|  |  |
| --- | --- |
| **Assignment No: 5** | |
| **Aim:** | Develop a C++ program to implement a queue using an array. The program should include basic queue operations: enqueue, dequeue, and display. |
| **Objective:** | The objective of this assignment is to understand how to implement a queue data structure using an array. By the end of this assignment, students will gain insights into static memory allocation, array manipulation, and queue operations (FIFO - First In First Out). |
| **Theory:** | A queue is a linear data structure that follows the First In, First Out (FIFO) principle. This means that elements are inserted at the rear (enqueue) and removed from the front (dequeue). In a queue implemented using an array:  - Enqueue (Insertion): Adds an element to the rear of the queue.  - Dequeue (Removal): Removes an element from the front of the queue.  - Display: Displays all the elements currently in the queue.  Queue Operations in an Array:  - Enqueue: Before adding a new element, we check if the queue is full. If it is not full, the new element is added at the rear of the queue.  - Dequeue: Before removing an element, we check if the queue is empty. If it is not empty, the element at the front is removed, and the front index is incremented.  - Display: We traverse the queue from the front to the rear and display all the elements.  Advantages of Queue Using Array:  1. Fixed Size: Easy to implement as the size of the array is predefined.  2. Time Efficient: Accessing elements in an array is fast (constant time complexity).    Limitations of Queue Using Array:  1. Static Memory Allocation: The size of the queue is fixed and cannot grow dynamically.  2. Queue Overflow: If the queue becomes full, no more elements can be added even if there is unused space at the front due to dequeuing.  Applications of Queue:  - Managing processes in CPU scheduling.  - Printer task scheduling.  - Asynchronous data handling (e.g., keyboard input).  - Breadth-first search in graphs. |
| **Algorithm:** | 1. Enqueue Operation:  1. Start.  2. Check if the queue is full (rear == size - 1).  - If full, print "Queue Overflow" and exit.  3. If the queue is not full:  - Increment the rear index.  - Insert the new element at `queue[rear]`.  4. If the front index is -1, set front to 0.  5. End.  2. Dequeue Operation:  1. Start.  2. Check if the queue is empty (front == -1 or front > rear).  - If empty, print "Queue is empty" and exit.  3. If the queue is not empty:  - Remove the element from `queue[front]`.  - Increment the front index.  - If front exceeds rear, reset front and rear to -1 (queue becomes empty).  4. End.  3. Display Operation:  1. Start.  2. Check if the queue is empty (front == -1).  - If empty, print "Queue is empty" and exit.  3. If not empty:  - Traverse the queue from `queue[front]` to `queue[rear]` and display each element.  4. End. |
| **Program:** | #include <iostream>  using namespace std;  #define MAX 100  class Queue {  private:  int queue[MAX];  int front, rear;  public:  Queue() {  front = -1;  rear = -1;  }    void enqueue(int value) {  if (rear == MAX - 1) {  cout << "Queue Overflow" << endl;  return;  }  if (front == -1) {  front = 0;  }  rear++;  queue[rear] = value;  cout << value << " enqueued into the queue." << endl;  }  void dequeue() {  if (front == -1 || front > rear) {  cout << "Queue is empty" << endl;  return;  }  cout << queue[front] << " dequeued from the queue." << endl;  front++;  if (front > rear) {  front = -1;  rear = -1;  }  }  void display() {  if (front == -1) {  cout << "Queue is empty" << endl;  return;  }  cout << "Queue elements: ";  for (int i = front; i <= rear; i++) {  cout << queue[i] << " ";  }  cout << endl;  }  };  int main() {  Queue q;  int choice, value;  do {    cout << "Menu:" << endl;  cout << "1. Enqueue" << endl;  cout << "2. Dequeue" << endl;  cout << "3. Display" << endl;  cout << "4. Exit" << endl;  cout << "Enter your choice: ";  cin >> choice;  switch (choice) {  case 1:  cout << "Enter value to enqueue: ";  cin >> value;  q.enqueue(value);  break;  case 2:  q.dequeue();  break;  case 3:  q.display();  break;  case 4:  cout << "Exiting..." << endl;  break;  default:  cout << "Invalid choice. Please try again." << endl;  }  } while (choice != 4);  return 0;  } |
| **Output:** | Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 1  Enter value to enqueue: 10  10 enqueued into the queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 1  Enter value to enqueue: 20  20 enqueued into the queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 3  Queue elements: 10 20  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 2  10 dequeued from the queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 3  Queue elements: 20  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 2  20 dequeued from the queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 3  Queue is empty  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 4  Exiting... |
| **Conclusion:** | |
| **Date:** | |
| **Staff Sign:** | |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***END**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |