Assignment 5: QueueADT and its application

Create an ADT for the circular queue data structure using Singly Linked List with the following functions. Each node which consists of Job ID and Burst time. [CO1, K3]

a. **createQueue(Q,size) – initialize size**

b. **enqueue(Q, data) – enqueue data into the queue**

c. **dequeue(Q)– returns the element at front**

d. **isEmpty(Q) – returns 1 if queue empty, otherwise returns 0**

e. **display(Q) – display the elements in queue**

Test the operations of queueADT with the following test cases

Operation	Expected Output
dequeue(Q)	Empty
enqueue(Q, J1, 2)	(J1,2)
enqueue(Q, J2, 13)	(J1,2), (J2,13)
enqueue(Q, J3, 5)	(J1,2), (J2,13),(J3,5)
dequeue(Q)	(J1,2)
display(Q)	(J2,13),(J3,5)
dequeue(Q)	(J2,13)
display(Q)	(J3,5)

Best practices to be followed:

- Design before coding
- Usage of algorithm notation
- Use of multi-file C program
- Versioning of code

Application using Queue

1. Job scheduling

Insert queue with the following contents

(J1,2), (J2,4), (J3,8), (J4,5), (J5,2), (J6,7), (J7,4), (J8,3) (J9,6) & (J10,6)
Insert the job into the queue whichever is empty. If it is not empty, insert the job into the queue whichever is having minimum average time

Display the jobs waiting in both the queues along with their CPU burst time.

2. Build Queue using stacks

DESIGN:

Queue Data Structure

QueADT.h

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

struct stack {
        int size;
        int top;
```

```
        int data[100];
};

void init(struct stack *s, int limit) {
        s->size = limit;
        s->top = -1;
}


int isFull(struct stack *s) {
        if (s->top < s->size - 1) {
        return 0;
        } else {
        return 1;
        }
}

int isEmpty(struct stack *s) {
        if (s->top == -1) {
        return 1;
        } else {
        return 0;
        }
}


void push(struct stack *s, int x) {
        if (!isFull(s)) {
        s->data[++(s->top)] = x;
        printf("\nPushed %d\n", x);
        } else {
        printf("\nStack is full");
        }
}

void pop(struct stack *s) {
        if (!isEmpty(s)) {
        s->top--;
        } else {
        printf("\nStack is empty");
        }
}


int peek(struct stack *s) {
        if (!isEmpty(s)) {
        return s->data[s->top];
```

```
        } else {
        return -1;
        }
}


struct QueueWithStacks {
        struct stack stack1;
        struct stack stack2;
};


void initQueue(struct QueueWithStacks *q, int limit) {
        init(&q->stack1, limit);
        init(&q->stack2, limit);
}

void enqueue(struct QueueWithStacks *q, int x) {
        while (!isEmpty(&q->stack2)) {
        push(&q->stack1, peek(&q->stack2));
        pop(&q->stack2);
        }
        push(&q->stack1, x);
}


int dequeue(struct QueueWithStacks *q) {
        while (!isEmpty(&q->stack1)) {
        push(&q->stack2, peek(&q->stack1));
        pop(&q->stack1);
        }
        int x = peek(&q->stack2);
        pop(&q->stack2);
        return x;
}
```

Main.c

```
int main() {
        struct QueueWithStacks queue;
        initQueue(&queue, 3);

        enqueue(&queue, 1);
        enqueue(&queue, 2);
        enqueue(&queue, 3);
        printf("Dequeued: %d\n", dequeue(&queue));
        printf("Dequeued: %d\n", dequeue(&queue));
```

```
        //enqueue(&queue, 4);
        printf("Dequeued: %d\n", dequeue(&queue));
        printf("Dequeued: %d\n", dequeue(&queue));

        return 0;
}
```

Output:

```
Pushed 1

Pushed 2

Pushed 3

Pushed 3

Pushed 2

Pushed 1
Dequeued: 1
Dequeued: 2
Dequeued: 3

Stack is emptyDequeued: -1
```

ADDITIONAL QUESTIONS:

QueADT.h

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

struct Job
{
```

```
        char name[10];
        int burst_time;
};

struct Queue
{
        struct Job data[100];
        int f, r, size;
};

void createQ(struct Queue *Q, int limit)
{
        Q->size = limit;
        Q->f = -1;
        Q->r = -1;
}

void Enqueue(struct Queue *Q, char name[], int burst_time)
{
        if ((Q->r + 1) % Q->size == Q->f)
        {
        printf("Queue is full\n");
        }
        else
        {
        if (Q->f == -1 && Q->r == -1)
        {
        Q->f = 0;
        Q->r = 0;
        }
        else
        {
        Q->r = (Q->r + 1) % Q->size;
        }

        strcpy(Q->data[Q->r].name, name);
        Q->data[Q->r].burst_time = burst_time;
        }
}

int Dequeue(struct Queue *Q)
{
        if (Q->f == -1 && Q->r == -1)
        {
        return -1;
        }
        else
        {
```

