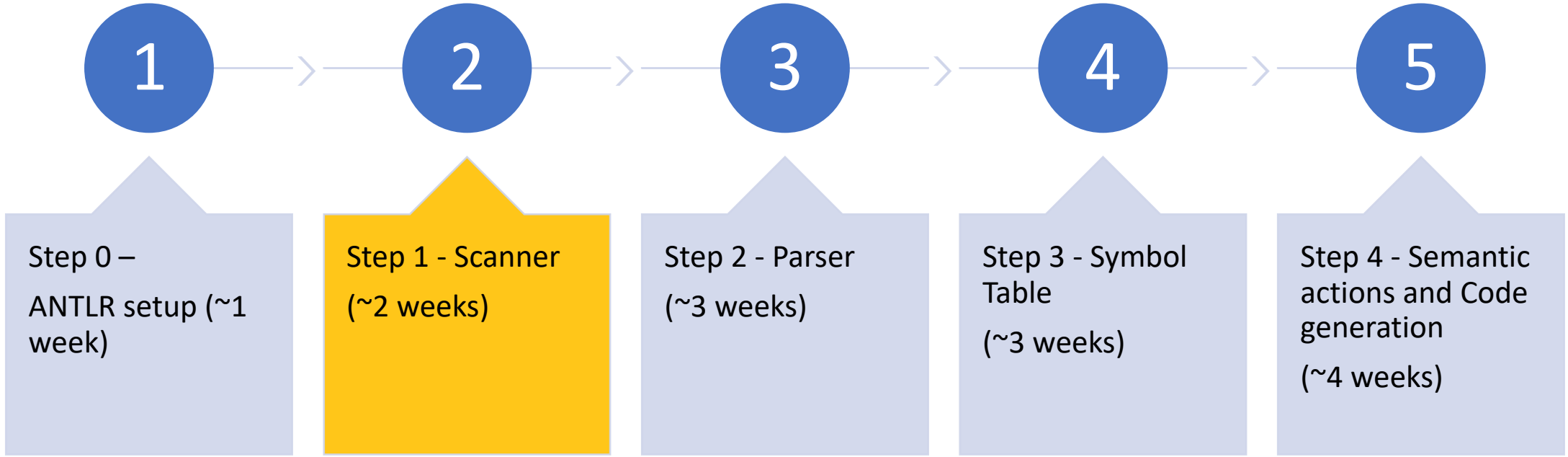
# Course Project

# Step 1

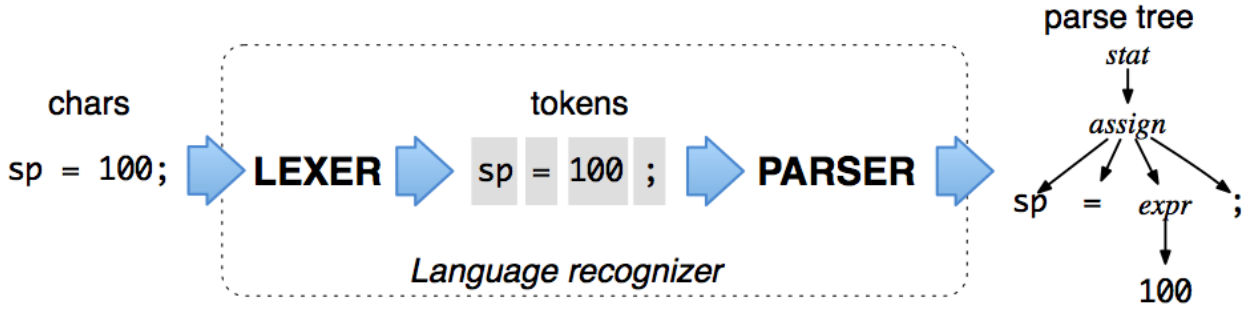
Scanner

# Four steps

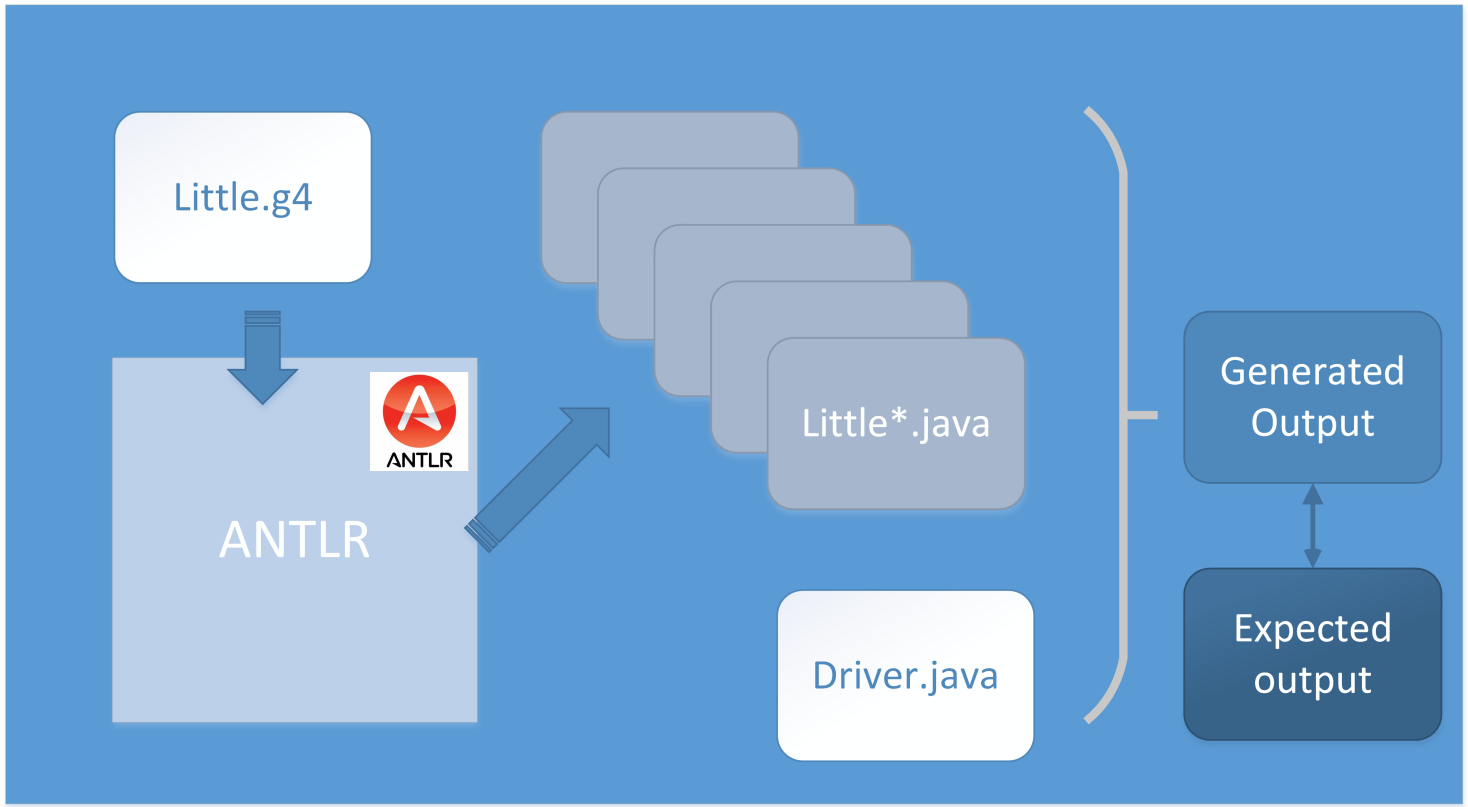

1
2
3
4
5
Step 0 –
ANTLR setup (~1 week)
Step 1 - Scanner
(~2 weeks)
Step 2 - Parser
(~3 weeks)
Step 3 - Symbol Table
(~3 weeks)
Step 4 - Semantic actions and Code generation
(~4 weeks)

# ANTLR (Another Tool for Language Recognition)

- ANTLR is a state-of-the-art scanner/parser generation toolkit Written in Java: http://www.antlr.org
- can be used with Java and python (and some others).
- Regardless of the programming language you choose to build your compiler, GTA will only provide conceptual help regarding implementation (i.e. no technical support for debugging any code).



# How ANTLR works



# Scanner

- A scanner's job is to convert a series of characters in an input file into a sequence of *tokens* -- the "words" in the program. So, for example, the input A := B + 4

Would translate into the following tokens:

IDENTIFIER (Value = "A")
OPERATOR (Value = ":=")
IDENTIFIER (Value = "B")
OPERATOR (Value = "+")
INTLITERAL (Value = "4")

# Token Definitions (LITTLE)

```
an IDENTIFIER token will begin with a letter, and be followed by any number of letters
and numbers.

IDENTIFIERS are case sensitive.

INTLITERAL: integer number
            ex) 0, 123, 678
FLOATLITERAL: floating point number available in two different format
                yyyy.xxxxxx or .xxxxxxx
            ex) 3.141592 , .1414 , .0001 , 456.98

STRINGLITERAL: any sequence of characters except '"'
            between '"' and '"'
            ex) "Hello world!" , "***********" , "this is a string"

COMMENT:
      Starts with "--" and lasts till the end of line
      ex) -- this is a comment
      ex) -- anything after the "--" is ignored

Keywords

PROGRAM,BEGIN,END,FUNCTION,READ,WRITE,IF,ELSE,FI,FOR,ROF,RETURN,INT,VOID,STRING,FLOAT,
WHILE, ENDIF, ENDWHILE

Operators
:= + - * / = != < > ( ) ; , <= >=
```

# What you need to do

- build a scanner that will take an input file (LITTLE source program) and output a list of all the tokens in the program.
  - For each token, you should output the token type (e.g., OPERATOR) and its value (e.g., +).
- Inputs/outputs (i.e. testcases) are provided.
  - Your outputs need to match our outputs *exactly* (they will be compared using Unix *diff*, though whitespace will be ignored).
  - Output a list of accepted tokens (both symbol and literal)
    - Symbol : *INTLITERAL*
    - Literal : *7*

# You turn in…

- The grammar dentition (*.g or *.g4)
- Your source program that is the driver for the lexing
- An executable called 'Micro' that takes care of setting up, compiling, and running your code.
  - You should be able to type './Micro', followed by a file name, in your shell and have your lexer execute. (Recommend a simple bash script)
- Any other files that were not generated. (Meaning you wrote them)

# You don’t turn in…

- The ANTLR jar
- All files in 'Step1_files’ archive
- All files that were auto generated