CSCI2100C Tutorial3

ZHANG Xinyun

xyzhang21@cse.cuhk.edu.hk

- Display the content of a queue
- Reverse a queue
- Implement stack using queue
- Implement queue using stack

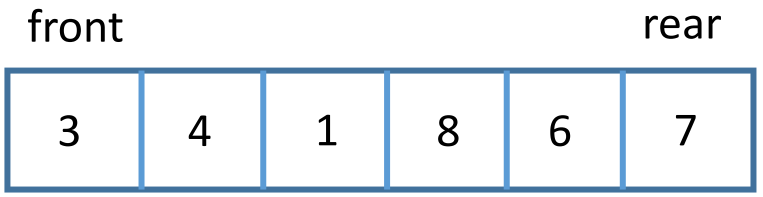
Exercise1

Display the content of a queue

Problem definition

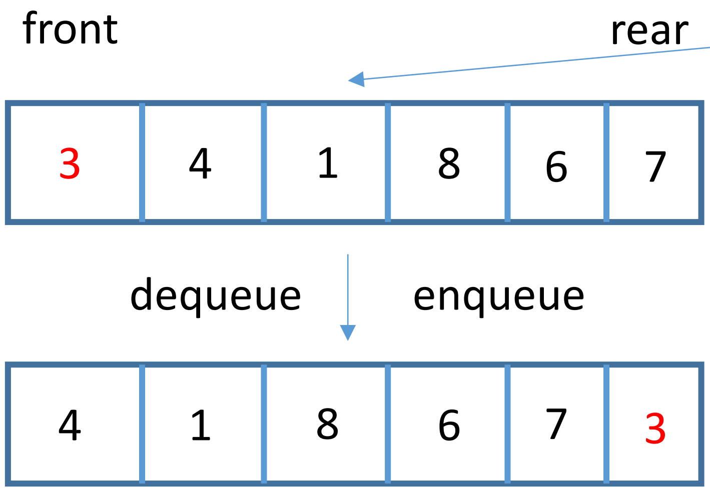
Write a program that displays the contents of a queue. The contents of the queue should be unchanged. Use the queue ADT to complete the exercise. Use the following function prototype, and define any function you use.

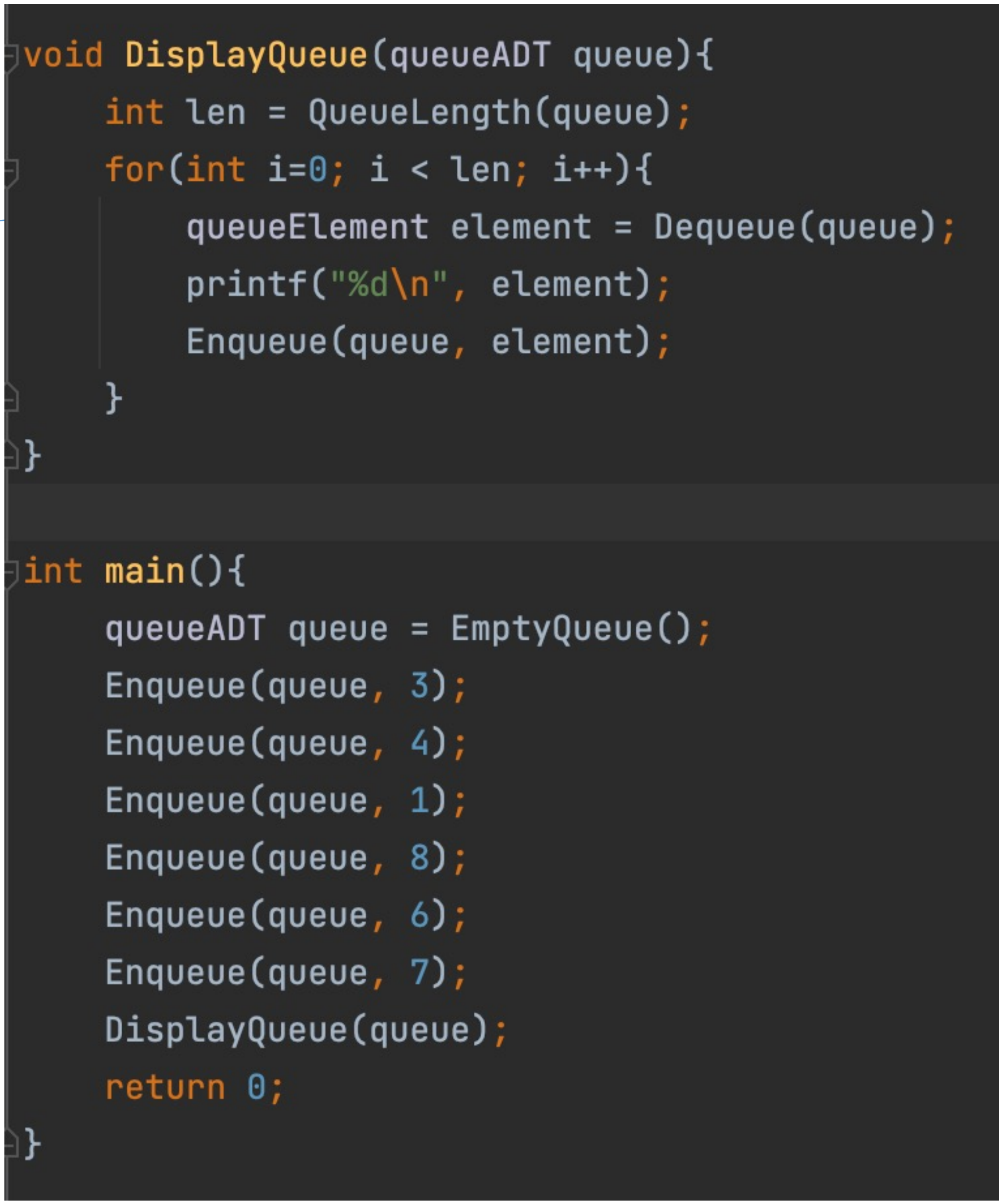
```
#include “queue.h”
void DisplayQueue(queueADT queue);
```



```
Out:  3    4    1    8    6    7
```

