Q1. Create a fixed memory implementation of queue.

<https://ideone.com/buNgO8>

Input file: rb.gy/shv4gu

Ans.

// C program for array implementation of queue

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

// A structure to represent a queue

struct Queue {

    int front, rear, size;

    unsigned capacity;

    int\* array;

};

// function to create a queue

// of given capacity.

// It initializes size of queue as 0

struct Queue\* createQueue(unsigned capacity)

{

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

    queue->capacity = capacity;

    queue->front = queue->size = 0;

    // This is important, see the enqueue

    queue->rear = -1;

    queue->array = (int\*)malloc(queue->capacity \* sizeof(int));

    return queue;

}

// Queue is full when size becomes

// equal to the capacity

int isFull(struct Queue\* queue)

{

    if(queue->size == queue->capacity){

        return 1;

    }

    return 0;

}

// Queue is empty when size is 0

int isEmpty(struct Queue\* queue)

{

    if(queue->size == 0){

        return 1;

    }

    return 0;

}

// Function to add an item to the queue.

// It changes rear and size

void enqueue(struct Queue\* queue, int item)

{

    if(isFull(queue)){

        printf("\nQueue is full!");

    }

    else{

        printf("\n%d enqueued to queue", item);

        queue->size++;

        queue->array[++queue->rear] = item;

    }

}

// Function to remove an item from queue.

// It changes front and size

int dequeue(struct Queue\* queue)

{

    int item = 0;

    if (isEmpty(queue))

        return INT\_MIN;

    else

    {

        queue->size--;

        item = queue->array[queue->front++];

    }

    return item;

}

// Function to display queue contents

void display(struct Queue\* queue)

{

    if (isEmpty(queue))

        printf("\nQueue is Empty");

    else

    {

        printf("\nQueue is: ");

        for(int i=queue->front; i<=queue->rear; i++){

            printf("%d ", queue->array[i]);

        }

        printf("\n");

    }

}

// Driver program to test enqueue, dequeue, display functions

int main()

{

    FILE \*fptr;

    if ((fptr = fopen("queue\_fixed\_input.txt","r")) == NULL)

    {

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

    }

    int cap, choice, data;

    printf("Enter Queue capacity: ");

    fscanf(fptr, "%d", &cap);

    printf("%d", cap);

    if(cap<1)

    {

        printf("\nQueue capacity cannot be less than 1");

        return 0;

    }

    // Create a queue

    // Below Line may not be the same as your code, but the variable name "queue" must be same

    struct Queue\* queue = createQueue(cap);

    // Start of Menu driven code

    do

    {

        printf("\nPress 1 to enqueue\nPress 2 to dequeue\nPress 3 to display\nPress 0 to quit");

        fscanf(fptr, "%d", &choice);

        printf("\n%d", choice);

        switch(choice)

        {

            case 1:

                printf("\nEnter value to insert: ");

                fscanf(fptr, "%d",&data);

                printf("%d", data);

                // Your function name and signature must be same as below

                enqueue(queue, data);

                break;

            case 2:

                // Your function name and signature must be same as below

                data = dequeue(queue);

                if(data!=INT\_MIN)

                    printf("\n%d is dequeued", data);

                else

                    printf("\nQueue is empty");

                break;

            case 3:

            // Your function name and signature must be same as below

                display(queue);

                break;

            case 0:

                printf("\nQuitting program");

                break;

            default:

                printf("\nInvalid choice");

                break;

