CS165
Practice Midterm-2 Problems
Spring 2020

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. 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 that students make. Moreover, in some cases we have provided more than one question of a certain type while the exam may contain just one. At the end of the document is an answer key, with explanations for some of the trickier 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.pol() 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.

```
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());
System.out.print(stack);
```

5. _____

6. _____

```
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());
System.out.print(lList);
```

7. _____

8. _____

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

9. _____

10. _____

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

11. _____

12. _____

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.

13. Write the regular expression for an account number that starts 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 ';'.

In the exam, this type of question will appear as a multiple-choice question.

_____

14. 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**.

In the exam, this type of question will appear as a multiple-choice question.

_____

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

In the exam, this type of question will appear as a multiple-choice question.

_____

_____

_____

16. Which of the following strings does not follow this regular expression: **[0-9].[a-zA-Z]$^{\{2-4\}}$\.bak**

- ◊ file.2.bak
- ◊ 2.file.bak
- ◊ 9.ab.bak

GRAMMARS AND PRODUCTION RULES

17. You are given 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.

<assignment> ::= <variable> = <literalInteger>;

<variable> ::= <letter>$^{+}$

<literalInteger> ::= <digit>$^{+}$

<letter> ::= <upperCaseLetter> | <lowerCaseLetter>

<lowerCaseLetter> ::= a | b | ... | z

<upperCaseLetter> ::= A | B | ... | Z

<digit> ::= 0 | 1 | ... | 9

Which ones are legal and which ones are not?

A. xyz = 1234;
B. int onlyLetters = 12345678;
C. v1 = v2;
D. v3 = 21;

18. Given the following production rules, give three examples of strings that are legal in the grammar defined by the rules.

<something> ::= A <digit>* B <punctuation>$^{?}$

<digit> ::= 0 | 1 | ... | 9

<punctuation> ::= % | & | # | @

_____

_____

_____

In the exam, this type of question will appear as a multiple-choice question.

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

```
<something> ::= <lowerCaseLetter> |
                <digit> <something> <digit>
<lowerCaseLetter> ::= a | b | ... | z
<digit>     ::= 0 | 1 | ... | 9
```

19. g _____

20. 1a1 _____

21. 012b10 _____

22. 43s21 _____

23. 87k4k78 _____

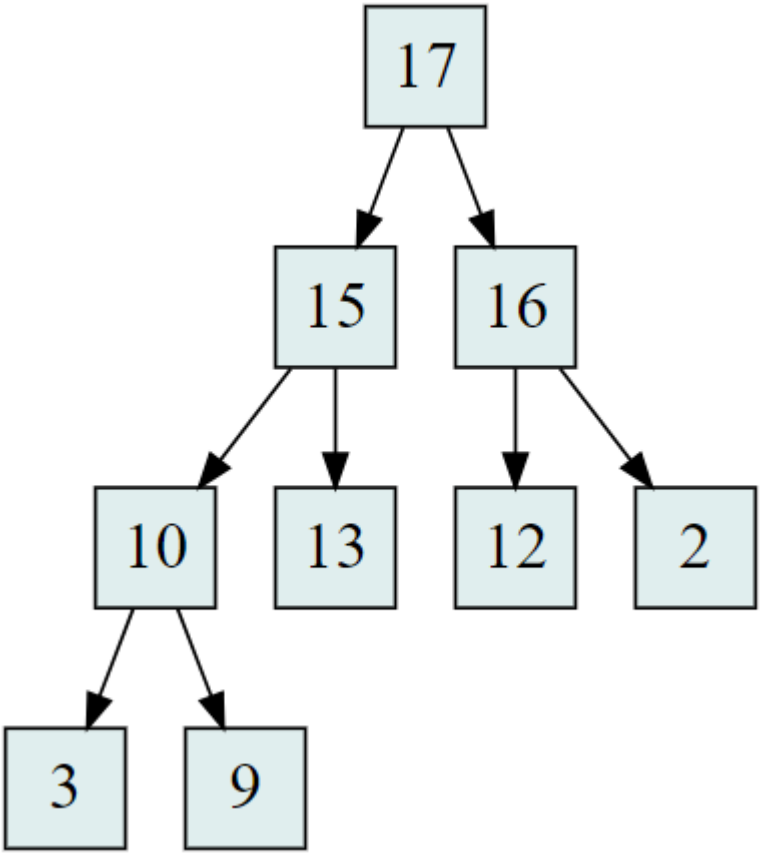
24. 123a321 _____

25. 1234321 _____

26. 888q999 _____

Heap Manipulation

The following binary tree satisfies the heap property: -



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

27. If we add “18” to the tree, which node will be at the root?

28. If we remove “17”, what node will we replace it with before swapping?

29. If we remove “17”, how many times will we need to swap down?

30. If we remove “17”, what will be the new root node when we’re done?

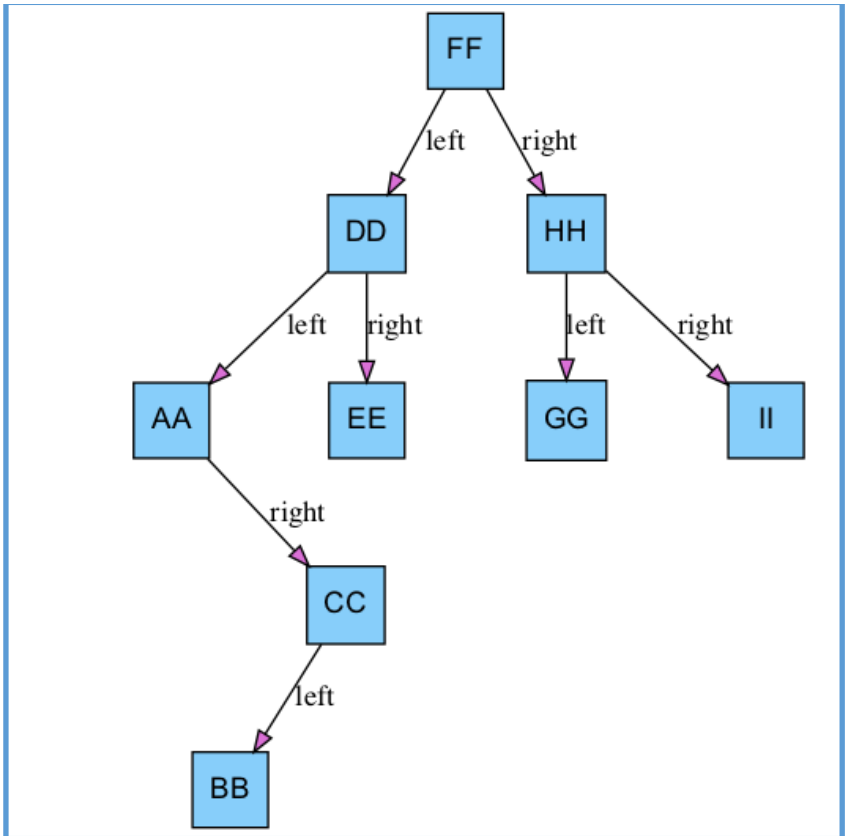
31. If we add any new node to our tree, the first step is to make it the child of which node?

32. If we add “14” to the tree, how many times will we need to swap up?

33. If we add any node, what is the maximum number of times we will need to swap up?

BST MANIPULATION

The BST shown below is about to have some nodes deleted.



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

34. Which node will need to be reconnected? _____

35. Which node will it be reconnected to? _____

36. 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 replacing it with a value from the left subtree:

37. Which node will need to be replaced? _____

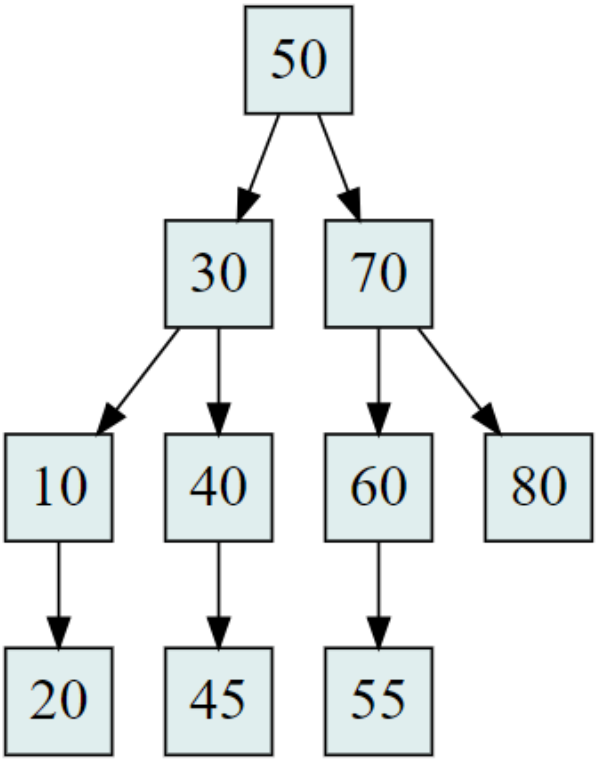
38. Which node will move to replace it? _____

39. Which node will need to be reconnected? _____

40. Which node will it be reconnected to? _____

41. Which side of the node will it be reconnected to (LEFT, RIGHT)? _____

The following BST will undergo some operations:



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.

42. Is “20” a left or right child of “10”? _____

43. Is “55” a left or right child of “60”? _____

44. If we add “65”, what node will it become a child of _____

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

46. If we remove “10”, what node can we replace it with? _____

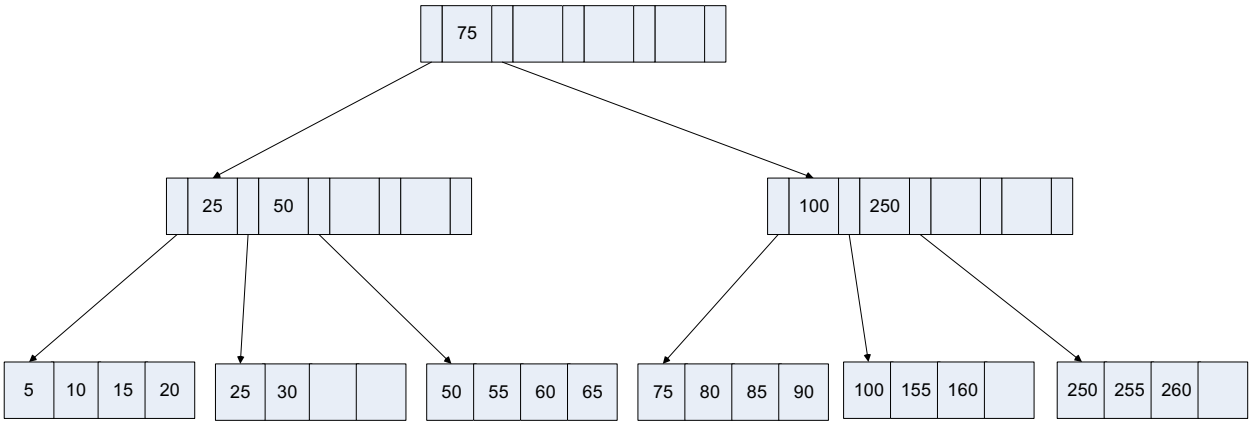
47. If we remove “70”, what node can we replace it with? _____

48. If we remove “30”, what node can we replace it with? _____

49. If we remove “50”, what node can we replace it with? _____

B+ trees

The following B+ tree will undergo some changes:



50. Is the root node an index node or a leaf node? _____

51. Is that always the case? _____

52. What value is the left-most value in the root node? _____

53. If I add the key 31 to the tree, will it split the index node?_____

54. If I add the key 31 to the tree, will it split the leaf node? _____

55. If I add 57 to the original tree, will it split the index node? _____

56. If I add 57 to the original tree, will it split the leaf node? _____

57. How many index nodes are there in the tree? _____

58. If I delete 30 from the tree, can I borrow from the left sibling ? _____

59. If I delete 30 from the tree, can I borrow from the right sibling? _____

60. How does the root node change if I add 82 to the original tree? _____

Prefix, Infix, and Post-fix expressions

What type of expressions are the following? Convert each expression to the other two types.

61. **p / (q + r) * s – t**

62. **p q + r s t / - ***

63. **- + * p q / r s t**

Evaluate the following post-fix expressions using Java's integer arithmetic.

64. **11 4 % 3 16 9 / * -**

65. **12 9 4 – 3 - / 4 7 % ***

Iterator and Iterable interfaces

Given the following declaration:

```
public class MyCollection implements Iterable<Shape>
```

The Shape class has a method `getArea()`.

