# **Project name: Go language Parser**

# **Team members: 정재우(20215184), 남동준(20195053)**

# **Team name: Naming variable is too hard**

# **Scope of the project**

**It receives and interprets codes that fit part of the Go program grammar, and outputs are proper parse tree of code. If an error occurs, an appropriate error message is output.**

**- Keywords (16/25)**

**ex) break, default, case, go, else, goto, package, switch, const, if, type, continue, for, import, return, var**

**- Operators and punctuation (27/48)**

**It receives arithmetic operator (+,-,\*,/,%,(,)) and logical operator(&&,||,!) and relational operator(<,>,<=,>=,==,!=)**

**It allows unary operator (-,--,++,!) and definition operator(:=) and comma(,)**

**It also allows reassign statements (+=,-=,\*=,\=,%=)**

**- Types (3/11)**

**ex) Boolean, Integer (Not float), string**

**- Identifiers**

**- Constants**

**ex) Integer, true, false, string of character**

# **Representative diagram to explain your project**

**Tools used: PLY (Python Lex-Yacc)**

**Input: String of code**

**Output (proper error message): If error occurred, print proper error message. Else, print proper parse tree.**

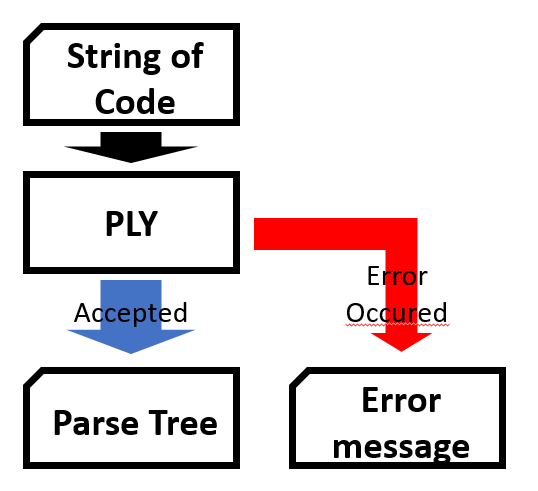


Fig.1 Diagram of Go parser

Input

# Limitations of your project

We allowed only the small part of go language grammar. Some grammars are not allowed in our project.

There is the list of the non-allowed grammar

- Definition and execution of function (except main function)

- local variable declaration and their scope (except main function)

- Numerical and array data type (We accept only integer)

- Unimplemented Keywords: interface, select, defer, go, map, struct, chan, goto, fallthrough, range

# Sample input and output



Fig.2 Proper input example1

Fig.3 Proper output example1

Fig.3 shows how the program return parse tree when given code is correct. We can see how the grammar is written. We can find the program checks the following grammar.

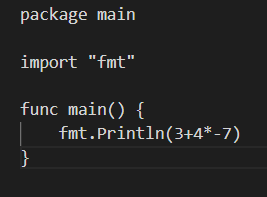


Fig.4 Proper input example2

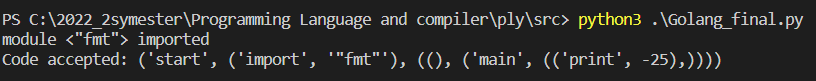


Fig.5 Proper output example2

* Precedence of the operator is well applied.

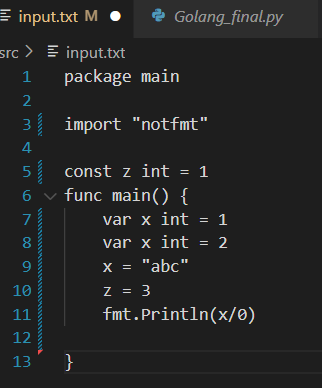


Fig.6 Error input example1

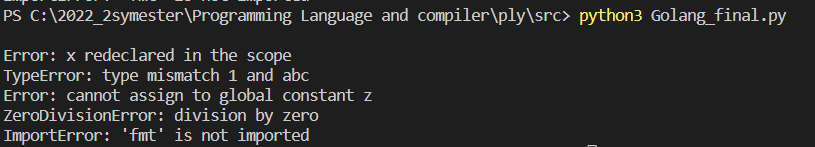


Fig.7 Error output example1

In Fig.7, we can find the program detects the errors and print the proper error statement.

