Structures

● Graded

Student

ANSHUMAN SINGH

Total Points

200 / 200 pts

Autograder Score

200.0 / 200.0

Passed Tests

Test 1 (10/10)
Test 10 (20/20)
Test 11 (20/20)
Test 12 (20/20)
Test 13 (20/20)
Test 14 (20/20)
Test 2 (10/10)
Test 3 (10/10)
Test 4 (10/10)
Test 5 (10/10)
Test 6 (10/10)
Test 7 (10/10)
Test 8 (10/10)
Test 9 (20/20)

Autograder Results

Test 1 (10/10)

Test 10 (20/20)

Test 11 (20/20)

Test 12 (20/20)

Test 13 (20/20)

Test 14 (20/20)

Test 2 (10/10)

Test 3 (10/10)

Test 4 (10/10)

Test 5 (10/10)

Test 6 (10/10)

Test 7 (10/10)

Test 8 (10/10)

Test 9 (20/20)

Submitted Files

hw.c Download

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


//Stack implementation

typedef struct {
    int *arr;
    int top;
    int max;
} Stack;

void initStack(Stack *stack, int max) {
    stack->arr=(int*)malloc(max*sizeof(int));
    stack->top = -1;
    stack->max=max;
}

void push(Stack *stack, int value) {
    if (stack->top >= stack->max - 1) {
        printf("-1\n");
        exit(0);
    }
    stack->arr[++stack->top] = value;
}

void pop(Stack *stack) {
    if (stack->top < 0) {
        printf("-1\n");
        exit(0);
    }
    printf("%d\n", stack->arr[stack->top--]);
}

void printStack(Stack *stack) {
    if (stack->top < 0) {
        printf("\n");
        return;
    }
    for (int i=0;i<=stack->top;i++) {
        printf("%d ", stack->arr[i]);
    }
    printf("\n");
}


//Queue Implementation

typedef struct {
    int *arr;
    int head;
    int tail;
    int max;
} Queue;

void initQueue(Queue *queue, int max) {
    queue->arr = (int *)malloc(max * sizeof(int));
    queue->head = 0;
    queue->tail = 0;
    queue->max = max;
}

void enqueue(Queue *queue, int value) {
    int next = (queue->tail + 1) % queue->max;
    if (next == queue->head) {
        printf("-1\n");
        exit(0);
    }
    queue->arr[queue->tail] = value;
    queue->tail = next;
}

void dequeue(Queue *queue) {
    if (queue->head == queue->tail) {
        printf("-1\n");
        exit(0);
    }
    printf("%d\n", queue->arr[queue->head]);
    queue->head = (queue->head + 1) % queue->max;
}

void printQueue(Queue *queue) {
    if (queue->head == queue->tail) {
        printf("\n");
        return;
    }
    int i = queue->head;
    while (i != queue->tail) {
        printf("%d ", queue->arr[i]);
        i = (i + 1) % queue->max;
    }
    printf("\n");
}


//Min-Heap Implementation

typedef struct {
    int *arr;
    int size;
    int capacity;
```

```c
} MinHeap;

void initHeap(MinHeap *heap, int capacity) {
    heap->arr = (int *)malloc(capacity * sizeof(int));
    heap->size = 0;
    heap->capacity = capacity;
}

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void minHeapify(MinHeap *heap, int index) {
    int k = index;
    int left = 2 * index + 1;
    int right = 2 * index + 2;

    if (left < heap->size && heap->arr[left] < heap->arr[k]) {
        k = left;
    }
    if (right < heap->size && heap->arr[right] < heap->arr[k]) {
        k = right;
    }
    if (k != index) {
        swap(&heap->arr[index], &heap->arr[k]);
        minHeapify(heap, k);
    }
}

void buildHeap(MinHeap *heap, int *elements, int numElements) {
    if (numElements > heap->capacity) {
        printf("-1\n");
        exit(0);
    }

    heap->size = numElements;
    for (int i = 0; i < numElements; i++) {
        heap->arr[i] = elements[i];
    }

    for (int i = (heap->size / 2) - 1; i >= 0; i--) {
        minHeapify(heap, i);
    }
}

void decreaseKey(MinHeap *heap, int index, int newValue) {
    if (index < 0 || index >= heap->size || heap->arr[index] <= newValue) {
        return;
    }
    heap->arr[index] = newValue;
    while (index > 0 && heap->arr[(index - 1) / 2] > heap->arr[index]) {
        swap(&heap->arr[index], &heap->arr[(index - 1) / 2]);
        index = (index - 1) / 2;
    }
}

void extractMin(MinHeap *heap) {
    if (heap->size <= 0) {
        printf("-1\n");
        exit(0);
    }
    printf("%d\n", heap->arr[0]);
    heap->arr[0] = heap->arr[heap->size - 1];
    heap->size--;
    minHeapify(heap, 0);
}

void printHeap(MinHeap *heap) {
    if (heap->size == 0) {
        printf("\n");
        return;
    }
    for (int i = 0; i < heap->size; i++) {
        printf("%d ", heap->arr[i]);
    }
    printf("\n");
}

void freeHeap(MinHeap *heap) {
    free(heap->arr);
}



//main function

int main() {
    int o, MAX;
    scanf("%d %d", &o, &MAX);

    if (o == 0) {
        Stack stack;
    initStack(&stack, MAX);

    int command, value;
    while (scanf("%d", &command) != EOF) {
        switch (command) {
            case 0:
                printStack(&stack);
                break;
            case 1:
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                free(stack.arr);
                return 0;
            default:
                return 0;
        }
    }
    free(stack.arr);
    }
    else if(o == 1){
        Queue queue;
    initQueue(&queue, MAX);

    int command, value;
    while (scanf("%d", &command) != EOF) {
        switch (command) {
            case 0:
                printQueue(&queue);
                break;
            case 1:
                scanf("%d", &value);
                enqueue(&queue, value);
                break;
            case 2:
                dequeue(&queue);
                break;
            case 3:
                free(queue.arr);
                return 0;
            default:
                return 0;
        }
    }
    free(queue.arr);
    }
    else{
        MinHeap heap;
    initHeap(&heap, MAX);

    int command, numElements, index, value;
    while (scanf("%d", &command) != EOF) {
        switch (command) {
            case 0:
                printHeap(&heap);
                break;
            case 1:
                scanf("%d", &numElements);
```

```c
                if (numElements > MAX) {
                    printf("-1\n");
                    freeHeap(&heap);
                    exit(0);
                }
                int *elements = (int *)malloc(numElements * sizeof(int));
                for (int i = 0; i < numElements; i++) {
                    scanf("%d", &elements[i]);
                }
                buildHeap(&heap, elements, numElements);
                free(elements);
                break;
            case 2:
                scanf("%d %d", &index, &value);
                decreaseKey(&heap, index - 1, value);
                break;
            case 3:
                extractMin(&heap);
                break;
            case 4:
                freeHeap(&heap);
                return 0;
            default:
                freeHeap(&heap);
                return 0;
        }
    }
    freeHeap(&heap);
    }
    return 0;
}

```