```
#include <iostream>
#include <stdexcept>

class MaxHeap {
private:
    int *heap;          // Pointer to array to store the heap elements
    int capacity;       // Maximum capacity of the heap
    int size;           // Current number of elements in the heap

    // Helper functions for heap operations
    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return 2 * i + 1; }
    int right(int i) { return 2 * i + 2; }

    // Moves the element at index `i` up to maintain max-heap property
    void heapifyUp(int i) {
        while (i > 0 && heap[parent(i)] < heap[i]) {
            std::swap(heap[i], heap[parent(i)]);
            i = parent(i);
        }
    }

    // Moves the element at index `i` down to maintain max-heap property
    void heapifyDown(int i) {
        int largest = i;
        int l = left(i);
        int r = right(i);

        if (l < size && heap[l] > heap[largest])
            largest = l;

        if (r < size && heap[r] > heap[largest])
            largest = r;

        if (largest != i) {
            std::swap(heap[i], heap[largest]);
            heapifyDown(largest);
        }
    }

public:
    // Constructor to initialize heap with a fixed capacity
    MaxHeap(int capacity) {
        this->capacity = capacity;
    heap = new int[capacity];
    size = 0;
}

// Destructor to clean up dynamically allocated array
~MaxHeap() {
    delete[] heap;
}

// Insert a new element into the heap
void push(int val) {
    if (size == capacity)
        throw std::overflow_error("Heap overflow");

    heap[size] = val;
    size++;
    heapifyUp(size - 1);
}

// Remove and return the maximum element
int pop() {
    if (size <= 0)
        throw std::underflow_error("Heap is empty");

    int root = heap[0];
    heap[0] = heap[size - 1];
    size--;
    heapifyDown(0);

    return root;
}

// Get the maximum element
int top() {
    if (size <= 0)
        throw std::underflow_error("Heap is empty");
    return heap[0];
}

// Check if the heap is empty
bool empty() const {
    return size == 0;
}
```

```
    // Get the current size of the heap
    int getSize() const {
        return size;
    }
};

int main() {
    MaxHeap maxHeap(10); // Create a max-heap with capacity 10

    // Insert elements into the heap
    maxHeap.push(10);
    maxHeap.push(30);
    maxHeap.push(20);
    maxHeap.push(5);

    std::cout << "Elements in max-heap priority queue:" << std::endl;
    while (!maxHeap.empty()) {
        std::cout << maxHeap.top() << " ";  // Access the top element
        maxHeap.pop();                      // Remove the top element
    }

    return 0;
}
```