Andy Luong CS 152 HW 1

1.	FORTRAN	LISP	COBOL	Simula	CLU	Smalltalk
Year Appeared	1957	1958	1959	1962	1975	1972
Creator(s)	IBM, John Backus	John McCarthy	Grace Hopper	Ole-Johan Dahl	Barbara Liskov	Alan Kay, Dan Ingalls, Adele Goldberg
New Feature	Derived type parameters	Common Lisp dialect and Artificial Intelligence programming	Data based off of Hierarchy	Introduced objects; precursor to object-oriented design	clusters (abstract), exception handling and iterators	object-oriented design, web development, virtual machine

Syntax	default = program, implicit none, end program
	
	define variables = “real : :”,
	
	comment = !
	
	Input
	------ 	
	program add
	implicit none
	
	real : a, b, sum
	a = 10
	b = 20
	sum = a + b
	print *, result
	
	end
		atom = string+number+special list = ()
		string = “”
		comment = ;
		
		Input
		--------- 	
		(write-line “Hello World”)
		(write (* 2 2))
		--------- 	
		
		Output
		-------- 	
		“Hello World”
		6
		------- 	
			hierarchy = struct lines end with ;
			
			Input
			------ 	
			struct Name {
			char first[10];
			char last[10]
			int age;
			};
			------ 	
				class, comment = !, BEGIN, OutText, END
				
				Input
				-------- 	
				BEGIN
				OutText("Hello World!");
				END
				-------- 	
				
				Output
				------ 	
				“Hello World”
				------- 	
					clusters are variables defined, proc, semicolons, end,
					
					Input
					------- 	
					complex = add
					rep = record [
					real: real,
					img: rea;]
					add = proc
					… end
					end complex
					------- 	
						class/object,
						methods, unary = “new”,
						binary = + - * /,
						keyword = new,
						
						Input
						----- 	
						Object class:
						push : item
						5 * 7
						------ 	

	program add ------- Output ------ 30 ------					

2. Sec 1.8 1.1
Java
(a) A lexical error, detected by the scanner = ‘!a’ since ! cannot start a string

(b) A syntax error, detected by the parser = no ‘;’ at the end of a line

(c) A static semantic error, detected by semantic analysis = using a variable that has not been correctly initialized like boolean x = 5.

(d) A dynamic semantic error, detected by code generated by the compiler = calling an element not within the array scope (ArrayIndexOutOfBounds)

(e) An error that the compiler can neither catch nor easily generate code to catch (this should be a violation of the language definition, not just a program bug) = using the name of a variable that is the same name of a method

3. Sec 2.6 2.1
Write regular expressions to capture the following.
(a) Strings in C.
These are delimited by double quotes (") = [“][“]
and may not contain newline characters. = [^\\n]
They may contain double-quote or backslash characters if and**or** only if those characters are “escaped” by a preceding backslash. = (|\\.|)*
["]([^\\n](|//.))*["]

(b) Comments in Pascal.
These are delimited by (* and *) =\(*(.|[\r\n])*?*\)
or by { and }. = \{(.|[\r\n])*?\}
They are not permitted to nest.
\(*(.|[\r\n])*?*\)|\{(.|[\r\n])*?\}

(c) Numeric constants in C.
[U|u|L|l|LL|ll]? = Integers = unsigned, long, long long

Octal = 0[0-7]*, Integer = [U|u|L|l|LL|ll]?

Octal Integer = 0[0-7]*[U|u|L|l|LL|ll]?

Decimal = [1-9][0-9]**, Integer = [U|u|L|l|LL|ll]?
Decimal Integer = [1-9][0-9]*[U|u|L|l|LL|ll]?

Hexadecimal = 0[Xx][0-9a-fA-F]+, Integer = [U|u|L|l|LL|ll]?
Hexadecimal Integer = 0[Xx][0-9a-fA-F]+[U|u|L|l|LL|ll]?

Decimal (FPN) = 0-9]*[.][0-9]+[eE], Floating Point Numbers = [+-]?[0-9]+[F|f|L|l]?
Decimal Floating Point Numbers = [0-9]*[.][0-9]+[eE][+-]?[0-9]+[F|f|L|l]?

Hexadecimal (FPN) = [0-9]*([.][0-9]+)?[pP], Floating Point Numbers = [+-]?[0-9]+[F|f|L|l]?
Hexadecimal Floating Point Numbers = [0-9]*([.][0-9]+)?[pP][+-]?[0-9]+[F|f|L|l]?

(d) Floating-point constants in Ada. These match the definition of real in Example 2.3, except that
(1) a digit is required on both sides of the decimal point,
(2) an underscore is permitted between digits, and
(3) an alternative numeric base may be specified by surrounding the nonexponent part of the number with pound signs, preceded by a base in decimal (e.g., 16#6.a7#e+2). In this latter case, the letters a . . f (both upper- and lowercase) are permitted as digits. Use of these letters in an inappropriate (e.g., decimal) number is an error, but need not be caught by the scanner.

Ada int = digit((_|E)digit)*
extended digit = digit a | b | c | d | e | f | A | B | C | D | E | F
Ada extended int = extended digit((_|E)extended digit)*

(e) Inexact constants in Scheme. Scheme allows real numbers to be explicitly inexact (imprecise). A programmer who wants to express all constants using the same number of characters can use sharp signs (#) in place of any lower-significance digits whose values are not known. A base-10 constant without exponent consists of one or more digits followed by zero of more sharp signs. An optional decimal point can be placed at the beginning, the end, or anywhere in-between. (For the record, numbers in Scheme are actually a good bit more complicated than this. For the purposes of this exercise, please ignore anything you may know about sign, exponent, radix, exactness and length specifiers, and complex or rational values.)

digit+#*(.#*)digit*.digit+ # *

(f) Financial quantities in American notation. These have a leading dollar sign ($), an optional string of asterisks (*—used on checks to discourage fraud), a string of decimal digits, and an optional fractional part consisting of a decimal point (.) and two decimal digits. The string of digits to the left of the decimal point may consist of a single zero (0). Otherwise it must not start with a zero. If there are more than three digits to the left of the decimal point, groups of three (counting from the right) must be separated by commas (,). Example: $**2,345.67. (Feel free to use “productions” to define abbreviations, so long as the language remains regular.)

nzdigit = 1 2 3 4 5 4 7 8 9
digit = 0 nzdigit
group = , digit digit digit
number = $ * * (0 nzdigit (E | digit | digit digit) group*) (E | . digit digit)