National University of Computer and Emerging Sciences



Name: Muhammad Suleman

Roll #: 22F-3350

Section: BCS-4B

Lab # 07

# Problem 01

## Simple Queue

#include <iostream>

using namespace std;

class Queue

{

private:

int \*queueArray;

int front;

int rear;

int size;

public:

Queue(int size)

: front(-1), rear(-1), size(size)

{

queueArray = new int[size];

}

bool isFull()

{

return (rear == size - 1);

}

bool isEmpty()

{

return (front == -1 && rear == -1);

}

void enqueue(int value)

{

if (isFull())

{

cout << "Cannot enqueue value (Queue is full)" << endl;

return;

}

else if (isEmpty())

{

front = 0;

rear = 0;

}

else

{

rear++;

}

cout << "Enqueue value: " << value << endl;

queueArray[rear] = value;

}

int dequeue()

{

if (isEmpty())

{

cout << "Cannot dequeue value (Queue is empty)" << endl;

return -1;

}

else if (front == rear)

{

int value = queueArray[front];

front = -1;

rear = -1;

return value;

}

else

{

int value = queueArray[front];

front++;

return value;

}

}

void display()

{

if (isEmpty())

{

cout << "Nothing to display (Queue is empty)" << endl;

return;

}

cout << "\n->Displaying queue elements<-\n" << endl;

cout << "Queue elements: ";

for (int i = front; i <= rear; i++)

{

cout << queueArray[i] << " ";

}

cout << endl;

}

};

int main()

{

Queue queue(5);

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.enqueue(40);

queue.enqueue(50);

queue.enqueue(60);

queue.display();

cout << "Dequeued value: " << queue.dequeue() << endl;

cout << "Dequeued value: " << queue.dequeue() << endl;

queue.display();

system("pause");

return 0;

}

## Circular Queue

#include <iostream>

using namespace std;

class CircularQueue

{

private:

int \*queueArray;

int front;

int rear;

int size;

public:

CircularQueue(int size)

: front(-1), rear(-1), size(size)

{

queueArray = new int[size];

}

bool isFull()

{

return ((rear + 1) % size == front);

}

bool isEmpty()

{

return (front == -1 && rear == -1);

}

void enQueue(int value)

{

if (isFull())

{

cout << "Cannot enqueue value (Queue is full)" << endl;

return;

}

else if (isEmpty())

{

front = 0;

rear = 0;

}

else

{

rear = (rear + 1) % size;

}

cout << "Enqueue value: " << value << endl;

queueArray[rear] = value;

}

int deQueue()

{

if (isEmpty())

{

cout << "Cannot dequeue value (Queue is empty)" << endl;

return -1;

}

else if (front == rear)

{

int value = queueArray[front];

front = -1;

rear = -1;

return value;

}

else

{

int value = queueArray[front];

front = (front + 1) % size;

return value;

}

}

void display()

{

if (isEmpty())

{

cout << "Nothing to display (Queue is empty)" << endl;

return;

}

cout << "\n->Displaying queue elements<-\n" << endl;

cout << "Queue elements: ";

int i = front;

while (i != rear)

{

cout << queueArray[i] << " ";

i = (i + 1) % size;

}

cout << queueArray[rear] << endl;

}

};

int main()

{

CircularQueue queue(5);

queue.enQueue(10);

queue.enQueue(20);

queue.enQueue(30);

queue.enQueue(40);

queue.enQueue(50);

queue.enQueue(60);

queue.display();

cout << "Dequeued value: " << queue.deQueue() << endl;

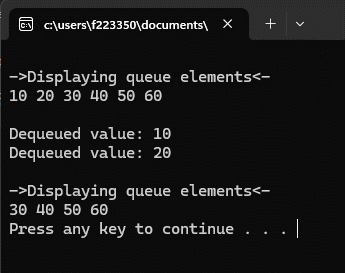
cout << "Dequeued value: " << queue.deQueue() << endl;

queue.display();

system("pause");

return 0;

}



## LinkedList Based (Dynamic)

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node()

: data(0), next(nullptr) {}

Node(int data)

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

};

class QueueLL

{

private:

Node\* front;

Node\* rear;

public:

QueueLL()

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

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;

}

}

int deQueue()

{

if (isEmpty())

{

cout << "Cannot dequeue value (Queue is empty)" << endl;

return -1;

}

Node\* temp = front;

int value = temp->data;

front = front->next;

if (front == nullptr)

{

rear = nullptr;

}

delete temp;

return value;

}

void display()

{

if (isEmpty())

{

cout << "Nothing to display (Queue is empty)" << endl;

return;

}

cout << "\n->Displaying queue elements<-" << endl;

Node\* current = front;

while (current != nullptr)

{

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

};

int main() {

QueueLL queue;

queue.enQueue(10);

queue.enQueue(20);

queue.enQueue(30);

queue.enQueue(40);

queue.enQueue(50);

queue.enQueue(60);

queue.display();

cout << "\nDequeued value: " << queue.deQueue() << endl;

cout << "Dequeued value: " << queue.deQueue() << endl;

