INTERPRETER REPORT

TEAM - 7

Introduction

Interpreter is a program which helps to execute the instructions which are given in high level language without converting it to machine code.

As it is a program it has some advantages and disadvantages

- **Advantages**
    - In terms of debugging, it is easier because it reads the code line by line and this helps to find errors immediately.
    - In terms of compatibility the interpreted code is independent of platforms that means it can run on different operating system and hardware configurations.
    - Interpreters can provide more detailed error messages because it executes line by line.
    - It takes less memory because it does not create any separate files unlike compliers.

- **Disadvantage**
    - In terms of execution speed, it is slower than complier as it executes code line by line.
    - Unlike the Compliers in terms of security Interpreter is vulnerable to security risks.

Language

The language we have chosen is ***Swift.*** Swift is a high-level general purpose, multi-paradigm programming language developed by Apple Inc. It is protocol-oriented language. It is developed to replace all languages based on C like objective C, C++, C.

Required dependencies to install in the system

We need a Editior like VS_code with JDK installed then open the Swift folder after extracting from the zip file in the editor and run the Swift.java in terminal and assign the path **com\craftinginterpreters\Test_file\test.swift** at the end and execute it.

Constructs Interpreted

- Assignment Statements
- Compound Statements
- Print Statements
- Conditional Statements
- Boolean Statements
- Arithmetic operations
- Loops

Assumptions and Constraints

- For our interpreter the print statement does not need any format specifier.
- Our interpreter cannot execute print(x,y). It only executes print(x).
- We assumed a small constraint in FOR loop by incriminating the variable value in loop itself.

LEXER

Lexer basically helps us to break down the code into tokens and categorize each token to corresponding type and assign the value to it. There are no predefined datatypes for variables in Swift, so it takes the datatype of assigned value.

If we consider n = 7, as 7 is an integer so n is identified with type integer of value 7.

```java
package com.craftinginterpreters.swift;

class Token {
    final TokenType type;
    final String lexeme;
    final Object literal;
    final int line;

    Token(TokenType type, String lexeme, Object literal, int line) {
        this.type = type;
        this.lexeme = lexeme;
        this.literal = literal;
        this.line = line;
    }

    public String toString() {
        return type + " " + lexeme + " " + literal;
    }
}
```

Token type and the value.

```java
package com.craftinginterpreters.swift;

enum TokenType {
    // Single-character tokens.
    LEFT_PAREN, RIGHT_PAREN, LEFT_BRACE, RIGHT_BRACE,
    COMMA, DOT, MINUS, PLUS, SEMICOLON, SLASH, STAR,


    // One or two character tokens.
    BANG, BANG_EQUAL,
    EQUAL, EQUAL_EQUAL,
    GREATER, GREATER_EQUAL,
    LESS, LESS_EQUAL,

    // Literals.
    IDENTIFIER, STRING, NUMBER,

    // Keywords.
    AND, CLASS, ELSE, FALSE, FUN, FOR, IF, NIL, OR,
    PRINT, RETURN, SUPER, THIS, TRUE, VAR, WHILE,LET,IN,REPEAT_WHILE,

    EOF,EOL,NEXTLINE,DOTDOT
}
```

The keywords in the mentioned above is not to be used for variable names.

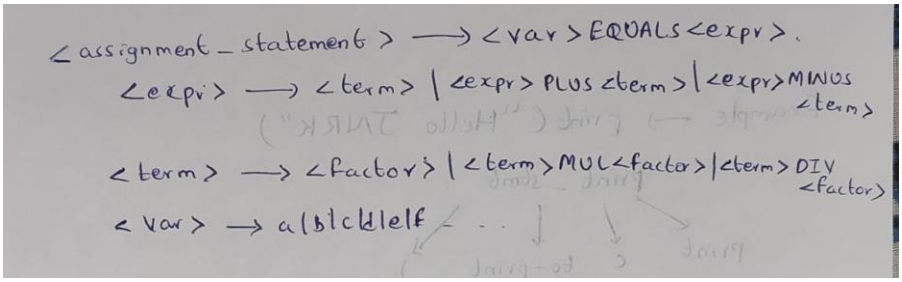
PARSER

The parser functionality is one of the most important functionalities in the interpreter. It helps us to perform analysis of the syntax. The generated tokens are transferred one by one and linked based on the order of execution. A tree is built with nodes storing execution data, and the information is processed via post-order traversal. This data is interpreted by conducting operations on the nodes, which gives the final result. Each type of data is enclosed within a class, making it easier to create nodes with appropriate features for successful implementation.

