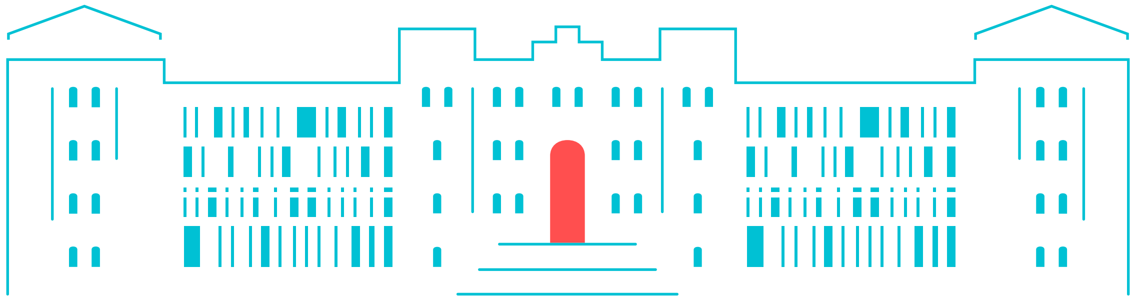
Go Front-End

TUHH
Hamburg
University of
Technology

29.06.2023



Wassim Alkhalil, Zana Gello, Ssu-Yung Yeh

Agenda

TUHH

- Introduction to Go
- Grammar
- Syntax of Go
- Semantic of Go
- Type Checking
- Conclusion

Introduction to Go

- Go is an open-source programming language.
- Developed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson.
- Statically typed and efficiently compiled.
- Robust package system promoting organized code and modularity.

Import and Package Features in Go

- Import in Go is used to import packages from other directories.
- Package in Go is used to group the source code.

```
-- IMPORT AND PACKAGE DECLARATIONS
DImport.   Def ::= "import" Library ;
DPackage.  Def ::= "package" Id ;

-- LITERALS
token Id (letter (letter | digit | '_')*) ;
token Library ('"' (letter | digit | '_' | '/' | '.')* '"') ;
```

Listing 1: import and package doclarations

Grammar Rules for Expressions

TUHH

- Statments in Go are used to control the flow of the program.
- The following are the grammar rules for the statments in Go:

```
-- STATEMENTS
SExprssion.         Statment    ::= Expression ";" ;
SDeclaration.       Statment    ::= Declaration ";" ;
SSimpleStatment.    Statment    ::= SimpleStatment ";" ;
SReturn.            Statment    ::= "return" Expression ";" ;
SReturnV.           Statment    ::= "return" ";" ;
SWhile.             Statment    ::= "for" Expression "{" [Statment] "}" ;
SFor.               Statment    ::= "for" SimpleStatment ";" Expression ";" Expression "{" [Statment] "}" ;
SForSimple.         Statment    ::= "for" Statment "{" [Statment] "}";
SBlock.             Statment    ::= "{" [Statment] "}" ;
SIf.                Statment    ::= "if" Expression "{" [Statment] "}" ;
SIfSimple.          Statment    ::= "if" SimpleStatment ";" Expression "{" [Statment] "}" ;
SIfElse.            Statment    ::= "if" Expression "{" [Statment] "}" "else" "{" [Statment] "}" ;
SIfElseSimple.      Statment    ::= "if" SimpleStatment ";" Expression "{" [Statment] "}" "else" "{" [Statment] "}" ;
```

Listing 2: statments

Grammar Rules for Basic Types

- The rule that define some basic types in Go.

```
-- BASIC TYPES
rules Type ::= "bool" | "int" | Id | "string";
```

Listing 3: basic types

Grammar Rules for Constants and Variables

- Const and Variable Declarations in Go are used to declare constants and variables.

```
-- declaration of constants or variables
rules Declaration::= ConstDeclaration | VariableDeclaration ;

DConstant. ConstDeclaration ::= "const" ConstSpecification ;
rules ConstSpecification ::= [Id] "=" [Expression] | [Id] Type "=" [Expression] ;

DVariable. VariableDeclaration   ::= "var" VariableSpecification ;
rules VariableSpecification ::= [Id] Type | [Id] Type "=" [Expression] | [Id] "=" [Expression];

rules SimpleStatment ::= ShortVariableDeclaration ;

SVarDecl. ShortVariableDeclaration ::= [Id] ":=" [Expression] ;
```