        }

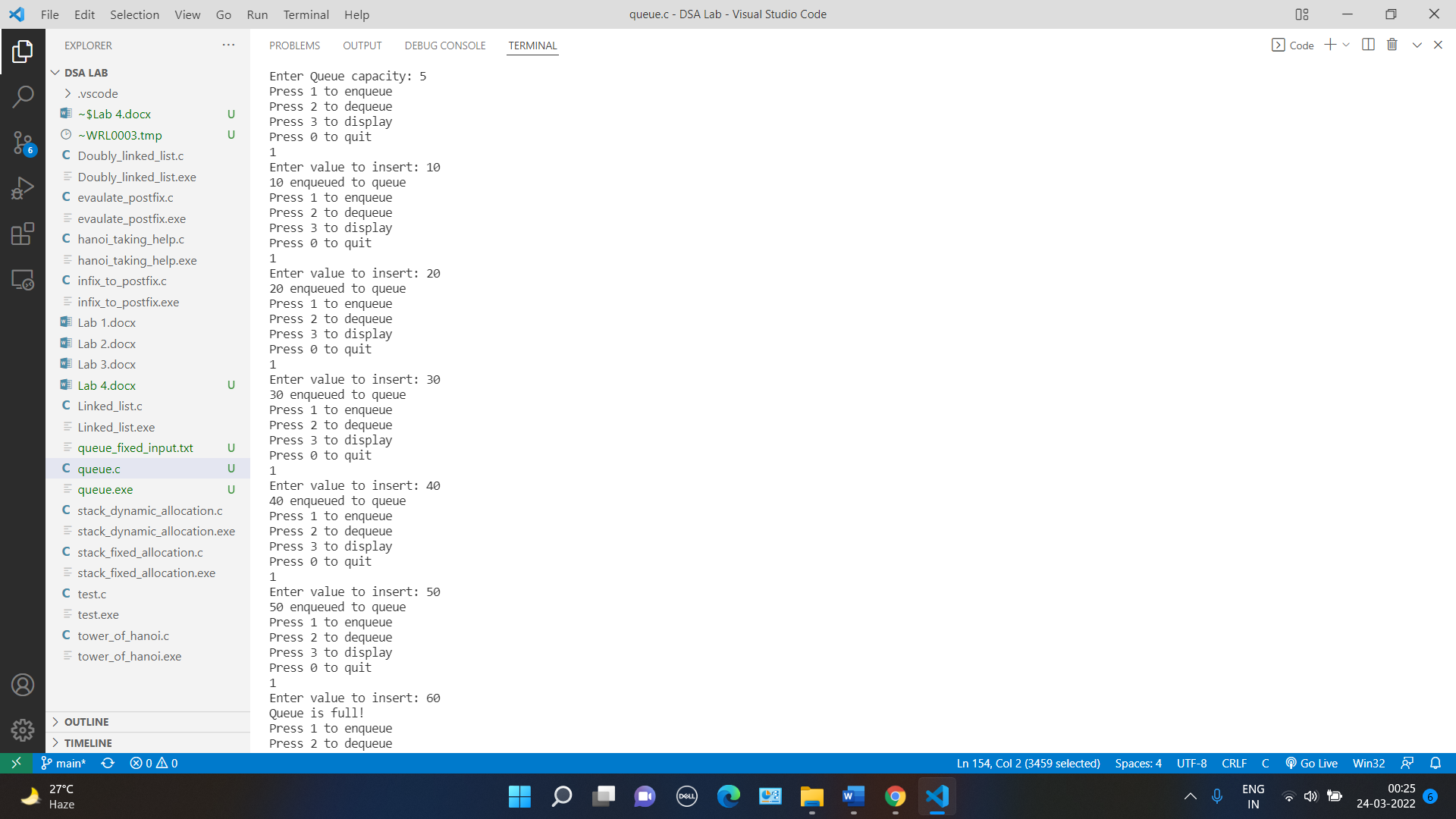
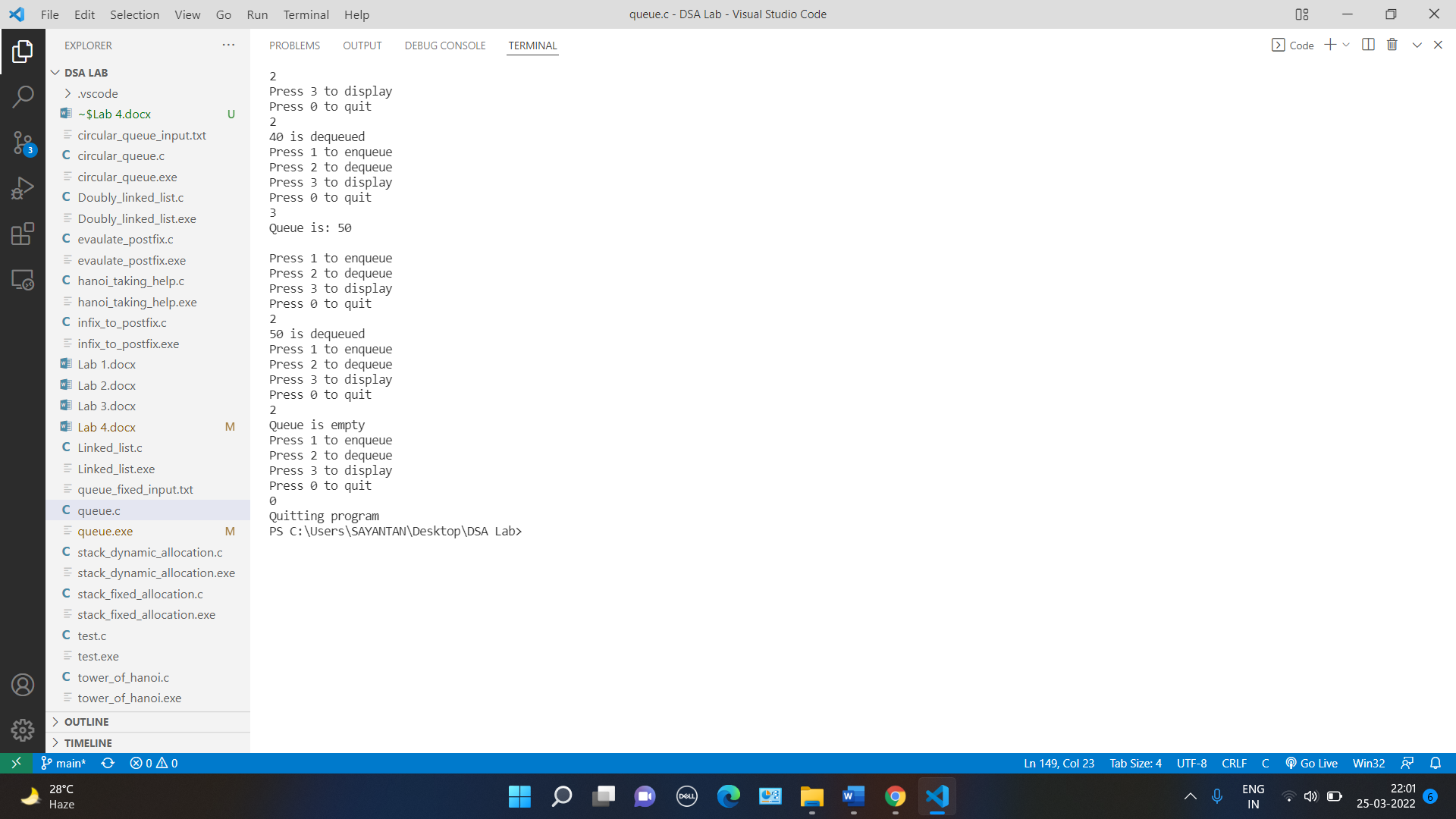
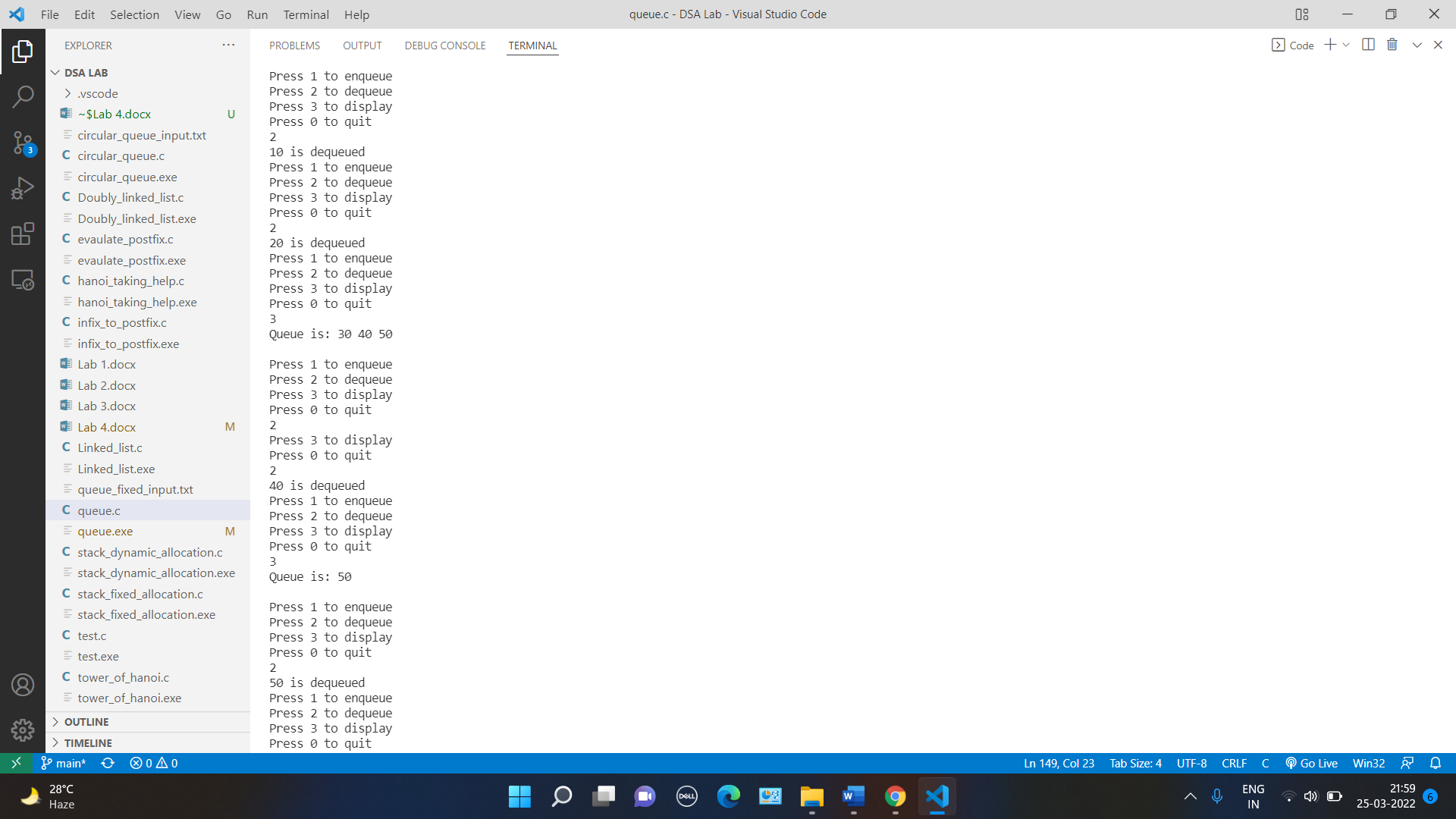
    }while(choice!=0);

