Task 3: Implementing Heap Operations

Code a min-heap in C# with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation.

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

#define INITIAL_CAPACITY 10

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

struct MinHeap* createMinHeap() {
```

```
    struct MinHeap* minHeap = (struct
MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->arr = (int*)malloc(INITIAL_CAPACITY *
sizeof(int));
    minHeap->size = 0;
    minHeap->capacity = INITIAL_CAPACITY;
    return 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* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void heapifyUp(struct MinHeap* minHeap, int i) {
    while (i > 0 && minHeap->arr[i] < minHeap-
>arr[parent(i)]) {
        swap(&minHeap->arr[i], &minHeap-
>arr[parent(i)]);
        i = parent(i);
    }
}

void heapifyDown(struct MinHeap* minHeap, int i) {
    int smallest = i;
    int left = leftChild(i);
    int right = rightChild(i);
```

```
    if (left < minHeap->size && minHeap->arr[left] <
minHeap->arr[smallest]) {
        smallest = left;
    }

    if (right < minHeap->size && minHeap->arr[right] <
minHeap->arr[smallest]) {
        smallest = right;
    }

    if (smallest != i) {
        swap(&minHeap->arr[i], &minHeap-
>arr[smallest]);
        heapifyDown(minHeap, smallest);
    }
}

void insert(struct MinHeap* minHeap, int value) {
    if (minHeap->size == minHeap->capacity) {
        minHeap->capacity *= 2;
```

```
        minHeap->arr = realloc(minHeap->arr, minHeap-
>capacity * sizeof(int));
    }
    minHeap->arr[minHeap->size++] = value;
    heapifyUp(minHeap, minHeap->size - 1);
}

int extractMin(struct MinHeap* minHeap) {
    if (minHeap->size == 0) {
        printf("Heap is empty\n");
        return -1; // or some other error indicator
    }
    int min = minHeap->arr[0];
    minHeap->arr[0] = minHeap->arr[minHeap->size - 1];
    minHeap->size--;
    heapifyDown(minHeap, 0);
    return min;
}

int getMin(struct MinHeap* minHeap) {
```

```
    if (minHeap->size == 0) {
        printf("Heap is empty\n");
        return -1; // or some other error indicator
    }
    return minHeap->arr[0];
}

int main() {
    struct MinHeap* minHeap = createMinHeap();
    insert(minHeap, 5);
    insert(minHeap, 3);
    insert(minHeap, 7);
    insert(minHeap, 2);
    insert(minHeap, 6);

    printf("Minimum element: %d\n",
getMin(minHeap)); // Output: 2

    printf("Extracted min: %d\n", extractMin(minHeap));
// Output: 2
```

```
    printf("Minimum element after extraction: %d\n", getMin(minHeap)); // Output: 3

    insert(minHeap, 1);
    printf("Minimum element after insertion: %d\n", getMin(minHeap)); // Output: 1

    free(minHeap->arr);
    free(minHeap);
    return 0;
}
```

Output: -