**CS 4700 Final 105 Points**

**Laryssa Revelli, A01979841**

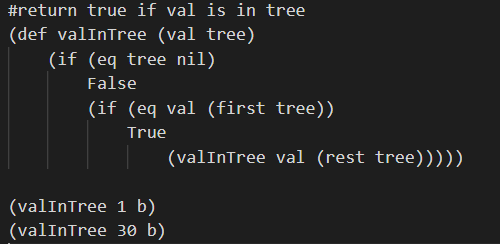
**Part One: eL language interpreter problems (9 pts each)**

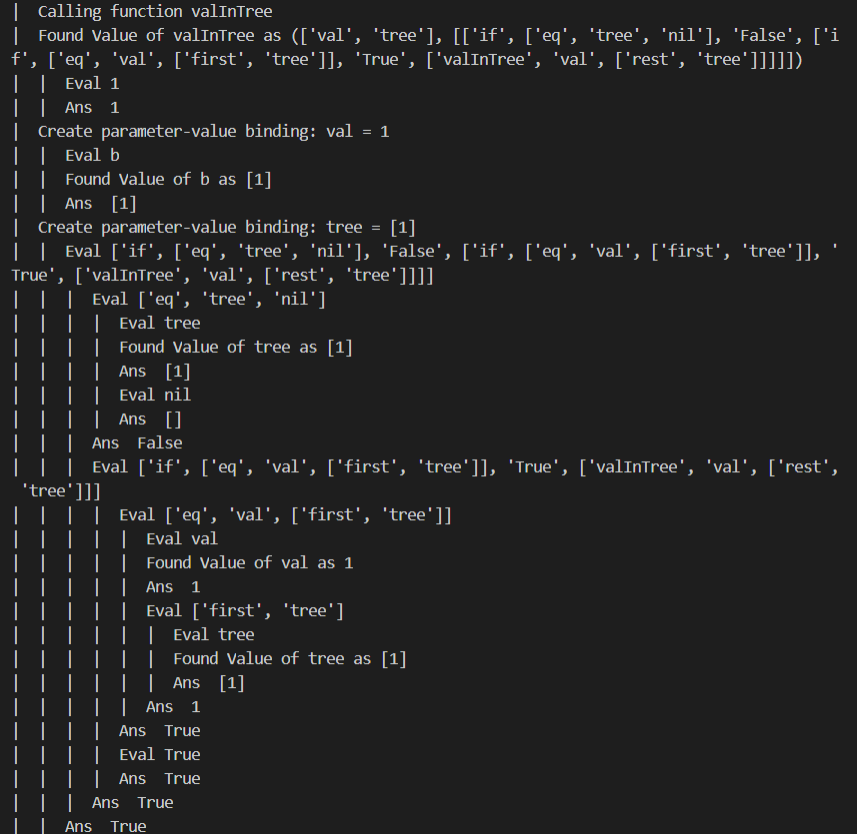
The code and associated files for our **eL** interpreter is included for your reference (see canvas)

**Q1) Binary Search in eL**

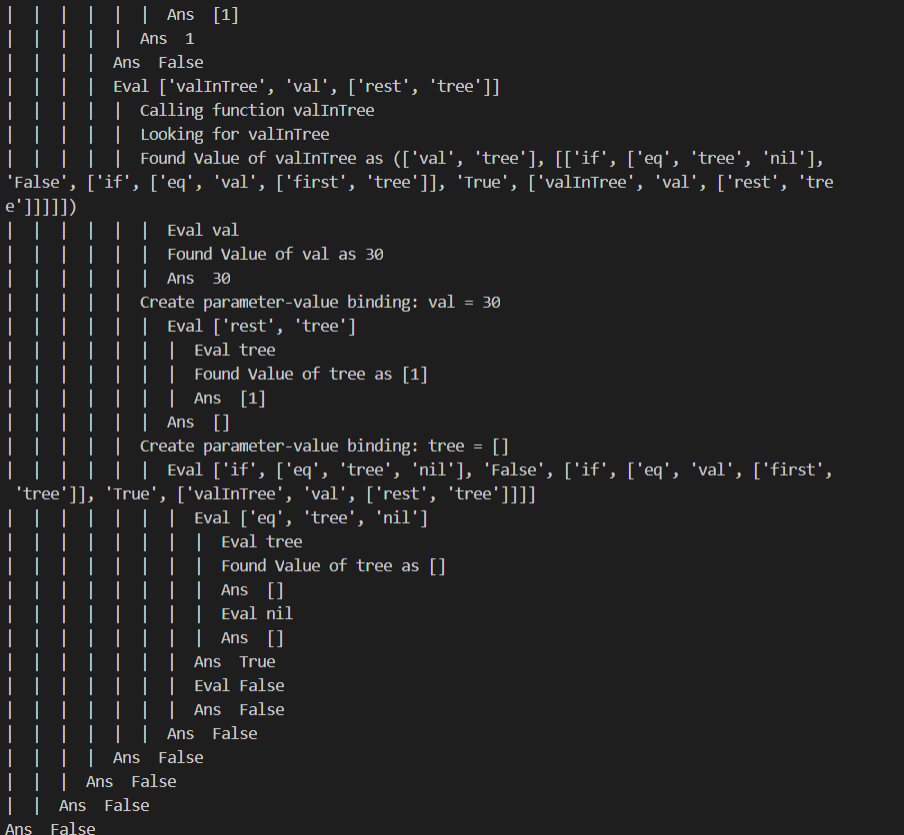
A simple way to represent a binary tree using **eL** data structures is to use the **eL** list. The root of the binary tree is stored at the first index of the list, (aka the first in order of items in the list). For every node, the left child is stored in the index that is twice the index of the parent node, and the right child is stored at the index after that. For example, for node at index n, the left child is found at index (\* 2 n) and the right child at (+ 1 (\* 2 n).

Write a function in **eL** that takes an integer and a binary tree and returns True if the integer is in the binary tree.

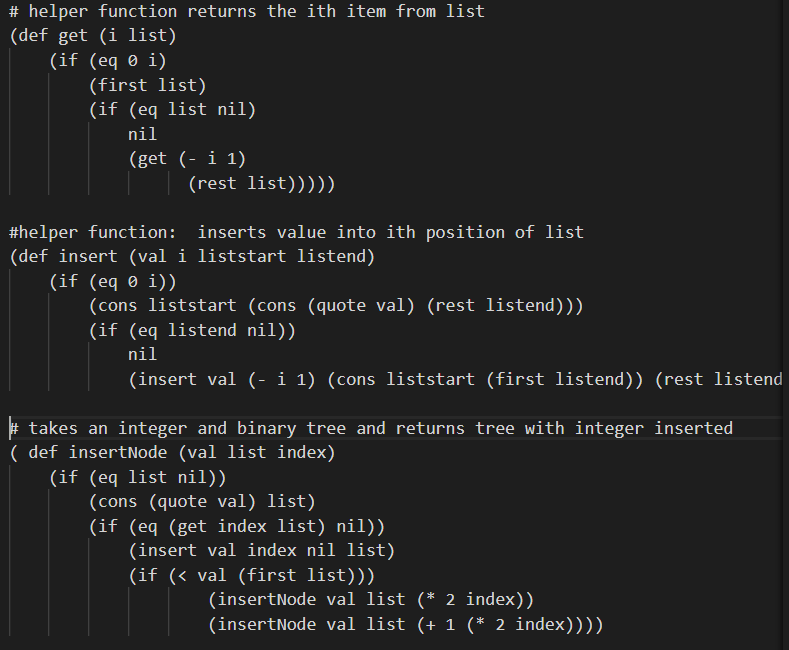


Stack trace calling (valInTree 1 [1])

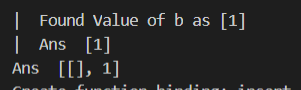
End of stack trace calling (valInTree 30 [1])



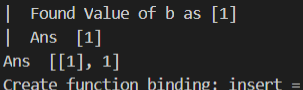
Write a function in **eL** that takes an integer and a binary tree and returns a new binary tree with the integer inserted.



(insertNode 1 nil 0)

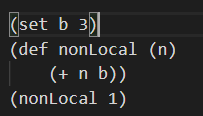
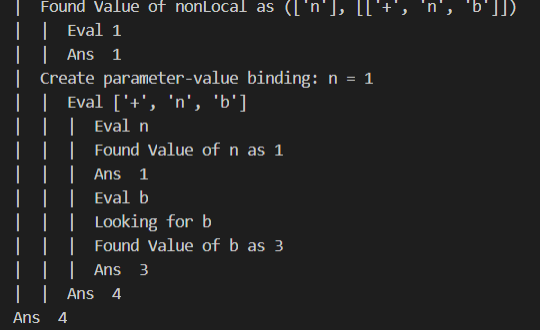


(insertNode 1 b 0)

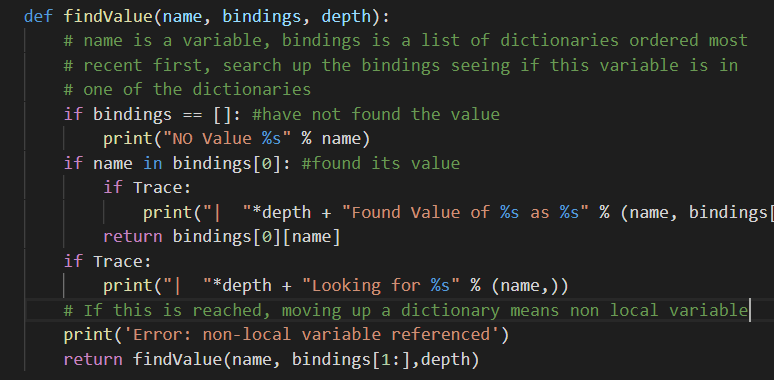


**Q2)** **eL** implements dynamic binding and allows access to the value of non-local variables. Here we define a local variable as one that has been set in the current block or is one of the parameters of the current function call. Answer the following questions:

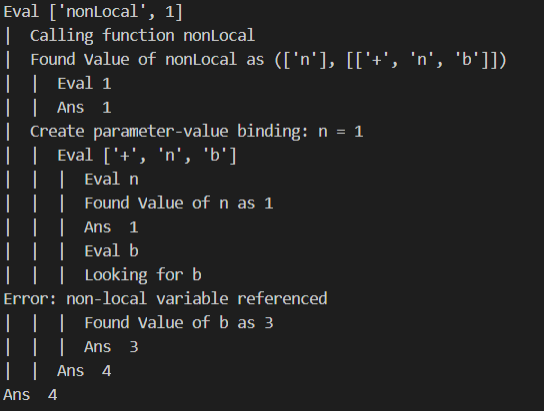
Write a few lines of **eL** code that illustrate this binding of non-local variables and show a trace of the code running.

Modify the **findValue** function so that **eL** will signal an error (just a printout) if a non-local variable is referenced. Show you code below.



Run the same **eL** code that you wrote above and show that is now signals an error.



**Q3)** The **eL** syntax checker reads in the file **eLgrammarDefinition.txt** and uses it to check the syntax of an **eL** program. To check the syntax it traverses through the parsed code and grammar tree checking correspondence. Answer the following questions

Referring to the book, what is the name of this kind of syntax checker?

Recognition device/ language recognizer

Can this checker detect when a function is called before it is defined? If not, what changes would be needed to the code? No need to actually code this up. Just describe how the check would work.

No, this would require executing only the def statements first, while parsing the other statements, tracking when there was an error of a function called that was not recognized. Then upon executing the statements if there is no error of a function not existing, you know that it was defined after the call to the function itself.

According to the grammar, an expression is a statement, but a statement is not an expression. What are some of the key differences?

A statement consists of expressions as well as variable assignments and function Definitions, whereas expressions consist solely of function calls or expression lists.

Many of the grammar rules are recursive such as

**<body> ::= ' | <statement> <body>**

Describe in words what this rule is saying

It is saying that a body is empty or a statement followed by another body (which could in turn consist of empty or more statements recursively). In essence, it implies that a body is a list of statements of any length.

Give two examples of a legal body statement in **eL**

1) (+ 1 2)

2) (set b 2)

(\* b (+ 1 b))

This rule could be defined as follows:

**<body> ::= ' | <body> <statement>**

Does this describe exactly the same set of legal body statements? If so, why is it not written this way? What is this kind of rule called?

Not exactly, written in this way specifies left associativity, and this requires modifying the grammar to use syntax analysis algorithms. When a grammar rule has its left hand side appearing at the beginning of the right hand side the rule is called left recursive.

**Q4)** The **eL** syntax is defined in the file **eLgrammarDefinition.txt**. Notice that the grammar allows numerical calculations on Boolean expressions and Boolean calculations on numerical expressions. This is because it is often convenient to have True be 1 and False be 0. These rules are defined in the following grammar rules:

**<numericalExpression> ::= <number> | (+ <expression> <expression>) |(- <expression> <expression>) |(\* <expression> <expression>) | (/ <expression> <expression>)**

**<booleanExpression> ::= True | False | (< <expression> <expression>) | (atom <expression>) | (and <expression> <expression>) |(or <expression> <expression>) |(not <expression>)**

**<controlExpression> ::= (if <expression> <expression> <expression>)**

Rewrite these rules so that numerical operators (**+, \*, \, +**) must have numerical expressions as arguments, Boolean operators (**and, or, not**) must have Boolean expressions as arguments and that the test condition of **if** must be a Boolean expression.

**<numericalExpression> ::= <number> | (+ <numericalExpression> <numericalExpression>) |(- <numericalExpression> <numericalExpression>) |(\* <numericalExpression> <numericalExpression>) | (/ <numericalExpression> <numericalExpression>)**

**<booleanExpression> ::= True | False | (< <numericalExpression> <numericalExpression>) | (atom <booleanExpression>) | (and <booleanExpression> <booleanExpression>) |(or <booleanExpression> <booleanExpression>) |(not <booleanExpression>)**

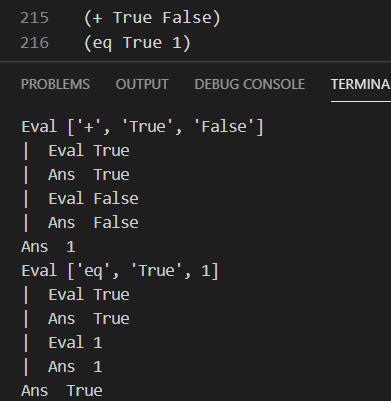
**<controlExpression> ::= (if <booleanExpression> <expression> <expression>)**

Rerun the **eL** code contained in the test file **code.el** and see if everything still works. Explain if any of our original code now causes a syntax error.

All code still passes due to a bug in syntax.py

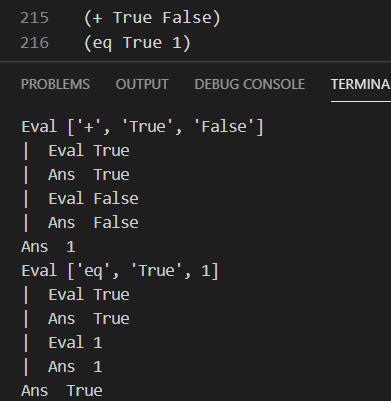
Write two new **eL** functions that work for the previous grammar but generates a syntax error with the new grammar. Include snippets that show this new behavior.

