1. Implement a Queue using an Array:

```
#include <iostream>

using namespace std;


class Queue {

private:

   int arr[100];

   int front, rear;


public:

   Queue() {

      front = -1;

      rear = -1;

   }


   void enqueue(int item) {

      if (rear == 99) {

         cout << "Queue is full\n";

         return;

      }

      if (front == -1) front = 0;

      arr[++rear] = item;

   }


   int dequeue() {

      if (front == -1 || front > rear) {

         cout << "Queue is empty\n";
```

```
            return -1;
        }
        return arr[front++];
    }

    void display() {
        if (front == -1 || front > rear) {
            cout << "Queue is empty\n";
            return;
        }
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

// Example usage
int main() {
    Queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.display();
    cout << q.dequeue() << endl;
    q.display();
    return 0;
```

```
}
```

2. Implement a Queue using Two Stacks

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

class Queue {
private:
   stack<int> stack1, stack2;

public:
   void enqueue(int item) {
      stack1.push(item);
   }

   int dequeue() {
      if (stack2.empty()) {
         while (!stack1.empty()) {
            stack2.push(stack1.top());
            stack1.pop();
         }
      }
      if (stack2.empty()) {
         cout << "Queue is empty\n";
         return -1;
      }
      int item = stack2.top();
```

```
        stack2.pop();
        return item;
    }
};

// Example usage
int main() {
    Queue q;
    q.enqueue(1);
    q.enqueue(2);
    cout << q.dequeue() << endl;
    q.enqueue(3);
    cout << q.dequeue() << endl;
    cout << q.dequeue() << endl;
    return 0;
}
```

3. Implement a Circular Queue

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

class CircularQueue {
private:
    int *queue;
    int front, rear, size;

public:
    CircularQueue(int size) {
```

```
    this->size = size;
    queue = new int[size];
    front = rear = -1;
}

void enqueue(int item) {
    if ((rear + 1) % size == front) {
        cout << "Queue is full\n";
        return;
    }
    if (front == -1) front = rear = 0;
    else rear = (rear + 1) % size;
    queue[rear] = item;
}

int dequeue() {
    if (front == -1) {
        cout << "Queue is empty\n";
        return -1;
    }
    int item = queue[front];
    if (front == rear) front = rear = -1;
    else front = (front + 1) % size;
    return item;
}

void display() {
```

```
        if (front == -1) {
            cout << "Queue is empty\n";
            return;
        }
        int i = front;
        while (i != rear) {
            cout << queue[i] << " ";
            i = (i + 1) % size;
        }
        cout << queue[i] << endl;
    }
};

// Example usage
int main() {
    CircularQueue cq(3);
    cq.enqueue(1);
    cq.enqueue(2);
    cq.enqueue(3);
    cq.display();
    cq.dequeue();
    cq.enqueue(4);
    cq.display();
    return 0;
}
```

4. Reverse a Queue

```
#include <iostream>
```

```
#include <queue>
#include <stack>
using namespace std;

void reverseQueue(queue<int> &q) {
    stack<int> s;
    while (!q.empty()) {
        s.push(q.front());
        q.pop();
    }
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }
}

// Example usage
int main() {
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);
    reverseQueue(q);
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
```

```
    return 0;
}
```

5. Find the First Negative Integer in Every Window of Size k

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

vector<int> firstNegativeInWindow(vector<int> &arr, int k) {
    deque<int> dq;
    vector<int> result;
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] < 0) dq.push_back(i);
        if (i >= k - 1) {
            if (!dq.empty() && dq.front() < i - k + 1) dq.pop_front();
            result.push_back(dq.empty() ? 0 : arr[dq.front()]);
        }
    }
    return result;
}

// Example usage
int main() {
    vector<int> arr = {12, -1, -7, 8, 15, 30, -5, 28};
    int k = 3;
    vector<int> result = firstNegativeInWindow(arr, k);
    for (int x : result) cout << x << " ";
```

```
    return 0;
}
```

6. Generate Binary Numbers from 1 to N

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

void generateBinaryNumbers(int n) {
    queue<string> q;
    q.push("1");
    for (int i = 0; i < n; i++) {
        string num = q.front();
        q.pop();
        cout << num << " ";
        q.push(num + "0");
        q.push(num + "1");
    }
}

// Example usage
int main() {
    generateBinaryNumbers(10);
    return 0;
}
```

7. Implement a Priority Queue

```
#include <iostream>
#include <queue>
```

```
using namespace std;


int main() {
    priority_queue<pair<int, string>, vector<pair<int, string>>, greater<>> pq;
    pq.push({2, "B"});
    pq.push({1, "A"});
    pq.push({3, "C"});


    while (!pq.empty()) {
        cout << pq.top().second << endl;
        pq.pop();
    }
    return 0;
}
```