    fclose(fptr);

    return 0;

}

Output

Q2. Create a fixed memory implementation of circular queue.

<https://ideone.com/buNgO8>

Input file: rb.gy/shv4gu

Ans.

// C program for array implementation of queue

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

// A structure to represent a queue

struct Queue {

    int front, rear, size;

    unsigned capacity;

    int\* array;

};

// function to create a queue

// of given capacity.

// It initializes size of queue as 0

struct Queue\* createQueue(unsigned capacity)

{

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

    queue->capacity = capacity;

    queue->front = queue->size = 0;

    // This is important, see the enqueue

    queue->rear = -1;

    queue->array = (int\*)malloc(queue->capacity \* sizeof(int));

    return queue;

}

// Queue is full when size becomes

// equal to the capacity

int isFull(struct Queue\* queue)

{

    if(queue->size == queue->capacity){

        return 1;

    }

    return 0;

}

// Queue is empty when size is 0

int isEmpty(struct Queue\* queue)

{

    if(queue->size == 0){

        return 1;

    }

    return 0;

}

// Function to add an item to the queue.

// It changes rear and size

void enqueue(struct Queue\* queue, int item)

{

    if(isFull(queue)){

        printf("\nQueue is full!");

    }

    else{

        printf("\n%d enqueued to queue", item);

        if(queue->rear == queue->capacity-1){

            queue->rear = 0;

            queue->array[queue->rear] = item;

        }

        else{

            queue->array[++queue->rear] = item;

        }

        queue->size++;

    }

}

// Function to remove an item from queue.

// It changes front and size

int dequeue(struct Queue\* queue)

{

    int item = 0;

    if (isEmpty(queue))

        return INT\_MIN;

    else

    {

        if(queue->front == queue->capacity-1){

            item = queue->array[queue->front];

            queue->front = 0;

        }

        else{

            item = queue->array[queue->front++];

        }

        queue->size--;

    }

    return item;

}

// Function to display queue contents

void display(struct Queue\* queue)

{

    if (isEmpty(queue))

        printf("\nQueue is Empty");

    else

    {

        printf("\nQueue is: ");

        if(queue->front<=queue->rear){

            for(int i=queue->front; i<=queue->rear; i++){

                printf("%d ", queue->array[i]);

            }

        }

        else{

            for(int i=queue->front; i<queue->capacity; i++){

                printf("%d ", queue->array[i]);

            }

            for(int i=0; i<=queue->rear; i++){

                printf("%d ", queue->array[i]);

            }

        }

        printf("\n");

    }

}

// Driver program to test enqueue, dequeue, display functions

int main()

{

    FILE \*fptr;

    if ((fptr = fopen("queue\_fixed\_input.txt","r")) == NULL)

    {

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

    }

    int cap, choice, data;

    printf("Enter Queue capacity: ");

    fscanf(fptr, "%d", &cap);

    printf("%d", cap);

    if(cap<1)

    {

        printf("\nQueue capacity cannot be less than 1");

        return 0;

    }

    // Create a queue

    // Below Line may not be the same as your code, but the variable name "queue" must be same

    struct Queue\* queue = createQueue(cap);

    // Start of Menu driven code

    do

    {

        printf("\nPress 1 to enqueue\nPress 2 to dequeue\nPress 3 to display\nPress 0 to quit");

        fscanf(fptr, "%d", &choice);

        printf("\n%d", choice);

        switch(choice)

        {

            case 1:

                printf("\nEnter value to insert: ");

                fscanf(fptr, "%d",&data);

                printf("%d", data);

                // Your function name and signature must be same as below

                enqueue(queue, data);

                break;

            case 2:

                // Your function name and signature must be same as below

                data = dequeue(queue);

                if(data!=INT\_MIN)

                    printf("\n%d is dequeued", data);

                else

                    printf("\nQueue is empty");

                break;

            case 3:

