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

typedef struct {
    int *arr;
    int capacity;
    int size;
} MinHeap;

int parent(int i) { return (i - 1) / 2; }
int leftChild(int i) { return (2 * i + 1); }
int rightChild(int i) { return (2 * i + 2); }

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

void minHeapify(MinHeap *heap, int idx) {
    int smallest = idx, left = leftChild(idx), right = rightChild(idx);
    if (left < heap->size && heap->arr[left] < heap->arr[smallest])
smallest = left;
    if (right < heap->size && heap->arr[right] < heap->arr[smallest])
smallest = right;
    if (smallest != idx) {
        swap(&heap->arr[idx], &heap->arr[smallest]);
        minHeapify(heap, smallest);
    }
}

void insertElement(MinHeap *heap, int element) {
    if (heap->size == heap->capacity) return;
    heap->arr[heap->size++] = element;
    for (int i = parent(heap->size - 1); i >= 0; i--) minHeapify(heap, i);
}

int deleteMax(MinHeap *heap) {
    if (heap->size <= 0) return -1;
    int max = heap->arr[0];
```

```
    for (int i = 1; i < heap->size; i++)
        if (heap->arr[i] > max) max = heap->arr[i];
    for (int i = 0; i < heap->size; i++) {
        if (heap->arr[i] == max) {
            heap->arr[i] = heap->arr[--heap->size];
            break;
        }
    }
    for (int i = parent(heap->size - 1); i >= 0; i--) minHeapify(heap, i);
    return max;
}

void buildHeap(MinHeap *heap, int *array, int n) {
    if (n > heap->capacity) return;
    for (int i = 0; i < n; i++) heap->arr[i] = array[i];
    heap->size = n;
    for (int i = parent(heap->size - 1); i >= 0; i--) minHeapify(heap, i);
}

void heapSort(int *array, int n) {
    MinHeap heap;
    heap.arr = (int *)malloc(n * sizeof(int));
    heap.capacity = n;
    heap.size = 0;
    buildHeap(&heap, array, n);
    for (int i = 0; i < n; i++) array[i] = deleteMax(&heap);
}

void printTree(int *arr, int size, int idx, int indent) {
    if (idx >= size) return;
    printTree(arr, size, rightChild(idx), indent + 4);
    for (int i = 0; i < indent; i++) printf(" ");
    printf("%d\n", arr[idx]);
    printTree(arr, size, leftChild(idx), indent + 4);
}

int main() {
    int capacity = 100;
    MinHeap heap;
    heap.arr = (int *)malloc(capacity * sizeof(int));
```

```
    heap.capacity = capacity;
    heap.size = 0;

    int arr[] = {1, 5, 6, 8, 9, 7, 3};
    int n = sizeof(arr) / sizeof(arr[0]);

    buildHeap(&heap, arr, n);
    printf("Min Heap:\n");
    printTree(heap.arr, heap.size, 0, 0);

    printf("\nInserting 2...\n");
    insertElement(&heap, 2);
    printTree(heap.arr, heap.size, 0, 0);

    printf("\nDeleting max...\n");
    int max = deleteMax(&heap);
    printf("Deleted max: %d\n", max);
    printTree(heap.arr, heap.size, 0, 0);

    printf("\nHeap Sort:\n");
    int sortArr[] = {1, 5, 6, 8, 9, 7, 3};
    heapSort(sortArr, n);
    for (int i = 0; i < n; i++) printf("%d ", sortArr[i]);
    printf("\n");

    free(heap.arr);
    return 0;
}
```

```
crosoft-MIEngine-Pid-deyzreif.g2n' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Min Heap:
        6
    3
        7
1
        9
    5
        8

Inserting 2...
        6
    3
        7
1
        9
    2
        5
            8

Deleting max...
Deleted max: 9
        6
    3
        7
1
        8
    2
        5

Heap Sort:
9 8 7 6 5 3 1
```

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

typedef struct Task {
    int priority;
    char description[100];
} Task;

typedef struct MaxHeap {
    Task *arr;
    int cap;
    int size;
} MaxHeap;

int parent(int i) { return (i - 1) / 2; }
int left(int i) { return (2 * i + 1); }
```

```c
int right(int i) { return (2 * i + 2); }

void swap(Task *x, Task *y) {
    Task temp = *x;
    *x = *y;
    *y = temp;
}

void heapify(MaxHeap *h, int idx) {
    int largest = idx;
    int l = left(idx);
    int r = right(idx);

    if (l < h->size && h->arr[l].priority > h->arr[largest].priority)
        largest = l;

    if (r < h->size && h->arr[r].priority > h->arr[largest].priority)
        largest = r;

    if (largest != idx) {
        swap(&h->arr[idx], &h->arr[largest]);
        heapify(h, largest);
    }
}

void insert(MaxHeap *h, Task t) {
    if (h->size == h->cap) {
        printf("Priority Queue Overflow\n");
        return;
    }

    h->arr[h->size++] = t;

    for (int i = parent(h->size - 1); i >= 0; i--) {
        heapify(h, i);
    }
}

Task extractMax(MaxHeap *h) {
    if (h->size <= 0) {
```

```
        printf("Priority Queue Underflow\n");
        Task t = {-1, "Empty"};
        return t;
    }

    Task max = h->arr[0];
    h->arr[0] = h->arr[--h->size];

    heapify(h, 0);
    return max;
}

void heapSort(Task *arr, int n) {
    MaxHeap h;
    h.arr = (Task *)malloc(n * sizeof(Task));
    h.cap = n;
    h.size = 0;

    for (int i = 0; i < n; i++) {
        insert(&h, arr[i]);
    }

    for (int i = n - 1; i >= 0; i--) {
        arr[i] = extractMax(&h);
    }

    free(h.arr);
}

void displayQueue(MaxHeap *h) {
    printf("Current tasks in the priority queue:\n");
    for (int i = 0; i < h->size; i++) {
        printf("Task: %s, Priority: %d\n", h->arr[i].description, h-
>arr[i].priority);
    }
}

int main() {
    int cap = 100;
    MaxHeap h;
```

```c
    h.arr = (Task *)malloc(cap * sizeof(Task));
    h.cap = cap;
    h.size = 0;

    Task tasks[] = {
        {3, "Task A"},
        {1, "Task B"},
        {5, "Task C"},
        {4, "Task D"},
        {2, "Task E"}
    };

    int n = sizeof(tasks) / sizeof(tasks[0]);

    for (int i = 0; i < n; i++) {
        insert(&h, tasks[i]);
    }


    printf("\nTasks by priority:\n");
    while (h.size > 0) {
        Task t = extractMax(&h);
        printf("Processing Task: %s with Priority: %d\n", t.description,
t.priority);
    }

    free(h.arr);
    return 0;
}
```

```
Tasks by priority:
Processing Task: Task C with Priority: 5
Processing Task: Task D with Priority: 4
Processing Task: Task A with Priority: 3
Processing Task: Task E with Priority: 2
Processing Task: Task B with Priority: 1
```