Project 3: The MiniJava Typechecker

Due: Tuesday, Nov. 14, 2006 by 11:00 pm.

In this assignment you will extend the initial MiniJava typechecker with the extensions described in the course project description handout.

You should complete the front-end processing for the extended MiniJava language, checking any legality constraints not handled earlier in the scanning or parsing phases. These include the following additions to the hierarchies in the SymbolTable package:

- Extend the `ResolvedType` hierarchy to support the `double` type.
- Extend the `ResolvedType` hierarchy to support the array type constructor, which stores its element type. The array type constructor follows structural type equivalence rules. MiniJava restricts Java by defining one array type to be a subtype of another array type only when the two array types are equivalent.
- Extend the `VarInterface` hierarchy to support static class variable declarations.

You should implement typechecking for the new and/or modified AST node classes, including the following:

- Allow static class variables to be declared, so that they may be legally referenced in variable reads and assignments.
- Allow `if` statements to omit their `else` clause.
- Check that a `for` statement's loop index variable was previously declared to be an `int`, that its initialization and update expressions return `int`s, and that its test expression returns a `boolean`.
- Check that a `break` statement only appears in the body of a `while` or `for` loop. (You may change the interface of the `Stmt.typecheck` operation to do this.)
- Check that an or (`||`) expression has `boolean` operands.
- Check that an array new expression has a size subexpression of type `int`.
- Check that an array length expression has an array subexpression that's an array.
- Check that an array lookup expression has an array subexpression that's an array and an index subexpression that's an `int`.
- Check that an array assignment statement has an array subexpression that's an array, an index subexpression that's an `int`, and a right-hand-side expression whose type is assignable to the array's element type.
- Allow `int`s to be assignable to `double`s, including in regular assignments, in array assignments, in parameter passing into a method, and in returning from a method.

- Allow the `+`, `-`, `*`, `/`, `<`, `<=`, `>=`, `>`, `==`, and `!=` operations to also be applied to `double`s, and, for binary operations, to mixes of `int`s and `double`s.
- Allow the `System.out.println` operation to also be applied to a `double`.

In all cases, as long as the MiniJava restrictions are satisfied, a MiniJava expression should have the same result type as the equivalent Java expression.

You only need to get the compiler front-end to work. You do not need to implement any back-end lowering or code generation.

Do the following:

1. Add and/or modify classes in the AST and SymbolTable subdirectories to typecheck the extended language.
2. Develop test cases that demonstrate that your extended typechecker works, both in cases that should now be legal and in cases that should be syntactically legal but semantically illegal. (Since the typechecker quits at the first error, you'll likely need several illegal test case files to test the different illegal cases.) You do not need to check for lexical or syntactic errors, just semantic errors. The `SamplePrograms` directory contains some files that should typecheck after you make your changes; some of the files should typecheck successfully with the initial version of the MiniJava compiler.

You can use the `-typecheck -printSymbolTables` options to the MiniJava compiler to just run the typechecking phase and print out the top-level symbol tables that it builds. See the `test_typechecker` target in the `Makefile` for an example, and feel free to make your own target(s) to make running the tests you like easier and more mechanical.

Turn in the following:

1. Your new and/or modified `AST/*.java` and `SymbolTable/*.java` files. Clearly identify any modifications to existing files using comments.
2. Your test cases, with names of the form *`name`*`.legal.java` for test cases that should typecheck successfully and *`name`*`.illegal.java` for test cases that should trigger typechecking errors.
3. A transcript of running your typechecker and printing out the resulting symbol tables on each of your test cases.

Create a single directory by the due date.