**Practical – 2**

AIM: Implementation and time analysis of Queue and its applications .

#include <iostream>

using namespace std;

void insert(int queue[], int data, int size, int \*front, int \*rear)

{

if (\*rear == size - 1)

{

cout << "Queue is overflow " << endl << endl;

return;

}

++(\*rear);

queue[\*rear] = data;

if (\*front == -1)

{

\*front = 0;

}

}

void dequeue(int queue[], int size, int \*front, int \*rear)

{

if (\*front == -1)

{

cout << "Queue is underflow!" << endl << endl;

return;

}

if (\*front == \*rear)

{

int temp = queue[(\*front)];

\*front = -1;

\*rear = -1;

cout << temp << " successfully deleted" << endl << endl;

return;

}

int temp = queue[(\*front)];

++(\*front);

cout << temp << " successfully deleted" << endl << endl;

}

void display(int queue[], int \*front, int \*rear)

{

if (\*rear == -1)

{

cout << "Null queue" << endl << endl;

return;

}

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

{

cout << queue[i] << " ";

}

cout << endl << endl;

}

int main()

{

int queue[10] = {};

int front = -1;

int rear = -1;

int size = sizeof(queue) / sizeof(queue[0]);

int choice, data;

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

cout << "1) Show Queue data " << endl;

cout << "2) Insert in queue " << endl;

cout << "3) Delete in queue " << endl;

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

do

{

cout << "Enter choice: ";

cin >> choice;

switch (choice)

{

case 1:

display(queue, &front, &rear);

break;

case 2:

cout << "Enter Data: ";

cin >> data;

insert(queue, data, size, &front, &rear);

break;

case 3:

dequeue(queue, size, &front, &rear);

break;

default:

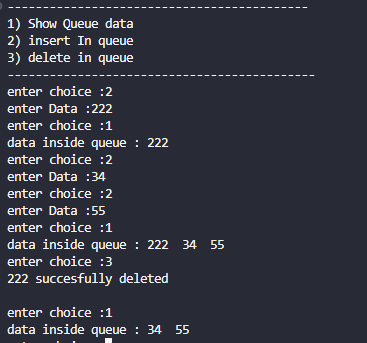
break;

}

} while (choice != 8);

}

**OUTPUT**



Circular Queue

#include <iostream>

using namespace std;

int front = -1;

int rear = -1;

int queue[10];

int size = sizeof(queue) / sizeof(queue[0]);

void insert(int data) {

if (rear == size - 1 && front == 0 || front != 0 && rear + 1 == front) {

cout << "Queue is overflow..";

return;

}

if (front != 0 && rear == size - 1) {

rear = 0;

return;

}

++rear;

queue[rear] = data;

if (front == -1) {

front = 0;

}

}

void display() {

cout << "queue -> ";

if (rear >= front) {

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

cout << queue[i] << " | ";

}

} else {

}

cout << endl;

}

void Delete() {

if (rear == -1) {

cout << "queue underflow" << endl;

return;

} else if (rear == 0 && front == size - 1) {

cout << "queue underflow" << endl << endl;

}

if (rear != 0 && front > rear) {

cout << "queue underflow 2 if " << endl << endl;

}

cout << queue[front] << "successfully deleted" << endl;

if (front == size - 1) {

front = 0;

}

if (front == rear) {

front = 0;

rear = 0;

}

if (front < rear) {

cout << "front ++ happenl";

++front;

}

}

int main() {

int choice, data;

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

cout << "1) Show Queue data " << endl;

cout << "2) Insert in queue " << endl;

cout << "3) Delete in queue " << endl;

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

do {

cout << "enter choice :";

cin >> choice;

switch (choice) {

case 1:

display();

break;

case 2:

cout << "enter Data :";

cin >> data;

insert(data);

break;

case 3:

Delete();

break;

default:

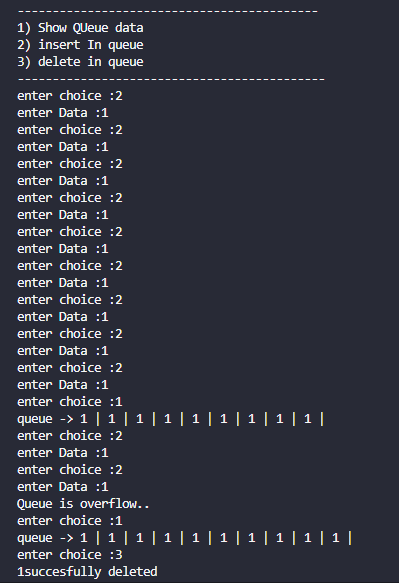
break;

}

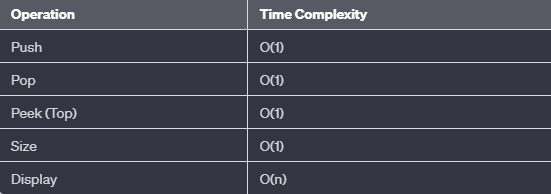
} while (choice != 8);

}

**OUTPUT**



Time analysis



Applications

* Job Scheduling
* Breadth-First Search (BFS)
* Print Queue Management
* Task Management in Operating Systems
* Request Handling in Web Servers
* Order Processing in Business
* Buffer Management
* Call Center Systems
* Banks and Customer Service Centers
* Data Buffering
* Binary Trees Level Order Traversal
* Bounded Buffer Problem
* Task Synchronization
* Event Handling in GUIs