Practice Multiple Choice Exam: Computer Science I (Stacks, Queues)

The following code implements a stack using an array. Several lines of the implementation have been omitted. Questions 1 - 5 will be about these missing lines.

```
#define SIZE 10
#define EMPTY -1

struct stack {
    int items[SIZE];
    int top;
};

// Initializes an empty stack.
void initialize(struct stack* stackPtr) {
     stackPtr->top = 0;
}

// Attempts to push value onto the stack pointed to by stackPtr. Returns
// 1 if the push was successful, 0 otherwise.
int push(struct stack* stackPtr, int value) {

    if (full(stackPtr))
        return 0;

    stackPtr->items[/*** Q1 ***/] = value;
    /*** Q2 ***/
    return 1;
}

// Returns 1 iff the stack pointed to by stackPtr is full, 0 otherwise.
int full(struct stack* stackPtr) {
    return /*** Q3 ***/;
}

// If the stack is non-empty, the top of the stack is returned. Otherwise
// -1 (EMPTY) is returned.
int top(struct stack* stackPtr) {

    if (/*** Q4 ***/)
        return EMPTY;

    return /*** Q5 ***/;
}
```

The following code implements a queue using a linked list. Several lines of the implementation have been omitted. Questions 6 - 10 will be about these missing lines.

```
#define EMPTY -1

struct node {
    int data;
    struct node* next;
};

struct queue {
    struct node* front;
    struct node* back;
};

void init(struct queue* qPtr) {
    qPtr->front = NULL;
    qPtr->back = NULL;
}

// Pre-condition: qPtr points to a valid struct queue and val is the value to
//                enqueue into the queue pointed to by qPtr.
// Post-condition: If the operation is successful, 1 will be returned,
//      otherwise no change will be made to the queue and 0 will be returned.
int enqueue(struct queue* qPtr, int val) {

    struct node* temp = (struct node*)malloc(sizeof(struct node));


    if (temp != NULL) {
        temp->data = val;
        temp->next = NULL;

        if (qPtr->back != NULL)
            /*** Q6 ***/

        /*** Q7 ***/

        if (qPtr->front == NULL)
            qPtr->front = /*** Q8 ***/;

        /*** Q9 ***/
    }
    else
        /*** Q10 ***/
}
```

11) What is the value of the following postfix expression?

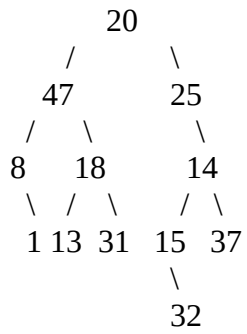
5 8 * 9 - 3 - 2 2 + /

12) What is the correct conversion of the following infix expression into postfix?

(4 + 5) / (8 - (2 + 3 * (5 - 4)))

13) Consider implementing a queue with a linked list, with only a pointer to the front of the queue. Which of the functions, in this implementation, would still have an O(1) run time? (Answer with a name that is appropriate based on the convention for queue functions.)

14) What is the following is the preorder traversal of the following binary tree?



15) Which line of code should replace /*** insert code ***/ in the function shown below, if the goal of the function is to return the number of nodes in the binary tree pointed to by root? Note: the node that root directly points to should be counted as well as all the ones "underneath" that one.

```
struct treeNode {
    int data;
    treeNode* left;
    treeNode* right;
};
int max(int a, int b) {
    if (a > b) return a;
    return b;
}

int countNodes(struct treeNode* root) {
    if (root == NULL) return 0;
    /*** insert code ***/
}
```

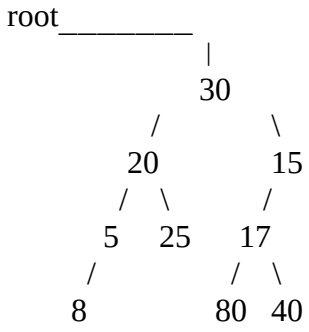
16) For a tree with n nodes, what is the average case run-time of the completed function shown in the previous question?

17) What is the output of the function call f(root) where f is the function shown below and root is a pointer to the root of the tree shown below? (Note: use the struct defintion from the two previous questions.)

```
void f(struct treeNode* root) {

    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL)
        return root->data;

    int left = f(root->left);
    int right = f(root->right);
    return left + right;
}
```



18) The depth of a node in a binary tree is the distance of that node from the root. Write a **recursive** function that takes in a pointer to the root of a binary tree and returns the ***sum of the depths*** of the nodes of the tree. (For example, a complete binary tree of 7 nodes has 1 node with depth 0, 2 nodes with depth 1 and 4 nodes with depth 2, for a sum of depths of nodes of 0 + 2(1) + 4(2) = 10. Use the struct definition and function prototype given below. You may also assume that curDepth in sumDepthRec represents the depth of root within the whole binary tree.

```
typedef struct treenode {
    int data;
    struct treenode *left;
    struct treenode *right;
} treenode;

double sumDepth(treenode* root) {
    return sumDepthRec(root, 0);
}

double sumDepthRec(treenode* root, int curDepth) {
    // Fill in
}
```