Saman Rastgar TA-1
Due : Monday 09/24

Exercises 1.8

(Q1.1) Language Used=Java

a) A lexical error will occur in java, if you to assign a char to an integer.
b) A syntax error will occur in java, if you forget to include a semicolon at an end of a statement in java.
c) A static semantic error will occur in java, if you access a method that has not been declared in the class.
d) A dynamic semantic error will occur in java, if you access an element of an array that does not exist. Example: Index out of bound .
e) An error that the compiler can neither catch nor easily generate code to catch would be if we were to use a method name as a variable.

(Q1.8) Make Analysis
The make utility searches for the target's dependency and create a chain of dependency and it compiles the chain of file in the order they have been added.But this tool does not always recompile the previously compiled code.Dependency management uses timestamp to keep track of the compiled files and starts from the part that needed to be compiled.
~>As mentioned make looks for dependent targets but you can always include comment on the dependent file which does not affect the compile procedure(The dependency management described is not accurate).
~>compiling procedure for make file fails when it can't find the file description for dependent file.

Exercises 2.6

(Q2.1) Regular Expressions

a) Strings in C are captured by: string ? “”(!{\, “, \n} | \!{\n})* ”
b) Comments in Pascal are captured by: Pascal comment ? (* (!(*) | (*!())))* *+)

c) Numeric constants in C:

C_constant → int_const | fp_const
int_const → (oct | dec | hex) int_suffix
oct_int → 0 oct_digit*
dec_int → nonzero_digitdec_digit*
hex_int → (0x | 0X) hex_digithex_digit*
oct_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
dec_float → dec_digit*|.dec_digit*|E | e hex_float → e | ϵ exponent | ϵ type → long | unsigned
long | longlong |unsigned longlong exponent → + | - | ϵ unsigned → U | u float → F | f long → L |
l longlong → LL | ll

d) Floating point constant in Ada :

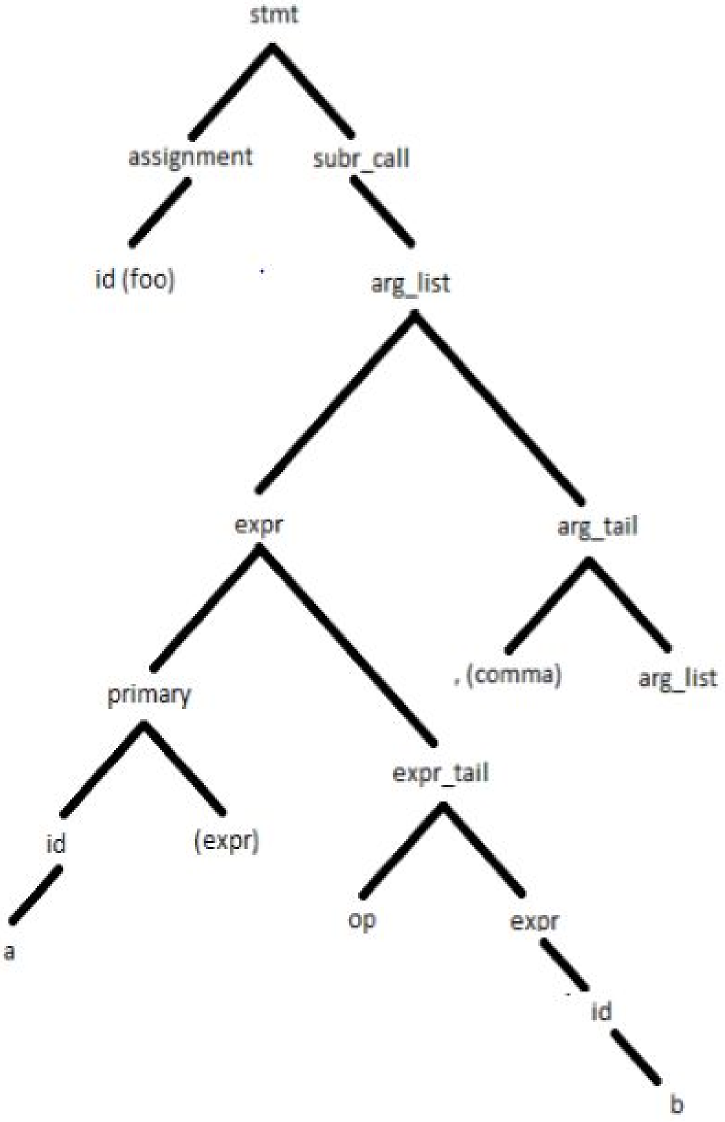
Ada_int → digit ((_|?) digit)*
Extended_digit → digit | A | a | B | b | C | c | D | d | E | e | F | f
Ada_extended_int → extended digit ((_|?) extended digit)*
Ada_FP_num → ((Ada_int ((.Ada_int | ?))
| (Ada_int# Ada_extended_int ((.Ada_extended_int) | ?) #))   (((e | E) (+ | - | ?) Ada_int) | ?)

e) Showing inexact constant in scheme:
Digit + # * (.# * | ϵ) | digit* .digit + # *

f) nonzero digit → 1|2|3|4|5|6|7|8|9
digit → 0 | nonzero digit
group_type → , digit
Number → $ * * (0 | nonzero digit (ϵ | digit | digit digit) group_type *) (ϵ | .digit digit

(Q2.13) Grammer interpretation

(a) Construct parse tree



(b) Give a canonical (rightmost) derivation of this same string.

stmt → assignment,subr_call
subr_call → arg_list
arg_list → expr,args_tail
args_tail →,,arg_list
expr → primary,expr_tail
expr_tail → op,expr
expr →id
expr →primary, expr_tail
primary →id,expr
assignment →id

(Q2.17)

program → stmt_list $$
stmt_list → stmt_list stmt
stmt_list → stmt
stmt → id:= expr
stmt → read it
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 → + | -
mult_op → * | /
stmt → if condition then stmt_list fi
→ while condition do stmt_list od
condition → expr relation expr
relation → <
→ >
→ <=
→ >=→ =
→ !=