Answer





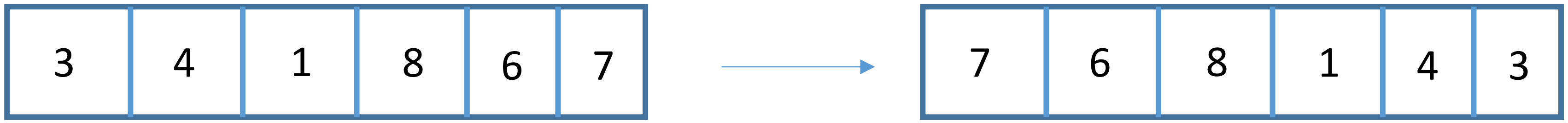
```
void DisplayQueue(queueADT queue){
    int len = QueueLength(queue);
    for(int i=0; i < len; i++){
        queueElement element = Dequeue(queue);
        printf("%d\n", element);
        Enqueue(queue, element);
    }
}

int main(){
    queueADT queue = EmptyQueue();
    Enqueue(queue, 3);
    Enqueue(queue, 4);
    Enqueue(queue, 1);
    Enqueue(queue, 8);
    Enqueue(queue, 6);
    Enqueue(queue, 7);
    DisplayQueue(queue);
    return 0;
}
```

Exercise2

Reverse a queue

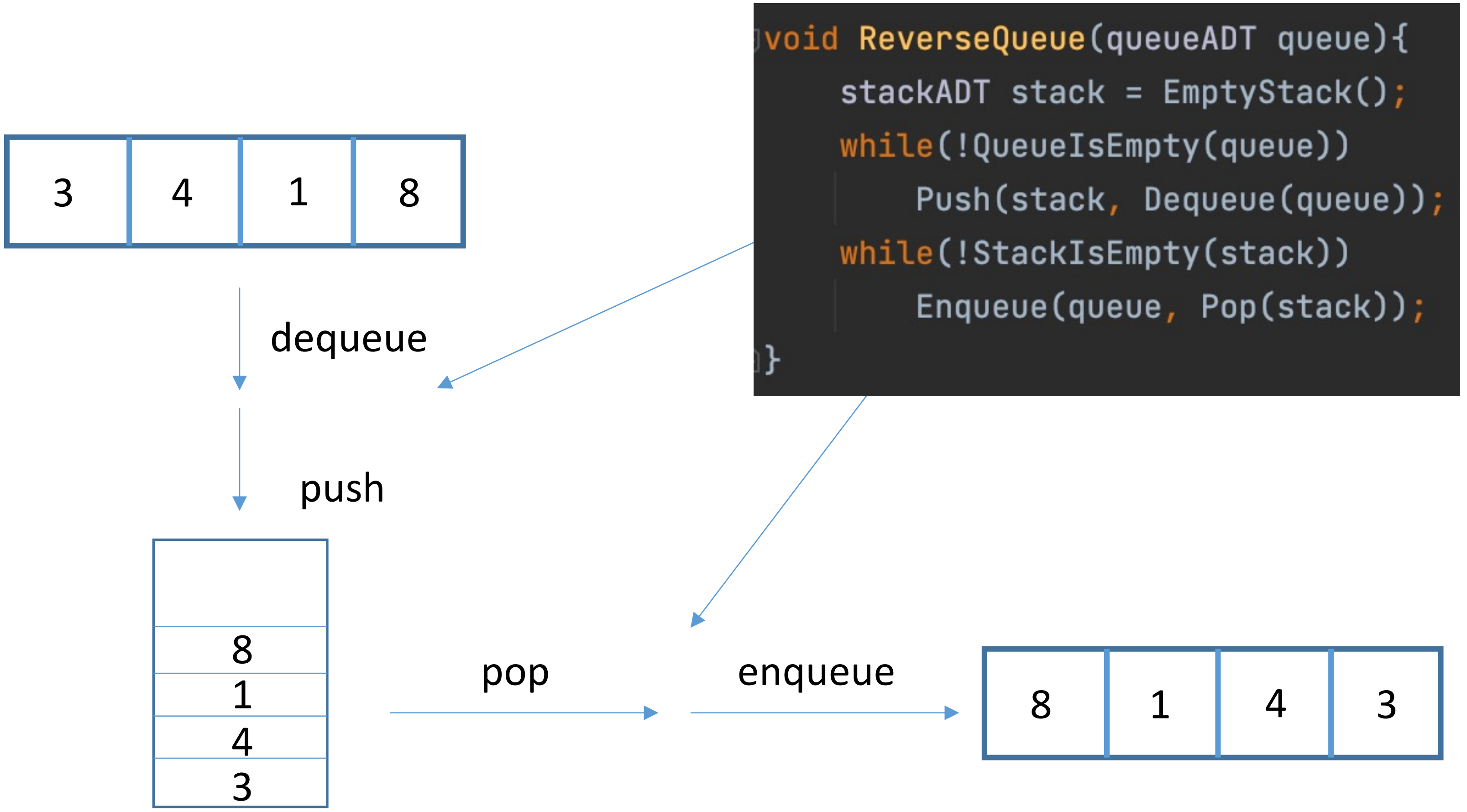
Problem definition

Write the C function ReverseQueue() in the application file **(NOT** the implementation file!). The function accepts a queueADT argument, and reverses the elements stored in it.



```
void ReverseQueue(queueADT queue);
```

Answer



```
void ReverseQueue(queueADT queue){
    stackADT stack = EmptyStack();
    while(!QueueIsEmpty(queue))
        Push(stack, Dequeue(queue));
    while(!StackIsEmpty(stack))
        Enqueue(queue, Pop(stack));
}
```

PS: There’s also an approach to use only queue to complete reversion.

