CS165
Practice Final Exam Questions

A full list of topics for the Final is available as a separate PDF document linked from the Course Schedule page. The final exam is ***comprehensive***. Thus, when you review the material, you must include the topics that were covered before Midterm 1, between Midterm 1 and Midterm 2, and after Midterm 2.

For your convenience, we have included the practice materials from the three parts in **a single** document. These items are in **reverse order** (i.e., most recent first).

PART I

This part of the practice problem set corresponds to Weeks 13-15.

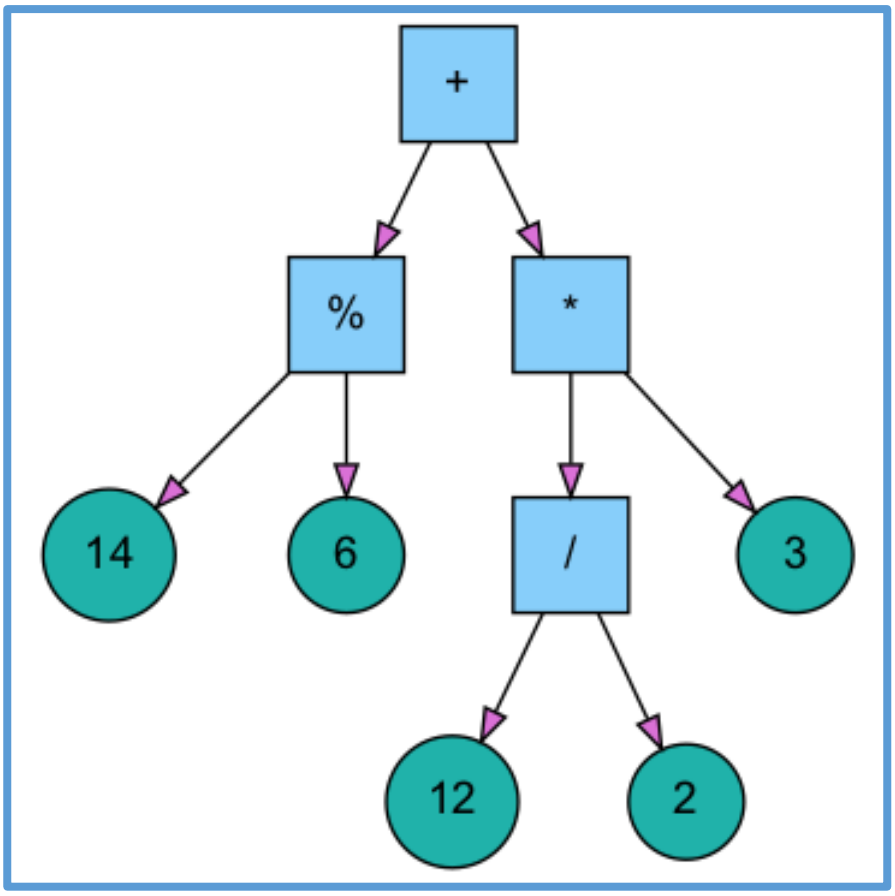
Hashing

Graphs

Expression Trees

EXPRESSION TREES – 1

Using the expression tree shown below, answer the following questions:



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

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

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

4. 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? Do this using the tree, and verify the answer using the infix expression.

EXPRESSION TREES – 2

Given the infix expression (29 – 3) * 4 / 6 + 68 % (3 + 10), answer the following questions.

5. Create an expression tree that represents this expression.

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

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

8 . 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? Do this using the tree, and verify the answer using the infix expression.

HASHING

9. With **linear probing**, in which entry in the left hash table will the colliding object with key **15** be stored? Which entry would be used for **quadratic probing**? The table size is **13**, and the hash function is a simple modulo of the integer key. Also list the indices that are tried unsuccessfully.

LINEAR PROBING:

QUADRATIC PROBING:

Index:	Key:
0	39
1	53
2	67
3	81
4	95
5	18
6	
7	33
8	47
9	61
10	
11	
12	25

10. Suppose we use a double hashing scheme for a hash table of size 20.

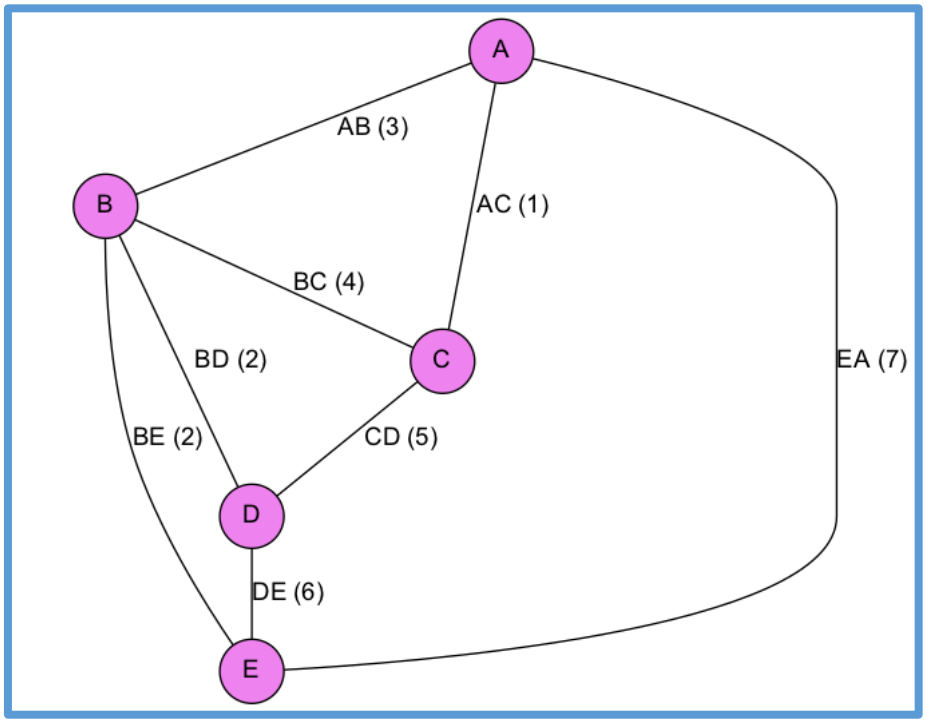
$h_1(k) = k\%20$

$h_2(k) = 12 - k\%12$

a. Insert 17. Which index? How many collisions?
b. Insert 27. Which index? How many collisions?
c. Insert 29. Which index? How many collisions?
d. Insert 47. Which index? How many collisions?

Index:	Key:
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	

UNDIRECTED GRAPHS (DFS, BFS)



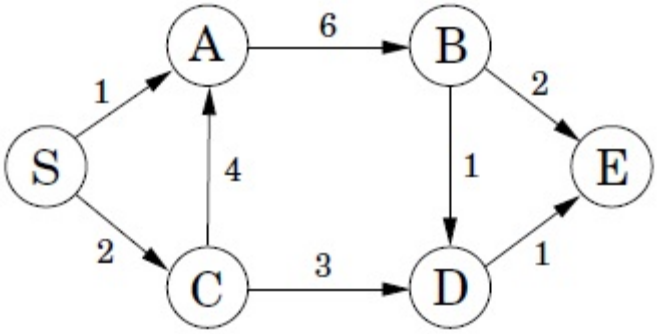
Note: In all algorithms, the tie breaking rule is to select the node with the lowest value, all other things being equal!

11. Please list the order of nodes for a depth-first search (DFS) and breadth-first search (BFS) starting at vertex A. All nodes should be listed:

DFS: _____

BFS: _____

12. DIRECTED GRAPHS (DFS, BFS, TOPOLOGICAL SORT)



Using the above directed graph, for each of the sequences given, tell if it it DFS, BFS, or TS sorted. Unlike the previous question, here there are no restrictions on which node gets chosen when there are multiple options to reach from a given node.

