The following is a collection of practice problems for the CS165: Data Structures and Applications second midterm. The questions are similar to what they would be on the exam, and contain a combination of free response and written code questions. It is recommended that you attempt to do the questions on your own, without using any outside resources. Some of the practice problems are designed to be trickier than those on the exam in order to cover some of the common mistakes students make. At the end of the document is an answer key, with explanations for some of the trickier answers. Answers for some problems are only as suggested, as there are many correct answers.
DATA STRUCTURES REVIEW

For problems 1-4, show what the program shown below would print.

**HINT:** Draw a picture of the queue and update the picture as it changes. As a reminder, Queue.offer(e) and Queue.add() inserts to a queue, Queue.remove() and Queue.poll() removes from a queue, and Queue.element() or Queue.peek() reads the queue without modifying it. Queues are first-in first-out (FIFO) data structures.

```java
public static void main(String[] args) {
    Queue<String> queue = new LinkedList<>();
    queue.add("C++");
    queue.add("Java");
    queue.add("C");
    queue.add("Python");
    queue.remove();
    System.out.println(queue.element()); // Question 1
    queue.offer("Java");
    queue.offer("C++");
    queue.remove();
    System.out.println(queue.peek()); // Question 2
    queue.poll();
    System.out.println(queue.peek()); // Question 3
    queue.offer("Fortran");
    queue.offer("C");
    System.out.println(queue); // Question 4
}
```

1. ___________________________
2. ___________________________
3. ___________________________
4. ___________________________
COLLECTIONS HIERARCHY

For each of the following blocks of code, give what the program would print. If an error would occur, write “error”.

Stack<Integer> stack = new Stack<>();
stack.push(5);
stack.push(8);
stack.peek();
stack.pop();
System.out.print(stack.isEmpty());

5. _________________________

List<Integer> lList = new LinkedList<>();
lList.add(3);
lList.add(4);
lList.add(5);
lList.add(6);
lList.remove(3);
lList.remove(4);
System.out.print(lList.size());

6. _________________________

Queue<Integer> queue = new Queue<>();
queue.add(0);
queue.offer(1);
queue.add(2);
queue.poll();
System.out.println(queue.element());

7. _________________________

List<Integer> array = new ArrayList<>();
array.add(1);
array.add(2);
array.add(3);
array.remove(2);
System.out.println(array.contains(2));

8. _________________________

REGULAR EXPRESSIONS

Follow the instructions below to write or interpret a regular expression. In regular expressions, [0-9] means any digit, [A-Za-z] means any letter, ? means 0 or 1 occurrences, + means 1 or more occurrences, * means 0 or more occurrences, {2,4} means between 2 and 4 occurrences, {3} means exactly 3 occurrences, . matches any character, and \. matches a period, and parentheses just group items.

9. Write the regular expression for an account number that start with the letter 'C', followed by exactly 6 digits from the set '0' to '8', followed by a dash '-', followed by 1 or more uppercase letters, and ending with a semicolon ';'.

CS165, Practice Midterm Problems, Fall 2018
10. Write the regular expression for a time string, that starts with the hour (2 digits), followed by a colon ':', followed by the minute (2 digits), optionally followed by a colon ':', and milliseconds (3 digits). The string must always finish with "am" or "pm". The first digit of the hours must be 0 or 1, and the first digit of minutes must be in the range 0..5, and the second digits of hours and minutes are in the range 0..9. For example, **10:59am** or **09:15pm**, or **04:20:347pm**.

11. List three strings that follow this regular expression: `[0-9].[a-zA-Z]\{2-4}\bak`

_________________________
_________________________
_________________________

**GRAMMARS AND PRODUCTION RULES**

12. Complete the production rules for an assignment statement for a simple language where the variables are groups of one or more letters (uppercase or lowercase), followed by an equals sign '='; followed by a literal integer, which is 1 or more digits, followed by a semicolon ';'. Do not worry about white space. For example, **xyz = 1234**; or **onlyLetters = 12345678**;.

```
<assignment> ::= 
<variable> ::= 
<literalInteger> ::= 
<letter> ::= 
<lowerCaseLetter> ::= a | b | ... | z 
<upperCaseLetter> ::= A | B | ... | Z 
<digit> ::= 0 | 1 | ... | 9
```
13. Given the following production rules, give three examples of strings that are legal in the grammar defined by the rules.

