THEORY – 5

PROGRAM -1

AIM- QUEUE OPERATIONS

|  |
| --- |
| A queue is a linear data structure that follows the First In, First Out (FIFO) principle, meaning that the element added first to the queue is the one that gets removed first. It resembles a real-life queue where people line up in a sequential manner and the first person to enter is the first person to exit. In programming, queues are commonly used for implementing tasks such as handling requests, managing resources, and maintaining the order of operations.  Here are the basic operations associated with a queue:  1. \*\*Enqueue (Insertion)\*\*:  - This operation adds an element to the rear (end) of the queue.  - If the queue is full, it results in a "Queue Overflow" condition.  2. \*\*Dequeue (Deletion)\*\*:  - This operation removes an element from the front (beginning) of the queue.  - If the queue is empty, it results in a "Queue Underflow" condition.  3. \*\*Peek (Front)\*\*:  - This operation retrieves the element at the front of the queue without removing it.  - It's useful for checking the next element to be dequeued.  - If the queue is empty, it returns a special value (e.g., -1) to indicate an empty queue.  4. \*\*IsEmpty\*\*:  - This operation checks whether the queue is empty or not.  - It returns `true` if the queue is empty; otherwise, it returns `false`.  5. \*\*IsFull\*\*:  - This operation checks whether the queue is full or not.  - It returns `true` if the queue is full; otherwise, it returns `false`.  Now, let's summarize these operations in a typical queue implementation:  ### Queue Operations in C (Array-based Implementation)  ```c  #include <stdio.h>  #define MAXSIZE 10 // Maximum size of the queue  int queue[MAXSIZE];  int front = 0; // Front of the queue  int rear = -1; // Rear of the queue  // Function to check if the queue is empty  int isEmpty() {  return front > rear;  }  // Function to check if the queue is full  int isFull() {  return rear == MAXSIZE - 1;  }  // Function to add an element to the queue (Enqueue)  void enqueue(int data) {  if (isFull()) {  printf("Queue Overflow\n");  } else {  rear++;  queue[rear] = data; // Add the element to the rear of the queue  }  }  // Function to remove an element from the queue (Dequeue)  int dequeue() {  if (isEmpty()) {  printf("Queue Underflow\n");  return -1; // Return a special value to indicate underflow  } else {  int data = queue[front]; // Get the front element  front++; // Increment front to remove the element  return data; // Return the dequeued element  }  }  // Function to retrieve the element at the front of the queue (Peek)  int peek() {  if (isEmpty()) {  printf("Queue is empty\n");  return -1; // Return a special value to indicate empty queue  } else {  return queue[front]; // Return the front element  }  }  // Function to display all elements in the queue  void display() {  if (isEmpty()) {  printf("Queue is empty\n");  } else {  printf("Queue elements are:\n");  for (int i = front; i <= rear; i++) {  printf("%d ", queue[i]);  }  printf("\n");  }  }  int main() {  // Enqueue elements to the queue  enqueue(10);  enqueue(20);  enqueue(30);  // Display the queue elements  display();  // Perform dequeue operation  int dequeuedValue = dequeue();  if (dequeuedValue != -1) {  printf("Dequeued value: %d\n", dequeuedValue);  }  // Display the front element using peek  int frontValue = peek();  if (frontValue != -1) {  printf("Front element: %d\n", frontValue);  }  return 0;  }  ```  In this C program:  - We define a queue using an integer array `queue` with a fixed size of `MAXSIZE`.  - The `front` variable indicates the index of the front element of the queue, and `rear` indicates the index of the rear element.  - The `isEmpty` function checks whether the queue is empty by comparing `front` with `rear`.  - The `isFull` function checks whether the queue is full by comparing `rear` with `MAXSIZE - 1`.  - The `enqueue` function adds elements to the queue at the rear if the queue is not full.  - The `dequeue` function removes elements from the queue from the front if the queue is not empty.  - The `peek` function retrieves the element at the front of the queue without removing it.  - The `display` function displays all elements of the queue from front to rear.  This implementation provides the basic functionalities of a queue using an array-based approach in C. You can modify and extend this code according to your requirements. |