Jaime Guevara

CS-354 Sec. 1: TA1

September 22, 2018

* 1. a) Lexical error – (Java) passing an @ symbol as an integer to the scanner in Java

b) Syntax error – (Java) strings are defined by “” if we were to print out a line using System.out.println() we would need to use quotes otherwise the parser will complain and throw an error.

c) Semantic error – (Java) when trying to compare values such as integers by using == is semantically wrong because it’s not comparing whether or not they have the same value, but rather whether they point to the same address, therefore in order to correct this error “.equals()” must be used to compare values.

d) Dynamic semantic error – (Java) when using a for-loop one can cause a dynamic semantic error if you set the stop point beyond the length of the array. (i.e. you try to access a non-existent array cell and trigger an OutOfBoundsException).

e) An error that the compiler can neither catch nor generate code to catch – (Java, though any language really) these are usually errors that don’t violate syntax, lexicon, or the dynamic of the language. An easy example is when a value is meant to change each iteration of a loop, but it never changes (i.e. a Logical error).

1.8) The make systems has its inaccuracies as the way in which its syntax is described, the target depends on the dependencies in order to be compiled, but even if only one of the files has changed, then the target must also be changed and recompiled. However, with the way that make works, one could put the same target in two separate lines and tell it to compile from both. However this is wrong, because if the source code depends on one of the files not in line with it, then make will try to compile anyway just to run into an error. Likewise, if the change done was only in including one more library/piece of code, make doesn’t just go out to look for that one needed library but for **all** libraries and any such other dependencies needed. Basically, it recompiles everything from scratch each time.

2.1) a) String in C – “Hello, I am a \”string\”\n” – should print out *Hello, I am a*  \_\_*“string”* and a newline.

b) Pascal comment – {I am a comment} or (\* I am a comment \*)

c) Numeric constants in C – const double c = 0.45;

int hex=0xFAFA;

d) Floating-point constant in Ada – const real height := 3\_1.93e3

e) Inexact constants in Scheme – 3.###

f) Financial quantities in American notation - $\*\*11,000.00

2.13) a) Construct a parse tree for foo(a, b):

b) stmt **->** subr\_call **->** id(foo) ( arg\_list ) **->** id(foo) (expr args\_tail) **->** id(foo) (expr ,arg\_list) **->** id(foo) (expr , expr args\_tail) **->** id(foo) (expr , expr ) **->** id(foo) (expr , primary expr\_tail ) **->** id(foo) (expr, primary ) **->** id(foo) (expr , id(b) ) **->** id(foo) (primary expr\_tail , id(b) ) **->** id(foo) (primary , id(b) ) **->** id(foo) (id(a) , id(b) )

2.17) *if -> factor comp\_op factor*

*while -> factor comp\_op factor* do *stmt\_list $$*

*comp\_op -> <*

*comp\_op -> >*

*comp\_op -> ==*

*comp\_op -> <=*

*comp\_op -> >=*

*comp\_op -> !=*