

Assignment Description

In this programming assignment, you will implement the translation from the AST of IC programs to the LIR language specified on the [web site](#) and validate the translated programs using the microLIR simulator. We expect you to build upon the code that you wrote in the previous programming assignments. You are required to implement the following:

Translation of functions Your compiler should translate all of the functions in the program using the AST and the information gathered during the semantic analysis phase. You compiler should also read the file `libc.sig` from the current directory (default), or the path specified by the `-L` option, parse it, and add it to the AST (if you haven't already done so). The compiler should be able to distinguish between calls to user-defined functions and library functions (in `libc.sig`) and emit the correct call instructions. Of course, the compiler should avoid attempting to emit a translation for the library functions themselves, as their implementation is provided externally.

Class layouts Your compiler should maintain information for each (user-defined) class about its fields and virtual functions. This information is used to determine the offsets for accessing fields (in `MoveField` instructions) and to generate the dispatch tables. In your translation, please print in comments the offset assignment for the fields of each class.

IMPORTANT: When generating field and method offsets for a class, don't forget to account for the fields and methods of its superclass and for overridden methods.

String literals Your compiler should extract all literal strings that exist in the program and translate them to LIR string literals. Instructions using the string literals should be aware of this and use the symbolic names given to the string literals.

Runtime checks Your compiler should emit code to check fragile instructions (see LIR documentation) and handlers for runtime errors that print an error message and gracefully terminate the execution. You may choose to implement this code in this assignment or in the next assignment. Although the checks and handlers are easier to implement in LIR, the resulting code would not be as optimal as it could be if these checks are implemented directly in assembly. We will check your translation only on programs that do not trigger runtime checks.

You may choose to implement any of the following optimizations to reduce the number of registers, using the techniques taught in the recitation:

Variables, constants, accumulators (4) Avoid unnecessarily storing local variables and constants in registers and use accumulator registers.

Reusing registers (8) Use the live registers stack technique to reuse registers.

Weighted register allocation (8) Apply the [Sethi and Ullman algorithm](#) to expressions with commutative operators on sub-trees with no side-effects to find the optimal traversal order.

We recommend that you implement any optimizations only **after** you validate your assignment against a suite of tests. The most important thing is to ensure you have a baseline translation that you trust to be correct. We also recommend that you keep two versions of the translation—the optimized and non-optimized translation—separately (instead of rewriting the baseline translation with optimizations). This limits the possibility of introducing translation errors with the optimizations.

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 (and `libc.sig`), construct the AST, conduct semantic analysis (reporting any errors it encounters), and translate the program to the LIR language. Your compiler must support the command-line options specified in the previous assignments, as well as the following command line options:

1. The “`-print-lir`” option: the compiler will print **into a file**—`file.lir`—the LIR program translated from `file.ic`. The file should be a legal LIR program that can be read and executed on the microLIR simulator giving exactly the same results as we intend the IC program to give.
2. The “`-opt-lir`” option: the compiler activates the LIR optimizations you choose to implement (otherwise it has not effect). You can use the `-stats` option in microLIR to see the difference between the optimized translation and the non-optimized translation.

You are also expected to design a suite of tests (IC programs) that demonstrate the correctness of your translation and the effect of optimizations on the translation.

Package Structure: You will implement the translation to LIR in a new sub-package `lir`.

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 your code structure and testing strategy. Include in this document a description of any interesting details of your translation (such as any runtime checks you chose to include), including any optimizations you choose to perform and their algorithms.
- Feedback. As in the previous assignments, please 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, your test cases, and documentation 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 `PA4-README.TXT` containing a description 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. If you choose to implement any optimizations, please include a clear documentation of the tests you use to demonstrate the optimizations. We should be able to test the optimized and non-optimized translations of these tests, and therefore the test should print information to the standard output.

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

GOOD LUCK!