CS 4240: Compilers and Interpreters, Homework 3, Spring 2019
Assigned: Wednesday, April 10, 2019 (3 problems, 100 points total)
Due in Canvas by 11:59pm on Tuesday, April 23rd
Automatic penalty-free extension until 11:59pm on Friday, April 26th
(All homeworks must be done independently. You may post clarification questions on Piazza, but be sure to make your post instructor-only if it includes part of your solution.)

Question 1: Regular Expressions **[30 points]**

For each of the languages described below in prose, write a regular expression that matches the language, or explain why no such regular expression exists.

a) All numerals in base 3 denoting unsigned integers in which successive digits do not increase and there are no unnecessary leading zeros. **[15 points]**
For example, the following numerals are in the language: 210, 22110, and 0. The following numerals are not in the language: 122, 010, and 00.

b) A language of program identifiers for a conventional programming language over the alphabet of lowercase and uppercase letters, digits, and the underscore. An identifier must start with a letter (lower or upper case) or an underscore. If an identifier starts with an underscore, that first symbol must be followed with either a letter or a digit. **[15 points]**
For example, the following strings are in the language: a_0, b10, and _a_0. The following strings are not in the language: 2a, 4_, and __a.

Sample Answer

a) 0 | 1⁺0* | 2⁺1*0*

b) ([a − zA − Z] | _ [a − zA − Z0 − 9]) ([a − zA − Z0 − 9_])*

Question 2: Context Free Grammars **[30 points]**

The following grammar G defines a language of simple imperative programs, akin to subset of the Tiger language. Each program consists of a sequence of formal parameters, a single statement that declares a sequence of variables, and a single statement that assigns the result of an expression to a variable. Terminal symbols are typeset in bold font and non-terminal symbols are typeset in italics. Note that only nonterminals can occur on the left side of a production.

imp → **main (** *vars* **) {** *decl assign* **}**	(1)
vars → ε	(2)
vars → *nevars*	(3)
nevars → **var**	(4)
nevars → *nevars* **, var**	(5)
decl → *type nevars* **;**	(6)
assign → **var =** *expr* **;**	(7)
expr → **var**	(8)
expr → **var** *op expr*	(9)
op → +	(10)
op → *	(11)
type → **int**	(12)
type → **float**	(13)

Your task is to generalize G to a grammar G′ that describes programs in which:

1. The body of main contains a sequence of declaration statements. **[15 points]**
2. The body of main contains a sequence of assignment statements. **[15 points]**

Your grammar G′ should be able to derive programs with multiple declaration statements and multiple assignment statements, such as the following program:

main () { int i, j, k; float f, g, h; i = j + k; f = g * h; }

Note that we are ignoring semantic errors (such as accesses to uninitialized variables) in this program example, and only focusing on the program's syntactic structure. Any solution that supports a sequence of declaration statements followed by a sequence of assignment statements is fine. However, we will not deduct points if someone submits a solution with interleaved declarations and assignments.

Sample Answer

$$
\begin{aligned}
\text{imp} &\rightarrow \texttt{main(vars)\{decls assigns\}} \\
\text{vars} &\rightarrow \epsilon \\
\text{vars} &\rightarrow \text{nevars} \\
\text{nevars} &\rightarrow \texttt{var} \\
\text{nevars} &\rightarrow \text{nevars, } \texttt{var} \\
\text{decls} &\rightarrow \epsilon \\
\text{decls} &\rightarrow \texttt{type nevars; decls} \\
\text{assigns} &\rightarrow \epsilon \\
\text{assigns} &\rightarrow \texttt{var} = \text{expr; assigns} \\
\text{expr} &\rightarrow \texttt{var} \\
\text{expr} &\rightarrow \texttt{varop expr} \\
\text{op} &\rightarrow + \\
\text{op} &\rightarrow * \\
\text{type} &\rightarrow \texttt{int} \\
\text{type} &\rightarrow \texttt{float}
\end{aligned}
$$

There was some ambiguity in the problem statement with respect to whether zero-length sequences are permitted, and whether declarations and assignments can be interleaved, so answers with reasonable assumptions will all be considered correct.
The prefix "ne" usually stands for "non-empty".

Question 3: LL(1) Parsers **[40 points]**

a) Is grammar G in Question 2 an LL(1) grammar? If so, explain why. If not, modify grammar G so as to obtain grammar G′′ that is LL(1) and derives exactly the same language as grammar G. **[10 points]**

b) Show all the FIRST and FOLLOW sets for grammar G or G′′ (if you created one) as illustrated in Lecture 22. **[10 points]**

c) Show all the FIRST+ sets for grammar G or G′′ (if you created one) as illustrated in Lecture 22. **[10 points]**

d) Show the LL(1) parsing table for grammar G or G′′ (if you created one) as illustrated in Lecture 22. **[10 points]**

Sample Answer

a) Grammar G from Question 2 is not LL(1) due to the following issues.

A. The two *'nevars'* productions (4) and (5) both have *'var'* in their FIRST+ sets.

B. The two *'expr'* productions (8) and (9) both have *'var'* in their FIRST+ sets.

C. It is also a left-recursive grammar.

Productions in an LL(1) grammar with the same LHS must not have common elements in their FIRST+ sets. Below is a sample grammar G'' that is LL(1) and derives the same language as grammar G.

imp → *main*(vars){decl assign}	(1)
vars → ϵ	(2)
vars → nevars	(3)
nevars → *var* nevars'	(4)
nevars' → ϵ	(5)
nevars' →, nevars	(6)
decl → type nevars;	(7)
assign → *var* = expr;	(8)
expr → *var* expr'	(9)
expr' → ϵ	(10)
expr' → op expr	(11)
op → +	(12)
op → *	(13)
type → *int*	(14)
type → *float*	(15)

b) Below are the FIRST and FOLLOW sets for the nonterminals in the sample grammar G''. Following the definitions of FIRST sets and FOLLOW sets on slide 19 from lecture 22, the FIRST sets of a terminal variable is a singleton set containing itself, and FOLLOW sets for terminal variables are undefined.

Symbol	FIRST	FOLLOW
imp	main	EOF
vars	ϵ var	)
nevars	var	) ;
nevars'	ϵ ,	) ;
decl	int float	var
assign	var	}
expr	var	;
expr'	ϵ + *	;
op	+ *	var
type	int float	var

c) The FIRST+ sets for the sample grammar G'' are as follows (set elements are separated by spaces):

Production	**FIRST$^+$**
imp → `main`(vars){decl assign}	`main`
vars → ϵ	ϵ)
vars → nevars	`var`
nevars → `var` nevars'	`var`
nevars' → ϵ	ϵ) ;
nevars' →, nevars	,
decl → type nevars;	`int float`
assign → `var` = expr;	`var`
expr → `var` expr'	`var`
expr' → ϵ	ϵ ;
expr' → op expr	+ *
op → +	+
op → *	*
type → `int`	`int`
type → `float`	`float`

d) The LL(1) parsing table for grammar G'' is as follows. The parsing table should not have a column for ε.

	EOF	,	;	(	)	{	}	+	*	=	int	float	main	var
imp													1	
vars					2									3
nevars														4
nevars'		6	5		5									
decl											7	7		
assign														8
expr														9
expr'			10					11	11					
op								12	13					
type											14	15		