22AIE112 DATA STRUCTURES AND ALGORITHMS
LABSHEET 6
QUEUE

Date : 30/7/23 **Roll No : AM.EN.U4AIE2209**

1.Design a menu driven program to implement a queue using a dynamic array. Include the options for enqueue, dequeue and display operations

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

#define MAX_SIZE 100

int* createQueue() {
int* queue = (int*)malloc(MAX_SIZE * sizeof(int));
return queue;
}

int isFull(int front, int rear) {
return ((rear + 1) % MAX_SIZE == front);
}

int isEmpty(int front, int rear) {
return (front == -1);
}

void enqueue(int* queue, int* front, int* rear, int data) {
if (isFull(*front, *rear)) {
printf("Queue is full. Enqueue operation not possible\n");
return;
}

if (*front == -1)
*front = 0;

*rear = (*rear + 1) % MAX_SIZE;
queue[*rear] = data;

printf("Enqueued element: %d\n", data);
}

int dequeue(int* queue, int* front, int* rear) {
if (isEmpty(*front, *rear)) {
printf("Queue is empty \n");
```

```
return -1;
}

int data = queue[*front];
if (*front == *rear)
*front = *rear = -1;
else
*front = (*front + 1) % MAX_SIZE;

printf("Dequeued element: %d\n", data);
return data;
}

void display(int* queue, int front, int rear) {
if (isEmpty(front, rear)) {
printf("Queue is empty.\n");
return;
}

printf("Queue elements: ");
int i = front;
while (i != rear) {
printf("%d ", queue[i]);
i = (i + 1) % MAX_SIZE;
}
printf("%d\n", queue[rear]);
}

void destroyQueue(int* queue) {
free(queue);
}

int main() {
int* queue = createQueue();
int front = -1;
int rear = -1;
int opCode, data;

while (1) {
printf("\nChoose the queue operation:\n");
printf("1. Enqueue\n");
printf("2. Dequeue\n");
printf("3. Display\n");
printf("4. Exit\n");
printf("Enter your opCode: ");
scanf("%d", &opCode);

switch (opCode) {
case 1:
printf("Enter the element to enqueue: ");
scanf("%d", &data);
```

```
enqueue(queue, &front, &rear, data);
break;
case 2:
dequeue(queue, &front, &rear);
break;
case 3:
display(queue, front, rear);
break;
case 4:
destroyQueue(queue);
printf("Exiting....\n");
return 0;
default:
printf("Invalid Op code\n");
}
}

return 0;
}
```

Output

```
Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode: 1
Enter the element to enqueue: 10
Enqueued element: 10

Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode: 1
Enter the element to enqueue: 20
Enqueued element: 20

Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode: 1
Enter the element to enqueue: 3
Enqueued element: 3

Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode: 2
Dequeued element: 10

Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode: 3
```

```
Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode: 2
Dequeued element: 10

Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode: 3
Queue elements: 20 3

Choose the queue operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your opCode:
```

2.Design a menu driven program to implement a circular queue using a dynamic array. Include the options for enqueue, dequeue and display operations

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

#define MAX_SIZE 100

int* createQueue(int maxSize) {
int* queue = (int*)malloc(maxSize * sizeof(int));
return queue;
}

int isFull(int front, int rear, int size) {
return ((rear + 1) % size == front);
}

int isEmpty(int front, int rear) {
return (front == -1);
}

void enqueue(int* queue, int* front, int* rear, int size, int data) {
```

```
if (isFull(*front, *rear, size)) {
printf("Queue is full. Cannot enqueue.\n");
return;
}

if (isEmpty(*front, *rear))
*front = 0;

*rear = (*rear + 1) % size;
queue[*rear] = data;

printf("Enqueued element: %d\n", data);
}

int dequeue(int* queue, int* front, int* rear, int size) {
if (isEmpty(*front, *rear)) {
printf("Queue is empty. Cannot dequeue.\n");
return -1;
}

int data = queue[*front];

if (*front == *rear)
*front = *rear = -1;
else
*front = (*front + 1) % size;

printf("Dequeued element: %d\n", data);
return data;
}

void display(int* queue, int front, int rear, int size) {
if (isEmpty(front, rear)) {
printf("Queue is empty.\n");
return;
}

printf("Queue elements: ");
int i = front;
do {
printf("%d ", queue[i]);
i = (i + 1) % size;
} while (i != (rear + 1) % size);
printf("\n");
}

void destroyQueue(int* queue) {
free(queue);
}

int main() {
```

```
int maxSize, choice, data;
int* queue;
int front = -1;
int rear = -1;

printf("Enter the maximum size of the queue: ");
scanf("%d", &maxSize);
queue = createQueue(maxSize);

while (1) {
printf("\nChoose the queue Operation\n");
printf("1. Enqueue\n");
printf("2. Dequeue\n");
printf("3. Display\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
case 1:
printf("Enter the element to enqueue: ");
scanf("%d", &data);
enqueue(queue, &front, &rear, maxSize, data);
break;
case 2:
dequeue(queue, &front, &rear, maxSize);
break;
case 3:
display(queue, front, rear, maxSize);
break;
case 4:
destroyQueue(queue);
printf("Exit..\n");
return 0;
default:
printf("Invalid choice!!\n");
}
}

return 0;
}
```

