```cpp
//Priority Queue
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

// Structure representing an object with an ID, order, and priority
struct Object {
    string id;
    int order;
    int priority;
};

// Structure representing a priority queue implemented using a min-heap
struct PriorityQueueHeap {
    vector<Object> arr;
};

// Function to check if the priority queue is empty
bool isEmpty(const PriorityQueueHeap& pq) {
    return pq.arr.empty();
}

// Function to maintain the heap property while moving an element up the heap
void heapifyUp(vector<Object>& arr, int index) {
    while (index > 0) {
        int parentIndex = (index - 1) / 2;
        if (arr[index].priority < arr[parentIndex].priority) {
            swap(arr[index], arr[parentIndex]);
            index = parentIndex;
        }
        else {
            break;
        }
    }
}

// Function to maintain the heap property while moving an element down the heap
void heapifyDown(vector<Object>& arr, int index) {
    int size = arr.size();
    while (true) {
        int leftChild = 2 * index + 1;
        int rightChild = 2 * index + 2;
        int smallest = index;

```

```
48          if (leftChild < size && arr[leftChild].priority < arr
              [smallest].priority) {
49              smallest = leftChild;
50          }
51
52          if (rightChild < size && arr[rightChild].priority < arr
              [smallest].priority) {
53              smallest = rightChild;
54          }
55
56          if (smallest != index) {
57              swap(arr[index], arr[smallest]);
58              index = smallest;
59          }
60          else {
61              break;
62          }
63      }
64  }
65
66  // Function to insert an object into the priority queue
67  void insert(PriorityQueueHeap& pq, const Object& obj) {
68      pq.arr.push_back(obj);
69      heapifyUp(pq.arr, pq.arr.size() - 1);
70  }
71
72  // Function to extract the object with the highest priority from the
      priority queue
73  Object extract(PriorityQueueHeap& pq) {
74      if (isEmpty(pq)) {
75          cerr << "Error: Priority queue is empty." << endl;
76          // Handle error accordingly, here we just return an Object with an
              empty string.
77          return Object{ "", 0, 0 };
78      }
79
80      Object result = pq.arr[0];
81      pq.arr[0] = pq.arr.back();
82      pq.arr.pop_back();
83      heapifyDown(pq.arr, 0);
84
85      return result;
86  }
87
88  // Function to remove an object with a given ID from the priority queue
89  void remove(PriorityQueueHeap& pq, const string& objectId) {
90      int index = -1;
91      for (int i = 0; i < pq.arr.size(); ++i) {
92          if (pq.arr[i].id == objectId) {
```

```
93              index = i;
94              break;
95          }
96      }
97
98      if (index == -1) {
99          cerr << "Error: Object with id " << objectId << " not found." <<
              endl;
100         return;
101     }
102
103     pq.arr[index].priority = INT_MIN; // Set priority to negative infinity
104     heapifyUp(pq.arr, index);
105     (void)extract(pq);
106 }
107
108 // Function to change the priority of an object with a given ID
109 void changePriority(PriorityQueueHeap& pq, const string& objectId, int
      newPriority) {
110     int index = -1;
111     for (int i = 0; i < pq.arr.size(); ++i) {
112         if (pq.arr[i].id == objectId) {
113             index = i;
114             break;
115         }
116     }
117
118     if (index == -1) {
119         cerr << "Error: Object with id " << objectId << " not found." <<
              endl;
120         return;
121     }
122
123     int oldPriority = pq.arr[index].priority;
124     pq.arr[index].priority = newPriority;
125
126     if (newPriority < oldPriority) {
127         heapifyUp(pq.arr, index);
128     }
129     else {
130         heapifyDown(pq.arr, index);
131     }
132 }
```