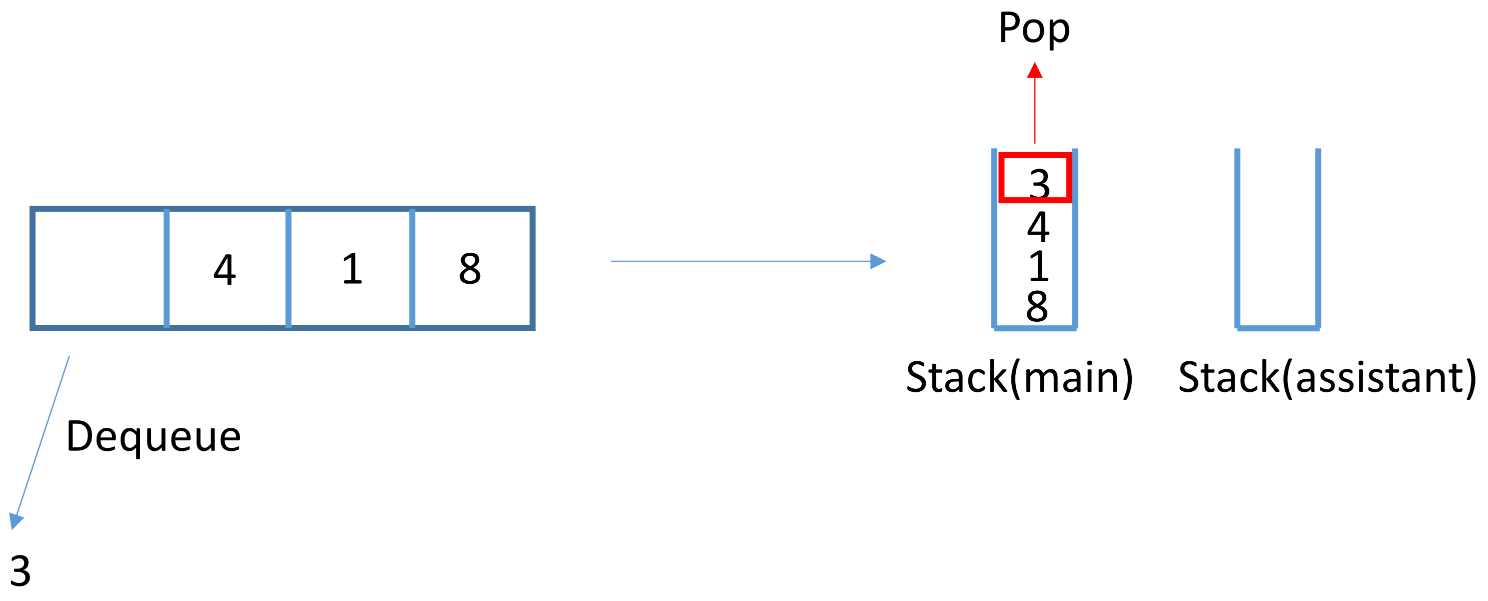
Exercise3

Implement queue using stack

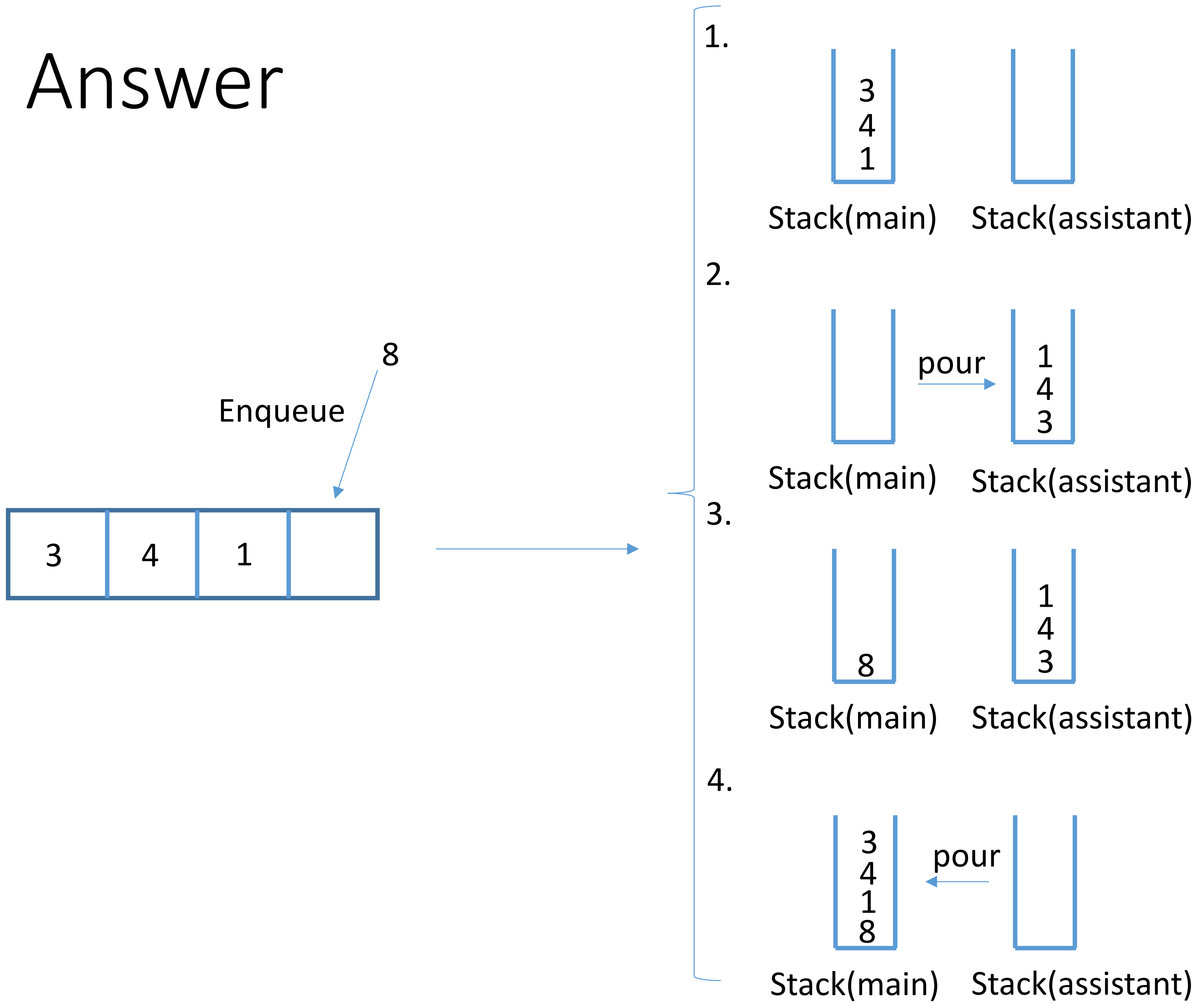
Problem Definition

Implement a first in first out (FIFO) queue using stacks. The implemented queue should support all the functions of a normal queue (enqueue, dequeue, etc.).
Hint: Make the top element of the stack becomes the front element in the queue. So we only need to design the function::enqueue.

Answer

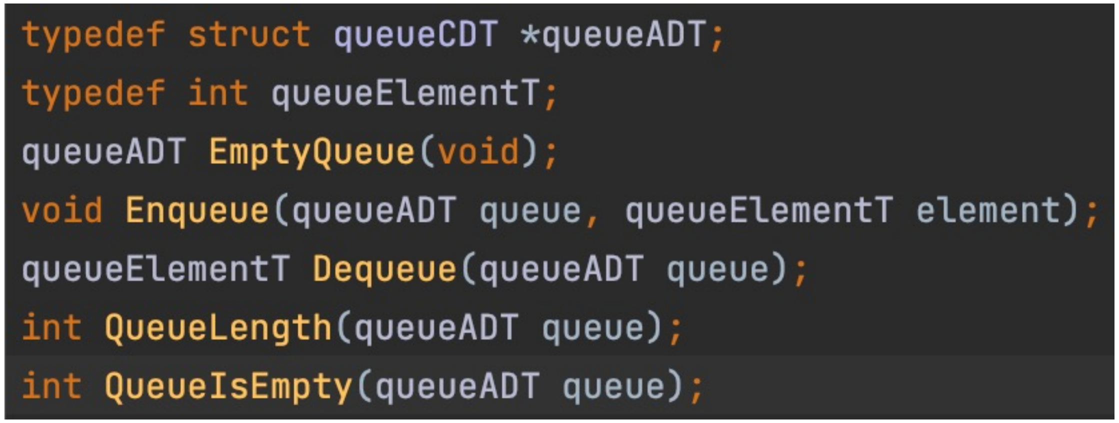


Answer


8
Enqueue
3
4
1
1.
3
