CSC435/535: Assignment 1 (Due: 11:55pm, Tuesday 26 January 2016)

Introduction

This assignment asks you to complete the lexical analysis and parsing phases of the front-end for the Go subset compiler. To avoid confusion, this subset will henceforth be named Goo.[1]

A brief summary of which features of Go are included in the Goo language is provided in the document `GooFeatures.pdf`. A precise syntax for the Goo language is provided in the document `GooSyntax.pdf`.

You are provided with a very incomplete lexer/parser specification using Antlr4 syntax in the `Goo.g4` file. (Comments in the file mark the places where something is missing.)

Assignment Description

- You must edit the `Goo.g4` file to add specifications for the parts detailed below. You will use the on-line reference for the Go language at this URL `https://golang.org/ref` to obtain precise specifications of the language elements and syntactic constructions needed for this assignment.
  Do read the notation conventions explained at the beginning of the Go language reference. Equivalences between those notations and the notations used by Antlr4 are listed below in Table 1. Note also the use of 'CamelCase' for compound names in both the Go reference and (by convention) in Antlr4 specifications. You should use Camel Case too.
- In the lexer part of the `Goo.g4` file, you must complete and correct the Antlr4 specifications for all the tokens which are incomplete. The text 'REPLACE ME' appears in the lines where the file must be edited.
  Notes:
  - Do not provide specifications for any additional tokens. All the necessary tokens are named in the file already. You are free to specify as many lexer fragments as you need.
  - There are some token types listed in the lexer part of the file which are tokens used in the Go grammar but not used in the Goo grammar. For example, `ImaginaryLit` is needed for Go but not needed for Goo. However you must provide definitions for all such tokens.
- Once the lexer part of the file has been finished, you must edit the grammar part of the `Goo.g4` file so that the Goo language is recognized. The comment '// there's a lot of missing stuff here' appears in places where one or more grammar rules are missing. The text 'EXPAND ME...' appears in the righthand sides of grammar rules where things are missing.

Advice on Completing this Assignment

1. The initial version of the `Goo.g4` file is already acceptable as input to Antlr4. You will make life a lot easier for yourself if you maintain this property.
2. Implement the missing language tokens one at a time and test that Antlr4 doesn't complain after each token addition. When you have a bunch of tokens defined, try out the lexer/parser that

1. The name Gingo (for "Gingo Is Not GO" was considered, but that is just too cute.

Table 1: Notation Equivalences

Example of Notation used in Go Reference	Corresponding Notation in Antlr4	Comments
`"<<"`	`'<<'`	Use only single quotes in Antlr4 for literal strings
`identifier`	`Identifier`	Terminal symbols (tokens) in Antlr4 must have names which begin with an uppercase letter
`FunctionDecl`	`functionDecl`	Nonterminal symbols in Antlr4 must have names which begin with a lowercase letter
`OperandName = identifier | QualifiedIdent .`	`operandName : Identifier | qualifiedIdent ;`	The example shows two rules which share the same LHS. Note the use of colon and semicolon in Antlr4.
`(Function | Signature)`	`(function | signature )`	Choices can be embedded in the RHS of rules
`[SimpleStmt ";"]`	`(simpleStmt ‘;’)?`	A group in square brackets is optional (0 or 1 occurrences), for which Antlr4 uses a question mark.
`{ImportDecl ";"}`	`(importDecl ';')*`	A group in curly braces is repeated 0 or more times, for which Antlr4 uses a Kleene star.

Antlr4 creates for you. There is an extra goal symbol of the grammar which matches 0 or more tokens in any order (without having to form a valid Goo program); the extra goal symbol is named ‘`anything`’. For example, if you have defined the `FloatLit` token, then a sequence of commands in a command window to try things out would go something like this (where ‘%’ represents the command prompt and `^D` is the end-of-input signal *Control-D*[1]):

```
% antlr4 Goo.g4
% javac Goo*.java
% grun Goo anything
123.45
36.e-45
^D
%
```

If no error messages are displayed, your tokens are being recognized. If you want to be extra sure, you can always add the `-tokens` option at the end of the `grun` command.

