Practice Questions for COP 3502 Exam #1 - Solutions

Topics: Dynamic Memory Allocation, Linked Lists, Stacks and Queues

Note: All of these questions are from past exams, so the grading criteria are included so you can see how I chose to assign points in that particular semester.

1) (20 pts) A class has *n* students, s_1 through s_n. Student s_i has taken t_i tests, each scored from 0 to 100. This data is entered via standard input using the following format:

First line stores the number of students, *n*.

The next *n* lines store the student data with the i^{th} of these lines storing the test information for student s_i.

Each of these lines starts with the integer, t_i, the number of tests taken by student s_i. This is followed by the t_i test scores for that student, in order.

Here is a small sample file for 3 students who've taken 5, 2 and 9 tests, respectively:

```
3
5 100 90 95 99 100
2 85 86
9 83 88 85 85 89 96 75 83 95
```

Complete the function below so that reads in this information from standard input and returns an array of arrays (the first array has length t_1, the second array has length t_2, etc.). Note since all of the required information is in the input, no parameters are needed for the function.

```
int** getTestData() {

    int i;                                      // 2 pts all var dec
    scanf("%d", &n);                            // 1 pt
    int** scores = malloc(sizeof(int*)*n);      // 4 pts

    for (i=0; i<n; i++) {                       // 2 pts
        int j, numScores;
        scanf("%d", &numScores);                // 1 pt
        scores[i] = malloc(sizeof(int)*numScores);    // 4 pts
        for (j=0; j<numScores; j++)             // 2 pts
            scanf("%d", &scores[i][j]);         // 2 pts
    }
    return scores;                              // 2 pts
}
```

2) (6 pts) Using the calloc function, write a single line of code to allocate room for `n` variables of type `struct item`. Assume that n is defined as an integer and stores a reasonable positive value and that the type `struct item` is declared. Name the array `myitems`.

`struct item* myitems = calloc(n, sizeof(struct item));`
Grading: 2 pts LHS, 1 pt calloc, 1 pt n, 2 pts sizeof

3) (11 pts) Write a function that takes in pointers to two sorted linked lists, combines them by rearranging links into one sorted linked list, effectively merging the two sorted lists into one, and returns a pointer to the new front of the list. Since no new nodes are being created and no old nodes are being deleted, your code should **NOT** have any mallocs or frees. Also, note that this destroys the old lists. If you try to print either listA or listB after calling merge, the lists are likely to print differently. (***Hint: This is probably much easier to do recursively.***) Fill in the function prototype provided below and use the struct provided below:

Solution

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

node* merge(node* listA, node* listB) {

    if (listA == NULL) return listB;     // 1 pt
    if (listB == NULL) return listA;     // 1 pt

    if (listA->data < listB->data) {              // 1 pt
        listA->next = merge(listA->next, listB); // 3 pts
        return listA;                             // 1 pt
    }
    listB->next = merge(listA, listB->next);      // 3 pts
    return listB;                                 // 1 pt
}
```

4) (5 pts) Complete the blanks in the segment of code below to read in a single integer from standard input into the variable numLevels and then allocate that many pointers to two dimensional arrays. (You will not allocate the space for any of the 2D arrays yet, just for the pointers to those arrays.)

```
int numLevels;

scanf("%d", &numLevels); // Grading: 2 pts

int*** game = malloc(numLevels*sizeof(int**)) ; // Grading: 3 pts
```

5) (13 pts) Complete the blanks in the segment of code below so that it reads in numLevels pairs of integers, r_i and c_i ($0 \le i <$ numLevels), where the $(i+1)^{th}$ pair are the dimensions of the i^{th} level, and dynamically allocates space for each level using the appropriate set of mallocs.

```
int i, j, numRows, numCols;
for (i=0; i<numLevels; i++) {

    scanf("%d%d", &numRows, &numCols);

    game[i] = malloc(numRows*sizeof(int*)) ; // Grading: 5 pts

    for (j=0; j<numRows; j++)                // Grading: 3 pts

        game[i][j] = malloc(numCols*sizeof(int)) ; // 5 pts

}
```

6) (2 pts) If the input to the segments of code above was

```
3
2 10
8 5
3 6
```

how many malloc statements would be executed by the segments of code above?

17 (Grading: 2 pts all or nothing, 1 pt for 13, 14 or 16, 0 otherwise)

7) (10 pts) A "strictly sorted string" is one where the Ascii values of each of the characters in the string are in strictly increasing order. For example, "Back" is a strictly sorted string since 'B' < 'a' < 'c' < 'k'. Write a function that takes in a pointer to the first letnode in a string and returns 1 if the string is strictly sorted, and 0 otherwise. (Note that we can compare the Ascii values of two character variables by simply using the usual relational operators, > , < , >=, <=, == and !=.)

```
int strictlySorted(letnode* word) {

    if (word == NULL) return 1;                    // 2 pts

    while (word->next != NULL) {                   // 2 pts
        if (word->letter >= word->next->letter)    // 3 pts
            return 0;                              // 1 pt
        word = word->next;                         // 1 pt
    }
    return 1;                                      // 1 pt
}
```

8) (5 pts) Write a function that takes in a pointer to the first charnode in a string and returns the length of that string. Please fill in the function prototype shown below:

```
int stringlen(letnode* word) { // code here }
```

```
int stringlen(letnode* word) {

    int res = 0;                    // 1 pt
    while (word == NULL) {          // 1 pt
        res++;                      // 1 pt
        word = word->next;          // 1 pt
    }
    return res;                     // 1 pt
}
```

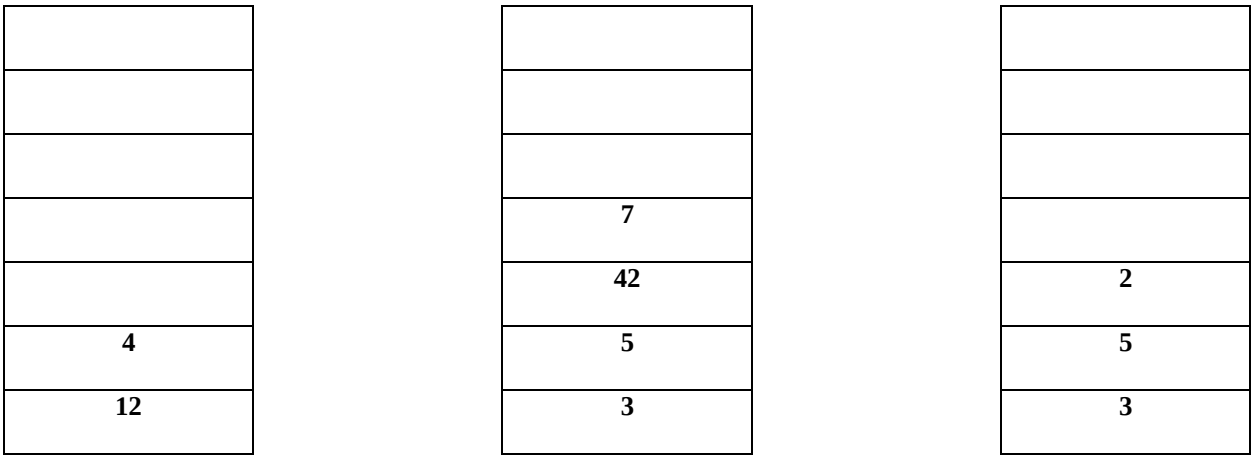
9) (15 pts) Write a function that takes pointers to two letnodes, first and second, returns a negative integer if the string pointed to by first comes before the string pointed to by second, lexicographically, 0 if the two strings are equal, or positive integer if the string pointed to by first comes after the string pointed to by second, lexicographically. (Essentially, implement the strcmp function.)

```
int stringcmp(letnode* first, letnode* second) { // code here}
```

```
int stringcmp(letnode* first, letnode* second) {

    while (first != NULL || second != NULL) {     // 2 pts

        if (first == NULL) return -1;             // 2 pts
        if (second == NULL) return 1;             // 2 pts

        if (first->letter != second->letter)           // 3 pts
            return first->letter - second->letter;     // 3 pts

        first = first->next;                      // 1 pt
        second = second->next;                    // 1 pt
    }

    return 0;                                     // 1 pt
}
```

10) (10 pts) Evaluate the following postfix expression, showing the state of the operand stack at the three points A, B and C indicated below:

				A				B				C		
12	6	2	-	/	5	42	7	/	4	-	*	+		



Value of the Expression: **13**

Grading: 2 pts for each stack, 4 pts for the answer. May give partial if a single error caused a propagation error. (Take off 1 pt per actual error.)

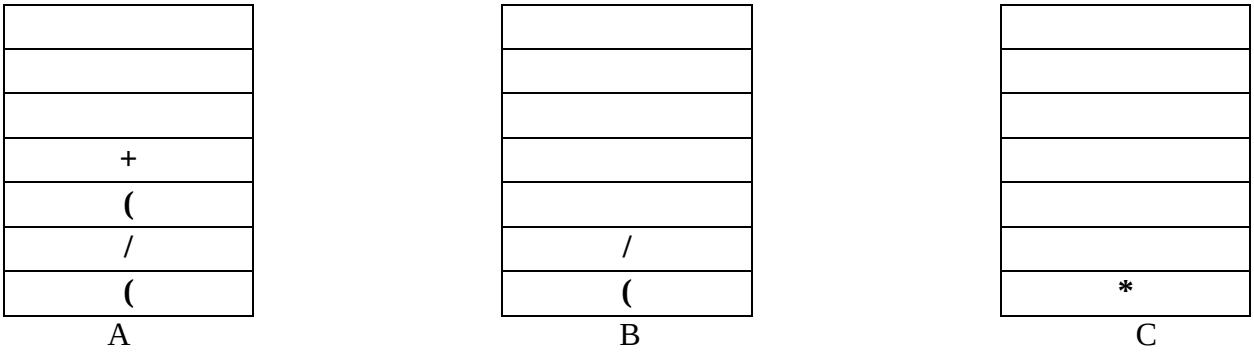
11) (10 pts) Circle either True or False about each of the following assertions about queues.

a) A queue is a Last In, First Out (LIFO) abstract data structure.	True	**False**
b) If a queue is implemented with a regular linked list, with a pointer to the front of the queue only, the enqueue operation would take Θ(n) time for a list with n elements. (Θ indicates proportional to n.)	**True**	False
c) If a queue is implemented with a regular linked list, with a pointer to the front of the queue only, the dequeue operation would take Θ(n) time for a list with n elements.	True	**False**
d) A queue must be implemented with a linked list.	True	**False**
e) A queue allows for access to any of its elements in O(1) time.	True	**False**

Grading: 2 pts each, all or nothing

12) (10 pts) Convert the following infix expression to postfix, showing the state of the operator stack at the three points A, B and C indicated below:

((42 - 9) / (2 + 5 - 2 * 2)[A] - 2 * 3)[B] *[C] (3 + 4)



Equivalent Postfix Expression:

42 9 - 2 5 + 2 2 * - / 2 3 * - 3 4 + *

Grading: 2 pts for each stack, 4 pts for the expression, if a single error propagates, take off 1 pt per error instead