            // Your function name and signature must be same as below

                display(queue);

                break;

            case 0:

                printf("\nQuitting program");

                break;

            default:

                printf("\nInvalid choice");

                break;

        }

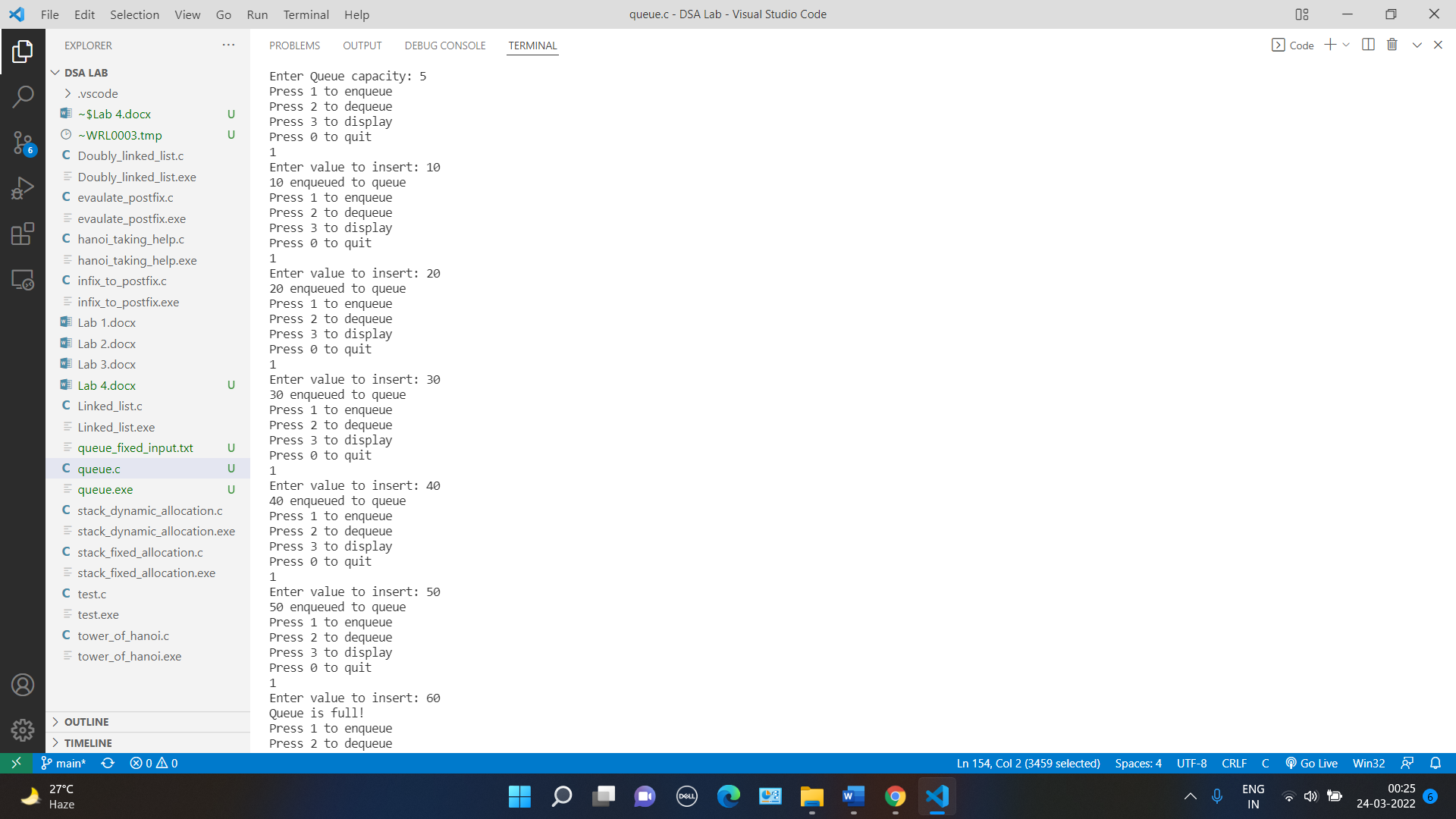
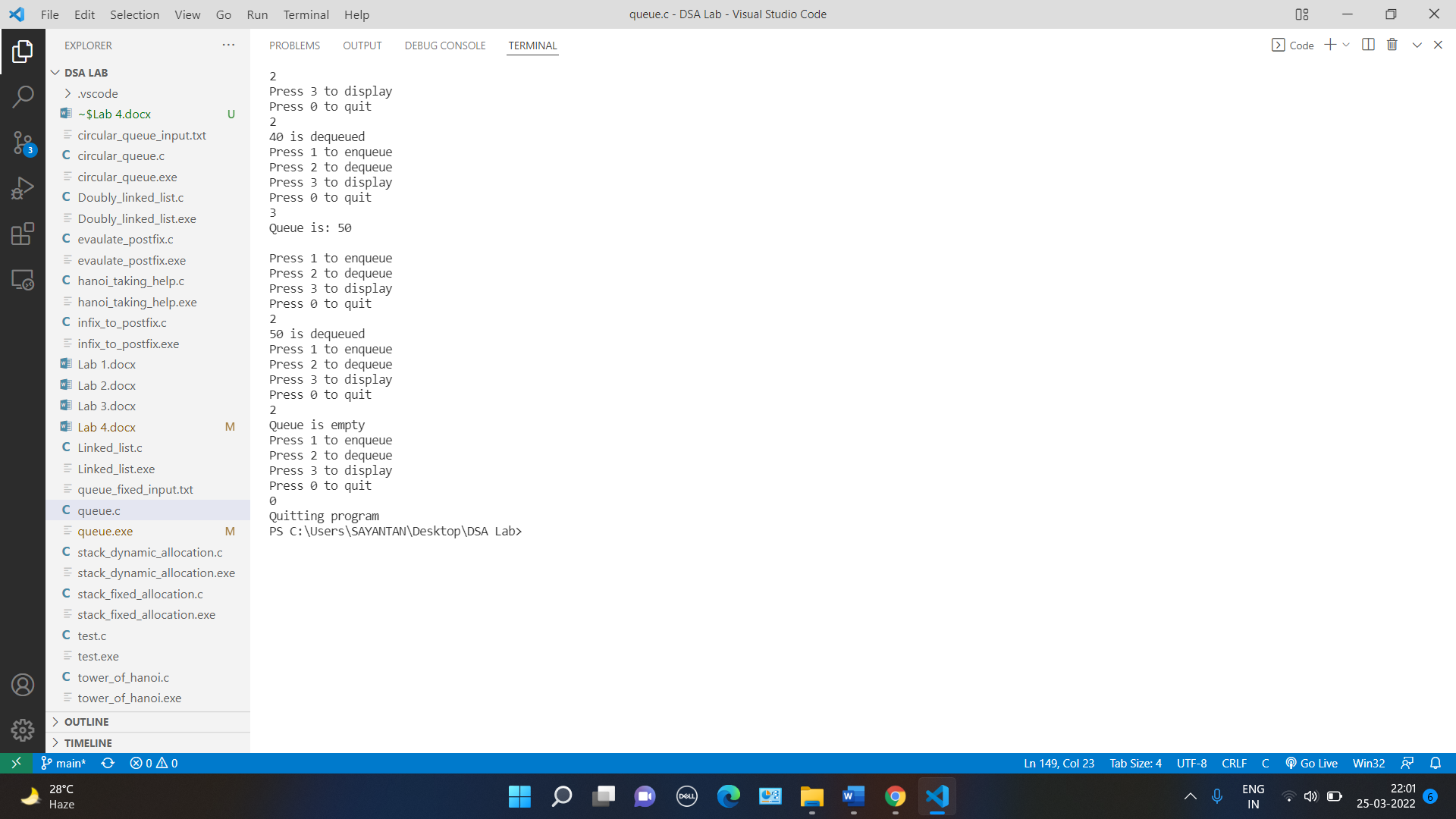
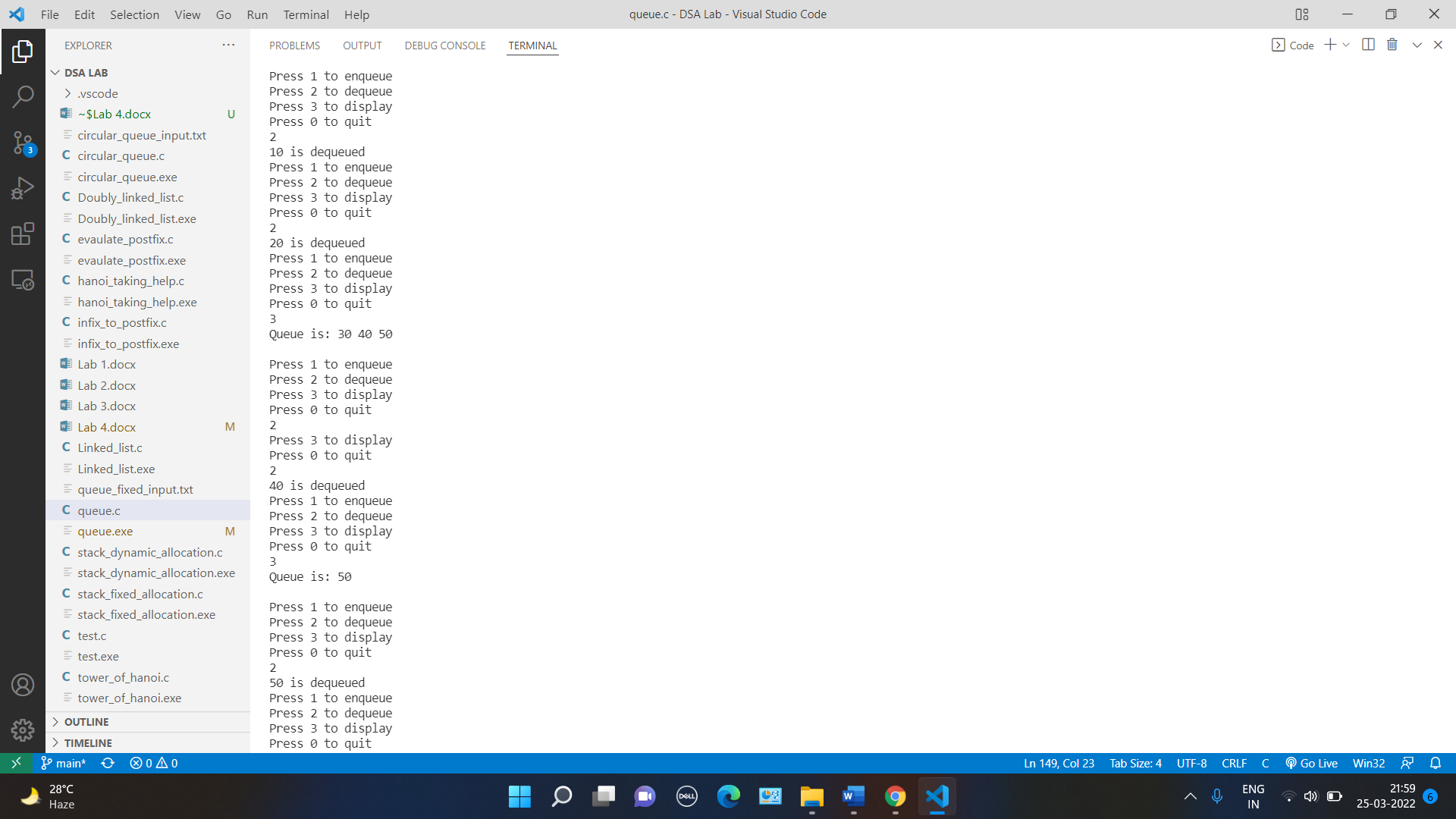
    }while(choice!=0);

