good luck everyone!!! you will kick this exam in the balls. thanks for using my study guide <333

1.

Write the bytecode generation for the following BL Program. Use symbolic names for jump conditions and primitive instructions.

PROGRAM Mystery IS

INSTRUCTION myInstruction IS

turnright

IF next-is-enemy THEN

turnleft

END IF

END myInstruction

BEGIN

IF next-is-wall THEN

move

infect

WHILE next-is-not-empty

turnleft

END WHILE

ELSE

IF true THEN

myInstruction

END IF

skip

END IF

skip

END Mystery

Note: not all memory locations may be used

| 0 | JUMP\_IF\_NOT\_NEXT\_IS\_WALL |
| --- | --- |
| 1 | 11 |
| 2 | MOVE |
| 3 | INFECT |
| 4 | JUMP\_IF\_NOT\_NEXT\_IS\_NOT\_EMPTY |
| 5 | 9 |
| 6 | TURNLEFT |
| 7 | JUMP |
| 8 | 4 |
| 9 | JUMP |
| 10 | 18 |
| 11 | JUMP\_IF\_NOT\_TRUE |
| 12 | 17 |
| 13 | TURNRIGHT |
| 14 | JUMP\_IF\_NOT\_NEXT\_IS\_ENEMY |
| 15 | 17 |
| 16 | TURNLEFT |
| 17 | SKIP |
| 18 | SKIP |
| 19 | HALT |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |

2.

Implement the following method that returns the location specified by the next unconditional jump in the given program, given what the bug sees, the current memory location and the compiled program. If there is no unconditional jump, it returns -1. Assume you are given methods conditionalJumpCondition and isPrimitiveInstructionByteCode, specified by the contracts below.

/\*\*

\* Returns the value of the condition in the given conditional jump

\* {@code condJump} given what the bug sees {@code wbs}. Note that if

\* {@code condJump} is the byte code for the conditional jump

\* JUMP\_IF\_NOT\_condition, the value returned is the value of the

\* "condition" part of the jump instruction.

\*

\* **@param** wbs

\* the {@code CellState} indicating what the bug sees

\* **@param** condJump

\* the byte code of a conditional jump

\* **@return** the value of the conditional jump condition

\* **@requires** [condJump is the byte code of a conditional jump]

\* **@ensures** <pre>

\* conditionalJumpCondition =

\* [the value of the condition of condJump given what the bug sees wbs]

\* </pre>

\*/

private static boolean conditionalJumpCondition(CellState wbs,

int condJump) {

/\*\*

\* Returns whether the given integer is the byte code of a BugsWorld

\* virtual

\* machine primitive instruction (MOVE, TURNLEFT, TURNRIGHT, INFECT,

\* SKIP, HALT).

\*

\* **@param** byteCode

\* the integer to be checked

\* **@return** true if {@code byteCode} is the byte code of a primitive

\* instruction or false otherwise

\* **@ensures** <pre>

\* isPrimitiveInstructionByteCode =

\* [true iff byteCode is the byte code of a primitive instruction]

\* </pre>

\*/

private static boolean isPrimitiveInstructionByteCode(int byteCode);

/\*\*

\* Returns the location specified by the next unconditional jump to

\* execute in

\* compiled program {@code cp} given what the bug sees {@code wbs} and

\* starting from location {@code pc}.

\*

\* **@param** cp

\* the compiled program

\* **@param** wbs

\* the {@code CellState} indicating what the bug sees

\* **@param** pc

\* the program counter

\* **@return** the location specified by the next unconditional jump to

\* execute

\* **@requires** <pre>

\* [cp is a valid compiled BL program] and

\* 0 <= pc < cp.length and

\* [pc is the location of an instruction byte code in cp, that is, pc

\* cannot be the location of an address]

\* </pre>

\* **@ensures** <pre>

\* [return the address specified by the next unconditional jump that

\* should be executed in program cp given what the bug sees wbs and

\* starting execution at address pc in program cp]

\* </pre>

\*/

public static int nextUnconditionalJumpAddress(int[] cp, CellState wbs,

int pc) {

int result = -1;

int i = pc;

boolean found = false;

while (i < cp.length && !found) {

if (isPrimitiveInstructionByteCode(cp[i])) {

i++;

} else if (cp[i] == Instruction.JUMP.byteCode()) {

result = cp[i + 1];

found = true;

} else {

if (conditionalJumpCondition(wbs, cp[i])) {

i += 2;

} else {

i = cp[i + 1];

}

}

}

return result;

}

3.

