Q. 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;

        }

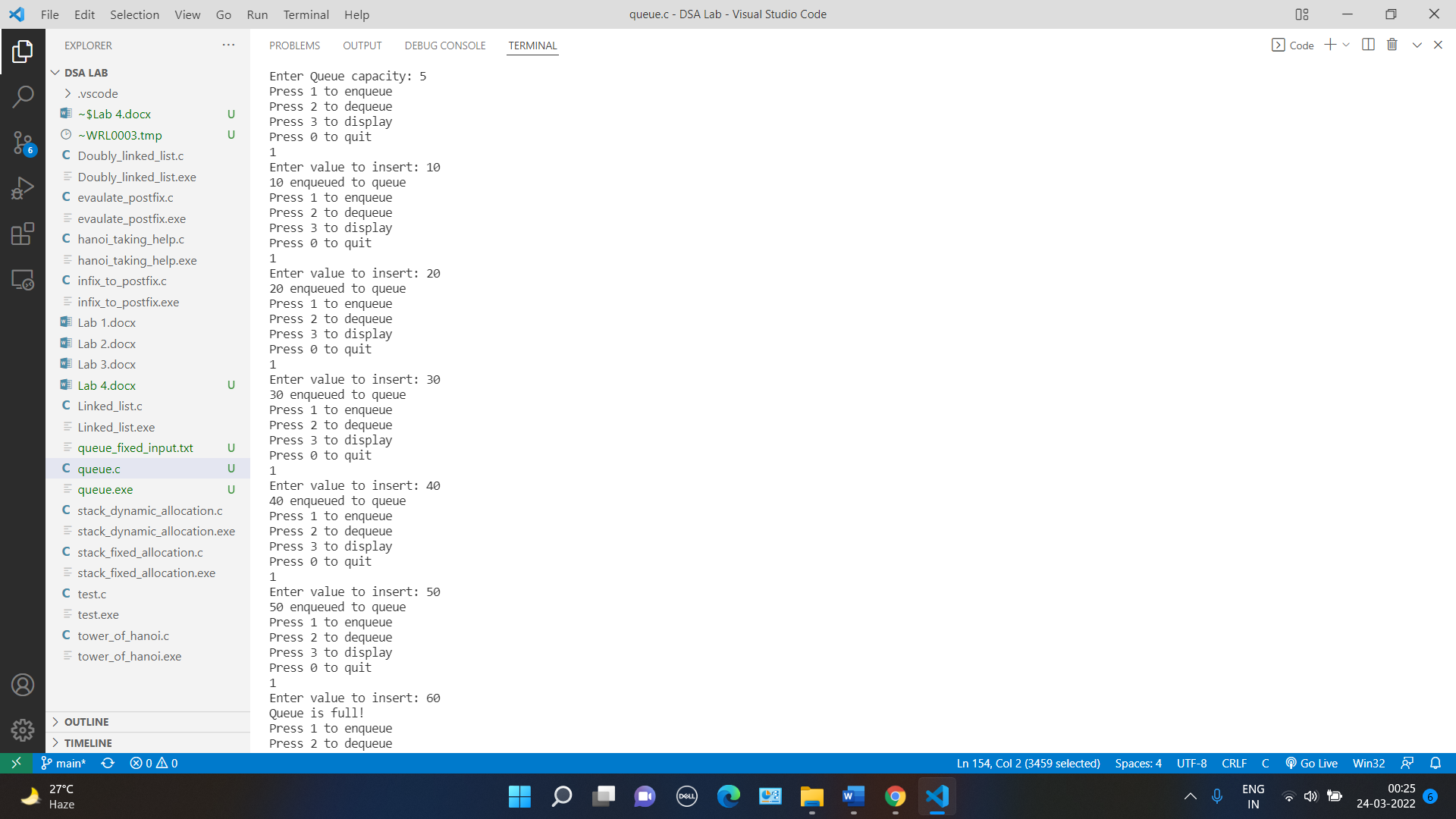
