Data Structures and Algorithms

Lab Week 3 - Stacks and Queues

Stack

- Stack is a LIFO (Last In First Out) data structure.
- Elements are always added and removed from the same end, which is the top of the stack.

Stack Operations

stack.h

```c
typedef struct stack {
    int *data;
    int top;
    int capacity;
} stack;

stack *create_stack(); // Create a new stack
bool is_empty(stack *s); // Check if stack is empty
void push(stack *s, int val); // Push an element to the stack
int top(stack *s); // Get the top element of the stack
int pop(stack *s); // Remove the top element from the stack
void display(stack *s); // Display the stack
int size(stack *s); // Get the size of the stack
```

stack.c

```c
#define INIT_SIZE 10

stack *create_stack() {
    stack *s = (stack *)malloc(sizeof(stack));
    s->data = (int *)malloc(sizeof(int) * INIT_SIZE);
    s->top = -1;
    s->capacity = 10;
    return s;
}

bool is_empty(stack *s) {
    return s->top == -1;
}

void push(stack *s, int val) {
    if (s->top == s->capacity - 1) {
        s->data = (int *)realloc(s->data, sizeof(int) * s->capacity * 2);
        s->capacity *= 2;
    }
    s->data[++s->top] = val;
}

int top(stack *s) {
    if (is_empty(s)) {
        return -1;
    }
    return s->data[s->top];
}

int pop(stack *s) {
    if (is_empty(s)) {
        return -1;
    }
    return s->data[s->top--];
}

void display(stack *s) {
    for (int i = 0; i <= s->top; i++) {
        printf("%d ", s->data[i]);
    }
    printf("\n");
}

int size(stack *s) {
    return s->top + 1;
}
```

Queue

- Queue is a FIFO (First In First Out) data structure.
- Elements are added at the rear end and removed from the front end.

Queue Operations

queue.h

```
typedef struct queue {
    int *data;
    int front;
    int size;
    int capacity;
} queue;

queue *create_queue(); // Create a new queue
bool is_empty(queue *q); // Check if queue is empty
```

```c
void enqueue(queue *q, int val); // Add an element to the back of the queue
int dequeue(queue *q); // Remove an element from the front of the queue
int front(queue *q); // Get the front element of the queue
void display(queue *q); // Display the queue
int size(queue *q); // Get the size of the queue
```

queue.c

```c
#define INIT_SIZE 10

queue *create_queue() {
    queue *q = (queue *)malloc(sizeof(queue));
    q->data = (int *)malloc(sizeof(int) * INIT_SIZE);
    q->front = 0;
    q->size = 0;
    q->capacity = INIT_SIZE;
    return q;
}

bool is_empty(queue *q) {
    return q->size == 0;
}

void enqueue(queue *q, int val) {
    if (q->size == q->capacity) {
        int *new_data = (int *)malloc(sizeof(int) * q->capacity * 2);
        for (int i = 0; i < q->size; i++) {
            new_data[i] = q->data[(q->front + i) % q->capacity];
        }
        free(q->data);
        q->data = new_data;
        q->front = 0;
        q->capacity *= 2;
    }
    int rear = (q->front + q->size) % q->capacity;
    q->data[rear] = val;
    q->size++;
}

int dequeue(queue *q) {
    if (is_empty(q)) {
        return -1;
    }
    int val = q->data[q->front];
    q->front++;
    q->front %= q->capacity;
    q->size--;
    return val;
}

int front(queue *q) {
    if (is_empty(q)) {
        return -1;
    }
    return q->data[q->front];
}

void display(queue *q) {
    for (int i = q->front; i < q->front + q->size; i++) {
        printf("%d ", q->data[i]);
    }
    printf("\n");
}

int size(queue *q) {
    return q->size;
}
```