Write a recursive descent parser for the following grammar that parses a mathematical string of single digit integers into a Queue of ints. Each method is an instance method which will be called on a Queue<Integer>.

ex. <> and <1,2,3> are valid in this grammar. The result would be a Queue of integers with that mathematical representation.

mathString -> < [ elements ] >

elements -> entry {, entry }

entry -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

/\*\*

\* Parses the queue of tokens into this queue

\*

\* @replaces this

\*

\* @updates tokens

\*

\* @requires

\* tokens is a valid expression as defined by the grammar,

\* end\_of\_input is a suffix of tokens

\* @ensures

\* this = the queue modeled by the expression represented in tokens

\*/

public void parseMathString(Queue<String> tokens) {

this.clear();

tokens.dequeue();

if (Integer.CanParseInt(tokens.front())){

this.parseElements(tokens);

}

tokens.dequeue();

}

/\*\*

\* Parses the queue of tokens into this queue

\*

\* @updates this

\*

\* @updates tokens

\*

\* etc.

\*/

public void parseElements(Queue<String> tokens){

this.parseEntry(tokens);

while (tokens.front().equals(",")){

tokens.dequeue();

this.parseEntry(tokens);

}

}

/\*\*

\* Parses the queue of tokens into this queue

\*

\* @updates this

\*

\* @updates tokens

\*

\* etc.

\*/

public void parseEntry(Queue<String> tokens){

this.enqueue(Integer.parseInt(tokens.dequeue()));

}

4.

The following comparator sorts strings alphabetically regardless of case. Here is a sample of a list sorted according to this comparator:

alpha

bar

Beta

cat

Foo

foo

gamma

Zeta

zeta

private static class StringLT implements Comparator<String> {

@Override public int compare(String o1, String o2) {

return o1.compareToIgnoreCase(o2);

}

}

Change the comparator so that it sorts the same way, but is consistent with equals. If two strings are spelled the same but differ in case, it does not matter which comes first, as long as they appear consecutively in the sorted list.

private static class StringLT implements Comparator<String> {

@Override public int compare(String o1, String o2) {

int result = o1.compareToIgnoreCase(o2);

if(result == 0){

result = o1.compareTo(o2);

}

return result;

}

}

5.

Suppose fast direct access has just been discovered for strings with dynamic size! You are implementing Sequence5 based on this recent discovery. You have implemented all of the kernel methods, but some of the secondary method implementations are too slow, and don’t take advantage of fast direct access. How can you make sure Sequence5 has different implementations for its secondary methods?

Override them in the implementing class (Sequence5)

6.

Implement the equals method for Stack<T>

public boolean equals(Object obj){

if(obj == this){

return true;

}

if(obj == null){

return false;

}

if(!(obj instanceof Stack<?>)){

return false;

}

Stack<?> x = (Stack<?>) obj;

if(x.length()!= this.length()){

return false;

}

Iterator<T> it1 = this.iterator();

Iterator<?> it2 = x.iterator();

while(it1.hasNext()){

T thisNext = it1.next();

Object xNext = it2.next();

if(!thisNext.equals(xNext)){

return false;

}

}

return true;

}

7.

Write a method that, given a java.util Map<String, Integer> increments the value if it matches the given key

/\*\*

\* given {@code Map}, increments the value if it matches the given

\* {@code key}

\*

\* **@param** map

\* the given map

\* **@param** key

\* the given key

\* **@ensures** if map contains key, map.value(key) is incremented

\*/

public static void incrementValue(Map<String, Integer> map, String key) {

Set<Map.Entry<String, Integer>> entrySet = map.entrySet();

Iterator<Map.Entry<String, Integer>> it = entrySet.iterator();

while (it.hasNext()) {

Map.Entry<String, Integer> next = it.next();

if (next.getKey().equals(key)) {

int newVal = next.getValue();

newVal++;

next.setValue(newVal);

}

}

}

8.

