```cpp
#include <bits/stdc++.h>
using namespace std;

//queue using stack
class Queue {
public:
    stack<int> st1; // push stack
    stack<int> st2; // pop stack

    bool empty() {
        return st1.empty() && st2.empty();
    }

    void push(int ele) {
        st1.push(ele);
    }

    int pop() {
        if (empty()) {
            throw runtime_error("Queue Underflow: Cannot pop from empty queue");
        }

        if (st2.empty()) {
            while (!st1.empty()) {
                st2.push(st1.top());
                st1.pop();
            }
        }

        int ele = st2.top();
        st2.pop();
        return ele;
    }

    int peek() {
        if (empty()) {
            throw runtime_error("Queue is empty");
        }

        if (st2.empty()) {
            while (!st1.empty()) {
                st2.push(st1.top());
                st1.pop();
            }
        }

        return st2.top();
    }
};

int main() {
    Queue q;
    q.push(5);
```

```
    q.push(10);
    q.push(20);
    cout << "Front: " << q.peek() << endl;
    cout << "Popped: " << q.pop() << endl;
    cout << "Front after pop: " << q.peek() << endl;
    cout << "Is empty? " << q.empty() << endl;
    q.pop();
    cout << "Is empty? " << q.empty() << endl;

    return 0;
}




// print first negative integer in every window of size k=3
// Method-1 (Less Optimized) - iterating all the elements(positive ele present before
the first -ve ele) for finding first -ve ele
void display(queue<int> q){
    while(!q.empty()){
        if(q.front() < 0){
            cout<<q.front()<<" ";
            return;
        }
        q.pop();
    }
    cout<<0<<" ";
}
int main(){
    vector<int> v = {2,-3,-4,-2,7,8,9,-10};
    queue<int> q;
    int k = 3;
    for(int i=0; i<k-1; i++){
        q.push(v[i]);
    }
    for(int i=k-1; i<v.size(); i++){
        q.push(v[i]);
        display(q);
        q.pop();
    };
    return 0;
}


//Method-2 (Optimized) - Only Storing -ve ele's , eliminating iterations
void display(queue<int> q, vector<int> v){
    if(q.empty()){
        cout<<0<<" ";
        return;
    }
    cout<<v[q.front()]<<" ";
}

int main(){
```

```
    vector<int> v = {2,-3,-4,-2,7,8,9,-10};
    queue<int> q;
    int k = 3;
    for(int i=0; i<k-1; i++){
        if(v[i]<0){
            q.push(i);
        }
    }
    for(int i=k-1; i<v.size(); i++){
        if(v[i]<0){
            q.push(i);
        }
        while(!q.empty() and q.front() <= i-k){
            q.pop();
        }
        display(q,v);
    }
    return 0;
}
```

```
//dequeue using array
#include <iostream>
using namespace std;

class Dequeue {
    int start, end, size, count;
    int *arr;

public:
    Dequeue(int n) {
        size = n;
        start = 0;
        end = n - 1;
        count = 0;
        arr = new int[n];
    }

    bool isFull() {
        return count == size;
    }

    bool isEmpty() {
        return count == 0;
    }

    void push_start(int x) {
        if (isFull()) {
            cout << "Dequeue Overflow\n";
            return;
        }
        start = (start - 1 + size) % size;
```

```
        arr[start] = x;
        count++;
    }

    void push_back(int x) {
        if (isFull()) {
            cout << "Dequeue Overflow\n";
            return;
        }
        end = (end + 1) % size;
        arr[end] = x;
        count++;
    }

    int front() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return -1;
        }
        return arr[start];
    }

    int back() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return -1;
        }
        return arr[end];
    }

    void pop_start() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return;
        }
        start = (start + 1) % size;
        count--;
    }

    void pop_back() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return;
        }
        end = (end - 1 + size) % size;
        count--;
    }
};

int main() {
    Dequeue d(5);
    cout << d.front() << endl;      // No Ele
    d.push_back(3);
    d.push_start(2);
```

```
    cout << d.front() << endl;      // 2
    cout << d.back() << endl;       // 3
    d.pop_start();                  // remove 2
    cout << d.front() << endl;      // 3
    d.push_start(5);
    d.push_start(6);
    d.push_back(7);                 // Should fill it
    d.push_back(8);
    d.push_back(9);                 //should thrown an overflow error

    return 0;
}
```