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, 2/2024

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 D96. To complete this assignment, you need to:

- Read carefully the specification of ZCode language
- Download and unzip file assignment3.zip
- If you are confident on your Assignment 2, copy your ZCode.g4 into src/main/zcode/-parser and your ASTGeneration.py into src/main/zcode/astgen and you can test your Assignment 3 using ZCode 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/zcode/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 ZCode language.

The input of the checker is in the AST of a ZCode 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/zcode/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/ Parameter/ Function

The declaration must be unique in its scope which is formally described as in ZCode specification. Otherwise, the exception `Redeclared(<kind>, <identifier>)` is released, where `<kind>` is the kind (`Variable`/`Parameter`/`Function`) of the identifier in the second declaration.

2.2 Undeclared Identifier/Function

- The exception `Undeclared(Identifier(), <identifier-name>)` is released when there is an identifier is used but its declaration cannot be found. The identifier can be a variable or a parameter.
- `Undeclared(Function(), <function-name>)` is released if there do not exist any function with that name. The function usage (as the function call) could not be allowed before its declaration.

2.3 Type Mismatch In Expression

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

- For an array subscripting (index operator) E1[E2], E1 must be in array type and E2 must be a list of number.
- For a binary and unary expression, the type rules are described in the ZCode specification.
- For a function call <function-name>(<args>), the callee <method name> must have non-void as return type (The `VoidType` is a class representing no return anything in a function). The type rules for arguments and parameters are the same as those mentioned in a procedure call.

2.4 Type Cannot Be Inferred

An identifier could be in unknown type as variables in `var` and `dynamic` form or the return type of a function. If an identifier is used but its type has not been inferred yet, the exception TypeCannotBeInferred(<statement>) will be released. To infer the type of a variable or the return type of a function, ZCode reads the program from the beginning to the end and applies the following rules:

- When a variable is initialized in its declaration, the type of the variable is also the type of the initialized expression.

- The type of an identifier (variable or function) must be inferred in the first appearance of the identifier's usage of and cannot be changed. If its type cannot be inferred in the first usage, the innermost statement containing the first use of the identifier is sent with the exception.
- If an expression can be inferred to some type but some of its components cannot be inferred to any type, the innermost statement containing the type-unresolved component will be associated with the exception. For example, the expression in the right hand side of the statement `y <- a + foo(x)` can be inferred to type number as the result of + is in type number and y, a and the return type of foo can also be inferred to type number, but we cannot infer the type of x, then the exception is raised with the assignment statement.
- A call statement to a type-unresolved function is valid when all its parameter types can be inferred by the corresponding argument types and its return type can be inferred to `VoidType`. If there exists at least one type-unresolved parameter, the exception is raised with the call statement. Note that if the number of the arguments is not the same as the number of the arguments, the exception concerned in Section 2.5 is raised.
- A function call to a type-unresolved type function is valid if all its parameter types and the return type can be resolved. Otherwise, the innermost statement containing the function call is associated to the exception. Note that if the number of the arguments is not the same as the number of the arguments, the exception concerned in Section 2.3 is raised.
- The types of both sides of an assignment must be the same so that if one side has resolved its type, the other side can be inferred to the same type. If both sides cannot be resolved their types, the exception is raised with the assignment.
- For each statement, all variables appear in the statement, must have type resolved otherwise the innermost statement containing the type-unresolved variable will be associated with the raised exception.

2.5 Type Mismatch In Statement

A statement must conform the corresponding type rules for statements, otherwise the exception **TypeMismatchInStatement**(<statement>) is released. The type rules for statements are as follows:

- The type of a conditional expression in an if or in a for statement must be boolean.
- For an assignment statement, the left-hand side can be in any type except void type. The right-hand side (RHS) is either in the same type as that of the LHS or in the type that can coerce to the LHS type. When LHS is in an array type, RHS must be in array type whose size is the same and whose element type can be either the same or able to coerce to the element type of LHS.

- For a call statement <method name>(<args>), the callee must have `VoidType` as return type. The number of arguments and the number of parameters must be the same. In addition, the type of each argument must be the same as the corresponding parameter.
- For a return statement, if the return type of the enclosed function is `VoidType`, the expression in the return statement must be empty. Otherwise, the type of the return expression must be the same as the return type of the function.
- For a declaration, if there is the initialization expression in a variable declaration, the type of the declaration and initialization expression must conform the type rule for an assignment described above.

2.6 No definition for a function

In ZCode, a function could be introduced as a declaration without the body (as the definition). If exists a function without the definition in the program, the exception `NoDefinition(<name>)`, `<name>` is the name of function with the first appearance is not defined.

2.7 Break/Continue not in loop

A break/continue statement must be inside directly or indirectly a loop otherwise the exception **MustInLoop**(<statement>) must be thrown.

2.8 No entry point

There must be a function whose name is **main** without any parameter and return nothing in a ZCode program. Otherwise, the exception **NoEntryPoint()** is released.

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.