Caitlyn Goetz

Textbook Assignment 1

February 9, 2016

Edition 3

1.1 a) In C, using a letter in hex that isn't recognized as hex code such as 0x89abr. The r would throw a lexical error because the scanner would not know what to do with it.

b) In C, a common syntax error that is the bane of many programmers existence is when there is a missing semicolon at the end of a statement.

c) In C, something that would cause a static semantic error would be trying to pass a float value into an array where the return and acceptance values are not the same. Or forgetting to declare a variable.

d) In C, a dynamic semantic error would be caused by trying to access something that is outside of the index of an array or string.

e) In Java, a hard to detect error would be one where the name of a method is being used as the name of a variable.

1.6 This seems fairly good in the very beginning when you are adding new features and after you have finished the basic features of the program and you are adding new methods to make the program support some other features. However, if you are simply changing variable values or names, little stuff like that, recompiling the whole thing would waste a lot of time since the only things that are changing would be the values and names in the one program. If you added a change to a file and forgot to make another file dependent on that file then the program would fail to upgrade that "parent" file and your program would not work as you expected it would.

2.1 a) Strings in C. \”(\\.|[^”])*\”

b) Comments in Pascal. \(*(.*?)*\) or \{(.*?)\}

c) Numeric constants in C.

constant -> int_constant | floatpt_constant

int_constant -> (octal_int | dec_int | hex_int) int_suffix

octal_int -> 0 oct_digit*

dec_int -> nonzero_digit dec_digit*

hex_int -> (0x | 0X) hex_digit hex_digit*

octal_digit -> 0|1|2|3|4|5|6|7

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

dec_digit -> 0 | nonzero_digit

hex_digit -> dec_digit |a|b|c|d|e|f|A|B|C|D|E|F

int_suffix -> ^| u_suffix (l_suffix | ll_suffix | ^) | l_suffix (u_suffix | ^) | ll_suffix (u_suffix | ^)

u_suffix -> u | U

l_suffix -> l | L

ll_suffix -> ll | LL

floatpt_constant -> (dec_floatpt | hex_floatpt)(f | F | l | L | ^)

dec_floatpt -> dec_digit dec_digit* (E|e)exponent | dec_digit* (.dec_digit | dec_digit.) dec_digit* ((E|e)exponent | ^)

hex_floatpt -> (0x |0X) hex_digit* (.hex_digit | ^| hex_digit.) hex_digit* (P | p)exponent

exponent -> (+ | - | ^) dec_digit dec_digit*

d) Floating-point constants in Ada. Int -> digit ((_|{})digit)*

extended digit -> digit|a|b|c|d|e|f|A|B|C|D|E|F

extended int -> extended digit ((_|{})extended digit)*

FP -> ((int((.int|{}))|(int # extended int((. Extended int|{})#))(((e|E)(+|-|{})int)|{})

e) Inexact constants in Scheme. Digit+ #* (. #* | {}) | digit* . digit+ #*

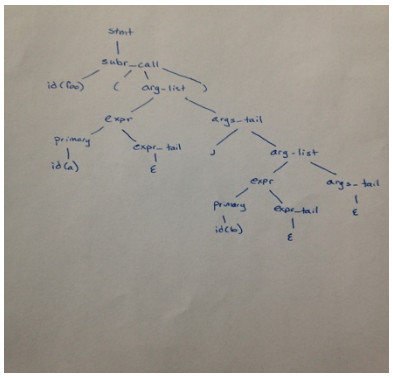
f) Financial quantities in American notation. Non zero digit -> 1|2|3|4|5|6|7|8|9

digit -> 0|non zero digit

group -> , digit digit digit

number -> $**(0|non zero digit ({}|digit|digitdigit) group*) ({}| .digit digit)

2.13 a)



b) stmt

subr call

id (arg list)

id (expr args tail)

id (expr , arg list)

id (expr , expr args tail)

id (expr , expr)

id (expr , primary expr tail) id (expr , primary)

id (expr , id)

id (primary expr tail , id) id (primary , id)

id (id , id)

c) The id is both of the first assignment and the first subr_call so it is in both of the stmt->assignment and stmt->subr_call. It is also in the first id and subr_call so it is also in the primary->id and primary->subr_call. Since that makes the stmt and primary not disjointed then the grammar is not LL(1).

d) stmt → id stmt_tail

stmt_tail → (arg_list)

→ := expr

primary → id primary_tail

→ (expr)

primary_tail → (arg_list)

→ ε

2.17 *program -> stmt_list $$*

stmt_list -> stmt_list stmt

stmt_list - > stmt

stmt -> id := expr

stmt -> read id

stmt -> write expr

expr -> term

expr -> expr add_op term

term -> factor

term -> term mult_op factor

factor -> (expr)

factor -> id

factor -> number

add_op -> +

add_op -> -

*multiplication_op -> **

multiplication_op -> /

stmt -> if condition then *stmt_list* fi

-> while condition do *stmt_list* od

condition -> expr relation expr

relation -> <

-> >

-> *<=*

-> *>=*

-> *=*

-> *!=*