	DFS	BFS	TS
SACBDE			
SCDEAB			
SABDEC			
SCDBEA			

ANSWERS

EXPRESSION TREES – 1

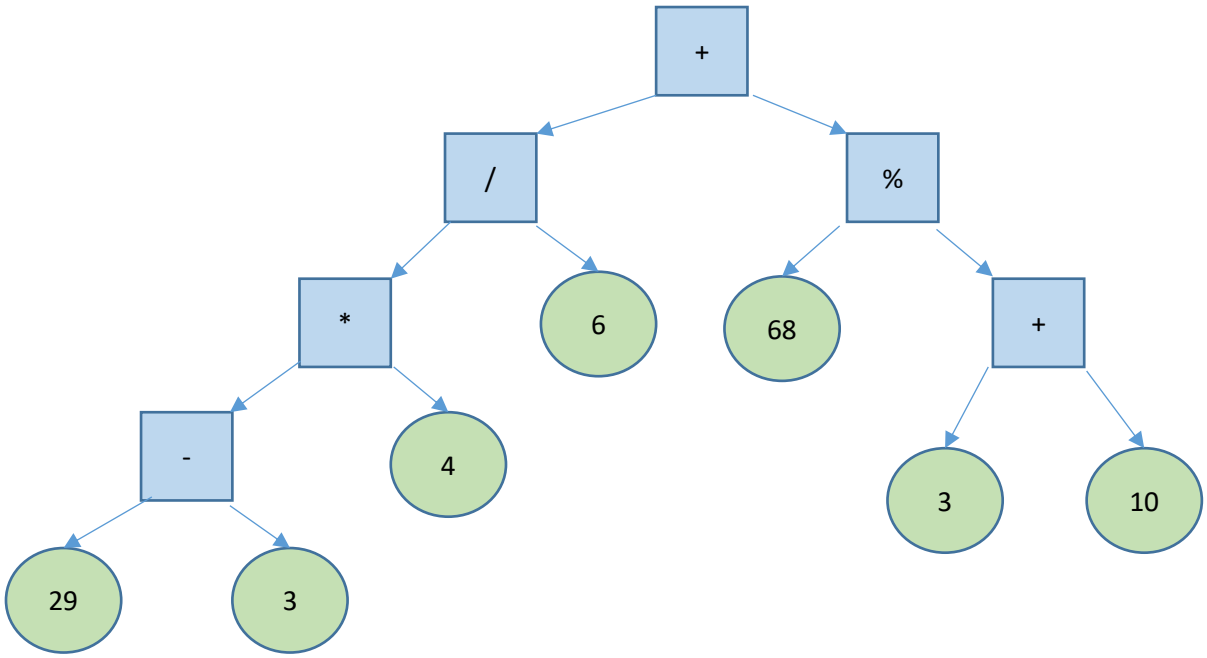
1. 14 6 % 12 2 / 3 * +

2. (14 % 6) + (12 / 2 * 3)

3. + % 14 6 * / 12 2 3

4. 20

5. Tree for this expression: (29 – 3) * 4 / 6 + 68 % (3 + 10)

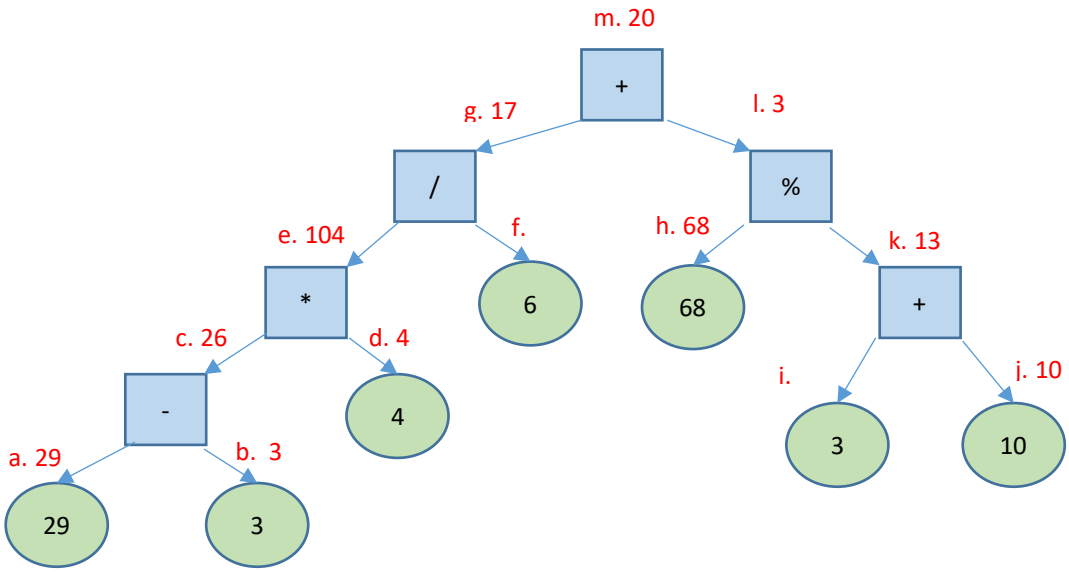


EXPRESSION TREES – 2

6. 29 3 – 4 * 6 / 68 3 10 + % +

7. + / * – 29 3 4 6 % 68 + 3 10

8. Tree evaluation is shown below. The letters a, b, c, … , m indicate the order in which values go up to the root. The infix calculation is 20.



