CIS*2520 Mock Exam (Summer 2017)

All questions are available on Courselink via
`https://courselink.uoguelph.ca/d2l/le/content/470592/Home`.

Unit 1 Lists, Stacks and Queues

1. Define the following with respect to software design:

   a. Modularity

   b. Reuse

   c. Maintainability

   d. Coupling

   e. Cohesion

2. Define Test-Driven Development.

3. List ten different programming languages as well as their level of abstraction (Moderate, High, Very High).

   1. _____
   2. _____
   3. _____
   4. _____
   5. _____
   6. _____
   7. _____
   8. _____
   9. _____
   10. _____

4. Write pseudocode for a multiply and a subtract operation for the fraction ADT. What does the integer portion of the Fraction struct represent? List any additional operation(s) that would be required in order to make use of that part of the struct. (Fraction ADT available on Courselink)

5. Write the following three algorithms:

    1. The algorithm for the addToLocation() operation for the List ADT. This operation should add an element to the list at a specific location in the list (identified by a number). Use the same specification format as has been used to describe operations in this lesson. Include all necessary parameters and return values in the signature of your specification. Be sure to include preconditions and postconditions.

    2. The algorithm to delete a node from the nth position of a linked list. The operation should return the deleted data and ensure that the remaining elements of the list are properly connected.

    3. The algorithm for inserting a node in sorted order given an array implementation of a list. The algorithm should take the data as a parameter.

6. Would a double linked list or a single linked list be a better choice for encapsulation in a Stack ADT? Justify your opinion.

7. Write the algorithms for push() and pop() given an array implementation of a stack. Show the stack struct definition as well.

8. Create a design or prototype of a reverse polish calculator program. You should limit operations to + - * and /. Use a stack ADT in your design.

9. What is the Computation complexity of the enqueue() and dequeue() operations expressed in Big O notation?

10. What additional information must be kept track of in order to use a conventional array as a circular buffer? Give the Queue struct that you would use if you were writing a circular buffer queue implementation.

11. Write the algorithm for dequeuing an item from a circular buffer queue.

12. Which of the following statements about queues is untrue?

    a. Queues can have elements inserted at any position in the data structure.
    b. The first element inserted into a queue will be the first element taken out of the queue.
    c. Queues can be found in the real world.
    d. The size of a queue data structure is bounded only by the size of the computer memory.
    e. A queue is somewhat similar to a stack.

13. Which List operation would be most likely to be the one encapsulated if you were writing a remove operation for a queue?

    a. addHead(elementToBeAdded)
    b. length()
    c. removeBack()
    d. removeHead()
    e. insert(position, elementToBeAdded)

14. Given the following queue: A B b E r S T, where A is the front of the queue, what will the queue content be after two remove operations?

    a. A B b E r S T
    b. b E r S T
    c. A B b E r S
    d. T A B b E r
    e. The queue will be empty.

15. Given the following queue: t y 5 8 i 2 d t e, what should a call to length() return?

    a. 9

    b. 8

    c. 10

    d. It will generate an error because of the mixed data types.

    e. 3

Unit 2 Hash Table and Binary Trees

1. If you were implementing a student record system for a small music school with fewer than 200 students, and were using associative arrays as the data structure, how would you implement your associative array? Would you make a different choice if your student record system was for a university?

2. Write a 'to-do list' of the things you would need to do (including research and learning) in order to implement the borrowers' library example. Try to make the list very specific. Use the to-do list to estimate the amount of time it would take you to implement and test the application (round up to the nearest hour).

3. Using the division method, calculate hash values for the following set of keys:

$$\{54, 77, 82, 13, 991, 308, 68, 45, 1001, 73\}$$

   Calculate once using a table size of 11. Calculate a second time using a table size of 12. Do you notice anything about the distributions of the calculated values?

4. The expected search time for Linear probing can be calculated by the following formula:

   i. $O(1 + \frac{1}{1-loadFactor})$ for a successful search.

   ii. $O(1 + \frac{1}{(1-loadFactor)^2})$ for an unsuccessful search.

