CS4100/5100 COMPILER DESIGN PROJECT CODE GENERATION
FALL 2020

CAUTION: Failure to read this specification completely and carefully will adversely affect your grade in CS4100/5100!

The fourth step of the compiler project is to modify the recursive descent parser (Syntax Analyzer) so that it will generate intermediate language ‘quad codes’ which can be executed using the interpreter function built during Phase 1, the Foundations part of the project. The specifications are as follows:

1) **Syntax Analyzer:** The code generation phase must be built by modifying the recursive descent parser from phase 3. If the instructions for the lexical analyzer and syntax analyzer were followed carefully and completely, the code generation task will be very straightforward. If your syntax analyzer does not work as specified, it needs to be fixed before attempting to add the code generation to it. Attempting to take shortcuts by neglecting to repair any syntax problems first will only add to the difficulties.

2) **Interpreter:** Your program must incorporate the ‘Interpreter’ function already built in Phase 1. The minor modification of implementing the both INTEGER and FLOAT math operations, selecting the correct operation based on the data types of the operands obtained from the Symbol Table. The interpreter function, of course, must have access to both your symbol table structure and your quad table structure. The interpreter must have an echo ‘trace mode’ printout, as before, which simply indicates the current program counter and quad data that will be executed next, along the lines of:

```
    PC = 5, Executing: Add, 2, 5, 11
    PC = 6, Executing: Branch, 0, 0, 8
    PC = 8, Executing: Sub, 3, 5, 12
```

and so on. This will be used for debugging the generated code.

3) **Symbol Table:** Code generation will use the same Symbol Table from the first assignment for storing the label identifiers, variable identifiers, and numeric constants detected by GetNextToken. Each table entry requires the name (lexeme), kind indicator (label, variable, or constant), type field (string, integer, or float), and a matching *value* field to store the current value (recall that labels will store a quad address, variables and constants will store their typed numeric or string value). A flag must be set after the declarations section of the program, indicating to the Symbol Table ADT that no further declarations are allowed, so that attempts to add new identifiers to the Symbol Table will generate a non-fatal ‘undeclared identifier’ compiler error. The proper handling of an undeclared identifier will be to display an ‘undeclared identifier’ error, then add it to the symbol table as if it were declared as a variable, in order to allow the parsing to continue as completely as possible.

4) **Quad Table:** As before, the Quad table should store up to 1000 rows of quads, each quad consisting of 4 integer values: an opcode and three operands. The ‘AddQuad(opcode, op1, op2, op3)’ function should be used to add to the table sequentially. An integer ‘NextQuad’ function or field, indicating the index where the **next** quad will be added, must be accessible to the parser for branch instruction building.

5) **Language Subset To Be Implemented:** All parsers should recognize the entire P20 language

as specified in the CFG. For the code generation phase, **the test programs will not use array variables**, so you will not be required to generate code to handle arrays. No CFG changes are required for already-working Syntax Analyzers.

CODE GENERATION FEATURES REQUIRED FOR ALL STUDENTS:

All CFG content for **Assignment, IF/ELSE, FOR,** and **WHILE** statements, INTEGER variable types and **expressions**;

INTEGER math only;

Numeric range checking is NOT to be implemented.

6) **Errors And Warnings To Detect:** The parser must detect all the errors specified in phase 3 of the project.

FINAL DETAILS:

The grade sheet specifies levels of completeness needed in order to receive full credit, along with additional extra-credit features. **Read carefully to avoid surprises.**