```c
// C program for building Heap from Array

#include <stdio.h>

// To heapify a subtree rooted with node i which is
// an index in arr[]. N is size of heap
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

void heapify(int arr[], int N, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < N && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < N && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        swap(&arr[i], &arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, N, largest);
    }
}

// Function to build a Max-Heap from the given array
void buildHeap(int arr[], int N)
{
```

```
40:     // Index of last non-leaf node
41:     int startIdx = (N / 2) - 1;
42:
43:     // Perform reverse level order traversal
44:     // from last non-leaf node and heapify
45:     // each node
46:     for (int i = startIdx; i >= 0; i--) {
47:         heapify(arr, N, i);
48:     }
49: }
50:
51: // A utility function to print the array
52: // representation of Heap
53: void printHeap(int arr[], int N)
54: {
55:     printf("Array representation of Heap is:\n");
56:
57:     for (int i = 0; i < N; ++i)
58:         printf("%d ",arr[i]);
59:     printf("\n");
60: }
61:
62: // Driver's Code
63: int main()
64: {
65:     // Binary Tree Representation
66:     // of input array
67:     //            1
68:     //         / \
69:     //         3   5
70:     //    / \     / \
71:     //    4   6 13 10
72:     // / \ / \
73:     // 9 8 15 17
74:     int arr[] = {1, 3, 5, 4, 6, 13, 10, 9, 8, 15, 17};
75:
76:     int N = sizeof(arr) / sizeof(arr[0]);
77:
78:     // Function call
```

```
79:     buildHeap(arr, N);
80:     printHeap(arr, N);
81:
82:     // Final Heap:
83:     //           17
84:     //        / \
85:     //        15  13
86:     //        / \     / \
87:     //   9    6 5 10
88:     //   / \ / \
89:     //   4 8 3 1
90:
91:     return 0;
92: }
93:
```