   Create a chart showing the expected search times (successful and unsuccessful) for the following load factors: .10, .20., .30, .40, .50, .60, .70, .80, .85, .90, .95.
   What do these numbers tell you?

5. Suppose you have two hash functions H1 and H2 where H1(87) = 10, H2(87) = 3 and H1(42)=10, H2(42)=7. Further suppose that the key 87 is inserted into the table first and then the key of 42. Show the sequence of table positions tried when using random probing with a constant of 37 and a table size of 11.

6. The expected time for a search using Double hashing can be calculated by the following formula:

   i. $O(\frac{1}{loadFactor} \times (1 + ln(\frac{1}{1-loadFactr})))$ for a successful search.

   ii. $O(\frac{1}{1-loadFactor})$ for an unsuccessful search.

   Create a chart showing the expected search times (successful and unsuccessful) for the following load factors: .10, .20., .30, .40, .50, .60, .70, .80, .85, .90, .95.
   What do these numbers tell you?

7. The loadFactor of a hash table using separate chaining to resolve collisions is calculated as the number of **keys / the number of chains**. The expected length of search, successful or unsuccessful, of Separate Chaining is $O(1 + loadFactor)$.

   a. Calculate a chart of the predicted search times for the following load factors: .10, .20., .30, .40, .50, .60, .70, .80, .85, .90, .95.

   b. What do these numbers tell you?

c. What factors would you consider when selecting a collision resolution approach for a hash table?

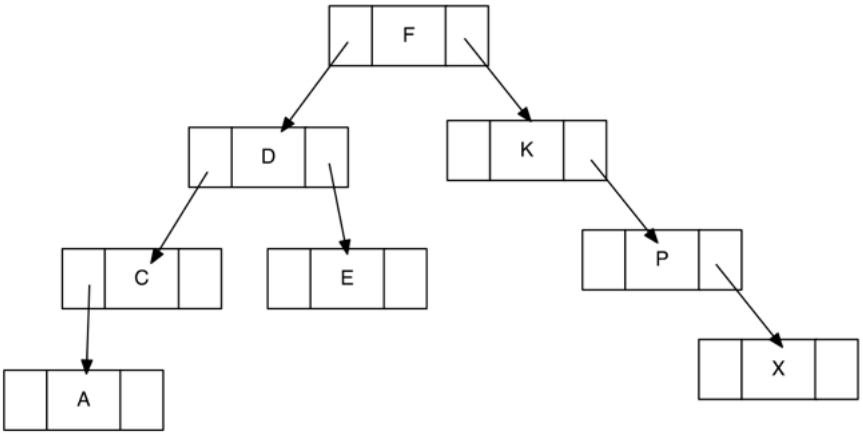
8. The order that nodes are inserted into a tree can greatly affect the structure of the tree. Draw the binary tree that is constructed by adding the seven names in the following order:

    1. Sleepy
    2. Bashful
    3. Doc
    4. Dopey
    5. Sneezy
    6. Happy
    7. Grumpy

9. Trace the algorithms for level-order traversal using a hand drawn tree. What order are the nodes of the tree printed in? Can you tweak the algorithm to reverse the order?

10. One of the most frequent uses of a binary tree is as a search tree. The object is to search through the data to find out if a particular element is part of the data. Write a recursive find procedure for a binary tree. Start with the root node of the tree. Your procedure can be in *C* or in pseudocode. Be sure to pass in the compare function pointer.

11. Give the in-order traversal and the pre-order traversal of the tree shown below.



12. Write the pseudocode or C code for a compare operator. A compare operator must return three distinct values, one for when the first parameter is larger, one for when the second parameter is larger, and one for when the two parameters are equal. The convention is to use 1, 0, and -1 for the three values. Zero for equal values, 1 for the first string being larger, -1 for the second.

13. The delete operation relies on a recursive function to find the minimum value in a subtree. The minimum value will always be the leftmost leaf of the subtree. Write the findMinimum function (in pseudocode or in C).

Unit 3 Heaps and Priority Queues

Unit 4 Balanced Trees and Sorting