Output

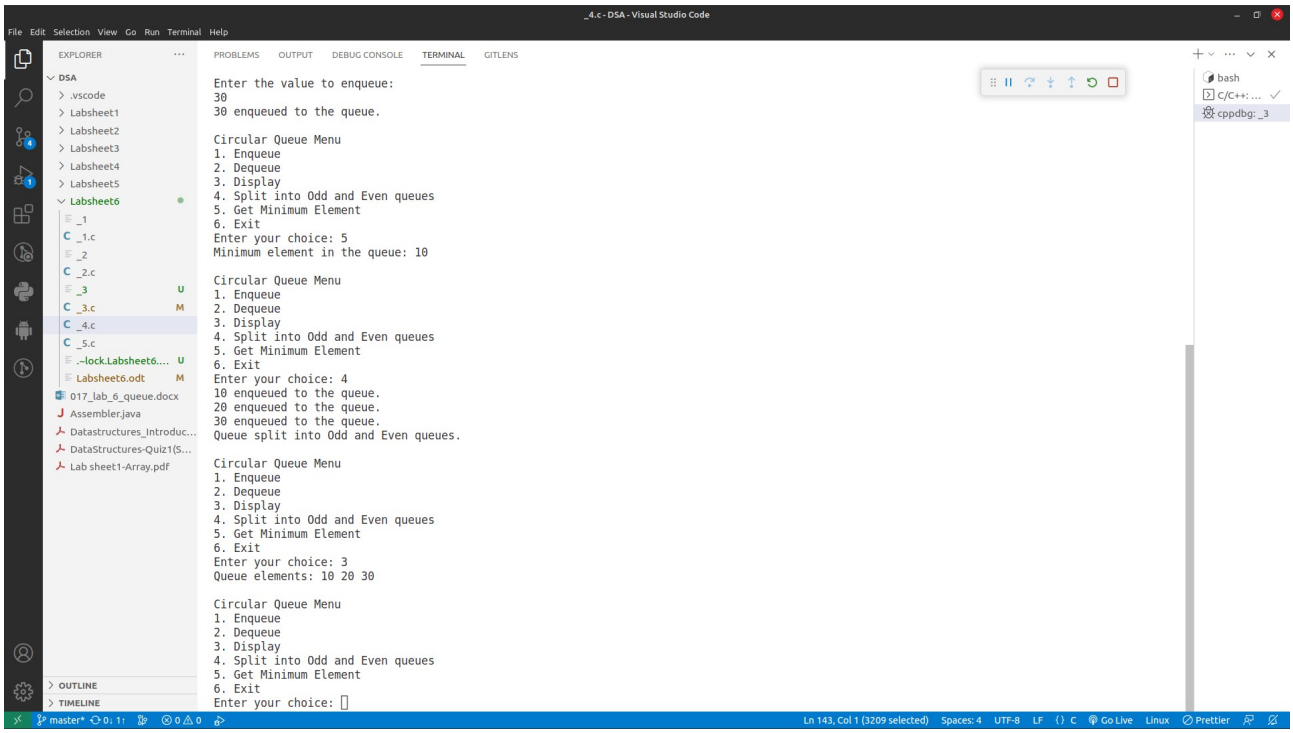
```
Choose the queue Operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 10
Enqueued element: 10

Choose the queue Operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 20
Enqueued element: 20

Choose the queue Operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 30
Enqueued element: 30

Choose the queue Operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued element: 10

Choose the queue Operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 20 30
```

3.Implement the following methods in the above circular queue: (a) splitq(), to split a queue into two queues so that all items in odd positions are in one queue and those in even positions are in another queue. (b) getminElement () to return the minimum element in a queue.

