ESE-2025 Lab 3

Name: Shreya Mamadapur
Student ID: C0774035
Instructor: Takis Zourntos

Introduction

In this report, we are going through the definition and basic understanding of the implementation of stacks and queues in C. We will discover how to push in and pop out the data from stacks or queues(It is called enqueuing and dequeuing in queues).

Discussion

Stacks: Stacks is a data structure in C wherein the data can be stored in one direction - LIFO (Last-In-First-Out).
Adding/Inserting data into the stack is called the push operation. Retrieving data from the stack is called pop operation.
In stacks, there would be only one pointer to point at the latest value entering on top of the stack.
Queues: Queues are the data structure in C wherein the data is stored in FIFO(First-In Frist-Out) fashion.
Adding/Inserting data into the queue is called the enqueuing operation. Retrieving data from the queue is called the dequeuing operation.
In queues, we would need two pointers, one pointing the first data entered and the other pointing the latest data entered.

In the code attached, the tail of the queue is increased when a new element is added into it and its head is increased when an element is removed from it. And when the head equals the tail it can be said that the queue is empty. A queue overflow check is also included by which the program shows an overflow error if the queue is tried to increase above the size of the array(L).
The main program shows the functions in action. For more details please refer to the code in the appendix.

Conclusion

By implementing codes to depict the operations of stacks and queues, we understand how the stacks and queues actually work.

ESE-2025 Lab 3

Appendix

Source code:

```
/*
 * queue.c
 *
 * a simple, non-pointer queue implementation for the storage of integers
 *
 *
 *  Created on: Jun. 14, 2020
 *      Author: Shreya
 *      Adapted from: Takis (stack.c)
 */
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>

#define L 1024 // size of integer data structure

/************************
 * GLOBAL VARIABLES
 ************************
 */
int q[L]; // Queue storage
long q_tail = -1; // the Q tail, it increases as data is put in the q
long q_head = -1;  // the Q head, it increases as data is removed from the q

/*
 ************************
 * FUNCTIONS
 ************************
 */

/*
 * Queue_empty(s) implementation
 */
bool queue_empty(void)
{
        bool result;
```

```
	if (q_tail == q_head)//When Q is empty location of head == loc of tail
	{
		result = true;
	}
	else
	{
		result = false;
	}
	return result;
}

/*
 * Queue implementation
 */
void queue(int x)
{
	q[++q_tail] = x;
	if(q_tail >= L)//Q grows over the size of the array
	{
		 printf("Queue Overflow");
	}
	return;
}

/*
 * dequeue implementation
 */
int dequeue(void)
{
	if (queue_empty())
	{
		printf("underflow error\n");
		exit(EXIT_FAILURE);
	}
	else
	{
		return q[++q_head];
	}
}
```

ESE-2025 Lab 3

```
/************************
 * MAIN CODE
 ************************
 */
int main()
{
	int loadarr[] = { 52, -29, 36, -821, 790, -650, 1125, 72, 0, 68, 33, 59 };
	size_t N = sizeof(loadarr) / sizeof(int);

	/* print out contents of array*/
	printf("data to be loaded on to the stack:\n");
	for (size_t i = 0; i != N; ++i)
	{
		printf("%d ", loadarr[i]);
	}
	printf("\n");

	/* load Q */
	printf("Put data onto Q...\n");
	for (size_t i = 0; i != N; ++i)
	{
		queue(loadarr[i]);
	}

	/* Deque */
	printf("Dequeing the queue...\n");
	while (queue_empty() == false)
	{
		printf("%d ", dequeue());
	}

	// exit normally
	return EXIT_SUCCESS;
}
```