HASHING

9. **LINEAR PROBING:**

Key = 15 maps to index 2, which is occupied.
Tries 3 (2+1), 4 (2+2), 5 (2+3) unsuccessfully,
Index 6 (2+4) is successful.
4 collisions.

QUADRATIC PROBING:

Key = 15 maps to index 2, which is occupied.
Tries 3 (2+1) unsuccessfully.
Index 6 (2+4) is successful.
2 collisions.

10. **DOUBLE HASHING:**

a. 17 % 20 = 3. Index 17. No collision.
b. 27 % 20 = 7. Index 7. No collision.
c. 29 % 20 = 9. Index 9. No collision.
d. 47 % 20 = 7. It is filled (with 27). Use second hash function.
   12 – 47 % 12 = 12 – 11 = 1
   So, new index is 7 + 1 = 8.
   Overall, 1 collision. Index = 8

11. **UNDIRECTED GRAPHS (DFS, BFS)**

DFS: A B C D E
BFS: A B C E D

12. **DIRECTED GRAPHS (DFS, BFS, Topological Sort)**

	DFS	**BFS**	**TS**
SACBDE	No	Yes	No
SCDEAB	Yes	No	No
SABDEC	Yes	No	No
SCDBEA	No	No	No

PART II

This part of the practice problem set corresponds to Weeks 6-12.

Linked Lists

Stacks and Queues

Heaps and Priority Queues

Expressions – Infix, Prefix, Postfix

Iterator and Iterable Interfaces

Binary Search Trees

B+ Trees

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 _____

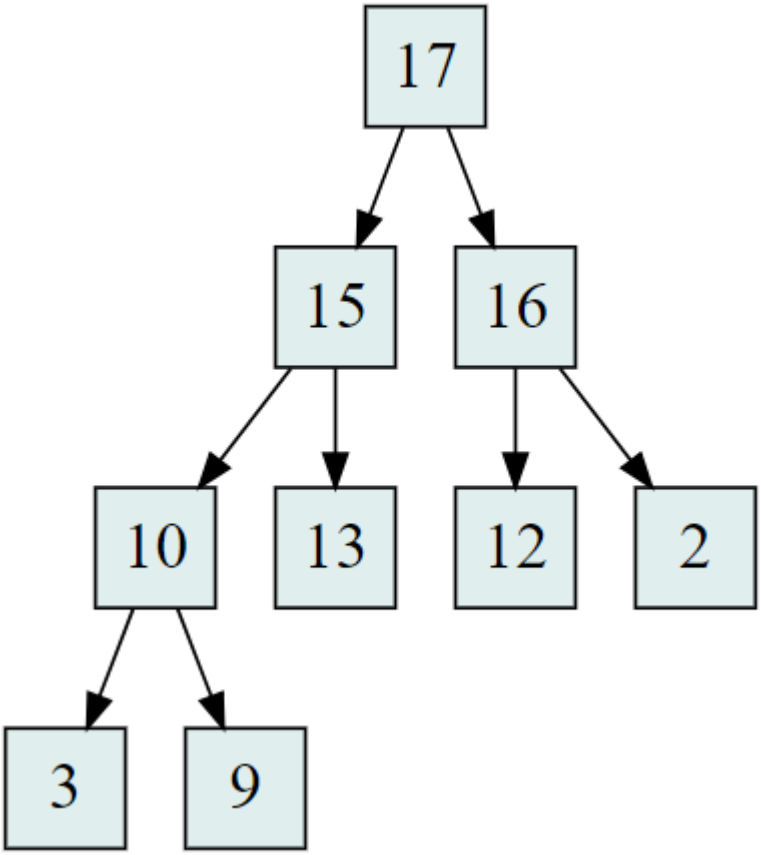
24. 123a321 _____

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?

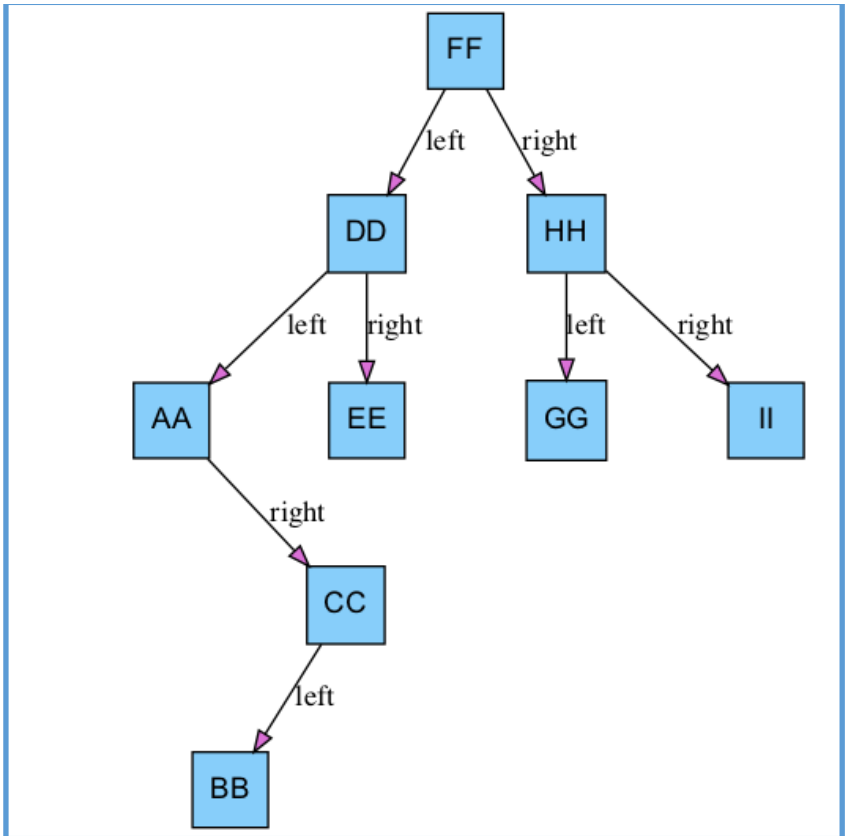
31. If we add any new node to our tree, the first step is to make it the child of which node?

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? _____

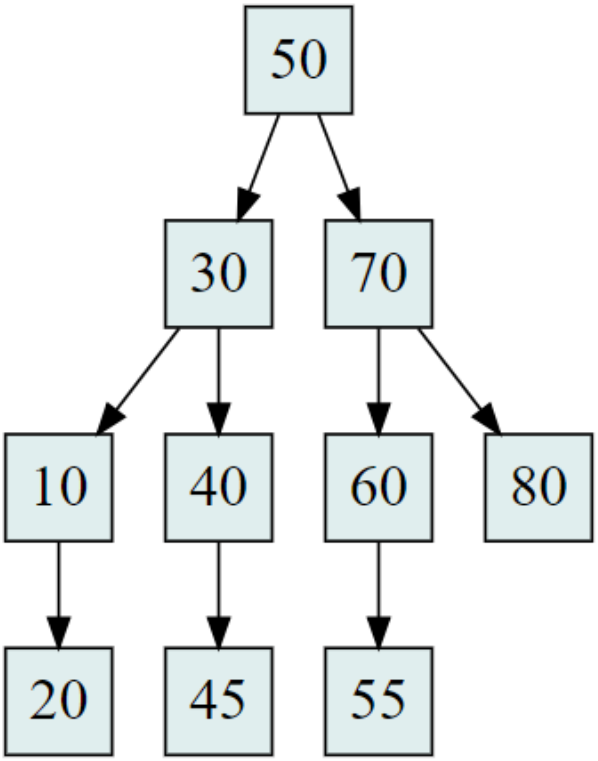
38. Which node will move to replace it? _____

