October 19, 2016.

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México
División de Diseño, Ingeniería y Arquitectura
Departamento de Tecnologías de Información y Computación

Compiler Design Second Exam

Instructor: Ariel Ortiz Course Number and Section: Tc3048.1

Name: ______________________________ Student ID: ______________

Apegándome al Código de Ética de los Estudiantes del Tecnológico de Monterrey, me comprometo a que mi actuación en este examen esté regida por la honestidad académica. En congruencia con el compromiso adquirido con dicho código, realizaré este examen de forma honesta y personal, para reflejar, a través de él, mi conocimiento y aceptar, posteriormente, la evaluación obtenida.

Firma: ______________________

VERY IMPORTANT: This exam should be solved individually. Any evidence of cheating or fraud will be punished with a DA (Academic Dishonesty) grade. This punishment is for both the person who copies and for the person who allows to be copied.

General Instructions

Download the exam related file `exam2.zip` from the course web site. This ZIP contains several program source files, yet you must only modify the files called `English.cs` and `sexp.y`. **Type your name and student ID in a comment at the top of these two source files.**

The corresponding executable files will be built using the provided `Makefile`.

Once you finished your exam, create a compressed ZIP file called `exam_solution.zip` containing only your modified `English.cs` and `sexp.y` files. Upload the ZIP file using the course website.

Time limit: 90 minutes.

Problems

1. **(60%)** You have the following BNF grammar for a tiny portion of the English language:

$$
\begin{aligned}
&\langle \textit{sentence} \rangle \rightarrow \langle \textit{noun-phrase} \rangle\ \langle \textit{verb-phrase} \rangle\ \textbf{EOF}\\
&\langle \textit{noun-phrase} \rangle \rightarrow \textbf{ARTICLE}\ \langle \textit{adj-list} \rangle\ \textbf{NOUN}\ \langle \textit{pp-list} \rangle\\
&\langle \textit{noun-phrase} \rangle \rightarrow \textbf{NAME}\\
&\langle \textit{noun-phrase} \rangle \rightarrow \textbf{PRONOUN}\\
&\langle \textit{verb-phrase} \rangle \rightarrow \textbf{VERB}\ \langle \textit{noun-phrase} \rangle\ \langle \textit{pp-list} \rangle\\
&\langle \textit{adj-list} \rangle \rightarrow \epsilon\\
&\langle \textit{adj-list} \rangle \rightarrow \langle \textit{adj-list} \rangle\ \textbf{ADJ}\\
&\langle \textit{pp-list} \rangle \rightarrow \epsilon\\
&\langle \textit{pp-list} \rangle \rightarrow \langle \textit{pp-list} \rangle\ \langle \textit{pp} \rangle\\
&\langle \textit{pp} \rangle \rightarrow \textbf{PREP}\ \langle \textit{noun-phrase} \rangle
\end{aligned}
$$

Complete the `Parser` class in the `English.cs` source file in order to have a *recursive descent parser* that checks the syntax of English sentences according to the above grammar (remember to translate it first into an LL(1) grammar). Note that the symbols written in bold uppercase are *terminal symbols* that correspond to the token categories recognized by the provided lexical analyzer (the `Scanner` class).

The following table shows some examples of some inputs and their expected outputs:

Input	**Output**
`the man saw the woman`	`Syntax fine.`
`dick took it to jane`	`Syntax fine.`
`a big blue ball hit the little green man`	`Syntax fine.`
`janet liked the blue man on the little table with the big green ball`	`Syntax fine.`
`the man`	`Bad syntax.`
`man saw woman`	`Bad syntax.`
`blue man hit mark`	`Bad syntax.`
`a dick took those to the little woman`	`Bad syntax.`

2. **(40%)** The files `sexp.y` and `sexp_lex.l` are *bison* and *flex* files that implement a simple s-expression (sort of like Lisp) interpreter, fairly similar to the one built in our previous class. It only supports addition and multiplication of integer numbers. The plus (`+`) and times (`*`) operators only accept two operands. Thus, an expression like "`(+ 1 2)`" is valid, but "`(+ 1 2 3)`" is not.

   Modify the code in `sexp.y` so that the plus and times operators can accept a variable number of operands. The minimum number of operands should be zero, and there should be no upper limit. The plus operator with no operands should evaluate to zero, while the times operator with no operands should evaluate to one.

   Examples:

Expression	**Result**
42	42
(+)	0
(*)	1
(+ 2)	2
(* 5)	5
(+ 2 3)	5
(* 2 3)	6
(+ 1 2 3)	6
(* 1 2 3)	6
(+ 1 2 3 4 5 6 7 8 9 10)	55
(* 1 2 3 4 5 6 7 8 9 10)	3628800
(* (* (+ (*) (*)) (+ (*) (*) (*))) (+ (*) (*) (*) (*) (*) (*) (*)))	42