With previous grammar:

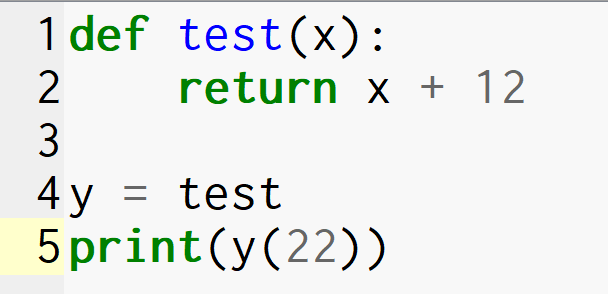


With new grammar: passes but it shouldn’t

This should have had a syntax error:



**Q5)** In many languages, such as Python, it is possible to assign the value of a variable to a name of a function, then call the function by using the variable. Here is a simple example in Python.



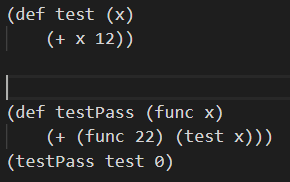
Answer the following questions:

Write a similar example in **eL** and test to see if it works. Explain what happened.

It worked! It set y to have the parameters and body of the function test. Then when y was called with parameter 22, it used the saved function definition and executed to return 34.

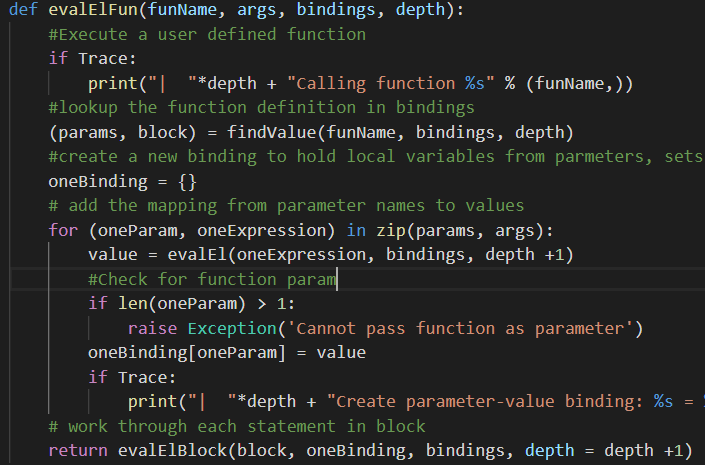
Write another example where one of the arguments to an **eL** function called **testPass** is another function that is called within **testPass**. Write some code that tests this. Explain what happened.

This also worked for me. It looked up func and had assigned it to the definition for test. Test also was found and called as normal.

