In this assignment, you will use what you have learned to write a small compiler. The compiler will translate programs that describe simple line drawings from one language to another.

The basic input language

The basic input language (inspired by LOGO) has the following commands:

- Pen down
- Pen up
- Forward <distance>
- Left
- Right
- Goto <point> \textit{<point> is a pair of coordinates or a named point}
- Repeat <number of times> ... End
- Begin <name> ... End \textit{Defines a procedure}
- Point <name> <coordinates> \textit{Defines a point}
- <procedure-name>

The result of a program in this language is a drawing. Assume that the coordinate 0 0 is the upper-left corner, and initially the turtle is facing toward the upper-right corner. X coordinates increase to the right, and Y coordinates increase down the page.

The pen commands control whether movement implies drawing a line, or just movement. Lines are drawn when the pen is down. The Forward command moves the turtle forward (in the current facing direction) for the specified distance. If the pen is down, a line is drawn. Goto is similar, but the movement is direct to the specified point, regardless of the direction that the turtle is facing. “Goto” does not change the direction of the turtle, even if a line is drawn. Left and right make 90 degree turns in the directions specified. You can assume that the turtle spins in place and no line is drawn. The only control structure is Repeat. It takes an integer number of times and repeats the set of commands between the Repeat and End statements N times.

There are two types of definitions in this language. First, points can be named. The point command takes a name and coordinates and assigns the name to the coordinates. Named points can be used as the target of a goto. Next, procedures can be named (there are no parameters, however). Procedures are preceded by a begin command (naming the procedure) and are followed by an end command. You can assume that procedures are not nested. Outside of a point or begin statement, the use of a name that has not been defined is an error.
Final result of compilation

The final result of compilation is a sequence of “line statements”. Each statement is simply “line x1 y1 x2 y2” and draws a line between the specified points. Actually generating the drawing is not required, though I’m pretty sure there are existing programs that can interpret input of this form.

Example:

Input:

    Begin square
    Pen down
    Repeat 4
    Forward 20
    Right
    End
    Pen up
    End

    Point start 0 0
    Point middle 100 100

    Goto start
    Square
    Goto middle
    Square

Result:

    line 0 0 20 0
    line 20 0 20 20
    line 20 20 20 0
    line 20 0 0 0
    line 100 100 120 100
    line 120 100 120 120
    line 120 120 100 120
    line 100 120 100 100

Assignment:

While your final goal is to complete a compiler for this language, you should consider this assignment in three phases:

1) Syntax checking -- read a program and generate messages for syntax errors (only). For this phase, you will use FLEX for lexical analysis and BISON for parsing. Be sure to
document your token set (and their regular expressions) and your grammar. For this phase, you can assume that all names are valid. Demonstrate your syntax checker on both correct and incorrect programs. Try to generate error messages that are as helpful as possible. (50 points / 100)

2) Semantic analysis and symbol tables -- in this phase, you should generate a symbol table and check that all names are valid when used. Your symbol table should include each symbol’s name and its definition. The easiest way to represent a function definition might be as a string. For this phase, be sure to document your symbol table representation, and print your symbol table at the end of compilation (or when an unrecoverable error is found). For extra credit, consider error recovery -- can you find multiple errors in a program? Demonstrate your semantic analyzer on correct and incorrect programs. (25 points / 100)

3) Compilation (including code generation). For each valid program, generate a string in the final language. (If this is printed using cout, you can easily save it to a file by redirecting the output, e.g. compiler > outfile.) You can either do this in one pass, or in two passes. If you do it in two passes, you should first modify your semantic analyzer to generate a program in a simplified language of your definition. One suggestion would be to generate an equivalent program that has only pen, forward, goto, and turning commands. The second phase will then interpret the simplified program to generate the line statements. You can probably use the same lexical analyzer for both phases, but you will need two different Bison files. Carefully document both your design and implementation and demonstrate your compiler on several correct programs. (25 points / 100)

4) **If you finish too soon (extra credit):**
Add more power to your language. Some possibilities might include: functions with parameters, conditional statements and/or while loops, local variables (or other scoping ideas), or even recursion (for which conditionals and parameters are a prerequisite).

**Deliverables:**
- Files (*.lex and *.y) for each phase of compilation. Also include any c++ or .h files that are used in compilation, and instructions for compiling each phase.

- Writeups for each phase, including documentation and evaluation as described for each phase above. Even if a phase is not complete, partial credit can be earned by documentation and a test plan.