    fclose(fptr);

    return 0;

}

Output

Q3. Create a dynamic memory implementation of the circular queue.

Code: ideone.com/QD7VE7

i/p file: rb.gy/gdy4ho

Ans.

// A C program to demonstrate linked list based implementation of circular queue

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

// A linked list (LL) node to store a queue entry

struct QNode {

    int key;

    struct QNode\* next;

};

// The queue, front stores the front node of LL and rear stores the

// last node of LL

struct Queue {

    struct QNode \*front, \*rear;

};

// A utility function to create a new linked list node.

struct QNode\* newNode(int k)

{

    struct QNode\* temp = (struct QNode\*)malloc(sizeof(struct QNode));

    temp->key = k;

    temp->next = NULL;

    return temp;

}

// A utility function to create an empty queue

struct Queue\* createQueue()

{

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

    q->front = q->rear = NULL;

    return q;

}

// The function to add a key k to q

void enQueue(struct Queue\* q, int k)

{

    // Create a new LL node

    // If queue is empty, then new node is front and rear both

    // Else Add the new node at the end of queue and change rear

     struct QNode \*node = newNode(k);

     if(q->front == NULL && q->rear == NULL){

         node->next = node;

         q->front = q->rear = node;

     }

     else{

         q->rear->next = node;

         node->next = q->front;

         q->rear = node;

     }

}

// Function to remove a key from given queue q

int deQueue(struct Queue\* q)

{

    // If queue is empty, return INT\_MIN.

    int item = 0;

    if(q->front == NULL && q->rear == NULL){

        return INT\_MIN;

    }

    else if(q->front == q->rear){

        item = q->front->key;

        q->front->next = q->rear->next = NULL;

        free(q->front);

        free(q->rear);

        q->front = q->rear = NULL;

    }

    else{

        item = q->front->key;

        struct QNode \*node = q->front;

        q->front = q->front->next;

        q->rear->next = q->front;

        node->next = NULL;

        free(node);

    }

    return item;

}

void display(struct Queue\* q)

{

    // If queue is empty, printf Queue is Empty in this function.

    if(q->front == NULL && q->rear == NULL){

        printf("\nQueue is Empty!\n");

    }

    else{

        struct QNode \*node = q->front;

        printf("\nQueue is: ");

        do{

            printf("%d ", node->key);

            node = node->next;

        }while (node!=q->front);

        printf("\n");

    }

}

// Driver program to test above functions

int main()

{

    FILE \*fptr;

    if ((fptr = fopen("circular\_queue\_LL\_input.txt","r")) == NULL)

    {

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

    }

    int choice, data;

    // Below line may be different but variable name "queue" must be same

    struct Queue\* queue = createQueue();

    do

    {

        printf("\nPress 1 to enqueue\nPress 2 to dequeue\nPress 3 to display\nPress 0 to quit");

        fscanf(fptr, "%d", &choice);

        printf("\n%d", choice);

        switch(choice)

        {

            case 1:

                printf("\nEnter value to insert: ");

                fscanf(fptr, "%d",&data);

                printf("%d", data);

                enQueue(queue, data);

                break;

            case 2:

                data = deQueue(queue);

                if(data!=INT\_MIN)

                    printf("\n%d is dequeued", data);

                else

                    printf("\nQueue is empty");

                break;

            case 3:

                display(queue);

                break;

            case 0:

                printf("\nQuitting program");

                break;

            default:

                printf("\nInvalid choice");

                break;

