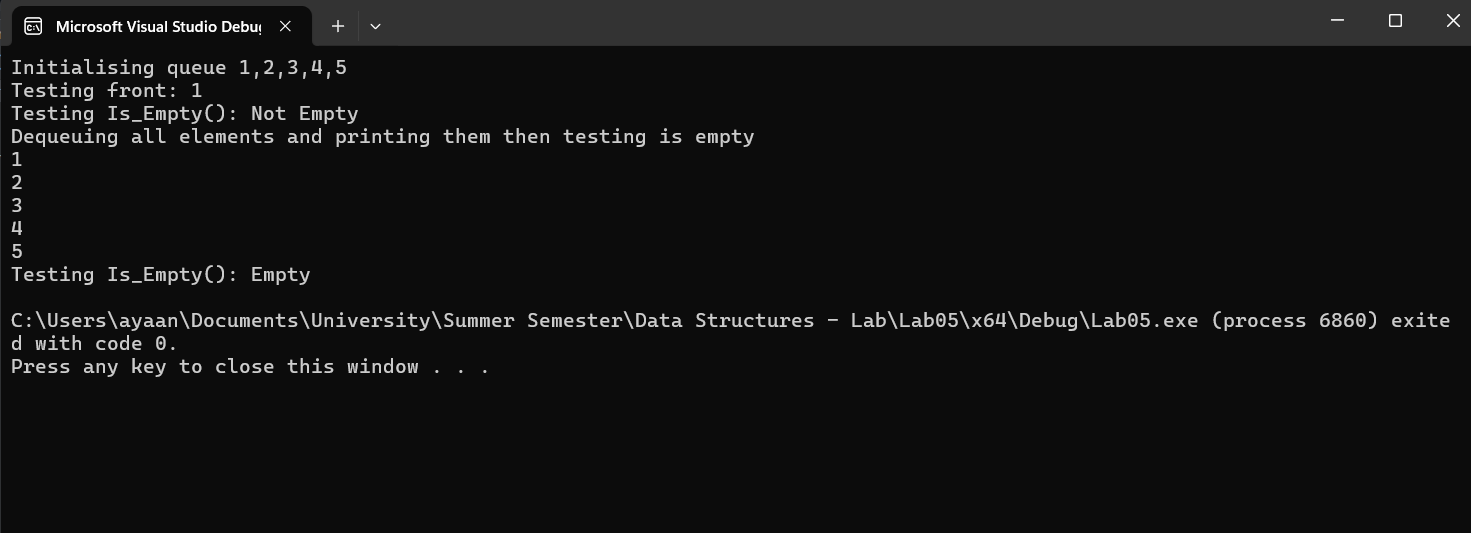
Q1:



#include<iostream>

using namespace std;

class Queue

{

public:

int\* arr;

int front;

int rear;

int size;

int count;

Queue()

{

size = 10;

arr = new int[size];

front = 0;

rear = -1;

count = 0;

}

Queue(int x)

{

size = x;

arr = new int[size];

front = 0;

rear = -1;

count = 0;

}

bool is\_empty()

{

if (count == 0)

{

return 1;

}

return 0;

}

bool is\_full()

{

if (count >= size)

{

return 1;

}

return 0;

}

void enqueue(int x)

{

if (is\_full())

{

cout << "Queue is full\n";

}

rear = (rear + 1) % size;

count++;

arr[rear] = x;

}

int dequeue()

{

if (is\_empty())

{

cout << "Empty queue\n";

return -1;

}

int final;

final = arr[front];

front = (front + 1) % size;

count--;

return final;

}

int Front()

{

if (!is\_empty())

{

return arr[front];

}

cout << "Empty queue\n";

}

};

int main()

{

cout << "Initialising queue 1,2,3,4,5\n";

Queue q1;

q1.enqueue(1);

q1.enqueue(2);

q1.enqueue(3);

q1.enqueue(4);

q1.enqueue(5);

cout << "Testing front: " << q1.Front() << endl;

cout << "Testing Is\_Empty(): ";

if (!q1.is\_empty())

cout << "Not Empty\n";

else

cout << "Empty\n";

cout << "Dequeuing all elements and printing them then testing is empty\n";

while (!q1.is\_empty())

{

cout << q1.dequeue() << endl;

}

cout << "Testing Is\_Empty(): ";

if (!q1.is\_empty())

cout << "Not Empty\n";

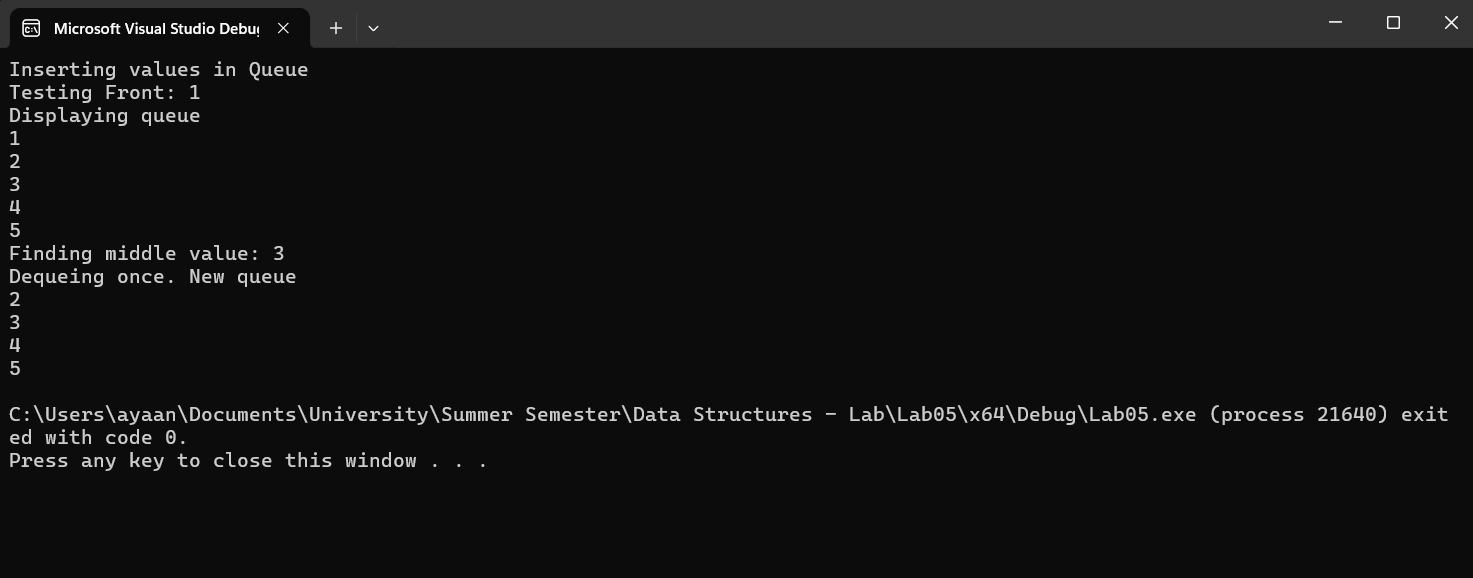
else

cout << "Empty\n";

return 0;

}

Q2:



template<class T>

class Node

{

public:

T data;

Node\* next;

Node()

{

data = 0;

next = NULL;

}

Node(T x)

{

data = x;

next = NULL;

}

void setData(T x)

{

data = x;

}

void setNext(Node\* ptr)

{

next = ptr;

}

T getData()

{

return data;

}

Node\* getNext()

{

return next;

}

};

template <typename T>

class Queue

{

private:

Node<T>\* front;

Node<T>\* rear;

Node<T>\* current;

int size;

public:

Queue()

{

front = NULL;

rear = NULL;

current = NULL;

size = 0;

}

bool is\_empty()

{

if (front == NULL)

{

return 1;

}

return 0;

}

void enqueue(T item)

{

Node<T>\* ptr = new Node<T>(item);

if (is\_empty())

{

front = ptr;

rear = ptr;

return;

}

rear->setNext(ptr);

rear = ptr;

size++;

}

Queue(Queue<T>& ptr)

{

Node<T>\* temp = ptr.front;

front = NULL;

rear = NULL;

current = NULL;

size = ptr.size;

while (temp != NULL)

{

enqueue(temp->data);

temp = temp->next;

}

}

Queue& operator=(const Queue<T>& q1)

{

if (this != &q1)

{

clear();

Node<T>\* temp = q1.front;

while (temp != nullptr)

{

enqueue(temp->data);

temp = temp->next;

}

}

return \*this;

}

T Front()

{

if (is\_empty())

return -1;

return front->getData();

}

int size2()

{

if (is\_empty()) {

return 0;

}

int count = 0;

Node<T>\* current = front;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

T dequeue()

{

if (is\_empty())

{

return -1;

}

T item\_temp;

Node<T>\* temp = front;

item\_temp = temp->getData();

front = front->next;

delete temp;

return item\_temp;

}

void clear()

{

if (is\_empty())

return;

while (!is\_empty())

{

dequeue();

}

}

void display()

{

Node<T>\* ptr = front;

while (!is\_empty())

{

cout << Front() << endl;

front = front->next;

}

front = ptr;

}

T return\_middle()

{

int size3 = size / 2;

int count = 0;

while (count < size3)

{

count++;

dequeue();

}

return Front();

}

};