Merge k sorted elements in C++

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

struct Pair {
   int li;   // List index
   int di;   // Data index (current index in the list)
   int val;  // Value at current index in the list

   Pair(int li, int di, int val) {
      this->li = li;
      this->di = di;
      this->val = val;
   }

   bool operator>(const Pair& other) const {
      return val > other.val;
   }
};

vector<int> mergeKSortedLists(vector<vector<int>>&
lists) {
   vector<int> rv;

   // Min-heap priority queue
   priority_queue<Pair, vector<Pair>, greater<Pair>>
pq;

   // Initialize the priority queue with the first
element from each list
   for (int i = 0; i < lists.size(); ++i) {
      if (!lists[i].empty()) {
         pq.push(Pair(i, 0, lists[i][0]));
      }
   }

   while (!pq.empty()) {
      Pair p = pq.top();
      pq.pop();

      // Add the current value to result vector
      rv.push_back(p.val);

      // Move to the next element in the same list
      p.di++;
      if (p.di < lists[p.li].size()) {
         p.val = lists[p.li][p.di];
         pq.push(p);
      }
   }

   return rv;
}

int main() {
   vector<vector<int>> lists = {
      {10, 20, 30, 40, 50},
      {5, 7, 9, 11, 19, 55, 57},
      {1, 2, 3}
   };
```

Step-by-Step Execution:

1. **Initialization:**
   - We declare a vector rv to store the merged result.
   - We use a **min-heap priority queue** to manage the smallest elements of each list.
   - The Pair struct stores:
     - li: Index of the list from which the element comes.
     - di: Index of the element in that list.
     - val: The value of the element.
2. **Inserting the First Elements into the Min-Heap:** We push the first element of each list into the min-heap:
   - From list 0: {10, 20, 30, 40, 50}, push 10.
   - From list 1: {5, 7, 9, 11, 19, 55, 57}, push 5.
   - From list 2: {1, 2, 3}, push 1.

   At this point, the priority queue (min-heap) looks like this:

   {1, 5, 10}
3. **Processing the Min-Heap:**
   - Pop the smallest element (1) from the heap, add it to the result list rv.
   - Push the next element from the same list (list 2) into the heap, which is 2.

   Now, the heap looks like:

   {2, 5, 10}

   - Pop the smallest element (2), add it to rv.
   - Push the next element from list 2 into the heap, which is 3.

   Now, the heap looks like:

   {3, 5, 10}

   - Pop the smallest element (3), add it to rv.
   - No more elements left in list 2.

   Now, the heap looks like:

   {5, 10}

   - Pop the smallest element (5), add it

```
    vector<int> mlist = mergeKSortedLists(lists);

    for (int val : mlist) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

to rv.

- Push the next element from list 1 into the heap, which is 7.

Now, the heap looks like:

{7, 10}

- Pop the smallest element (7), add it to rv.
- Push the next element from list 1 into the heap, which is 9.

Now, the heap looks like:

{9, 10}

- Pop the smallest element (9), add it to rv.
- Push the next element from list 1 into the heap, which is 11.

Now, the heap looks like:

{10, 11}

- Pop the smallest element (10), add it to rv.
- Push the next element from list 0 into the heap, which is 20.

Now, the heap looks like:

{11, 20}

- Pop the smallest element (11), add it to rv.
- Push the next element from list 1 into the heap, which is 19.

Now, the heap looks like:

{19, 20}

- Pop the smallest element (19), add it to rv.
- Push the next element from list 1 into the heap, which is 55.

Now, the heap looks like:

{20, 55}

- Pop the smallest element (20), add it to rv.
- Push the next element from list 0

	into the heap, which is 30. Now, the heap looks like: {30, 55} ○ Pop the smallest element (30), add it to rv. ○ Push the next element from list 0 into the heap, which is 40. Now, the heap looks like: {40, 55} ○ Pop the smallest element (40), add it to rv. ○ Push the next element from list 0 into the heap, which is 50. Now, the heap looks like: {50, 55} ○ Pop the smallest element (50), add it to rv. ○ No more elements left in list 0. Now, the heap looks like: {55} ○ Pop the smallest element (55), add it to rv. ○ Push the next element from list 1 into the heap, which is 57. Now, the heap looks like: {57} ○ Pop the smallest element (57), add it to rv. ○ No more elements left in list 1. **Final Output:** The merged result stored in rv is: 1 2 3 5 7 9 10 11 19 20 30 40 50 55 57s
Output: 1 2 3 5 7 9 10 11 19 20 30 40 50 55 57	