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? _____

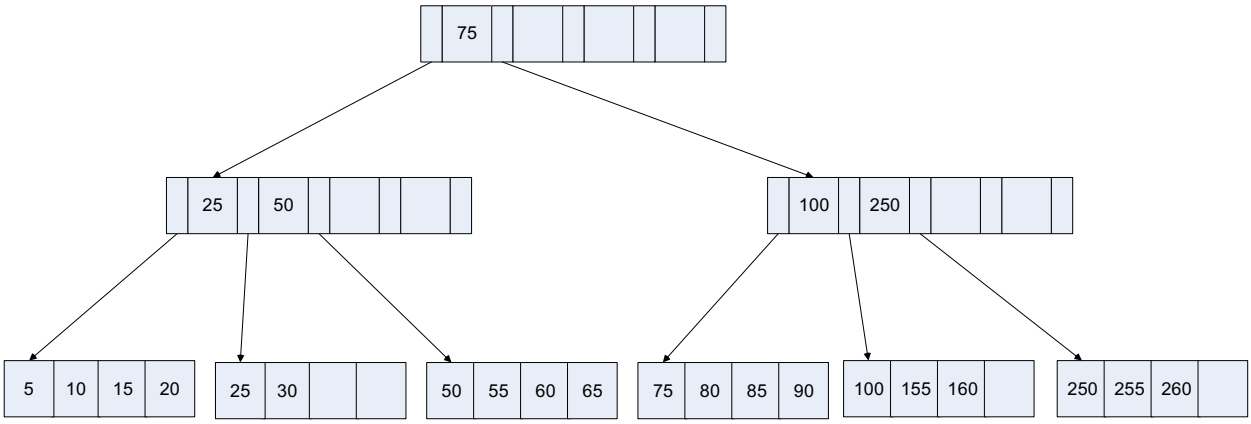
47. If we remove “70”, what node can we replace it with? _____

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());
}
```

PART III

This part of the practice problem set corresponds to Weeks 1-6.

Review of prerequisite materials

Software testing

Recursion

Inheritance, polymorphism

Overloading and overriding

Constructor chaining

Abstract classes and interfaces

For problems 1-4, circle the single answer that best answers the question. (3X4 = 12)

1. Select the correct description below:

a) In black-box testing, test inputs are derived from the implementation (code).

b) In white-box testing, test inputs are derived from the implementation (code).

c) Branch coverage requires that every decision evaluates to true or false, but not necessarily both.

d) Black-box testing requires statement coverage.

2. Select the best description of the utility of **inheritance** as implemented by the Java language:

a) Allows a programmer to extend classes (to specialize them) without code duplication.

b) Allows a programmer to override legacy code that works with new code that has defects!

c) Allows a programmer to write a subclass that overrides private methods in the super class.

d) Allows a programmer to write a concrete subclass without overriding abstract methods.

3. Select the best description of **polymorphism** as implemented by the Java language:

a) Prevents multiple objects in an inheritance hierarchy from being stored in a single collection.

b) Ensures that the methods associated with the class used to create the object (not the type of the reference variable) are called.

c) Provides dynamic binding which allows any method in any class to be loaded and executed.

d) Lets the programmer decide which method in the inheritance tree is called for a given object.

4. Select the best statement concerning the visibility modifiers:

a) The public modifier makes an instance variable and methods of a class visible to other public (but not private) methods in other classes.

b) The protected modifier makes an instance variable and methods of a class visible only to the subclasses of the class.

c) The private modifier makes an instance variable in an object inaccessible to objects of the same class.

d) The private modifier makes an instance variable in an class inaccessible to other classes.

JUnit assertions (4X2 = 8 points)

Give the declarations below, state which of the listed assertions pass:

```
int x = 10;
int y = 41;
double val = 4.0/3;
String word = "story".substring(2,2);
int[] nums = {0, 0, 0, 0, 0};
int[] copy = nums;
```

Assertion 1:

```
assertFalse(y%x == 0);
```

Assertion 2:

```
assertEquals(copy.length,0);
```

Assertion 3:

```
assertEquals(nums, copy);
```

Assertion 4:

```
assertNotNull(word);
```

Recursion (6 points)

The code for the recursive method combRec is given below.

