**Lab-03: Implementation of Queue using Array**

**Theory**

Queue data structure is a linear data structure that follows FIFO (First In First Out) principle, so the first element inserted is the first to be popped out.

**Basic operations in Queue data structure**

1. **Enqueue:** adds (or stores) an element to the end of the queue
2. **Dequeue:** Removal of elements from the queue
3. **Peek or front:** Acquires the data element available at the front node of the queue without deleting it.
4. **rear:** This operation returns the element available at the front node of the queue without deleting it.
5. **isFull:** Validates if the queue is full
6. **isEmpty:** Checks if the queue is empty

**Algorithm for Enqueue operation:**

1. Check if the queue is full
2. If the queue is full, return overflow error and exit
3. If the queue is not full, increment the rear pointer to point to the next empty space.
4. Add the data element to the queue location, where the rear is pointing
5. return success

**Algorithm for Dequeue operation:**

1. Check if the queue is empty
2. If the queue is empty, return the underflow error and exit
3. If the queue is not empty, access the data where the front is pointing
4. Increment the front pointer to point to the next available data element
5. return success

**Source Code**

**Lab-03: Implementation of Queue using Array**

**Theory**

Queue data structure is a linear data structure that follows FIFO (First In First Out) principle, so the first element inserted is the first to be popped out.

**Basic operations in Queue data structure**

1. **Enqueue:** adds (or stores) an element to the end of the queue
2. **Dequeue:** Removal of elements from the queue
3. **Peek or front:** Acquires the data element available at the front node of the queue without deleting it.
4. **rear:** This operation returns the element available at the front node of the queue without deleting it.
5. **isFull:** Validates if the queue is full
6. **isEmpty:** Checks if the queue is empty

**Algorithm for Enqueue operation:**

1. Check if the queue is full
2. If the queue is full, return overflow error and exit
3. If the queue is not full, increment the rear pointer to point to the next empty space.
4. Add the data element to the queue location, where the rear is pointing
5. return success

**Algorithm for Dequeue operation:**

1. Check if the queue is empty
2. If the queue is empty, return the underflow error and exit
3. If the queue is not empty, access the data where the front is pointing
4. Increment the front pointer to point to the next available data element
5. return success

**Source Code**

#include<stdio.h>

#include<stdbool.h>

#include<stdlib.h>

#define MAX 10

int queue[MAX];

int front = -1;

int rear = -1;

bool isEmp(){

if(front ==-1){

return true;

}

else{

return false;

}

}

bool isFull(){

if(front==MAX-1){

return true;

}

else{

return false;

}

}

void enqueue(int num) {

if (!isFull()) {

if (isEmp()) { // If the queue is initially empty

front = 0;

}

rear++;

queue[rear] = num;

printf("Enqueued %d successfully!\n", num);

} else {

printf("Couldn't insert, queue is full!\n");

}

}

int dequeue() {

int num;

if (!isEmp()) {

num = queue[front];

if (front == rear) { //case when there is only one element in the queue

front = -1;

rear = -1;

} else {

front++;

}

printf("Dequeued %d successfully!\n", num);

return num;

} else {

printf("Couldn't dequeue, queue is empty!\n");

return -1;

}

}

int main() {

int choice, num;

while (1) { // Infinite loop, exit when choice is 0

printf("\nMenu:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display Queue\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the number to be enqueued: ");

scanf("%d", &num);

enqueue(num);

break;

case 2:

dequeue();

break;

case 3:

if (isEmp()) {

printf("Queue is empty!\n");

} else {

printf("Queue elements are:\n");

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

printf("%d ", queue[i]);

}

printf("\n");

}

break;

case 4:

printf("Exiting the program.\n");

return 0; // Exit the program

default:

printf("Invalid choice, please try again.\n");

break;

}

}

}

**Discussion and Conclusion**

From this lab activity, we got the basic idea about the queue operation in data structure. We learned about the basic operations that can be performed on queue using array like rear, enqueue, dequeue, isEmpty, isFull and peek. We learned that the queue follows FIFO(First In First Out) principle. This implementation of the queue in C using arrays demonstrates the main functionality of that data structure. Besides being the time efficient operations (constant time enqueue and dequeue), it has a fixed size array, which becomes a problem in dynamic situations where the number of elements is unknown.

**Discussion and Conclusion**

From this lab activity, we got the basic idea about the queue operation in data structure. We learned about the basic operations that can be performed on queue using array like rear, enqueue, dequeue, isEmpty, isFull and peek. We learned that the queue follows FIFO(First In First Out) principle. This implementation of the queue in C using arrays demonstrates the main functionality of that data structure. Besides being the time efficient operations (constant time enqueue and dequeue), it has a fixed size array, which becomes a problem in dynamic situations where the number of elements is unknown.

Name: Biraj Pandey

Roll no: ACE079BCT023