COMP 520 Compiler Design
Individual Assignment #2

The Rest of MiniLang!

Due: Monday, February 10, 11:59 PM

Overview

In the second and final assignment of `MiniLang`, we will build the remaining phases of the compiler including semantics and code generation. The scanner and parser will not be re-evaluated directly, but they must accept valid programs to generate code. Therefore, you should begin by fixing all unhandled test cases from the first submission. If you did not manage to complete the first assignment, we will show sample solutions written `flex/bison` and `SableCC 3` during office hours.

The typechecking specification is found at `http://www.cs.mcgill.ca/~cs520/2020/assignments/Assignment2_Specifications.pdf`. Note that this may change as we discover the intricacies of language design - report any issues/challenges you encounter and we will discuss in class!

Question 1: *Example Programs* (5 points)

Given the `Minilang` specification, write the following example programs 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. Each program should be minimally sufficient to trigger the error, so think carefully. Include a comment at the start of each file describing the intended issue.

Note that although only 5 test programs are required, you should prepare a more substantial set for debugging. As part of the evaluation, we will test a comprehensive suite for 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 the symbol table and type checker for `MiniLang`, using the semantics we decided on in class. Your compiler must implement 2 modes supplied as a command-line argument:

- `symbol`: Outputs the symbol table to `stdout`. 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.

Normal output (`OK`, symbols) must be sent to `stdout` and your compiler must exit with status code 0. If an error occurs, the message must be displayed on `stderr` and your compiler must exit immediately with status code 1. Error messages should be descriptive and formatted "`Error: <description>`". Follow the output specifications **exactly**.

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. Be sure you test your generated code!

Submission

On myCourses, hand in a tar.gz file of the form `firstname_lastname.tar.gz` (all lowercase, use .tar.gz). The structure of files inside the tarball is given below.

A `README` file should include your student ID, as well as any resources that were consulted as part of your submission (other students, websites, repositories, etc.). If no resources were consulted, please state "I worked alone". As this is an individual assignment, all code must represent your own efforts – please check with us if in doubt!

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