COP4020 Programming Assignment 2
CALC Interpreter/Translator
Due March 5, 2017

Purpose

This project is intended to give you experience in using a parser generator (Yacc) a parser, writing a syntax specification (grammar) for a language, performing parsing and semantic analysis (attribute grammar), and practicing error handling in a compiler.

Summary

Your task is to write an interpreter/translator for a simple calculator whose programming language, CALC, contains basic language constructs such as variables and assignment statements. The interpreter/translator will be written using a compiler generator (YACC). Your program should perform the functions of both the interpreter and translator. The program should call the lexical analyzer (Lex) for the next token, parse the token stream, report grammatical errors, perform static and dynamic semantic checks, interpret the program statement-by-statement (including print statements which produce outputs).

You will need to construct a YACC specification file calc.y which will be used in conjunction with the provided lexer.l to create your interpreter/translator. The executable can be built (and tested with p0.cal, for example) in the following way:

```
$ yacc –d calc.y
$ flex –i lexer.l
$ gcc lex.yy.c y.tab.c –lfl
$ ./a.out < proj2_tests/p0.cal
```

The included proj2.tar contains the needed lexer.l file, a sample executable, and a directory of test files. You can check your output against the sample executable in the following way:

```
$ ./proj2_linprog < proj2_tests/p0.cal
```

This will also generate a mya.cpp against which you can check your C++ translation. Your translation is valid as long as it can be compiled and executed with correct output. **Do not change the lexer.l file to suit your needs – it will be used as is to test your calc.y file**.

Syntax Specification

The syntax of the CALC language is described by a set of syntax diagrams in syntax.pdf. A syntax diagram is a directed graph with one entry and one exit. Each path through the graph defines an allowable sequence of symbols. For example, the structure of a valid CALC program is defined by the first syntax diagram. The occurrence of the name of another diagram such as declaration and compound statement indicates that any sequence of symbols defined by the other diagram may occur at that point. The following is an example valid CALC program.

```
program xyz is
begin
var a, b;
a = 2;
b = 3;
var c;
c = a + b;
print c
end
```

Your first task in this assignment is to develop a context free grammar for the CALC language from the syntax diagrams.

YACC Specification

Your second task in this assignment is to express your grammar as a YACC specification. You will want to run your specification through YACC to ensure that the grammar produces no parsing conflicts (compiled with yacc -v). If conflicts are indicated by YACC, you should alter your grammar to eliminate them without changing the language accepted by your grammar, or ensure that YACCs handling of the conflict agrees with the CALC language specification. It is suggested that you first develop a parser that merely prints out ACCEPT for syntactically correct CALC programs and REJECT with error messages for incorrectly structured CALC programs, calling the lexical analyzer for the tokens.

After you have guaranteed that your YACC specification is syntactically sound, you will extend your YACC grammar by adding attributes and semantic rules with actions to develop an interpreter/translator for CALC that does various static and dynamic semantics checks, performs the calculation specified in the CALC program, and translates the CALC program into a C++ program. Besides reporting grammatical errors, the interpreter/translator also performs the following semantics checks:

- Duplicate declaration: when a variable is declared multiple times.
- Undeclared variable: when a variable is used in the program, but not declared or before it is declared.
- Uninitialized variable: when a variable is referenced before initialized.
- Divided by 0 error: when the denominator in a division expression is 0.

The program can exit after reporting one semantic/syntax error. If there is no error, the program should (1) execute each statement and output the result of the expression in each print statement (interpreter function), and (2) produce a semantically equivalent C++ program called mya.cpp with equivalent expressions/assignments (translator function). Test out the sample executable to see what is expected.

To facilitate semantic checks and program interpretation and translation, a symbol table must be created to store variables and the related information. The symbol table will need to have at least three fields: the name of the variable, the value of the variable (integer type only in CALC), and a tag indicating whether the variable has been initialized. A good idea would be to create an array of structs, where each struct corresponds to an entry in the symbol table. You may limit the size of this array to something reasonable. When the parser encounters an identifier in the declaration, the identifier must be inserted into the symbol table. When the parser encounters an identifier in an expression, it must look up the symbol table to check if the variable has been initialized and obtain its value (if initialized). After the parser processes an assignment statement, the value of the variable in the left hand side of the assignment statement must be updated in the symbol table.

The generated mya.cpp file should have the same number of statements as the source program. Each statement in mya.cpp should have the same expression (may differ only in notation) as the original program. For the example CALC program described, the corresponding C++ program would be similar to the following:

```
#include <iostream>
using namespace std;
int main() {
  int a, b;
  a = 2;
  b = 3;
  int c;
  c = a + b;
  cout << c << "\n";
  return 0;
}
```

Interpreting and translating CALC with YACC is relatively simple. Basically, for every assignment statement, the program first evaluates the value of the expression in the right

hand side of the statement. This is similar to the trans example we gave in class. After that, the interpreter updates the value of the variable in the left hand side of the statement. When processing a print statement, the interpreter evaluates the value of the expression and prints the result to the standard output. To facilitate translation, each expression can have a string attribute that stores the string representation of the C++ expression.

Your parser should print appropriate error messages. You do not have to implement any error recovery. Your program should have similar behavior as the sample executable provided.

Assignment Submission

Submissions are due on March 5, 11:59 PM, before which you should submit your calc.y file to Blackboard.

- Recognize correct programs and detect incorrect programs (60).
- Semantic checks and error reporting (20).
- Correct calculation (10).
- Correct translation (10).
- Extra 5 points for each person who first reports an error in the sample executable.

Your program will be tested with a series of CALC programs. Some of the testing programs are provided in the project package. You are encouraged to modify the programs to further test your scanner. If your calc.y file cannot be used to create a y.tab.c file, 0 points will be given.