4
1
Stack(main)
Stack(assistant)
2.
pour
1
4
3
Stack(main)
Stack(assistant)
3.
8
1
4
3
Stack(main)
Stack(assistant)
4.
3
4
1
8
pour
Stack(main)
Stack(assistant)

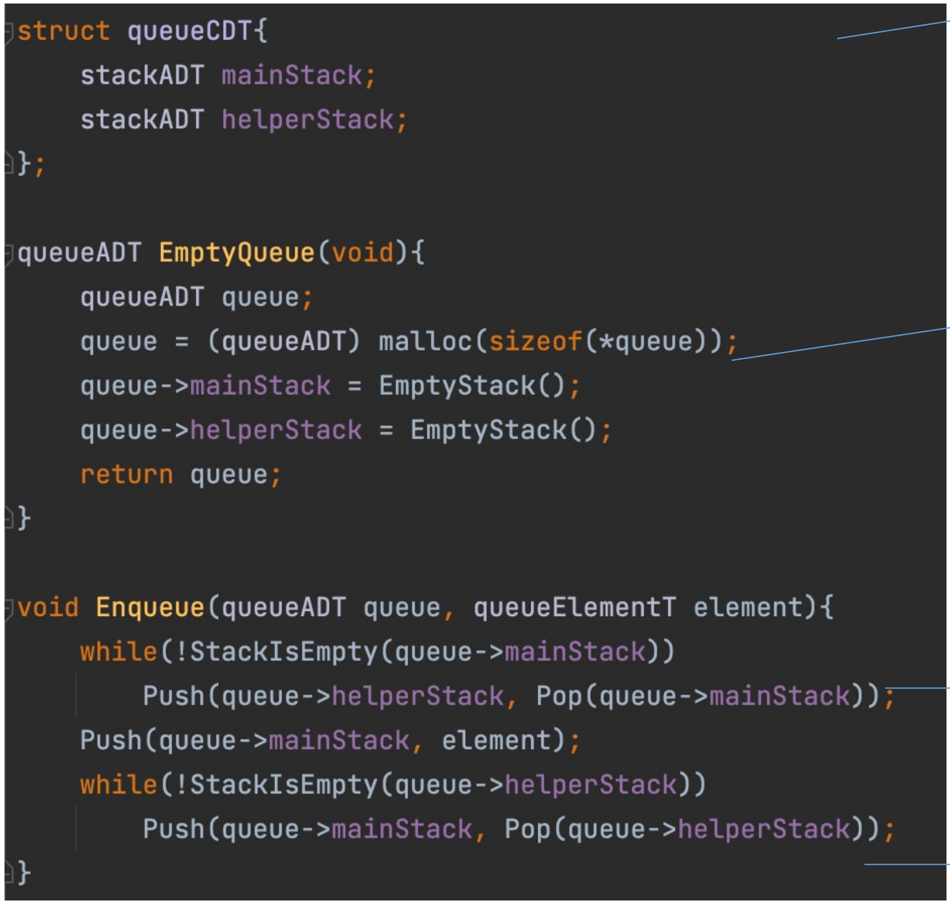
Answer stackqueue.h

// create a new queueADT



```c
typedef struct queueCDT *queueADT;
typedef int queueElementT;
queueADT EmptyQueue(void);
void Enqueue(queueADT queue, queueElementT element);
queueElementT Dequeue(queueADT queue);
int QueueLength(queueADT queue);
int QueueIsEmpty(queueADT queue);
```