Listing 4: constants and variables

Legal and Illegal Syntax in Go

- The following are some legal and illegal syntax in Go:
  - Legal:
    - Using semicolons as statement terminators is legal, but not necessary. The line break is treated as a semicolon: ;
    - Declaring a variable: `var x int`
    - Defining a function: `func hello() {...}`
    - If-else statements: `if x > y {...} else {...}`
    - Loop structure: `for i := 0; i < 10; i++ {...}`
  - Illegal:
    - Misusing keywords: `var func int`
    - Incorrect variable declaration: `var x, y = int`
    - Variables cannot be redeclared in the same scope.
    - There is no 'while' keyword; only 'for' can be used for looping.

Syntax of Import and Package Declarations

- This Example shows the syntax of the import and package declarations and for loop in Go.

```
package main
// import "fmt"

func factorial (n int) int {
    var result int;
    if n == 0 {
        result = 1;
    } else {
        result = n * factorial(n-1);
    }
    return result;
}

func main () int {
    for i := 0; i < 10; i++ {
        var x int;
        x = factorial(i);
        // fmt.Println(x);
    }
    return 0;
}
```

Listing 5: For Loop

Syntax of While Loop and if Statement

```
package main

func counter(x int, y int) int {
    for x <= 40 {
        x += 1;
    }

    if x == 40 {
            result = x / y;
    }
    return result;
}

func main() {
    var x, y int;
        x = 30;
        y = 2;
        divideByTwo = counter(x, y);
}
```

Listing 6: While Loop

Syntax of Function, Const and Variable Declarations

```
package main

const c = 3;
func Add(a int, b int) int {
    var z int;
        z = a + b;
        return z;
}

func main () int {
        var x, y, m int;
        x = 1;
        y = 2;
        m = Add(x, y);
        return 0;
}
```

Listing 7: functions

Semantics

- Semantic is the process of checking if the code is semantically correct.
- Semeantic errors are caused by logical flaws, incorrect conditions, or improper use of variables and language constructs.
- Even though code may be syntactically correct, it can still be erroneous if its semantics are incorrect.

Semantic of While Loop

- The semantic error shows that the type of the expression in the for loop is not boolean.

```
func main () int{
        var a int;
        var sum int;
        sum = 0;
        a = 0;
        for a = 10 {
                sum += a;
                a += 1;
        }
}
```

Listing 8: While Loop

```
GROUP-06@debian:~/Documents/FrontEnd/P4$ ./compiler Tests/illegal/wrongWhile.go
TYPE ERROR
Error *** in function main: type 'bool' mismatched with type 'int'
```

Semantic of For Loop

- the semantic error shows that second expression in the for loop is not boolean.

```
package main

func main () int {
        var sum int;
        sum = 1;
        for i := 0; i + 10; i++ {
                sum += i;
        }
        return 0;
}
```

Listing 9: For Loop

```
GROUP-06@debian:~/Documents/4-frontend/P4$ ./compiler Tests/illegal/wrongFor.go
TYPE ERROR
Error *** in function main: type 'bool' mismatched with type 'int'
```

Semantic of Return Statement

- The semantic error shows that the type of the return statement is not the same as the function return type.

```
func Add(a int, b int) int {
        z = a + b;
        return;
}
func main () int {
        Add(1,1);
        return 0;
}
```

Listing 10: Return Statement

```
GROUP-06@debian:~/Documents/4-frontend/P4$ ./compiler Tests/illegal/wrongReturnType.go
TYPE ERROR
Error *** in function Add: type 'int' mismatched with type 'void'
```

Summary

- The grammar of the subset of Go language is completed.
- We will continue to work on the test cases and complete the type checker.
- Short live demo of the compiler.

Thank You For Your Attention!

Questions?