Trevor Smith TA1

CS 354

2/22/2021

1.1) Using Java for each question part

A. Lexical error: Attempting to pass in a non-numerical character as an integer to the scanner.
B. Syntax error: Calling `System.out.print` instead of `System.out.print()` would be a syntax error.
C. Static semantic error: Using == when you should be using `.equals()` would be an example of a static semantic error.
D. Dynamic semantic error: An `IndexOutOfBoundsException` would be a result of a dynamic semantic error, where someone may accidentally try to reach beyond the length of an array.
E. An error that the compiler would not be able to catch would be a logical error. One logical error, for example, is an off-by-one error, where an iterator is accidentally incremented one too many or too few times.

1.8) Make has its inefficiencies and inaccuracies in the sense that it only recompiles in regards to dependencies between files rather than dependencies between specific code snippets within said files. Say file A depends on file B, and there is a small change in file B which does not actually have an effect on file A. However, since file A depends on file B as a whole, everything must be recompiled.

2.1) Regular expressions

A. String in C -> “(Char | Pair)*”
   Char -> any character except newline, “, or \
   Pair -> \” | \\
B. Pascal comments -> {?*} + ‘(*’ ?* ’*)’
   ? -> any character or number
C. Numeric constants in C -
   validOctal : 0 || 1 || 2 || 3 || 4 || 5 || 6 || 7
   octal : ‘0’validOctal*
   validHex : A || B || C || D || E || F
   | a || b || c || d || e || f
   | validOctal || 8 || 9
   hex : ‘0x’validHex*
   | ‘0X’validHex*

D. Inexact constants in Scheme -> digit* #* (#* | e) | digit* .digit #*

E. Financial quantities in American notation -> $ **(0 | NonZeroDigit(e | digit | digit digit) group*) (e | .digit digit)

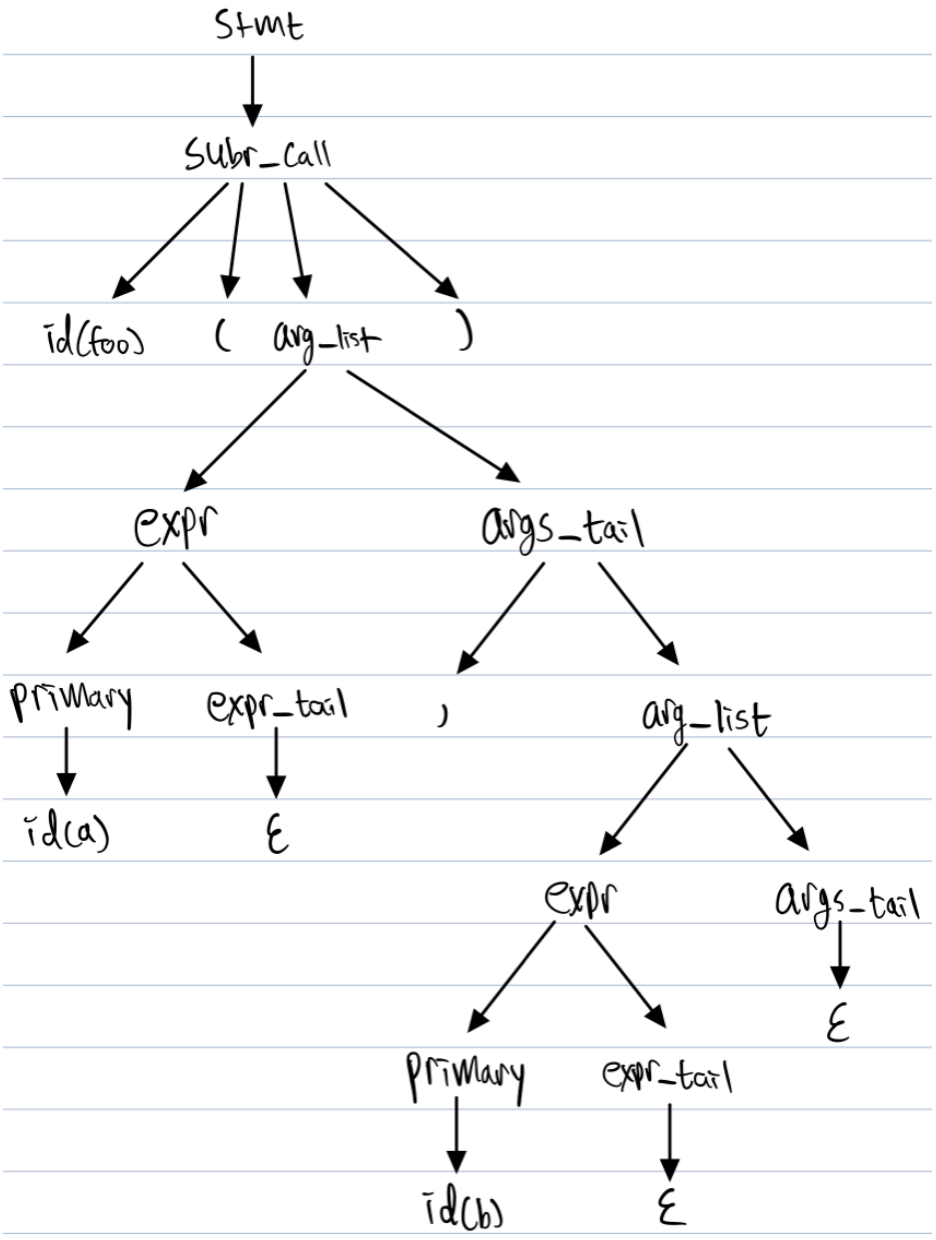
group -> , digit digit digit

digit -> 0 | NonZeroDigit

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

2.13)

(a)



Trevor Smith TA1

CS 354

2/22/2021

(b)

stmt => subr_call
=> id(arg_list)
=> id(expr args_tail)
=> id(expr, arg_list)
=> id(expr, expr args_tail)
=> id(expr, expr e)
=> id(expr, primary expr_tail e)
=> id(expr, primary expr_tail)
=> id(expr, primary e)
=> id(expr, primary)
=> id(expr, id)
=> id(primary expr_tail, id)
=> id(primary e, id)
=> id(primary, id)
=> id(id, id)
(foo) (a) (b)
foo(a, b)

2.17)

stmt —> if condition then stmt_list fi
—> while condition do stmt_list od
condition —> expr relation expr
relation —> <
—> >
—> ≤
—> ≥
—> =
—> ≠