COSC 455

Programming Languages: Design and Implementation

Fall 2025

**Lab Exercise #1**

*If you never did you should. These things are fun and fun is good.*

* Dr. Seuss

**Name:** Blessing Abumere **/ 10**

**Goals:** The intention of this lab exercise is to get some initial exposure to lexical and syntax analysis using [Backus-Naur Form (BNF)](https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form) and gain initial experience in simple grammars using ANTLR (<http://www.antlr.org/>). ANTLR (**AN**other **T**ool for **L**anguage **R**ecognition) is a free, open source parser generator tool that can be used to implement programming languages and domain-specific languages (DSLs). In doing so, you should start to develop an understanding of how program languages are designed, lexical and syntactic analysis, familiarity with parser generator tools and developing BNF/EBNF grammars.

**Environment:** Throughout this lab we will use the ANTLRWorks 1.5.2 tool that I have provided on Blackboard (Labs/ Lab 1). This is an executable JAR file that you should be able to run directly from the browser or by double-clicking the executable JAR file; *however, it will run better from the command line, as follows*:

- java -jar <name of file>

If you are lazy (like me), you can simply drag and drop the jar file into a command line window to capture the full name.

For simplicity, we are using an older version of ANTLRWorks, not the current version, that can be found at <http://www.antlr.org/>. ANTLR is a very powerful tool that can do many of the things we will talk about during the first part of the semester, including defining grammars in BNF, producing parse and abstract syntax trees and generating the necessary source code for lexical and syntax analysis (Java, Python, Perl, C++, C#, Scala, Ruby, JavaScript, etc.). In this introductory lab, we will only use a small part of the ANTLR tool.

**Slack Channel:** All/any questions, problems and/or announcements for this lab should be directed to the course’s #lab1 Slack channel.

**Submission:** All lab exercises should be submitted in a .docx or .odt file using this document as a template via Blackboard and have a naming convention of *FirstNameLastNameLab1*. For example, if your name is Homer Simpson, you would submit a single file via Blackboard named *HomerSimpsonLab1.docx*. The file should contain this lab description with your answers to the questions.

**Deadline:** Submitted via [Blackbeard](https://www.youtube.com/watch?v=iA3rIjM3Vds) by 11:59pm on [Tuesday](https://www.youtube.com/watch?v=-EOD8w-3Tz4), September 2, 2025.

1. **ANTLR Introduction.** (2 points) Consider the USPS language we briefly described in class, defined as follows:

*All packages sent internationally have a 13 character tracking number assigned to them. The tracking numbers are in exactly the following form: two letters (A-Z) followed by nine numbers (0-9), followed by the two letters (A-Z) comprising a country code (US, ZH, KR, etc.).*

I have provided an ANTLR-based grammar, loosely based on [Backus-Naur Form (BNF)](http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form) on Blackboard, named usps.g, that implements our USPS language. We will be using this grammar specification so that we can examine example parse trees and generate the parser for this language.

**Task 1.** Start ANTLR and open the simple usps.g grammar that I have provided for our USPS language. This will present the grammar editor screen divided into 3 main parts: the upper right section is the editor where you define a grammar in [EBNF](https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_Form), the upper left provides an outline view of lexical tokens and parser rules that you have defined, and the lower half provides a window for output. You should also notice the 4 tabs in the lower left corner: Syntax Diagram, Interpreter, Console and Debugger. We will use the first 3 in this lab.

**Task 2.** In any grammar, the first thing to do is determine what the legal [lexemes](https://en.wikipedia.org/wiki/Lexeme) are so that we can define them in the grammar. To define lexemes in ANTLR, we must use all capital letters for the lexical definition. For example, using the language described above, I have defined the legal lexemes for upper-case letters in ANLTR with the following ANTLR rule:

LETTER : 'A'..'Z';

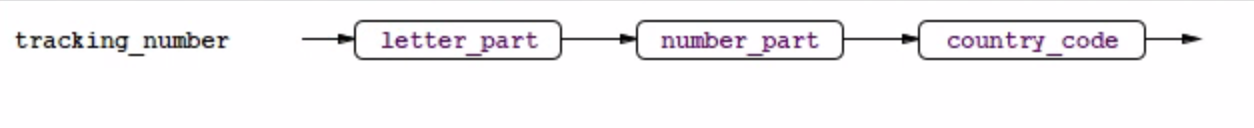
This rule defines the legal lexemes for an upper-case letter to be all letters A thru Z. I have similarly done this for the numbers (i.e., 0-9) and country codes (i.e., KR, US, ZH) allowed in this language. This defines all the legal tokens that a lexical analyzer (a.k.a., scanner) will recognize.

When designing a grammar, once you have the necessary lexical rules, you should be able to view the generated syntax diagrams of each rule by highlighting a rule and selecting the Syntax Diagram tab in the lower left-hand corner and putting the cursor on one of the lexical or syntax rules. ANTLR Syntax Diagrams use the same syntax as in the textbook (see Chapters 1 and 2): square boxes for terminal symbols (i.e., tokens) and circular boxes for non-terminal symbols (i.e., parser rules). Viewing the Syntax Diagram will be helpful while designing grammars later in the semester so that you can visually see the language that your BNF rules are describing.

**Task 3.** Now that we have introduced the lexical tokens defined for this grammar, we must introduce the syntax of the grammar through EBNF parser rules. To define parser rules in ANTLR, we must use lowercase letters to define the rules. In the language described above, any legal sentence must start with an upper-case letter and may or may not be followed by other characters. Based on this, we will define the start state, parser rule as follows:

tracking\_number : letter\_part number\_part country\_code;

This defines a parser rule, named tracking\_number, that requires whatever is defined in the syntax rule named letter\_part followed by whatever is defined in the syntax rule named number\_part followed by a country\_code token. You can confirm this by examining the Syntax Diagram as follows:



**Task 4.** Once any grammar is implemented, it should be tested with some example inputs. First, we will use a valid USPS tracking number, “JD849375286US”. To do so, click on the Interpreter tab in the lower left hand corner, select “tracking\_number” from the dropdown menu to note it as the start state of our language, enter in the input string in the lower-left edit, and hit the “play” button located next to the dropdown menu where you selected “tracking\_number”. This will generate a parse tree on the right-hand side to confirm that this input sentence is a correct, parsable sentence in our defined grammar.

Provide a screen shot of your parse tree. To do so, right click on the parse tree, select Export as a Bitmap image, save it and copy it here.

A diagram of a diagram

AI-generated content may be incorrect.

Repeat this process for an invalid USPS tracking number, “ABC492286943ZH”. Provide a screen shot of your parse tree. Additionally, briefly describe what kind of error (i.e., lexical error, syntax error, semantic error) this is and why it is an error.

A diagram of a system

AI-generated content may be incorrect.

I believe this is a lexical error.

This is an error because the letter\_part can only have two letters and this one has “ABC” (3 letters) instead of just “AB”.

**Task 5.** One of the main uses of ANTLR in the “real world” is to generate the lexical and syntax analyzers’ source code for [domain-specific languages](http://en.wikipedia.org/wiki/Domain-specific_language). This is how many compilers/interpreters are built today. To demonstrate this for this “tracking\_number” language, select the Generate menu and select Show Parser Code and briefly examine the syntax analysis code specific to this language. Do the same to briefly examine the lexical analysis code specific to this language by selecting Show Lexer Code from the Generate Menu. Note, for some reason (I’m not sure why…), this may generate an error on lab computers. Sometimes, this can be rectified by pointing ANTLR (under the Preferences menu item) to the Java JDK; if you are using a department lab computer, the JDK is located at C:\cs\Java\. If this does work for you, provide the generated lexical & syntax analysis code. If you get an error, no worries – move on.



A screenshot of a computer

AI-generated content may be incorrect.

1. **Designing** [**Simple**](https://www.youtube.com/watch?v=KlcHi4c_22w) **Yodic.** (2 points) Consider the following language description of Yodic, loosely based on the odd syntax of Yoda:

*Yodic follows an object-subject-verb structure. In our simplified version of Yodic, the only allowed objects are ‘unicorns’, ‘netflix’, ‘homework’ and ‘turtles’. The allowed subjects are ‘you’, ‘we’, and ‘i’’. Finally, the only allowed verbs are ‘fear’, ‘like’, ‘have’ and ‘watch’.*

**Task 1.** Create a new grammar file, named yodic.g, and define the legal lexical tokens in the grammar for this Yodic language. You will likely want to define separate lexical rules for the objects, subjects and verbs.

