**DSA Assignment 5**

[**https://github.com/PrathamAsrani/DSA\_C/blob/master/assignment\_stack.c**](file:///D:\DSA%20Practicals\DSA%20Assignment%201.docx#_top)

#include <stdio.h>

#include <stdlib.h>

struct Queue

{

    int \*data;

    int front, rear, size;

};

/\* Queue Basic Function : \*/

struct Queue \*create(int size);

int isFull(struct Queue q);

int isEmpty(struct Queue q);

void enqueue(struct Queue \*q, int value);

void dequeue(struct Queue \*q);

void queueTraversal(struct Queue q);

int swap(int \*a, int \*b);

int reverse(struct Queue \*q);

/\* Queue Basic Function : \*/

struct Array

{

    int \*arr;

    int index, size;

};

/\*Array Implimentation Function\*/

void intialize(struct Array \*a, int size);

int createArray(struct Array \*a, int value);

void display(struct Array a);

/\*Array Implimentation Function\*/

/\*Combinational function : \*/

void combine(struct Queue q1, struct Queue q2, struct Array \*a)

{

    int s = (q1.size + q2.size);

    intialize(a, s);

    int j = 0;

    for (int i = 0; i < (q1.size + q2.size); i++)

    {

        if (i < q1.size)

        {

            createArray(a, q1.data[i]);

        }

        else

        {

            createArray(a, q2.data[j]);

            j++;

        }

    }

}

int partition(struct Array a, int low, int high)

{

    int pivot = a.arr[high];

    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)

    {

        if (a.arr[j] < pivot)

        {

            i++;

            swap(&a.arr[i], &a.arr[j]);

        }

    }

    swap(&a.arr[i + 1], &a.arr[high]);

    return (i + 1);

}

void quickSort(struct Array a, int low, int high)

{

    if (low < high)

    {

        int pi = partition(a, low, high);

        quickSort(a, low, pi - 1);

        quickSort(a, pi + 1, high);

    }

}

/\*Combinational function : \*/

int main()

{

    // Creating queue

    int m, n;

    printf("Enter size of Queue 1: ");

    scanf("%d", &n);

    printf("Enter size of Queue 2: ");

    scanf("%d", &m);

    struct Queue \*q1 = create(m);

    struct Queue \*q2 = create(n);

    // Operations on queue

    printf("Enter elements for Queue 1:\n");

    for (int i = 0; i < m; i++)

    {

        int data1;

        printf("%d : ", i + 1);

        scanf("%d", &data1);

        enqueue(q1, data1);

    }

    printf("Enter elements for Queue 2:\n");

    for (int i = 0; i < n; i++)

    {

        int data1;

        printf("%d : ", i + 1);

        scanf("%d", &data1);

        enqueue(q2, data1);

    }

    printf("Queue 1: ");

    queueTraversal(\*q1);

    reverse(q1);

    printf("After reversing Queue 1: ");

    queueTraversal(\*q1);

    printf("Queue 2: ");

    queueTraversal(\*q2);

    reverse(q2);

    printf("After reversing Queue 2: ");

    queueTraversal(\*q2);

    // inserting queues in arrat

    struct Array array;

    combine(\*q1, \*q2, &array);

    display(array);

    printf("The array after sorting\n");

    quickSort(array, 0, array.size);

    display(array);

    return 0;

}

// the return type of createQueue function is the address of object of the same template. Therefore the initializer used as struct Queue

struct Queue \*create(int size)

{

    struct Queue \*ptr = (struct Queue \*)malloc(sizeof(struct Queue));

    ptr->front = -1;

    ptr->rear = -1;

    ptr->size = size;

    ptr->data = (int \*)malloc(ptr->size \* sizeof(int));

    return ptr;

}

int isFull(struct Queue q)

{

    if (q.front == q.rear)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

int isEmpty(struct Queue q)

{

    if (q.rear == -1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

void enqueue(struct Queue \*q, int value)

{

    q->rear += 1;

    if (isFull(\*q))

    {

        printf("Queue is full\n");

    }

    else

    {

        \*(q->data + q->rear) = value;

    }

}

void dequeue(struct Queue \*q)

{

    if (isEmpty(\*q))

    {

        printf("Queue is empty\n");

    }

    else

    {

        q->front += 1;

    }

}

void queueTraversal(struct Queue q)

{

    for (int i = 0; i < q.rear + 1; i++)

    {

        printf("%d ", q.data[i]);

    }

    printf("\n");

}

int swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int reverse(struct Queue \*q)

{

    printf("The queue reversing started\n");

    for (int i = 0; i < q->size / 2; i++)

    {

        swap(&q->data[i], &q->data[q->size - i - 1]);

    }

}

// Array :

void intialize(struct Array \*a, int size)

{

    a->index = -1;

    a->size = size;

    a->arr = (int \*)malloc(a->size \* sizeof(int));

}

int createArray(struct Array \*a, int value)

{

    a->index += 1;

    a->arr[a->index] = value;

    // printf("%d\n", a->arr[a->index]);

}

void display(struct Array a)

{

    printf("The array : ");

    for (int i = 0; i < a.size; i++)

    {

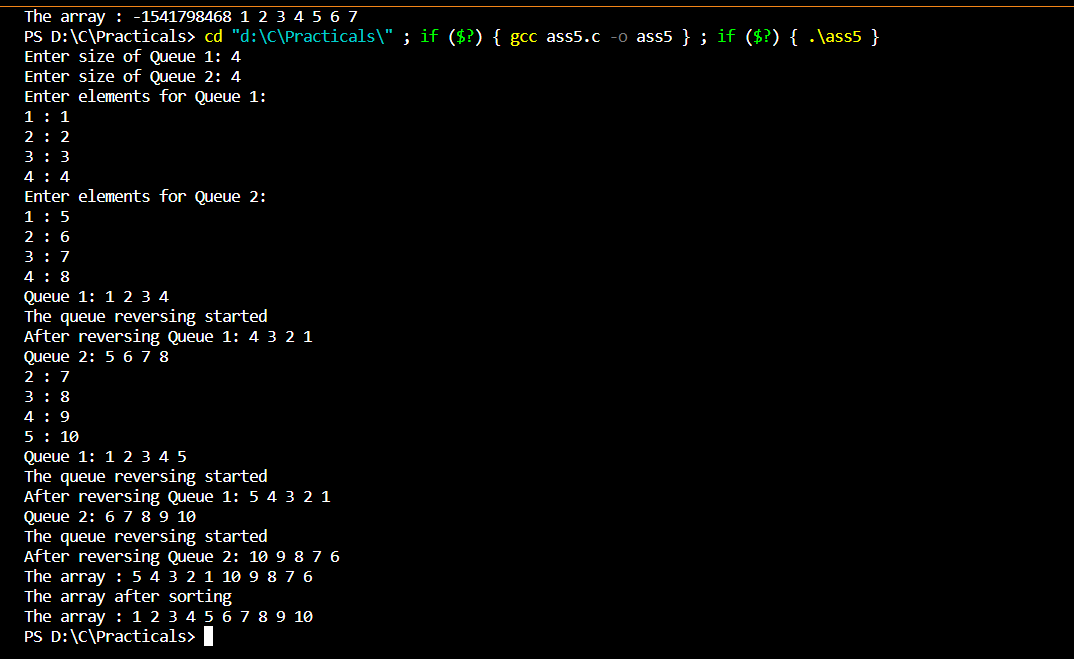
        printf("%d ", (a.arr[i]));

    }

    printf("\n");

}

**Output:**



**Result : Hence we successfully combined two queue in one array and implemented quick sort.**