Computer Engineering Department, S.V.N.I.T. Surat.
B Tech (CO) –II[nd] Year semester-III
Course: *Data Structures CO203*

Assignment-VI

Q1.) Implement double ended priority queue [DeQueue]

The library is managing request for Book in first come first serve manner. Teacher's requests are given higher priority than that of students. Design the task of library using the most suitable Data Structure.

Code:

```c
// Implementation of "Output Restricted Deque"
// Insertion from Both Ends, But Deletion from Only One End Based on Priority

#include <stdio.h>
#include <conio.h>
#define MAX 100

int deque[MAX];
int left = -1, right = -1;

void insert_right(int);
void insert_left(int);
void delete_right(void);
void delete_left(void);
void display(void);

// Teacher is Inserted from Left & Student is Inserted from Right

// Student Request Time Array
int stime[MAX];

// Teachers Request Time Array
int ttime[MAX];

int main()
{
    int i;

    int stud_req = 0;
    int mxn = -1;
    printf("\nEnter the Number of Student Request : ");
    scanf("%d", &stud_req);

    for (i = 0; i < stud_req; i++)
    {
```

```
        printf("Enter Student %d Request Time : ", i + 1);
        scanf("%d", &stime[i]);
        if (stime[i] > mxn)
            mxn = stime[i];
    }

    int teah_req = 0;
    printf("\nEnter the Number of Teacher Request : ");
    scanf("%d", &teah_req);
    for (i = 0; i < teah_req; i++)
    {
        printf("Enter Teacher %d Request Time : ", i + 1);
        scanf("%d", &ttime[i]);
        if (ttime[i] > mxn)
            mxn = ttime[i];
    }

    //Considering Both Student and Teacher Request Time to be Sorted

    //Student is Inserted from Right
    for (i = stud_req - 1; i >= 0; i--)
    {
        insert_right(stime[i]);
    }

    // Teacher is Inserted from Left
    for (i = teah_req - 1; i >= 0; i--)
    {
        insert_left(ttime[i]);
    }

    // display();

    int cur_time;
    // Total Elements in Queue
    int popcnt = stud_req + teah_req;

    int tcnt = 1;
    int scnt = 1;

    printf("Order of Execution of Request : \n\n");

    if (stud_req <= 0 && teah_req <= 0)
    {
        if (stud_req == 0 && teah_req == 0)
        {
            printf("No Request to be Processed!\n");
        }
        else
        {
```

```
            printf("Invalid Input!\n");
        }
        return 0;
    }

    while (popcnt != 0)
    {
        // Teacher Request Time
        int t = deque[left];
        // Student Request Time
        int s = deque[right];

        if (t == s)
        {
            // POP Teacher
            if (tcnt <= teah_req)
            {
                printf("Teacher %d\n", tcnt);
                tcnt++;
                delete_left();
            }
            else
            {
                printf("Student %d\n", scnt);
                scnt++;
                delete_right();
            }
        }
        else
        {
            if (t < s && tcnt <= teah_req)
            {
                // POP Teacher
                printf("Teacher %d\n", tcnt);
                tcnt++;
                delete_left();
            }
            else
            {
                // s < t && scnt<=stud_req
                // POP Student
                printf("Student %d\n", scnt);
                scnt++;
                delete_right();
            }
        }
        popcnt--;
    }

    return 0;
```

```
}

//-------INSERT AT RIGHT-------
void insert_right(int val)
{
    if ((left == 0 && right == MAX - 1) || (left == right + 1))
    {
        printf("\nOVERFLOW");
        return;
    }
    if (left == -1) //if queue is initially empty
    {
        left = 0;
        right = 0;
    }
    else
    {
        if (right == MAX - 1)
            right = 0;
        else
            right = right + 1;
    }
    deque[right] = val;
}

//-------INSERT AT LEFT-------
void insert_left(int val)
{
    if ((left == 0 && right == MAX - 1) || (left == right + 1))
    {
        printf("\nOVERFLOW");
        return;
    }
    if (left == -1) //if queue is initially empty
    {
        left = 0;
        right = 0;
    }
    else
    {
        // To Avoid -ve Index in Array
        // We Insert from Back of deque
        if (left == 0)
            left = MAX - 1;
        else
            left = left - 1;
    }
    deque[left] = val;
}
```

```c
//-------DELETE FROM RIGHT-------
void delete_right()
{
    if (left == -1)
    {
        printf("\nUNDERFLOW");
        return;
    }
    // printf("\nThe deleted element is %d\n", deque[right]);
    // int d = deque[right];
    //Queue has only one element
    if (left == right)
    {
        left = -1;
        right = -1;
    }
    else
    {
        if (right == 0)
            right = MAX - 1;
        else
            right = right - 1;
    }
    return;
}

//-------DELETE FROM LEFT-------
void delete_left()
{
    if (left == -1)
    {
        printf("\nUNDERFLOW");
        return;
    }
    // printf("\nThe deleted element is %d\n", deque[left]);
    // int d = deque[left];
    if (left == right) //Queue has only one element
    {
        left = -1;
        right = -1;
    }
    else
    {
        if (left == MAX - 1)
            left = 0;
        else
            left = left + 1;
    }
    return;
}
```

```c
//-------DISPLAY-------
void display()
{
    int front = left, rear = right;

    if (front == -1)
    {
        printf("\nQueue is Empty\n");
        return;
    }

    printf("\nDeQueue Elements : ");

    if (front <= rear)
    {
        while (front <= rear)
        {
            printf("%d ", deque[front]);
            front++;
        }
    }
    else
    {
        while (front <= MAX - 1)
        {
            printf("%d ", deque[front]);
            front++;
        }
        front = 0;
        while (front <= rear)
        {
            printf("%d ", deque[front]);
            front++;
        }
    }
    printf("\n");
}
```

