CS 3323 - Principles of Programming Languages
Assignment 5

This assignment is optional, worth 3 points, and will be directly added to your midterm/final exams. Neither is it is not required to work in groups.

The objective of this programming assignment is to add two types of predicated assignment instructions, a special case of control-flow, to a simple compiler being developed in this course. Instructions are provided below.

First, download the files grammar.y, scanner.yy, icode.cc, icode.hh, symtab.cc, symtab.hh, inputs-outputs.tar.gz, Makefile, run-all.sh and driver.cc from the assignment directory space. The code base provided is almost identical to that of the fourth programming assignment, with minor modifications to grammar.y, scanner.yy and the icode files. However, **this code base is not compatible with that of assignment 4**.

In the same spirit of the fourth assignment, perform the necessary modifications to grammar.y to add support for two new programming constructs: a basic predicated assignment and predicated assignment with short-circuit. You should only work on the grammar file (grammar.y).

The assignment is due on April 29th, 2020, 11:59pm. **No late assignments will be accepted**.

Perform the following modifications to the compiler:

1. (1.5pt) The basic predicated assignment (Lines 230–236) can be implemented with a single semantic action. It will consist of a single jump forward with an address to be defined later, the assignment to be predicated, and the definition of the target address. Note that, unlike the tasks of the fourth assignment, the target of the jump will be determined within the same semantic action. Placeholders are provided in the semantic action. You can use the same macros provided in the previous assignment (INSTRUCTION_LAST and INSTRUCTION_NEXT) to complete this task. You have to decide which to use. See the test cases (pred-assign-false.smp, pred-assign-true.smp and pred-assign-increment-true.smp) and their corresponding outputs to further understand the semantics of the new construct.

2. (1.5pt) The predicated assignment with short-circuit (Lines 238–249) can be implemented with two semantic actions. The spirit of this construct is very similar to an if-then (without the else branch). However, in addition to skipping the execution of the assignment itself, it will also skip the execution of the expression to be assigned. Placeholders are provided in both semantic actions. It will consist of a jump forward with an address to be defined later, the assignment to be predicated, and the definition of the target address (See instructions of the fourth assignment to recall how to store, pass and retrieve values stored in the parser stack). You can use the same macros provided (INSTRUCTION_LAST and INSTRUCTION_NEXT) to complete this task. See the test cases (pred-assign-sc-false.smp and pred-assign-sc-true.smp) and their output files to further understand the behavior of this construct.

A number of test cases are provided (see inputs-outputs.tar.gz). Also note that the current version of this compiler does not support <=, >=, == nor ! = (as in C syntax), nor is compatible with that of the fourth assignment.

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

Grading will be performed based on the output of your compiler. Your output should match exactly the one in the .out files. Each .smp file has a corresponding output file. See the contents of the inputs-outputs directory (after decompressing the tar.gz file).

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/bison.html`

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.

Each student should upload their modified grammar.y renamed to ABCDEFGHI.y, where ABCDEFGHI is the 9-digit code identifying the student (not the 4+4).