(h==NULL)
119:         return;
120:     free(h->array);
121:     free(h);
122:     h=NULL;
123: }
124:
125: //Build an entire heap from a given list of keys
126:  //(Heapify an array)
127: void buildsize(struct heap* h,int A[],int n){
128:     if(h==NULL)
129:         return;
130:
131:     while(n>h->capacity)
132:         resizeheap(h);
133:     //Inserting one by one
134:     for(int i=0;i<n;i++){
135:         h->array[i]=A[i];
136:         h->count=n;
137:         for(int i=(n-1)/2;i>=0;i++)
138:             percolateDown(h,i);
139:     }
140: }
141:
142: //Do heapsort algo in algo analysis part(Appli of heap ADT)
143:
144: //Find max element in a min heap
145:
146: int findmaxinminheap(struct heap* h){
147:     int max=-1;
148:     for(i=(h->count+1)/2;i<h->count;i++)
149:         if(h->array[i]>max)
150:             max=h->array[i];
151:
152: }
153:
154: //Deleting the ith indexed element in minheap
155: int Delete(struct heap* h,int i){
156:     int key;

```

```
157:     if(n<i){
158:         printf("Wrong position");
159:         return;}
160:     key=h->array[i];
161:     h->array[i]=h->array[h->count-1];
162:     h->count--;
163:     percolateDown(h,i);
164:     return key;
165: }
166:
167:
168:
169:
170:
171:
```