National University of Computer and Emerging Sciences



Name: Muhammad Suleman

Roll #: 22F-3350

Section: BCS-4B

Lab # 08

# Problem 01

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node(int value)

: data(value), next(nullptr) {}

};

class Queue

{

private:

Node\* front;

Node\* rear;

public:

Queue()

: front(nullptr), rear(nullptr) {}

~Queue()

{

while (!isEmpty())

{

dequeue();

}

}

bool isEmpty()

{

return front == nullptr;

}

void enqueue(int value)

{

Node\* newNode = new Node(value);

if (isEmpty())

{

front = newNode;

rear = newNode;

}

else

{

rear->next = newNode;

rear = newNode;

}

cout << value << " enqueued" << endl;

}

int dequeue()

{

if (isEmpty())

{

cout << "Unable to dequeue (Queue is empty)" << endl;

return INT\_MIN;

}

Node\* temp = front;

front = front->next;

if (front == nullptr)

{

rear = nullptr;

}

cout << temp->data << " dequeued" << endl;

int value = temp->data;

delete temp;

return value;

}

int getFront()

{

if (isEmpty())

{

cout << "No front element (Queue is empty)" << endl;

return -1;

}

return front->data;

}

int getRear()

{

if (isEmpty())

{

cout << "No rear element (Queue is empty)" << endl;

return -1;

}

return rear->data;

}

void QueueProcessing(Queue &q1, Queue &q2, Queue &q3)

{

while (!q3.isEmpty() || !q2.isEmpty())

{

q1.dequeue();

if (!q2.isEmpty())

q1.enqueue(q2.dequeue());

if (!q3.isEmpty())

q2.enqueue(q3.dequeue());

}

while (!q1.isEmpty())

{

q1.dequeue();

}

}

};

int main()

{

Queue queue1;

Queue queue2;

Queue queue3;

for (int i = 0; i < 5; i++)

{

queue1.enqueue(i + 1);

}

for (int i = 5; i < 10; i++)

{

queue2.enqueue(i + 1);

}

for (int i = 10; i < 15; i++)

{

queue3.enqueue(i + 1);

}

Queue temp;

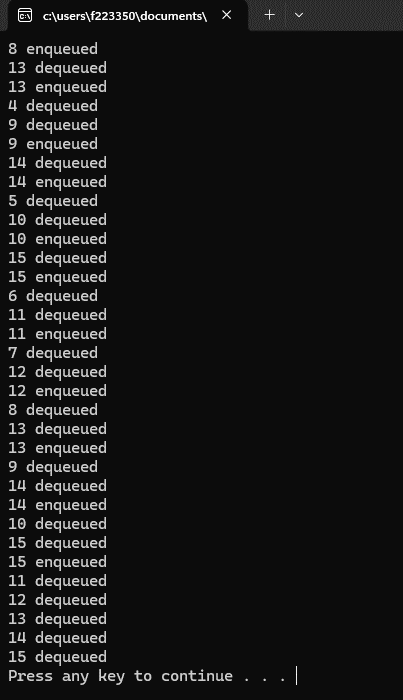
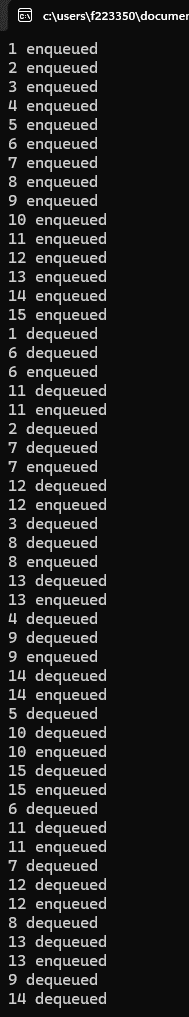
temp.QueueProcessing(queue1, queue2, queue3);

system("pause");

return 0;

}

## Output



# Problem 02

#include <iostream>

#include <string>

using namespace std;

class Car

{

private:

int carNum;

string parkingLocation;

public:

Car(int carNum)

: carNum(carNum), parkingLocation("Parking not available") {}

Car(int carNum, string location)

: carNum(carNum), parkingLocation(location) {}

void setParkingLocation(string location)

{

parkingLocation = location;

}

int getCarNum()

{

return carNum;

}

string getParkingLocation()

