🔍 First Negative Number in Every Window of Size `k`

📌 Problem Statement (Based on Your Code)

Given an array of integers (may contain negative values), and a window size `k`, you need to print the **first negative number** in every window of size `k`.

👨‍💻 Your Code

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

int main() {
    int arr[] = {2,-3,4,4,-7,-1,4,-2,6};
    int n = sizeof(arr)/sizeof(arr[0]);  // 🔢 Number of elements in array
    int k = 3;  // 🪟 Window size
    int p = -1; // 🔎 Index of first negative in current window

    // 🔁 Step 1: Initialize the first window
    for(int i=0; i<k; i++) {
        if(arr[i] < 0) {
            p = i;     // 👉 Found first negative in first window
            break;
        }
    }

    vector<int> ans(n-k+1, 0);  // 🧾 Output vector to store first negatives

    int i = 1;                // Left pointer of window
    int j = i + k - 1;          // Right pointer of window
```

```
// ✅ First window answer
ans[0] = (p != -1) ? arr[p] : 0;

// 🔁 Step 2: Slide the window
while(j < n) {
    if(p >= i) {
        // ✅ The negative number is still in the window
        ans[i] = arr[p];
    } else {
        // 🔄 Negative number has moved out — find new one
        p = -1;
        for(int x = i; x <= j; x++) {
            if(arr[x] < 0) {
                p = x;
                break;
            }
        }
        ans[i] = (p != -1) ? arr[p] : 0;
    }
    i++;
    j++;
}

// 🖨️ Print original array
for(int i = 0; i < n; i++) {
    cout << arr[i] << " ";
}
cout << endl;

// 🖨️ Print result
for(int i = 0; i < n - k + 1; i++) {
    cout << ans[i] << " ";
}
```

```
    return 0;
}
```

🧠 Logic Breakdown

Step	What Happens
🔍	First loop finds first negative in the **first window**
📦	`p` stores the index of that negative number
🔄	Window starts sliding using two pointers `i` and `j`
✅	If `p` (the last known negative) is still in the window, use it
🔃	If not, loop through the new window to find the first negative again
🧾	Store result in the `ans` vector
🖨️	Print original array and result array

🧪 Example Trace

Given:

```
arr = {2, -3, 4, 4, -7, -1, 4, -2, 6}
k = 3
```

Sliding Windows:

Window	First Negative
[2, -3, 4]	-3
[-3, 4, 4]	-3
[4, 4, -7]	-7
[4, -7, -1]	-7
[-7, -1, 4]	-7
[-1, 4, -2]	-1
[4, -2, 6]	-2

So, output will be:

```
-3 -3 -7 -7 -7 -1 -2
```

🧠 Time Complexity

Operation	Complexity
Initial window check	`O(k)`
Sliding windows	Worst case `O(n*k)` (when re-scanning window)
Overall	`O(n*k)` worst, but usually faster in practice

🔄 **Optimization Tip**: You can reduce time to `O(n)` using a `deque` for storing indices of negative numbers, but your method is still correct and clean! 💯

🏁 Final Output:

```
Original Array: 2 -3 4 4 -7 -1 4 -2 6
Answer:         -3 -3 -7 -7 -7 -1 -2
```

✅ Summary (Emoji Style)

Step	Task	Emoji Summary
🪟 Init	Check first window for negative	🔎
🧭 Track	Use index `p` to remember first negative	📍
🔄 Slide	Move window one step, update `p` if needed	⏭️🔄
🧾 Store	Save first negative in result array	📝
🖨️ Print	Print original and output arrays	📤
