Batch:	**D3**	**Roll No.: 16010123294**

Experiment / assignment / tutorial No. 5

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

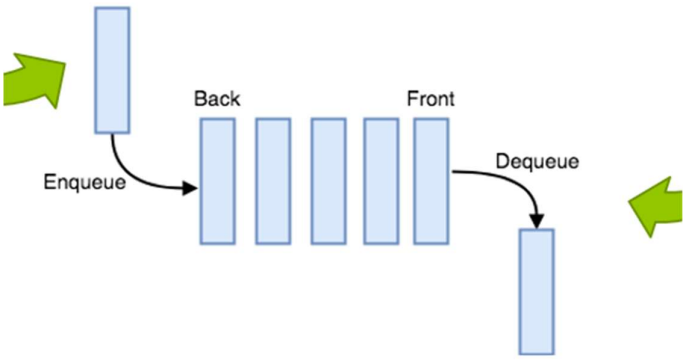
Title: Implementation of Queue operations (Static and Dynamic implementation)- Queue, circular queue, priority queue, and deque

Objective: To implement Basic Operations of Queues

Expected Outcome of Experiment:

CO	Outcome
2	Apply linear and non-linear data structure in application development.

Introduction:



Program source code:

```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int priority;
   int data;
   struct node *next;
};

struct node* newNode(int data, int priority) {
   struct node* temp = (struct node*)malloc(sizeof(struct node));
   temp->data = data;
   temp->priority = priority;
   temp->next = NULL;
   return temp;
}

void dequeue(struct node** head) {
   if (*head == NULL) {
      printf("Queue is empty\n");
      return;
   }
   struct node* temp = *head;
   *head = (*head)->next;
   free(temp);
}

void enqueue(struct node** head, int data, int priority) {
   struct node* temp = newNode(data, priority);

   if (*head == NULL || (*head)->priority > priority) {
      temp->next = *head;
      *head = temp;
   } else {
      struct node* start = *head;
      while (start->next != NULL && start->next->priority <= priority) {
         start = start->next;
      }
      temp->next = start->next;
```

```
        start->next = temp;
    }
}

void display(struct node *head) {
    if (head == NULL) {
        printf("Queue is empty\n");
        return;
    }
    struct node* temp = head;
    printf("Priority Queue: \n");
    while (temp != NULL) {
        printf("Data: %d, Priority: %d\n", temp->data, temp->priority);
        temp = temp->next;
    }
}

int main() {
    struct node* pq = NULL;
    int choice, data, priority;

    while (1) {
        printf("\nPriority Queue Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Data: ");
                scanf("%d", &data);
                printf("Enter Priority: ");
                scanf("%d", &priority);
                enqueue(&pq, data, priority);
                break;
            case 2:
                dequeue(&pq);
```

```
            break;
        case 3:
            display(pq);
            break;
        case 4:
            printf("Exiting\n");
            while (pq != NULL) {
                dequeue(&pq);
            }
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
```

Output Screenshots:

Enqueue & Display:

```
Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice:
1
Enter Data: 1
Enter Priority: 5

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter Data: 2
Enter Priority: 4

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
```

```
Enter your choice: 1
Enter Data: 3
Enter Priority: 3

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter Data: 4
Enter Priority: 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter Data: 5
Enter Priority: 1
```

```
3. Display
4. Exit
Enter your choice: 1
Enter Data: 5
Enter Priority: 1

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Priority Queue:
Data: 5, Priority: 1
Data: 4, Priority: 2
Data: 3, Priority: 3
Data: 2, Priority: 4
Data: 1, Priority: 5
```

Dequeue and Display

```
Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Priority Queue:
Data: 4, Priority: 2
Data: 3, Priority: 3
Data: 2, Priority: 4
Data: 1, Priority: 5

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
```

```
Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Priority Queue:
Data: 3, Priority: 3
Data: 2, Priority: 4
Data: 1, Priority: 5

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
```

```
Priority Queue:
Data: 2, Priority: 4
Data: 1, Priority: 5

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Priority Queue:
Data: 1, Priority: 5
```

```
Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is empty

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue is empty
```

Exit

```
Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting
```

Conclusion:-

Learned about implementing priority queue using linked list and applied the same on the code
got to know about various implementation of queues as well

Post lab questions:

1. What are the key considerations in choosing between a circular array and a linked list for the implementation of a queue?

Memory: Circular arrays require a fixed size, while linked lists are dynamic.
Efficiency: Circular arrays offer O(1) access time but may involve resizing; linked lists provide flexible size but have O(n) access time.
Overhead: Linked lists use extra memory for pointers, while circular arrays don't need pointers.

2. Describe how a deque can be implemented using both arrays and linked lists. What are the advantages and disadvantages of each approach?

Array-based deque:

Implemented using a circular array.

Advantages: Fast access (O(1)) to both ends, memory locality.

Disadvantages: Fixed size, resizing overhead.

Linked list-based deque:

Doubly linked list with pointers to both ends.

Advantages: Dynamic size, no need for resizing.

Disadvantages: Extra memory for pointers, slower access (O(n) traversal).

3. Discuss how different types of queues can be used in real-world applications, such as job scheduling, CPU task management, and customer service systems.

Job Scheduling: Uses priority queues to manage tasks, where jobs with higher priority (e.g., urgent tasks) are processed first, ensuring efficiency in systems like OS schedulers.

CPU Task Management: Utilizes circular queues for round-robin scheduling, where tasks are cyclically processed in a fixed time slot, balancing resource allocation.

Customer Service Systems: Employs simple FIFO queues, ensuring customers are served in the order they arrived, maintaining fairness and order in systems like call centers.