

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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  //Stack implementation
6
7  typedef struct {
8      int *arr;
9      int top;
10     int max;
11 } Stack;
12
13 void initStack(Stack *stack, int max) {
14     stack->arr=(int*)malloc(max*sizeof(int));
15     stack->top = -1;
16     stack->max=max;
17 }
18
19 void push(Stack *stack, int value) {
20     if (stack->top >= stack->max - 1) {
21         printf("-1\n");
22         exit(0);
23     }
24     stack->arr[++stack->top] = value;
25 }
26
27 void pop(Stack *stack) {
28     if (stack->top < 0) {
29         printf("-1\n");
30         exit(0);
31     }
32     printf("%d\n", stack->arr[stack->top--]);
33 }
34
35 void printStack(Stack *stack) {
36     if (stack->top < 0) {
37         printf("\n");
38         return;
39     }
40     for (int i=0;i<=stack->top;i++) {
41         printf("%d ", stack->arr[i]);
42     }
43     printf("\n");
44 }
45
46
47 //Queue Implementation
48
49 typedef struct {
```

```

50     int *arr;
51     int head;
52     int tail;
53     int max;
54 } Queue;
55
56 void initQueue(Queue *queue, int max) {
57     queue->arr = (int *)malloc(max * sizeof(int));
58     queue->head = 0;
59     queue->tail = 0;
60     queue->max = max;
61 }
62
63 void enqueue(Queue *queue, int value) {
64     int next = (queue->tail + 1) % queue->max;
65     if (next == queue->head) {
66         printf("-1\n");
67         exit(0);
68     }
69     queue->arr[queue->tail] = value;
70     queue->tail = next;
71 }
72
73 void dequeue(Queue *queue) {
74     if (queue->head == queue->tail) {
75         printf("-1\n");
76         exit(0);
77     }
78     printf("%d\n", queue->arr[queue->head]);
79     queue->head = (queue->head + 1) % queue->max;
80 }
81
82 void printQueue(Queue *queue) {
83     if (queue->head == queue->tail) {
84         printf("\n");
85         return;
86     }
87     int i = queue->head;
88     while (i != queue->tail) {
89         printf("%d ", queue->arr[i]);
90         i = (i + 1) % queue->max;
91     }
92     printf("\n");
93 }
94
95
96 //Min-Heap Implementation
97
98 typedef struct {
99     int *arr;
100     int size;
101     int capacity;

```

```

102 } MinHeap;
103
104 void initHeap(MinHeap *heap, int capacity) {
105     heap->arr = (int *)malloc(capacity * sizeof(int));
106     heap->size = 0;
107     heap->capacity = capacity;
108 }
109
110 void swap(int *a, int *b) {
111     int temp = *a;
112     *a = *b;
113     *b = temp;
114 }
115
116 void minHeapify(MinHeap *heap, int index) {
117     int k = index;
118     int left = 2 * index + 1;
119     int right = 2 * index + 2;
120
121     if (left < heap->size && heap->arr[left] < heap->arr[k]) {
122         k = left;
123     }
124     if (right < heap->size && heap->arr[right] < heap->arr[k]) {
125         k = right;
126     }
127     if (k != index) {
128         swap(&heap->arr[index], &heap->arr[k]);
129         minHeapify(heap, k);
130     }
131 }
132
133 void buildHeap(MinHeap *heap, int *elements, int numElements) {
134     if (numElements > heap->capacity) {
135         printf("-1\n");
136         exit(0);
137     }
138
139     heap->size = numElements;
140     for (int i = 0; i < numElements; i++) {
141         heap->arr[i] = elements[i];
142     }
143
144     for (int i = (heap->size / 2) - 1; i >= 0; i--) {
145         minHeapify(heap, i);
146     }
147 }
148
149 void decreaseKey(MinHeap *heap, int index, int newValue) {
150     if (index < 0 || index >= heap->size || heap->arr[index] <= newValue) {
151         return;
152     }
153     heap->arr[index] = newValue;

```

```

154     while (index > 0 && heap->arr[(index - 1) / 2] > heap->arr[index]) {
155         swap(&heap->arr[index], &heap->arr[(index - 1) / 2]);
156         index = (index - 1) / 2;
157     }
158 }
159
160 void extractMin(MinHeap *heap) {
161     if (heap->size <= 0) {
162         printf("-1\n");
163         exit(0);
164     }
165     printf("%d\n", heap->arr[0]);
166     heap->arr[0] = heap->arr[heap->size - 1];
167     heap->size--;
168     minHeapify(heap, 0);
169 }
170
171 void printHeap(MinHeap *heap) {
172     if (heap->size == 0) {
173         printf("\n");
174         return;
175     }
176     for (int i = 0; i < heap->size; i++) {
177         printf("%d ", heap->arr[i]);
178     }
179     printf("\n");
180 }
181
182 void freeHeap(MinHeap *heap) {
183     free(heap->arr);
184 }
185
186
187
188 //main function
189
190 int main() {
191     int o, MAX;
192     scanf("%d %d", &o, &MAX);
193
194     if (o == 0) {
195         Stack stack;
196         initStack(&stack, MAX);
197
198         int command, value;
199         while (scanf("%d", &command) != EOF) {
200             switch (command) {
201                 case 0:
202                     printStack(&stack);
203                     break;
204                 case 1:
205                     scanf("%d", &value);

```

```
206     push(&stack, value);
207     break;
208 case 2:
209     pop(&stack);
210     break;
211 case 3:
212     free(stack.arr);
213     return 0;
214 default:
215     return 0;
216 }
217 }
218 free(stack.arr);
219 }
220 else if(o == 1){
221     Queue queue;
222     initQueue(&queue, MAX);
223
224     int command, value;
225     while (scanf("%d", &command) != EOF) {
226         switch (command) {
227             case 0:
228                 printQueue(&queue);
229                 break;
230             case 1:
231                 scanf("%d", &value);
232                 enqueue(&queue, value);
233                 break;
234             case 2:
235                 dequeue(&queue);
236                 break;
237             case 3:
238                 free(queue.arr);
239                 return 0;
240             default:
241                 return 0;
242         }
243     }
244     free(queue.arr);
245 }
246 else{
247     MinHeap heap;
248     initHeap(&heap, MAX);
249
250     int command, numElements, index, value;
251     while (scanf("%d", &command) != EOF) {
252         switch (command) {
253             case 0:
254                 printHeap(&heap);
255                 break;
256             case 1:
257                 scanf("%d", &numElements);
```

```
258     if (numElements > MAX) {
259         printf("-1\n");
260         freeHeap(&heap);
261         exit(0);
262     }
263     int *elements = (int *)malloc(numElements * sizeof(int));
264     for (int i = 0; i < numElements; i++) {
265         scanf("%d", &elements[i]);
266     }
267     buildHeap(&heap, elements, numElements);
268     free(elements);
269     break;
270 case 2:
271     scanf("%d %d", &index, &value);
272     decreaseKey(&heap, index - 1, value);
273     break;
274 case 3:
275     extractMin(&heap);
276     break;
277 case 4:
278     freeHeap(&heap);
279     return 0;
280 default:
281     freeHeap(&heap);
282     return 0;
283 }
284 }
285 freeHeap(&heap);
286 }
287 return 0;
288 }
289
```