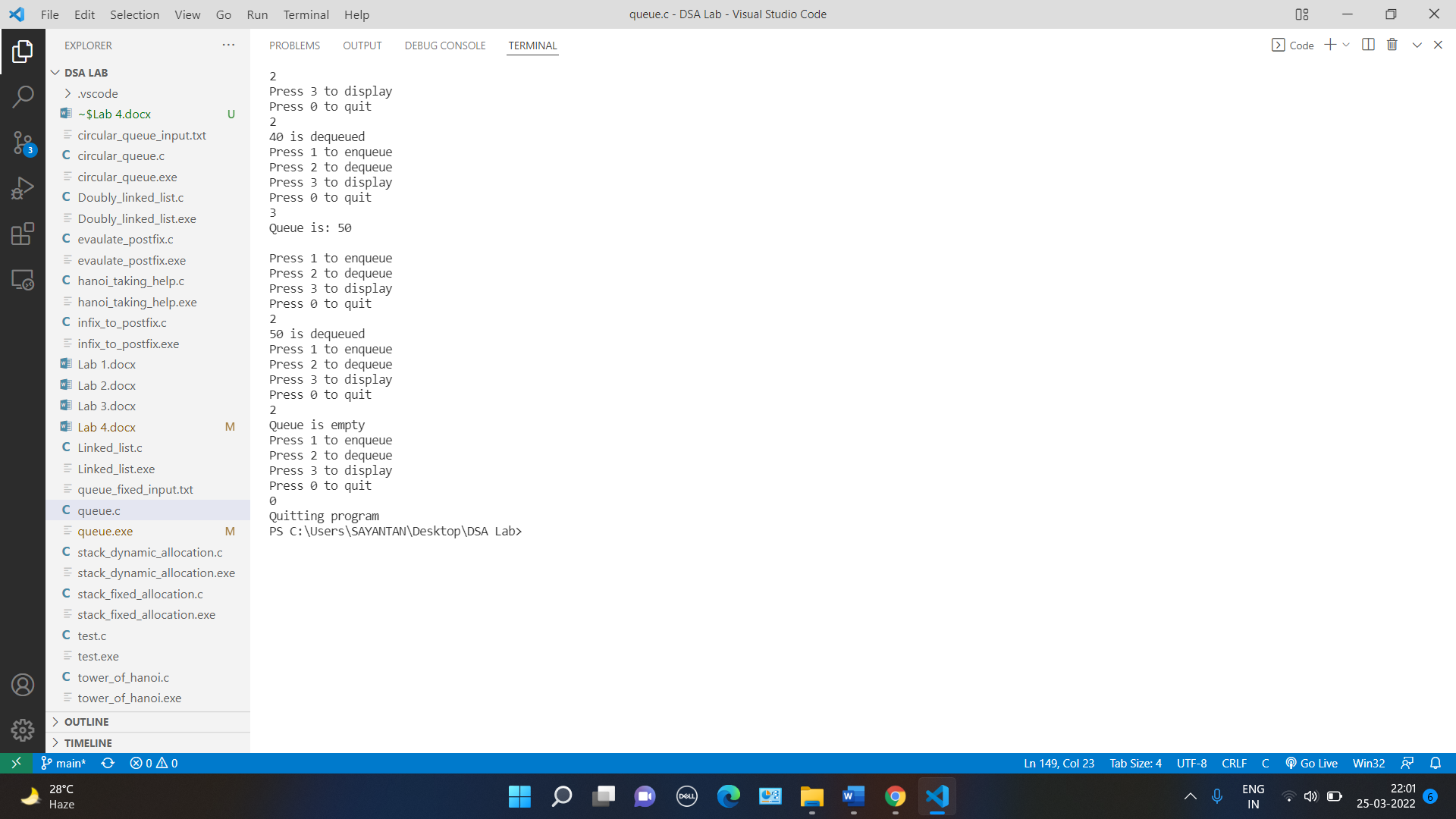
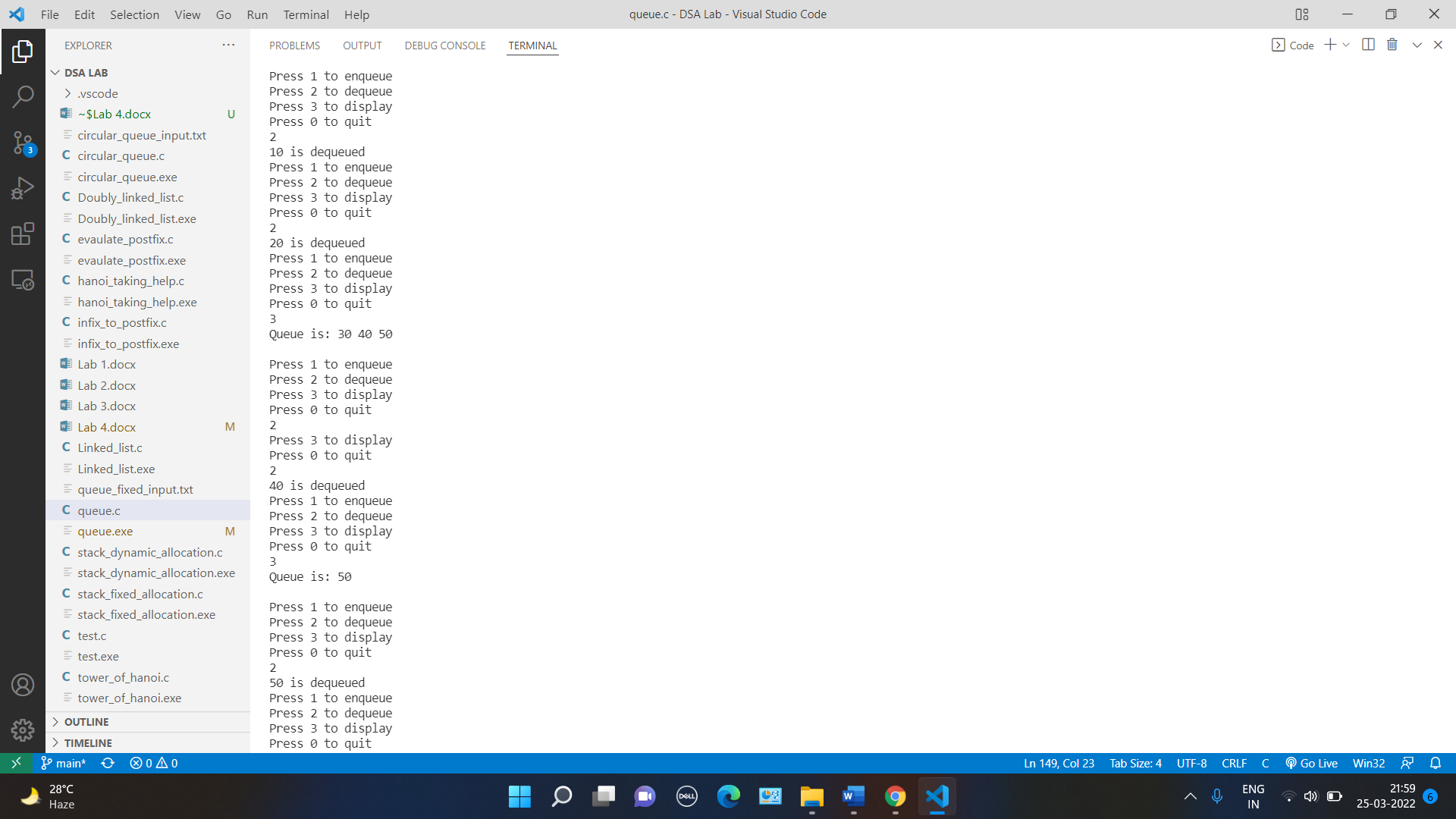
    }while(choice!=0);