Write a main method that takes a file as input, the name of an output location, and a number of lines to be copied from the input file and printed to the output file. First, declare that it simply throws an IOException, then catch and handle all the IOExceptions. Take the input as command-line arguments

public static void main(String[] args) throws IOException {

String inputFile = args[0];

String outputLocation = args[1];

String numberOfLines = args[2];

int num;

try {

num = Integer.parseInt(numberOfLines);

if (num < 0) {

System.err.println("number of lines should be positive");

}

} catch (NumberFormatException e) {

System.err.println("number of lines must be in integer format");

return;

}

BufferedReader fileReader = new BufferedReader(

new FileReader(inputFile));

PrintWriter fileOutput = new PrintWriter(

new BufferedWriter(new FileWriter(outputLocation)));

String s = fileReader.readLine();

int i = 0;

while (s != null && i < num) {

fileOutput.println(s);

s = fileReader.readLine();

i++;

}

fileReader.close();

fileOutput.close();

}

public static void main(String[] args) {

String inputFile = args[0];

String outputLocation = args[1];

String numberOfLines = args[2];

int num;

try {

num = Integer.parseInt(numberOfLines);

if (num < 0) {

System.err.println("number of lines should be positive");

}

} catch (NumberFormatException e) {

System.err.println("number of lines must be in integer format");

return;

}

BufferedReader fileReader;

try {

fileReader = new BufferedReader(new FileReader(inputFile));

} catch (IOException e) {

System.err.println("Error opening file");

return;

}

try {

PrintWriter fileOutput = new PrintWriter(

new BufferedWriter(new FileWriter(outputLocation)));

try {

String s = fileReader.readLine();

int i = 0;

while (s != null && i < num) {

fileOutput.println(s);

s = fileReader.readLine();

i++;

}

} catch (IOException e) {

System.err.println("Error reading file");

}

fileOutput.close();

} catch (IOException e) {

System.err.println("Error creating file");

}

try {

fileReader.close();

} catch (IOException e) {

System.err.println("Error closing file");

}

}

9.

Suppose you have a program that keeps track of all books in a library. A book is composed of a title and an author. Each time a book is created, it is added to the library. There can be more than one copy of a book, but if all the copies are checked out, the book is not available. Implement the following interface according to this idea and the method contracts, using static class members.

/\*\*

\* A book modeled by a title and author

\*

\* **@author** Izzy Smith

\*

\*/

public interface Book {

//im trying with this javadoc cut me some slack

/\*\*

\* reports the title

\*

\* **@return** the title of this book

\*

\* **@ensures** title = this.title

\*/

String title();

/\*\*

\* reports the author

\*

\* **@return** the author of this book

\*

\* **@ensures** author = this.author

\*/

String author();

/\*\*

\*

\* reports whether this is available

\*

\* **@return** whether this book is available in the global library

\*

\* **@ensures** available = (number of total copies of the book) != 0

\*/

boolean available();

/\*\*

\* checks this book out of the library

\*

\* **@requires** available == true;

\*

\* **@ensures** (number of total copies of the book) = (number of total

\* copies of the book - 1)

\*

\*/

void checkOut();

/\*\*

\* **@ensures** (number of total copies of the book) = (number of total

\* copies of the book + 1)

\*/

void checkIn();

}

import java.util.Map;

import java.util.TreeMap;

/\*\*

\* Implementation of the Book interface

\*

\* **@author** Izzy Smith

\*

\*/

public class Book1 implements Book {

//private members

private String title;

//in the format Lastname, Firstname

private String author;

//in the format title, Lastname, Firstname

private String fullIdentifier;

private static Map<String, Integer> library = new TreeMap<String,

Integer>();

public Book1(String title, String author) {

this.title = title;

this.author = author;

this.fullIdentifier = title + ", " + author;

if (library.containsKey(this.fullIdentifier)) {

int count = library.get(this.fullIdentifier);

count++;

library.replace(this.fullIdentifier, count);

} else {

library.put(this.fullIdentifier, 1);

}

}

@Override

public String title() {

return this.title;

}

@Override

public String author() {

return this.author;

}

@Override

public boolean available() {

return library.get(this.fullIdentifier) != 0;

}

@Override

public void checkOut() {

int count = library.get(this.fullIdentifier);

count--;

library.replace(this.fullIdentifier, count);

}

@Override

public void checkIn() {

int count = library.get(this.fullIdentifier);

count++;

library.replace(this.fullIdentifier, count);

}

@Override

public String toString() {

return this.fullIdentifier;

}

}

10.

Write a recursive static method that reverses all the elements of a tree.

example:

before-

7

/ | \

4 6 5

/ | \ / \

3 2 1 3 9

after-

7

/ | \

5 6 4

/ \ / | \

9 3 1 2 3

/\*\* im trying pls ik these method contracts are whack

\* reverses all the elements of a tree

\*

\* **@param** <T>

\* the type of elements in the tree

\* **@param** t

\* the tree of be reversed

\* **@updates** t

\* **@ensures** reverses the order of the children of t and reverses each

\* child

\*/

public static <T> void reverse(Tree<T> t) {

if (t.size() > 1) {

Sequence<Tree<T>> children = t.newSequenceOfTree();

T root = t.disassemble(children);

for (int i = 0; i < children.length(); i++) {

reverse(children.entry(i));

}

children.flip();

t.assemble(root, children);

}

}

11.

Implement Set on Stack in the following class:

import java.util.Iterator;

import components.set.Set;

import components.set.SetSecondary;

import components.stack.Stack;

import components.stack.Stack1L;

/\*\*

\* {@code Set} represented as a {@code Queue} of elements with implementations

\* of primary methods.

\*

\* **@param** <T>

\* type of {@code Set} elements

\* **@convention** |$this.elements| = |entries($this.elements)|

\* **@correspondence** this = entries($this.elements)

\*/

public class Set3<T> extends SetSecondary<T> {

/\*

\* Private members --------------------------------------------------------

\*/

/\*\*

\* Elements included in {@code this}.

\*/

private Stack<T> elements;

/\*\*

\* Finds {@code x} in {@code s} and, if such exists, moves it to the top

\* of {@code s}.

\*

\* **@param** <T>

\* type of {@code Stack} entries

\* **@param** s

\* the {@code Stack} to be searched

\* **@param** x

\* the entry to be searched for

\* **@updates** s

\* **@ensures** <pre>

\* perms(q, #q) and

\* if <x> is substring of q

\* then <x> is prefix of q

\* </pre>

\*/

private static <T> void moveToTop(Stack<T> s, T x) {

assert s != null : "Violation of: s is not null";

//assume correct implementation

}

/\*\*

\* Creator of initial representation.

\*/

private void createNewRep() {

this.elements = new Stack1L<T>();

}

/\*

\* Constructors -----------------------------------------------------------

\*/

/\*\*

\* No-argument constructor.

\*/

public Set3() {

this.createNewRep();

}

/\*

\* Standard methods -------------------------------------------------------

\*/

//clear, newInstance and transferFrom not included for cleanliness

/\*

\* Kernel methods ---------------------------------------------------------

\*/

@Override

public final void add(T x) {

assert x != null : "Violation of: x is not null";

assert !this.contains(x) : "Violation of: x is not in this";

this.elements.push(x);

}

@Override

public final T remove(T x) {

assert x != null : "Violation of: x is not null";

assert this.contains(x) : "Violation of: x is in this";

moveToTop(this.elements, x);

return this.elements.pop();

}

@Override

public final T removeAny() {

assert this.size() > 0 : "Violation of: this /= empty\_set";

return this.elements.pop();

}

@Override

public final boolean contains(T x) {

assert x != null : "Violation of: x is not null";

boolean result = false;

for (T e : this.elements) {

if (e.equals(x)) {

result = true;

}

}

return result;

}

@Override

public final int size() {

return this.elements.length();

}

//Iterator removed for cleanliness

}

12.

Implement renameInstruction. Sorry no twists or pizazz here just good practice.

//no solution given since this is an actual SW2 assignment from earlier

13.