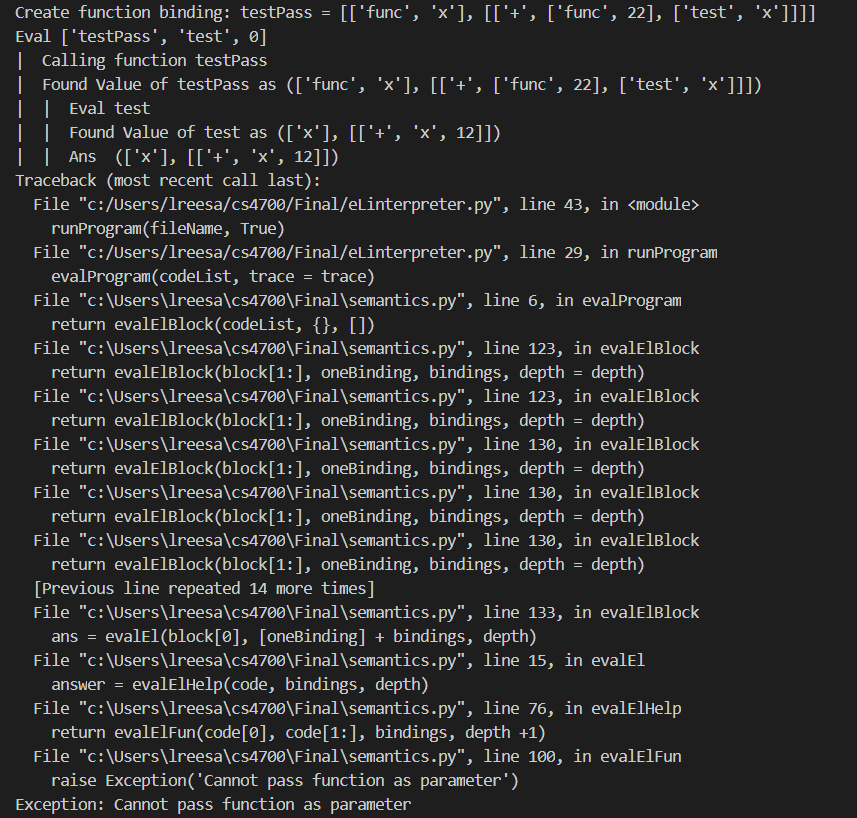


What changes would be needed to the **eL** interpreter code to disallow this feature? Show the code snippets below and an example run that shows the new semantics where passing functions as values causes an error.

Code:



Run same code, this time error thrown



**Part Two: General programming language questions from the book and slides (4 pts each)**

1. Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in Fortran. Develop the access functions for both of these arrangements for three-dimensional arrays. (Ch. 6)

http://sce2.umkc.edu/csee/leeyu/class/CS441/HW/hw2-sol.pdf

Subscript ranges for 3d: min(1), min(2), min(3), max(1), max(2), max(3)

Sizes of subscript ranges: size(1), size(2), size(3)

Element size = 1

Row Major:

location(a[i,j,k]) = (addr of a[min(1),min(2),min(3)) + (i-min(1))size(3) + (j-min(2))size(2) + (k-min(3))

Column Major:

location(a[i,j,k]) = (addr of a[min(1),min(2),min(3)) + (k-min(3))size(1) + (j-min(2))size(2) + (i-min(1))

1. Write a short discussion of what was lost and what was gained in Java’ designers’ decision to not include the pointers of C++. (Ch. 6)

The first thing that comes to mind when thinking of pointers is the security flaw inherent in them. So by leaving out pointers, Java designers avoided issues such as memory corruption and errors, and stack overruns. One major thing Java designers missed out on is the memory efficiency and speed that pointers allow, when programmers can optimize code.

1. What are the arguments both for and against the exclusive use of Boolean expressions in the control statements in Java (as opposed to also allowing arithmetic expressions, as in C++)? (Ch. 7)

The argument for exclusively using Boolean expressions in control statements is that typing errors are avoided, resulting in more reliability. The argument against the restriction is that it does decrease the level of writability, as there are a wide range of types not allowed in a control statement.

1. Assume the following rules of associativity and precedence for expressions: (Ch 7)

Precedence Highest \*, /, not

+, –, &, mod

(unary)

=, /=, < , <=, >=, >

and

Lowest or, xor

Associativity Left to right

Show the order of evaluation of the following expressions by parenthesizing all subexpressions and placing a superscript on the right parenthesis to indicate order. For example, for the expression a + b \* c + d the order of evaluation would be represented as ((a + (b \* c)1)2 + d)3

a \* b - 1 + c

(((a \* b)^1 – 1)^2 + c) ^ 3

a \* (b - 1) / c mod d

((((a \* (b – 1) ^1)^2 / c)^3 % d)^4

\*Mod was not listed so I assumed the same precedence as /.

