CS1 Exam 2 Review Questions (3/21/2017) & Answers

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 ***/;
}
```

1) What expression should replace /*** Q1 ***/?

a) value b) stackPtr->top-1 **c) stackPtr->top**

d) stackPtr->top+1 e) None of the Above

Reason: The initialize function sets top to 0 and this corresponds to an empty stack. So, whatever top equals is actually the location where the next push must be made.

2) What line of code should replace /*** Q2 ***/?

a) stackPtr->top++; b) stackPtr->top--; c) top++;

d) top--; e) None of the Above

Reason: We have added an item to the stack so the new top (first empty slot) is one higher than it used to be.

3) What expression should replace /*** Q3 ***/?

a) top == 0 b) stackPtr->top == 0 c) stackPtr->top == SIZE-1

d) stackPtr->top == SIZE e) None of the Above

Reason: top is the location we would add the next push. But, if this location is SIZE, that is an array out of bounds and the stack is full.

4) What expression should replace /*** Q4 ***/?

a) top == -1 b) stackPtr->top == -1 **c) stackPtr->top == 0**

d) stackPtr->top == SIZE e) None of the Above

Reason: When top is 0 as previously discussed, the stack is empty. We want to detect this situation here so we don't return a garbage value.

5) What expression should replace /*** Q5 ***/?

a) items[top-1] b) stackPtr->items[top-1] c) items[top]

d) stackPtr->items[stackPtr->top] **e) None of the Above**

Reason: The actual top of the stack is not at index top but one less than it. The proper way to access top is stackPtr->top. Thus, the desired line of code is:

stackPtr->items[stackPtr->top-1]

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 ***/
}
```

6) What line of code should replace /*** Q6 ***/?

a) qPtr = temp; b) qPtr->front = temp; c) qPtr->back = temp;
d) qPtr->front->back = temp; **e) None of the Above**

Reason: We need to attach qPtr->back to tmp. The correct way to do this is qPtr->back->next = tmp;

7) What line of code should replace /*** Q7 ***/?

a) qPtr = temp; b) qPtr->front = temp; **c) qPtr->back = temp;**
d) qPtr->front->back = temp; e) None of the Above

Reason: Now, we just want to move the back pointer to point to the new back of the list, which is temp. c) is the choice that does this.

8) What expression should replace /*** Q8 ***/?

a) temp b) qPtr->front c) qPtr
d) qPtr->back->next e) None of the Above

Reason: When enqueing into an empty queue, the front pointer must be also set to the new node storing the enqueued item, which is temp.

9) What line of code should replace /*** Q9 ***/?

a) return 0; **b) return 1;** c) return qPtr->front->data;
d) return qPtr->back->data e) None of the Above

Reason: If we get here, our code was successful. We are supposed to return 1 in this case.

10) What line of code should replace /*** Q10 ***/?

a) return 0; b) return 1; c) return qPtr->front->data;
d) return qPtr->back->data e) None of the Above

Reason: if we get here, our malloc wasn't successful. Thus, we must return 0 to signal that the push was not successful.

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

5 8 * 9 - 3 - 2 2 + /

a) 4 **b) 7** c) 21 d) 28 e) None of the Above

Reason: We push 5 and 8, then multiply these two (popping them off), then push 40 back onto the stack. Then we push 9, pop off 9 and 40, calculate 40 - 9 = 31, and push this back onto the stack. Then we push 3, pop 3 and 31, calculate 31 - 3 = 28 and push this. Then we push 2, and the other 2, then pop both off, add 2 + 2 = 4 and push 4 onto the stack. Finally, we pop off 4 and 28, divide 28 by 4 to get 7 and push this back to the stack as our final answer.

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

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

a) 4	5	+	/	8	-	2	+	3	*	5	-	4
b) 4	**5**	**+**	**8**	**2**	**3**	**5**	**4**	**-**	*****	**+**	**-**	**/**
c) 4	5	+	8	2	-	3	5	4	*	+	-	/
d) 4	5	+	8	2	3	5	*	-	4	+	-	/

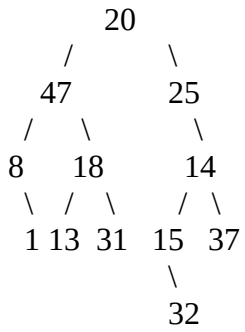
e) None of the Above

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.)

a) enqueue **b) dequeue** c) recurse d) printAll e) None of the Above

Reason: We dequeuer from the front, so we have immediate access and just need to free to front node and move our front pointer. Enqueus would take O(n) time, where n is the number of items in the queue.

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



a) 8, 1, 37, 13, 18, 31, 20, 25, 15, 32, 14, 37
b) 20, 47, 8, 1, 18, 13, 31, 25, 14, 32, 15, 37
c) 20, 47, 8, 1, 18, 13, 31, 25, 14, 15, 32, 37
d) 1, 8, 13, 31, 18, 47, 32, 15, 37, 14, 25, 20
e) None of the Above

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 ***/
}
```

a) return countNodes(root->left) + countNodes(root->right) + 1;
b) return countNodes(root->left) + countNodes(root->right);
c) return max(countNodes(root->left),countNodes(root->right))+1;
d) return max(countNodes(root->left),countNodes(root->right));
e) None of the Above

Reason: We must count 1 for the root, and add this to all the nodes on the left and right.

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

a) O(1) b) O(lg n) **c) O(n)** d) O(nlg n) e) None of the Above

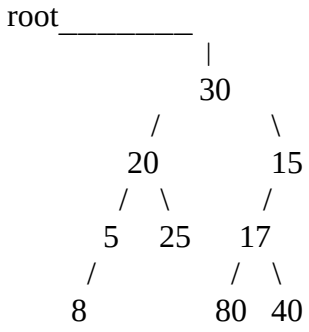
Reason: Our recursive code basically visits each node once and has some constant overhead per recursive call.

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;
}
```



a) 0 b) 150 **c) 153** d) 230 e) None of the Above

Reason: This code adds the values of all leaf nodes in the designated tree. In this tree, the leaf nodes store 5, 25, 80 and 40 for a sum of 153.

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) {

    if (root == NULL) return 0;

    return curDepth + sumDepthRec(root->left, curDepth+1) +
                      sumDepthRec(root->right, curDepth+1);
}
```