1. Queue

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

class Queue {
private:
    int front, rear, size;
    int *arr;

public:
    Queue(int size) {
        this->size = size;
        arr = new int[size];
        front = 0;
        rear = -1;
    }

    // Enqueue operation
    void enqueue(int x) {
        if (rear == size - 1) {
            cout << "Queue is Full!" << endl;
            return;
        }
        arr[++rear] = x;
    }

    // Dequeue operation
    void dequeue() {
        if (front > rear) {
            cout << "Queue is Empty!" << endl;
            return;
```

```
        }
        cout << "Dequeued: " << arr[front++] << endl;
    }

    // Front operation
    int getFront() {
        if (front > rear) {
            cout << "Queue is Empty!" << endl;
            return -1;
        }
        return arr[front];
    }

    // Check if queue is empty
    bool isEmpty() {
        return front > rear;
    }
};

int main() {
    Queue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);

    cout << "Front element: " << q.getFront() << endl;

    q.dequeue();
```

```
    cout << "Front element: " << q.getFront() << endl;

    q.dequeue();
    q.dequeue();

    return 0;
}
```

2. Enqueue based on Priority

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

class PriorityQueue {
private:
    struct Element {
        int value;
        int priority;
    };

    vector<Element> queue;

    // Comparator to sort elements by priority (higher priority first)
    static bool compare(Element a, Element b) {
        return a.priority > b.priority;
    }

public:
    // Enqueue operation
    void enqueue(int x, int priority) {
        Element newElement = {x, priority};
```

```
        queue.push_back(newElement);
        // Sort the queue so that highest priority element comes first
        sort(queue.begin(), queue.end(), compare);
    }

    // Dequeue operation
    void dequeue() {
        if (queue.empty()) {
            cout << "Queue is Empty!" << endl;
            return;
        }
        cout << "Dequeued: " << queue.front().value << endl;
        queue.erase(queue.begin());
    }

    // Front operation
    int getFront() {
        if (queue.empty()) {
            cout << "Queue is Empty!" << endl;
            return -1;
        }
        return queue.front().value;
    }

    // Check if queue is empty
    bool isEmpty() {
        return queue.empty();
    }
};

int main() {
```

```
    PriorityQueue pq;

    pq.enqueue(10, 1); // Value: 10, Priority: 1
    pq.enqueue(20, 3); // Value: 20, Priority: 3
    pq.enqueue(30, 2); // Value: 30, Priority: 2

    cout << "Front element: " << pq.getFront() << endl; // Should be 20, as it has the highest priority

    pq.dequeue(); // Dequeues element with priority 3 (20)
    cout << "Front element after dequeue: " << pq.getFront() << endl; // Should be 30

    pq.dequeue(); // Dequeues element with priority 2 (30)
    pq.dequeue(); // Dequeues element with priority 1 (10)

    pq.dequeue(); // Should show "Queue is Empty!"

    return 0;
}
```

3. Dequeue

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

class Queue {
private:
    int *arr;
    int front, rear, size;

public:
    Queue(int size) {
        this->size = size;
```

```
    arr = new int[size];
    front = 0;
    rear = -1;
}

// Enqueue operation
void enqueue(int x) {
    if (rear == size - 1) {
        cout << "Queue is Full!" << endl;
        return;
    }
    arr[++rear] = x;
}

// Dequeue operation
void dequeue() {
    if (front > rear) {
        cout << "Queue is Empty!" << endl;
        return;
    }
    cout << "Dequeued: " << arr[front] << endl;
    front++;
}

// Front operation
int getFront() {
    if (front > rear) {
        cout << "Queue is Empty!" << endl;
        return -1;
    }
    return arr[front];
```

```
    }

    // Check if queue is empty
    bool isEmpty() {
        return front > rear;
    }
};

int main() {
    Queue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);

    cout << "Front element: " << q.getFront() << endl;

    q.dequeue(); // Removes 10
    cout << "Front element after dequeue: " << q.getFront() << endl;

    q.dequeue(); // Removes 20
    q.dequeue(); // Removes 30

    return 0;
}
```