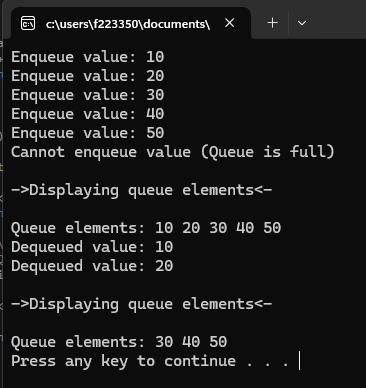
queue.display();

system("pause");

return 0;

}

### LinkedList Based Output



# Problem 02

#include <iostream>

using namespace std;

class CircularQueue

{

private:

int \*queueArray;

int front;

int rear;

int size;

public:

CircularQueue(int size)

: front(-1), rear(-1), size(size)

{

queueArray = new int[size];

}

bool isFull()

{

return ((rear + 1) % size == front);

}

bool isEmpty()

{

return (front == -1 && rear == -1);

}

void enQueue(int value)

{

if (isFull())

{

cout << "Cannot enqueue value (Queue is full)" << endl;

return;

}

else if (isEmpty())

{

front = 0;

rear = 0;

}

else

{

rear = (rear + 1) % size;

}

//cout << "Enqueue value: " << value << endl;

queueArray[rear] = value;

}

int deQueue()

{

if (isEmpty())

{

cout << "Cannot dequeue value (Queue is empty)" << endl;

return -1;

}

else if (front == rear)

{

int value = queueArray[front];

front = -1;

rear = -1;

return value;

}

else

{

int value = queueArray[front];

front = (front + 1) % size;

return value;

}

}

void display()

{

if (isEmpty())

{

cout << "Nothing to display (Queue is empty)" << endl;

return;

}

cout << "Queue elements: ";

int i = front;

while (i != rear)

{

cout << queueArray[i] << " ";

i = (i + 1) % size;

}

cout << queueArray[rear] << endl;

}

void rearrange()

{

if (isEmpty())

{

cout << "Cannot rearrange elements (Queue is empty)" << endl;

return;

}

int halfSize = size / 2;

CircularQueue extraQueue(halfSize);

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

{

extraQueue.enQueue(deQueue());

}

while (!extraQueue.isEmpty())

{

enQueue(extraQueue.deQueue());

enQueue(deQueue());

}

}

};

int main()

{

CircularQueue queue(8);

queue.enQueue(3);

queue.enQueue(2);

queue.enQueue(5);

queue.enQueue(6);

queue.enQueue(7);

queue.enQueue(9);

queue.enQueue(8);

queue.enQueue(4);

cout << "\n->Before arrangement<-\n" << endl;

queue.display();

cout << "\n->After arrangement<-\n" << endl;

queue.rearrange();

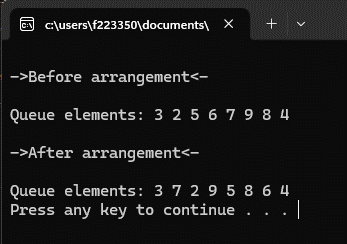
queue.display();

system("pause");

return 0;

}

## Output



# Problem 03

#include <iostream>

#include <string>

using namespace std;

class Player

{

public:

string name;

int ID;

int score;

string team;

int matchesPlayed;

Player()

{

name = "";

ID = 0;

score = 0;

team = "";

matchesPlayed = 0;

}

};

class Node

{

public:

Player player;

Node\* next;

Node(Player newPlayer)

{

player = newPlayer;

next = nullptr;

}

};

class Queue

{

private:

Node\* front;

Node\* rear;

int size;

public:

Queue()

{

front = nullptr;

rear = nullptr;

size = 0;

}

bool isEmpty()

{

return (front == nullptr);

}

void enqueue(Player player)

{

Node\* newNode = new Node(player);

if (isEmpty())

{

front = newNode;

rear = newNode;

}

else

{

rear->next = newNode;

rear = newNode;

}

size++;

}

void dequeue()

{

if (isEmpty())

{

cout << "Cannot dequeue player record (Queue is empty)" << endl;

return;

}

Node\* temp = front;

front = front->next;

delete temp;

size--;

}

Player frontPlayer()

{

if (isEmpty())

{

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

Player emptyPlayer;

return emptyPlayer;

}

return front->player;

}

void display()

{

if (isEmpty())

{

cout << "No player records to display (Queue is empty)" << endl;

return;

}

Node\* current = front;

while (current != nullptr)

{

Player player = current->player;

cout << "Name: " << player.name << endl;

cout << "ID: " << player.ID << endl;

cout << "Score: " << player.score << endl;

cout << "Team: " << player.team << endl;

cout << "Matches Played: " << player.matchesPlayed << endl;

cout << "---------------------------" << endl;

current = current->next;

}

}

bool isDuplicate(int ID)

{

Node\* current = front;

while (current != nullptr)

{

if (current->player.ID == ID)

{

return true;

}

current = current->next;

}

return false;

}

void insertPlayer()

{

Player newPlayer;

cout << "Enter player details:" << endl;

cout << "Name: ";

cin >> newPlayer.name;

do {

cout << "ID: ";

cin >> newPlayer.ID;

if (isDuplicate(newPlayer.ID))

