

Assignment Description

In this programming assignment, you will implement the semantic analysis phases for IC, starting with construction of the symbol tables. We expect you to build upon the code that you wrote in the previous programming assignments. You are required to implement the following:

Symbol Tables and Types. Design a hierarchy of symbol tables and a hierarchy of types. Your design should allow each AST node to access the symbol table corresponding to its current scope (e.g. class, method, or block scope), and each entry in the symbol table should have information about the type of the identifier stored in that entry. Any errors which occur during symbol table construction (such as multiply declared identifiers) are considered semantic errors. *You can construct the symbol tables either during the parsing phase, or separately, in a subsequent phase.* Your constructed symbol tables should be available to all remaining phases of the compiler. In the rest of your compiler, you will refer to program symbols (e.g., variables, methods, etc.) using references to their symbol table entries.

Semantic Checks. After you have constructed the AST and the symbol tables, your compiler will analyze the program and perform semantic checks. These semantic checks include: (1) scope rules (Section 10 in the IC specification); (2) type-checking rules (Section 15), including a check that the class hierarchy is a tree and checking correct overriding of instance methods in subclasses; (3) checking that the program contains a single `main` method with the correct signature, (4) that `break` and `continue` statements appear only inside loops, (5) that the `this` keyword is only used in instance methods, and (6) that the library class has the correct name.

You are **not** required to check that a local variable is used only after it has been initialized and that a method with a non-void return type returns a value on every control path.

Error Handling. When your compiler encounters an error it should report information about the error, such as the token and the line number where the error has occurred, or a message describing the violated semantic rule. It is not required to report more than one error; the execution may terminate after the first lexical, syntactic, or semantic error.

Command line invocation: Your compiler must be invoked with a single file name as argument:

```
java IC.Compiler <file.ic> [options]
```

With this command, the compiler will parse the input file, construct the AST and symbol tables, will perform the semantic checks, and will report any error it encounters. Your compiler must also support two command-line options to dump internal information about the AST and the symbol tables:

1. The “`-print-ast`” option: the compiler will print at `System.out` a textual description of the AST for the input file. This is the same option as the one in the previous assignment, with the addition of type and symbol table information for each AST node. Each AST node will be printed on a separate line, and must have information about what the children nodes are (for instance, using numerical identifiers for the nodes). Each node will provide all its internal information (see the following example). All its children will be printed indented with respect to the position of the parent node. Please, use tab or 4 spaces for indentation. Example:

```
CLASSDECL name=Integer
fields:
FIELDDECL type=int name=m
FIELDDECL type=boolean[] name=dummy
methods:
```

```
METHODDECL name=toString ret_type=string
formal arguments:
...// dont forget the indentation!
...
```

Feel free to add properties, it's just a guideline, not a reference.

2. The “-dump-symtab” option: the compiler will print a textual description of the symbol tables and the global type table. Each entry in the symbol table will be printed on a separate line; the information for each entry should include the name of the identifier, its kind (variable, method, etc.), and its type. Different symbol tables will be separated by blank lines. You must indicate, for each symbol table, what are its children tables. When printing out the type table, print each type on a separate line. For each type include its name, its id, and the id of its superclass if it exists. For example:

```
BASE name=int id=1
CLASS name=Math id=2
...
CLASS name=AdvancedMath id=3 superid=2
```

You are also expected to design a suite of tests (IC programs) that demonstrate the correctness of your semantic analysis and detection of various kinds of errors.

Package Structure: You will implement the new components of the compiler as sub-packages of the IC package. You will have a sub-package for each of the following: 1) the symbol tables, 2) the representation of types, and 3) the semantic checks.

What to Turn In

As in any other large program, much of the value in a compiler is in how easily it can be maintained. For this reason, a high value is placed on both clarity and brevity – both in documentation and code. Turn in on paper at class (or mailbox 268):

- A brief, clear, and concise document describing the your code structure and testing strategy. Include in this document a description of your type hierarchy, symbol table representation, and a description of the semantic analysis your compiler performs.
- Feedback. As in the first assignment, please provide one paragraph to tell us your overall thoughts about the assignment: how much time you spent on it, what was the most difficult or more interesting part, and how you think it could be made better.

Electronic Submission Instructions. Turn in your code electronically in your team account on the due date. Please include only the source files and your test cases in your submission. Please organize your top-level directory structure as follows:

- `src` - all of your source code.
- `doc` - documentation, including your write-up and a `PA3-README.TXT` containing a description of the class hierarchy in your `src` directory, brief descriptions of the major classes, any known bugs, and any other information that we might find useful when grading your assignment.
- `test` - any test cases you used in testing your project.

Note: Failure to submit your assignment in the proper format may result in deductions from your grade.

GOOD LUCK!