CS202M_PROGRAMMING METHODOLOGY

INTERPRETER

TEAM-7

Language : SCALA

Introduction

This is team-7 consisting of

- CS21B036 - VIKRAM.N
- CS21B020 - MANOJ.G
- CS21B032 - SIVA SAI.M
- CS21B030 - VIDHYA BHUSHAN.M
- CS21B058 - TEJESWARA REDDY.S
- CS21B046 - SHABAB REHMANI

We had chosen scala language to write interpreter for 2 basic constructs of it.

SCALA:

Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It seamlessly integrates features of object-oriented and functional languages.

CONSTRUCTS IMPLEMENTED :

- **If else(nested / single) :**

  A small code snippet of if-else statement in scala is given below

```
object Demo {
   def main(args: Array[String]) {
      var x = 30;

      if( x < 20 ){
         println("This is if statement");
      } else {
         println("This is else statement");
      }
   }
}
```

- **Do-while :**

  A small code snippet of do-while statement in scala is given below

```
object Demo {
   def main(args: Array[String]) {
      // Local variable declaration:
      var a = 10;

      // do loop execution
      do {
         println( "Value of a: " + a );
         a = a + 1;
      }
      while( a < 20 )
   }
}
```

Backus Naur Form (bnf) followed:

Program : : = stmtlist

stmtlist : : = stmt[stmtlist]

| {Declare_stmt}

Declare_stmt : : = `var' variable = expr

Stmt : : = variable = expr

| stmt1

| empty

stmt1 : : = if(conditional_stmt) {nl} stmt [; | \n] else stmt]

Assign_stmt : : = var=expr

Conditional_stmt : : = expr (comparison_operator) expr

| bool

Comparison_operator : : = <

| >

| ==

| <=

| >=

expr : : = term ((+|-) term)

term : : = factor ((*|/|%) factor)

Factor : : = +factor

| -factor

| Int

| (expr)

| variable

LEXER :

A lexer performs lexical analysis and breaks down an input stream of words into tokens and returns them to the parser.

We are having the lexer class, which is having an **important methods** like

- **self.advance( ) -** It Advance the `pos' pointer and set the `current_char' variable.
- **get_next_token( ) -** This method is responsible for breaking a sentence apart into tokens, One token at a time.
- **_id( ) -** It handles identifiers and reserved keywords.
- **skip_whitespace( ) -** It skips whitespaces given in the input.
- skip_comment()

The tokens that our lexer can return can be seen below

```
INTEGER = 'INTEGER'
PLUS = 'PLUS'
MINUS = 'MINUS'
MUL = 'MUL'
DIV = 'DIV'
REM = 'REM'
LPAREN = 'LPAREN'
RPAREN = 'RPAREN'
LCURL = 'LCURL'
RCURL = 'RCURL'
ID = 'ID'
ASSIGN = 'ASSIGN'
SEMI = 'SEMI'
DOT = 'DOT'
OBJECT = 'OBJECT'
VAR = 'VAR'
COLON = 'COLON'
COMMA = 'COMMA'
EOF = 'EOF'
DEF = 'def'
IF = 'IF'
ELSE = 'ELSE'
TRUE = 'TRUE'
FALSE = 'FALSE'
OBJECT = 'OBJECT'
DO = 'DO'
WHILE = 'WHILE'
```

PARSER :

Input : tokens that are generated from the lexer.

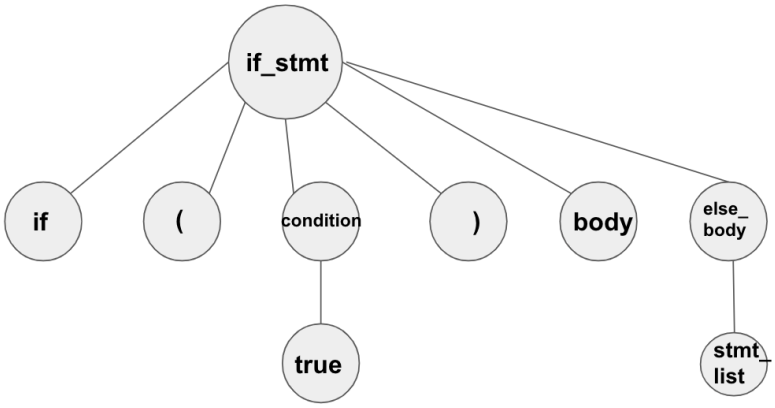
It verifies that the string can be the correct grammar of the scala language and reports any syntax errors and produces a abstract syntax tree.

An abstract syntax tree is the tree representation of the source code of a program that conveys structure of the source code, and Each node in the tree represents a construct occurring in the source code

For example if a given input doesn't follow the bnf then an error is thrown.

EXAMPLE OF ABSTRACT SYNTAX TREE FOR IF-ELSE :

- **If-else :**



Similarly we can have an abstract syntax tree for do-while.

A DEMO OF WORKING OF OUR CODE

- **A sample input testcase :**

  object demo{ var x: int = 2; if(x<3) {x=x+1;} else{ x=x-1;}; do{x=x+1;} while(x<10)}

```
enter your input hereobject demo{ var x: int = 2; if(x<3) {x=x+1;} else{
    x=x-1;}; do{x=x+1;} while(x<10)}
parse
demo
id
demo
Token(LCURL, '{')
body
Token(LCURL, '{')
stmtlist
Token(VAR, 'VAR')
stmt
decleration stmt
var eaten
Token(ID, 'x')
Token(COLON, ':')
done
done
Token(INT, 'INT')
Token(ASSIGN, '=')
expr
term
factor
Token(INTEGER, 2)
stmt
stmt1
ifstmt
```

PLEASE REFER CODE FOR MORE INFORMATION.

COMPOSITION :

- **Learning syntax of scala :- CS21B032,CS21B036,CS21B046.**
- **BNF : CS21B032,CS21B036**
- **LEXER : CS21B058,CS21B020**

- **PARSER : CS21B030,CS21B020**
- **AST & REMAINING CODE: CS21B030**
- **README FILE : CS21B058**