Project 3: Static Checker

Peng-Sheng Chen

Department of Computer Science
National Chung Cheng University
2018

Type Checking

- Type checking is the processes of identifying errors in a program based on explicitly or implicitly stated type information.

Static Checker

- In the third project, you need to implement a static checker for your C subset.
- The checker needs to:
    - Create a symbol table
    - Insert the type of each variable into the symbol table.
    - Perform the type checking for each expression.

Type Checking Rules

- Each variable must be declared before it is used.
- Each identifier can be only declared once.
- The types of the operands of an operator must be the same.
- The types of the two sides of an assignment must be the same.
- … **You can add your rules for your C subset.**

Type Checking (2)

- The type checker needs to report an error message for each type error detected.
- Each type error message should contain the line number where the error is detected and an explanation of the error.
- The format for printing a type error message is as follows:

“Type Error:” line number “:” the error message.

Example

```
1.  void main()
2.  {
3.     int num;
4.     int s;
5.     int index;
6.     float s;
7.
8.     k = 0;
9.     num = index + 3.21;
10. }
```

Type Error: 6: Redeclared identifier.

Type Error: 8: Undeclared identifier.

Type Error: 9: Type mismatch for the operator + in an expression.

Type Error: 9: Type mismatch for the two sides of an assignment.

請繳交至ECOURSE

- A file describes your type checking rules and your C subset. (MS-WORD file)
- The source codes:
  - ANTLR grammar file, myChecker.g.
  - A program to call your static checker, myChecker_test.java.
  - Testing programs. (at least 3 programs)
- A readme file (pure text file) describes how to compile and execute your type checker.
- A “Makefile”.
- **Due Date: May 30 (Wednesday), 24:00pm, 2018.**

Example (1-1)

```
grammar myChecker;

options {
    language = Java;
}

@header {
    // import packages here.
    import java.util.HashMap;
}

@members {
    boolean TRACEON = false;
    HashMap<String,Integer> symtab = new
HashMap<String,Integer>();

}
```

symtab

ID <String>	Data type <Integer>

Example (1-2)

```
type returns [int attr_type]
     : INT    { attr_type=1; }
     | FLOAT { attr_type=2; }
     | VOID
     | CHAR
     ;
```

Example (1-3)

```
declarations: type Identifier ';' declarations
{
   if (symtab.containsKey($Identifier.text)) {
      System.out.println("Type Error: " +
                                  $Identifier.getLine() +
                                 ": Redeclared identifier.");
   } else {
   /* Add ID and its attr_type into the symbol table. */
    symtab.put($Identifier.text, $type.attr_type);
   }
};
```

Example (1-4)

```
statement returns [int attr_type]
    : Identifier '=' arith_expression ';'
{
   if (symtab.containsKey($Identifier.text)) {
      attr_type = symtab.get($Identifier.text);
   } else {
      /* Add codes to report your error and handle this error */
      attr_type = -2;
   }

   if (attr_type != $arith_expression.attr_type) {
      System.out.println("Type Error: " +
                          $arith_expression.start.getLine() +
                         ": Type mismatch for the two silde operands
in an assignment statement.");
      attr_type = -2;
   }
};
```

Generate Your Programs

- Generate a parser and lexer.

```
% java -cp antlr-3.4-complete.jar \
org.antlr.Tool myChecker.g
```

↓

```
Generate 3 files:
- myCheckerLexer.java
- myCheckerParser.java
- myChecker.tokens
```