{

return parkingLocation;

}

};

class Node

{

public:

Car car;

Node\* next;

Node\* prev;

Node(Car car)

: car(car), next(nullptr), prev(nullptr) {}

};

class Deque

{

private:

Node\* front;

Node\* rear;

public:

Deque()

: front(nullptr), rear(nullptr) {}

bool isEmpty()

{

return front == nullptr;

}

void insert\_at\_beg(Car car)

{

Node\* newNode = new Node(car);

if (isEmpty())

{

front = rear = newNode;

}

else

{

newNode->next = front;

front->prev = newNode;

front = newNode;

}

cout << "Car " << car.getCarNum() << " is inserted at beginning" << endl;

}

void insert\_at\_end(Car car)

{

Node\* newNode = new Node(car);

if (isEmpty())

{

front = rear = newNode;

}

else

{

rear->next = newNode;

newNode->prev = rear;

rear = newNode;

}

cout << "Car " << car.getCarNum() << " is inserted at end" << endl;

}

void delete\_fr\_beg()

{

if (isEmpty())

{

cout << "Deque is empty" << endl;

}

else if (front == rear)

{

delete front;

front = rear = nullptr;

}

else

{

Node\* temp = front;

front = front->next;

front->prev = nullptr;

delete temp;

}

}

void delete\_fr\_rear()

{

if (isEmpty())

{

cout << "Deque is empty" << endl;

}

else if (front == rear)

{

delete rear;

front = rear = nullptr;

}

else

{

Node\* temp = rear;

rear = rear->prev;

rear->next = nullptr;

delete temp;

}

}

string search(int carNum)

{

Node\* current = front;

while (current != nullptr)

{

if (current->car.getCarNum() == carNum)

{

return current->car.getParkingLocation();

}

current = current->next;

}

return "Car not found";

}

};

int main()

{

Deque deque;

deque.insert\_at\_beg(Car(123, "CFD parking"));

deque.insert\_at\_end(Car(456, "CFD hostel parking"));

deque.insert\_at\_beg(Car(789));

cout << "Search results:" << endl;

cout << "Car 123: " << deque.search(123) << endl;

cout << "Car 456: " << deque.search(456) << endl;

cout << "Car 789: " << deque.search(789) << endl;

deque.delete\_fr\_beg();

deque.delete\_fr\_rear();

cout << "Search results after deletion:" << endl;

cout << "Car 123: " << deque.search(123) << endl;

cout << "Car 456: " << deque.search(456) << endl;

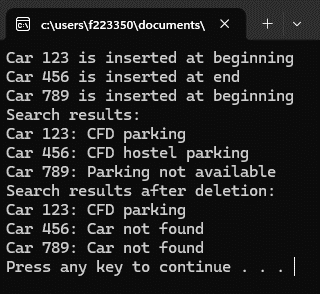
cout << "Car 789: " << deque.search(789) << endl;

system("pause");

return 0;

}

## Output



# Problem 03

#include <iostream>

using namespace std;

class Order

{

public:

string description;

double price;

};

class PizzaSystem

{

private:

Order \*orderArray;

int front;

int rear;

int maxOrders;

int count;

public:

PizzaSystem(int maxOrders)

: front(0), rear(-1), count(0), maxOrders(maxOrders)

{

orderArray = new Order[maxOrders];

}

void placeOrder(string description, double price)

{

if (count >= maxOrders)

{

cout << "Maximum order limit reached" << endl;

return;

}

rear = (rear + 1) % maxOrders;

orderArray[rear].description = description;

orderArray[rear].price = price;

count++;

cout << "Order placed successfully!" << endl;

}

double calculateBill()

{

double totalBill = 0.0;

int i = front;

int numOrders = count;

while (numOrders > 0)

{

totalBill += orderArray[i].price;

i = (i + 1) % maxOrders;

numOrders--;

}

return totalBill;

}

};

int main()

{

PizzaSystem pizzaSystem(10);

pizzaSystem.placeOrder("Pizza Margherita", 1000);

pizzaSystem.placeOrder("Pepperoni Pizza", 900);

pizzaSystem.placeOrder("Garlic Bread", 300);

double bill = pizzaSystem.calculateBill();

cout << "Total bill: Rs." << bill << endl;

system("pause");

return 0;

}

## Output