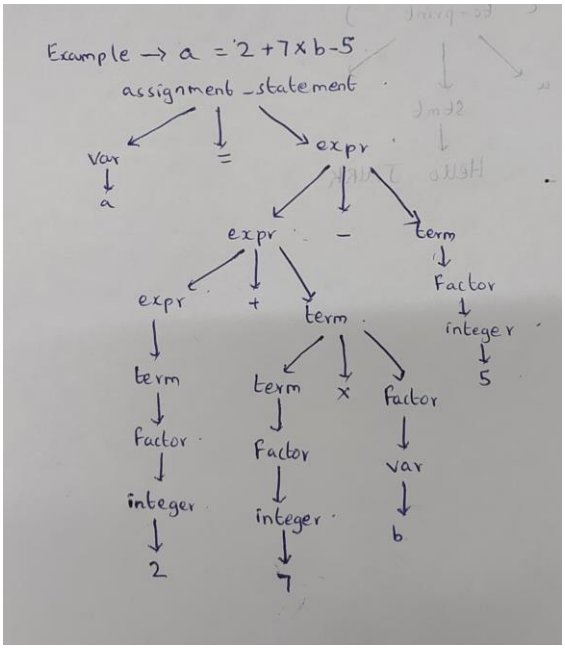
BNF AND Parse Tree

- **Assignment Statements:**

  BNF-

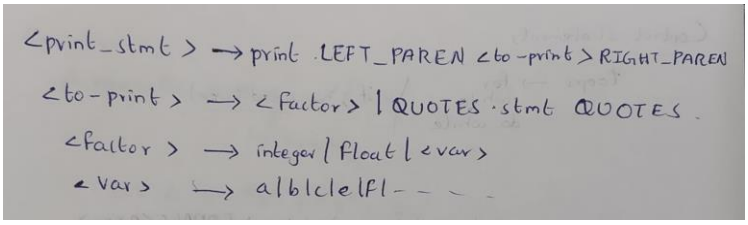


  PARSE TREE-

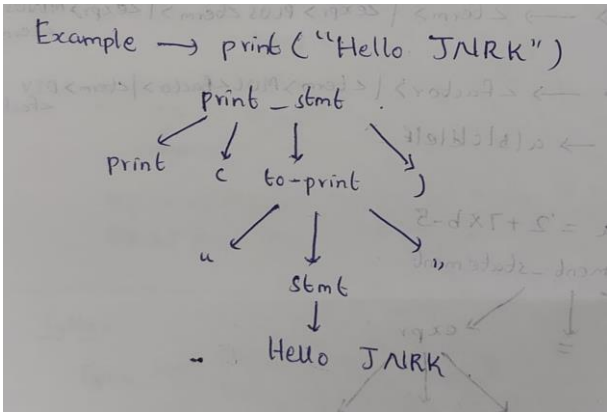


- **Print Statement-**

  BNF-



PARSE TREE-



INTERPRETER

Interpreter takes input from the parser tree generated by parser. The tree is traversed, and the result is computed.

If the syntax of the program is wrong, parse tree is not generated which throws an error.

CHALLENGES:

The challenges we are faced while writing interpreter are.

- We faced difficulty to understand how the var and let works and difference between them.
- We faced difficulty how to assign a value for a variable without declaring its datatype.
- Difficulty in writing a code without semicolon (';')
- Writing 1_2_3 as 123 took us a lot of time and performing operations using it gave us a thrilling experience of coding.

LEARNINGS:

We have learnt many things for this project few of them are.

1. We learned how the interpreter works.
2. We learned how to convert the given grammar into an equivalent code.
3. We learned some parts of new language Swift how and why it is developed and how to implement it.

Team-7

Team Members and their Contributions:

1.G JASWANTH - (CS22B020)

Parser,Interpreter,BNF

2.NISHCHITH-(CS22B021)

Parser,Interpreter,Report

3.RASHMITHA-(CS22B050)

Lexer,Interpreter,Report

4.ROHITH-(CS22B022)

BNF,Lexer,test_files

5.SAI KOWSHIK-(CS22B004)

Lexer,BNF,test_files