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

#define MAX_SIZE 5

struct Deque {
int arr[MAX_SIZE];
int front, rear;
};

struct Deque* createDeque() {
struct Deque* deque = (struct Deque*)malloc(sizeof(struct Deque));
deque->front = -1;
deque->rear = -1;
return deque;
}

bool is_empty(struct Deque* deque) {
return deque->front == -1;
}

bool is_full(struct Deque* deque) {
return (deque->rear + 1) % MAX_SIZE == deque->front;
}

void insertFront(struct Deque* deque, int data) {
if (is_full(deque)) {
printf("Deque is full. InsertFront operation failed.\n");
return;
}

if (is_empty(deque)) {
deque->front = deque->rear = 0;
} else {
deque->front = (deque->front - 1 + MAX_SIZE) % MAX_SIZE;
}

deque->arr[deque->front] = data;
printf("%d inserted at the front of the deque.\n", data);
}

void insertLast(struct Deque* deque, int data) {
```

```
if (is_full(deque)) {
printf("Deque is full. InsertLast operation failed.\n");
return;
}

if (is_empty(deque)) {
deque->front = deque->rear = 0;
} else {
deque->rear = (deque->rear + 1) % MAX_SIZE;
}

deque->arr[deque->rear] = data;
printf("%d inserted at the last of the deque.\n", data);
}

void deleteFront(struct Deque* deque) {
if (is_empty(deque)) {
printf("Deque is empty. DeleteFront operation failed.\n");
return;
}

int data = deque->arr[deque->front];
if (deque->front == deque->rear) {
deque->front = deque->rear = -1;
} else {
deque->front = (deque->front + 1) % MAX_SIZE;
}

printf("%d deleted from the front of the deque.\n", data);
}

void deleteLast(struct Deque* deque) {
if (is_empty(deque)) {
printf("Deque is empty. DeleteLast operation failed.\n");
return;
}

int data = deque->arr[deque->rear];
if (deque->front == deque->rear) {
deque->front = deque->rear = -1;
} else {
deque->rear = (deque->rear - 1 + MAX_SIZE) % MAX_SIZE;
}

printf("%d deleted from the last of the deque.\n", data);
}

int getFront(struct Deque* deque) {
if (is_empty(deque)) {
printf("Deque is empty.\n");
return -1;
```

```
}

return deque->arr[deque->front];
}

int getRear(struct Deque* deque) {
if (is_empty(deque)) {
printf("Deque is empty.\n");
return -1;
}

return deque->arr[deque->rear];
}

void display(struct Deque* deque) {
if (is_empty(deque)) {
printf("Deque is empty.\n");
return;
}

int i = deque->front;
printf("Deque elements: ");
do {
printf("%d ", deque->arr[i]);
i = (i + 1) % MAX_SIZE;
} while (i != (deque->rear + 1) % MAX_SIZE);
printf("\n");
}

int main() {
struct Deque* deque = createDeque();

insertFront(deque, 10);
insertLast(deque, 20);
insertFront(deque, 30);
deleteFront(deque);
deleteLast(deque);
insertLast(deque, 25);
insertFront(deque, 40);
insertFront(deque, 50);

printf("Rear element: %d\n", getRear(deque));
printf("Front element: %d\n", getFront(deque));

display(deque);

