|  |  |
| --- | --- |
| **Assignment No: 7** | |
| **Aim:** | Develop a C++ program to implement a queue using a linked list. 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 a linked list. By the end of this assignment, students will have a thorough understanding of dynamic memory allocation, linked lists, and queue operations (FIFO - First In First Out). |
| **Theory:** | A queue is a 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 using a linked list:  - 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.  drawbacks of implementing a queue using an array compared to using a linked list:  1. Fixed Size:  - In an array-based implementation, the size of the queue is fixed. Once the array is full, you can't add more elements unless you resize the array, which can be inefficient.  - In a linked list, the queue can grow dynamically, allowing for flexible size management.  2. Inefficient Memory Utilization:  - Even if there are dequeued elements at the front of the array, that memory is not reused unless you perform a circular queue implementation or shift the elements, which can be costly.  - In a linked list, memory is allocated as needed, and deallocated when elements are removed, making better use of memory.  3. Shifting Elements (Without Circular Queue):  - In a basic array queue (non-circular), when you dequeue elements, you may need to shift the remaining elements forward, which is time-consuming and inefficient for large queues.  - Linked lists do not require shifting, as elements can be removed or added by simply updating pointers.  4. Wasted Memory:  - If the array is oversized (to prevent overflow), unused slots can lead to wasted memory.  - Linked lists only use as much memory as necessary for the elements they hold.  5. Resizing Overhead:  - If the queue grows beyond the array’s size, you need to reallocate a larger array and copy existing elements, which can cause overhead.  - Linked lists avoid resizing altogether since they can grow dynamically.  These drawbacks make the linked list implementation more flexible, especially when the queue size is not known in advance or varies frequently.  A linked list is a dynamic data structure where each node contains:  - A data part that stores the element.  - A pointer (or reference) to the next node.  Advantages of Queue using Linked List:  1. Dynamic Memory Allocation: Unlike arrays, linked lists do not require a predefined size. The queue size grows dynamically as needed.  2. Efficient Memory Usage: Nodes are created and deleted dynamically, optimizing memory usage.  3. No Overflow Condition: As long as memory is available, a queue implemented with a linked list does not overflow.  Applications of Queue:  - CPU scheduling.  - I/O buffer management.  - Task scheduling in operating systems.  - Handling asynchronous data (like input from a keyboard or mouse). |
| **Algorithm:** | **1. Enqueue Operation:**  1. Start.  2. Create a new node.  3. Insert the data into the node.  4. If the queue is empty (i.e., head is NULL):  - Set both head and tail to the new node.  5. Otherwise:  - Attach the new node to the `next` of the tail.  - Update the tail to point to the new node.  6. End.  **2. Dequeue Operation:**  1. Start.  2. If the queue is empty (i.e., head is NULL):  - Print "Queue is empty".  3. Otherwise:  - Remove the element from the front (head).  - Move the head to the next node.  4. Free the memory of the dequeued node.  5. End.  **3. Display Operation:**  1. Start.  2. If the queue is empty (i.e., head is NULL):  - Print "Queue is empty".  3. Otherwise:  - Traverse from head to tail and print each element.  4. End. |
| **Program:** | #include <iostream>  using namespace std;  struct Node {  int data;  Node\* next;  };  class Queue {  private:  Node\* head;  Node\* tail;  public:  Queue() : head(nullptr), tail(nullptr) {}  void enqueue(int value) {  Node\* newNode = new Node();  newNode->data = value;  newNode->next = nullptr;  if (head == nullptr) {  head = tail = newNode;  } else {  tail->next = newNode;  tail = newNode;  }  cout << value << " enqueued to queue." << endl;  }    void dequeue() {  if (head == nullptr) {  cout << "Queue is empty." << endl;  return;  }  Node\* temp = head;  cout << head->data << " dequeued from queue." << endl;  head = head->next;  if (head == nullptr) {  tail = nullptr;  }  delete temp;  }  void display() {  if (head == nullptr) {  cout << "Queue is empty." << endl;  return;  }  Node\* current = head;  cout << "Queue elements: ";  while (current != nullptr) {  cout << current->data << " ";  current = current->next;  }  cout << endl;  }  ~Queue() {  while (head != nullptr) {  dequeue();  }  }  };  int main() {  Queue q;  int choice, value;  do {  cout << "\nMenu:\n";  cout << "1. Enqueue\n";  cout << "2. Dequeue\n";  cout << "3. Display\n";  cout << "4. Exit\n";  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 program." << 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 to queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 1  Enter value to enqueue: 20  20 enqueued to queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 1  Enter value to enqueue: 30  30 enqueued to queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 3  Queue elements: 10 20 30  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 2  10 dequeued from queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 3  Queue elements: 20 30  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 2  20 dequeued from queue.  Menu:  1. Enqueue  2. Dequeue  3. Display  4. Exit  Enter your choice: 2  30 dequeued from 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 program. |
| **Conclusion:** | |
| **Date:** | |
| **Staff Sign:** | |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***END**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |