VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PRINCIPLES OF PROGRAMMING LANGUAGES - CO3005

ASSIGNMENT 3

Static Checker

HO CHI MINH CITY, 3/2025

ASSIGNMENT 3

Version 1.0

After completing this assignment, you will be able to

- explain the principles how a compiler can check some semantic constraints such as type compatibility, scope constraints,... and
- write a medium (300 - 500 lines of code) Python program to implement that.

1 Specification

In this assignment, you are required to write a static checker for a program written in MiniGo. To complete this assignment, you need to:

- Read carefully the specification of MiniGo language
- Download and unzip file assignment3.zip
- If you are confident on your Assignment 2, copy your MiniGo.g4 into src/main/minigo/parser and your ASTGeneration.py into src/main/minigo/astgen and you can test your Assignment 3 using MiniGo input like the first two tests (400-401).
- Otherwise (if you did not complete Assignment 2 or you are not confident on your Assignment 2), don't worry, just input AST as your input of your test (like test 402-403).
- Modify StaticCheck.py in src/main/minigo/checker to implement the static checker and modify CheckSuite.py in src/test to implement 100 testcases for testing your code.

2 Static checker

A static checker plays an important role in modern compilers. It checks in the compiling time if a program conforms to the semantic constraints according to the language specification. In this assignment, you are required to implement a static checker for MiniGo language.

The input of the checker is in the AST of a MiniGo program, i.e. the output of the assignment 2. The output of the checker is nothing if the checked input is correct, otherwise, an error message is released and the static checker will stop immediately.

For each semantics error, students should throw corresponding exception given in StaticError.py inside folder src/main/minigo/checker/ to make sure that it will be printed out the same as expected. Every test-case has at most one kind of error. The semantics constraints required to check in this assignment are as follows.

2.1 Redeclared Variable/ Constant/ Parameter/ Type/ Function/ Method/Prototype/Field

A declaration must be unique within its scope, as formally defined in the MiniGo specification. Otherwise, the exception `Redeclared(<kind>, <identifier>)` is raised, where `<kind>` specifies whether the identifier in the second declaration is a `Variable`, `Constant`, `Parameter`, `Type`, `Function`, `Method`, `Prototype`, or `Field`. The scopes of variables, constants, parameters, and functions are defined in the MiniGo specification. Types have a global scope. The scope of a field and a method is limited to the struct they belong to. The scope of a prototype is limited to the interface it belongs to. The scope of a variable declared in the initialization of a for loop is the loop body.

2.2 Undeclared Identifier/Function/Method/Field

- The exception `Undeclared(Identifier(), <identifier-name>)` is raised when an identifier is used but its declaration cannot be found. The identifier can be a variable, constant, or parameter.
- `Undeclared(Function(), <function-name>)` is raised if no function with that name exists.
- `Undeclared(Method(), <method name>)` is raised if no method with that name exists in the corresponding receiver (struct/interface).
- `Undeclared(Field(), <field name>)` is raised if no field with that name exists in the corresponding struct.

2.3 Type Mismatch

An expression/statement must conform the type rules for expressions/statements, otherwise the exception **TypeMismatch**(<expression>/<statement>) is released. The type rules are as follows:

- For an assignment statement, the left-hand side (LHS) must have a type other than `VoidType`. The right-hand side (RHS) must either have the same type as the LHS or be an integer type when the LHS is a float type. If the LHS has an array type, the RHS must also have an array type of the same size, and its element type must either be the same as or an integer type when the element type of the LHS is a float type. Additionally, if the LHS has an interface type, the RHS may have a struct type, provided that the struct type implements all prototypes declared in the interface. If the LHS is an undeclared scalar variable, it is implicitly declared with the type of the RHS.

- For a variable declaration, if a variable declaration includes an initialization expression, the declared type and the type of the initialization expression must follow the type rules for assignment described above.
- The type of a conditional expression in an if or in a for statement must be boolean. The type of initialization and update parts of a for loop must conform the type rules of for assignments and variable declarations, respectively. The type of components in a for loop in a range is described in the MiniGo Specification.
- A call statement must invoke a function/method with a return type of VoidType. Conversely, a function/method call within an expression must return a type other than VoidType. The number of arguments in the call must match the number of parameters in the function/method definition. Each argument must have the exact same type as its corresponding parameter. For a method call, the receiver must have a struct or interface type.
- If the return type of the function is VoidType, the return statement must not include an expression. Otherwise, the return expression must have the exact same type as the function's return type.
- For an array subscripting expression E1[E2]...[En], E1 must be of an array type, and each index E2, ..., En must be of an integer type.
- For a field access E1.<name>, the E1 must be of a struct type.
- For a binary and unary expression, the type rules are described in the MiniGo specification.

3 Plagiarism

You must complete the assignment by yourself and do not let your work seen by someone else. If you violate any requirement, you will be punished by the university rule for plagiarism.

4 Submissions

This assignment requires you submit 2 files: StaticCheck.py containing class StaticChecker with the entry method check, and CheckSuite.py containing 100 testcases.

File StaticCheck.py and CheckSuite.py must be submitted in "Assignment 3 - Submission".

The deadline is announced in course website and that is also the place where you MUST submit your code.