Assignment #4

Date Due: April 4, 2018

Total: 100 marks

Instructions

INSTRUCTIONS

- Combine all your files (all program code and any relevant output from program testing) into a single a compressed file. Please use a filename which includes your UPEI username and is of the form: username_ast4.zip(or username_ast4.tar.gz); submit the zip(tar.gz) file via moodle.
- Some explanations may be submitted on paper (the expanded grammar, scanner construction, etc).

Requirements for building a parser

The purpose of this assignment is to create a Yacc based parser for your programming language.

The assignment splits the problem into three smaller tasks which are, then, integrated to yield a parser for your language. This division allows for easier debugging of the grammar.

For each part, build a parser using Yacc. Eliminate all conflicts from the grammar. Use lex to build a lexical analyzer.

You will need four versions of the lexical analyzer to properly test each part.

The action routines in the parser should write to standard output ***the production being used to make a reduction***. That is, your task is to produce a derivation for any valid program. You can use the example in Figure 3.23 on page 143 and Examples E20, E21, E22 (beside lex part in E1-E19).

1. (15 marks) Create a grammar for a sequence of expressions; use a delimiter such as a comma or semicolon to separate them. The expressions match exactly those from your programming language. Expressions should permit the use of arithmetic, logical and relational operators. Assignments could appear in expressions as in C or in statements as in Pascal. Do not attempt to do type checking in this grammar.

   Sample input could be given as:

   ```
   510.30; (104 - x)*y+x; myIDnumber >= 10000 ; x / 2 <> (101* (ab + cd));
   ```

2. (15 marks) Create a separate grammar for a sequence of statements. In this grammar, expression should be treated as a terminal or token.

3. (15 marks) Create a grammar for the remainder of your language. The grammar rules for declarations should be part of this grammar. "Statement" should be treated as a token. This grammar represents the entire contents of a compilation file. For example, in C, it would be a sequence of external declarations and definitions of variables and functions. The parser should include syntax error checking and elementary error recovery.

4. (30 marks) Combine the grammars from 1, 2, and 3. This gives a grammar for a complete program file in your language.

   Provide a (simple) type checking system for your attribute grammar. That is, add appropriate semantic actions to your parser to ensure static type checking

5. (20 marks) Intermediate Representation

   There are a couple of choices.

   (a) Build an abstract syntax tree which can be traversed to provide a translation to machine code. If you do this, then just build the syntax tree. As output, print the syntax tree, level by level using a breadth-free traversal.

   (b) Translate the program to three-address code. A partial translation is acceptable. Outline the type of language constructs which are translated.

6. (30 marks) Design a number of test cases to provide reasonable confidence that your compilers are correct. Run your test cases and produce a file for the output of each test (record the execution in a file).

   Demonstrate the parsing of valid programs for the grammars above, and as well, show how the parser responds to syntax errors.

   (a) Testing and Verification of Program Correctness (25 marks)

   (b) Provide an overall summary of your test runs and the confidence that you have that it is working correctly.(15 marks)