    fclose(fptr);

    return 0;

}

Output

Q. 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;

        }

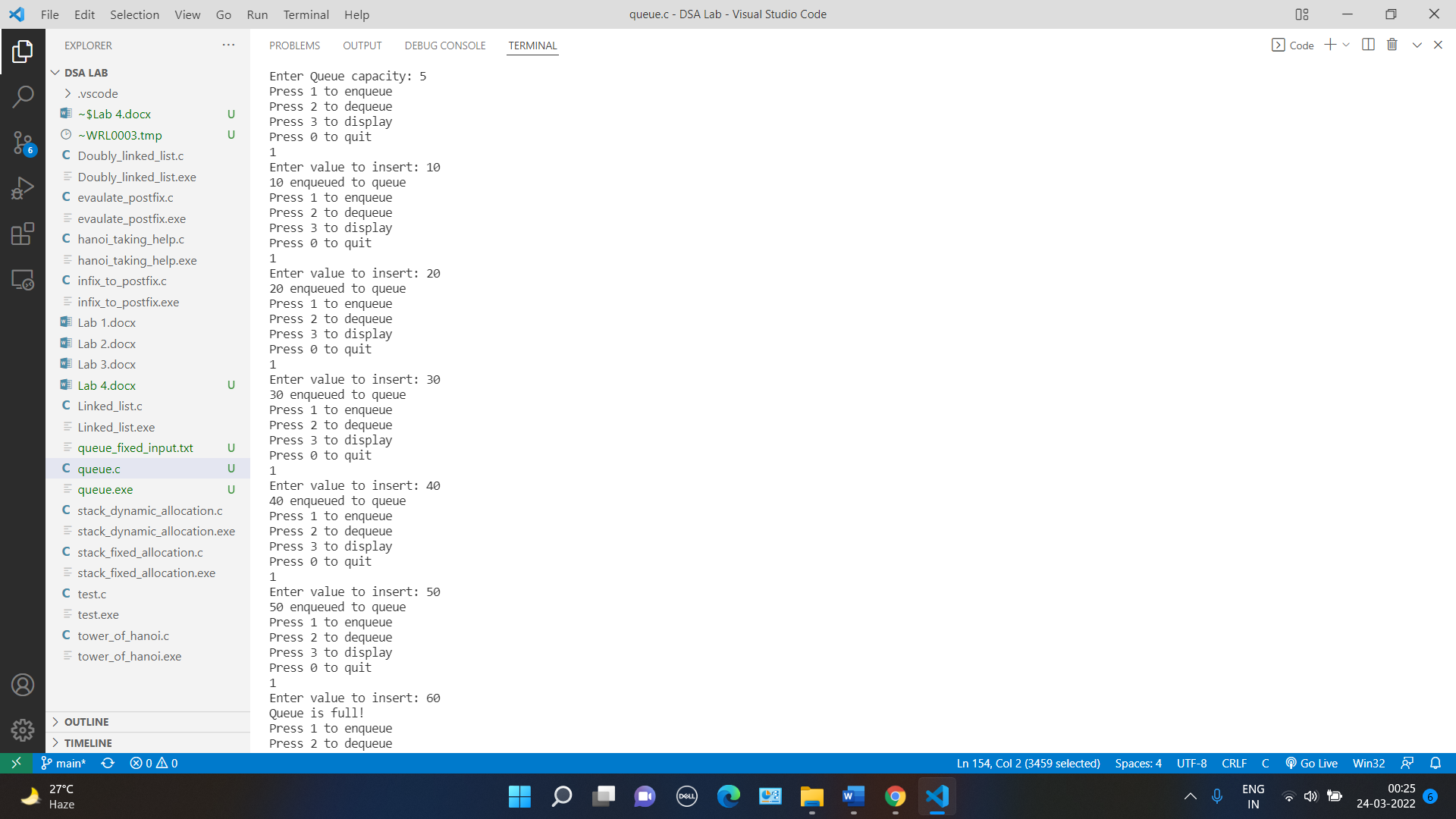
