Assignment 4: Code Generation for Clike

Worth rest of the Project grade
Due in D2L: Monday December 8th at 11:59 pm (just before Midnight)

Overview
This assignment involves generating code for clike. You will translate into three-address assignments, then translate these into assembly code for a MIPS processor and output the assembly code generated. You will then use the command-line MIPS simulator spim on lectura to run the generated code. The program will be called clike. In general, the code you generate should conform to the requirements of the Operational Characteristics handout. Additionally, it is expected that problems from previous phases of the project have been fixed before the assignment is turned in.

Output
As before, your program will take its input from stdin and generate all output on stdout. Error messages, if any, will be sent to stderr. In case any errors are encountered during lexical, syntactic, or semantic analysis, you should also stop generating code, but continue to process the user program as best as possible.

What will your program output? Legal MIPS assembly that can be run directly inside the spim simulator. Note that your output should be pure ASCII text; we are not asking you to assemble the code, that's what spim is for.

Strategy
This final program is due near the end of the semester, but this is a long assignment.

This assignment should be broken up into two assignments (Intermediate Code Generation and Final Code Generation), but the choices you make for Intermediate Code Generation may be different from your neighbor and still be perfectly valid. Also, we can only run the final MIPS code you output to grade that your programs are correct. Having said that, it is important to have an intermediate format and part of your grade on this assignment will depend on that. The intermediate format will be critical to your sanity for debugging and getting things right.

A suggested strategy to finishing the project.

1. Design the Intermediate Language (see An Intermediate Code Language for Clike handout)
2. Code up the enums and basic structs
3. Code up printing the intermediate language (three-address instructions). The intermediate language forms the core of this stage of your compiler: you want to make sure you can print out your code to debug it WAAAAAY before you actually generate MIPS code.
4. Start generating your intermediate code for expressions
5. Start generating your intermediate code for statements
6. Run many, many test cases and make sure you can generate proper output (what you expect) for the intermediate code. The legal test cases from assignment 3 are a good place to start.
7. Finish and generate code for globals
8. More testing
9. Finish and generate MIPS code
10. Make sure you don't break anything: make sure all the illegal test cases from assignment 3 still are illegal and output messages.
Before Thanksgiving, you should be comfortable generating intermediate code and printing it on the output. If you did everything right, the final code generation will just be (almost) a macro replacement of intermediate code for MIPS assembly.

## Building and Running

We will building and running your code on lectura.cs.arizona.edu. You will turn in at least 3 files: a Makefile, a Flex file named 'tokenout.l' and a Bison file named 'clike.y'. The Makefile builds an executable named clike on lectura. Like most UNIX tools, clike reads from standard input and writes to standard output:

```
% make clike
% clike < input.txt > output.txt
# reads standard input and writes to standard output
```

(We strongly suggest you support 'make clean' and 'make test' in your Makefile as well to make your life easier).

The file tokenout.l will be built using the UNIX command tool flex and create lex.yy.c which will be compiled and folded into the final executable. The clike.y should use bison to create clike.tab.c and (probably) clike.tab.h. You will want to have flex and bison call back and forth.

Your program also needs to accept a command-line option that forces it to only emit Intermediate Code:

```
% clike -im < test_program.c > intermediate_code.im
```

This way you can always examine the intermediate representation (This is similar to how gcc -E forces only the preprocessor to be run). We will be testing the -im option as part of the grading.

We will be discussing intermediate code and final code generation throughout the rest of the semester.

## Grading

To test your compiler, we will be writing a number of well-formed clike programs, compiling them with your clike compiler then running them (within spim) to see if they generate the proper output. The final goal of a compiler is to generate proper assembly code; we will be running the assembly code generated by your compiler to make sure it works as expected. Your generated MIPS assembly has to run (no segmentation faults) and output the expected answers.

We will be posting some of the tests we will use, but we encourage you to share and discuss test cases on Piazza. Some of our test cases will be available on:

```
http://www.cs.arizona.edu/classes/cs453/spring13/Testing4
```

We reserve the right to add more test cases. Note that the testing framework will be checking both the standard output and standard error. We will be running some of the assignment 3 testcases (if not all) to make sure you still are doing syntactic and semantic analysis (i.e., you haven't broken anything from assignments 1, 2 or 3).

You will also be graded on being valgrind clean: In other words, your program should have any memory leaks OR memory errors that valgrind finds.

If time permits, we may may have trophies for ‘fastest compiler’ and ‘fastest generated code’.