MYY802: CiScal Compiler

Due on Monday, March 13, 2017

George Manis

G.Zachos, A.Konstantinidis

March 5, 2017

Contents

About **3**
- CiScal language . . . 3
- CiScal Compiler . . . 3
- Using the compiler . . . 3

Compilation Phases **3**

Lexical Analyzer **4**
- Implementation details . . . 5
  - Custom Classes . . . 5
  - Data Structures . . . 5
  - Return value . . . 6

Syntax Analyzer **6**

About

CiScal language

CiScal is a minimal programming language that has borrowed its characteristics from C and Pascal.

CiScal Compiler

CiScal Compiler (CSC) was developed during the MYY802 - Compilers course at the Department of Computer Science and Engineering, University of Ioannina.

Using the compiler

To learn how to use CSC run `./csc.py` and the information below will be printed to console:

```
Usage: ./csc.py [OPTIONS] {-i|--input} INFILE
Available options:
        -h, --help                Display this information
        -v, --version             Output version information
        -I, --interm              Keep intermediate code (IC) file
        -C, --c-equiv             Keep IC equivalent in C lang file
        --save-temps              Equivalent to -IC option
        -o, --output OUTFILE      Place output in file: OUTFILE
```

Compilation Phases

...

Lexical Analyzer

The Finite-State Machine (FSM) diagram in Figure 1 is a partial graphical representation of the finite automata implemented in the `lex()` function and which is used to convert the input sequence of characters into a sequence of language tokens. In addition to the states shown below, there are fourteen (14) more accepting states that correspond to characters: '+', '−', '⋆', '/', '=', ',', ';', '{', '}', '(', ')', '[', ']' and 'eof'. Transition to these states is triggered only from initial state s_0.

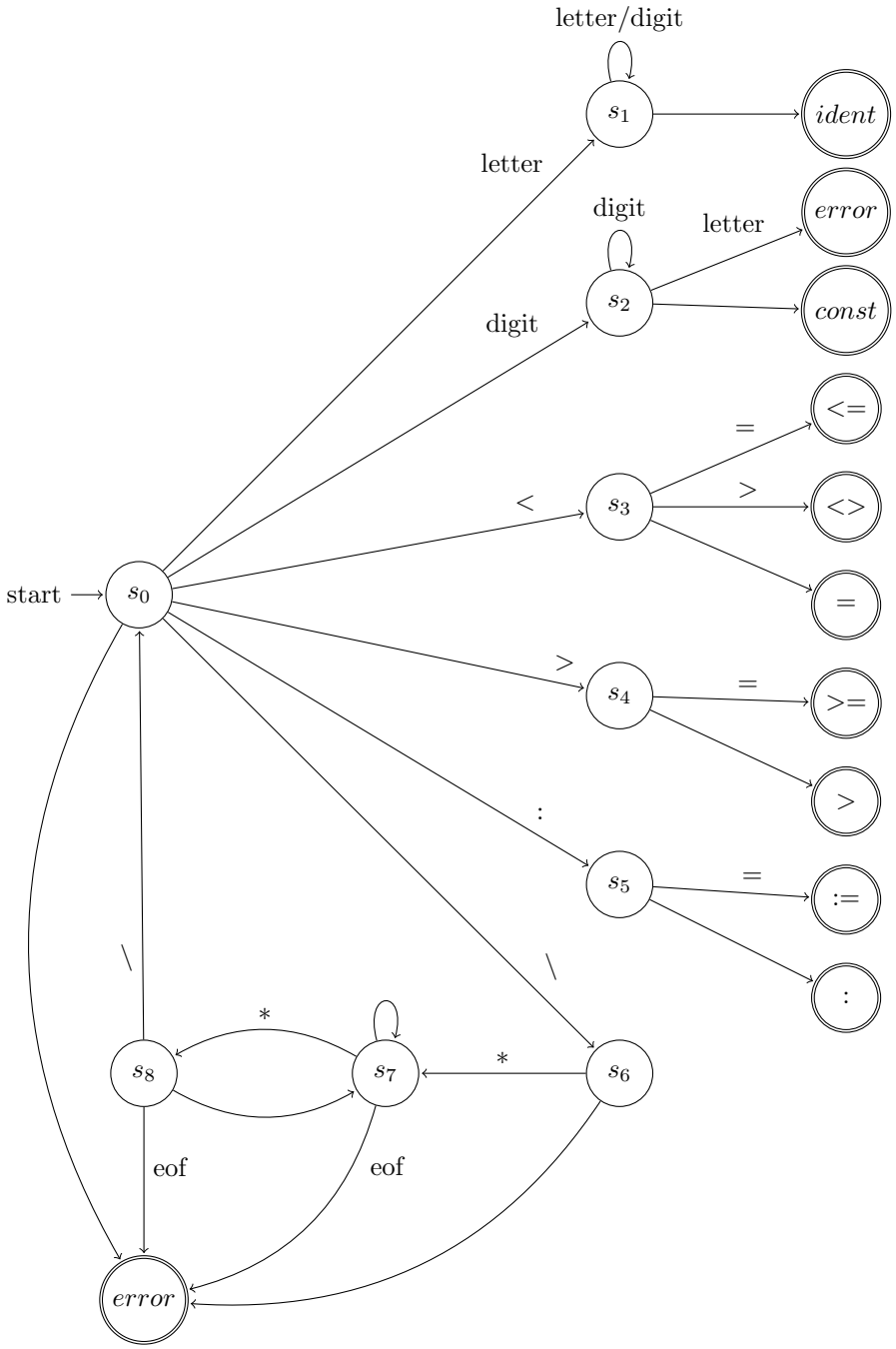

Figure 1: Partial FSM diagram of lexical analyzer's finite automata

Implementation details

Custom Classes

The `Token` class was defined to group all useful information related to a token and that should be available to the syntax analyzer. This information includes:

- `tktype`: token type (attribute of the `TokeType` enumeration)
- `tkval`: the actual token value
- `tkl`: the line of the input file that the token was found
- `tkc`: the offset of the token's first character from the start of line `tkl`

Listing 1: Token Class

```
class Token():
    def __init__(self, tktype, tkval, tkl, tkc):
        self.tktype, self.tkval, self.tkl, self.tkc = tktype, tkval, tkl, tkc

    def __str__(self):
        return  '(' + str(self.tktype)+ ', \'' + str(self.tkval) \
                + '\', ' + str(self.tkl) + ', ' + str(self.tkc) + ')'
```

Data Structures

The `token` dictionary maps actual keyword values to the corresponding `TokenType` attributes and serves code simplicity.

Listing 2: Token type/value dictionary

```
tokens   = {
    '(':           TokenType.LPAREN,
    ')':           TokenType.RPAREN,
    '{':           TokenType.LBRACE,
    '}':           TokenType.RBRACE,
    '[':           TokenType.LBRACKET,
    ']':           TokenType.RBRACKET,
    ',':           TokenType.COMMA,
    ':':           TokenType.COLON,
    ';':           TokenType.SEMICOLON,
    '<':           TokenType.LSS,
    '>':           TokenType.GTR,
    '<=':          TokenType.LEQ,
    '>=':          TokenType.GEQ,
    '=':           TokenType.EQL,
    '<>':          TokenType.NEQ,
    ':=':          TokenType.BECOMES,
    '+':           TokenType.PLUS,
    '-':           TokenType.MINUS,
    '*':           TokenType.TIMES,
    '/':           TokenType.SLASH,
    'and':         TokenType.ANDSYM,
    'not':         TokenType.NOTSYM,
    'or':          TokenType.ORSYM,
```

```
24      'declare':    TokenType.DECLARESYM,
25      'enddeclare': TokenType.ENDDECLSYM,
26      'do':         TokenType.DOSYM,
27      'if':         TokenType.IFSYM,
28      'else':       TokenType.ELSESYM,
29      'exit':       TokenType.EXITSYM,
30      'procedure':  TokenType.PROCSYM,
31      'function':   TokenType.FUNCSYM,
32      'print':      TokenType.PRINTSYM,
33      'call':       TokenType.CALLSYM,
34      'in':         TokenType.INSYM,
35      'inout':      TokenType.INOUTSYM,
36      'select':     TokenType.SELECTSYM,
37      'program':    TokenType.PROGRAMSYM,
38      'return':     TokenType.RETURNSYM,
39      'while':      TokenType.WHILESYM,
40      'default':    TokenType.DEFAULTSYM,
41      'EOF':        TokenType.EOF}
```

Return value

The `lex()` function returns an object of type `Token` to the syntax analyzer.

Syntax Analyzer

...