3. Now you can start inserting missing grammar rules or completing rules which have missing parts. If you are careful about the order in which you tackle this, you do not have to complete the whole

1. Remember to use *Control-Z* when trying this on Windows systems.

Goo grammar before you start testing. As long as the rules you have added do not reference non-terminal symbols which are not yet defined, you can use the name of one of your non-terminal symbols as a goal symbol when running the parser. For example, if you have just added the rules for the goto statement, your test run could look something like this:

```
% antlr4 Goo.g4
% javac Goo*.java
% grun Goo gotoStmt
   /* comment */ goto Lab99 ;
^D
%
```

If there is no output, your input was parsed successful. If you would like to see *how* it was parsed, add the `-tree` or `-gui` option to the `grun` command.

4. Be sure to test that infix operators (binary operators) in expressions are being used with the correct precedence and associativity. You will need to enable the `-tree` or `-gui` option to check that.
5. When the Goo grammar is reasonably complete, you can start testing it on sample Go programs (as long as that Go program does not use syntactix constructs outside the Goo subset). It's easy to copy and paste sample Go programs from the online tutorials to create your own test cases. A collection of sample Go programs is included with the assignment materials.
6. Carefully read the Go Language Specification to check for things you may have missed. In particular, note that identifier names (as well as runes and string literals) can include characters outside the ASCII or extended ASCII character sets. To get this language feature right, look and see how it is done in Java8 and then copy that. (The Antlr4 specification for Java8 is provided.) It is not plagiarism if you copy reasonable amounts of other people's work and acknowledge the source in a comment.

Evaluation Criteria

You will be evaluated on these criteria.

- Meeting the submission requirements.
- Implementing a correct lexer (scanner) for the full Go language.
- Implementing a correct syntax analyzer (parser) for the Goo subset of Go, following the same syntactic structure as provided in the `GooSyntax.pdf` document.
  Note: it is easy, but incorrect, to recognize a language while using the wrong syntactic structure. For example, recognizing the expression `a*b+c` giving the `+` operator higher precedence than `*` does not affect the language being recognized but it would be wrong nevertheless.

The Provided Materials

File Name	Description
`Goo.g4`	A textfile containing an incomplete Antlr4 specification for Goo.
`GooSyntax.pdf`	The grammar rules for Goo in the same format as used for defining the grammar of Go in the Go Programming Language Specification.
`GooFeatures.pdf`	A one page summary of what Go features are included in Goo. (This summary will be made more precise as needed for later assignments.)
`GooProgs.zip`	A zipfile containing many small sample Go programs which should be parsable when using the Goo grammar.
`Java8.g4`	The Antlr4 specification for Java8. You should borrow ideas for specifying identifiers, character constants (runes) and string constants from Java8 because both Go and Java8 use UTF8 encodings for names and such constants.

Submission Requirements

1. You must submit exactly one textfile with the name `Goo.g4`. No other name and no other capitalization of the name is acceptable. It must be submitted via the course website on conneX.
2. The comment at the start of the file must be edited so that it lists the names and student numbers of the team members who collaborated on the assignment.
3. Do not remove or change the grammar rules for the extra goal symbol '`anything`'. (The marker needs that to test the lexer part of your submission.) In fact, you should not remove definitions for any of the nonterminals and terminals defined in the supplied file; you should only be adding or completing definitions.
4. The submitted file must not produce any error messages or warning messages when processed by Antlr4 or when the generated Java files are compiled. Errors that prevent your lexer and parser from running will result in an automatic zero for the assignment. (If you cannot get everything working, submit an Antlr4 file which specifies the parts you could get working and dummy out the nonworking parts.)
5. The project should be completed in teams of either 2 or 3 persons. The ideal size is 2 people. (Teams of size one are permitted, but you will receive no extra credit nor allowances for doubling your workload in this manner.) All team members *must* participate. To encourage and to reward active participation, you can expect the midterm test and the final exam to contain at least one question on the minutiae of Antlr4 usage and Goo syntax. Team memberships can be changed after each assignment.
6. Late assignments are accepted with a penalty: 10% (of maximum score) for 24 hours late, 25% for 48 hours late, 50% for 72 hours late. You can resubmit as often as you want. The time of the latest submission determines which late penalty, if any, is to be applied.