Assignment 3

CS 550 Programming Languages
Instructor: Jeremy Johnson
Due: Thursday May 2 by 9am

Mini Language Compiler (version 1) (100 points)

In this assignment you will implement a compiler for version 1 of the mini language (i.e. the version without procedures). The target langauge for your compiler will be the RAL instruction set, and you will be able to run your compiled programs on the RAM simulator from Mini Language Compiler Lecture. Note that you must first extend the RAM simulator to provide MUL and JMN instructions as outlined in the exercises from Mini Language Compiler Lecture. You should follow the approach outlined in Mini Language Compiler Lecture. You may reuse the parser and data structure constructed for the mini language interpreter from Mini Language Lecture Part II. You can simply replace the eval method by a translate method, which should construct a list (or vector) of symbolic RAL instructions. Each of the different classes that define the program data structure (e.g. Expr, StmtList, AssignStmtn, IfStmt, WhileStmt) will need a translate method. These methods are implemented using the approach outlined in Mini Langauge Compiler Lecture. In addition to the translate method you should provide an optimize method, which applies peephole optimization as outlined in Mini Language Compiler Lecture. Other optimizations may be applied but they are not necessary.

You will also need to modify the symbol table to store information such as the address in memory that a variable or constant is stored, the type of entry in the symbol table (e.g. constant, variable, temporary). Note that the address field may be "unknown".

You will also need to implement a link method which converts the symbolic RAL instructions to absolute RAL instructions (i.e. hard coded addresses). This may be done after the program is translated and all necessary information is in the symbol table (i.e. the number of constants and their values, the number of program variables, the number of temporary variables, and the number of instructions along with the instruction addresses where labels occur. The link method can be applied to unoptimized or optimized code.

Finally, a compile method simply calls the translate and link methods. The compile method should take an option that indicates whether optimization is performed. After compiling you can dump the RAL program to a file (or standard out) and then you may execute it on the RAM simulator.

Background Information

Background information is available in the lectures on Mini Language Interpreter and Mini Language Compiler.

What to do

Modify the mini language interpreter from the Mini Language Lecture Part 2 (the one without procedures) to provide

1. a translate method that produces symbolic RAL instructions to exectute the program on the RAM simulator
2. an optimize method that performs peephole optimization

3. a link method that produces absolute (i.e. hard coded addresses) code, corresponding to the translated symbolic RAL code, that can be simulated on the RAM simulator
4. a compile method which calls translate, optionally calls optimize, and then calls link
5. an output method that dumps the compiled RAL program

You must test your compiler and optimizer on several mini language programs.

How to submit

Students should submit their solution electronically via Blackboard Vista. Only one submission is required per group. The group leader for the assignment should submit the assignment.

You should provide your modified RAM simulator that you used to execute your compiled programs. The README file should describe all files that are included, contain instructions how to build [you should use make or Ant to build you program and make sure it successfully builds and runs on the CS compute server tux] and use the code, and outline how the code works. The README file should also contain a list of all group members and the group leader for the submitted assignment. You should also indicate how you tested your code. If your program is not working, you should clearly state this in the README file. Code should be documented (clear specifications and comments for any tricky parts of the code).

Since you can do this in a number of languages, submit a makefile or Ant file. Here is a sample makefile. Please ask for help w/makefiles.

Submit the following files to Bb Vista:

- **`makefile`** — your makefile, with the folowing targets:
  - `view` — display (using the `more` utility) all of your source code (excluding the modified RAM)
  - `compile` — Does whatever you need to do to produce:
    1. *symbolic* RAL code
    2. *linked* RAL code
    3. *(optional)* – You optimised RAL code

    You can name the the files as you wish. Your interpreter will read stdin, as previously.

  - `view-trans` — Use `cat` to display your *symbolic* RAL program (produced in `translate`) to stdout.
  - `view-link` — Use `cat` to display your compiled (not optimised) RAL program (produced in `translate` to stdout.
  - `view-op` — Use `cat` to display your compiled, optimised RAL program (produced in `translate` to stdout. If you didn't provide optimisation, echo "NOT IMPLEMENTED"
  - `run` — invoke `~jjohnson/bin/ram` to run your program. Let output go to stdout.
  - `run-op` — invoke `~jjohnson/bin/ram` to run your optimised program. If you didn't provide optimisation, echo "NOT IMPLEMENTED"
  - `clean` — remove all binaries and intermediate files
- all of your **source** files. Intermediate files, such as `.class`, `.o`, and executable binaries will be deleted before I start, so, must be built from source.
- **`README`** — describe all files that are included, contain instructions how to build and use the code, and outline how the code works. You should also indicate how you tested your code. If your program is not working, you should clearly state this in the README file. Code should be documented (clear specifications and comments for any tricky parts of the code).

Submit a gzipped tar file, called A5.tar.gz (the tar file should contain a directory called A5 which contains the files). The tar file should contain source code, instructions how to run your programs, sample input and output files, and a README file.

Note: Please pay attention to file names. E.g., `makefile` is *not* `Makefile`, and `README` is *not* `README.txt`.