Experiment No. : 4

Title: Implementation of Static Priority Queue

Batch: SY-IT(B3) **Roll No.: 16010423076** **Experiment No.: 4**

Aim: Write a menu driven program to implement a static priority queue supporting following operations.

1. Create empty queue,
2. Insert an element on the queue,
3. Delete an element from the queue,
4. Display front, rear element or
5. Display all elements of the queue.

Resources Used: Turbo C/ C++ editor and compiler.

Theory:

Priority Queue -

A Priority Queue is a special type of queue in which each element is associated with a priority, and the elements are dequeued in order of their priority. Elements with higher priority are served before elements with lower priority, regardless of their order in the queue. In a static priority queue, the maximum size of the queue is fixed, and no dynamic resizing occurs.

In this assignment, we implement a static priority queue using an array structure in C. The array stores both the elements (data) and their corresponding priority. Operations like insertion, deletion, and displaying elements are performed based on the priority values.

Algorithm :

1. Create Empty Queue:
   - Initialize the `front` and `rear` of the queue to `-1`, indicating the queue is empty.

2. Insert Element:
   - Check if the queue is full by comparing the rear with `MAX-1`.
   - If the queue is empty, insert the element at the front (0th index).
   - Otherwise, insert the element based on its priority. If an element with a higher priority already exists, shift elements to the right and insert the new element in the correct position.

3. Delete Element:
   - Check if the queue is empty.
   - Remove the element at the front of the queue and adjust the `front` pointer.
   - If after deletion the queue becomes empty, reset `front` and `rear` to `-1`.

4. Display Queue:

- If the queue is empty, print "Queue is empty".
- Otherwise, display the elements along with their priorities from `front` to `rear`.

Program:

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

#define MAX 4

struct PriorityQueue {
   int data[MAX];
   int front, rear;
};

void createQueue(struct PriorityQueue* pq)
{
pq->front = pq->rear = -1;
}

int isEmpty(struct PriorityQueue* pq)
{
return pq->front == -1;
}

int isFull(struct PriorityQueue* pq)
{
return pq->rear == MAX - 1;
}

void insert(struct PriorityQueue* pq, int value)
{
   if (isFull(pq))
   {
      printf("Queue is full. Cannot insert %d.\n", value);
      return;
   }

   if (isEmpty(pq))
   {
      pq->front = pq->rear = 0;
      pq->data[pq->rear] = value;
   }
   else
   {
      int i = pq->rear;
      while (i >= pq->front && pq->data[i] < value) {
```

```
            pq->data[i + 1] = pq->data[i];
            //Shift the element to make space
            i--;
        }
        pq->data[i + 1] = value;
        pq->rear++;
    }
    printf("Inserted %d\n", value);
}

void delete(struct PriorityQueue* pq) {
    if (isEmpty(pq))
    {
        printf("Queue is empty. Cannot delete.\n");
        return;
    }
    printf("Deleted %d\n", pq->data[pq->front]);

    if (pq->front == pq->rear) {
        pq->front = pq->rear = -1;
    } else {
        pq->front++;
    }
}

void display(struct PriorityQueue* pq)
{
    if (isEmpty(pq)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Priority Queue elements:\n");
    for (int i = pq->front; i <= pq->rear; i++) {
        printf("%d ", pq->data[i]);
    }
    printf("\n");
}

int main() {
    struct PriorityQueue pq;
    createQueue(&pq);

    int choice, value;
    while (1) {
        printf("\nPriority Queue Operations:\n");
        printf("1. Create queue\n");
        printf("2. Insert (enqueue)\n");
        printf("3. Delete (dequeue)\n");
        printf("4. Display all elements\n");
        printf("5. Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createQueue(&pq);
                printf("Queue created.\n");
                break;
            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(&pq, value);
                break;
            case 3:
                delete(&pq);
                break;
            case 4:
                display(&pq);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Results:

Output Clear

```
/tmp/8AeXeW8EBD.o

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 1
Queue created.

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 2
Enter value to insert: 8
Inserted 8

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 2
Enter value to insert: 5
```

```
Inserted 5

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 2
Enter value to insert: 2
Inserted 2

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 2
Enter value to insert: 2
Inserted 2

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
```

```
Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 3
Deleted 8

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 4
Priority Queue elements:
5 2 2

Priority Queue Operations:
1. Create queue
2. Insert (enqueue)
3. Delete (dequeue)
4. Display all elements
5. Exit
Enter your choice: 5
```

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

Conclusion:

From this experiment, I learned how to implement and manage a static priority queue using arrays in C. I gained practical experience with priority-based insertion, deletion, and handling queue operations efficiently.

.

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

Signature of faculty in-charge with date :

References:

Books/ Journals/ Websites:

- Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.