\[
\text{<something>} ::= \text{A <digit>^* B <punctuation>^?} \\
\text{<digit>} ::= 0 \mid 1 \mid \ldots \mid 9 \\
\text{<punctuation>} ::= \% \mid & \mid \# \mid @
\]

Which of the following strings are valid for these production rules? (write true or false)

\[
\text{<something>} ::= \text{<lowerCaseLetter>} \mid \\
\text{<digit> <something> <digit>} \\
\text{<lowerCaseLetter>} ::= a \mid b \mid \ldots \mid z \\
\text{<digit>} ::= 0 \mid 1 \mid \ldots \mid 9
\]

14. g ______________ 18. 87k4k78 ____________
15. 1a1 ____________ 19. 123a321 ____________
16. 012b10 __________ 20. 1234321 ____________
17. 43s21 ____________ 21. 888q999 ____________
Heap Manipulation

The following binary tree satisfies the heap property:

```
  17
 /   \
15   16
/   / \
10  13 12
/ \  / \
 3  9  2
```

Assuming each question starts from the tree above, i.e. the operations are not cumulative, answer the following questions while maintaining the heap property:

22. If we add “18” to the tree, which node will be at the root?
23. If we remove “17”, what node will we replace it with before swapping?
24. If we remove “17”, how many times will we need to swap down?
25. If we remove “17”, what will be the new root node when we’re done?
26. If we add any new node to our tree, the first step is to make it the child of which node?
27. If we add “14” to the tree, how many times will we need to swap up?
28. If we add any node, what is the maximum number of times we will need to swap up?
EXPRESSION TREES

The expression tree shown below is correct for the following expression:

\[(12 \times (19 + 1) \mod 12 - 2 \times (12 \mod 5)) - 29.\]

29. Show the postfix expression represented by the tree, with spaces between each token, and no leading or trailing spaces.

30. Show the prefix expression represented by the tree, with spaces between each token, and no leading or trailing spaces.

31. What does the expression evaluate to, assuming integer math and the normal Java order of operations, which are of course reflected in the prefix and postfix forms and the tree?
The BST shown below is about to have some nodes deleted. Please answer questions 15-24 below.

Starting with the BST tree shown above, if we wish to delete node AA:

32. Which node will need to be reconnected?  
33. Which node will it be reconnected to?  
34. Which side of the node will it be reconnected to (LEFT, RIGHT)?

Starting with the BST tree shown above, if we wish to delete node DD:

35. Which node will need to be replaced?  
36. Which node will move to replace it?  
37. Which node will need to be reconnected?  
38. Which node will it be reconnected to?  
39. Which side of the node will it be reconnected to (LEFT, RIGHT)?
The following BST will undergo some operations:

```
      50
     /   \
   30    70
  / |  / \
10 40 60 80
 / | |    |   |
20 45 55  
```

Assuming all the operations start from the tree above, i.e. they are not cumulative, answer the following questions. Some may have multiple correct answers.

40. Is “20” a left or right child of “10”? ____________

41. Is “55” a left or right child of “60”? ____________

42. If we add “65”, what node will it become a child of? ____________

43. If we add “25”, what node will it become a child of? ____________

44. If we remove “10”, what node can we replace it with? ____________

45. If we remove “70”, what node can we replace it with? ____________

46. If we remove “30”, what node can we replace it with? ____________

47. If we remove “50”, what node can we replace it with? ____________
JAVA CODING

For these problems, you can write them in Eclipse and test your code. However, you will not have an IDE available during the exam and will not be able to run your code. As a result, I would suggest attempting to answer these without an IDE first. Minor syntax errors will not reduce your grade.