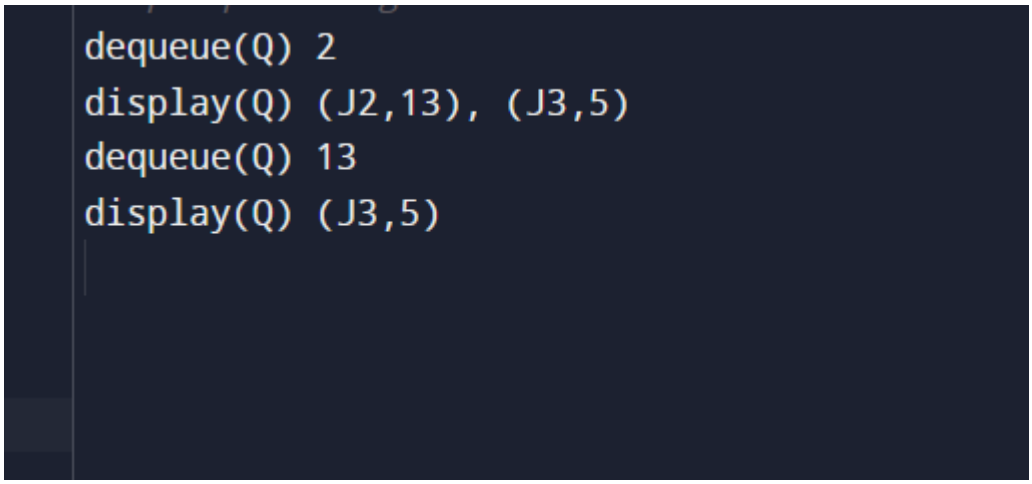
```
        int burst_time = Q->data[Q->f].burst_time;

        if (Q->f == Q->r)
        {
        Q->f = -1;
        Q->r = -1;
        }
        else
        {
        Q->f = (Q->f + 1) % Q->size;
        }

        return burst_time;
        }
}

int isEmpty(struct Queue *Q)
{
        return Q->f == -1 && Q->r == -1;
}

void display(struct Queue *Q)
{
        int i = Q->f;
        if (i != -1) {
        printf("(%s,%d)", Q->data[i].name, Q->data[i].burst_time);
        i = (i + 1) % Q->size;
        }
        while (i != (Q->r + 1) % Q->size)
        {
        printf(", (%s,%d)", Q->data[i].name, Q->data[i].burst_time);
        i = (i + 1) % Q->size;
        }
        printf("\n");
}
```

Main.c

```
int main(void)
{
        struct Queue Q;
        createQ(&Q, 10);

        // Insert jobs as specified
        Enqueue(&Q, "J1", 2);
        Enqueue(&Q, "J2", 13);
        Enqueue(&Q, "J3", 5);
```

```
    // Test the operations
    printf("dequeue(Q) %d\n", Dequeue(&Q));

    printf("display(Q) ");
    display(&Q);

    printf("dequeue(Q) %d\n", Dequeue(&Q));

    printf("display(Q) ");
    display(&Q);

    return 0;
}
```

Output:



2.Build a Queue using stuck

ALGORITHM:

Input: Two stacks :inputStack and outputStack
Output:void

Operation:

Initialize Queue:

Initialize both “inputStack” and “outputStack”.

Enqueue:
To enqueue an element:
1.While the “**outputStack**”is not empty, pop elements from the “outputStack” and

push them onto the “**inputStack”.**

2.Push the new element onto the “inputStack”.

Dequeue:
To dequeue an element:
1.if the “ouputStack” is not empty,pop an element from the “outputStack”.
2.if the “ouput”is empty,and the “inputStack”is not empty:
While the “inputSatck” is not empty,pop elements from the “inputStack” and push them onto the “outputStack”.
Pop an element from the “output”.

QueADT.h

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

// Stack structure
struct Stack {
	int data[100];
	int top;
};

// Initialize a stack
void initializeStack(struct Stack *stack) {
	stack->top = -1;
}

// Check if the stack is empty
int isEmpty(struct Stack *stack) {
	return stack->top == -1;
}

// Check if the stack is full
int isFull(struct Stack *stack) {
	return stack->top == MAX_SIZE - 1;
}

// Push an element onto the stack
void push(struct Stack *stack, int value) {
	if (!isFull(stack)) {
	stack->data[++stack->top] = value;
	}
}
```

```
// Pop an element from the stack
int pop(struct Stack *stack) {
        if (!isEmpty(stack)) {
        return stack->data[stack->top--];
        }
        return -1; // Stack is empty
}

// Queue structure using two stacks
struct QueueUsingStacks {
        struct Stack inputStack;  // For enqueue operation
        struct Stack outputStack; // For dequeue operation
};

// Initialize the queue
void initializeQueue(struct QueueUsingStacks *queue) {
        initializeStack(queue->inputStack);
        initializeStack(queue->outputStack);
}

// Enqueue an element into the queue
void enqueue(struct QueueUsingStacks *queue, int value) {
        if (!isFull(queue->inputStack)) {
        while (!isEmpty(queue->outputStack)) {
        push(queue->inputStack, pop(queue->outputStack));
        }
        push(queue->inputStack, value);
        printf("Enqueue: %d\n", value);
        }
}

// Dequeue an element from the queue
int dequeue(struct QueueUsingStacks *queue) {
        if (!isEmpty(queue->outputStack)) {
        int value = pop(queue->outputStack);
        printf("Dequeue: %d\n", value);
        return value;
        } else if (!isEmpty(queue->inputStack)) {
        while (!isEmpty(queue->inputStack)) {
        push(queue->outputStack, pop(queue->inputStack));
        }
        int value = pop(queue->outputStack);
        printf("Dequeue: %d\n", value);
        return value;
        }
        printf("Dequeue: Queue is empty\n");
        return -1;
```

```
}
```

main.h

```
int main() {
	struct QueueUsingStacks* queue;
	initializeQueue(&queue);
	enqueue(queue, 1);
	enqueue(queue, 2);
	enqueue(queue, 3);
	dequeue(queue);
	dequeue(queue);
	dequeue(queue);
	enqueue(queue, 4);
	enqueue(queue, 5);
	dequeue(queue);
	dequeue(queue);

	return 0;
}
```

Output: