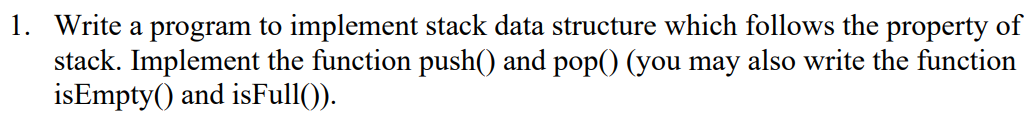
[LAB SUBMISSION – 3](https://github.com/shrishtinigam/DSA_Lab/tree/main/Lab_3)

# Meher Shrishti Nigam – 20BRS1193



**Solution in 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 illustration 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");

}

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);

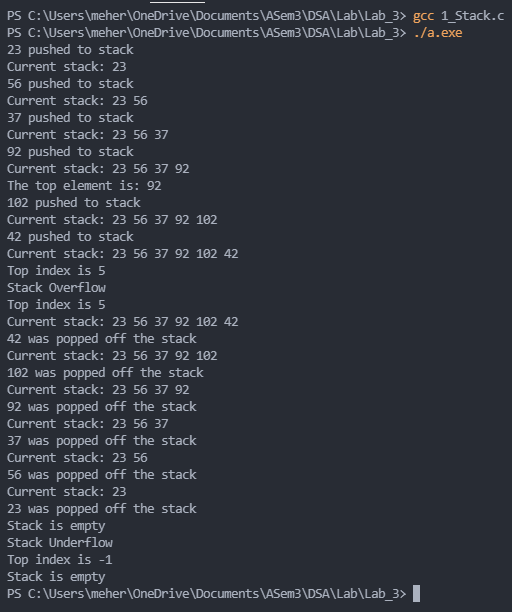
    printStack(stack\_1);

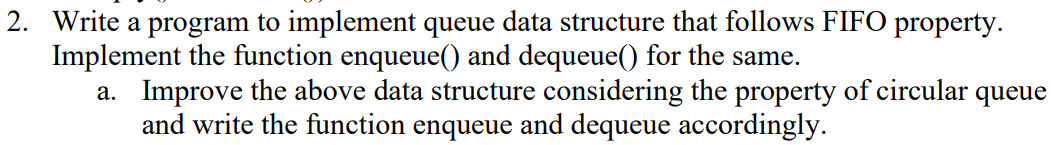
    pop(stack\_1); *// stack underflow*

    printf("Top index is %d\n", stack\_1->top);

    printStack(stack\_1);

}





*// 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 illustration 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);

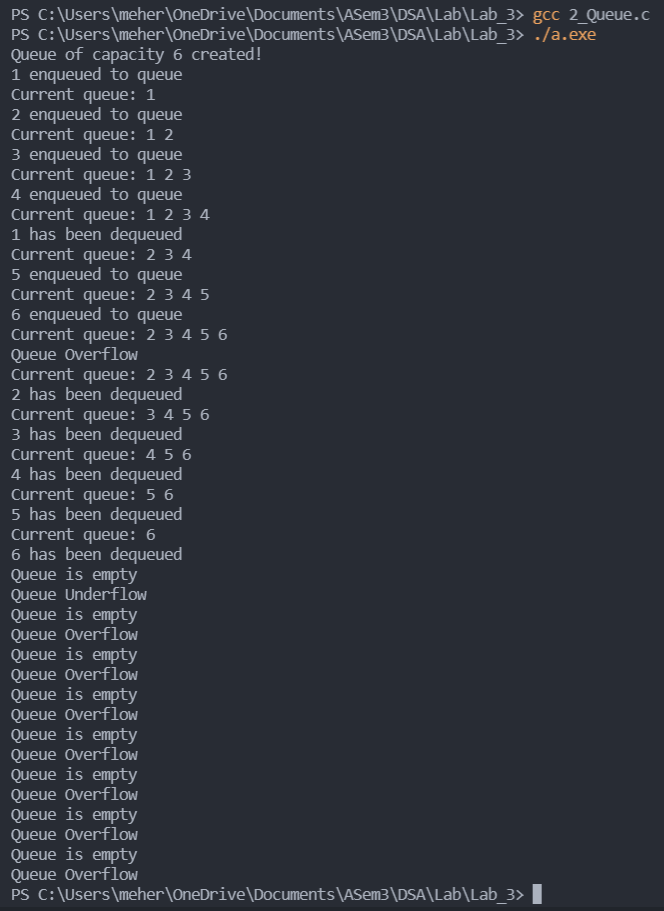
    printQueue(queue\_1);

    enqueue(queue\_1, 6);

    printQueue(queue\_1);

    enqueue(queue\_1, 7);

}



*// 2a\_CircularQueue.c*

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#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 illustration 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);

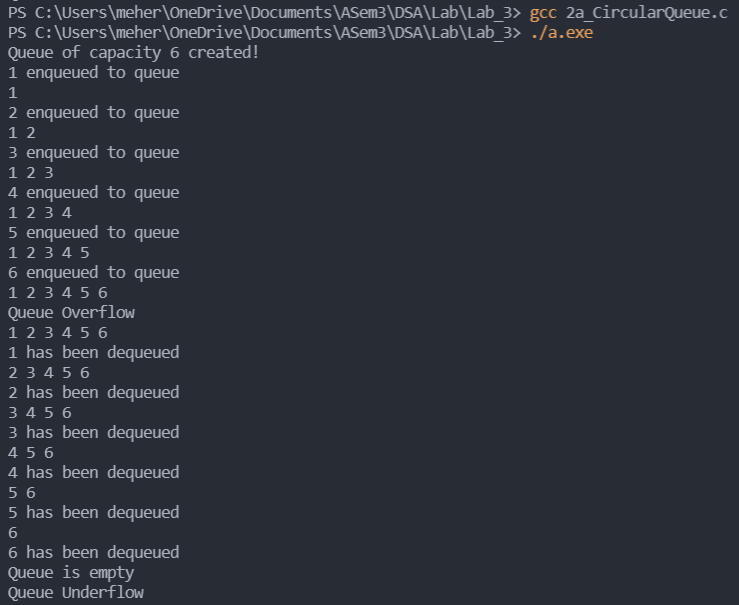
    printCQ(queue\_1);

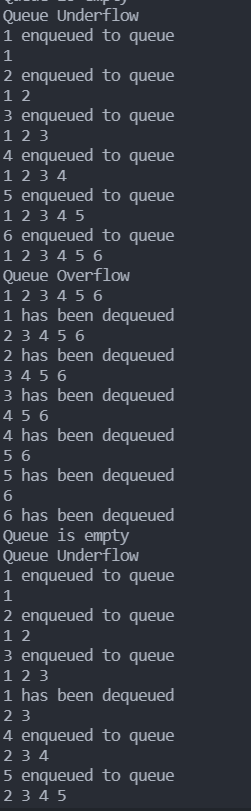
    dequeue(queue\_1);

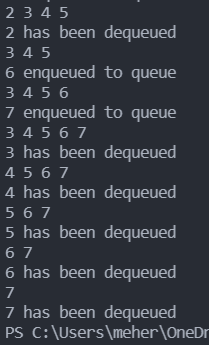
    printCQ(queue\_1);

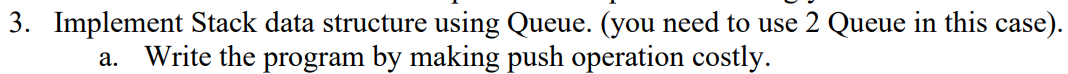
    dequeue(queue\_1);

}









*// 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;

}

*// This function should not be available, but it is used here for illustration 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);

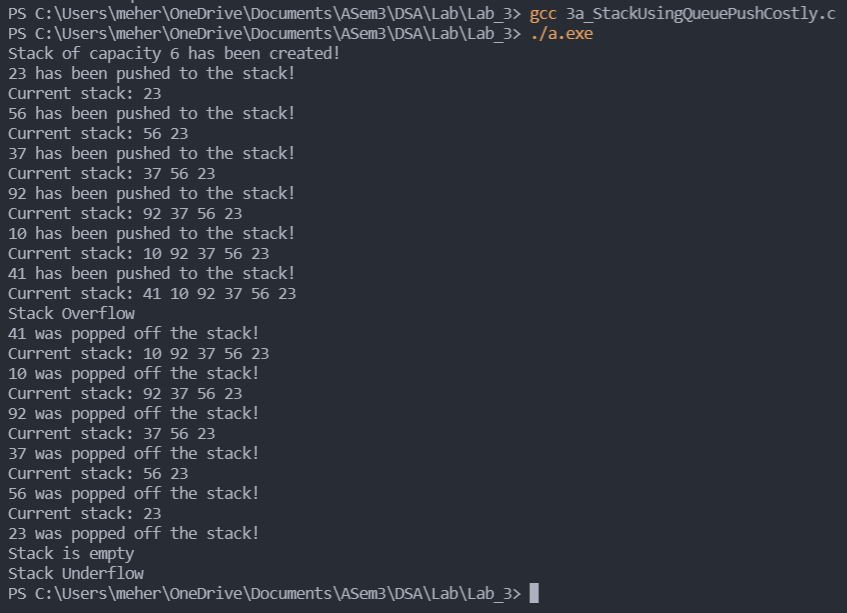
    printStack(stack\_1->queue\_1);

    pop(stack\_1);

    printStack(stack\_1->queue\_1);

    pop(stack\_1);

