COMP 520 Compiler Design
Individual Assignment #2

The Rest of `MiniLang`!

Due: Tuesday, February 13, 11:59 PM

Overview:

In the second and final assignment of `MiniLang`, we will be building the remaining phases of the compiler that perform semantics checks and generate code. Before starting this assignment, you should fix all unhandled test cases from the first submission. We will not re-evaluate your scanner and parser directly, but they must accept all valid programs to be useful for generating code. If you did not manage to complete the first assignment, we will show sample solutions written `flex/bison` and `SableCC 3` in class and office hours.

Question 1: *Example Programs* (5 points)

Write the following example programs ending with file extension `.min`

(a) **Type-Incorrect Example Programs** (5 points)

Write five incorrect `MiniLang` programs which each exhibit a *different* typechecking or declaration error. Ideally, each program should be minimally sufficient to trigger the error, so think carefully about your program. Include a comment at the start of each file describing the intended issue.

Note that although we only require 5 small example programs for this question, you should prepare a more substantial test bank for debugging your submission. As part of evaluating your work, we will execute our own comprehensive test suite that covers all language features.

Question 2: *AST* (4 points)

Create the AST for your `MiniLang` compiler. For those using `flex+bison` this requires writing the boilerplate code for your AST structure and adding the appropriate parser actions. For `SableCC`, you must implement the correct CST→AST transformations.

Question 3: *Pretty Printer* (4 points)

Implement a pretty printer for `MiniLang` under the `pretty` compiler mode, outputting the pretty code to `stdout`. The pretty-printed output should be parsable by your compiler, and be equivalent to the original program. Be sure to check the invariant we saw in class

$$pretty(parse(pretty(parse(P)))) \equiv pretty(parse(P))$$

Question 3: *Symbol Table and Type Checking* (12 points)

Implement a symbol table and type checker for `MiniLang`, ensuring that you implement the semantics we decided on in class. If there is a declaration or type error, your compiler should emit an error message and exit. For this question you should implement 2 compiler modes:

- `symbol`: Outputs the symbol table to `stdout`. If an error occurs, write the message to `stderr` and exit with status code 1 (note that if an error occurs the symbol table might be incomplete)
- `typecheck`: Outputs `OK` if the input is type correct, or an appropriate error message

Question 4: *Code Generation* (10 points)

If the input program successfully typechecks, then your compiler can generate equivalent code in the target language. Implement the last compiler mode, `codegen`, which produces an output file in `C` and writes `OK` to `stdout` (no errors should occur). If the input file is named `foo.min`, then the output file should be named `foo.c` and be output to the same directory as the input file.

Code generated by this question should be self-contained and compatible with any standard `C` compiler. Make sure you test your generated code.

Submission

On myCourses, hand in a tar.gz file of the form `firstname_lastname.tar.gz` (all lowercase, please use .tar.gz). The structure of files inside that tarball should be:

```
firstname_lastname/
    README   (Name, student ID, any special directions for the TAs)
    programs/
        2-typecheck/
            invalid/ (Fve syntactically incorrect programs)
    src/     (Source code and build files)
    build.sh (Updated build script)
    run.sh   (Updated run script)
```