```
public long combRec(long n, long k) {
     if (n==k || k==0)
          return 1;
     else
          return combRec(n-1,k-1) +  combRec(n-1,k);
}
```

How many calls are made to to combRec (inclusive of the original call) when you execute:

```
System.out.println(combRec(4,2));
```

Inheritance (4X4 + 5 =21 points)

```
public class Book {
   protected int pages;
   protected String name;
   public Book(int pages, String name){
      this.pages = pages; this.name = name;
   }
   public String motto(){
      return this + " The greatest book!";
   }
   public String toString(){
      return "Book: " + name + "\npages: " + pages ;
   }
}
public class Dictionary extends Book {
   private int defs;
   public Dictionary (int pn, String dn, int defs){
      super(pn,dn);
      pages = 2*pages;
      this.defs = defs;
   }
   public int getDefs(){
      return defs;
   }
   public String motto(){
      return("This dictionary is a fine " + super.motto() );
   }
   public String toString(){
      return super.toString() + " defs: " + defs;
   }
 }
 public class Words {
   public static void main(String[] args) {
      Dictionary D = new Dictionary(200, "Webster", 10000);
      Book limbo = new Book(150, "Alice in Wonderland");
      System.out.println(limbo.motto());
      limbo = D;
      System.out.println(limbo.motto());

      // what is wrong with the following line and how can it be fixed?
      //int defs = limbo.getDefs();    //
   }
}
```

What are the 4 lines printed above?

_____

_____

_____

_____

What is wrong with the last line in the class and how would you fix it? _____

Inheritance (4X4 =16 points)

Show what the program shown below would print.
HINT: Consider the inheritance hierarchy and polymorphism

```
public class PolymorphismProgram {

   public static class Vehicle {
      public void printMe() {
         System.out.println("Vehicle");
      }
   }
   public static class Car extends Vehicle {
      public void printMe() {
         System.out.println("Car");
      }
   }
   public static class Chevy extends Car {
      public void printMe() {
         System.out.println("Chevy");
      }
   }
   public static class Ford extends Car {
      public void printFord() {
         System.out.println("Ford");
      }
   }
   public static void main(String[] args) {
      Vehicle a = new Vehicle();
      Vehicle b = new Car();
      Vehicle c =  new Chevy();
      Car d = new Ford();
      a.printMe();
      b.printMe();
      c.printMe();
      d.printMe();
   }
}
```

_____

_____

_____

_____

Constructor Chaining (15 points)

What does this program print?

```
class P {
	public P() {
		System.out.println("P");
	}
	public P(int x) {
		System.out.println("P" + x);
	}
}

class Q extends P {
	public Q(int x) {
		System.out.println("Q"+x);
	}
	public Q() {
		this(4);
		System.out.println("Q");
	}
}


public class ChainingPractice {

	public static void main(String[] args) {
		Q q1 = new Q(3);
		Q q2 = new Q();
	}
}
```

Interfaces (4X4 =16 points)

```
import java.util.Arrays;
public class State implements Comparable<State> {
	private String name;
	private int year;

	public State(String name, int year) {
		this.name = name;
		this.year = year;
	}
	@Override
	public int compareTo(State o) {
		if(this.year > o.year)
			return 1;
		else if(this.year < o.year)
			return -1;
		else
			return 0;
	}
	@Override
	public boolean equals(Object o) {
		if(o instanceof State) {
			State other = (State)o;
			return (name.equals(other.name));
		}
		return false;
	}
	@Override
	public String toString() {
		return name + " " + year;
	}
	public static void main(String[] args) {
		State s1 = new State("Idaho", 1890);
		State s2 = new State("Maine", 1820);
		State s3 = new State("Texas", 1845);
		State s4 = new State("Utah", 1896);
		State[] list = new State[4];

		list[0] = s1;
		list[1] = s2;
		list[2] = s3;
		list[3] = s4;
		System.out.println(Arrays.toString(list));          // Line 1

		System.out.println(s1.compareTo(s2));  // Line 2
		System.out.println(s2.equals("Maine"));// Line 3
		System.out.println(s3.compareTo(s4));  // Line 4

		double total=0;
		for(State s: list) {
			total = total + s.year;
		}
		System.out.println((int)total/4); // Line 5
	}
}
```

What does this program print?

Generics (3X2 =6 points)

Assume that the appropriate import statement for ArrayList exists.

```
// Fragment A
ArrayList<String> a1 = new ArrayList<>();
a1.add("Hi");
a1.add("Bye");
Collections.sort(a1);
System.out.println(a1);
```

TRUE/FALSE: This fragment compiles without errors or warnings.
TRUE/FALSE: This fragment executes without exceptions.

