**Simple Queue**

#include <iostream>

#include <queue>

int main() {

std::queue<int> q;

q.push(10);

q.push(20);

q.push(30);

while (!q.empty()) {

std::cout << q.front() << " ";

q.pop();

}

return 0;

}

**2. Priority Queue**

**Max-Heap (Default)**

#include <iostream>

#include <queue>

int main() {

std::priority\_queue<int> pq;

pq.push(10);

pq.push(20);

pq.push(15);

while (!pq.empty()) {

std::cout << pq.top() << " ";

pq.pop();

}

return 0;

}

**Min-Heap**

#include <iostream>

#include <queue>

#include <vector>

int main() {

std::priority\_queue<int, std::vector<int>, std::greater<int>> pq;

pq.push(10);

pq.push(20);

pq.push(15);

while (!pq.empty()) {

std::cout << pq.top() << " ";

pq.pop();

}

return 0;

}

**3. Deque (Double-Ended Queue)**

#include <iostream>

#include <deque>

int main() {

std::deque<int> dq;

dq.push\_back(10);

dq.push\_front(20);

dq.push\_back(30);

for (int x : dq) {

std::cout << x << " ";

}

std::cout << std::endl;

dq.pop\_front();

dq.pop\_back();

for (int x : dq) {

std::cout << x << " ";

}

return 0;

}

**4. Circular Queue**

#include <iostream>

#define SIZE 5

class CircularQueue {

private:

int items[SIZE], front, rear;

public:

CircularQueue() : front(-1), rear(-1) {}

bool isFull() {

return (front == 0 && rear == SIZE - 1) || (front == rear + 1);

}

bool isEmpty() {

return front == -1;

}

void enqueue(int element) {

if (isFull()) {

std::cout << "Queue is full\n";

return;

}

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

rear = (rear + 1) % SIZE;

items[rear] = element;

}

void dequeue() {

if (isEmpty()) {

std::cout << "Queue is empty\n";

return;

}

std::cout << "Removed: " << items[front] << "\n";

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

else front = (front + 1) % SIZE;

}

void display() {

if (isEmpty()) {

std::cout << "Queue is empty\n";

return;

}

int i = front;

while (true) {

std::cout << items[i] << " ";

if (i == rear) break;

i = (i + 1) % SIZE;

}

std::cout << "\n";

}

};

int main() {

CircularQueue cq;

cq.enqueue(10);

cq.enqueue(20);

cq.enqueue(30);

cq.display();

cq.dequeue();

cq.display();

cq.enqueue(40);

cq.enqueue(50);

cq.enqueue(60);

cq.display();

return 0;

}