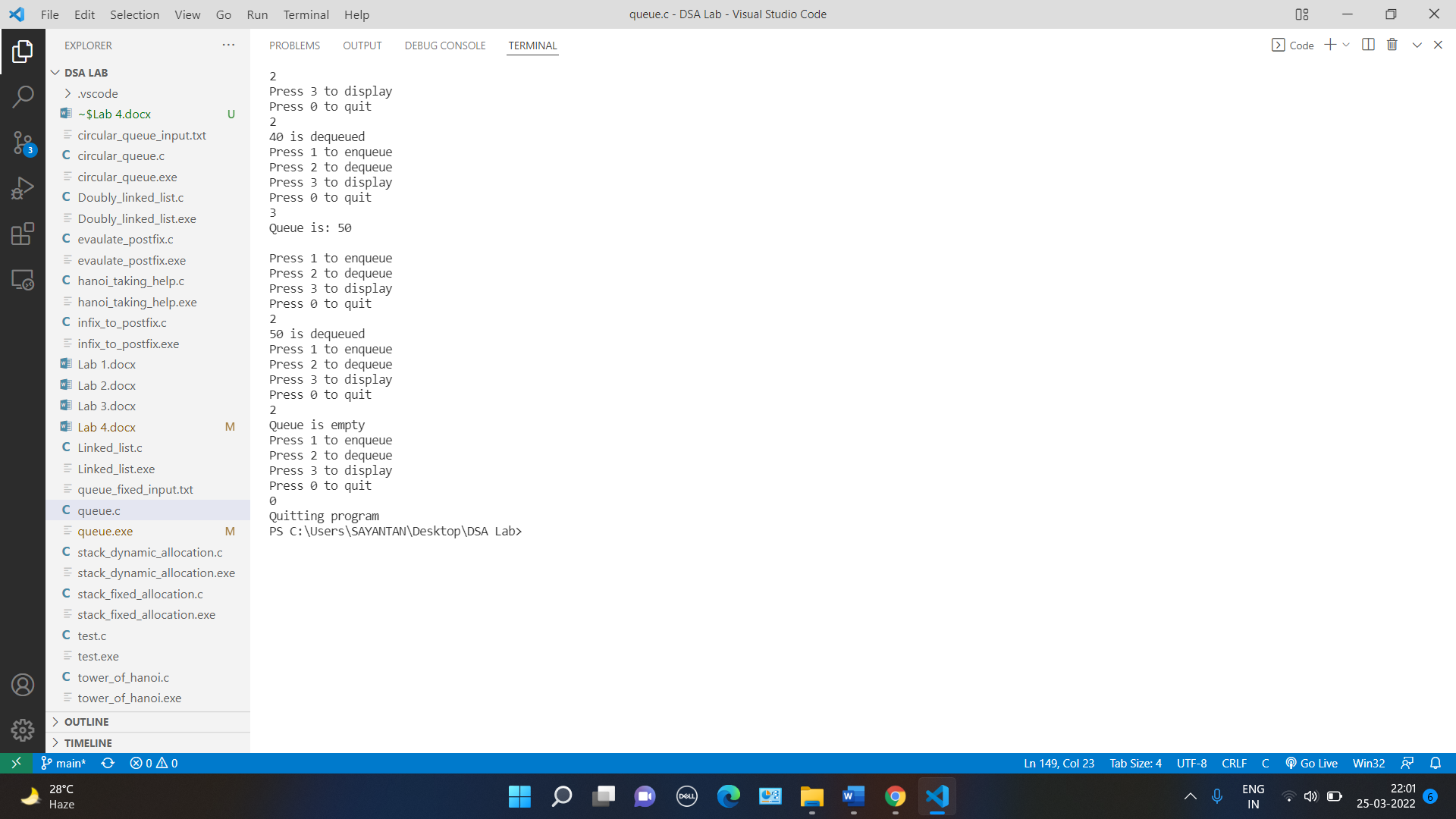
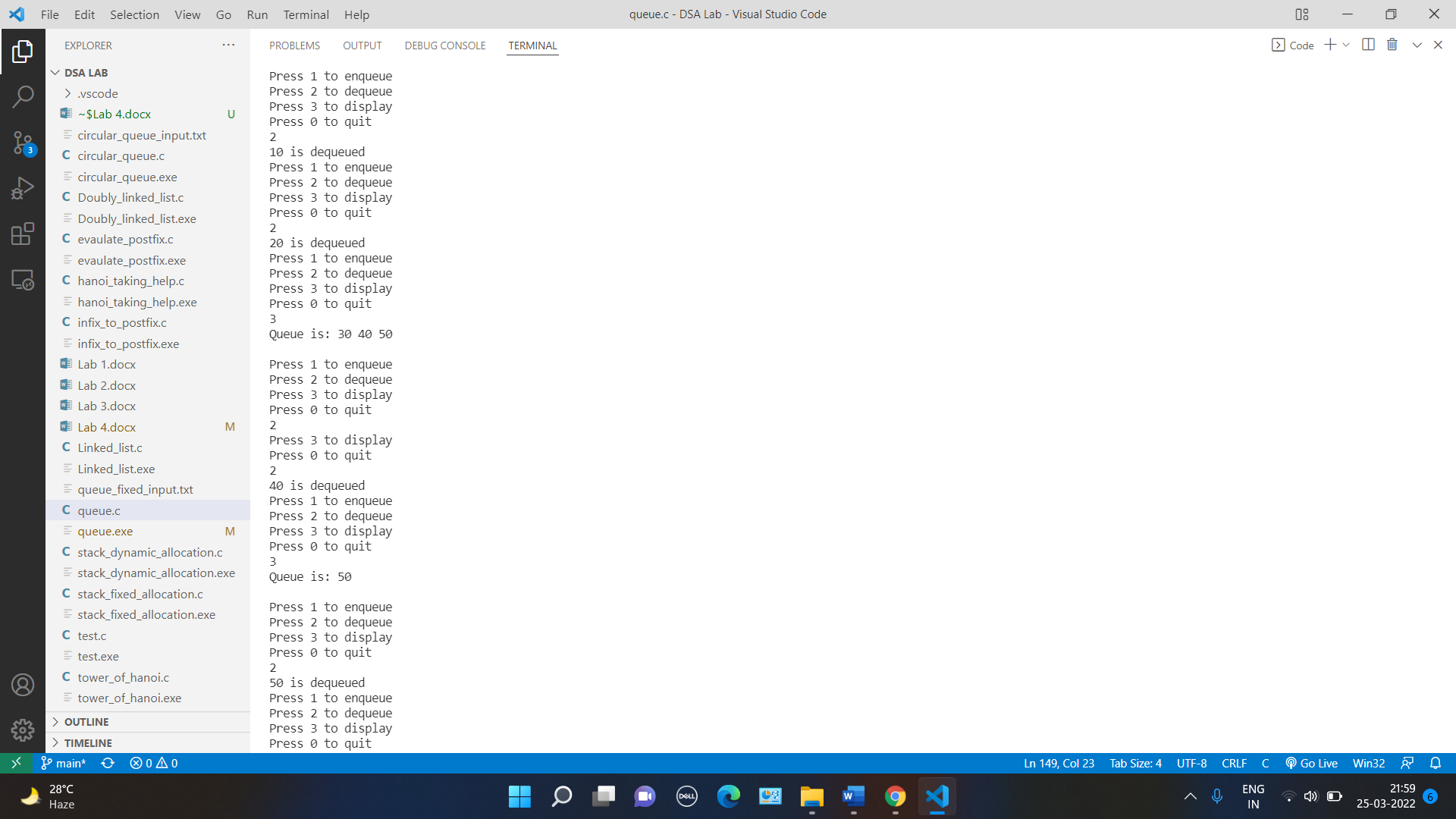
    }while(choice!=0);

