K-Largest Elements in C++

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

void solve(int n, vector<int>& arr, int k) {
   priority_queue<int, vector<int>, greater<int>>
pq; // Min-heap

   for (int i = 0; i < arr.size(); ++i) {
      if (i < k) {
         pq.push(arr[i]);
      } else {
         if (arr[i] > pq.top()) {
            pq.pop();
            pq.push(arr[i]);
         }
      }
   }

   vector<int> result;
   while (!pq.empty()) {
      result.push_back(pq.top());
      pq.pop();
   }

   for (int j = result.size() - 1; j >= 0; --j) {
      cout << result[j] << " ";
   }
   cout << endl;
}

int main() {
   vector<int> num = {44, -5, -2, 41, 12, 19, 21, -6};
   int k = 2;
   solve(num.size(), num, k);

   return 0;
}
```

Input:

Array: {44, -5, -2, 41, 12, 19, 21, -6}
k = 2

Step 1: Initialize Min-Heap (pq)

We use a priority_queue<int, vector<int>, greater<int>> to create a min-heap.

Step 2: Process Array

- **Iteration 0 (i = 0):**
  - Push 44 into the heap.
    Min-Heap: {44}
- **Iteration 1 (i = 1):**
  - Push -5 into the heap.
    Min-Heap: {-5, 44}
- **Iteration 2 (i = 2):**
  - Compare -2 with the heap's top (-5):
    -2 > -5, so:
    - Pop -5 from the heap.
    - Push -2 into the heap.
      Min-Heap: {-2, 44}
- **Iteration 3 (i = 3):**
  - Compare 41 with the heap's top (-2):
    41 > -2, so:
    - Pop -2 from the heap.
    - Push 41 into the heap.
      Min-Heap: {41, 44}
- **Iteration 4 (i = 4):**
  - Compare 12 with the heap's top (41):
    12 < 41, so we skip this element.
    Min-Heap remains unchanged: {41, 44}
- **Iteration 5 (i = 5):**
  - Compare 19 with the heap's top (41):
    19 < 41, so we skip this element.
    Min-Heap remains unchanged: {41, 44}
- **Iteration 6 (i = 6):**
  - Compare 21 with the heap's top (41):
    21 < 41, so we skip this element.
    Min-Heap remains unchanged: {41, 44}
- **Iteration 7 (i = 7):**
  - Compare -6 with the heap's top (41):
    -6 < 41, so we skip this element.
    Min-Heap remains unchanged: {41,

	44} **Step 3: Extract and Store Result** The min-heap contains the top k largest elements: {41, 44}. Extract them and reverse the order to print the largest first. Result: [44, 41]
Output: 44 41	