Answer stackqueue.c



```c
struct queueCDT{
    stackADT mainStack;
    stackADT helperStack;
};

queueADT EmptyQueue(void){
    queueADT queue;
    queue = (queueADT) malloc(sizeof(*queue));
    queue->mainStack = EmptyStack();
    queue->helperStack = EmptyStack();
    return queue;
}

void Enqueue(queueADT queue, queueElementT element){
    while(!StackIsEmpty(queue->mainStack))
        Push(queue->helperStack, Pop(queue->mainStack));
    Push(queue->mainStack, element);
    while(!StackIsEmpty(queue->helperStack))
        Push(queue->mainStack, Pop(queue->helperStack));
}
```

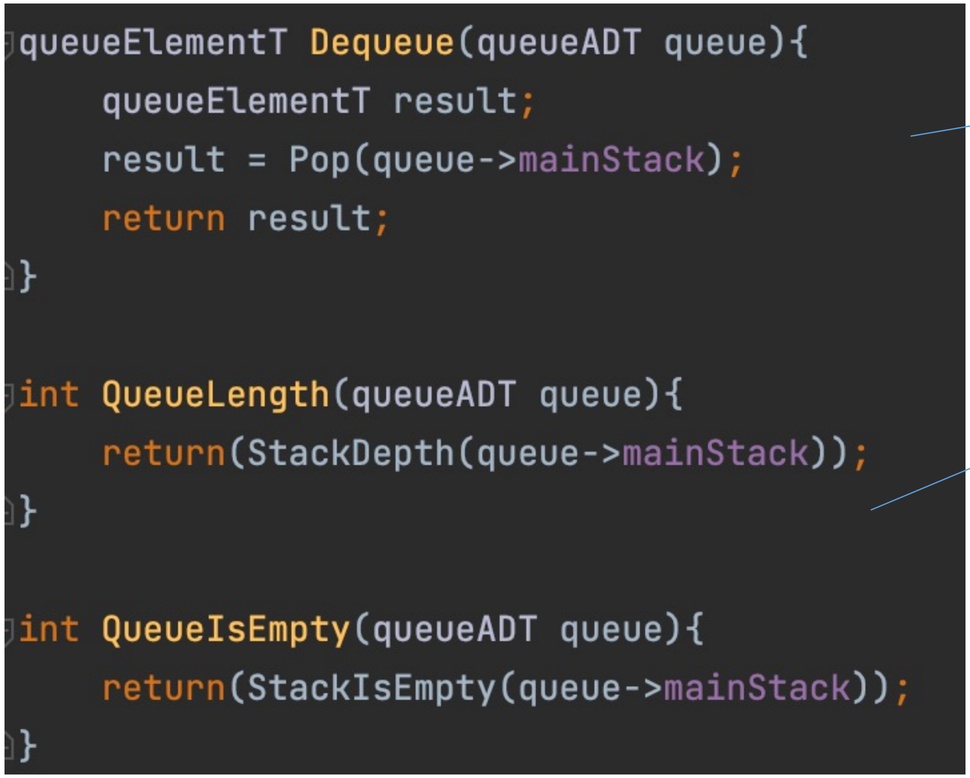
// define two stacks

// Initialize two stacks

// mainStack -> helperStack

// helperStack -> mainStack

Answer stackqueue.c



```c
queueElementT Dequeue(queueADT queue){
    queueElementT result;
    result = Pop(queue->mainStack);
    return result;
}

int QueueLength(queueADT queue){
    return(StackDepth(queue->mainStack));
}

int QueueIsEmpty(queueADT queue){
    return(StackIsEmpty(queue->mainStack));
}
```

// pop from mainStack

// simply call the
// stack function

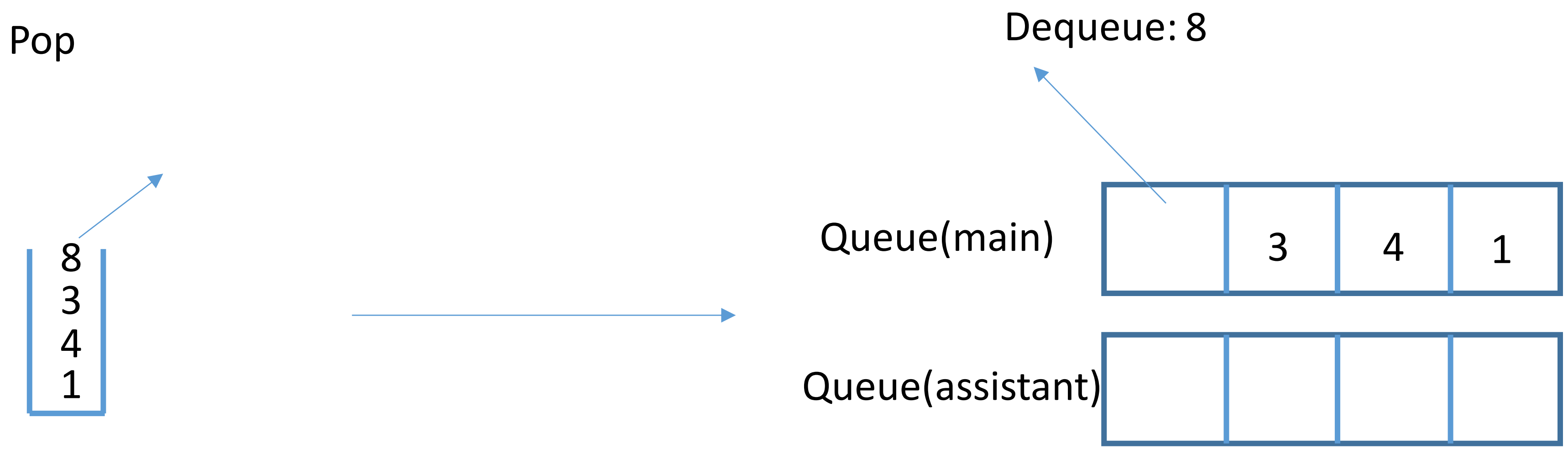
Exercise4

Implement stack using queue

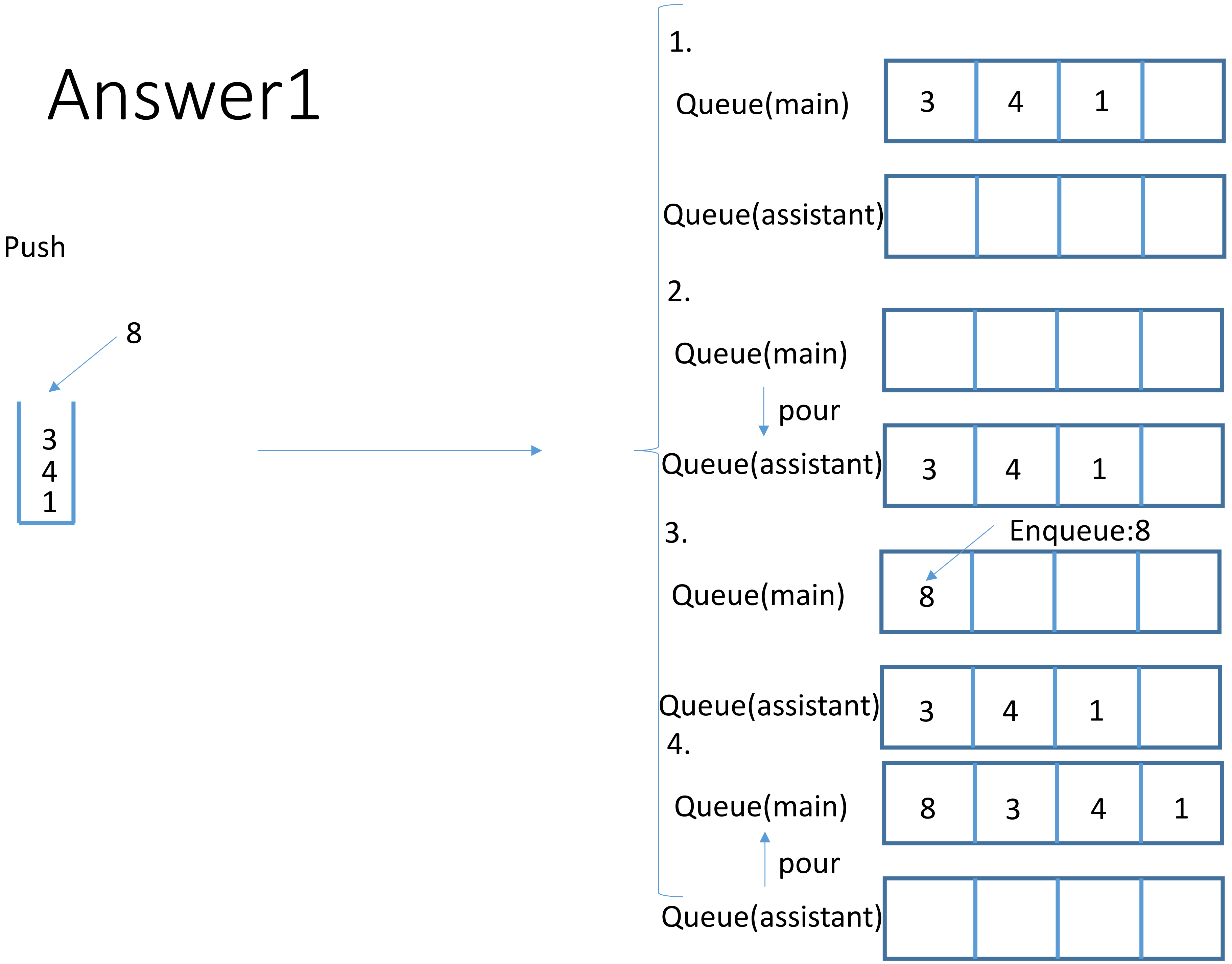
Problem Definition

Implement a last in first out (LIFO) stack using queues. The implemented stack should support all the functions of a normal stack (push, pop, etc.).
Hint: Make the front element of the queue becomes the top element in the stack. So we only need to design the function::push.

Answer1



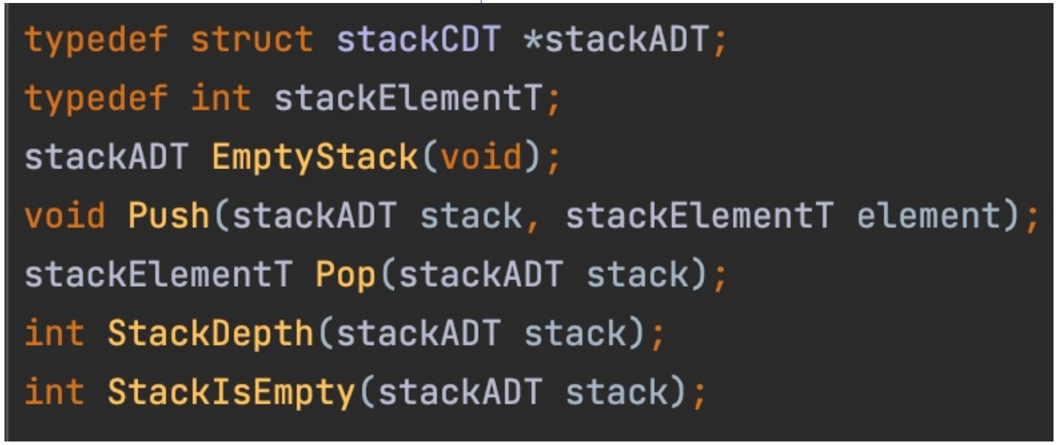
Answer1



Push
8
3
4
1
1.
Queue(main)
3
4
1
Queue(assistant)
2.
Queue(main)
pour
Queue(assistant)
3
4
1
3.
Enqueue:8
Queue(main)
8
Queue(assistant)
3
4
1
4.
Queue(main)
8
3
4
1
pour
Queue(assistant)

Answer1 queuestack.h

// create a new stackADT



```c
typedef struct stackCDT *stackADT;
typedef int stackElementT;
stackADT EmptyStack(void);
void Push(stackADT stack, stackElementT element);
stackElementT Pop(stackADT stack);
int StackDepth(stackADT stack);
int StackIsEmpty(stackADT stack);
```

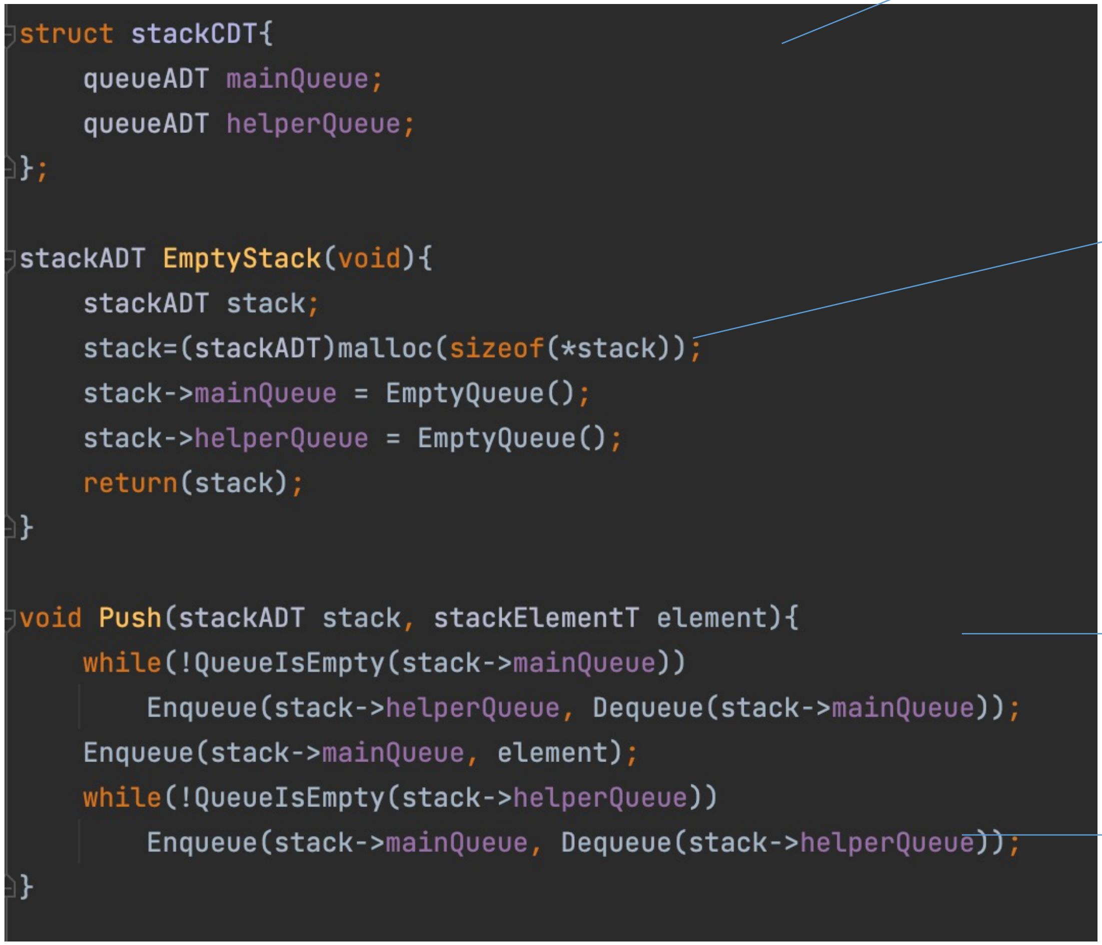
Answer1 queuestack.c

// define two queues

// Initialize two queues

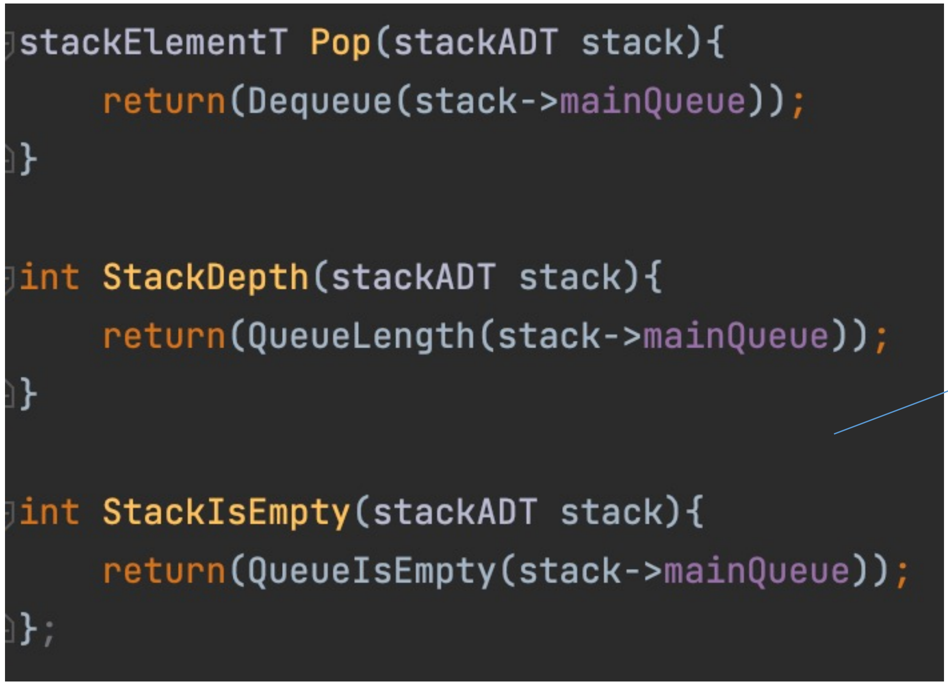
// mainQueue->helperQueue

// helperQueue->mainQueue



```c
struct stackCDT{
    queueADT mainQueue;
    queueADT helperQueue;
};

stackADT EmptyStack(void){
    stackADT stack;
    stack=(stackADT)malloc(sizeof(*stack));
    stack->mainQueue = EmptyQueue();
    stack->helperQueue = EmptyQueue();
    return(stack);
}

void Push(stackADT stack, stackElementT element){
    while(!QueueIsEmpty(stack->mainQueue))
        Enqueue(stack->helperQueue, Dequeue(stack->mainQueue));
    Enqueue(stack->mainQueue, element);
    while(!QueueIsEmpty(stack->helperQueue))
        Enqueue(stack->mainQueue, Dequeue(stack->helperQueue));
}
```