* Local variable x is redeclared in the main scope. It occurs redeclaration error.
* Local variable x is reassigned in the main scope and type is mismatched.
* Global constant variable z is reassigned.
* Divide or modular with zero is prohibited. Zero division error.
* Using print without Import “fmt”. It occurs import error.

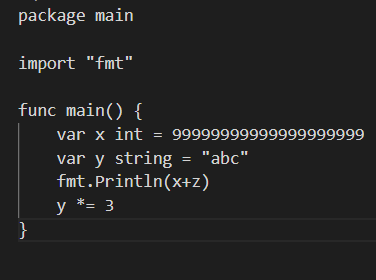


Fig.8 Error input example2

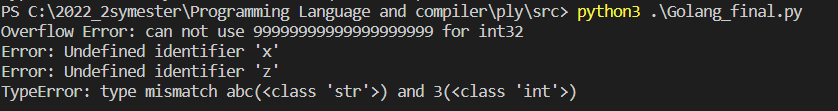
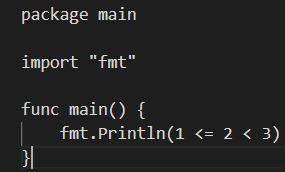


Fig.9 Error output example2

In Fig.9, we can find the program detects the errors and print the proper error statement.

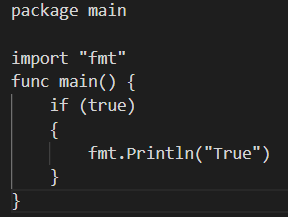
* Input integer is too big(overflow in int32). It occurs overflow error.
* Undefined identifier z is used.
* String type variable does not support \*= operator. Operator and variable type mismatched.

Fig.10 Error input and output example3



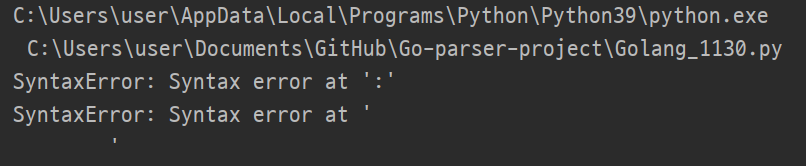
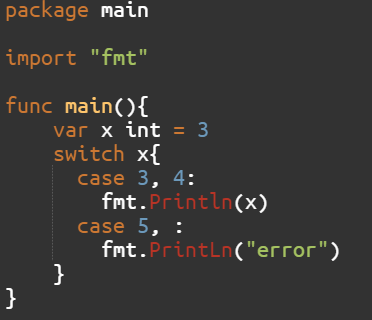
* Relational operator does not expand. (Only takes two arguments)

Fig.11 Error input and output example4



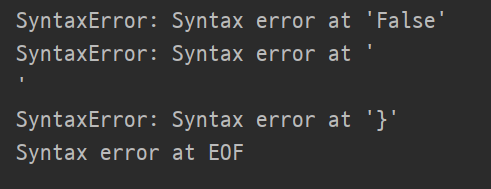
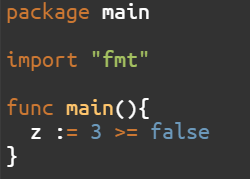
- If statement must have condition statement and open bracket “(“ in the same line.

Fig.12 Error input and output example5



* Switch statement allows multiple expression using comma (“,”)
* But, if number of comma and expression mismatches, it occurs error.

Fig.13 Error input and output example6



* Expression over relative operator must be equal.

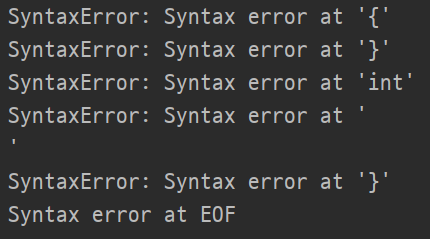
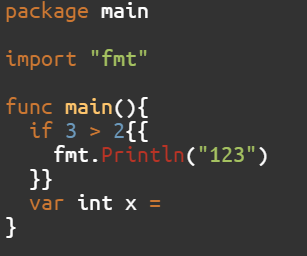


Fig.14 Error recovery example

* If error occurred, it ignores current token and determines after tokens are correct.
* Program found an error about double open bracket and close bracket.
* It also found an assign error after the if statement. ( Error recovered!)