}





*// 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;

}

*// This function should not be available, but it is used here for illustration 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);

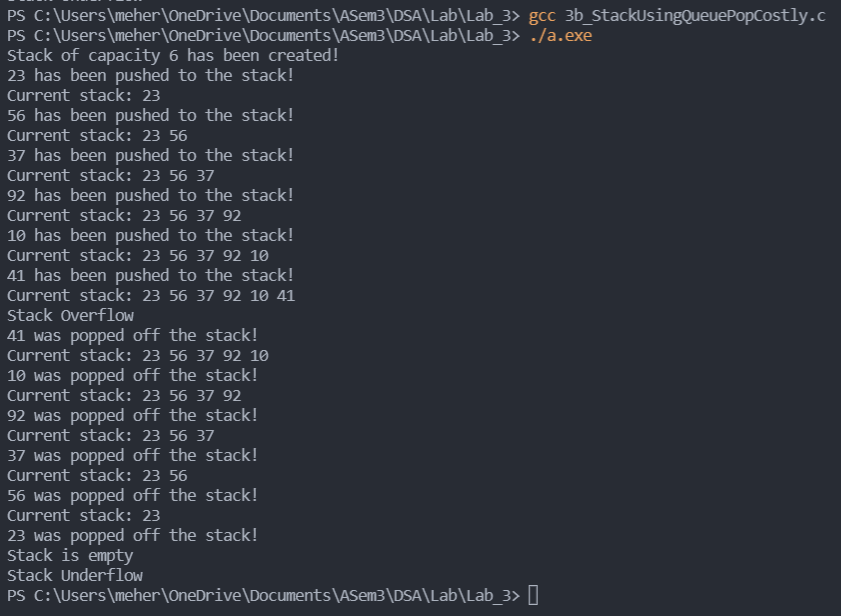
    printStack(stack\_1->queue\_1);

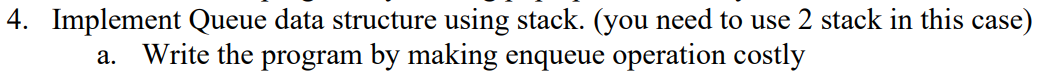
    pop(stack\_1);

    printStack(stack\_1->queue\_1);

    pop(stack\_1);

}





*// 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;

}

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 illustration 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);

    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);

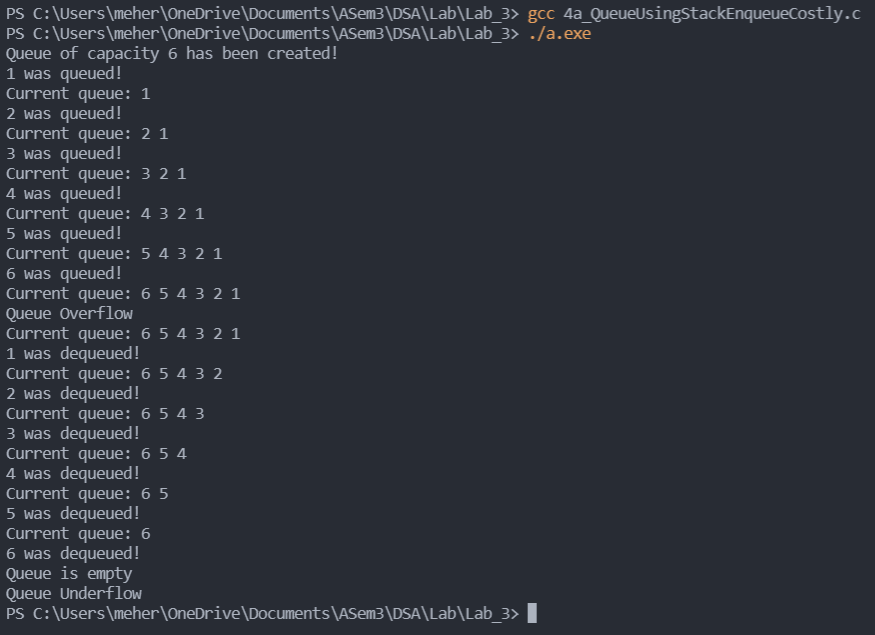
    printQueue(queue\_1->stack\_1);

    dequeue(queue\_1);

    printQueue(queue\_1->stack\_1);

    dequeue(queue\_1);

