LAB SUBMISSION – 3

Meher Shrishti Nigam – 20BRS1193

1. Write a program to implement stack data structure which follows the property of stack. Implement the function push() and pop() (you may also write the function isEmpty() and isFull()).

Solution in C

```c
// 1_Stack.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct Stack {
    int top;
    unsigned max;
    int * array;
}Stack;

Stack * createStack(unsigned max)
{
    Stack * stack = (Stack *) malloc(sizeof(Stack));
    stack->max = max;
    stack->top = -1;
    stack->array = (int *) malloc(stack->max * sizeof(int));
    return stack;
}

bool isFull(Stack * stack)
{
    return stack->top == stack->max - 1;
}

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

void push (Stack * stack, int x)
{
    if(isFull(stack))
    {
        printf("Stack Overflow\n");
```

```
        return;
    }
    stack->array[++(stack->top)] = x;
    printf("%d pushed to stack\n", x);
}

int pop(Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    int item = stack->array[stack->top];
    stack->top = stack->top - 1;
    printf("%d was popped off the stack\n", item);
    return item;
}

int peek(Stack * stack)
{
    if(isEmpty(stack))
    {
        return INT_MIN;
    }
    return stack->array[stack->top];
}

// This function should not be available, but it is used here for illustrati
on purposes
void printStack(Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Stack is empty\n");
        return;
    }
    printf("Current stack: ");
    for(int i = 0; i < stack->top + 1; i++)
    {
        printf("%d ", stack->array[i]);
    }
    printf("\n");
}
```

```c
int main()
{
    Stack * stack_1 = createStack(6); // creating stack with max size 6
    // pushing numbers onto stack
    push(stack_1, 23);
    printStack(stack_1);
    push(stack_1, 56);
    printStack(stack_1);
    push(stack_1, 37);
    printStack(stack_1);
    push(stack_1, 92);
    printStack(stack_1);
    printf("The top element is: %d\n", peek(stack_1));
    push(stack_1, 102);
    printStack(stack_1);
    push(stack_1, 42);
    printStack(stack_1);
    printf("Top index is %d\n", stack_1->top);
    push(stack_1, 63); // stack overflow
    printf("Top index is %d\n", stack_1->top);
    printStack(stack_1);

    // popping numbers off the stack
    pop(stack_1);
    printStack(stack_1);
    pop(stack_1);
    printStack(stack_1);
    pop(stack_1);
    printStack(stack_1);
    pop(stack_1);
    printStack(stack_1);
    pop(stack_1);
    printStack(stack_1);
    pop(stack_1);
    printStack(stack_1);
    pop(stack_1);
    printStack(stack_1);
    pop(stack_1); // stack underflow
    printf("Top index is %d\n", stack_1->top);
    printStack(stack_1);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 1_Stack.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
23 pushed to stack
Current stack: 23
56 pushed to stack
Current stack: 23 56
37 pushed to stack
Current stack: 23 56 37
92 pushed to stack
Current stack: 23 56 37 92
The top element is: 92
102 pushed to stack
Current stack: 23 56 37 92 102
42 pushed to stack
Current stack: 23 56 37 92 102 42
Top index is 5
Stack Overflow
Top index is 5
Current stack: 23 56 37 92 102 42
42 was popped off the stack
Current stack: 23 56 37 92 102
102 was popped off the stack
Current stack: 23 56 37 92
92 was popped off the stack
Current stack: 23 56 37
37 was popped off the stack
Current stack: 23 56
56 was popped off the stack
Current stack: 23
23 was popped off the stack
Stack is empty
Stack Underflow
Top index is -1
Stack is empty
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

2. Write a program to implement queue data structure that follows FIFO property. Implement the function enqueue() and dequeue() for the same.
    a. Improve the above data structure considering the property of circular queue and write the function enqueue and dequeue accordingly.

```c
// 2_Queue.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct Queue {
    int front, rear;
    unsigned max;
    int* array;
}Queue;

struct Queue* createQueue(unsigned max)
{
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->max = max;
    queue->front = -1;
    queue->rear = -1;
    queue->array = (int*)malloc(queue->max * sizeof(int));
    printf("Queue of capacity %d created!\n", max);
    return queue;
}
bool isFull(Queue* queue)
{
    return (queue->rear == (queue->max) - 1);
}

bool isEmpty(Queue* queue)
{
    return (queue->front == -1 || queue->front > queue->rear);
}

void enqueue(Queue* queue, int item)
{
    if (isFull(queue))
    {
        printf("Queue Overflow\n");
        return;
    }
    if(queue->front == -1 && queue->rear == -1)
```

```
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else
    {
        queue->rear = queue->rear + 1;
    }
    queue->array[queue->rear] = item;
    printf("%d enqueued to queue\n", item);
}

int dequeue(Queue* queue)
{
    if (isEmpty(queue))
    {
        printf("Queue Underflow\n");
        return INT_MIN;
    }
    int item = queue->array[queue->front];
    queue->front = queue->front + 1;
    printf("%d has been dequeued\n", item);
    return item;
}
// This function should not be available, but it is used here for illustrati
on purposes
void printQueue(Queue * queue)
{
    if(isEmpty(queue))
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Current queue: ");
    for(int i = queue->front; i <= queue->rear; i++)
    {
        printf("%d ", queue->array[i]);
    }
    printf("\n");
}

int main()
{
    Queue * queue_1 = createQueue(6);
```

```
    enqueue(queue_1, 1);
    printQueue(queue_1);
    enqueue(queue_1, 2);
    printQueue(queue_1);
    enqueue(queue_1, 3);
    printQueue(queue_1);
    enqueue(queue_1, 4);
    printQueue(queue_1);
    dequeue(queue_1); // A dequeue
    printQueue(queue_1);
    enqueue(queue_1, 5);
    printQueue(queue_1);
    enqueue(queue_1, 6);
    printQueue(queue_1);
    enqueue(queue_1, 7);
    printQueue(queue_1);
    dequeue(queue_1);
    printQueue(queue_1);
    dequeue(queue_1);
    printQueue(queue_1);
    dequeue(queue_1);
    printQueue(queue_1);
    dequeue(queue_1);
    printQueue(queue_1);
    dequeue(queue_1);
    printQueue(queue_1);
    dequeue(queue_1);
    printQueue(queue_1);
    // Once the a full queue has been dequeued completely, it can't be used
again.
    enqueue(queue_1, 1);
    printQueue(queue_1);
    enqueue(queue_1, 2);
    printQueue(queue_1);
    enqueue(queue_1, 3);
    printQueue(queue_1);
    enqueue(queue_1, 4);
    printQueue(queue_1);
    enqueue(queue_1, 5);
    printQueue(queue_1);
    enqueue(queue_1, 6);
    printQueue(queue_1);
    enqueue(queue_1, 7);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 2_Queue.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Queue of capacity 6 created!
1 enqueued to queue
Current queue: 1
2 enqueued to queue
Current queue: 1 2
3 enqueued to queue
Current queue: 1 2 3
4 enqueued to queue
Current queue: 1 2 3 4
1 has been dequeued
Current queue: 2 3 4
5 enqueued to queue
Current queue: 2 3 4 5
6 enqueued to queue
Current queue: 2 3 4 5 6
Queue Overflow
Current queue: 2 3 4 5 6
2 has been dequeued
Current queue: 3 4 5 6
3 has been dequeued
Current queue: 4 5 6
4 has been dequeued
Current queue: 5 6
5 has been dequeued
Current queue: 6
6 has been dequeued
Queue is empty
Queue Underflow
Queue is empty
Queue Overflow
Queue is empty
Queue Overflow
Queue is empty
Queue Overflow
Queue is empty
Queue Overflow
Queue is empty
Queue Overflow
Queue is empty
Queue Overflow
Queue is empty
Queue Overflow
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

```c
// 2a_CircularQueue.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
```

```c
#include <limits.h>

typedef struct Queue {
    int front, rear;
    unsigned max;
    int* array;
}Queue;

Queue* createQueue(unsigned max)
{
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->max = max;
    queue->front = -1;
    queue->rear = -1;
    queue->array = (int*)malloc(queue->max * sizeof(int));
    printf("Queue of capacity %d created!\n", max);
    return queue;
}
bool isFull(Queue* queue)
{
    return ((queue->front == 0 && queue->rear == (queue->max) - 1)||(queue->front == queue->rear+1));
}

bool isEmpty(Queue* queue)
{
    return (queue->front == -1);
}

void enqueue(Queue* queue, int item)
{
    if (isFull(queue))
    {
        printf("Queue Overflow\n");
        return;
    }
    if(queue->front == -1 && queue->rear == -1)
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else if((queue->rear == (queue->max) - 1) && queue->front != 0)
    {
        queue->rear = 0;
```

```
    }
    else
    {
        queue->rear = queue->rear + 1;
    }
    queue->array[queue->rear] = item;
    printf("%d enqueued to queue\n", item);
}

int dequeue(Queue* queue)
{
    if (isEmpty(queue))
    {
        printf("Queue Underflow\n");
        return INT_MIN;
    }
    int item = queue->array[queue->front];
    if(queue->front == queue->rear)
    {
        queue->front = -1;
        queue->rear = -1;
    }
    else if(queue->front == (queue->max - 1))
    {
        queue->front = 0;
    }
    else
    {
        queue->front = queue->front + 1;
    }
    printf("%d has been dequeued\n", item);
    return item;
}

// This function should not be available, but it is used here for illustrati
on purposes
void printCQ(Queue * queue)
{
    if(isEmpty(queue))
    {
        printf("Queue is empty\n");
        return;
    }
```

```
    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->max; i++)
        {
            printf("%d ", queue->array[i]);
        }
        for(int i = 0; i <= queue->rear; i++)
        {
            printf("%d ", queue->array[i]);
        }
    }
    printf("\n");
}

int main()
{
    Queue * queue_1 = createQueue(6);
    enqueue(queue_1, 1);
    printCQ(queue_1);
    enqueue(queue_1, 2);
    printCQ(queue_1);
    enqueue(queue_1, 3);
    printCQ(queue_1);
    enqueue(queue_1, 4);
    printCQ(queue_1);
    enqueue(queue_1, 5);
    printCQ(queue_1);
    enqueue(queue_1, 6);
    printCQ(queue_1);
    enqueue(queue_1, 7);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
```

```
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);

    enqueue(queue_1, 1);
    printCQ(queue_1);
    enqueue(queue_1, 2);
    printCQ(queue_1);
    enqueue(queue_1, 3);
    printCQ(queue_1);
    enqueue(queue_1, 4);
    printCQ(queue_1);
    enqueue(queue_1, 5);
    printCQ(queue_1);
    enqueue(queue_1, 6);
    printCQ(queue_1);
    enqueue(queue_1, 7);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    // Thus, a circular queue can be used as many times as required.
    enqueue(queue_1, 1);
    printCQ(queue_1);
    enqueue(queue_1, 2);
    printCQ(queue_1);
    enqueue(queue_1, 3);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    enqueue(queue_1, 4);
```

```
    printCQ(queue_1);
    enqueue(queue_1, 5);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    enqueue(queue_1, 6);
    printCQ(queue_1);
    enqueue(queue_1, 7);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
    printCQ(queue_1);
    dequeue(queue_1);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 2a_CircularQueue.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Queue of capacity 6 created!
1 enqueued to queue
1
2 enqueued to queue
1 2
3 enqueued to queue
1 2 3
4 enqueued to queue
1 2 3 4
5 enqueued to queue
1 2 3 4 5
6 enqueued to queue
1 2 3 4 5 6
Queue Overflow
1 2 3 4 5 6
1 has been dequeued
2 3 4 5 6
2 has been dequeued
3 4 5 6
3 has been dequeued
4 5 6
4 has been dequeued
5 6
5 has been dequeued
6
6 has been dequeued
Queue is empty
Queue Underflow
```

```
Queue Underflow
1 enqueued to queue
1
2 enqueued to queue
1 2
3 enqueued to queue
1 2 3
4 enqueued to queue
1 2 3 4
5 enqueued to queue
1 2 3 4 5
6 enqueued to queue
1 2 3 4 5 6
Queue Overflow
1 2 3 4 5 6
1 has been dequeued
2 3 4 5 6
2 has been dequeued
3 4 5 6
3 has been dequeued
4 5 6
4 has been dequeued
5 6
5 has been dequeued
6
6 has been dequeued
Queue is empty
Queue Underflow
1 enqueued to queue
1
2 enqueued to queue
1 2
3 enqueued to queue
1 2 3
1 has been dequeued
2 3
4 enqueued to queue
2 3 4
5 enqueued to queue
2 3 4 5
```

```
2 3 4 5
2 has been dequeued
3 4 5
6 enqueued to queue
3 4 5 6
7 enqueued to queue
3 4 5 6 7
3 has been dequeued
4 5 6 7
4 has been dequeued
5 6 7
5 has been dequeued
6 7
6 has been dequeued
7
7 has been dequeued
PS C:\Users\meher\OneDr
```

3. Implement Stack data structure using Queue. (you need to use 2 Queue in this case).
    a. Write the program by making push operation costly.

```c
// 3a_StackUsingQueuePushCostly.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct Queue {
    int front, rear;
    unsigned max;
    int* array;
}Queue;

typedef struct Stack {
    Queue * queue_1;
    Queue * queue_2;
}Stack;

Stack * createStack(unsigned max)
{
    Stack * stack = (Stack*)malloc(sizeof(Stack));

    stack->queue_1 = (Queue*)malloc(sizeof(Queue));
    stack->queue_1->max = max;
    stack->queue_1->front = -1;
    stack->queue_1->rear = -1;
    stack->queue_1->array = (int*)malloc(stack->queue_2->max * sizeof(int));

    stack->queue_2 = (Queue*)malloc(sizeof(Queue));
    stack->queue_2->max = max;
    stack->queue_2->front = -1;
    stack->queue_2->rear = -1;
    stack->queue_2->array = (int*)malloc(stack->queue_2->max * sizeof(int));

    printf("Stack of capacity %d has been created!\n", max);
    return stack;
}
bool isFull(Queue* queue)
{
    return ((queue->front == 0 && queue->rear == (queue->max) - 1)||(queue-
>front == queue->rear+1));
}
```

```
bool isEmpty(Queue* queue)
{
    return (queue->front == -1);
}

void enqueue(Queue* queue, int item)
{
    if(isFull(queue))
    {
        // printf("Queue Overflow\n");
        return;
    }
    if(queue->front == -1 && queue->rear == -1)
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else if((queue->rear == (queue->max) - 1) && queue->front != 0)
    {
        queue->rear = 0;
    }
    else
    {
        queue->rear = queue->rear + 1;
    }
    queue->array[queue->rear] = item;

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

int dequeue(Queue* queue)
{
    if (isEmpty(queue))
    {
        //printf("Queue Underflow\n");
        return INT_MIN;
    }
    int item = queue->array[queue->front];
    if(queue->front == queue->rear)
    {
        queue->front = -1;
        queue->rear = -1;
    }
```

```
    else if(queue->front == (queue->max - 1))
    {
        queue->front = 0;
    }
    else
    {
        queue->front = queue->front + 1;
    }

    //printf("%d has been dequeued\n", item);
    return item;
}

void push(Stack * stack, int item)
{
    if(isFull(stack->queue_1))
    {
        printf("Stack Overflow\n");
        return;
    }
    while(!isEmpty(stack->queue_1))
    {
        enqueue(stack->queue_2, dequeue(stack->queue_1));
    }
    enqueue(stack->queue_1, item);
    while(!isEmpty(stack->queue_2))
    {
        enqueue(stack->queue_1, dequeue(stack->queue_2));
    }
    printf("%d has been pushed to the stack!\n", item);
}

int pop(Stack * stack)
{
    if(isEmpty(stack->queue_1))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    int item = dequeue(stack->queue_1);
    printf("%d was popped off the stack!\n", item);
    return item;
}
```

```c
// This function should not be available, but it is used here for illustrati
on purposes
void printStack(Queue * queue)
{
    if(isEmpty(queue))
    {
        printf("Stack is empty\n");
        return;
    }
    printf("Current stack: ");
    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->max; i++)
        {
            printf("%d ", queue->array[i]);
        }
        for(int i = 0; i <= queue->rear; i++)
        {
            printf("%d ", queue->array[i]);
        }
    }
    printf("\n");
}

int main()
{
    Stack * stack_1 = createStack(6);
    push(stack_1, 23);
    printStack(stack_1->queue_1);
    push(stack_1, 56);
    printStack(stack_1->queue_1);
    push(stack_1, 37);
    printStack(stack_1->queue_1);
    push(stack_1, 92);
    printStack(stack_1->queue_1);
    push(stack_1, 10);
    printStack(stack_1->queue_1);
    push(stack_1, 41);
```

```
    printStack(stack_1->queue_1);
    push(stack_1, 41);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 3a_StackUsingQueuePushCostly.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Stack of capacity 6 has been created!
23 has been pushed to the stack!
Current stack: 23
56 has been pushed to the stack!
Current stack: 56 23
37 has been pushed to the stack!
Current stack: 37 56 23
92 has been pushed to the stack!
Current stack: 92 37 56 23
10 has been pushed to the stack!
Current stack: 10 92 37 56 23
41 has been pushed to the stack!
Current stack: 41 10 92 37 56 23
Stack Overflow
41 was popped off the stack!
Current stack: 10 92 37 56 23
10 was popped off the stack!
Current stack: 92 37 56 23
92 was popped off the stack!
Current stack: 37 56 23
37 was popped off the stack!
Current stack: 56 23
56 was popped off the stack!
Current stack: 23
23 was popped off the stack!
Stack is empty
Stack Underflow
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

b. Write the program by making pop operation costly

```c
// 3b_StackUsingQueuePopCostly.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct Queue {
    int front, rear;
    unsigned max;
    unsigned size;
    int* array;
}Queue;

typedef struct Stack {
    Queue * queue_1;
    Queue * queue_2;
}Stack;

Stack * createStack(unsigned max)
{
    Stack * stack = (Stack*)malloc(sizeof(Stack));
    stack->queue_1 = (Queue*)malloc(sizeof(Queue));
    stack->queue_1->max = max;
    stack->queue_1->front = -1;
    stack->queue_1->rear = -1;
    stack->queue_1->array = (int*)malloc(stack->queue_2->max * sizeof(int));
    stack->queue_1->size = 0;

    stack->queue_2 = (Queue*)malloc(sizeof(Queue));
    stack->queue_2->max = max;
    stack->queue_2->front = -1;
    stack->queue_2->rear = -1;
    stack->queue_2->array = (int*)malloc(stack->queue_2->max * sizeof(int));
    stack->queue_1->size = 0;

    printf("Stack of capacity %d has been created!\n", max);
    return stack;
}
bool isFull(Queue* queue)
{
    return ((queue->front == 0 && queue->rear == (queue->max) - 1)||(queue-
>front == queue->rear+1));
}
```

```
bool isEmpty(Queue* queue)
{
    return (queue->front == -1);
}

void enqueue(Queue* queue, int item)
{
    if (isFull(queue))
    {
        printf("Queue Overflow\n");
        return;
    }
    if(queue->front == -1 && queue->rear == -1)
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else if((queue->rear == (queue->max) - 1) && queue->front != 0)
    {
        queue->rear = 0;
    }
    else
    {
        queue->rear = queue->rear + 1;
    }
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    return;
    //printf("%d enqueued to queue\n", item);
}

int dequeue(Queue* queue)
{
    if (isEmpty(queue))
    {
        printf("Queue Underflow\n");
        return INT_MIN;
    }
    int item = queue->array[queue->front];
    if(queue->front == queue->rear)
    {
        queue->front = -1;
        queue->rear = -1;
    }
```

```
    else if(queue->front == (queue->max - 1))
    {
        queue->front = 0;
    }
    else
    {
        queue->front = queue->front + 1;
    }
    queue->size = queue->size - 1;
    //printf("%d has been dequeued\n", item);
    return item;
}

void push(Stack * stack, int item)
{
    if(isFull(stack->queue_1))
    {
        printf("Stack Overflow\n");
        return;
    }
    enqueue(stack->queue_1, item);
    printf("%d has been pushed to the stack!\n", item);
}

int pop(Stack * stack)
{
    if(isEmpty(stack->queue_1))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    while(stack->queue_1->size != 1)
    {
        enqueue(stack->queue_2, dequeue(stack->queue_1));
    }
    int item = dequeue(stack->queue_1);
    while(!isEmpty(stack->queue_2))
    {
        enqueue(stack->queue_1, dequeue(stack->queue_2));
    }
    printf("%d was popped off the stack!\n", item);
    return item;
}
```

```c
// This function should not be available, but it is used here for illustrati
on purposes
void printStack(Queue * queue)
{
    if(isEmpty(queue))
    {
        printf("Stack is empty\n");
        return;
    }
    printf("Current stack: ");
    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->max; i++)
        {
            printf("%d ", queue->array[i]);
        }
        for(int i = 0; i <= queue->rear; i++)
        {
            printf("%d ", queue->array[i]);
        }
    }
    printf("\n");
}

int main()
{
    Stack * stack_1 = createStack(6);
    push(stack_1, 23);
    printStack(stack_1->queue_1);
    push(stack_1, 56);
    printStack(stack_1->queue_1);
    push(stack_1, 37);
    printStack(stack_1->queue_1);
    push(stack_1, 92);
    printStack(stack_1->queue_1);
    push(stack_1, 10);
    printStack(stack_1->queue_1);
    push(stack_1, 41);
```

```
    printStack(stack_1->queue_1);
    push(stack_1, 41);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
    printStack(stack_1->queue_1);
    pop(stack_1);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 3b_StackUsingQueuePopCostly.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Stack of capacity 6 has been created!
23 has been pushed to the stack!
Current stack: 23
56 has been pushed to the stack!
Current stack: 23 56
37 has been pushed to the stack!
Current stack: 23 56 37
92 has been pushed to the stack!
Current stack: 23 56 37 92
10 has been pushed to the stack!
Current stack: 23 56 37 92 10
41 has been pushed to the stack!
Current stack: 23 56 37 92 10 41
Stack Overflow
41 was popped off the stack!
Current stack: 23 56 37 92 10
10 was popped off the stack!
Current stack: 23 56 37 92
92 was popped off the stack!
Current stack: 23 56 37
37 was popped off the stack!
Current stack: 23 56
56 was popped off the stack!
Current stack: 23
23 was popped off the stack!
Stack is empty
Stack Underflow
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

4. Implement Queue data structure using stack. (you need to use 2 stack in this case)
    a. Write the program by making enqueue operation costly

```c
// 4a_QueueUsingStackEnqueueCostly.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct Stack {
    int top;
    unsigned max;
    int * array;
}Stack;

typedef struct Queue {
    Stack * stack_1;
    Stack * stack_2;
}Queue;

Queue * createQueue(unsigned max)
{
    Queue * queue = (Queue*)malloc(sizeof(Queue));
    queue->stack_1 = (Stack *) malloc(sizeof(Stack));
    queue->stack_1->max = max;
    queue->stack_1->top = -1;
    queue->stack_1->array = (int *) malloc(queue->stack_1-
>max * sizeof(int));

    queue->stack_2 = (Stack *) malloc(sizeof(Stack));
    queue->stack_2->max = max;
    queue->stack_2->top = -1;
    queue->stack_2->array = (int *) malloc(queue->stack_2-
>max * sizeof(int));

    printf("Queue of capacity %d has been created!\n", max);
    return queue;
}
bool isFull(Stack * stack)
{
    return stack->top == stack->max - 1;
}

bool isEmpty(Stack * stack)
{
    return stack->top == -1;
}
```

```c
void push (Stack * stack, int x)
{
    if(isFull(stack))
    {
        printf("Stack Overflow\n");
        return;
    }
    stack->array[++(stack->top)] = x;
    //printf("%d pushed to stack\n", x);
}

int pop (Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    return stack->array[stack->top--];
}

void enqueue(Queue * queue, int item)
{
    if(isFull(queue->stack_1))
    {
        printf("Queue Overflow\n");
        return;
    }
    if(isEmpty(queue->stack_1))
    {
        push(queue->stack_1, item);
        printf("%d was queued!\n", item);
        return;
    }
    while(!isEmpty(queue->stack_1))
    {
        push(queue->stack_2, pop(queue->stack_1));
    }
    push( queue->stack_1, item);
    while(!isEmpty(queue->stack_2))
    {
        push(queue->stack_1, pop(queue->stack_2));
    }
    printf("%d was queued!\n", item);
```

```
}

int dequeue(Queue * queue)
{
    if(isEmpty(queue->stack_1))
    {
        printf("Queue Underflow");
        return INT_MIN;
    }
    int item = pop(queue->stack_1);
    printf("%d was dequeued!\n", item);
    return item;
}
// This function should not be available, but it is used here for illustrati
on purposes
void printQueue(Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Current queue: ");
    for(int i = 0; i < stack->top + 1; i++)
    {
        printf("%d ", stack->array[i]);
    }
    printf("\n");
}

int main()
{
    Queue * queue_1 = createQueue(6);
    enqueue(queue_1, 1);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 2);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 3);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 4);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 5);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 6);
```

```c
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 7);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 4a_QueueUsingStackEnqueueCostly.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Queue of capacity 6 has been created!
1 was queued!
Current queue: 1
2 was queued!
Current queue: 2 1
3 was queued!
Current queue: 3 2 1
4 was queued!
Current queue: 4 3 2 1
5 was queued!
Current queue: 5 4 3 2 1
6 was queued!
Current queue: 6 5 4 3 2 1
Queue Overflow
Current queue: 6 5 4 3 2 1
1 was dequeued!
Current queue: 6 5 4 3 2
2 was dequeued!
Current queue: 6 5 4 3
3 was dequeued!
Current queue: 6 5 4
4 was dequeued!
Current queue: 6 5
5 was dequeued!
Current queue: 6
6 was dequeued!
Queue is empty
Queue Underflow
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

b. Write the program by making dequeue operation costly

```c
// 4b_QueueUsingStackDequeueCostly.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct Stack {
    int top;
    unsigned max;
    int * array;
}Stack;

typedef struct Queue {
    Stack * stack_1;
    Stack * stack_2;
}Queue;

Queue * createQueue(unsigned max)
{
    Queue * queue = (Queue*)malloc(sizeof(Queue));
    queue->stack_1 = (Stack *) malloc(sizeof(Stack));
    queue->stack_1->max = max;
    queue->stack_1->top = -1;
    queue->stack_1->array = (int *) malloc(queue->stack_1-
>max * sizeof(int));

    queue->stack_2 = (Stack *) malloc(sizeof(Stack));
    queue->stack_2->max = max;
    queue->stack_2->top = -1;
    queue->stack_2->array = (int *) malloc(queue->stack_2-
>max * sizeof(int));

    printf("Queue of capacity %d has been created!\n", max);
    return queue;
}

bool isFull(Stack * stack)
{
    return stack->top == stack->max - 1;
}

bool isEmpty(Stack * stack)
{
    return stack->top == -1;
```

```
}


void push (Stack * stack, int x)
{
    if(isFull(stack))
    {
        printf("Stack Overflow\n");
        return;
    }
    stack->array[++(stack->top)] = x;
    //printf("%d pushed to stack\n", x);
}

int pop (Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    return stack->array[stack->top--];
}


void enqueue(Queue * queue, int item)
{
    if(isFull(queue->stack_1))
    {
        printf("Queue Overflow\n");
        return;
    }
    push(queue->stack_1, item);
    printf("%d was queued!\n", item);
}


int dequeue(Queue * queue)
{
    if(isEmpty(queue->stack_1))
    {
        printf("Queue Underflow\n");
        return INT_MIN;
    }
```

```
    while(!isEmpty(queue->stack_1))
    {
        push(queue->stack_2, pop(queue->stack_1));
    }
    int item = pop(queue->stack_2);
    while(!isEmpty(queue->stack_2))
    {
        push(queue->stack_1, pop(queue->stack_2));
    }
    printf("%d was dequeued!\n", item);
    return item;
}

// This function should not be available, but it is used here for illustrati
on purposes
void printQueue(Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Current queue: ");
    for(int i = 0; i < stack->top + 1; i++)
    {
        printf("%d ", stack->array[i]);
    }
    printf("\n");
}

int main()
{
    Queue * queue_1 = createQueue(6);
    enqueue(queue_1, 1);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 2);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 3);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 4);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 5);
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 6);
```

```c
    printQueue(queue_1->stack_1);
    enqueue(queue_1, 7);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
    printQueue(queue_1->stack_1);
    dequeue(queue_1);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 4b_QueueUsingStackDequeueCostly.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Queue of capacity 6 has been created!
1 was queued!
Current queue: 1
2 was queued!
Current queue: 1 2
3 was queued!
Current queue: 1 2 3
4 was queued!
Current queue: 1 2 3 4
5 was queued!
Current queue: 1 2 3 4 5
6 was queued!
Current queue: 1 2 3 4 5 6
Queue Overflow
Current queue: 1 2 3 4 5 6
1 was dequeued!
Current queue: 2 3 4 5 6
2 was dequeued!
Current queue: 3 4 5 6
3 was dequeued!
Current queue: 4 5 6
4 was dequeued!
Current queue: 5 6
5 was dequeued!
Current queue: 6
6 was dequeued!
Queue is empty
Queue Underflow
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

5. Consider the following string

Reverse

Write a program (Use stack) to reverse the given string

```c
// 5_ReverseUsingStack.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

typedef struct Stack {
    int top;
    unsigned max;
    char * array;
}Stack;

Stack * createStack(unsigned max)
{
    Stack * stack = (Stack *) malloc(sizeof(Stack));
    stack->max = max;
    stack->top = -1;
    stack->array = (char *) malloc(stack->max * sizeof(char));
    return stack;
}

bool isFull(Stack * stack)
{
    return stack->top == stack->max - 1;
}

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


void push (Stack * stack, char x)
{
    if(isFull(stack))
    {
        printf("Stack Overflow\n");
        return;
    }
```

```
    stack->array[++(stack->top)] = x;
    // printf("%c pushed to stack\n", x);
}

int pop (Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    return stack->array[stack->top--];
}

int main()
{
    Stack * stack_1 = createStack(10);
    printf("Enter a string of length less than 10 that you want to reverse:\
n");
    char str[11];
    scanf("%s", str);
    for(int i = 0; i < strlen(str); i++)
    {
        push(stack_1, str[i]);
    }
    for(int i = 0; i < strlen(str); i++)
    {
        char x = pop(stack_1);
        printf("%c", x);
    }
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 5_ReverseUsingStack.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Enter a string of length less than 10 that you want to reverse:
REVERSE
ESREVER
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Enter a string of length less than 10 that you want to reverse:
PINEAPPLE
ELPPAENIP
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

6. Consider the following expression. Write a program to evaluate whether the given expression is balanced or not. (Use stack data structure to evaluate the expression)

{[{}{}]}[()]
[{}()]{})

```c
// 6_ExpressionBalancing.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

typedef struct Stack {
    int top;
    unsigned max;
    char * array;
}Stack;

Stack * createStack(unsigned max)
{
    Stack * stack = (Stack *) malloc(sizeof(Stack));
    stack->max = max;
    stack->top = -1;
    stack->array = (char *) malloc(stack->max * sizeof(char));
    return stack;
}

bool isFull(Stack * stack)
{
    return stack->top == stack->max - 1;
}

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

void push (Stack * stack, char x)
{
    if(isFull(stack))
    {
        //printf("Stack Overflow\n");
        return;
    }
    stack->array[++(stack->top)] = x;
```

```
    // printf("%c pushed to stack at %d\n", x, stack->top);
}

int pop (Stack * stack)
{
    if(isEmpty(stack))
    {
        //printf("Stack Underflow\n");
        return INT_MIN;
    }
    return stack->array[stack->top--];
}

bool match(char character1, char character2)
{
    if (character1 == '(' && character2 == ')')
        return true;
    else if (character1 == '{' && character2 == '}')
        return true;
    else if (character1 == '[' && character2 == ']')
        return true;
    return false;
}


int main()
{
    Stack * stack_1 = createStack(50);
    printf("Enter the expression:\n");
    char str[50];
    scanf("%s", str);
    for(int i = 0; i < strlen(str); i++)
    {
        if(str[i] == '(' || str[i] == '{' || str[i] == '[')
        {
            push(stack_1, str[i]);
        }
        else
        {
            char x = pop(stack_1);
            if(!match(x,str[i]))
            {
                printf("Not balanced.\n");
                return 0;
```

```
            }
        }
    }
    if(isEmpty(stack_1))
    {
        printf("Balanced.\n");
    }
    else
    {
        printf("Not balanced.\n");
    }
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc  6_ExpressionBalancing.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Enter the expression:
{[{}{}]}[()]
Balanced.
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc  6_ExpressionBalancing.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Enter the expression:
[{}()]{})
Not balanced.
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

7. Consider the following expression

   4+5*3-2

   Write a program to evaluate the above expression using stack data structure

```
// 7_InfixToPostfixOrPrefix.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
typedef struct Stack {
    int top;
    unsigned max;
    int * array;
}Stack;

Stack * createStack(unsigned max)
{
    Stack * stack = (Stack *) malloc(sizeof(Stack));
    stack->max = max;
    stack->top = -1;
```

```
    stack->array = (int *) malloc(stack->max * sizeof(int));
    return stack;
}

bool isFull(Stack * stack)
{
    return stack->top == stack->max - 1;
}

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

void push (Stack * stack, int x)
{
    if(isFull(stack))
    {
        printf("Stack Overflow\n");
        return;
    }
    stack->array[++(stack->top)] = x;
    //printf("%d pushed to stack\n", x);
}

int pop (Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    return stack->array[stack->top--];
}

int peek(Stack * stack)
{
    if(isEmpty(stack))
    {
        return INT_MIN;
    }
    return stack->array[stack->top];
}
```

```
int Postfix(char str[])
{
    int len = strlen(str);
    Stack * stack_1 = createStack(len + 1);
    for(int i = 0; i < len; i++)
    {
        if(isdigit(str[i]))
        {
            push(stack_1, (int)str[i] - (int)'0');
        }
        else
        {
            int x = pop(stack_1);
            int y = pop(stack_1);
            if(str[i] == '+')
                push(stack_1, y + x);
            else if(str[i] == '-')
                push(stack_1, y - x);
            else if(str[i] == '*')
                push(stack_1, y * x);
            else if(str[i] == '/')
                push(stack_1, y / x);
        }
    }
    return pop(stack_1);
}

int Prefix(char str[])
{
    int len = strlen(str);
    Stack * stack_1 = createStack(len + 1);
    for(int i = len-1; i >= 0; i--)
    {
        if(isdigit(str[i]))
        {
            push(stack_1, (int)str[i] - (int)'0');
        }
        else
        {
            int x = pop(stack_1);
            int y = pop(stack_1);
            if(str[i] == '+')
                push(stack_1, x + y);
            else if(str[i] == '-')
```

```
                push(stack_1, x - y);
            else if(str[i] == '*')
                push(stack_1, x * y);
            else if(str[i] == '/')
                push(stack_1, x / y);
        }
    }
    return pop(stack_1);
}

bool Precedence(char a, char b)
{
    if((a == '*' || a == '/') && (b == '+' || b == '-
')) // a has greater precedence
    {
        return true;
    }
    // both have same precedence or b has greater precedence
    return false;
}

int InfixViaPostfix(char str[])
{
    int len = strlen(str);
    char str2[30];
    Stack * stack_1 = createStack(30);
    int c = 0;
    for(int i = 0; i < len; i++)
    {
        if(isdigit(str[i]))
        {
            str2[c] = str[i];
            c++;
        }
        else
        {
            if(isEmpty(stack_1))
            {
                push(stack_1, str[i]);
            }
            else
            {
                if(Precedence(str[i], peek(stack_1)))
                {
```

```
                    push(stack_1, str[i]);
                }
                else
                {
                    while(!isEmpty(stack_1))
                    {
                        str2[c] = pop(stack_1);
                        c++;
                    }
                    push(stack_1, str[i]);
                }
            }
        }
    }
    while(!isEmpty(stack_1))
    {
        str2[c] = pop(stack_1);
        c++;
    }
    //printf("%s\n", str2);
    return Postfix(str2);
}

int InfixViaPrefix(char str[])
{
    int len = strlen(str);
    char str1[30];
    int c1 = 0;
    for(int i = len - 1; i >= 0; i--)
    {
        str1[c1] = str[i];
        c1++;
    }
    char str2[30];
    Stack * stack_1 = createStack(30);
    int c = 0;
    for(int i = 0; i < len; i++)
    {
        if(isdigit(str1[i]))
        {
            str2[c] = str1[i];
            c++;
        }
        else
```

```
        {
            if(isEmpty(stack_1))
            {
                push(stack_1, str1[i]);
            }
            else
            {
                if(Precedence(str1[i], peek(stack_1)))
                {
                    push(stack_1, str1[i]);
                }
                else
                {
                    while(!isEmpty(stack_1))
                    {
                        str2[c] = pop(stack_1);
                        c++;
                    }
                    push(stack_1, str1[i]);
                }
            }
        }
    }
    while(!isEmpty(stack_1))
    {
        str2[c] = pop(stack_1);
        c++;
    }
    c1 = 0;
    for(int i = len - 1; i >= 0; i--)
    {
        str1[c1] = str2[i];
        c1++;
    }
    //printf("%s\n", str1);
    return Prefix(str1);
}

int main()
{
    char str[30] = "4+5*3-2";
    printf("Evaluation of infix expression using Postfix evaluation %s is %d
\n", str, InfixViaPostfix(str)); // 453*+2-
```

```
    printf("Evaluation of infix expression using Prefix evaluation %s is %d\
n", str, InfixViaPrefix(str)); //+4-*532
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 7_InfixToPostfixOrPrefix.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Evaluation of infix expression using Postfix evaluation 4+5*3-2 is 17
Evaluation of infix expression using Prefix evaluation 4+5*3-2 is 17
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

8. Write a program to evaluate the postfix expression using stack data structure
   138*+ will be evaluated to 25
   545*+5/ will be evaluated to 5 (consider every digit as one single unit)

