

SG400 Compiler & Interpreter Design

KGA #1: Interpreter Project

This assignment supports the following course outcomes:

Design a theoretical compiler or interpreter on paper prior to any code implementation.

Translate a regular expression into the corresponding automaton that will be the basis for the interpreter written for that expression.

Create an interpreter for a defined high-level language that will execute on a computer.

Appreciate the challenges of designing compilers and interpreters for full programming languages and scripting languages.

Assignment Overview

For this assignment, you will write a program that takes a C++ program as input and interprets that program. You will output error messages if any part of the compilation process fails. You will implement the following four parts of an interpreter; lexical analysis, parsing, semantic analysis, and code execution.

You will convert a correct program written in a language defined by the below grammar to C++ during the code execution process. This will allow your interpreter to then execute the code as a typical C++ program after your interpreter has completed the compilation process.

program	class+
class	class TYPE {feature+}
feature	void ID (formal_list*) {expr_list} TYPE ID; TYPE ID = expr;
formal_list	formal formal_list, formal
formal	TYPE ID
expr_list	expr; expr; expr_list
expr_method_list	expr expr_method_list, expr
expr	ID ID.ID(expr_method_list) ID (expr_method_list) if (expr) {expr_list} else {expr_list} while (expr) {expr_list} new TYPE() expr > expr expr >= expr expr < expr expr <= expr

```
      | expr == expr
      | expr != expr
      | ID = expr
      | STRING_CONST
      | arith
      | BOOL_CONST

arith  (arith)
      | num
      | num + num
      | num - num
      | num * num
      | num / num

num    INT_CONST
      | ID
```

If an error occurs in one of the parts of the interpreter, you will recover gracefully and continue the interpretation process if possible. You will cycle back around until your source program executes error free. More specifically, if the lexical analysis phase of the interpreter finds errors in the input program, you will continue to run the input program through the parser, semantic analyzer, or code generator of your interpreter.

Deliverables

1. Write pseudo code covering the below elements:
 - From the above grammar, generate the corresponding regular expressions.
 - Convert the regular expressions into the corresponding Finite State Automata or State Diagram.
2. Generate code to complete the below tasks:
 - Create the lexical analyzer for the above grammar.
 - Create the parser from the output of the lexical analyzer.
 - Create the semantic analyzer from the output of the parser.
 - Create the code executor from the output of the semantic analyzer.
3. Reflection paper discussing the specific challenges and skill learned in completing this assignment.

Grading

The following grading rubrics will be used to evaluate your assignment. Your instructor will provide you with these rubrics along with this assignment handout.

Content Grading Rubric KGA #1 - 100% of assignment grade

Resources

Mak, Ronald. Writing Compilers and Interpreters – An Applied Approach Using C++ - Second Edition. John Wiley & Sons, Inc., 1996.

Lex and Flex lexical analyzer compiler compilers

(<http://dinosaur.compilertools.net/>)

Bison and YACC parser compiler compilers (<http://dinosaur.compilertools.net/>)