The following method is an instance method for a doubly-linked list with two smart nodes (same instance variables as the ListWithRetreat project). What does mystery(7) do? Draw the result of this method given the example list below.

public void mystery(T x){

Node p = new Node();

p.data = x;

this.lastLeft.next = p;

p.prev = this.lastLeft;

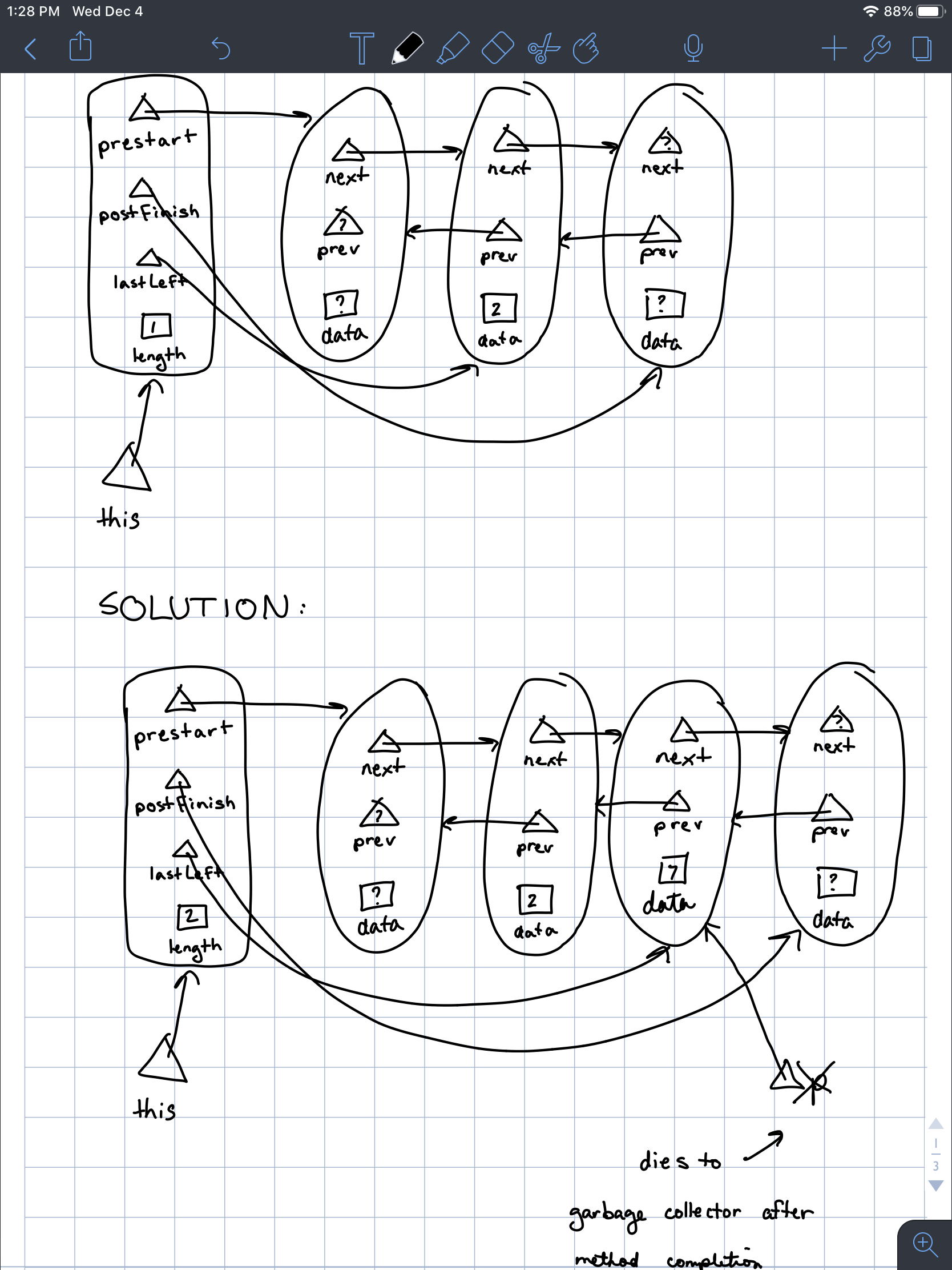
p.next = this.postFinish;

this.postFinish.prev = p;

this.lastLeft = p;

this.length++;

}



14.

implement the set kernel methods using a hashtable representation

public class Set4<T> extends SetSecondary<T> {

// I did have a solution here, but thinking about it, it’s a little too similar to MapWithHashing. the solution will remain a mystery.

15.

Write a recursive instance method that reverses all the elements of a binary tree. (assume reverses means the same as the similar question above).

