Project 1 – Using Lex and Yacc for Higher Level Abstraction
From Turtle Graphics Language to Postscript

This is an individual assignment, other than the partner for deliverable D5. You may discuss your ideas with others, but your work should be your own.

Deliverables and Deadlines:

F, 8/30	**D0: Obtain account and Create an svn repository**
F, 9/6	**D1: Understanding a lex spec by creating test cases**
F, 9/13	**D2: Extending a lex spec to augment the language lexeme set**
F, 9/20	**D3: Understanding a yacc spec by creating test cases**
F, 9/27	**D4: Extending a yacc spec to become a translator**
F, 10/4	**D5: Create test cases to break another person's translator**
F, 10/11	**D6: Test and Refine cycle – demo and svn/cvs directory**

Project Objectives: The student who completes this project should (1) be able to write regular expression specifications for a programming language's lexemes, (2) be able to write context free grammar specifications for a programming language's syntactic elements, (3) be able to construct a scanner and parser that accepts syntactically correct programs and rejects syntactically incorrect programs of a given programming language.

Procedure:

The major task of this project is to build a software system to translate programs written in a Turtle graphics-like programming language into Postscript. Turtle graphics, part of the LOGO programming language, is based on a relative cursor (the "turtle") moving and drawing on a Cartesian grid. The cursor has three attributes: position, orientation, and pen (which has color, width, up and down on the display), and moves with commands relative to its position.

Here is an example of a simple small program in Turtle graphics.

```
// the recursive dragon curve


PROCEDURE a(level,dist)
{ IF (level > 0) THEN
  { TURN -45; a(level-1,dist);
    TURN 90; b(level-1,dist);
    TURN -45;
  }
  ELSE
    GO dist;
}

PROCEDURE b(level,dist)
{ IF (level > 0) THEN
  { TURN 45; a(level-1,dist);
```

```
    TURN -90; b(level-1,dist);
    TURN 45;
  }
  ELSE
    GO dist;
}

UP; NORTH; GO 400; EAST; GO 150;
DOWN;

a(12, 5);
```

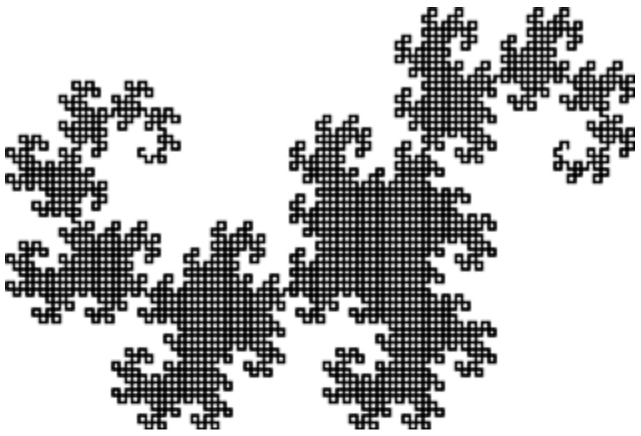
This is translated into the following PostScript program, which is perfectly illegible:

```
%!PS-Adobe
% generated by the simple turtle program

/sinphi 0.0 def
/cosphi 1.0 def
/state true def
/pi 4 1 1 atan mul def
newpath 0 0 moveto
/turtlea {
20 dict begin
/turtledist exch def
/turtlelevel exch def
turtlelevel 0 gt  { 45 neg 180 div pi mul sinphi cosphi atan add dup
sin /sinphi exch store cos /cosphi exch store
turtlelevel 1 sub turtledist turtlea
90 180 div pi mul sinphi cosphi atan add dup sin /sinphi exch store cos
/cosphi exch store
turtlelevel 1 sub turtledist turtleb
45 neg 180 div pi mul sinphi cosphi atan add dup sin /sinphi exch store
cos /cosphi exch store
 }{ turtledist dup cosphi mul exch sinphi mul state {rlineto} {rmoveto}
ifelse
 } ifelse
end } bind def
/turtleb {
20 dict begin
/turtledist exch def
/turtlelevel exch def
turtlelevel 0 gt  { 45 180 div pi mul sinphi cosphi atan add dup sin
/sinphi exch store cos /cosphi exch store
turtlelevel 1 sub turtledist turtlea
90 neg 180 div pi mul sinphi cosphi atan add dup sin /sinphi exch store
cos /cosphi exch store
turtlelevel 1 sub turtledist turtleb
45 180 div pi mul sinphi cosphi atan add dup sin /sinphi exch store cos
/cosphi exch store
 }{ turtledist dup cosphi mul exch sinphi mul state {rlineto} {rmoveto}
ifelse
 } ifelse
end } bind def
```

```
/state false def
/sinphi 1.0 store /cosphi 0.0 store
400 dup cosphi mul exch sinphi mul state {rlineto} {rmoveto} ifelse
/sinphi 0.0 store /cosphi 1.0 store
150 dup cosphi mul exch sinphi mul state {rlineto} {rmoveto} ifelse
/state true def
12 5 turtlea
2 setlinewidth
stroke
showpage
```

Using a PostScript Viewer such as psviewer or Adobe Acrobatreader, the generated postscript looks like:



This project is based on the paper and corresponding project - ``Teaching Compiler Construction and Language Design - Making the Case for Unusual Compiler Projects with PostScript as the Target Language," by Martin Ruckert, Munich University of Applied Sciences, Germany.

The overall end deliverable is a working scanner and parser that generate equivalent postscript for a drawing language that you partially design. The final product should accept and translate correctly specified drawing programs, and graciously terminate with an error message for incorrectly specified drawing programs.

The files for this project are located on orioles.acad.ece.udel.edu or mlb.acad.ece.udel.edu in /usa/pollock/cis471/project1.

Deliverables and Assessment:
D0: Svn repository.
Create yourself an svn repository, and a folder in the svn repository that you will use for this project assignment. See https://www.eecis.udel.edu/wiki/ececis-docs/index.php/FAQ/Subversion#toc6 for instructions. You should create your own

repository, and give me (username pollock) and the TA, access to it. You will be doing a manual start/stop on the service, so you give the port# when you start the process, and access the repository on that same port#. We will give out port#'s in class, all above 10000. You will be accessing svn.acad.ece.udel.edu by ssh from orioles (Solaris) or mlb (linux). Email the TA when you have your repository set up with the information for us to access it. You should not be giving out that information to other students.

D1: Understanding a lex spec
Go to the scanning directory under the project1 directory. Read the lex spec turtle.l, and create an extensive legal test case, and 3 illegal test cases. Name your test case files by yourlastname_legal-lex, yourlastname_ill0-lex, yourlastname_ill1-lex, yourlastname_ill2-lex. Place your test cases in your svn repository by the deadline.
Grading criteria:

- Abides by spec on number and naming of test cases
- complexity/coverage of the language features provided by the test cases

D2: Extending a lex spec to augment the language lexeme set
Design your own variation of the current drawing language, by changing rules to change the names of lexemes to your preferred names, and by adding rules to the lex spec to create a complete turtle-yourname.l spec that includes commands for:
471 students: conditionals, while-loops,
672 students: conditionals, while-loops, procedures with parameters.
Put your lex spec in your svn repo.

Grading criteria:

- extent of change to lexemes to demonstrate knowledge of lex spec
- completeness of new spec for new features
- correctness of spec in handling correct lexemes
- correctness of spec in error handling for illegal lexemes
- creativity

D3: Understanding a yacc spec
Read the yacc spec turtle.y and create an extensive syntactically legal test case, and 3 illegal test cases. Name your test case files by yourlastname_legal-syn, yourlastname_ill0-syn, yourlastname_ill1-syn, yourlastname_ill2-syn. Put your test cases into the svn.

Grading criteria:

- Abides by spec on number and naming of test cases
- Complexity/thoroughness of test cases

D4: Extending a yacc spec to augment the language syntax and postscript generation
Design you own variation of the syntax of the current drawing language, by changing the grammar rules and by adding rules to the yacc spec to create a complete turtle-yourname.y that includes features for

471 students: conditionals, while-loops,
672 students: conditionals, while-loops, procedures with parameters.

Add actions to your parser to generate the correct postscript. Test your translator on many test files. Be sure to include your test suite with your submission.

- Grading criteria: completeness of new spec for new features
- correctness of spec in handling correct syntax
- correctness of spec in error handling for illegal syntax

D5: Create test cases in attempt to break another person's scanner/parser
Choose a partner, and on your own, create a test suite with the goal of breaking your partner's translator. This part is graded based on the extensiveness of the test cases.
Grading criteria:

- Complexity/thoroughness of test cases
- Success in breaking the software

D6: Test, Refine Cycle and Final Demo
Perform the tests provided by your partner, and refine your translator. Test, refine, test,… Your svn directory of versions and test cases will be the deliverable as well as a scheduled demo with the TA.
Grading criteria:

- Demonstration of revision process using svn/cvs.
- Demo of working translator to TA:
    - Correctness of handling correct programs
    - Correctness of handling illegal programs
    - Extensiveness of test suite