SER 502 - Project

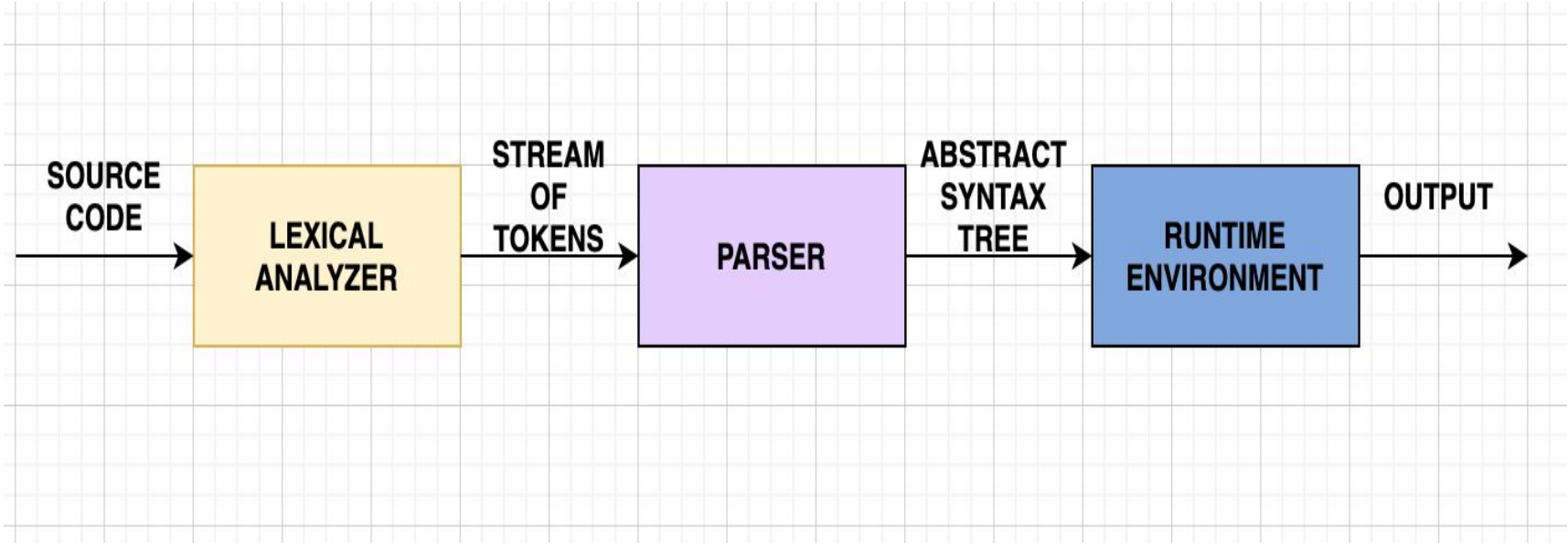
Team #10

Language - Marcel

Team members -

- Gayathri Sitaraman (1211143957)
- Prashant Singh (1215154005)
- Priyanka Zar (1218502948)
- Niket Bhave (1217472191)

Block Diagram of the Interpreter



Language Description

- The language we have developed uses a function based approach progressing towards an object oriented approach to problem solving.
- The language doesn’t deviate largely from the pre conceived and used notions of imperative programming. This is reflected in use of keywords and methods of execution.

Lexer

- The first component in the language is lexer, which is responsible for taking the user’s code as input and dividing it into a stream of tokens.
- The lexer has been implemented in Python. It makes use of normal python string operations and RegEx operations.
- The stream of tokens will then be passed to a parser for analysis and verification of tokens.
- Initially we made use of PLY library but due to increased complexity in use and higher number of dependencies we switched the implementation to regular Python.

Parser

- It checks the syntax of the input(i.e. stream of tokens) and also, gives the structural representation(i.e. abstract syntax tree) of it.
- In our parser, we have used DCG(Definite Clause Grammar) to implement the grammar of the language.
- Our parser checks the syntax of the following grammar in the language:
  - Main function
  - Commands
  - Declaration
  - Conditional statements(if then else)
  - Loops(for and while loops)
- Also, the allowed datatypes are integer(int), string(str), and, boolean(bool).
- Also, the allowed data structures are 1-Dimension and 2-Dimension lists. Functions also implemented.

Evaluator

- The evaluator is the final component of our language design.
- The evaluator will take parse tree as input and evaluate the nodes to provide the final solution to the input code.
- We have implemented the evaluator in Python. It makes use of nltk library to make it easier to process parse trees.
- Nltk converts the input tree to a nltk parse tree which provides distinct functionalities like finding the sibling, parent, or child of any node thus making traversal and subsequent evaluation easier.
- Data structures and functions are implemented in AST but not implemented in evaluator.

Grammar

Program ::= Block

Block ::= Main.

Main ::= Main { Declaration Command }

Declaration ::= Declaration ‘;’ Declaration

| Datatype Ident = Val

| const Datatype Ident = Val

| int Ident = Onedim

| int Ident = Twodim

Datatype ::= int | bool | str

Command ::= Command * ‘{‘ Command ‘}’ * | Ident = E ‘;’ | ‘if’ B ‘then’ ‘{‘ Command ‘}’ ‘else’ ‘{‘ Command ‘}’ ‘breakif’

| ‘while’ B ‘do’ ‘{‘ Command ‘}’ ‘breakwhile’ | ‘for’ ‘(‘ Ident ‘in range’ Num ‘)’ ‘{‘ Command ‘}’ ‘breakfor’

| ‘for’ ‘(‘ Ident ‘in range’ Ident ‘)’ ‘{‘ Command ‘}’ ‘breakfor’ | ‘print’ ‘(‘ S | Ident | Num | B ‘)’ ‘;’

| B ‘?’ Val ‘:’ Val | eps | Declaration

B ::= true | false | E == E | not B | E or E | E and E | E > E | E < E | E >= E | E <= E

| E != E

E ::= E + E | E - E | E * E | E / E | (E) | Ident := E | Ident | Num

S ::= “S Ident Num” | “S Num” | “Ident” | “S Ident “