**Task 2.** Develop the parser rules for this grammar. You will likely want a single syntax rule for defining Yodic. As you develop/troubleshoot your grammar, be sure to utilize the Syntax Diagram to visualize the rules you are defining.

**Task 3.** Provide your full ANTLR grammar here (copy and paste the code) and a screenshot for the parse trees generated by the following inputs: ‘homework i fear, ‘[i like turtles](https://www.youtube.com/watch?v=CMNry4PE93Y)’, and ‘netflix i binge’.

grammar yodic;

// lexical definitions

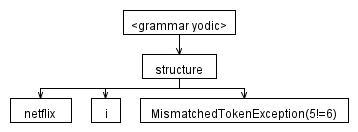
OBJECT  :   'unicorns'| 'netflix' | 'homework' | 'turtles' ;

SUBJECT :   'you'   | 'we' | 'i' ;

VERB    :   'fear' | 'like' | 'have' | 'watch' ;

// syntax definititions

structure : OBJECT SUBJECT VERB;

homework i fear
I like turtles


1. **Designing a Simple lolspeak Grammar.** (2 points) Consider the following language description of a simple subset of the Internet meme [lolspeak](http://en.wikipedia.org/wiki/Lolcat) language:

*The only valid sentences in our* [*lolspeak*](https://www.youtube.com/watch?v=en22xeReJm8) *language must be a noun phrase followed by a verb followed by another noun phrase. A noun phrase is simply an article followed by a noun. The allowed articles in our language are only ‘a’ and ‘teh; the allowed nouns in our language are only ‘kat’, ‘dawg’ and ‘cheezburgr’; and the allowed verbs in our language are only ‘ates’, ‘lovez’ and ‘hatez’.*

**Task 1.** Create a new grammar file, named lolspeak.g, and define the legal lexical tokens in the grammar for this language. You will likely want to define separate lexical rules for the nouns, verbs and articles.

**Task 2.** Develop the parser rules for this grammar. You will likely want a syntax rule for a sentence and a noun phrase. As you develop/troubleshoot your grammar, be sure to utilize the Syntax Diagram to visualize the rules you are defining.

**Task 3.** Provide your full ANTLR grammar here and a screenshot for the parse trees generated by the following inputs: ‘teh dawg hatez a kat’, ‘a kat bitez teh cheezburgr’, ‘teh cheezburgr lovez a katz’, and ‘teh kat ates dawg’.

grammar lolspeak;

// lexical definitions

ARTICLE : 'a' | 'teh' ;

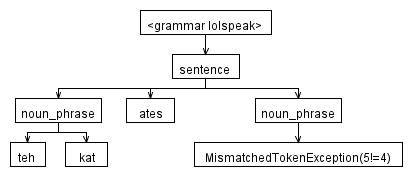
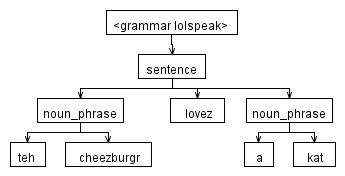
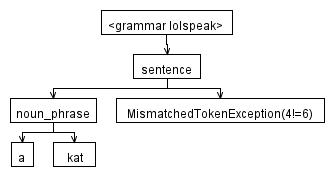
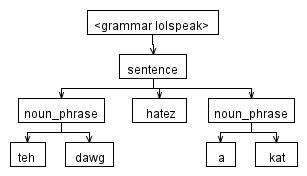
NOUN : 'kat' | 'dawg' | 'cheezburgr' ;

VERB : 'ates' | 'lovez' | 'hatez' ;

// syntax definitions

sentence : noun\_phrase VERB noun\_phrase;

noun\_phrase : ARTICLE NOUN;



**Task 4.** Of the example inputs from Task 3, which is a lexical error and why? Similarly, which is a syntax error and why?

I believe the lexical error happens for “a kat bitez teh cheezburgr” because “bitez” isn’t one of the definitions. “teh kat ates dawg” is the syntax error because it is wrong syntactically.

