CS 355 Programming Language Design
Homework 2: Turtle Redesign and Implementation
Due 03/10/14

1. Introduction

For this programming project, you will modify the turtle programming language and modify its interpreter (*turtle*) that you have implemented or going to complete based on its feedback in homework 1. The program will read the input source code from standard input and write a Portable Gray Map (PGM) image to standard output. This image contains the line drawing specified by the input program. You can view the image by standard image viewers.

We will add one new feature for Turtle about its declaration and scoping. In particular, you need to change Turtle grammar and the interpreter to support the following.

Declaration:
- Declare before use (like in C)
- Declare at the beginning of the program or a code block

Static Scoping:
- On block entry, all declarations of that block are processed and bindings added to symbol table
- On block exit, bindings are removed, restoring any previous bindings that may have existed

2. Turtle Programming Language

The grammar for our high-level turtle language is given in Figure 1. Curly braces {} and square brackets [] are used as meta-symbols as described in the caption to make the grammar more compact (they also hint at iteration and branching respectively in the corresponding recursive descent subroutines).

$$
\begin{array}{rcll}
\textit{program} & \rightarrow & \textit{stmt\_seq}\ \$ & (1)\\
\textit{stmt\_seq} & \rightarrow & \textit{stmt}\ \{\textit{stmt}\} & (2)\\
\textit{stmt} & \rightarrow & \textit{assign} \mid \textit{while\_stmt} \mid \textit{if\_stmt} \mid \textit{action} & (3)\\
\textit{assign} & \rightarrow & \texttt{IDENT ASSIGN}\ \textit{expr} & (4)\\
\textit{block} & \rightarrow & \textit{stmt}\ \{\textit{stmt}\} & (5)\\
\textit{while\_stmt} & \rightarrow & \texttt{WHILE}\ \textit{bool}\ \texttt{DO}\ \textit{block}\ \texttt{OD} & (6)\\
\textit{if\_stmt} & \rightarrow & \texttt{IF}\ \textit{bool}\ \texttt{THEN}\ \textit{block}\ \{\texttt{ELSIF}\ \textit{bool}\ \texttt{THEN}\ \textit{block}\}\ [\texttt{ELSE}\ \textit{block}]\ \texttt{FI} & (7)\\
\textit{action} & \rightarrow & \texttt{HOME} \mid \texttt{PENUP} \mid \texttt{PENDOWN} \mid \texttt{FORWARD}\ \textit{expr} & (8)\\
 & \rightarrow & \texttt{LEFT}\ \textit{expr} \mid \texttt{RIGHT}\ \textit{expr} \mid \texttt{PUSHSTATE} \mid \texttt{POPSTATE} & (9)\\
\textit{expr} & \rightarrow & \textit{term}\ \{+\ \textit{term} \mid -\ \textit{term}\} & (10)\\
\textit{term} & \rightarrow & \textit{factor}\ \{*\ \textit{factor} \mid /\ \textit{factor}\} & (11)\\
\textit{factor} & \rightarrow & -\ \textit{factor} \mid +\ \textit{factor} \mid (\ \textit{expr}\ ) \mid \texttt{IDENT} \mid \texttt{REAL} & (12)\\
\textit{bool} & \rightarrow & \textit{bool\_term}\ \{\texttt{OR}\ \textit{bool\_term}\} & (13)\\
\textit{bool\_term} & \rightarrow & \textit{bool\_factor}\ \{\texttt{AND}\ \textit{bool\_factor}\} & (14)\\
\textit{bool\_factor} & \rightarrow & \texttt{NOT}\ \textit{bool\_factor} \mid (\ \textit{bool}\ ) \mid \textit{cmp} & (15)\\
\textit{cmp} & \rightarrow & \textit{expr}\ \textit{cmp\_op}\ \textit{expr} & (16)\\
\textit{cmp\_op} & \rightarrow & = \mid \texttt{NE} \mid < \mid \texttt{LE} \mid > \mid \texttt{GE} & (17)
\end{array}
$$

Figure 1: Grammar for Turtle language. ***{a}*** *specifies that* ***a*** *can occur zero or more times and [****a****] denotes that* ***a*** *is optional. The* ***$*** *in the first production indicates that there should be no more tokens following stmt_seq****.*** *Production 2 represents a sequence of top-level statements, whereas Production 5 denotes a list*

of statements nested inside another statement construct.

The terminals (i.e., tokens) of the languages are +, -, *, /, (,), =, <, > , and the multi- character tokens are listed in Table 1. The reserved words are given in Table 2. The remainder of a line is ignored following a # symbol which allows the programmer to insert comments into the source code.

token	*regular expression*
`IDENT`	`[a-zA-Z\_][a-zA-Z0-9\_]*`
`REAL`	`[0-9]+([\.][0-9]*)?`
`ASSIGN`	`:=`
`NE`	`<>`
`LE`	`<=`
`GE`	`>=`

Table 1: Multi-character tokens.

OR	AND	NOT	WHILE
DO	OD	IF	THEN
ELSIF	ELSE	FI	HOME
PENDOWN	PENUP	FORWARD	RIGHT
LEFT	PUSHSTATE	POPSTATE	

Table 2: Reserved words.

3. What to submit

You will be provided with C source code for the scanner and turtle graphics functions along with some test input programs. Please test your code thoroughly before submission. Implement all grammar constructs. Useful error message concerning lexical or syntax errors (along with a graceful exit with a non-zero status code) are required. Please create an archive file that contains the following:

- A text file named README that contains the following:
    - Author and contact information (email address);
    - Brief overview of the project including what works and what doesn't work yet.
    - List of instructions necessary for compiling, linking, and executing your program;
    - List of all files in the archive.
- All source code.
- A design and implementation document describing ALL turtle grammar rules including the newly added ones and brief explanation on changes you make to the interpreter to support the new rules.
- A Makefile for building program.
- Any pertinent test files.

This homework will be submitted electronically on Angel. Please double check the files (e.g., download and open them) after you submit. It is due at midnight on the due date.