Sliding window maximum in C++

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

class SlidingWindowMaximum {
public:
    vector<int> maxSlidingWindow(vector<int>&
nums, int k) {
        int n = nums.size();
        vector<int> ans;
        deque<int> deque;

        // Process the first window of size k separately
        for (int i = 0; i < k; i++) {
            while (!deque.empty() && nums[deque.back()]
<= nums[i]) {
                deque.pop_back();
            }
            deque.push_back(i);
        }
        ans.push_back(nums[deque.front()]);

        // Process the rest of the elements
        for (int i = k; i < n; i++) {
            if (!deque.empty() && deque.front() == i - k) {
                deque.pop_front();
            }
            while (!deque.empty() && nums[deque.back()]
<= nums[i]) {
                deque.pop_back();
            }
            deque.push_back(i);
            ans.push_back(nums[deque.front()]);
        }

        return ans;
    }
};

int main() {
    SlidingWindowMaximum solution;

    // Example 1
    vector<int> nums1 = {1, 3, -1, -3, 5, 3, 6, 7};
    int k1 = 3;
    vector<int> result1 =
solution.maxSlidingWindow(nums1, k1);
    cout << "Max sliding window for nums1 and k=" <<
k1 << ": ";
    for (int num : result1) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

Dry Run Table:

Index i	Element nums[i]	Deque (indices)	Deque (values)	Max in window
0	1	[0]	[1]	-
1	3	[1]	[3]	-
2	-1	[1, 2]	[3, -1]	**3**
3	-3	[1, 2, 3]	[3, -1, -3]	**3**
4	5	[4]	[5]	**5**
5	3	[4, 5]	[5, 3]	**5**
6	6	[6]	[6]	**6**
7	7	[7]	[7]	**7**

🧠 **Explanation:**

- The deque stores **indices** of elements in the current window.
- It's maintained in **decreasing order of values**.
- For each new element:
    - Remove indices from the back if their value is smaller than current.
    - Remove the front index if it's out of the window range.
    - Push the current index to the deque.
    - The front of the deque always has the index of the **max** of current window.

✅ Final Output:
Max sliding window for nums1 and k=3: 3 3 5 5 6 7

Max sliding window for nums1 and k=3: 3 3 5 5 6 7