{

cout << "Player with the same ID already exists, Please enter a unique ID" << endl;

}

} while (isDuplicate(newPlayer.ID));

cout << "Score: ";

cin >> newPlayer.score;

cout << "Team: ";

cin >> newPlayer.team;

cout << "Matches Played: ";

cin >> newPlayer.matchesPlayed;

enqueue(newPlayer);

cout << "Player record inserted successfully" << endl;

}

void deletePlayer()

{

if (isEmpty())

{

cout << "No player records to delete (Queue is empty)" << endl;

return;

}

dequeue();

cout << "Player record deleted successfully" << endl;

}

void updatePlayer()

{

if (isEmpty())

{

cout << "No player records to update (Queue is empty)" << endl;

return;

}

int updateID;

cout << "Enter the ID of the player record to update: ";

cin >> updateID;

Node\* current = front;

while (current != nullptr)

{

if (current->player.ID == updateID)

{

cout << "Enter updated player details:" << endl;

cout << "Name: ";

cin >> current->player.name;

cout << "Score: ";

cin >> current->player.score;

cout << "Team: ";

cin >> current->player.team;

cout << "Matches Played: ";

cin >> current->player.matchesPlayed;

cout << "Player record updated successfully" << endl;

return;

}

current = current->next;

}

cout << "Player record with the given ID not found" << endl;

}

void searchPlayer()

{

if (isEmpty())

{

cout << "No player records to search (Queue is empty)" << endl;

return;

}

int searchID;

cout << "Enter the ID of the player record to search: ";

cin >> searchID;

Node\* current = front;

while (current != nullptr)

{

while (current != nullptr)

{

if (current->player.ID == searchID)

{

cout << "Player record found:" << endl;

cout << "Name: " << current->player.name << endl;

cout << "ID: " << current->player.ID << endl;

cout << "Score: " << current->player.score << endl;

cout << "Team: " << current->player.team << endl;

cout << "Matches Played: " << current->player.matchesPlayed << endl;

return;

}

current = current->next;

}

cout << "Player record with the given ID not found." << endl;

}

}

};

int main()

{

Queue playerQueue;

int choice;

do {

cout << "\n----- Player Queue Menu -----" << endl;

cout << "1. Insert Player" << endl;

cout << "2. Delete Player" << endl;

cout << "3. Update Player" << endl;

cout << "4. Search Player" << endl;

cout << "5. Display Player Records" << endl;

cout << "6. Exit" << endl;

cout << "Enter your choice (1-6): ";

cin >> choice;

switch (choice)

{

case 1:

playerQueue.insertPlayer();

break;

case 2:

playerQueue.deletePlayer();

break;

case 3:

playerQueue.updatePlayer();

break;

case 4:

playerQueue.searchPlayer();

break;

case 5:

playerQueue.display();

break;

case 6:

cout << "Exiting program. Goodbye!" << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 6);

system("pause");

return 0;

}

## Output



# Problem 04

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node()

: data(0), next(nullptr) {}

Node(int data)

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

};

class QueueLL

{

private:

Node\* front;

Node\* rear;

public:

QueueLL()

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

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;

}

}

int deQueue()

{

if (isEmpty())

{

cout << "Cannot dequeue value (Queue is empty)" << endl;

return -1;

}

Node\* temp = front;

int value = temp->data;

front = front->next;

if (front == nullptr)

{

rear = nullptr;

}

delete temp;

return value;

}

void display()

{

if (isEmpty())

{

cout << "Nothing to display (Queue is empty)" << endl;

return;

}

cout << "\n->Displaying queue elements<-" << endl;

Node\* current = front;

while (current != nullptr)

{

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

bool checkConsecutiveTriplesAfterK(int k)

{

if (isEmpty() || front->next == nullptr)

{

return false;

}

Node\* prev = front;

Node\* curr = front->next;

while (curr && curr->next)

{

int count = 2;

bool check = false;

while (count--)

{

if (curr->data - prev->data != 1)

{

return false;

}

check = true;

prev = curr;

curr = curr->next;

}

int k1 = k;

while (k1-- && curr)

{

prev = curr;

curr = curr->next;

}

}

return true;

}

};

int main() {

QueueLL queue;

queue.enQueue(1);

queue.enQueue(2);

queue.enQueue(3);

queue.enQueue(5);

queue.enQueue(7);

queue.enQueue(11);

queue.enQueue(12);

queue.enQueue(13);

queue.display();

int k = 2;

if (queue.checkConsecutiveTriplesAfterK(k))

{

cout << "Yes" << endl;

}

else

{

cout << "No" << endl;

}

QueueLL queue2;

queue2.enQueue(1);

queue2.enQueue(2);

queue2.enQueue(3);

queue2.enQueue(4);

queue2.enQueue(5);

queue2.enQueue(12);

queue2.enQueue(14);

queue2.enQueue(15);

queue2.display();

if (queue2.checkConsecutiveTriplesAfterK(k))

{

cout << "Yes" << endl;

}

else

{

cout << "No" << endl;

}

system("pause");

return 0;

}

## Output

