**COMP 520 Compiler Design**
**Individual Assignment #2**

The Rest of MiniLang! Your first compiler....

Due: Sunday, February 12

# Overview:

The purpose of this assignment is to get everyone familiar with the remaining phases of the compiler, within the context of a simple project.

Before moving onto the phases required by this compiler, you should fix all unhandled cases from the first assignment. While we will not re-evaluate your scanner or parser directly, your compiler frontend is expected to handle all valid programs. If you did not manage to complete the first assignment, we will show sample solutions written `flex/bison` and `SableCC` in class.

*Recall:* To simplify the grading and work for the TA's, please ensure that your compiler will run on the Trottier machines. A standard test script will been distributed on the course website to ensure your input and output formats conform to a common specification. Note that as part of the submission, you are required to submit modified:

- `build.sh`: to compile your compiler
- `run.sh`: to run your compiler

A README file as part of the tarball provides more detail.

# Question 1: *Type or declaration errors* (5 points)

Develop five small programs, where each program exhibits a different type or declaration error. Note: you should test your compiler on many more programs.

# Question 2: *AST and pretty print `MiniLang`* (10 points)

Implement an AST and pretty printer of the AST for `MiniLang`. Given an input program of the name `foo.min`, your compiler should write the pretty print to file `foo.pretty.min`. This pretty-printed file should be parsable by your compiler, in particular, check the invariant we saw in class:

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

## Question 3: *Symbol Table and Type Checking* (10 points)

Implement a simple symbol table and type checker for `MiniLang`, ensuring that you implement the semantics we decided in class. If there is a declaration or type error, then your compiler should emit an error message, and not proceed to generate code.

If the program being compiled is `foo.min`, then your compiler should print out your symbol table into file `foo.symbol.txt`. You should be able to print a symbol table, even if there is a declaration or type error in the program, although the symbol table information may be incomplete. This will help you in debugging.

## Question 4: *Code Generation* (10 points)

If the input program successfully passes type checking, then your compiler should produce an output `.c` file. If the input file is `foo.min`, then the output file should be `foo.c`. This output file should be compilable by any standard C compiler, without needing any other files. Make sure you test your generated code.

## What to hand in

Hand in a tar.gz file of the form `firstname_lastname.tar.gz`. The structure of the files inside that tar-ball should be:

```
firstname_lastname/
    README    (Your name, student ID, any special directions for the TA)
    programs/
       invalidtype/ (your five programs with decl or type errors)
    src/      (the source code and build files)
    build.sh  (a build script to compiler your compiler)
    run.sh    (a run script that when invoked as "run foo.min" will
               process the input file and will produce three outputs,
               foo.pretty.min, foo.symbol.txt and foo.c)
```

The .tar.gz file should be handed in via MyCourses.