1. **Designing a Simple Phone Number Grammar.** (2 points) Consider the following language description of the syntax of a US telephone number (based on [this](http://electronics.howstuffworks.com/question659.htm)):

*A valid US phone number follows the format: a country code, a dash, an area code, a dash, a prefix, a dash, and a line number. The country code for the US is ‘1’, the area code consists of three numbers, the prefix consists of three numbers and the line number consists of four numbers. For example, the following are valid US phone number based on our description: ‘1-855-*[*867-5309*](http://www.youtube.com/watch?v=6WTdTwcmxyo)*’ and ‘1-800-273-8255’.*

**Task 1.** Create a new grammar file, named usphonenumber.g, and define the legal lexical tokens and parser rules this language. As you develop/troubleshoot your grammar, be sure to utilize the Syntax Diagram to visualize the rules you are defining.

**Task 2.** Provide your full ANTLR grammar here and a screenshot for a few parse trees that you used to test the grammar, including both syntactically correct and incorrect inputs.

grammar usphonenumber;

// lexical definitions

DIGIT   :   '0'..'9';

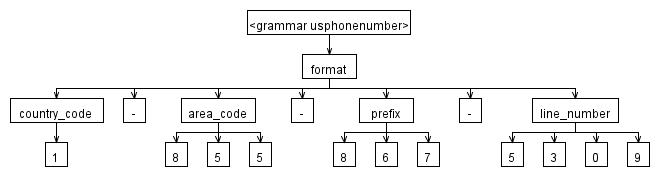
// syntax definitions

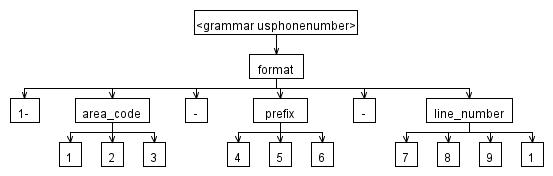
format : '1-' area\_code '-' prefix '-' line\_number ;

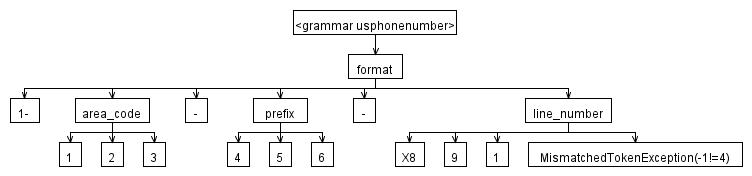
area\_code: DIGIT DIGIT DIGIT;

prefix: DIGIT DIGIT DIGIT;

line\_number: DIGIT DIGIT DIGIT DIGIT;







1. **Designing a Date Grammar.** (2 points) Consider the following language description of a simplified, MM/DD/YYYY date grammar:

*A basic date can be represented as MM/DD/YYYY format where valid dates are as follows: two digits for month (01–12), the slash separator, two digits for day (01–31), the slash separator, and four digits for year.*

**Task 1.** Create a new grammar file, named date.g, and define the legal lexical tokens and parser rules this language. As you develop/troubleshoot your grammar, be sure to utilize the Syntax Diagram to visualize the rules you are defining.

**Task 2.** Provide your full ANTLR grammar here and a screenshot for a few parse trees that you used to test the grammar, including both syntactically correct and incorrect inputs.

grammar date;

 // lexical definitions

DIGIT : '0' .. '9';

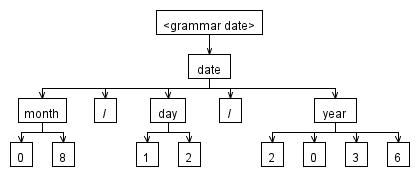
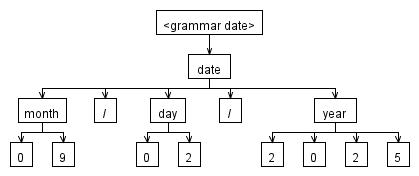
// syntax definitions

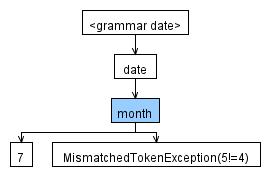
date : month '/' day '/' year ;

year : DIGIT DIGIT DIGIT DIGIT ;

month : DIGIT DIGIT ;

day : DIGIT DIGIT ;





**Note:** There are at least 3 [Easter eggs](https://en.wikipedia.org/wiki/Easter_egg_(media)) (this isn’t one of them) in this lab as well as one, still unidentified, in the Course Syllabus. If you find one, email and let me know via direct message on Slack. Happy hunting!