(a - b) / c & (d \* e / a - 3)

(((a - b)^1 / c)^2 & ((d \* (e / a)^3)^4 – 3)^5)^6

-a or c = d and e

((- a)^1 or ((c = d)^2 and e)^3)^4

a > b xor c or d <= 17

(((a > b)^1 xor c)^3 or (d <= 17)^2)^4

-a + b

(- (a + b)^1)^2

1. What are the arguments, pro and con, for Python’s use of indentation to specify compound statements in control statements? (Ch 8)

The obvious reason for Python’s use of indentation is that it forces programmers to write readable code. One con I have personally found is the difference between using tabs and 4 spaces. Mixing them up causes annoying bugs.

1. What are arguments for and against a user program building additional definitions for existing operators, as can be done in Python and C++? Do you think such user-defined operator overloading is good or bad? Support your answer. (Ch 9)

An argument for the user program building additional definitions for existing operators is that it allows for programmers to build programs more customized to certain domains. A con is that users can create things that aren’t necessarily correct, or against logical rules.

1. What are at least two arguments against the use of pass-by-name parameters? (Ch 9)

First, pass-by-name parameters are difficult to implement because of the particular procedures that have to be implemented a specific way. Secondly, it requires repeated evaluation of arguments, which can be very inefficient.

1. Show the stack with all activation record instances, including the dynamic chain, when execution reaches position 1 in the following skeletal program. This program uses the deep-access method to implement dynamic scoping. (Ch 10)

void fun1() {

float a;

. . .

}

void fun2() {

int b, c;

. . .

}

void fun3() {

float d;

. . . 🡨 1

}

void main() {

char e, f, g;

. . .

}

The calling sequence for this program for execution to reach fun3 is

main calls fun2

fun2 calls fun1

fun1 calls fun1

fun1 calls fun3

Activation Record Instances for:

fun3: local d

dynamic link, returns to fun1

fun1: local a

dynamic link, returns to fun2

fun2: local b, c

dynamic link, returns to main

Main: Local e, f, g

1. Some software engineers believe that all imported entities should be qualified by the name of the exporting program unit. Do you agree? Support your answer. (Ch 11)

When imported entities are qualified by the name of the exporting program unit, it requires the programmer to understand in a small way the dependencies of the package. It also allows for some optimization in the way that only needed packages are installed. For these reasons I agree that all imported entities should be qualified by the name of the exporting program unit.

1. Write an analysis of the similarities of and differences between Java packages and C++ namespaces. (Ch 11)

Java packages and C++ namespaces both are mechanisms for organizing classes. In namespaces, one can place each library in its own namespace and qualify the names in the program with the name of the namespace when used outside their namespace. Packages on the other hand allow classes to download faster as a group and organize them according to category or functionality.

1. Why are destructors rarely used in Java but essential in C++? (Ch 11)

Java has built-in garbage collection, and C++ doesn’t have any of that functionality, so destructors serve that purpose.

1. Explain why naming encapsulations are important for developing large programs. (Ch 11)

Encapsulation provides for logical organization of programs, security for private objects or classes, prevents library name conflicts, and allows for partial recompilation.

1. What is the primary reason why all Java objects have a common ancestor? (Ch 12)

The main reason for Java objects all having a common ancestor is to give every class some basic functionality, as well as have uniformity and clear organization.

1. Find a tutorial on COBOL and determine how exception handling is done in COBOL programs. (Ch 14)

ILE COBOL allows for error-handling APIs, initiating deliberate dumps, or other user-written error-handling routines. It can handle errors in string, arithmetic, input-output and soft-merge operations as well as exceptions on the CALL statement.

1. A functional language could use some data structure other than the list. For example, it could use strings of single-character symbols. What primitives would such a language have in place of the CAR, CDR, and CONS primitives of Scheme? (Ch 15)

Considering CAR stands for ‘contents of address register’ and CDR stands for ‘contents of decrement register’, these could still be valid inputs to a CONS (two-field record) as single-character inputs, to create a string, which is essentially a list of single-character symbols. Instead of having CAR and CDR ‘point’ to other elements of a list, you would simply concatenate and/or edit strings.