A variable that has been declared and appropriately initialized:

```
MyCollection figures
```

66. **Write the Java statement that gets an iterator object from figures.**

67. **Write a loop using the iterator object obtained in Q66 to print the individual areas of each shape stored in figures.**

68. **Write a foreach-loop using the Iterable object figures to print the individual areas of each shape stored in figures.**

69. Conceptual questions.

These questions are listed below to make you think carefully. Please refer to the course materials if you don't know the answer. In the midterm, we will ask questions related to these topics in the form of True/False, Multiple-choice, Multiple-answers, or fill in the blanks.

a. What benefit do data structures provide to a programmer?

b. What are the differences between List, Set, and Map in the collection hierarchy?

c. For what operations/situations are data structures such as ArrayLists, Stacks, and Queues more efficient than arrays? For what operations/situations are they less efficient?

d. What is the advantage of using a Heap? What operations are more efficient compared to other structures?

e. What are the properties of a Heap?

f. What is meant by pre-order, in-order, post-order traversal, and level-order traversal of a heap?

g. Which traversal on a heap produces the elements in sorted order?

h. For the heap given on page 7, write the four traversals. Which one is like the array that contains the heap?

i. What is the advantage of using a Binary Search Tree? What operations are more efficient in a BST than in say, ArrayList, LinkedList?

j. What are the properties of a Binary Search Tree?

k. What is meant by pre-order, in-order, post-order traversal, and level-order traversal of a Binary Search Tree?

l. Which traversal on a Binary Search Tree produces the elements in sorted order?

m. For the Binary Search Tree given on page 8, write the four traversals.

n. What are the differences between a B+ Tree and a Binary Search Tree? What are the advantages of using a B+ Tree over a Binary Search Tree?

o. Why do computer scientists use grammars and regular expressions?

p. If a class A implements the Comparable<E> interface, what method does a need to implement? What is the signature of this method? What does this method let you do?

q. If a class A implements the Iterable<E> interface, what method does A need to implement? What does this method let you do? How do you write a for-each loop to process the elements in A?

r. If a class A implements the Iterator<E> interface, what methods does A need to implement? What do these methods let you do? How do you write a loop to process the elements in A?

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. **[5]**
7. **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)).
8. **Cannot print the list because of above error.**
9. **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).
10. **Cannot print the list because of above error.** But be sure to know what the methods do in case we used a correct instantiation of a class that implements Queue.
11. **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.
12. **[1,2]**
13. **C[0-8]$^{\{6\}}$-[A-Z]$^{+}$;**
14. **[01][0-9]:[0-5][0-9](:[0-9]{3})?(am|pm)**
15. **0%ABcd.bak 1-Ab.bak 6.AbC.bak** These are just a few examples.
16. **file.2.bak**
17. **A is legal. Rest illegal.**
18. **AB& A5B A122837827B%** Again, these are just a few examples.

19 – 26 : 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 digits on both sides of the letters. The digits do not need to be the same.

27. 18
28. 9
29.2
30.16
31.13
32.1
33.3

34.CC
35.DD
36.Left
37.DD
38.CC
39.BB
40.AA
41.Right

42.Right
43.Left
44.60
45.20
46.20
47.60 or **80**
48.20 or **40**
49.45 or **55**

50.Index Node
51.No, it can start as a leaf node
52.75
53.No
54.No
55.No
56.Yes
57.3
58.Yes
59.Yes
60.It doesn't change.

61. p / (q + r) * s – t

This is an infix expression.
Prefix expression: - * / p + q r s t
Postfix expression: p q r + / s * t –

62. p q + r s t / – *

This is a postfix expression.
Infix expression: (p + q) * (r – s / t)
Prefix expression: * + p q – r / s t

63. - + * p q / r s t

This is a prefix expression.
Infix expression: (p * q) + (r / s) – t
Postfix expression: p q * r s / + t –

64. 0

65. 24

66.
```
Iterator<Shape> shapesIterator = figures.iterator();
```

67.
```
while (shapesIterator.hasNext()) {
 System.out.println(shapesIterator.next().getArea());
}
```

68.
```
for (Shape s : figures) {
 System.out.println(s.getArea());
}
```