        }

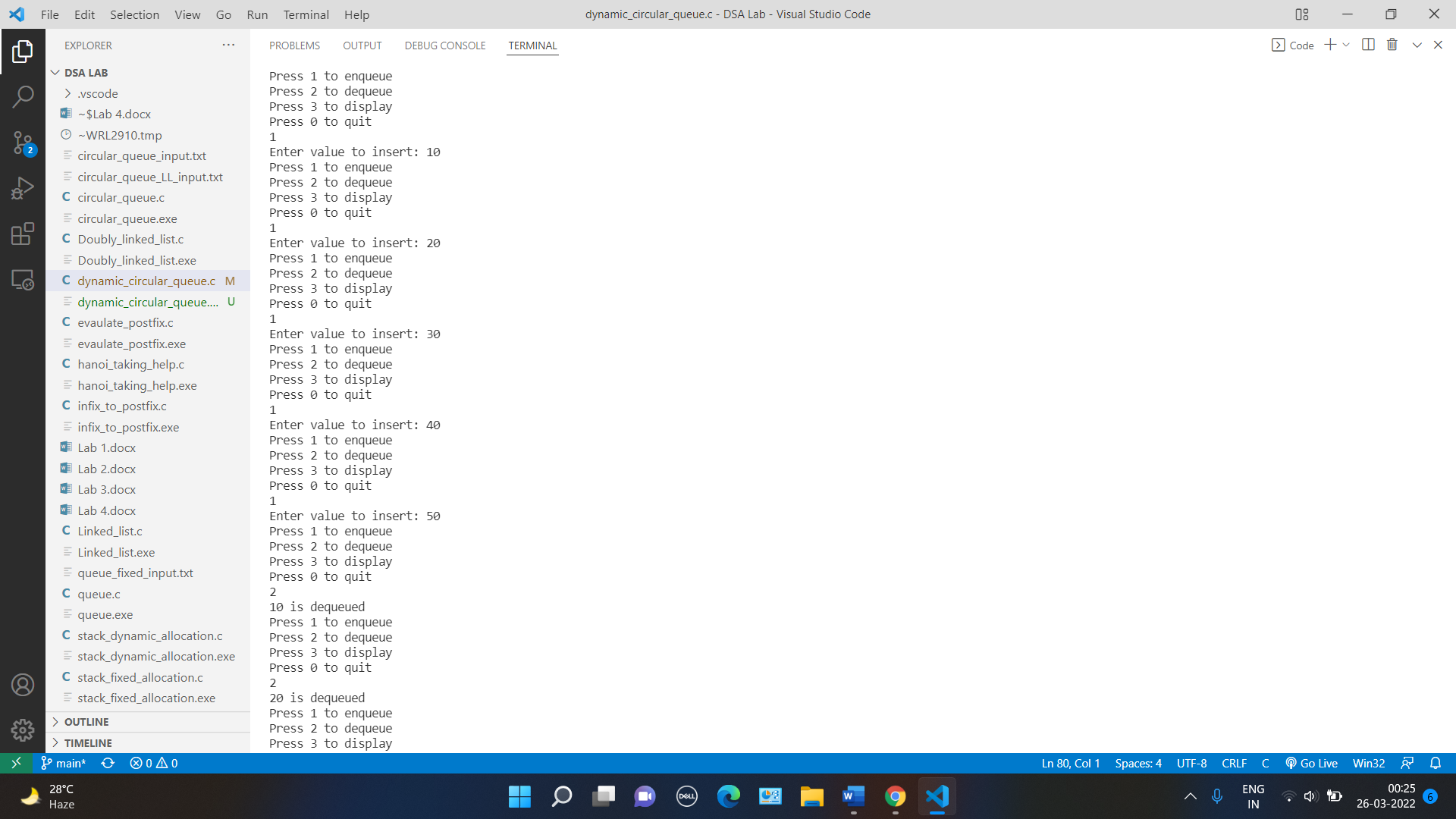
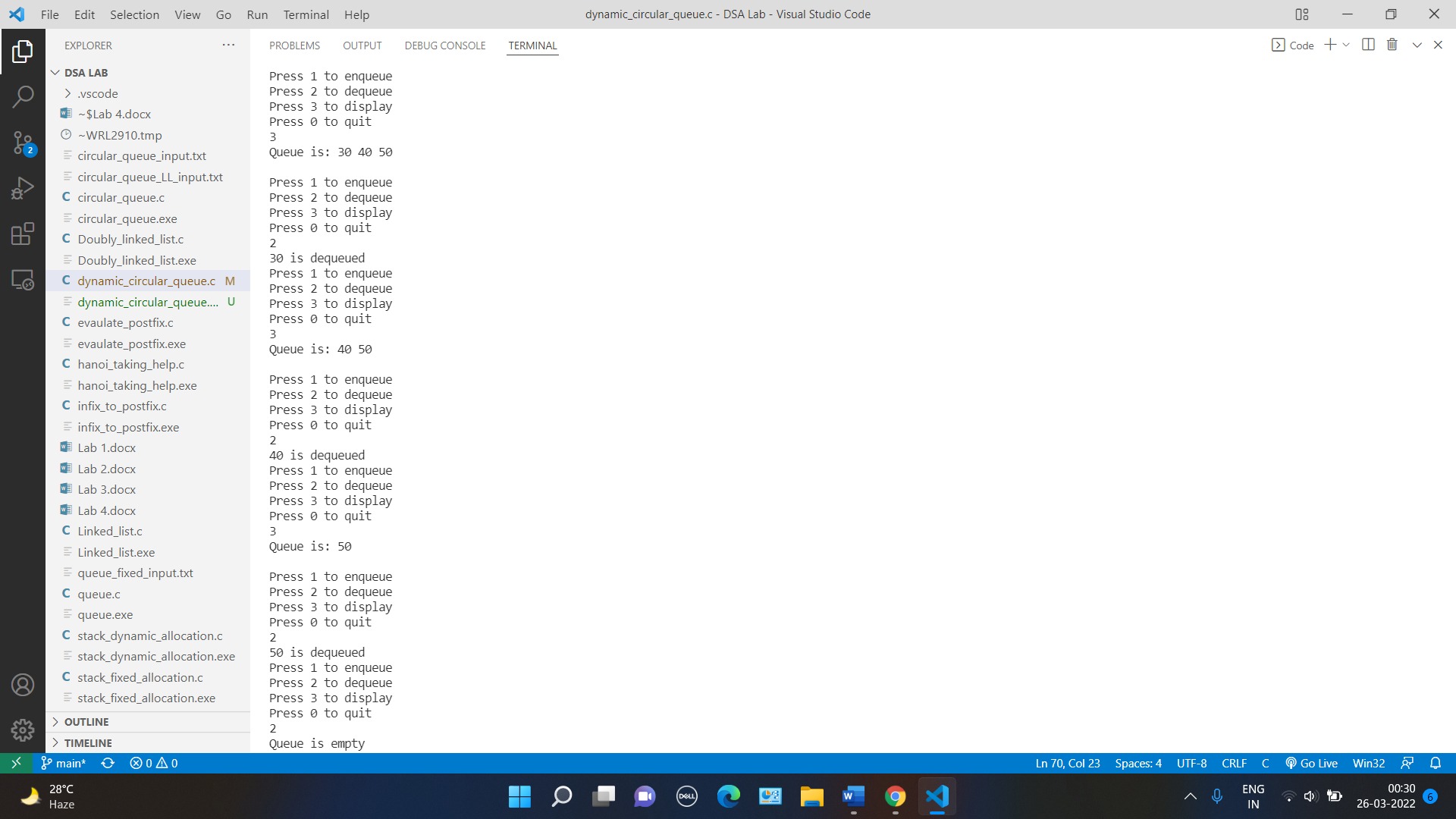
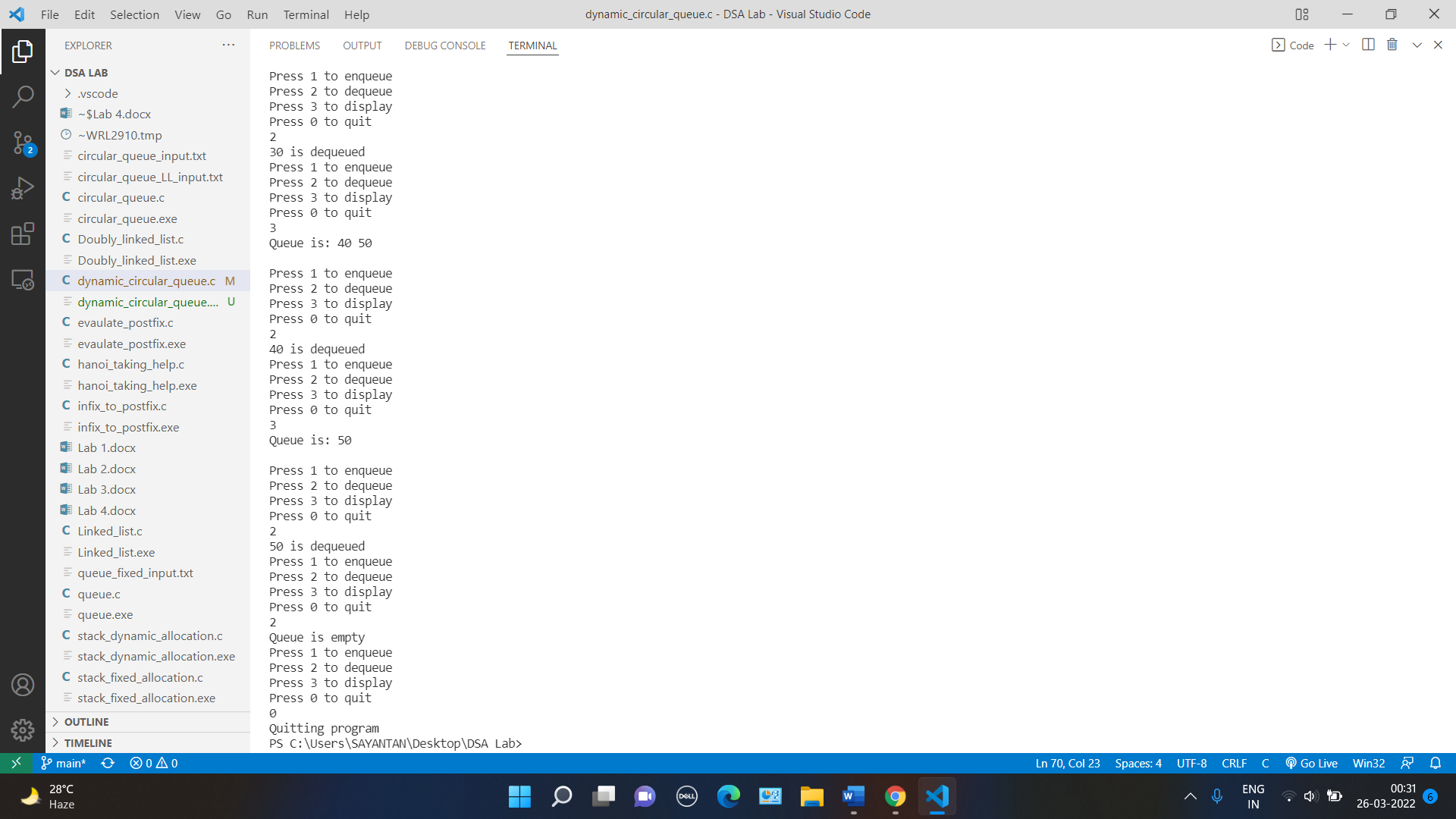
    }while(choice!=0);

