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

class Queue {
private:
    int* q_array;
    int capacity;
    int q_size;
    int q_front;
    int q_back;

    void copyQueue(const Queue& other) {
        capacity = other.capacity;
        q_size = other.q_size;
        q_front = other.q_front;
        q_back = other.q_back;

        q_array = new int[capacity];
        for (int i = 0; i < capacity; ++i) {
            q_array[i] = other.q_array[i];
        }
    }

public:
    Queue(int cap = 10) : capacity(cap), q_size(0), q_front(0), q_back(-1) {
        q_array = new int[capacity];
    }

    Queue(const Queue& other) {
        copyQueue(other);
    }

    Queue& operator = (const Queue& other) {
        if (this != &other) {
            delete[] q_array;
            copyQueue(other);
        }
        return *this;
    }

    ~Queue() {
        delete[] q_array;
    }
```

```
bool Empty() const {
    return q_size == 0;
}

int Size() const {
    return q_size;
}

void Clear() {
    q_size = 0;
    q_front = 0;
    q_back = -1;
}

int Front() const {
    if (Empty()) {
        throw std::out_of_range("Queue is empty");
    }
    return q_array[q_front];
}

int Back() const {
    if (Empty()) {
        throw std::out_of_range("Queue is empty");
    }
    return q_array[q_back];
}

void Enqueue(int value) {
    if (q_size == capacity) {
        throw out_of_range("Queue is full");
    }
    q_back = (q_back + 1) % capacity;
    q_array[q_back] = value;
    ++q_size;
}

void Dequeue() {
    if (Empty()) {
        throw out_of_range("Queue is empty");
    }
    q_front = (q_front + 1) % capacity;
    --q_size;
}
```

```
};

int main() {
    Queue q(5);
    q.Enqueue(1);
    q.Enqueue(2);
    q.Enqueue(3);

    cout << "Front: " << q.Front() << "\n";
    cout << "Back: " << q.Back() << "\n";

    q.Dequeue();
    cout << "Front after dequeue: " << q.Front() << "\n";

    return 0;
}
```