Project 4: The MiniJava Intermediate Code Generator

Due: Wednesday, November 28, 12:30 pm.

In this assignment you will extend the initial MiniJava intermediate code generator to implement the extensions described in the course project description handout. This will complete your extended MiniJava compiler!

You should implement the `lower` operations for all the new language constructs and features in the extended MiniJava language. This includes inserting coercions from integers to doubles wherever necessary, as well as generating tests for null array references, out-of-bounds array references and negative-sized array creations. To support `System.out.println` on doubles, a new runtime function should be added to the `Target/Runtime/runtime.c` file. The `lower` operations should use the IL classes defined in the `IL` subdirectory, but you should not make any changes to these IL classes.

After lowering, your lowered IL program should be able to be translated into x86 target code, which then compiles, links, and runs correctly. (The x86 code generator is already fully implemented; you should not make any changes to it.)

In all cases, as long as the MiniJava language restrictions are satisfied, a MiniJava program should compile into an executable program that then runs the same as the equivalent Java program.

Do the following:

1. Add and/or modify classes in the `AST` and/or `SymbolTable` subdirectories to perform lowering, and modify the `Target/Runtime/runtime.c` file to include any new runtime functions you need. (You should not modify any files in the `IL` or `Target` subdirectories, other than the `runtime.c` file.)
2. Develop test cases that demonstrate that your extended compiler works properly, both in cases that should now compile and run successfully and in cases that should now compile successfully but throw exceptions when run. (Since execution ends with the first exception, you'll likely need several excepting test

case files to test the different excepting cases.) You may assume that your test cases pass all lexical, syntactic, and semantic checks, and you may assume that all MiniJava constructs from the initial language (before your extensions) are compiled and executed correctly; you only need to test compilation and execution of the new language features. The `SamplePrograms` directory contains some files that should compile and execute successfully after you make your changes; some of the files should compile and execute successfully with the initial version of the MiniJava compiler.

You can use the `-lower -printIL` options to the MiniJava compiler to just run the lowering phase and print out the IL program that it produces. See the `test_lowering` target in the `Makefile` for an example. You can use the `-printCode` option (the `-codegen` option is the default) to the MiniJava compiler to run the full compiler and print out the assembly code that it produces. See the `test_codegen` target in the `Makefile` for an example, which also compiles the `runtime.c` file, runs the assembler on the generated assembly file, links it with the compiled `runtime.c` file, and finally runs the linked executable program. (This target should be run only on an x86 machine, so that the generated x86 assembly code can be compiled and run successfully.) 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`, `SymbolTable/*.java`, and/or `Target/Runtime/runtime.c` 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 compile and run successfully and *`name`*`.illegal.java` for test cases that should compile successfully but throw exceptions when run.
3. A transcript of running your intermediate code generator and printing out the resulting IL program (not the final assembly code) on each of your test cases.
4. A transcript of running the compiled code for each of your test cases.

As with the last project, name your root project directory MiniJava, and submit the directory. Put your test programs in the SamplePrograms directory. Zip it up and submit it.