Answer1 queuestack.c



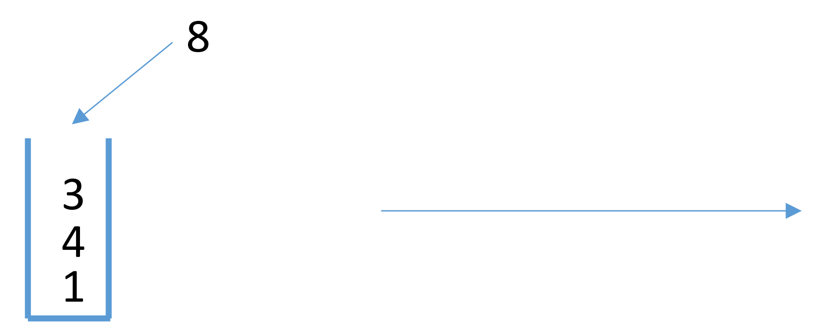
```
stackElementT Pop(stackADT stack){
    return(Dequeue(stack->mainQueue));
}

int StackDepth(stackADT stack){
    return(QueueLength(stack->mainQueue));
}

int StackIsEmpty(stackADT stack){
    return(QueueIsEmpty(stack->mainQueue));
};
```

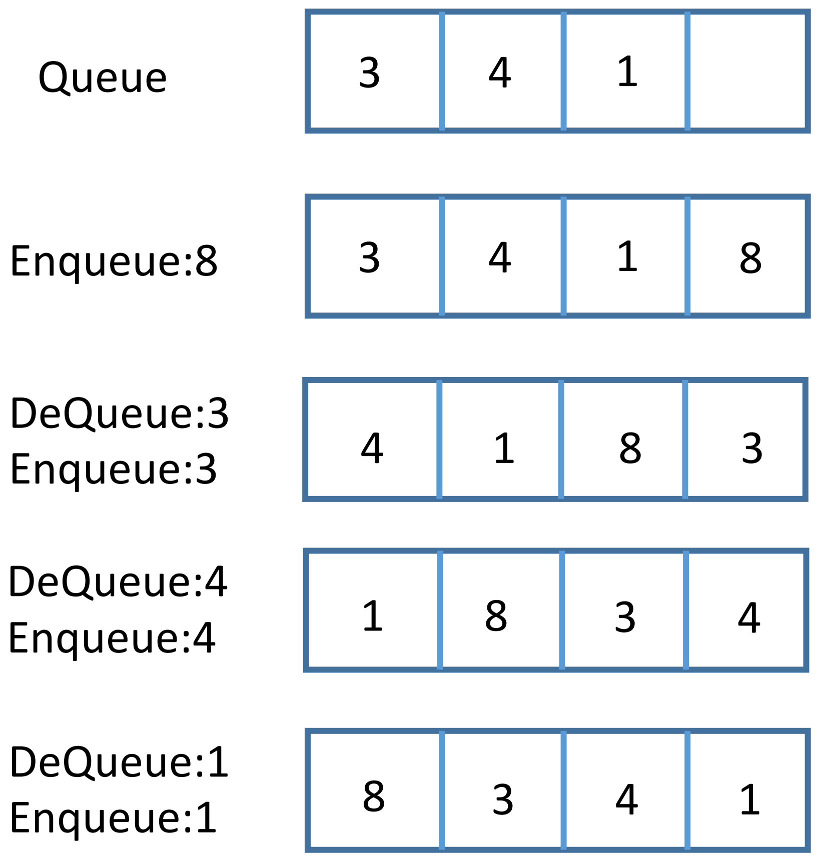
// dequeue from mainQueue

// simply call the
// queue function

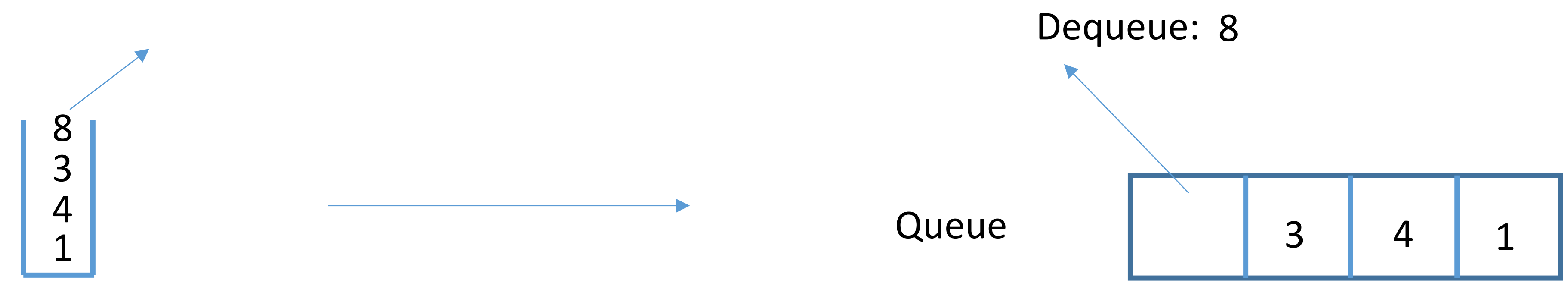
Answer2 using one queue

Push





Pop


8
3
4
1
Dequeue: 8
Queue
3 4 1