```
// 8_PostfixStack.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
typedef struct Stack {
    int top;
    unsigned max;
    int * array;
}Stack;

Stack * createStack(unsigned max)
{
    Stack * stack = (Stack *) malloc(sizeof(Stack));
    stack->max = max;
    stack->top = -1;
    stack->array = (int *) malloc(stack->max * sizeof(int));
    return stack;
}

bool isFull(Stack * stack)
{
    return stack->top == stack->max - 1;
}

bool isEmpty(Stack * stack)
{
    return stack->top == -1;
}
```

```c
void push (Stack * stack, int x)
{
    if(isFull(stack))
    {
        printf("Stack Overflow\n");
        return;
    }
    stack->array[++(stack->top)] = x;
    //printf("%d pushed to stack\n", x);
}

int pop (Stack * stack)
{
    if(isEmpty(stack))
    {
        printf("Stack Underflow\n");
        return INT_MIN;
    }
    return stack->array[stack->top--];
}

int Postfix(char str[])
{
    int len = strlen(str);
    Stack * stack_1 = createStack(len + 1);
    for(int i = 0; i < len; i++)
    {
        if(isdigit(str[i]))
        {
            push(stack_1, (int)str[i] - (int)'0');
        }
        else
        {
            int x = pop(stack_1);
            int y = pop(stack_1);
            if(str[i] == '+')
                push(stack_1, y + x);
            else if(str[i] == '-')
                push(stack_1, y - x);
            else if(str[i] == '*')
                push(stack_1, y * x);
            else if(str[i] == '/')
                push(stack_1, y / x);
        }
```

```
    }
    return pop(stack_1);
}

int main()
{
    char str[30] = "138*+";
    printf("Postfix evaluation of %s is %d\n", str, Postfix(str));
    char str2[30] = "545*+5/";
    printf("Postfix evaluation of %s is %d\n", str2, Postfix(str2));
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 8_PostfixStack.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Postfix evaluation of 138*+ is 25
Postfix evaluation of 545*+5/ is 5
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

(Prefix evaluator in the github repository)

9. Consider a scenario where one line is to submit one form. Once the line started moving the authority realized announced that few students from final year should be into another line, resulting creation of another line. The formation of other line is created as: Once the authority gets that the student belong to final year they push them to other line. The other line starts from the last student that has join the line. At the end both lines will be displayed on the screen. Write a program to demonstrate the above scenario.
   Input: 1,2,3,2,3,4,1,2,4,3
   Output: 1,2,3,2,3,1,2,3
   4,4
   Input: 1,2,3,4

```
// 9_FinalYear.c
/*
*    Consider a scenario where one line is to submit one form. Once the line
 started
*    moving the authority realized announced that few students from final ye
ar should be
*    into another line, resulting creation of another line. The formation of
 other line is
*    created as: Once the authority gets that the student belong to final ye
ar they push them
*    to other line. The other line starts from the last student that has joi
n the line. At the
*    end both lines will be displayed on the screen. Write a program to demo
nstrate the
```

```
*    above scenario.
*    Input: 1,2,3,2,3,4,1,2,4,3
*    Output: 1,2,3,2,3,1,2,3
*    4,4
*    Input: 1,2,3,4
*/

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

typedef struct Queue {
    int front, rear;
    unsigned max;
    int* array;
}Queue;

struct Queue* createQueue(unsigned max)
{
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->max = max;
    queue->front = -1;
    queue->rear = -1;
    queue->array = (int*)malloc(queue->max * sizeof(int));
    // printf("Queue of capacity %d created!\n", max);
    return queue;
}
bool isFull(Queue* queue)
{
    return (queue->rear == (queue->max) - 1);
}

bool isEmpty(Queue* queue)
{
    return (queue->front == -1 || queue->front > queue->rear);
}

void enqueue(Queue* queue, int item)
{
    if (isFull(queue))
    {
        printf("Queue Overflow\n");
        return;
```

```
    }
    if(queue->front == -1 && queue->rear == -1)
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else
    {
        queue->rear = queue->rear + 1;
    }
    queue->array[queue->rear] = item;
    //printf("%d enqueued to queue\n", item);
}

int dequeue(Queue* queue)
{
    if (isEmpty(queue))
    {
        printf("Queue Underflow\n");
        return INT_MIN;
    }
    int item = queue->array[queue->front];
    queue->front = queue->front + 1;
    //printf("%d has been dequeued\n", item);
    return item;
}

int main()
{
    Queue * queue = createQueue(20);
    printf("Enter the number of students: \n");
    int n = 0;
    scanf("%d", &n);
    printf("Enter the year of %d students: \n", n);
    for(int i = 0; i < n; i++)
    {
        int x = 0;
        scanf("%d", &x);
        enqueue(queue, x);
    }
    Queue * queue_1 = createQueue(20);
    Queue * queue_2 = createQueue(20);
    for(int i = 0; i < n; i++)
    {
```

```
        int x = dequeue(queue);
        if(x == 4)
            enqueue(queue_2, x);
        else
            enqueue(queue_1, x);
    }
    printf("Non-final-year students' queue: ");
    while(!isEmpty(queue_1))
    {
        printf("%d ", dequeue(queue_1));
    }
    printf("\nFinal-year students' queue: ");
    while(!isEmpty(queue_2))
    {
        printf("%d ", dequeue(queue_2));
    }
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 9_FinalYear.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Enter the number of students:
10
Enter the year of 10 students:
1
2
3
2
3
4
1
2
4
3
Non-final-year students' queue: 1 2 3 2 3 1 2 3
Final-year students' queue: 4 4
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 9_FinalYear.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Enter the number of students:
4
Enter the year of 4 students:
1
2
3
4
Non-final-year students' queue: 1 2 3
Final-year students' queue: 4
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```

10. Consider the queue which is able to perform insertion and deletion from both the end.

Doubly Ended Queue

