Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu), Classes: MW 4:40-6:30pm, DLC304, Office Hours: MW 2-3pm.

Project 5: Type Checker

(Due Wednesday 11/30/11 – Firm Deadline!)

This project is to implement a type checker for the MINI compiler, in the form of a visitor to the AST nodes. Upon visiting an AST node, the visitor verifies that the components of the node satisfy MINI's semantic rules. It throws a `TypeException` with proper error message enclosed when an violation is found. For a correct program, no output is generated.

You should name your type-checking visitor program `TypeVisitor.java`. Note that this program is depedent on a symbol table being available. You may start programming this project at any time, but you'll need your `SymbolVisitor.java` program from project 4 to test it. (A refrerence version will be released after everyone turns in project 4.)

MINI's Type-Checking Rules

The highlights of MINI's type-checking rules are shown below. For cases that are not covered here (shouldn't be many), consult MINI's manual (and Java's rules if needed).

- Every name (class name, method name, variable name, etc.) used in a MINI program must be declared first. (Pay attention, in particular, to class names in declarations, since `SymbolVisitor` of the last project allows a variable to be declared with an unchecked class name.)
- Arithmetic operations must have `int` or `float` operands; logical operations must have `boolean` operands; equality comparisons ("`==`" and "`!=`") must have operands with the same (arbitrary) type; other relational operations must have `int` or `float` operands. In an `int`/`float` mixed operation, the `int` operand will be coerced to `float` before the operation.
- The object with which a method or a field is accessed must be a class object; the object with which an indexing operation is performed must be an array object; and array indices must be of type `int`.
- The lhs of an `assignment` must be an l-value, i.e. it must be an `Id`, a `Member`, or an `ArrayElm` node; the rhs must either be of the same type or a subtype of the lhs.
- The test of `while` and `if` must be of type `boolean`.
- The number and types of actual arguments to a method call must match those of the method's formal parameters, although an actual argument could be of a subtype of its corresponding formal parameter.
- A return value for a method must match that of the declared type; a method with a non-void return type must contain a return statement.
- The argument of `System.out.println` must be of a basic type (i.e. `int`, `float`, or `boolean`) or a string literal. Here we have a small technical issue. The TypeVI interface requires that the visit routine for each expression node returns a `Type` object, yet there is no corresponding type for `StrVal` in MINI's AST system. We use a trick to handle this case: we borrow `BasicType(BasicType.Int)` to represent the type of a `StrVal` node. Since `StrVal` nodes can only appear in a print statement, this is not going to cause any problem.

MINI's Type-Equivalence Model

You need to write a routine that implements MINI's type-equivalence model. This routine is needed at several places in the visitor program, e.g. assignment statements, parameters to a method call, and initial values in variable declarations. The following is a sketch of this routine.

```
private boolean compatible(Type t1, Type t2) throws Exception {
  // if t1==t2 or both are the same basic type
  //   return true
  // else if both are ObjType   // name equivalence
  //   if (their class ids are the same, or
  //       t2's cid matches an ancestor's cid of t1)
  //     return true
  // else if both are ArrayType // structure equivalence
  //   recursively test the compatibility of their elements' types
  // else
  //   return false
}
```

Project Organization

Copy and decompress the file `proj5-code.tar`. You'll see a `proj5` directory and several subdirectories:

- `ast` and `astpsr` — containing AST nodes and an AST parser; no change from Project 4.
- `symbol` — containing symbol table definitions. You need to copy your or the reference `SymbolVisitor.java` program into this subdirectory when it becomes available.
- `typechk` — containing a driver program `TestType.java` and an exception definition file `TypeException.java`.
- `tst` — containing tests and their expected output; a new set of programs, `errt*.ast`, provide a broad coverage of type errors.

There is also a `Makefile` and a `runt` script. To run `TypeVisitor` on `tst/test01`, you do:

`./runt tst/test01.ast`.

Submission

Submit your program `TypeVisitor.java` through the "Dropbox" on the D2L class webpage.