1st Create a queue

#include <iostream>

#define MAX 100

Using namespace std;

class Queue {

private:

    int arr[MAX];

    int front;

    int rear;

    int size;

public:

    Queue() {

        front = 0;

        rear = -1;

        size = 0;

    }

    // Enqueue operation

    void enqueue(int value) {

        if (size == MAX) {

            cout << "Queue is full!\n";

            return;

        }

        rear = (rear + 1) % MAX;

        arr[rear] = value;

        size++;

        cout << value << " added to the queue.\n";

    }

    // Dequeue operation

    void dequeue() {

        if (isEmpty()) {

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

            return;

        }

        cout << arr[front] << " removed from the queue.\n";

        front = (front + 1) % MAX;

        size--;

    }

    // Peek the front element

    int peek() {

        if (isEmpty()) {

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

            return -1;

        }

        return arr[front];

    }

    // Check if the queue is empty

    bool isEmpty() {

        return size == 0;

    }

    // Display the queue

    void display() {

        if (isEmpty()) {

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

            return;

        }

        cout << "Queue elements: ";

        for (int i = 0; i < size; i++) {

            cout << arr[(front + i) % MAX] << " ";

        }

        cout << "\n";

    }

};

int main() {

    Queue q;

    int choice, value;

    do {

        cout << "\nQueue Operations:\n";

        cout << "1. Enqueue\n";

        cout << "2. Dequeue\n";

        cout << "3. Peek\n";

        cout << "4. Display\n";

        cout << "5. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

        case 1:

            cout << "Enter the value to enqueue: ";

            cin >> value;

            q.enqueue(value);

            break;

        case 2:

            q.dequeue();

            break;

        case 3:

            value = q.peek();

            if (value != -1) {

cout << "Front element: " << value << "\n";

            }

            break;

        case 4:

            q.display();

            break;

        case 5:

            cout << "Exiting...\n";

            break;

        default:

            cout << "Invalid choice!\n";

        }

    } while (choice != 5);

    return 0;

}

2nd Stack

#include <iostream>

#define MAX 1000 // Define the maximum size of the stack

class Stack {

private:

    int top;             // Index of the top element

    int arr[MAX];        // Array to store stack elements

public:

    Stack() { top = -1; } // Constructor initializes top to -1

    // Method to push an element onto the stack

    bool push(int value) {

        if (top >= (MAX - 1)) {

            std::cout << "Stack Overflow\n";

            return false;

        } else {

            arr[++top] = value;

            std::cout << value << " pushed onto the stack\n";

            return true;

        }

    }

    // Method to pop an element from the stack

    int pop() {

        if (top < 0) {

            std::cout << "Stack Underflow\n";

            return 0;

        } else {

            int value = arr[top--];

            return value;

        }

    }

    // Method to peek at the top element of the stack

    int peek() {

        if (top < 0) {

            std::cout << "Stack is Empty\n";

            return 0;

        } else {

            return arr[top];

        }

    }

    // Method to check if the stack is empty

    bool isEmpty() {

        return (top < 0);

    }

};

int main() {

    Stack stack;

    stack.push(10);

    stack.push(20);

    stack.push(30);

    std::cout << stack.pop() << " popped from the stack\n";

    std::cout << "Top element is: " << stack.peek() << "\n";

    std::cout << "Stack is " << (stack.isEmpty() ? "empty" : "not empty") << "\n";

    return 0;

}

3.

#include <bits/stdc++.h>

**using** **namespace** std;

// Queue elements after sortedIndex are

// already sorted. This function returns

// index of minimum element from front to

// sortedIndex

**int** minIndex(queue<**int**> &q, **int** sortedIndex)

{

**int** min\_index = -1;

**int** min\_val = INT\_MAX;

**int** n = q.size();

**for** (**int** i=0; i<n; i++)

    {

**int** curr = q.front();

        q.pop();  // This is dequeue() in C++ STL

        // we add the condition i <= sortedIndex

        // because we don't want to traverse

        // on the sorted part of the queue,

        // which is the right part.

**if** (curr <= min\_val && i <= sortedIndex)

        {

            min\_index = i;

            min\_val = curr;

        }

        q.push(curr);  // This is enqueue() in

                       // C++ STL

    }

**return** min\_index;

}

// Moves given minimum element to rear of

// queue

**void** insertMinToRear(queue<**int**> &q, **int** min\_index)

{

**int** min\_val;

**int** n = q.size();

**for** (**int** i = 0; i < n; i++)

    {

**int** curr = q.front();

        q.pop();

**if** (i != min\_index)

            q.push(curr);

**else**

            min\_val = curr;

    }

    q.push(min\_val);

}

**void** sortQueue(queue<**int**> &q)

{

**for** (**int** i = 1; i <= q.size(); i++)

    {

**int** min\_index = minIndex(q, q.size() - i);

        insertMinToRear(q, min\_index);

    }

}

// driver code

**int** main()

{

  queue<**int**> q;

  q.push(30);

  q.push(11);

  q.push(15);

  q.push(4);

  // Sort queue

  sortQueue(q);

  // Print sorted queue

**while** (q.empty() == **false**)

  {

     cout << q.front() << " ";

     q.pop();

  }

  cout << endl;

**return** 0;

}

4.

#include <bits/stdc++.h>

**using** **namespace** std;

// Utility function to print the queue

**void** Print(queue<**int**>& Queue)

{

**while** (!Queue.empty()) {

        cout << Queue.front() << " ";

        Queue.pop();

    }

}

// Function to reverse the queue

**void** reverseQueue(queue<**int**>& Queue)

{

    stack<**int**> Stack;

**while** (!Queue.empty()) {

        Stack.push(Queue.front());

        Queue.pop();

    }

**while** (!Stack.empty()) {

        Queue.push(Stack.top());

        Stack.pop();

    }

}

// Driver code

**int** main()

{

    queue<**int**> Queue;

    Queue.push(10);

    Queue.push(20);

    Queue.push(30);

    Queue.push(40);

    Queue.push(50);

    Queue.push(60);

    Queue.push(70);

    Queue.push(80);

    Queue.push(90);

    Queue.push(100);

    reverseQueue(Queue);

    Print(Queue);

}

5.

#include <bits/stdc++.h>

**using** **namespace** std;

**struct** Queue {

    stack<**int**> s1, s2;

**void** enQueue(**int** x)

    {

        // Move all elements from s1 to s2

**while** (!s1.empty()) {

            s2.push(s1.top());

            s1.pop();

        }

        // Push item into s1

        s1.push(x);

        // Push everything back to s1

**while** (!s2.empty()) {

            s1.push(s2.top());

            s2.pop();

        }

    }

    // Dequeue an item from the queue

**int** deQueue()

    {

        // if first stack is empty

**if** (s1.empty()) {

**return** -1;

        }

        // Return top of s1

**int** x = s1.top();

        s1.pop();

**return** x;

    }

};

// Driver code

**int** main()

{

    Queue q;

    q.enQueue(1);

    q.enQueue(2);

    q.enQueue(3);

    cout << q.deQueue() << '\n';

    cout << q.deQueue() << '\n';

    cout << q.deQueue() << '\n';

**return** 0;

}