/\*

\* reverses {@code this}

\* @**requires** this is not null

\* **@ensures** this = compose(root, right.reverse(), left.reverse())

\*/

public <T> void reverse(){

if (this.sise()>0){

BinaryTree<T> left = this.newInstance();

BinaryTree<T> right = this.newInstance();

T root = this.disassemble(left, right);

left.reverse();

right.reverse();

this.assemble(root, right, left);

}

}

16.

Draw the AST for the following BL Statement:

WHILE true DO

turnleft

IF next-is-empty THEN

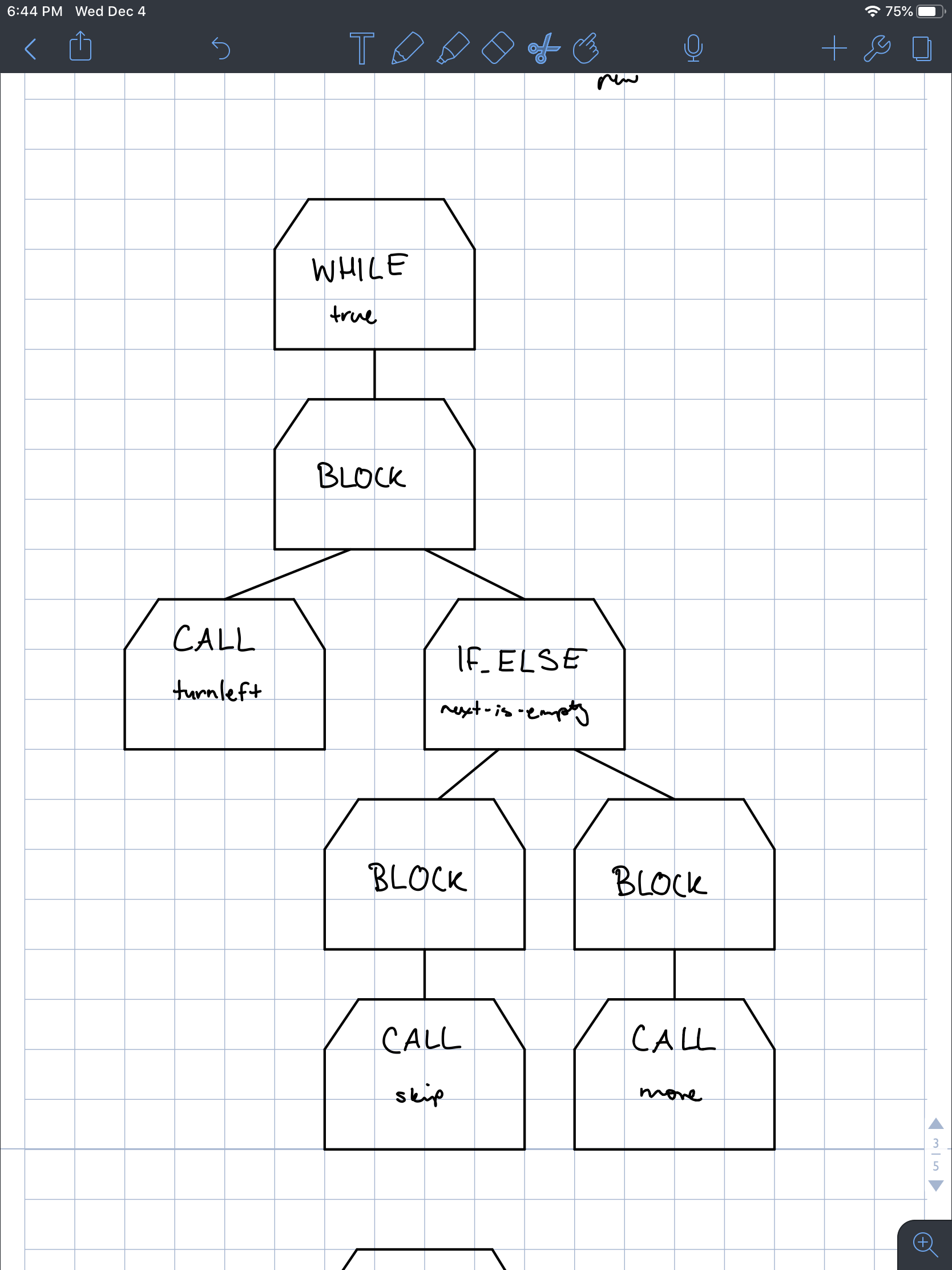
skip

ELSE

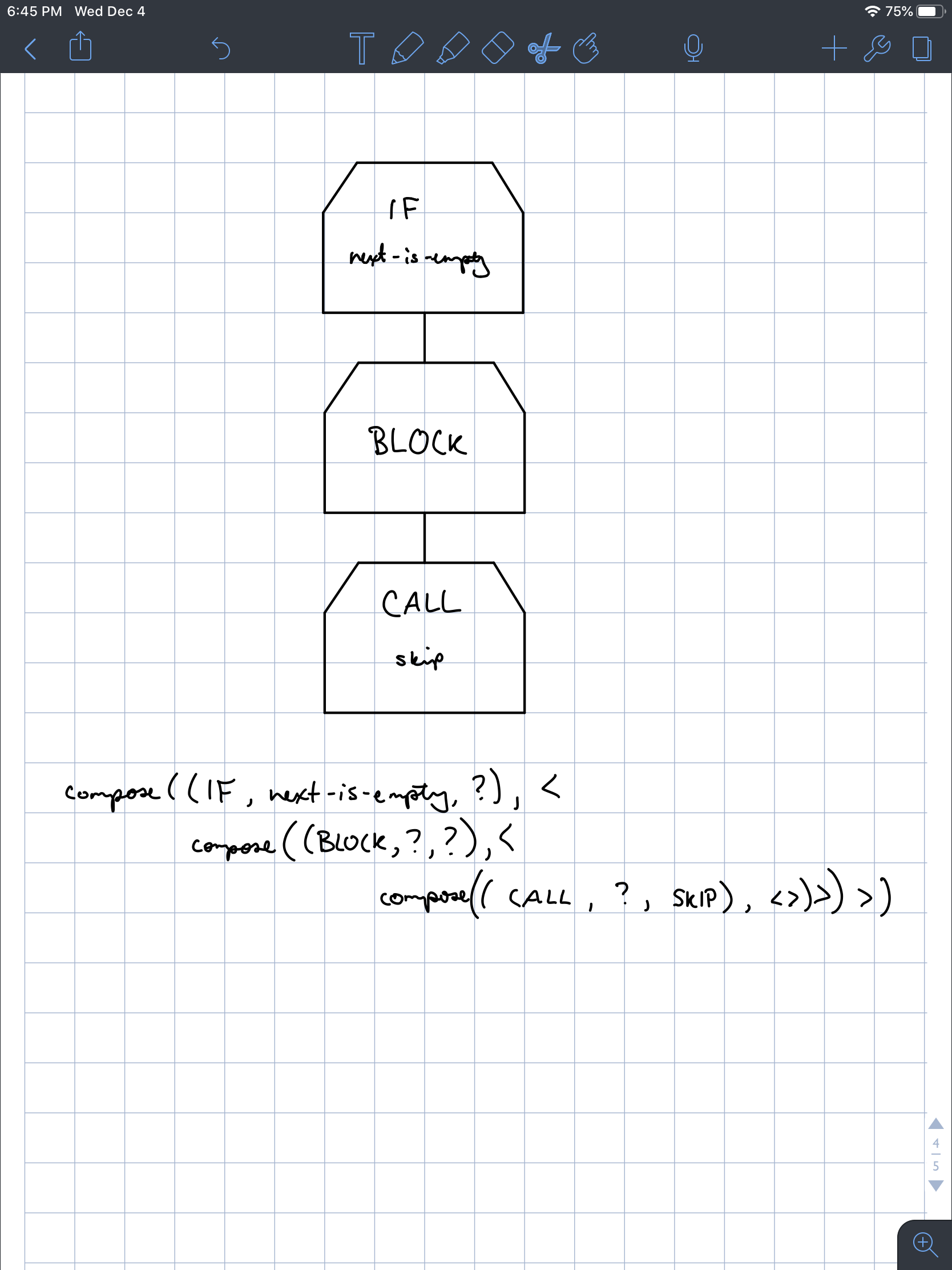
move

END IF

END WHILE



17. write the following AST in mathematical notation



18.

Write the mathematical notation for the following program (draw the AST instead of using the compose function for any trees)

PROGRAM MyProgram IS

BEGIN

WHILE true DO

turnleft

IF next-is-empty THEN

skip

ELSE

move

END IF

END WHILE

END MyProgram

