CS 3323 - Principles of Programming Languages
Assignment 3

The objective of this programming assignment is to build a functional compiler with basic I/O support and arithmetic computations. Instructions are provided below.

First, download the files grammar.y, scanner.yy, icode.cc, icode.hh, symtab.cc, symtab.hh, inputs-int.tar.gz, inputs-float.tar.gz, Makefile, run-all.sh and driver.cc from the assignment directory space. Then, perform the necessary modifications to grammar.y and to icode.cc to add support for floating point operations. You should only work on the grammar file (grammar.y) and in icode.cc.

The assignment is due on April 1st, 2020, 11:59pm. Both modified files (grammar.y and icode.cc) must be uploaded by then. Late policy deduction applies.

Perform the following modifications to the compiler:

1. (2.0pt) In grammar.y, complete the semantic actions corresponding to the floating point arithmetic binary operations of the non-terminals *a_expr* and *a_term*: (OP_FADD, OP_FSUB, OP_FMUL and OP_FDIV). You should query the **datatype** field of a temporary symbol to determine the correct operation to be generated.

2. (1.0pt) In icode.cc, function **run**, implement the run-time actions for the floating point arithmetic binary operations OP_FADD, OP_FSUB, OP_FMUL and OP_FDIV. These four operations are specific to the datatype DTYPE_FLOAT.

3. (0.5pt) In icode.cc, function **run**, complete the run-time action of the floating point arithmetic unary operation OP_UMIN for the DTYPE_FLOAT datatype (**float**). The current implementation only supports DTYPE_INT (a C/C++ int).

4. (0.5pt) In icode.cc, function **run**, complete the run-time action of the operation OP_LOAD_CST for the datatype DTYPE_FLOAT. It should be very similar to the DTYPE_INT case.

5. (0.5pt) In icode.cc, function **run**, complete the run-time action of the operation OP_LOAD for the datatype DTYPE_FLOAT. It should be very similar to the DTYPE_INT case.

6. (0.5pt) In icode.cc, function **run**, complete the run-time action of the operation OP_STORE for the datatype DTYPE_FLOAT. It should be very similar to the DTYPE_INT case.

A number of test cases are provided, both for the **int** (directory **inputs-int**) and **float** (directory **inputs-float**) language datatypes. The current implementation of the provided compiler works with all the test cases of the directory **inputs-int**. You can test your progress with the files in **inputs-float**. Note that the output should be practically identical to the corresponding **int** test case.

For convenience, a Makefile is also provided, but you are not required to use it. The Makefile will build two binaries, **simple.exe** and **simple-debug.exe**. The latter will output the symbol table, instruction table and a number of debug prints. Grading will be performed with **simple.exe** . If you need to add debug printing information, always enclose it between "#ifdef _SMP_DEBUG_" and "#endif".

To test a single input file, run: ./simple.exe < inputfile.smp

You can also test **all** the test cases of a single directory with the script **run-all.sh**. It expects the directory name to test.

Several online resources can be found in the web, for instance:

- `https://www.gnu.org/software/bison/manual`

- `https://www.lysator.liu.se/c/ANSI-C-grammar-y.html#multiplicative-expression`

More resources can be found by searching for the key terms: yacc/bison parser generator.

Do not change any other file. Do not print anything to the output outside of the conditional compilation directives.

Every student should **upload the two modified files in a directory named: ABCDEFGHI**, where ABCDEFGHI is the 9-digit code identifying the student (not the 4+4). You should only upload the files grammar.y and icode.cc.