Lab 3
Stacks and Queues

Download any template files from the course page.

1) It's required to implement Stack (Array Based or Linkedlist Based) with the following functions:-
   a) Init: It initializes stack so that there are no elements inserted.
   Prototype → `void init(Stack *s);`
   b) Pop: It removes the last inserted element in the stack and returns it.
   Prototype → `Item pop(Stack *s);`
   c) Push: It inserts element at the top of the stack.
   Prototype → `void push(Stack *s, Item value);`
   d) Top: It returns the last inserted element in the stack without removing it.
   Prototype → `Item top(Stack *s);`
   e) isEmpty: It returns 1 if stack is empty or 0 otherwise.
   Prototype → `int isEmpty(Stack *s);`
   f) Destroy: It frees the memory and returns it to the system
   Prototype → `void destroy(Stack *s);`
2) Write a C function that takes a infix expression as input and convert it to postfix. Use your stack in the conversion.
   Prototype → `void infixToPostfix(char* infix, char* postfix);`
   Note that infix input is the infix expression and you should return the postfix expression.
3) Write a C function that takes a postfix expression as input and shows the value of the expression as output. The input will be a postfix (not infix) and you have to use your stack implementation to evaluate the expression.
   Prototype → `float evaluatePostfix(char* postfix);`
4) The main should take a string as input from user, convert it to postfix notation using `infixToPostfix()`, and then call `evaluatePostfix()`. ***ASSUME*** *that there is a space between each token in the input expression.*

Examples

- *Input:* `2 + 3 * 4`
  - *Postfix:* `2 3 4 * +`
  - *Result:* `14.000000`
- *Input:* `2 + ( -2.5 + 3.14 ) * ( -5.4 + 8.1 ) ^ ( -0.5 )`
  - *Postfix:* `2 -2.5 3.14 + -5.4 8.1 + -0.5 ^ * +`
  - *Result:* `2.389492`

5) It's required to implement Queue using the linked list implementation with the following functions:-
   a) **Init:** It initializes queue, and make the head pointer equals NULL.
      Prototype → `void init(Queue *q);`
   b) **Dequeue:** It returns the front of the queue after removing it from the queue.
      Prototype → `Item dequeue(Queue *q);`
   c) Enqueue: It adds a new element to the back of the queue.
      Prototype → `void enqueue(Queue *q, Item value);`
   d) isEmpty: It returns 1 if queue is empty or 0 otherwise.
      Prototype → `int isEmpty(Queue *q);`
   e) Destroy: It frees the memory and returns it to the system
      Prototype → `void destroy(Queue *q);`
6) One of the main functions of an operation system (OS) is to schedule programs in executions (aka "processes"). Round Robin is a popular scheduling algorithm, where new processes join the back of a FIFO queue. The OS picks the process at the front of the queue to run on the processor. After a specified time (called quantum), the OS preempts the process, takes it away from the processor, puts it at the back of the queue, and picks the process at the front of the queue to run on the processor, and so on. If a process halts (i.e., finishes execution), it leaves the processor and the OS picks the process from the queue front to run on the processor.
   You will be given a file with the processes information. Each process will have: (1) a name, (2) the time slot that the process gets created and joins the queue for the first time, and (3) the time needed for its execution. Fill in the function `RoundRobin` that prints the scheduling of the processes, showing when each process halts and when the processor is idle (no process is running and the queue is empty). Assume, quantum of 1 time slot.

Example Input:

```
P1 0 3
P2 0 2
P3 3 1
P4 10 3
```

Example Output:

```
P1 (0-->1)
P2 (1-->2)
P1 (2-->3)
P2 (3-->4) P2 halts
P3 (4-->5) P3 halts
P1 (5-->6) P1 halts
idle (6-->7)
idle (7-->8)
idle (8-->9)
idle (9-->10)
P4 (10-->11)
P4 (11-->12)
P4 (12-->13) P4 halts
EOF
```

How to submit?

- Rename the template to be your student id. If your id is 9876, then rename the file to 9876-stack.c and 9876-queue.c
- Upload the source code using **https://goo.gl/forms/2NFm6PtaWrzPuuo12**

Notes

- You should work individually.