    fclose(fptr);

    return 0;

}

Output

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

Code: ideone.com/QD7VE7

i/p file: rb.gy/gdy4ho

Ans.

#include<stdio.h>

#include<stdlib.h>

void hanoi(int disk\_no, char from, char aux, char to){

    if(disk\_no!=0)

    {

        hanoi(disk\_no-1, from, to, aux);

        printf("Disk %d is moved from %c to %c\n", disk\_no, from, to);

        hanoi(disk\_no-1, aux, from, to);

    }

}

int main(){

    int n;

    printf("Enter the number of disks: ");

    scanf(" %d", &n);

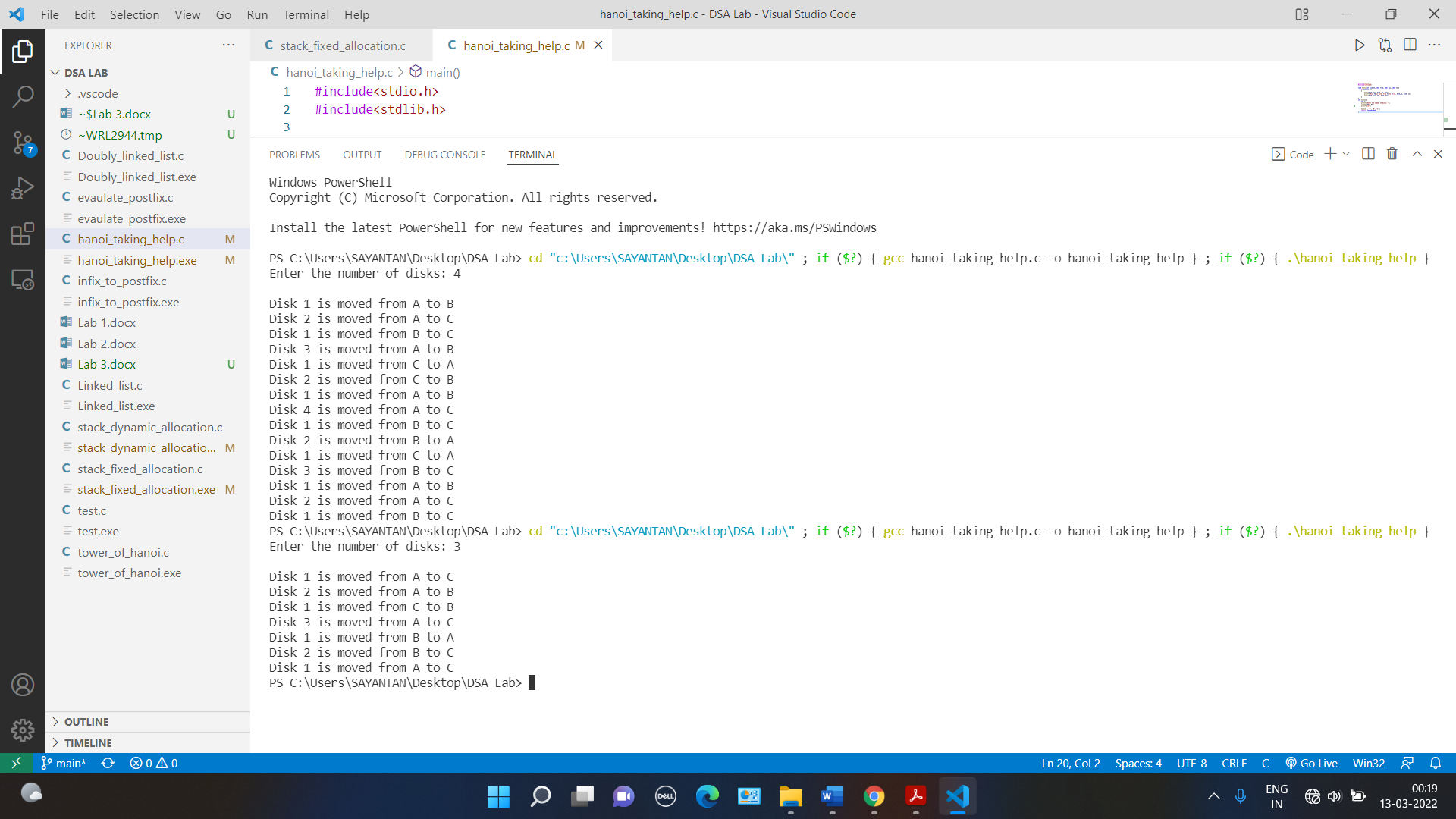
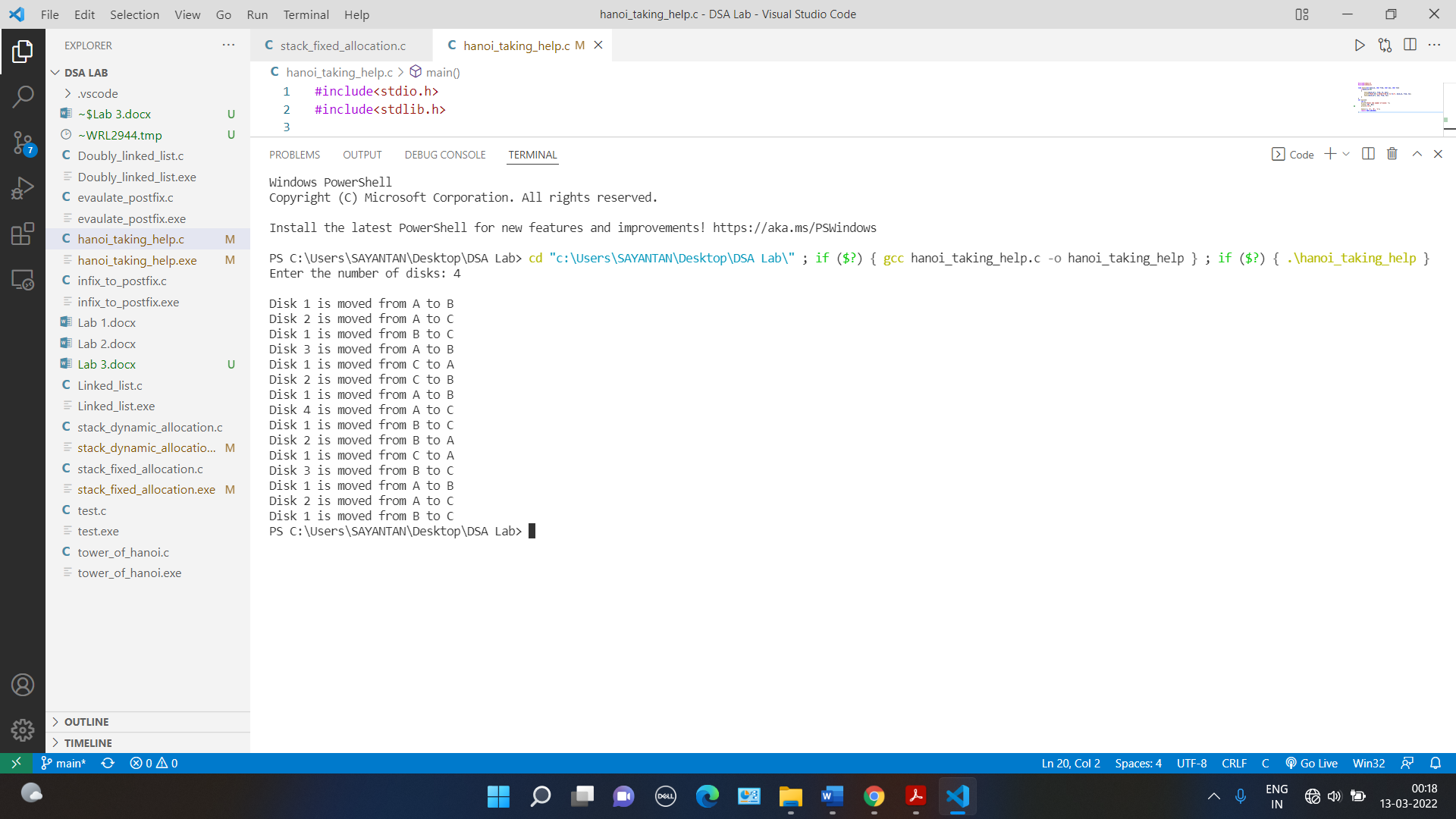
printf("\n");