48. Write code to insert an element in a BST by descending from the root to where the node should be found. Return false if the element is already in the tree, otherwise insert the element into the correct place and return true. Recursion is not needed. There is an inner class Node has the instance variables element, left, and right, and the element has a compareTo method. HINT: You must keep track of the current node and its parent.

```java
public boolean insert(E e) {
    if (root == null)
        root = new Node(e); // Create a new root
    else {
        // Locate the parent node
        TreeNode<E> parent = null;
        TreeNode<E> current = root;
        while (current != null) {
            if (current.element.compareTo(e) > 0)
                parent = current;
            current = current.left;
        }
        if (parent == null)
            current = new Node(e);
        else
            current = new Node(e, parent, null);
    }
    size++;
    return true; // Element inserted successfully
}
```

// Create the new node and attach it to the parent node
Write code that will insert an element into a linked list implementation of a queue structure, where the head of the list is the first element added and the tail of the list is the most recently added element. Remember to handle edge cases.

```java
class Node { // Node class
    Node next, prev;
    E element;

    Node(E element) { this.element = element; }
}

public boolean offer(E e) {
    // Your code here
    size++;
    return true;
}
```
**Warning:** The following page contains the answer key. Only check the answer after you have attempted and are confident with your answer. If you are struggling, it is recommended to check your notes or the textbook before looking at the answer key.

1. Java
2. C
3. Python
4. [Python, Java, C++, Fortran, C]
5. false
6. error – The remove() method has two signatures- remove(int index) and remove(Object o). If we pass an int to remove, it will remove the item at that index. When we do remove(3), it actually removes index 3, which is 6. When we try to remove index 4, we get an IndexOutOfBoundsException. To remove a specified element from a list of integers, rather than at an index, do remove(new Integer(x)).
7. error – Queue is an interface in Java, so we can’t instantiate it. I would recommend checking out the Java Collections Framework graph in Liang Chapter 20 (you can also find the chart in the slides on the CS165 website).
8. true – Again, remove(int index) will remove the index at 3, which won’t give us an error, but means that contains(2) will return true.

9. C[0-8][6]-[A-Z]*;
10. [01][0-9]:[0-5][0-9]:[0-9]:[3]? (am|pm)
11. 0%ABcd.bak 1-Ab.bak 6.AbC.bak These are just a few examples.
12. 

<assignment> ::= <variable> = <literalInteger>;
<variable> ::= <letter>*
<literalInteger> ::= <digit>*

<letter> ::= <upperCaseLetter> | <lowerCaseLetter>
<lowerCaseLetter> ::= a | b | ... | z
<upperCaseLetter> ::= A | B | ... | Z
<digit> ::= 0 | 1 | ... | 9
13. AB& A5B A122837827B% Again, these are just a few examples.

14 – 21: T, T, F, T, F, T, F, T The production rules describe a string in which the center character is a lowercase letter, and recursively defines itself to be a balanced number of digits on both sides of the letters. The digits do not need to be the same.

22. 18
23. 9
24. 2
25. 16
26. 13
27. 1
28. 3

29. 12 19 1 + * 12 % 2 12 5 % * –

30. – % * 12 + 19 1 12 * 2 % 12 5

31. – 4

32. CC
33. DD
34. Left
35. DD
36. CC
37. BB
38. AA
39. Right

40. Right
41. Left
42. 60
43. 20
44. 20
45. 60 or 80
46. 20 or 40
47. 45 or 55

48.
public boolean insert(E e) {
    if (root == null)
        root = new Node(e); // Create a new root
    else {
        // Locate the parent node
        TreeNode<E> parent = null;
        TreeNode<E> current = root;
        while (current != null)
            if (e.compareTo(current.element) < 0) {
                parent = current;
                current = current.left;
            } else if (e.compareTo(current.element) > 0) {
                parent = current;
                current = current.right;
            } else
                return false; // Duplicate node not inserted

        // Create the new node and attach it to the parent node
        if (e.compareTo(parent.element) < 0)
            parent.left = new Node(e);
        else
            parent.right = new Node(e);
    }
    size++;
    return true; // Element inserted successfully
}
49.

```java
public boolean add(E e) {
    Node newNode = new Node(e); // Create a new node

    if (head == null) { // Edge Case: Queue is empty
        head = newNode;
        tail = newNode;
    } else {
        newNode.prev = tail; // Since the tail is most recently added
        tail.next = newNode.prev;
        tail = newNode;
    }

    size++;
    return true;
}
```