return 0;
}
```



4.Implement the following operations on Deque(Double Ended Queue) using a circular array.

(a) insertFront(): Adds an item at the front of Deque.
(b) insertLast(): Adds an item at the rear of Deque.
(c) deleteFront(): Deletes an item from front of Deque.
(d) deleteLast(): Deletes an item from rear of Deque.
(e) getFront(): Gets the front item from queue.
(f) getRear(): Gets the last item from queue.
(g) isEmpty(): Checks whether Deque is empty or not.
(h) isFull(): Checks whether Deque is full or not.
(i) display(): Display queue elements starting from front to rear

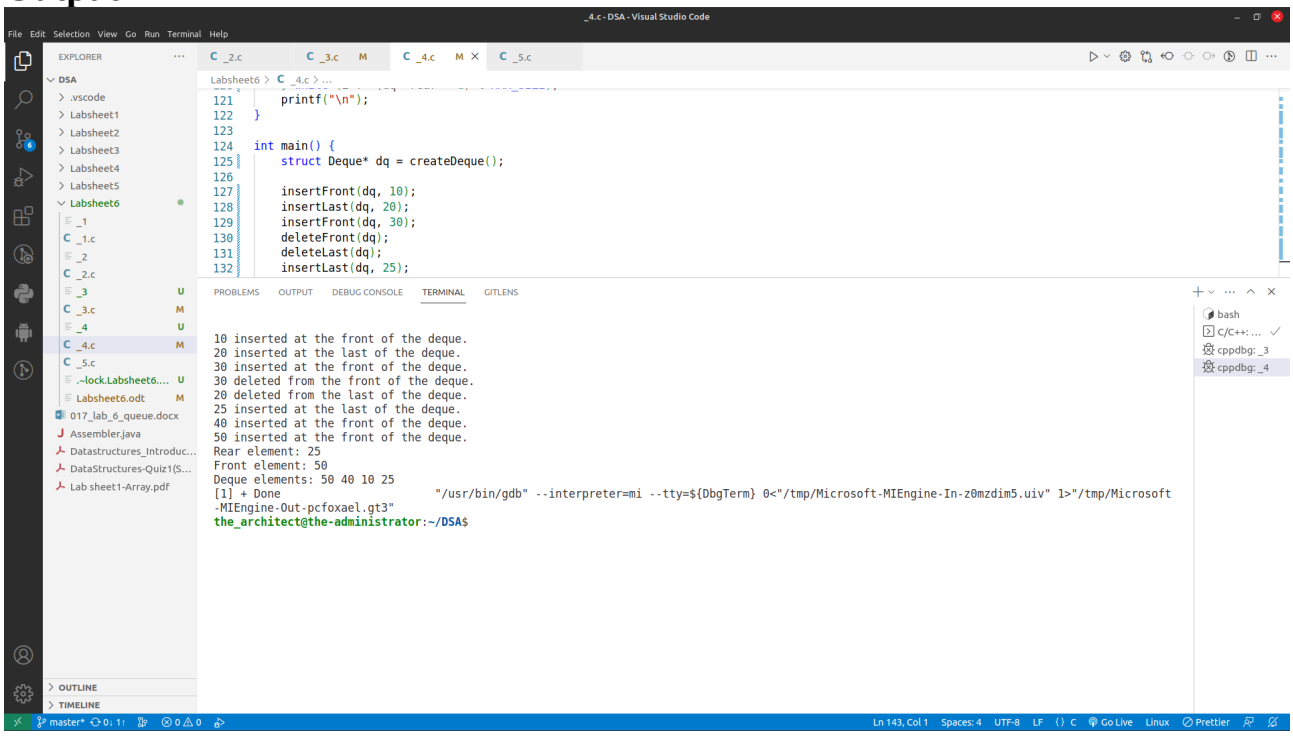
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_SIZE 5

struct Deque {
int elements[MAX_SIZE];
int front, rear;
};

struct Deque* createDeque() {
struct Deque* dq = (struct Deque*)malloc(sizeof(struct Deque));
dq->front = -1;
```

```
dq->rear = -1;
return dq;
}

bool is_empty(struct Deque* dq) {
return dq->front == -1;
}

bool is_full(struct Deque* dq) {
return (dq->rear + 1) % MAX_SIZE == dq->front;
}

void insertFront(struct Deque* dq, int data) {
if (is_full(dq)) {
printf("Deque is full. InsertFront operation failed.\n");
return;
}

if (is_empty(dq)) {
dq->front = dq->rear = 0;
} else {
dq->front = (dq->front - 1 + MAX_SIZE) % MAX_SIZE;
}

dq->elements[dq->front] = data;
printf("%d inserted at the front of the deque.\n", data);
}

void insertLast(struct Deque* dq, int data) {
if (is_full(dq)) {
printf("Deque is full. InsertLast operation failed.\n");
return;
}

if (is_empty(dq)) {
dq->front = dq->rear = 0;
} else {
dq->rear = (dq->rear + 1) % MAX_SIZE;
}

dq->elements[dq->rear] = data;
printf("%d inserted at the last of the deque.\n", data);
}

void deleteFront(struct Deque* dq) {
if (is_empty(dq)) {
printf("Deque is empty. DeleteFront operation failed.\n");
return;
}

int data = dq->elements[dq->front];
```

```
if (dq->front == dq->rear) {
dq->front = dq->rear = -1;
} else {
dq->front = (dq->front + 1) % MAX_SIZE;
}

printf("%d deleted from the front of the deque.\n", data);
}

void deleteLast(struct Deque* dq) {
if (is_empty(dq)) {
printf("Deque is empty. DeleteLast operation failed.\n");
return;
}

int data = dq->elements[dq->rear];
if (dq->front == dq->rear) {
dq->front = dq->rear = -1;
} else {
dq->rear = (dq->rear - 1 + MAX_SIZE) % MAX_SIZE;
}

printf("%d deleted from the last of the deque.\n", data);
}

int getFront(struct Deque* dq) {
if (is_empty(dq)) {
printf("Deque is empty.\n");
return -1;
}

return dq->elements[dq->front];
}

int getRear(struct Deque* dq) {
if (is_empty(dq)) {
printf("Deque is empty.\n");
return -1;
}

return dq->elements[dq->rear];
}

void display(struct Deque* dq) {
if (is_empty(dq)) {
printf("Deque is empty.\n");
return;
}

int i = dq->front;
printf("Deque elements: ");
```

```
do {
printf("%d ", dq->elements[i]);
i = (i + 1) % MAX_SIZE;
} while (i != (dq->rear + 1) % MAX_SIZE);
printf("\n");
}

int main() {
struct Deque* dq = createDeque();

insertFront(dq, 10);
insertLast(dq, 20);
insertFront(dq, 30);
deleteFront(dq);
deleteLast(dq);
insertLast(dq, 25);
insertFront(dq, 40);
insertFront(dq, 50);

printf("Rear element: %d\n", getRear(dq));
printf("Front element: %d\n", getFront(dq));

display(dq);

return 0;
}
```

Output



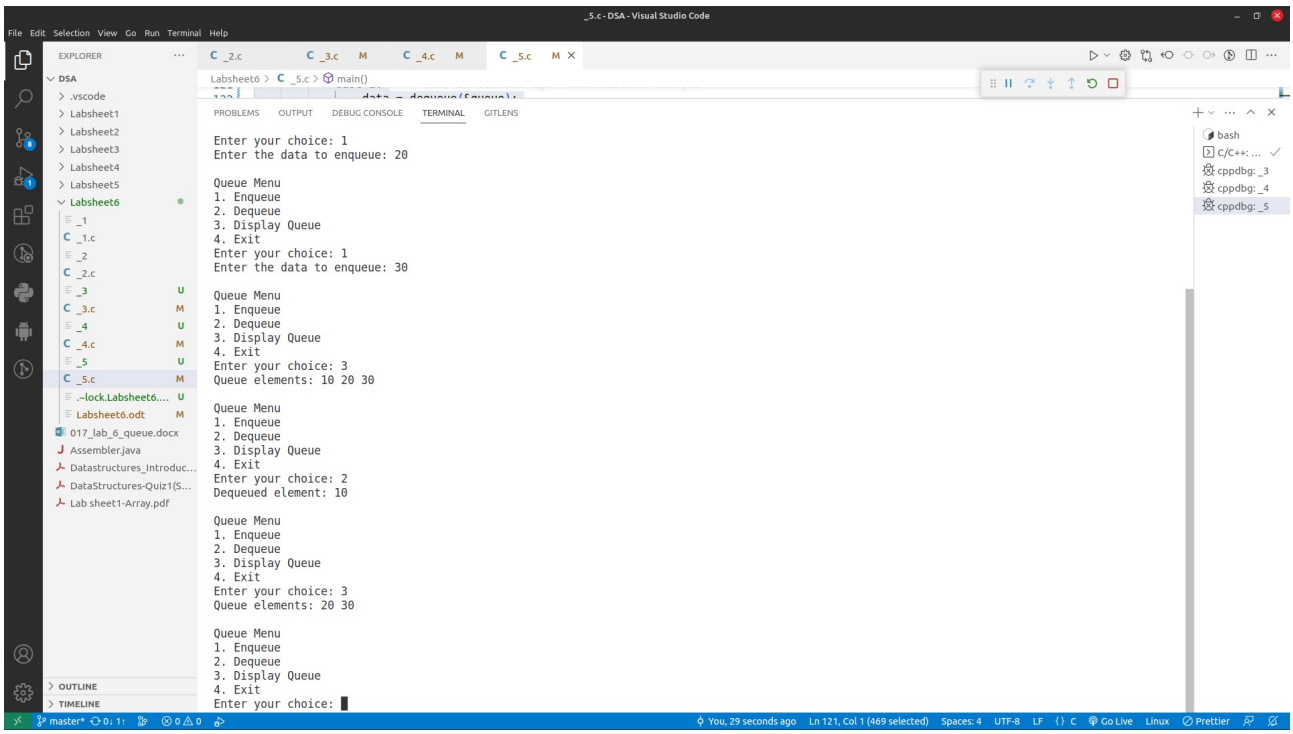
5. You are given a stack data structure with push and pop operations. Implement a queue using instances of stack data structure and operations on it.(ie, Implement queue using stack)

