ESE-2014 Queue implementation using pointers

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).

Pointers to the queue structure are passed as arguments into the enqueue() and dequeue() function. And since the struct is passed as a pointer, the functions will be able to modify the contents of the actual stuct. The '->' operator is used inside the functions to access the internal members of the struct, this is because the functions are working on a pointer to the stuct and not the actual stuct itself. If it was the actual struct we would have used the '.' operator to access the stuct members.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. We also gain a little familiarity with pointers.

ESE-2014 Queue implementation using pointers

Appendix

Source Code:

```
/*
 * queue.h
 * Headerfile
 *  Created on: Jun. 20, 2020
 *    Author: Shreya
 *    Adapted from: Takis
 */

#ifndef INCLUDES_STACK_H_
#define INCLUDES_STACK_H_

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

#define     N     1024
/*
 * struct and typedef declarations
 */
struct queue_struct
{
      int data[N];
      size_t head;
      size_t tail;
};
typedef struct queue_struct queue_t;


bool queue_empty(queue_t *s);

void enqueue(queue_t *s, int x);

int dequeue(queue_t *s);

#endif /* INCLUDES_STACK_H_ */
```

ESE-2014 Queue implementation using pointers

```c
/*
 * Queue.c Functions
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "queue.h"


/* *************** FUNCTIONS ***************** */
/*
 * stack_empty() implementation
 */
bool queue_empty(queue_t *s)
{
	if (s->head == s->tail)
	{
		return true;
	}
	else
	{
		return false;
	}
}

/*
 * push() implementation
 */
void enqueue(queue_t *s, int x)
{
	s->data[(s->tail)++] = x; /* equivalent to: s -> data [(s->top)] = x;
(s->top)++; */
	/* also equivalent to: (s->top)++; s -> data [(s->top)-1] = x; */
	return;
}

/*
 * pop() implementation
 */
int dequeue(queue_t *s)
{
```

ESE-2014 Queue implementation using pointers

```
	if (queue_empty(s))
	{
		printf("underflow error!");
		exit(EXIT_FAILURE);
	}
	else
	{
		return (*s).data[(s->head)++]; /* could also write:
(s->data)[--(s->top)]; */
	}
}


/*
 * proj_queue.c
 * MAIN CODE
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "queue.h"

int main()
{
	/* declare and initialize variables */
	queue_t queue;
	queue.head = 0;
	queue.tail = 0;

	int loadarr[] = { 52, -29, 36, 1154, 72, 0, 68, 44, 33, 59 };
	size_t L = sizeof(loadarr)/sizeof(int);

	/* load queue */
	size_t i;
	for (i = 0; i != L; ++i)
	{
		printf("Queue.tail = %zu, ", queue.tail);
		enqueue(&queue, loadarr[i]);
		printf("Enqueuing %d\n", loadarr[i]);
	}
```

ESE-2014 Queue implementation using pointers

```
	/* dequeuing */
	int x;
	while (queue_empty(&queue) == false)
	{
		printf("Queue Head = %zu, ", queue.head);
		x = dequeue(&queue);
		printf("Dequeue %d\n", x);
	}
	/* test error function */
	//x = pop(&stack); /* comment this out to avoid error */
	return EXIT_SUCCESS;
}
```