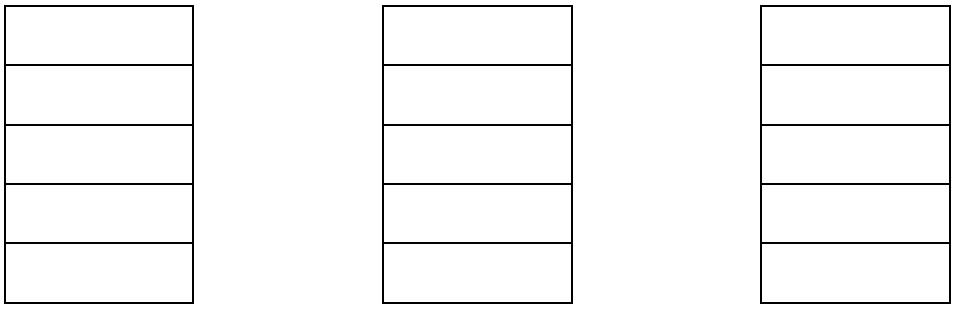
COP 3502 Study Group Sheet: Stacks, Queues

Directions: Work together as a group to try to solve these problems. Talk through issues and see if you can convince yourselves of the right path to move forward. In groups with a TA/ULA, towards the end of the session some of the solutions will be covered. At the end of the week, the solutions will be posted for everyone.

1) Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (1, 2, and 3) in the infix expression.

1 2 3

A + B * ((C / D) + E * F) * G



1 2 3

Resulting postfix expression:

2) Consider the following function:

```
void doTheThing(void)
{
  int i, n = 9;  // Note: There are 9 elements in the following array.
  int array[] = {3, 18, 58, 23, 12, 31, 19, 26, 3};

  Stack *s1 = createStack();
  Stack *s2 = createStack();
  Queue *q = createQueue();

  for (i = 0; i < n; i++)
    push(s1, array[i]);

  while (!isEmptyStack(s1))
  {
     while (!isEmptyStack(s1))
       enqueue(q, pop(s1));   // pop element from s1 and enqueue it in q
     while (!isEmptyQueue(q))
       push(s2, dequeue(q));  // dequeue from q and push onto s2

     printf("%d ", pop(s2));  // pop from s2 and print element

     while (!isEmptyStack(s2))
       push(s1, pop(s2));     // pop from s2 and push onto s1
  }
  printf("Tada!\n");

  freeStack(s1);
  freeStack(s2);
  freeQueue(q);
}
```

What will be the <u>exact</u> output of the function above? (You may assume the existence of all functions written in the code, such as *createStack()*, *createQueue()*, *push()*, *pop()*, and so on.)

3) Evaluate the following postfix expression shown below, using the algorithm that utilizes an operand stack. Put the value of the expression in the slot provided and show the state of the operand stack (in this case the stacks should just have numbers in them) at each of the indicated points in the expression:

3 4 2 1 + * 24 **A** 6 / 9 **B** 4 - **C** + - /

A

B

C

Value of the Postfix Expression: _____

4) Suppose we have implemented a stack using a linked list. The structure of each node of the linked list is shown below. The stack structure contains a pointer to the head of a linked list and an integer, size, to indicate how many items are on the stack.

```
typedef struct node {
    int num;
    struct node* next;
} node;

typedef struct Stack {
    struct node *top;
    int size;
} stack;
```

Write a function that will pop off the contents of the input stack and push them onto a newly created stack, returning a pointer to the newly created stack. In effect, your function should reverse the order of the items in the original stack, placing them in a new stack. Assume you have access to all of the usual stack functions. Assume that when you push an item onto the stack, its size automatically gets updated by the push function. Similarly for pop, size gets updated appropriately when you pop an item from a stack. Do NOT call pop or peek on an empty stack.

```
void push(stack *s, int number); // Pushes number onto stack.
int pop(stack *s);  // Pops value at top of stack, returns it.
int peek(stack *s); // Returns value at top of stack.
int isEmpty(stack *s); // Returns 1 iff the stack is empty.


stack* reverseStack(stack* s) {

    stack *newS = malloc(sizeof(stack));









    return newS;
}
```

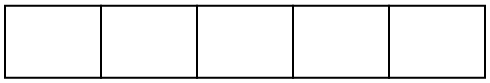
5) Consider the circular array implementation of a queue named Q, implemented with the structure shown below.

```
struct queue {
    int *array;
    int num_elements;
    int front;
    int capacity;
};
```

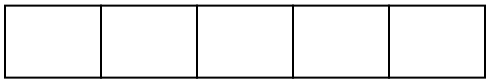
Suppose the queue is created with a capacity of 5 and front and num_elements are initialized to 0. Trace the status of the queue by showing the valid elements in the queue and the position of front after each of the operations shown below. Indicate front by making bold the element at the front of the queue.

1. enqueue(Q, 50);
2. enqueue(Q, 34);
3. enqueue(Q, 91);
4. x = dequeue(Q);
5. enqueue(Q, 23);
6. y = dequeue(Q);
7. enqueue(Q, y);
8. enqueue(Q, 15);
9. enqueue(Q, x);
10. x = dequeue(Q);

After stmt #1:

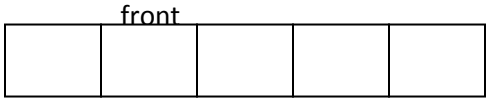


After stmt #2:

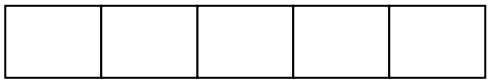


After stmt #3:

front

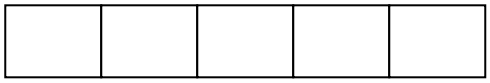


After stmt #4:

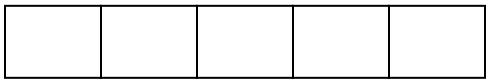


After stmt #5:

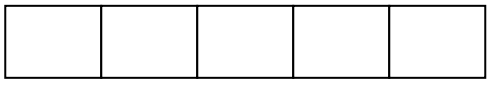
After stmt #6:



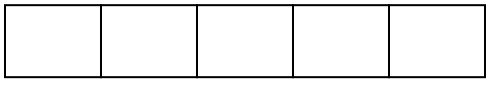
After stmt #7:



After stmt #8:



After stmt #9:



After stmt #10: