## Specification

Have your compiler generate MIPS code for J-- files from your compiler's AST from Milestone 3 part one. The generated assembly code should be assembled and run with spim in ~aycock/411/bin. The qtspim program is also provided there as a convenience, but the final testing must be done on spim on the CPSC machines.

Test files and the reference compiler may be found in the usual location. Your compiler's MIPS output needs to produce the correct result when it's assembled and run, but does not need to match the reference compiler's MIPS output.

### All the Fine Print

* Your code must be written in C, C++, Java, or Go.
* Your parser may use any of Yacc, Bison, CUP, or Goyacc if you choose, as before, but you do not have to. This includes C, C++, and Java parsers that Bison produces. Your scanner may still use lex, flex, or JFlex as before.
* If you are using C or C++, you may ignore dynamic memory deallocation if you want.
* Your program must accept exactly one command-line argument: the pathname of an input file.
* The standard input should be ignored.
* Normal (non-error) output, in this case MIPS code, must go to the standard output.
* Error and warning messages must go to standard error. You may exit after an error message. Any errors and warnings, at a minimum, must identify the input file name, the line number (or as close to it as may be ascertained), and a human-readable diagnostic message that's as specific as possible.
* Your program must return an appropriate return value: zero on success, nonzero on failure or error.
* Under no circumstances can your program exit in an uncontrolled fashion, like a segmentation fault, bus error, or exception.
* Your code should be documented in a professional manner, and should mention any limitations that your program has.
* You should observe good coding practices; also, any compiled code of yours must compile cleanly with no warnings. (C/C++: compile with -Wall.) Idioms must be appropriate to the language you have chosen. Be sure to check for error conditions and handle exceptions appropriately.
* There are two exceptions to the “no warnings” rule above. First, if the warning is due to tool-generated code that you can't do anything about, you don't need to worry about it. Second, if the parser generator complains about a shift/reduce conflict on the dangling else, you may leave it as is. In both cases, you should be verifying that the cause of the warning is what you think it is!
* Your project submission must be compiled and tested on the CPSC machines.

I hope you've benefited from reflecting on your code, both its strengths and weaknesses, and practicing self-assessment in conjunction with feedback from your TA. For pragmatic reasons, however, taking into account that we're at the end of term along with other considerations, we'll be using a more conventional assessment for the project.

In your CPSC Gitlab repo, for each test file that you have working, have the following two items checked in:

1. The MIPS code your compiler generates for the test file;
2. The output from running your compiler's generated MIPS code through spim on the CPSC Linux machines. These should be located in separate files whose filename should tie them to the input file. For example, for the input file gen.t1, you would have the files gen.t1.out and gen.t1.s in your repo.

There are 21 test files, 1 mark each. To ward off some concerns that should be non-issues after the milestones:

* If the TA doesn't find your buildable source code and your compiler's output checked in to your CPSC Gitlab repo, you will not receive any marks for the project, sorry.
* If you have not run your compiler and provided the outputs for the TA as described above, again you won't be receiving any marks. As a programmer, it's your job to test your code.
* Your TA may decide to build and run your compiler themselves. If your code doesn't build and run on the CPSC Linux machines, you won't receive any marks.