Sample Input:

(Assuming execution of every request is 1 sec)

Student request at time in seconds: 1,4,6,7,8

Teacher's request at time in seconds: 1,3,5,7,10

Sample Output : [T1->S1-> T2->S2-> T3->S3-> T4->S4-> S5->T5]

```
Enter the Number of Student Request : 5
Enter Student 1 Request Time : 1
Enter Student 2 Request Time : 4
Enter Student 3 Request Time : 6
Enter Student 4 Request Time : 7
Enter Student 5 Request Time : 8

Enter the Number of Teacher Request : 5
Enter Teacher 1 Request Time : 1
Enter Teacher 2 Request Time : 3
Enter Teacher 3 Request Time : 5
Enter Teacher 4 Request Time : 7
Enter Teacher 5 Request Time : 10
Order of Execution of Request :

Teacher 1
Student 1
Teacher 2
Student 2
Teacher 3
Student 3
Teacher 4
Student 4
Student 5
Teacher 5
```

Sample Input:

(Assuming execution of every request is 1 sec)

Student request at time in seconds: 1

Teacher's request at time in seconds: 1

Sample Output : [T1->S1]

```
Enter the Number of Student Request : 1
Enter Student 1 Request Time : 1

Enter the Number of Teacher Request : 1
Enter Teacher 1 Request Time : 1
Order of Execution of Request :

Teacher 1
Student 1
```

Sample Input:

(Assuming execution of every request is 1 sec)

Student request at time in seconds:

Teacher's request at time in seconds:

Sample Output : [No Request]

```
Enter the Number of Student Request : 0

Enter the Number of Teacher Request : 0
Order of Execution of Request :

No Request to be Processed!
```

Sample Input:

(Assuming execution of every request is 1 sec)

Student request at time in seconds: 2,3

Teacher's request at time in seconds: 1,3,5

Sample Output : [T1->S1-> T2->S2-> T3]

```
Enter the Number of Student Request : 2
Enter Student 1 Request Time : 2
Enter Student 2 Request Time : 3

Enter the Number of Teacher Request : 3
Enter Teacher 1 Request Time : 1
Enter Teacher 2 Request Time : 3
Enter Teacher 3 Request Time : 5
Order of Execution of Request :

Teacher 1
Student 1
Teacher 2
Student 2
Teacher 3
```