    fclose(fptr);

    return 0;

}

Output

Q4. Create a dynamic memory implementation of minimum priority queue.

Code: ideone.com/EnQO33

i/p file: rb.gy/i3nm0v

Ans.

// A C program to demonstrate linked list based implementation of Minimum Priority queue

// Here the value of data is the priority and lower value means higher priority

#include <stdio.h>

#include <stdlib.h>

#include<limits.h>

// A linked list (LL) node to store a queue entry

struct QNode {

    int key;

    struct QNode\* next;

};

// The queue, front stores the front node of LL and rear stores the

// last node of LL

struct Queue {

    struct QNode \*front, \*rear;

};

// A utility function to create a new linked list node.

struct QNode\* newNode(int k)

{

    struct QNode\* temp = (struct QNode\*)malloc(sizeof(struct QNode));

    temp->key = k;

    temp->next = NULL;

    return temp;

}

// A utility function to create an empty queue

struct Queue\* createQueue()

{

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

    q->front = q->rear = NULL;

    return q;

}

// The function to add a key k to q

void enQueue(struct Queue\* q, int k)

{

    // Create a new LL node

    // Create a pointer to the front node

    struct QNode\* node =  newNode(k);

    // If queue is empty, then new node is front and rear both

    if(q->front == NULL){

        q->front = q->rear = node;

    }

    else{

        if(node->key <= q->front->key){

            node->next = q->front;

            q->front = node;

        }

        else{

            struct QNode\* store = q->front;

            while(store->next!=NULL && store->next->key<node->key){

                store = store->next;

            }

            node->next = store->next;

            store->next = node;

        }

    }

    // ELSE

        // Special Case: The head of list has lesser

        // priority than new node. So insert new

        // node before head node and change head node.

            // Insert New Node before head

        // ELSE case

            // Traverse the list and find a

            // position to insert new node

            // Either at the ends of the list

            // or at required position

}

// Function to remove a key from given queue q

int deQueue(struct Queue\* q)

{

    int item;

    // If queue is empty, return INT\_MIN.

    // Store previous front and move front one node ahead

    // If front becomes NULL, then change rear also as NULL

    if(q->front == NULL && q->rear == NULL){

        item = INT\_MIN;

    }

    else{

        struct QNode\* node = q->front;

        if(node->next == NULL){

            item = node->key;

            q->front = NULL;

            q->rear = NULL;

            node->next = NULL;

            free(node);

        }

        else{

            q->front = q->front->next;

            if(q->front == NULL){

                q->rear = q->rear->next;

            }

            node->next = NULL;

            item = node->key;

            free(node);

        }

    }

    return item;

    // Free node

    // Return removed value

}

void display(struct Queue\* q)

{

    // If queue is empty, print "Queue is Empty" in this function.

    if(q->front == NULL && q->rear == NULL){

        printf("\nQueue is Empty!\n");

    }

    else{

        struct QNode \*node = q->front;

        printf("\nQueue is: ");

        do{

            printf("%d ", node->key);

            node = node->next;

        }while (node!=NULL);

        printf("\n");

    }

    // Else traverse the list and print each element

}

// Driver program to test above functions

int main()

{

    FILE \*fptr;

    if ((fptr = fopen("priority\_queue\_LL\_input.txt","r")) == NULL)

    {

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

    }

    int choice, data;

    // Below line may be different but variable name "queue" must be same

    struct Queue\* queue = createQueue();

    do

    {

        printf("\nPress 1 to enqueue\nPress 2 to dequeue\nPress 3 to display\nPress 0 to quit");

        fscanf(fptr,"%d", &choice);

        printf("\n%d", choice);

        switch(choice)

        {

            case 1:

                printf("\nEnter value to insert: ");

                fscanf(fptr, "%d",&data);

                printf("%d", data);

                enQueue(queue, data);

                display(queue);

                break;

            case 2:

                data = deQueue(queue);

                if(data!=INT\_MIN)

                {

                    printf("\n%d is dequeued", data);

                    display(queue);

                }

                else

                    printf("\nQueue is Empty!");

                break;

            case 3:

                display(queue);

                break;

            case 0:

                printf("\nQuitting program");

                break;

            default:

                printf("\nInvalid choice");

                break;

        }

    }while(choice!=0);

    return 0;

}

Output

