|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Data Structures Laboratory (using C/C++)** | **Semester:** | **III** |
| **Date of Performance:** |  | **Batch No:** | **A - 3** |
| **Faculty Name:** | **Prof. Om Goswami** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 4**

**Title: Queue**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **To understand the concept of queue.**  **Write a program for Circular/Priority/Double Ended queue using Linked list.**  **Given A [] = {11,33,55,10,66}, perform enqueue, dequeue operations and display queue contents.** |

|  |
| --- |
| **COs to be achieved:** |
| **CO1:** Understand and implement the different data structures used in problem solving.  **CO2:** Apply linear and non-linear data structure in application development. |

|  |
| --- |
| **Books/Journals/Websites referred:** |
| 1. Data Stuctures by Reema Thareja 2. [Introduction to Circular Queue - GeeksforGeeks](https://www.geeksforgeeks.org/introduction-to-circular-queue/) |

|  |
| --- |
| **Tools required:** |
| DEV C/C++ compiler/ Code blocks C compiler |

|  |
| --- |
| **Theory:** |
| A queue is a first-in, first-out (FIFO) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the rear and removed from the other end called the front. Like stacks, queues can be implemented by using either arrays or linked lists. |

|  |
| --- |
| **Implementation details:** |
| 1. **Enlist all the Steps followed and various options explored.**  * **Data Structure Selection**: The first step was to decide which data structure to implement. A circular queue was chosen because it efficiently manages a fixed-size collection of elements. * **Defining the Node Structure**: A struct Node was defined to represent the elements of the circular queue. Each node contains an integer value and a pointer to the next node. * **Function Definitions**: Functions for enqueue, dequeue, and display operations were defined. These functions were designed to work with the circular structure of the queue. * **Main Function**: The main function was created to demonstrate the circular queue operations using an example array A []. The elements from the array were enqueued into the circular queue, and then dequeue operations were performed. * **Display Function**: A display function was added to visualize the contents of the circular queue at various points during the program execution.  1. **Explain your program logic and methods used.**  * **Enqueue Operation**: The enqueue function creates a new node and inserts it at the rear of the circular queue. If the queue is empty, both front and rear pointers are set to the new node. Otherwise, the rear pointer is updated to the new node, and the new node is linked to the front to maintain the circular structure. * **Dequeue Operation**: The dequeue function removes the element from the front of the circular queue and updates the front pointer. If the queue becomes empty after dequeue, both front and rear pointers are set to NULL. * **Display Operation**: The display function traverses the circular queue from front to rear and prints the values of all elements, taking into account the circular structure.  1. **Explain the Importance of the approach followed by you.**  * **Circular Queue Usage**: The program demonstrates the use of a circular queue, which is valuable for situations where you need to efficiently manage and cycle through a fixed-size collection of elements. Examples include scheduling tasks in a round-robin fashion or implementing a buffer with limited memory. * **Dynamic Memory Allocation**: The program showcases dynamic memory allocation for nodes, which is essential for managing memory efficiently when working with data structures like queues. * **User Interaction**: The program allows the user to input expressions, enqueue them, and dequeue them, providing a practical example of interacting with a circular queue. * **Error Handling**: The program includes error handling for situations such as attempting to dequeue from an empty queue or dividing by zero. |

|  |
| --- |
| **C/C++ Code implemented:** |
| **Program for circular queue.**  // CIRCULAR QUEUE  #include <stdio.h>  #include <stdlib.h>  // node in circular queue  struct Node {      int data;      struct Node\* next;  };  struct Node\* front = NULL;  struct Node\* rear = NULL;  // enqueue  void enqueue(int item) {      struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));      newNode->data = item;      newNode->next = NULL;      if (front == NULL) {          front = newNode;      } else {          rear->next = newNode;      }      rear = newNode;      rear->next = front; // to make it circular  }  // dequeue  int dequeue() {      if (front == NULL) {          printf("Queue is empty. Cannot dequeue.\n");          return -1;      }      int item = front->data;      struct Node\* temp = front;      if (front == rear) {          front = rear = NULL;      } else {          front = front->next;          rear->next = front; // to make it circular      }      free(temp);      return item;  }  // display  void display() {      if (front == NULL) {          printf("Queue is empty\n");          return;      }      struct Node\* temp = front;      do {          printf("%d ", temp->data);          temp = temp->next;      } while (temp != front);      printf("\n");  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050\n");      int A[] = {11, 33, 55, 10, 66};      int n = sizeof(A) / sizeof(A[0]);      printf("\nEnqueueing elements into the circular queue:\n");      for (int i = 0; i < n; i++) {          enqueue(A[i]);      }      printf("Queue contents: ");      display();      printf("\nDequeuing elements from the circular queue:\n");      while (front != NULL) {          int item = dequeue();          printf("Dequeued: %d\n", item);      }      printf("\nQueue contents: ");      display();      return 0;  } |

|  |
| --- |
| **Output/ program results after execution:** |
|  |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **Write applications of the queue.**   Queues are a fundamental data structure used in various applications where data needs to be managed in a specific order. Some common applications of queues include:   * 1. **Task Scheduling:** Queues can be used to schedule tasks based on priority or the order in which they were received.   2. **Traffic Management:** Queues can be used to manage traffic flow in transportation systems, such as airport control systems or road networks.   3. **Operating systems:** Operating systems often use queues to manage processes and resources. For example, a process scheduler might use a queue to manage the order in which processes are executed.   4. **Network protocols:** Network protocols like TCP and UDP use queues to manage packets that are transmitted over the network. Queues can help to ensure that packets are delivered in the correct order and at the appropriate rate.   5. **Printer queues:** In printing systems, queues are used to manage the order in which print jobs are processed. Jobs are added to the queue as they are submitted, and the printer processes them in the order they were received.   6. **Web servers:** Web servers use queues to manage incoming requests from clients. Requests are added to the queue as they are received, and they are processed by the server in the order they were received.   7. **Breadth-first search algorithm:** The breadth-first search algorithm uses a queue to explore nodes in a graph level-by-level. The algorithm starts at a given node, adds its neighbors to the queue, and then processes each neighbor in turn.   These are just a few examples of the many applications of queues in various domains, emphasizing their importance in managing data and tasks in an orderly and controlled manner. Queues are a fundamental concept in computer science and play a crucial role in solving a wide range of problems efficiently. |

|  |
| --- |
| **Conclusion:** |
| In conclusion, through the experimentation with queue data structures and the development of programs for Circular, Priority, and Double-Ended queues using linked lists, we gained understanding of queue concepts and their practical applications in real-world scenarios. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |