Practice Questions for COP 3502 Exam #1

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

Note: A common complaint of my classes is that past tests don't help prepare students for current tests. I strongly believe in testing for conceptual understanding, so I intentionally make questions that can be solved with similar tools but look different than past questions. I believe that students sometimes use the past exams I post in a poor manner. I believe they try to "memorize" the mold of the question without necessarily trying to understand the underlying concepts. I post the past questions not for students to try to memorize question molds (and I understand that this works extremely well in other classes), but to see some applications of concepts we've learned in class and past ways in which I've tested for those concepts. I typically make sure on an exam not to ask something that is an exact mold of an example that I did in class, since my goal is precisely to distinguish between those who have learned by memorizing molds and those who have learned how to apply a general concept. So, when studying, please keep this in mind. Please try to focus on the concepts and not the specific answers. Too often, on an exam, I see a carbon copy of a solution to a past exam question which is completely irrelevant to the actual question I posed on the exam. Also, in this review, I've given way too many questions to cover in 50 minutes. That is intentional. I'd rather provide you more materials than too few. The intent here is for your TA to maybe go over 3 of these, and leave the rest for you as potential practice. Solutions for this will be posted after the recitation.

1) 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 ith 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() {//Fill in answer.}
```

2) 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`.

---

3) 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. Fill in the function prototype provided below and use the struct provided below:

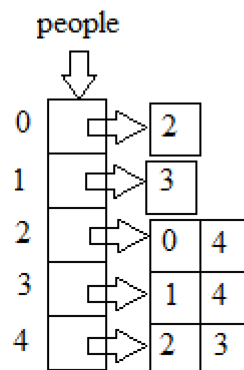
```
typedef struct node {
    int data;
    struct node* next;
} node;

node* merge(node* listA, node* listB) { // fill in code }
```

4) An efficient way to store data about the relationships amongst a group of n people is as follows: give each person a unique label from 0 to n-1. For each person, create a dynamically sized array, the size of the number of acquaintances that person has. For example, if person 0 has 3 acquaintances, then their array would be size three and each item in this array would be the number of one of their acquaintances. Since the people are labeled 0 to n-1, we can store each person's array, in an array itself! Consider an example with 5 people, where we have the following acquaintance pairs:

(0, 2), (1, 3), (2, 4), (3, 4)

We would store this in an array of size 5, where each item is an array as follows:



Write a function that takes in a file pointer to a file that stores data about a set of acquaintances, dynamically allocates the array of arrays described above and returns a pointer to that array of arrays. The file format is as follows:

The first line contains a single positive integer n, the number of people described in the file. The following n lines contain information about the acquaintances of each of person 0 through person n-1. Each of these lines starts with a positive integer n_i, the number of acquaintances of person i. The following n_i values are the numbers of the people that person i is acquainted with, in increasing order. It's guaranteed that the data is consistent. If person i is acquainted with person j, then person j will be acquainted with person i. The input file storing the data shown in the previous picture is as follows:

```
5
1 2
1 3
2 0 4
2 1 4
2 2 3
```

Please fill in the function prototype given on the next page. (Note: To actually implement this in a meaningful way, we'd have to store the value of n and the sizes of each of the arrays so that we could properly iterate through the structure. I haven't asked you to do these tasks in order to make the solution shorter.)

```
int** readgraph(FILE* ifp) { // fill in code }
```

<u>Questions 4, 5, 6</u>

Consider a video game with multiple levels, starting at 0, where each level can be represented with a two-dimensional array of integers. For the following questions you will read in information about the number of levels and the dimensions of each of the levels and allocate the appropriate amount of space to store the information.

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;

_____________________________ ;

int*** game = ______________________________________________________ ;
```

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 ith 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);

    _________ = ______________________________________ ;

    _____________________________________________

        ______________________________________________

}
```

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?

____

Questions 7, 8, 9

Consider using a linked list to store a string. Each node in the list would store a character and a pointer to the node storing the next character. The pointer for the node storing the last character in the string would point to NULL. Each of the following questions will be based on this premise using the struct shown below:

```
typdef struct letnode {
    char letter;
    struct letnode* next;
} letnode;
```

7) 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) { // fill in code }
```

8) 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 }
```

9) 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}
```

10) 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	-		*	+

A	B	C

Value of the Expression: _________________

11) 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 $\Theta(n)$ 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 $\Theta(n)$ 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

12) 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 + [A] 5 - 2 * 2) [B] - 2 * 3) * [C] (3 + 4)

A	B	C

Equivalent Postfix Expression:

____