```
// Fragment B
ArrayList a2 = new ArrayList();
a2.add("Hello");
a2.add(25);
Collections.sort(a2);
System.out.println(a2);
```

TRUE/FALSE: This fragment compiles without errors or warnings.
TRUE/FALSE: This fragment executes without exceptions.

```
// Fragment C
ArrayList<Integer> a3 = new ArrayList<>();
a3.add(45);
a3.add("45");
Collections.sort(a3);
System.out.println(a3);
```

TRUE/FALSE: This fragment compiles without errors or warnings.
TRUE/FALSE: This fragment executes without exceptions.

ANSWERS

1. b
2. a
3. b
4. d

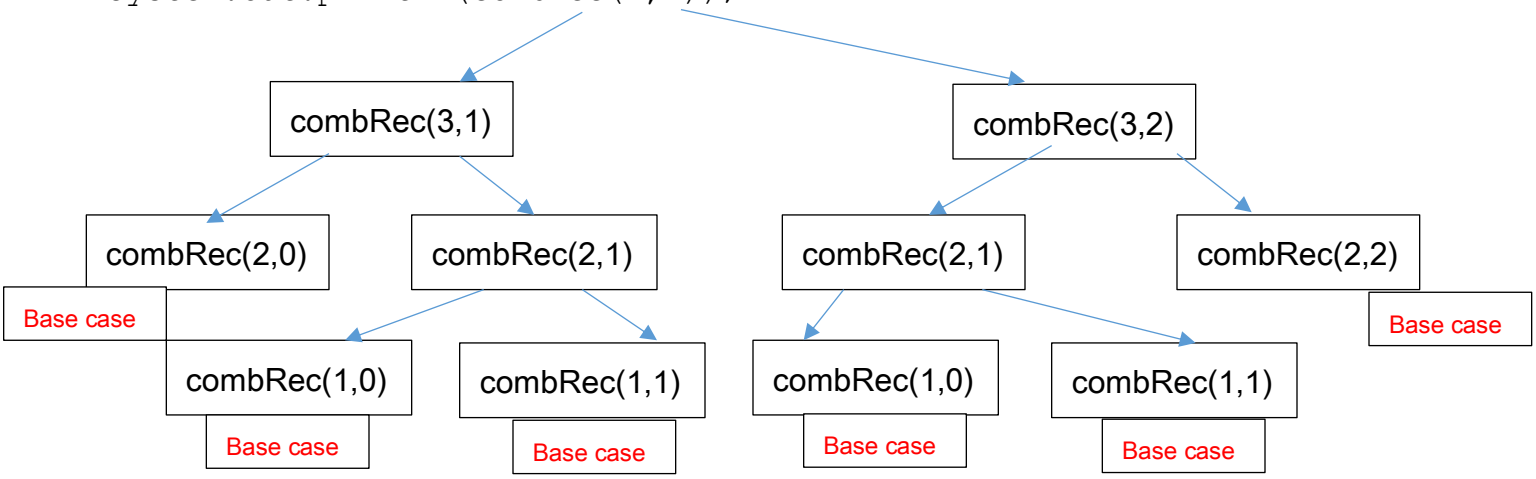
JUnit

Pass, Fail, Pass, Pass

Recursion

11

```
System.out.println(combRec(4,2));
```



Inheritance (Book and Words)

What are the 4 lines printed above?

```
Book: Alice in Wonderland
pages: 150 The greatest book!
This dictionary is a fine Book: Webster
pages: 400 defs: 10000 The greatest book!
```

What is wrong with the last line in the class and how would you fix it? `limbo is of type Book and doesn't have the getDefs method. Cast limbo to Dictionary.`

Inheritance (Polymorphism Program)

```
Vehicle
Car
Chevy
Car (since printMe is not overridden in Ford)
```

Constructor Chaining

```
P
Q3
P
Q4
Q
```

Interfaces

```
[Idaho 1890, Maine 1820, Texas 1845, Utah 1896]
1
false
-1
1862
```

Generics

Part A

TRUE: This fragment compiles without errors or warnings.

TRUE: This fragment executes without exceptions.

Part B

FALSE: This fragment compiles without errors or warnings.

FALSE: This fragment executes without exceptions.

Part C

FALSE: This fragment compiles without errors or warnings.

FALSE: This fragment executes without exceptions. (doesn't execute because it doesn't compile)