```cpp
// A C++ program to demonstrate common Binary Heap Operatio
#include<iostream>
#include<climits>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int heap_size; // Current number of elements in min heap
public:
    // Constructor
    MinHeap(int capacity);

    // to heapify a subtree with the root at given index
    void MinHeapify(int );

    int parent(int i) { return (i-1)/2; }

    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2*i + 2); }

    // to extract the root which is the minimum element
    int extractMin();

    // Decreases key value of key at index i to new_val
    void decreaseKey(int i, int new_val);

    // Returns the minimum key (key at root) from min heap
    int getMin() { return harr[0]; }

    // Deletes a key stored at index i
```

```
40:     void deleteKey(int i);
41:
42:     // Inserts a new key 'k'
43:     void insertKey(int k);
44: };
45:
46: // Constructor: Builds a heap from a given array a[] of giv
47: MinHeap::MinHeap(int cap)
48: {
49:     heap_size = 0;
50:     capacity = cap;
51:     harr = new int[cap];
52: }
53:
54: // Inserts a new key 'k'
55: void MinHeap::insertKey(int k)
56: {
57:     if (heap_size == capacity)
58:     {
59:         cout << "\nOverflow: Could not insertKey\n";
60:         return;
61:     }
62:
63:     // First insert the new key at the end
64:     heap_size++;
65:     int i = heap_size - 1;
66:     harr[i] = k;
67:
68:     // Fix the min heap property if it is violated
69:     while (i != 0 && harr[parent(i)] > harr[i])
70:     {
71:     swap(&harr[i], &harr[parent(i)]);
72:     i = parent(i);
73:     }
74: }
75:
76: // Decreases value of key at index 'i' to new_val. It is as
77: // new_val is smaller than harr[i].
78: void MinHeap::decreaseKey(int i, int new_val)
```

```
 79: {
 80:     harr[i] = new_val;
 81:     while (i != 0 && harr[parent(i)] > harr[i])
 82:     {
 83:     swap(&harr[i], &harr[parent(i)]);
 84:     i = parent(i);
 85:     }
 86: }
 87:
 88: // Method to remove minimum element (or root) from min heap
 89: int MinHeap::extractMin()
 90: {
 91:     if (heap_size <= 0)
 92:         return INT_MAX;
 93:     if (heap_size == 1)
 94:     {
 95:         heap_size--;
 96:         return harr[0];
 97:     }
 98:
 99:     // Store the minimum value, and remove it from heap
100:     int root = harr[0];
101:     harr[0] = harr[heap_size-1];
102:     heap_size--;
103:     MinHeapify(0);
104:
105:     return root;
106: }
107:
108:
109: // This function deletes key at index i. It first reduced v
110: // infinite, then calls extractMin()
111: void MinHeap::deleteKey(int i)
112: {
113:     decreaseKey(i, INT_MIN);
114:     extractMin();
115: }
116:
117: // A recursive method to heapify a subtree with the root at
```

```
118: // This method assumes that the subtrees are already heapif
119: void MinHeap::MinHeapify(int i)
120: {
121:     int l = left(i);
122:     int r = right(i);
123:     int smallest = i;
124:     if (l < heap_size && harr[l] < harr[i])
125:         smallest = l;
126:     if (r < heap_size && harr[r] < harr[smallest])
127:         smallest = r;
128:     if (smallest != i)
129:     {
130:         swap(&harr[i], &harr[smallest]);
131:         MinHeapify(smallest);
132:     }
133: }
134:
135: // A utility function to swap two elements
136: void swap(int *x, int *y)
137: {
138:     int temp = *x;
139:     *x = *y;
140:     *y = temp;
141: }
142:
143: // Driver program to test above functions
144: int main()
145: {
146:     MinHeap h(11);
147:     h.insertKey(3);
148:     h.insertKey(2);
149:     h.deleteKey(1);
150:     h.insertKey(15);
151:     h.insertKey(5);
152:     h.insertKey(4);
153:     h.insertKey(45);
154:     cout << h.extractMin() << " ";
155:     cout << h.getMin() << " ";
156:     h.decreaseKey(2, 1);
```

```
157:     cout << h.getMin();
158:     return 0;
159: }
160:
```