Assignment 5

CSCI 3136: Principles of Programming Languages

Due Mar 19, 2018

Banner ID: _____ Name: _____

Banner ID: _____ Name: _____

Banner ID: _____ Name: _____

In this assignment, you will continue your work towards an interpreter for the TOY language. The description of TOY can be found at http//www.cs.dal.ca/~nzeh/Teaching/3136/Assignments/toylang.html. In the last assignment, you implemented a scanner that converts the input text into a stream of tokens. In this assignment, you implement an LL(1) parser that checks whether the token stream output by the scanner forms a syntactically correct TOY program as specified by the grammar given in the description of the TOY language. If it is, the parser should print the parse tree of the program. If it is not, the parser should report failure.

1 Programming Languages You May Use

To keep things manageable for the TAs, you must choose one of three languages to implement your scanner: Python, Java or C/C++. Since the semantic analyzer in the next programming assignment builds on the parser, I will make a sample implementation available so you can complete the next assignment even if you fail the current one. Note, however, that I will only make a Python implementation available. Thus, if you choose a language other than Python for this assignment and fail to complete it, you may be forced to switch to Python in the next assignment.

2 Requirements

Implement your parser in a file `parser.py`, `Parser.java` or `parser.(c|cpp)`. You may split your code into multiple files, but the main file must be `parser.(py|java|c|cpp)`. The program should expect one command line argument, the name of a TOY program file. If the program is syntactically correct, the parser should print the parse tree as illustrated below. If it is not, the parser should report which token it does not know how to deal with, including its position in the input text. (If you use my sample implementation of a scanner, then the position of the current token is available through `scanner.pos` if `scanner` is the name of your scanner object.) As an illustration of the output for a correct input program, the program

```
fun fibonacci // n -- F_n

  // TOY supports local functions
  fun fib // n-i F_{i-1} F_i -- F_n
    rotl dup 0 =
    [ drop swap drop ]
    [ 1 - rotr dup rotl + fib ] ??
  .

  0 1 fib
.
```

should result in the following output:

```
Program
  TopDef
    FunDef
      KwFun (fun)
      Identifier (fibonacci)
      Body
        Statement
          Def
            TopDef
              FunDef
                KwFun (fun)
                Identifier (fib)
                Body
                  Statement
                    SimpleStatement
                      Identifier (rotl)
```

```
Body
  Statement
    SimpleStatement
      Identifier (dup)
  Body
    Statement
      SimpleStatement
        Int (0)
    Body
      Statement
        SimpleStatement
          Identifier (=)
      Body
        Statement
          Lambda
            FunLambda
              KwLambdaBegin ([)
              Body
                Statement
                  SimpleStatement
                    Identifier (drop)
                Body
                  Statement
                    SimpleStatement
                      Identifier (swap)
                  Body
                    Statement
                      SimpleStatement
                        Identifier (drop)
                    Body
              KwLambdaEnd (])
        Body
          Statement
            Lambda
              FunLambda
                KwLambdaBegin ([)
                Body
                  Statement
                    SimpleStatement
                      Int (1)
                  Body
                    Statement
                      SimpleStatement
                        Identifier (-)
                    Body
                      Statement
                        SimpleStatement
                          Identifier (rotr)
                      Body
                        Statement
                          SimpleStatement
                            Identifier (dup)
                        Body
                          Statement
                            SimpleStatement
                              Identifier (rotl)
                          Body
```

```
                                                Statement
                                                  SimpleStatement
                                                    Identifier (+)
                                                Body
                                                  Statement
                                                    SimpleStatement
                                                      Identifier (fib)
                                                  Body
                                      KwLambdaEnd (])
                                Body
                                  Statement
                                    SimpleStatement
                                      Identifier (??)
                                  Body
                        KwEnd (.)
                Body
                  Statement
                    SimpleStatement
                      Int (0)
                  Body
                    Statement
                      SimpleStatement
                        Int (1)
                    Body
                      Statement
                        SimpleStatement
                          Identifier (fib)
                      Body
            KwEnd (.)
        Program
```

Yes, this is a lot of output for such a short program, but such is the nature of parse trees. This is why we will introduce abstract syntax trees in class soon, which provide a much more compact representation of the program structure but are not acceptable as the program output for this assignment.

3 Submission Instructions

Put your scanner implementation into a single zip-file containing all code required to run it, including the scanner used to translate the input text into a token stream and any other supplementary files that may be needed. Email this file to Yaser Alkayale (yaser.alkayale@dal.ca) before midnight, March 19, 2018.