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

#define MAX_SIZE 100

struct Stack {
int elements[MAX_SIZE];
int top;
};

void init_stack(struct Stack* stack) {
stack->top = -1;
}

bool is_stack_empty(struct Stack* stack) {
return stack->top == -1;
}

bool is_stack_full(struct Stack* stack) {
return stack->top == MAX_SIZE - 1;
}

void push(struct Stack* stack, int data) {
if (is_stack_full(stack)) {
printf("Stack overflow. Push operation failed.\n");
return;
}

stack->top++;
stack->elements[stack->top] = data;
}

int pop(struct Stack* stack) {
if (is_stack_empty(stack)) {
printf("Stack underflow. Pop operation failed.\n");
return -1;
}

int data = stack->elements[stack->top];
stack->top--;
return data;
}

struct Queue {
struct Stack stack1;
```

```
struct Stack stack2;
};

void init_queue(struct Queue* queue) {
init_stack(&(queue->stack1));
init_stack(&(queue->stack2));
}

bool is_queue_empty(struct Queue* queue) {
return is_stack_empty(&(queue->stack1)) && is_stack_empty(&(queue->stack2));
}

void enqueue(struct Queue* queue, int data) {
if (is_stack_full(&(queue->stack1))) {
printf("Queue is full. Enqueue operation failed.\n");
return;
}

push(&(queue->stack1), data);
}

int dequeue(struct Queue* queue) {
if (is_queue_empty(queue)) {
printf("Queue is empty. Dequeue operation failed.\n");
return -1;
}

if (is_stack_empty(&(queue->stack2))) {
while (!is_stack_empty(&(queue->stack1))) {
int data = pop(&(queue->stack1));
push(&(queue->stack2), data);
}
}

return pop(&(queue->stack2));
}

void display_queue(struct Queue* queue) {
printf("Queue elements: ");
struct Stack* s1 = &(queue->stack1);
struct Stack* s2 = &(queue->stack2);

for (int i = s2->top; i >= 0; i--) {
printf("%d ", s2->elements[i]);
}

for (int i = 0; i <= s1->top; i++) {
printf("%d ", s1->elements[i]);
}

printf("\n");
```

```
}

int main() {
struct Queue queue;
init_queue(&queue);

int choice, data;

while (1) {
printf("\nQueue Menu\n");
printf("1. Enqueue\n");
printf("2. Dequeue\n");
printf("3. Display Queue\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
case 1:
printf("Enter the data to enqueue: ");
scanf("%d", &data);
enqueue(&queue, data);
break;
case 2:
data = dequeue(&queue);
if (data != -1)
printf("Dequeued element: %d\n", data);
break;
case 3:
display_queue(&queue);
break;
case 4:
printf("Exiting the program.\n");
exit(0);
default:
printf("Invalid choice. Please try again.\n");
}
}

return 0;
}
```

Output


_5.c - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
DSA
.vscode
Labsheet1
Labsheet2
Labsheet3
Labsheet4
Labsheet5
Labsheet6
_1
_1.c
_2
_2.c
_3
_3.c
_4
_4.c
_5
_5.c
~lock.Labsheet6....
Labsheet6.odt
017_lab_6_queue.docx
Assembler.java
Datastructures_Introduc...
DataStructures-Quiz1(S...
Lab sheet1-Array.pdf
OUTLINE
TIMELINE
_2.c
_3.c M
_4.c M
_5.c M
Labsheet6 > _5.c > main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS
Enter your choice: 1
Enter the data to enqueue: 20
Queue Menu
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the data to enqueue: 30
Queue Menu
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements: 10 20 30
Queue Menu
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
Dequeued element: 10
Queue Menu
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements: 20 30
Queue Menu
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice:
bash
C/C++: ...
cppdbg: _3
cppdbg: _4
cppdbg: _5
master*
You, 29 seconds ago
Ln 121, Col 1 (469 selected)
Spaces: 4
UTF-8
LF
C
Go Live
Linux
Prettier