CSCI 4355 Programming Language Concepts
Fall 2023
Term Project
Due Date: 11/19/2023

In this project you are to implement a scanner and a recursive descent parser for a small programming language (let's call it Hawk). Your program will take as input a program written in the provided language. In addition to parsing the input programs, your program must generate the proper errors when encountered.

The following is the grammar that you will use:

```
Rule 01: PROGRAM → program DECL_SEC begin STMT_SEC end; |
                    program begin STMT_SEC end;
Rule 02: DECL_SEC → DECL | DECL DECL_SEC
Rule 03: DECL → ID_LIST : TYPE ;
Rule 04: ID_LIST → ID | ID , ID_LIST
Rule 05: ID → (_ | a | b | … | z | A | … | Z) (_ | a | b | … | z | A |
… | Z | 0 | 1 | … | 9)*
Rule 06: STMT_SEC → STMT | STMT STMT_SEC
Rule 07: STMT →  ASSIGN | IFSTMT | WHILESTMT | INPUT | OUTPUT
Rule 08: ASSIGN →  ID := EXPR ;
Rule 09: IFSTMT →  if COMP then STMT_SEC end if ; |
                   if COMP then STMT_SEC else STMT_SEC end if ;
Rule 10: WHILESTMT →  while COMP loop STMT_SEC end loop ;
Rule 11: INPUT →  input ID_LIST;
Rule 12: OUTPUT → output ID_LIST | output NUM;
Rule 13: EXPR → FACTOR | FACTOR + EXPR | FACTOR – EXPR
Rule 14: FACTOR → OPERAND | OPERAND * FACTOR | OPERAND / FACTOR
Rule 15: OPERAND → NUM | ID | ( EXPR )
Rule 16: NUM → (0 | 1 | ... | 9)+ [.(0 | 1 | … | 9)+]
Rule 17: COMP → ( OPERAND = OPERAND ) | ( OPERAND <> OPERAND ) |
         ( OPERAND > OPERAND ) | ( OPERAND < OPERAND )
Rule 18: TYPE → int | float | double
```

This grammar has 18 rules. It also has reserved words in it indicating that they cannot be used for identifiers. Non-terminal symbols are those that are capitalized, and terminal symbols are those that are lowercase. Many rules have alternative choices, for example Rule 09 has two alternatives, one without an else, and one with and else.

The following are the lexemes for the language:

- Reserved words: program, begin, end, if, then, else, input, output, int, while, loop.
- Operators: assignment (:=), less than (<), greater than (>), equals (=), not equals (<>), plus (+), minus (-) , multiply (*), divide (/) and parentheses.

- The ‘;’ is also used to terminate statements and the ‘,’ and the ‘:’ are used when declaring variables.
- Identifiers: start with a letter or an ‘_’ followed by any number of letters, digits or underscores.
- Numbers: Either integer numbers (max 10 digits), or floating point numbers (max 10 digits).

Upon encountering a variable in a declaration section, you must add it to the symbol table. This means that a redeclaration of a variable should produce an error and exit the program. The use of a variable before it is declared should also produce an error indicating an undeclared identifier and exit the program.

Other errors that could be generated upon scanning and parsing a program:

- Illegal symbol
- Illegal identifier
- Illegal number
- Parse errors encountered by expecting a symbol and not getting that symbol.

All errors must indicate the line number in the file. All error message must be descriptive as well indicating what happened precisely. Upon encountering an error, you must exit the program immediately after reporting the error and its location.

You will be provided with some test cases to use to test your program. However, your implementation should work with other test cases. So, you are encouraged to generate your own test cases to make them work with your parser.

In the assignment, you will be turning it in incremental. The first part should be only a scanner to print out the tokens and lexemes. The second part should be just a parser without the tree. The third should be a parser with the tree.

Programs that pass the parser and the scanner should output the left-hand side of the rules that it followed. No need to worry about producing rules 5, 16, and 18 as they are more about the scanner than the parser. Make sure no spaces before or after the output are produced. You must produce the output as describe, see the below examples. Programs that have errors should produce the same output until they reach an error and then produce a statement indicating the error type and exit.

Given the following program:

```
program  x, y: int;

begin
    input x,y;
    y:=x+y;
     output y;
end;
```

The output should be:

```
PROGRAM
DECL_SEC
DECL
ID_LIST
ID_LIST
STMT_SEC
STMT
INPUT
ID_LIST
ID_LIST
STMT_SEC
STMT
ASSIGN
EXPR
FACTOR
OPERAND
EXPR
FACTOR
OPERAND
STMT_SEC
STMT
OUTPUT
ID_LIST
```

Given the following program:

```
program
  x, y: double;
  e: int;
begin
  input x;
  y := 1;
  if (x > 0) then
   e := 0;
   while (x > 0)
   loop
     y := y * x;
     x := x –1;
   end loop;
  else
    e := 1;
  end if;
  output e, x, y;
end;
```

The output should be:

```
DECL_SEC
DECL
ID_LIST
ID_LIST
DECL_SEC
DECL
ID_LIST
STMT_SEC
STMT
INPUT
ID_LIST
STMT_SEC
STMT
ASSIGN
EXPR
FACTOR
OPERAND
STMT_SEC
STMT
IF_STMT
COMP
OPERAND
OPERAND
STMT_SEC
STMT
ASSIGN
```

EXPR
FACTOR
OPERAND
STMT_SEC
STMT
WHILE_STMT
COMP
OPERAND
OPERAND
STMT_SEC
STMT
ASSIGN
EXPR
FACTOR
OPERAND
FACTOR
OPERAND
STMT_SEC
STMT
ASSIGN
EXPR
FACTOR
OPERAND
EXPR
FACTOR
OPERAND
STMT_SEC
STMT
ASSIGN
EXPR
FACTOR
OPERAND
STMT_SEC
STMT
OUTPUT
ID_LIST
ID_LIST
ID_LIST