}





*// 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 illustration 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);

    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);

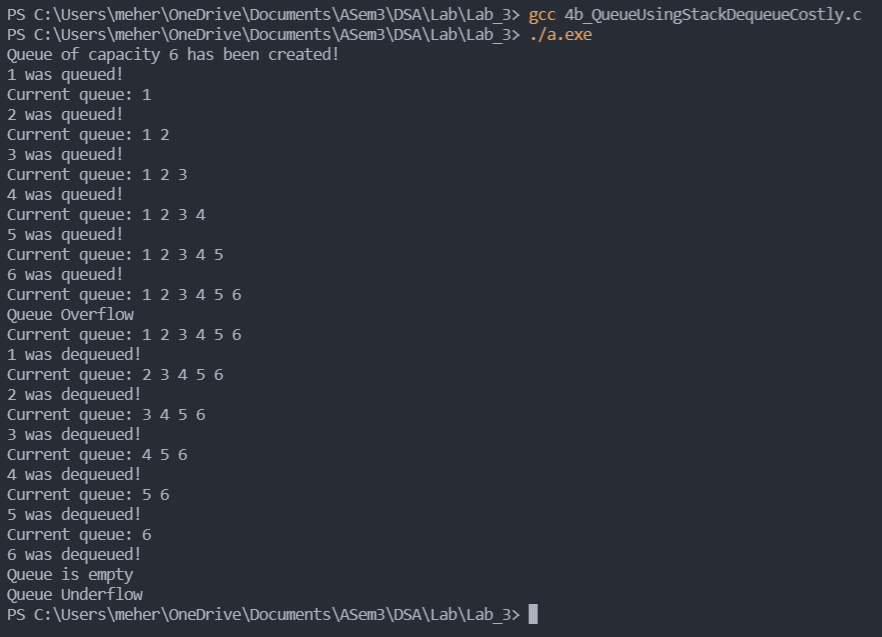
    printQueue(queue\_1->stack\_1);

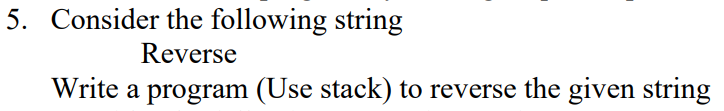
    dequeue(queue\_1);

    printQueue(queue\_1->stack\_1);

    dequeue(queue\_1);

}





*// 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++)

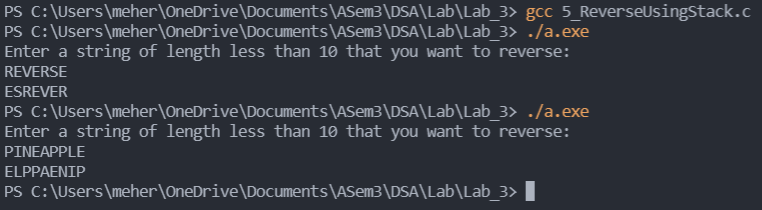
    {

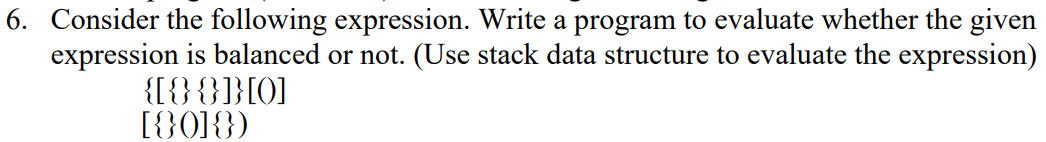
        char x = pop(stack\_1);

        printf("%c", x);

    }

}





*// 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");

    }

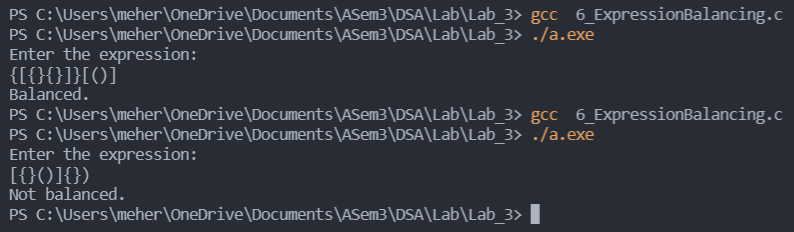
    else

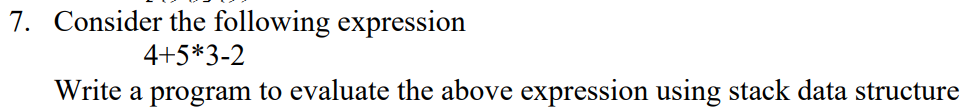
    {

        printf("Not balanced.\n");

    }

}





*// 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()

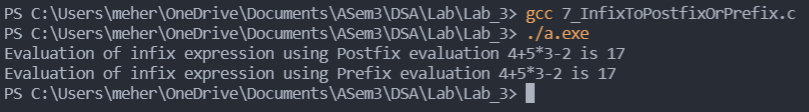
{

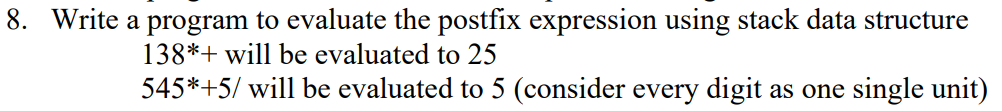
    char str[30] = "4+5\*3-2";

    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*

}





*// 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;

}

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()

{

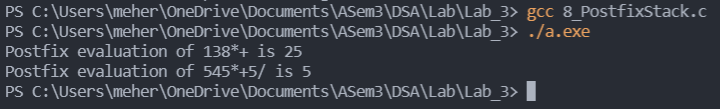
    char str[30] = "138\*+";

    printf("Postfix evaluation of %s is %d\n", str, Postfix(str));

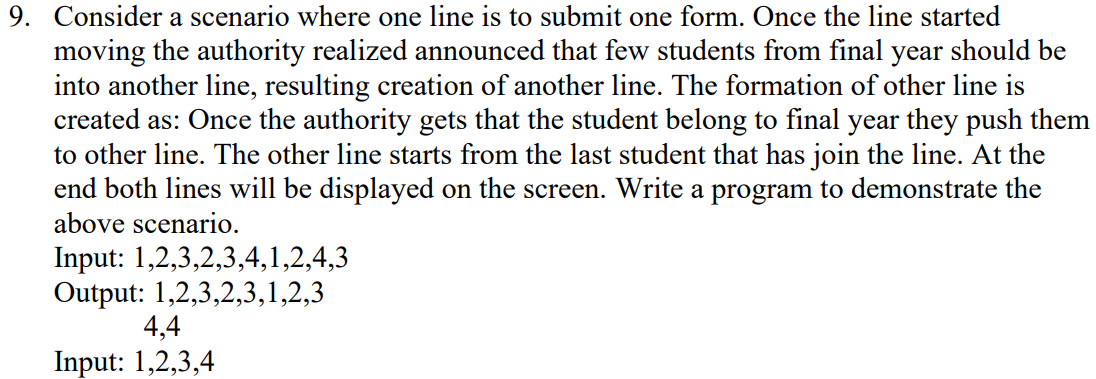
    char str2[30] = "545\*+5/";

    printf("Postfix evaluation of %s is %d\n", str2, Postfix(str2));

}



[**(Prefix evaluator in the github repository)**](https://github.com/shrishtinigam/DSA_Lab/blob/main/Lab_3/8_PrefixStack.c)



*// 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 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*

*\*/*

#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: ");

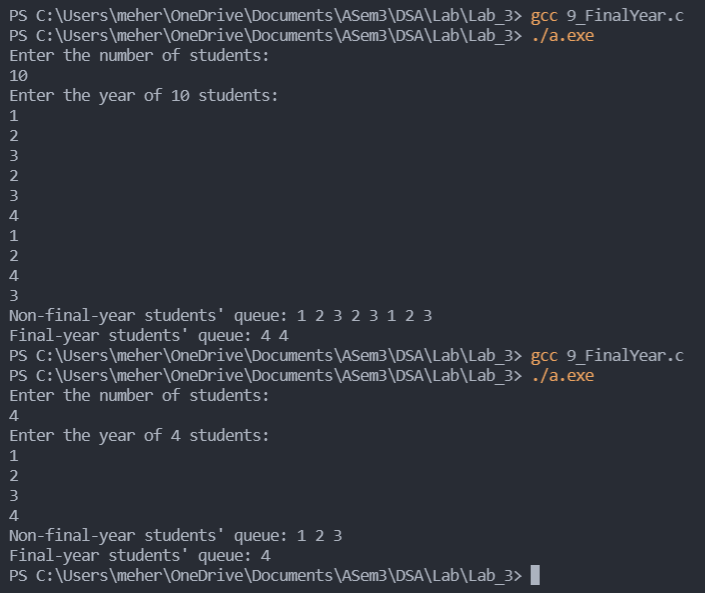
    while(!isEmpty(queue\_2))

    {

        printf("%d ", dequeue(queue\_2));

    }

}





**Doubly Ended Queue**

*// 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 illustration 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);

}