    hanoi( n, 'A', 'B', 'C');

    return EXIT\_SUCCESS;

}

Output

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

Code: ideone.com/QD7VE7

i/p file: rb.gy/gdy4ho

Ans.

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<math.h>

struct Node{

    int val;

    struct Node\* next;

};

struct Stack

{

    int top;

    struct Node\* head;

};

struct Stack\* createStack()   //dynamic memory stack

{

    struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

    if (!stack)

        return NULL;

    stack->top = -1;

    stack->head = NULL;

    return stack;

}

int isEmpty(struct Stack\* stack)

{

    return stack->top == -1 ;

}

int peek(struct Stack\* stack)

{

    return stack->head->val;

}

int pop(struct Stack\* stack)

{

    int store;

    if (!isEmpty(stack)){

        struct Node\* temp = stack->head;

        stack->head = stack->head->next;

        store = temp->val;

        free(temp);

        stack->top--;

        return store;

    }

    return '$';

}

void push(struct Stack\* stack, int op)

{

    struct Node\* store = (struct Node\*)malloc(sizeof(struct Node));

    if(store){

        store->next = NULL;

        store->val = op;

        if (!isEmpty(stack)){

            store->next = stack->head;

            stack->head = store;

        }

        else{

            stack->head = store;

        }

        stack->top++;

    }

    else{

        printf("Out of memory!");

    }

}

int main(){

    int op1, op2, flag = 1;

    char ch;

    printf("Enter your postfix expression (end it with $): ");

    struct Stack \*stack = createStack();

    if(stack == NULL){

        printf("Could not create stack!");

    }

    else{

        do{

            scanf("%c", &ch);

            if(isalnum(ch)){

                push(stack, ch - '0');

            }

            else if(ch == '$')

                break;

            else{  //this means ch is an operation symbol

                op1 = pop(stack);

                op2 = pop(stack);

                if(op2 == '$'){

                    flag = 0;

                    break;

                }

                switch (ch)

                {

                case '+':

                    op2 = op2 + op1;

                    break;

                case '-':

                    op2 = op2 - op1;

                    break;

                case '/':

                    op2 = op2/op1;

                    break;

                case '\*':

                    op2 = op2 \* op1;

                    break;

                case '^':

                    op2 = pow(op2, op1);

                    break;

                default:

                    break;

                }

                push(stack, op2);

            }

        }while(ch!='$');

        if(flag == 1){

            printf("Answer: %d", pop(stack));

        }

        else{

            printf("Invalid expression!");

        }

    }

    return EXIT\_SUCCESS;

}

Output