```c
// 10_DoublyEndedQueue.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct DQueue {
    int front, rear;
    unsigned max;
    int* array;
}DQueue;

DQueue* createDQueue(unsigned max)
{
    DQueue* queue = (DQueue*)malloc(sizeof(DQueue));
    queue->max = max;
    queue->front = -1;
    queue->rear = -1;
    queue->array = (int*)malloc(queue->max * sizeof(int));
    printf("Queue of capacity %d created!\n", max);
    return queue;
}
bool isFull(DQueue* queue)
{
    return ((queue->front == 0 && queue->rear == (queue->max) - 1)||(queue-
>front == queue->rear + 1));
}

bool isEmpty(DQueue* queue)
{
    return ((queue->front == -1) && (queue->rear == -1));
}

void push_back(DQueue* queue, int item)
{
    if (isFull(queue))
    {
        printf("Queue Overflow\n");
        return;
    }
    if(queue->front == -1 && queue->rear == -1)
```

```
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else if((queue->rear == (queue->max) - 1) && queue->front != 0)
    {
        queue->rear = 0;
    }
    else
    {
        queue->rear = queue->rear + 1;
    }
    queue->array[queue->rear] = item;
    printf("%d was pushed back to queue\n", item);
}

void push_front(DQueue* queue, int item)
{
    if (isFull(queue))
    {
        printf("Queue Overflow\n");
        return;
    }
    if(queue->front == -1 && queue->rear == -1)
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else if(queue->front == 0)
    {
        queue->front = queue->max - 1;
    }
    else
    {
        queue->front = queue->front - 1;
    }
    queue->array[queue->front] = item;
    printf("%d was pushed in front of queue\n", item);
}

int pop_front(DQueue* queue)
{
    if (isEmpty(queue))
    {
```

```
        printf("Queue Underflow\n");
        return INT_MIN;
    }
    int item = queue->array[queue->front];
    if(queue->front == queue->rear)
    {
        queue->front = -1;
        queue->rear = -1;
    }
    else if(queue->front == (queue->max - 1))
    {
        queue->front = 0;
    }
    else
    {
        queue->front = queue->front + 1;
    }
    printf("%d was popped from front\n", item);
    return item;
}

int pop_rear(DQueue* queue)
{
    if (isEmpty(queue))
    {
        printf("Queue Underflow\n");
        return INT_MIN;
    }
    int item = queue->array[queue->rear];
    if(queue->front == queue->rear)
    {
        queue->front = -1;
        queue->rear = -1;
    }
    else if(queue->rear == 0)
    {
        queue->rear = queue->max - 1;
    }
    else
    {
        queue->rear = queue->rear - 1;
    }
    printf("%d was popped from rear\n", item);
    return item;
```

```
}

// This function should not be available, but it is used here for illustrati
on purposes
void printDQ(DQueue * queue)
{
    if(isEmpty(queue))
    {
        printf("Dqueue is empty\n");
        return;
    }

    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->max; i++)
        {
            printf("%d ", queue->array[i]);
        }
        for(int i = 0; i <= queue->rear; i++)
        {
            printf("%d ", queue->array[i]);
        }
    }
    printf("\n");
}

int main()
{
    DQueue * queue_1 = createDQueue(10);
    push_front(queue_1, 10);
    printDQ(queue_1);
    push_back(queue_1, 20);
    printDQ(queue_1);
    push_front(queue_1, 30);
    printDQ(queue_1);
    push_back(queue_1, 40);
    printDQ(queue_1);
    push_front(queue_1, 50);
```

```
    printDQ(queue_1);
    push_back(queue_1, 60);
    printDQ(queue_1);
    push_front(queue_1, 70);
    printDQ(queue_1);
    push_back(queue_1, 80);
    printDQ(queue_1);
    push_front(queue_1, 90);
    printDQ(queue_1);
    push_back(queue_1, 100);
    printDQ(queue_1);
    pop_front(queue_1);
    printDQ(queue_1);
    pop_rear(queue_1);
    printDQ(queue_1);
    pop_front(queue_1);
    printDQ(queue_1);
    pop_rear(queue_1);
    printDQ(queue_1);
    pop_front(queue_1);
    printDQ(queue_1);
    pop_rear(queue_1);
    printDQ(queue_1);
    pop_front(queue_1);
    printDQ(queue_1);
    pop_rear(queue_1);
    printDQ(queue_1);
    pop_front(queue_1);
    printDQ(queue_1);
    pop_rear(queue_1);
    printDQ(queue_1);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> gcc 10_DoublyEndedQueue.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3> ./a.exe
Queue of capacity 10 created!
10 was pushed in front of queue
10
20 was pushed back to queue
10 20
30 was pushed in front of queue
30 10 20
40 was pushed back to queue
30 10 20 40
50 was pushed in front of queue
50 30 10 20 40
60 was pushed back to queue
50 30 10 20 40 60
70 was pushed in front of queue
70 50 30 10 20 40 60
80 was pushed back to queue
70 50 30 10 20 40 60 80
90 was pushed in front of queue
90 70 50 30 10 20 40 60 80
100 was pushed back to queue
90 70 50 30 10 20 40 60 80 100
90 was popped from front
70 50 30 10 20 40 60 80 100
100 was popped from rear
70 50 30 10 20 40 60 80
70 was popped from front
50 30 10 20 40 60 80
80 was popped from rear
50 30 10 20 40 60
50 was popped from front
30 10 20 40 60
60 was popped from rear
30 10 20 40
30 was popped from front
10 20 40
40 was popped from rear
10 20
10 was popped from front
20
20 was popped from rear
Dqueue is empty
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_3>
```