Geaux

Project 1

Project 1 - Lexer

- A lexer is a program that takes in text as input and spits out “tokens”
- Tokens are just java objects that represents the original text

int => INT

string => STRING

const => CONST

Why is this useful?

- A lexer is supposed to make sense of unstructured text
  - We want to *collect* strings of chars into a singular token
- In the previous cases “int” => new Token(INT)
  - “int” is 3 chars
  - INT is a singular java object
  - … kind of useful, I guess…
- For String literals however:
  - “Hello World” => STRING_LITERAL(“Hello World”)
  - Way more useful

ANTLR4

- We are using ANTLR4
- ANTLR4 is a Lexer and Parser Generator
- Today, we will use it as a Lexer Generator
- In project 2, we will generate a Parser

```
lexer grammar gLexer;

@header {
   package Parse.antlr_build;
}


@members {
   StringBuilder sb;
   private int stringToInt(String target) {
      // TODO Implement me!
      return 0;
   }
}

fragment ALPHA
   : [A-Za-z]
   ;
fragment DIGIT
   : [0-9]
   ;

ADD
   : '+'
   ;
```

```java
// Generated from Parse/gLexer.g4 by ANTLR 4.13.2

   package Parse.antlr_build;

import org.antlr.v4.runtime.Lexer;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.TokenStream;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.misc.*;

@SuppressWarnings({"all", "warnings", "unchecked", "unused",
public class gLexer extends Lexer {
»  static { RuntimeMetaData.checkVersion("4.13.2", RuntimeMe

»  protected static final DFA[] _decisionToDFA;
»  protected static final PredictionContextCache _sharedCont
»  »  new PredictionContextCache();
»  public static final int
»  »  ADD=1;
»  public static String[] channelNames = {
»  »  "DEFAULT_TOKEN_CHANNEL", "HIDDEN"
»  };

»  public static String[] modeNames = {
»  »  "DEFAULT_MODE"
»  };

»  private static String[] makeRuleNames() {
»  »  return new String[] {
»  »  »  "ALPHA", "DIGIT", "ADD"
»  »  };
»  }
»  public static final String[] ruleNames = makeRuleNames();

»  private static String[] makeLiteralNames() {
»  »  return new String[] {
»  »  »  null, "'+'"
»  »  };
»  }
»  private static final String[] _LITERAL_NAMES = makeLitera
»  private static String[] makeSymbolicNames() {
```

Using ANTLR

- ANTLR is 2 things:
  - A generator:
    - We provide the grammar, and it generates the java file
  - A runtime library
    - The generated file needs certain imports provided by ANTLR’s library

```
6  import org.antlr.v4.runtime.Lexer;
7  import org.antlr.v4.runtime.CharStream;
8  import org.antlr.v4.runtime.Token;
9  import org.antlr.v4.runtime.TokenStream;
10 import org.antlr.v4.runtime.*;
11 import org.antlr.v4.runtime.atn.*;
12 import org.antlr.v4.runtime.dfa.DFA;
13 import org.antlr.v4.runtime.misc.*;
```

How do we use ANTLR?

- I have provided two things:
  - A shell script for operating ANTLR
  - An “ANTLR Driver”
    - This is a program we write that “Drives” the lexer
    - Once the lexer java file is generated, we still need to call it!
- I have also included the ANTLR tool, and runtime library
  - It’s in the lib folder!

The project - Geaux

- We are making a lexer for our C-like language: Geaux.
- Our lexer will be similar to a C lexer, with some deviations
- C lexer documentation can be found here:
  - https://port70.net/~nsz/c/c89/c89-draft.html#3.1

constant:

floating-constant

integer-constant

enumeration-constant

character-constant

integer-constant:

decimal-constant integer-suffix$_{opt}$

octal-constant integer-suffix$_{opt}$

hexadecimal-constant integer-suffix$_{opt}$

string-literal:

" s-char-sequence$_{opt}$ "

L" s-char-sequence$_{opt}$ "

s-char-sequence:

s-char

s-char-sequence s-char

s-char:

any member of the source chara

escape-sequence

Keywords, Operators, Punctuators

- examples/keywords.g contains all of the ones we will be implementing.
- It includes a couple keywords not in the C standard, such as fun and var.

Preprocessing

- We are not implementing anything from the preprocessing section

Grading

- Your project will be expected to have the same output as my project
  - A binary of my version is provided
- I will be using a test case generator to generate 10 random test cases. You will lose a point for each test case where our outputs don’t line up
  - This system is new, so I will be going back over the results manually to verify that they are reasonable
- You have been provided 100 generated tests

Some notes

- Opt means optional
- If you have any question, better to ask than not.

Grading

- Some sticking points:
  - String escape sequences must not show up as “hello \n world”
  - I expect to see:

    hello

    world

- Just make sure it matches my output

Video

1. How well does your project work?
    a. What's wrong with it, if anything?
    b. Show it working on the 10 random test cases
2. What was the experience like doing the project?
    a. Did you have trouble with ANTLR?
    b. How long did it take you?
    c. What strategies for group coding, AI use, time management, etc led to success, and which ones didn't work as well?
3. Not long, 5 minutes

AI

- ANTLR documentation is… not good (imo)
- I used AI to explain to me how ANTLR works
  - It worked incredibly well
- I don’t want to make you do the same if you don’t want to
- I provided an example grammar that demonstrates what you will need to know to do this project

DEMO