UCSC Silicon Valley Extension

C Programming, Advanced

Assignment 3 - part 1: Data structures and graphs
Instructor : Radhika Grover

Include test cases in all of the following programs to demonstrate that the program executes correctly. The output carries 50% of the points for that problem. No points will be awarded if the program does not compile.

Linked List-based Stack

1. Write a C program to implement a stack using a linked list. The stack should support the following operations:
    a. void push(int item) : Create a new node with the given *item* and add it to the front of the list.
    b. void pop(): remove the item at the front of the list but do not return it.
    c. Node *peek(): return the item at the front of the list but do not remove it.

```
//Test case 1
push(stack, 1);// s: 1
push(stack, 2); // s: 2,1
peek(stack);//2
push(stack, 5); // s: 5,2,1
push(stack, 3); // s: 3,5,2,1
push(stack, 3); // s: 3,3,5,2,1
push(stack, 4); // s: 4,3,3,5,2,1
push(stack, 5); // s: 5,4,3,3,5,2,1
pop(stack); // s: 4,3,3,5,2,1
pop(stack); // s: 3,3,5,2,1
peek(stack);//3
pop(stack); // s: 3,5,2,1
pop(stack); // s: 5,2,1
peek(stack);// 5
pop(stack); // s:2,1
pop(stack); // s:1
pop(stack); // s: empty
pop(stack); // Error message
```

Queue

2. Write a C program to implement a **queue** using an array. The queue supports the following operations:
   a. size(queue): returns the number of items in the queue.
   b. enqueue(queue, item): adds the item to the end of the queue.
   c. dequeue(queue): removes and returns the item from the front of the queue.

```
//Test case 1
enqueue(queue, 1);//q: 1
enqueue(queue, 2); //q: 1,2
size(queue);//2
enqueue(queue, 5); //q:1, 2, 5
enqueue(queue, 3); //q:1, 2, 5, 3
enqueue(queue, 3); //q:1, 2, 5, 3, 3
enqueue(queue, 4); //q:1, 2, 5, 3, 3, 4
enqueue(queue, 5); //q:1, 2, 5, 3, 3, 4, 5
size(queue);//7
dequeue(queue); //q: 2, 5, 3, 3, 4, 5
dequeue(queue); //q: 5, 3, 3, 4, 5
size(queue);//5
```

Queue with two stacks

3. A stack is a LIFO (last in first out) structure, whereas a queue is FIFO (first in first out) structure. Write a program to implement a queue using two stacks.

The operations are described below :

a. `enqueue (queue, item)` : adds an item to the end of the queue.
b. `dequeue(queue)` : removes an item from the front of the queue.
c. `is_empty(queue)` : true if there are no items in the queue.
d. `get_size(queue)` : returns the number of elements in the queue.

The enqueue and dequeue operation works as follows :

```
enqueue (queue, x) : push x on the first stack
dequeue (queue) :if the second stack is empty:
                    pop each element in the first stack and
                    push it on the second stack;
                 then
        pop the top item of the second stack;
```

Note: check for underflow during the dequeue operation

```
//Test case 1
is_empty(queue); // True
enqueue(queue, 1);//q: 1
enqueue(queue, 2); //q: 1,2
```

```
size(queue);//2
enqueue(queue, 5); //q:1, 2, 5
enqueue(queue, 3); //q:1, 2, 5, 3
enqueue(queue, 3); //q:1, 2, 5, 3, 3
enqueue(queue, 4); //q:1, 2, 5, 3, 3, 4
enqueue(queue, 5); //q:1, 2, 5, 3, 3, 4, 5
size(queue);//7
dequeue(queue); //q: 2, 5, 3, 3, 4, 5
dequeue(queue); //q: 5, 3, 3, 4, 5
is_empty(queue); // False
size(queue);//5
```

Deque

4. Write a program to implement a deque using a linked list so that the following operations are possible :

a. `push(d, x)` : Insert item x on the front end of deque d.
b. `pop(d)` : Removing the front item from deque d and return it.
c. `inject(d, x)` : Insert item x on the rare end of deque d.
d. `eject(d)` : Remove the rear item from deque d and return it.
e. Write routines to support the deque that take O(1) time per operation. Should you use a singly linked or doubly linked list to implement the deque?

```
// Testcase 1
push(d, 7);
push(d, 6);
push(d 7);
push(d, 8);
print(d); //8, 7, 6, 7
pop(d);
pop(d);
print(d); //6, 7

// Testcase 2
inject(d, 7);
inject(d, 6);
inject(d, 7);
inject(d, 8);
print(d); // 7, 6, 7, 8
pop(d);
pop(d);
print(d); // 7, 8
eject(d);
eject(d);
```

```
        print(d); // List is empty

        // Testcase 3
        inject(d, 2);
        inject(d, 6);
        print(d); // 2, 6
        eject(d);
        eject(d);
        print(d); //List is empty
        push(d, 8);
        push(d, 9);
        print(d); // 9, 8
        pop(d);
       print(d); // 8

        // Testcase 4
        pop(d); // Error message
        eject(d); //Error message
        push(d, 2);
        push(d, 3);
        push(d, 4);
        push(d, 5);
        push(d, 6);
        push(d, 7);
        push(d, 8);
        push(d, 9);
        inject(d, 5);
        print(d); // 9, 8, 7, 6, 5, 4, 3, 2, 5
        pop(d);
        pop(d);
        print(d); // 7, 6, 5, 4, 3, 2, 5
        eject(d);
        eject(d);
    print(d); // 7, 6, 5, 4, 3
```