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, 3/2022

ASSIGNMENT 3

Version 1.1

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 D96 language
- Download and unzip file assignment3.zip
- If you are confident on your Assignment 2, copy your CSlang.g4 into src/main/CSlang/-parser and your ASTGeneration.py into src/main/CSlang/astgen and you can test your Assignment 3 using CSlang 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/CSlang/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 CSlang language.

The input of the checker is in the AST of a CSlang 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/CSlang/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/Attribute/Class/Method/Parameter

An identifier must be declared to be used in its scope. The scope of an identifier (variable, constant, attribute, class, method, parameter) is informally described as in Section 7 of CSlang specification. However, the declaration must be unique to be resolved. Particularly, a class name must be unique in the whole program. A member name must be unique in the class where it is declared. A parameter/variable/constant name declared in the body of a method must be unique. A variable/constant name declared in a block must be unique. Otherwise, the exception **Redeclared**(<kind>,<identifier>) is released, where <kind> is the kind (Variable/Constant/Attribute/Class/Method/Parameter) of the identifier in the second declaration.

2.2 Undeclared Identifier/Attribute/Method/Class

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, constant or parameter. Exception Undeclared(Class(),<class name>) is released in similar situation for a class usage. Exception Undeclared(Attribute(),<attribute name>) is released when there is an access to an attribute of a class but there is no declaration of the attribute of the class. Exception Undeclared(Method(),<method name>) is released in similar situation for a method invocation

Note that CSlang is an OO language so all instance members of a class may be inherited by its subclasses.

2.3 Cannot Assign To Constant

The left-hand side (LHS) of an assignment statement operator cannot be declared as a constant (immutable field/variable), otherwise, the exception **CannotAssignToConstant**(<statement>) is released.

An assignment operator may appear in an assignment or a for statement. If the error happens in an assignment statement, the assignment statement is passed in the error message. In the case of a for statement, just the assignment part in this statement is printed out in the error message.

2.4 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 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 E.<method name>(<args>), E must be in class type; the callee must have void as return type. For a static call statement <method name>(<args>), where the method name is at identifier, the called must have void as return type. In addition, for parameter passing, the rule for an assignment is applied to parameter passing where a parameter is considered as the LHS and the corresponding argument is the RHS. The number of parameters must be equal to the number of arguments.
- For a return statement, the return expression can be considered as RHS of an implicit assignment whose LHS is the return type.

2.5 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 subcripting E1[E2], E1 must be in array type and E2 must be integer.
- For a binary and unary expression, the type rules are described in the CSlang specification.
- For a method call E.<method name>(<args>), E must be in class type; the callee <method name> must have non-void as return type. For a static method call <method name>(<args>), where the <method name> is at identifier, the return type must be non-void type. The type rules for arguments and parameters are the same as those mentioned in a call statement. The number of parameters must be equal to that of arguments.
- For an attribute access E.id, E must be in class type.
- For a new expression, the corresponding class must have a constructor method whose number of parameters must be equal to that of arguments and the type of each argument can be coerced to the type of corresponding parameter.

2.6 Type Mismatch In Declaration

The exception **TypeMismatchInDeclaration**(<Decl>) must be released if one of these following requirements does not satisfy:

- If there is the initialization expression in a variable/constant/attribute declaration, the type of the declaration and initialization expression must conform the type rule for an assignment described above.
- A constant must be declared with an initialization expression.
- The type of a variable, constant, parameter or attribute cannot be void.

For example, the following declarations cause type mismatch in declaration error.

```
const a: int = 1.2;
const b: float;
var c: void;
```

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 Illegal Array Literal

All literals in an array literal must be in the same type otherwise the exception **IllegalArrayLiteral**(<array literal>) must be thrown. For example, [1, 2.0] is an example of this error.

2.9 No entry point

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

3 Submissions

This assignment requires you submit 2 times:

3.1 The first submission

This first submission requires you submit 2 files: StaticCheck.py containing class StaticChecker with the entry method check, and CheckSuite.py containing 20 testcases. In this submission, you just check the following errors:

- Redeclared Class
- Redeclared Attribute
- Redeclared Method
- No entry point

3.2 The second submission

The second submission also requires you submit the same 2 files: StaticCheck.py and CheckSuite.py. These files are extension version of the first submission where StaticCheck.py must check all the required errors and CheckSuite.py includes 100 test-cases.

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

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

4 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.

5 Change log

- Modify from 1.0
  - Section 2.1: "a class name must be unique in the whole program" and "A member name must be unique in the class where it is declared."
  - Section 2.9: @main