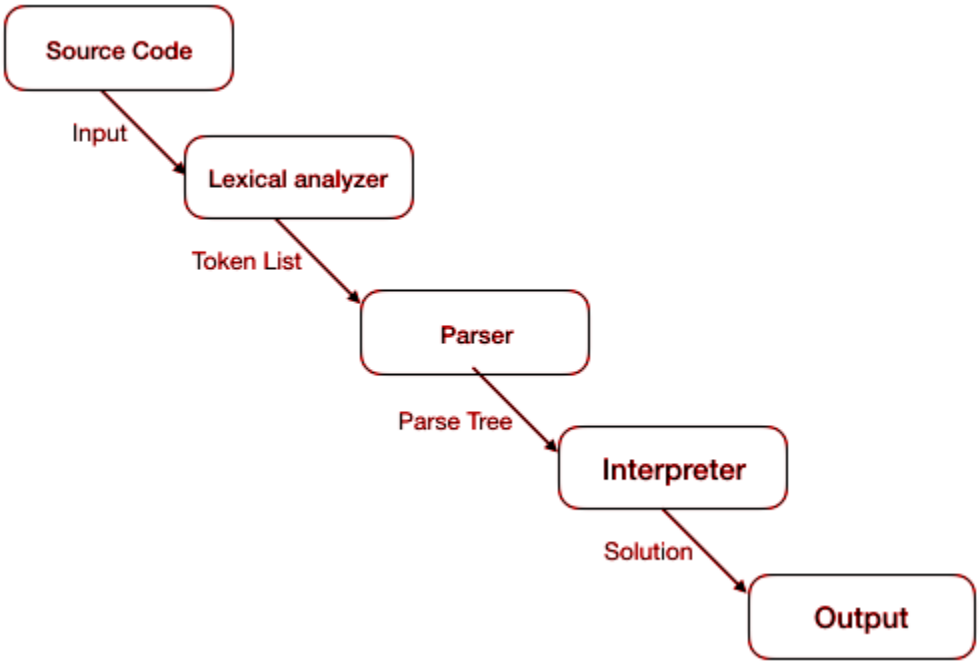
SER 502 - Project - Team 7 - Spring 2021

Authors: Gautham Krishna, Itiparna Mahala, Abhishek Mohabe, Apoorva Giliyal
Programming Language Name - **GIAA**
Programming Language Extension - **.GIAA**

The language compilation is demonstrated basis the following stages of execution:



1. **Source code:** The source code contains the program that will be executed by the GIAA programming language. The source code file will be saved in the file extension of .giaa

2. **Lexical Analyzer:** The first phase of a compiler is lexical analysis. The modified source code from language preprocessors is taken as input and broken down into a series of tokens by removing any whitespace or comments in the source code.
    a. Design language – Prolog+Python

3. **Parser:** It makes sure the source code follows the correct syntax as defined by the language GIAA. A parse tree is generated by reading the tokens one at a time from the token list. It throws syntax error if all the tokens do not get parsed.
    a. Design language – Prolog
    b. Data structure used – List
    c. Parsing technique – Top-down parsing using unification
    d. Grammar used – Definite Clause Grammar (DCG)

4. **Interpreter:** It takes the parse tree as input and using operational and denotational semantics executes the program. Top down parsing of each node is done with simultaneous evaluation. The evaluation process is tracked in a prolog list.
    a. Design language – Prolog
    b. Data structure used – List data structure

GIAA Programming Language Design

1. **Data Types**
    1. INTEGER
    2. BOOL
    3. STRING

2. **Identifiers**
   It can contain lowercase , uppercase letters , numbers and special characters like $.

3. **Variable Declaration**
   Variables can be initialized with the allowed data types. Some examples of the variable declarations are given below:
   Eg 1: int a = 4;
   Eg 2: int x;

4. **Loops**
   It will support ‘for’ ,’forrange’ and ‘while’ condition loop. The loops will run through the block of statements as long as the statement remains true. If the condition is reached as false then the control will exit the loop.

5. **Terminators**
    a. The program in GIAA has a program block which starts with “Begin” and concludes with the “End”.
    b. Every statement must end with a “ ; “.
    c. Additionally, the language allows the user to use a white space in order to separate multiple tokens in the language and increase the program readability.

6. **Comments**
    a. Comments in GIAA language can be written followed by the special symbol “@”.
    b. The language shall support only the single line of comments.

7. **Conditionals**
    a. The GIAA language supports the “if-else” conditional construct that can be used to evaluate the decision making.
    b. The condition written after “if” is evaluated if it is true, otherwise “elseif” statement is evaluated, otherwise the block of code under “else” will get executed.

8. **Operators**
    a. The GIAA language supports a basic arithmetic operations such as addition, subtraction, multiplication, and division which are represented by ‘+’, ‘-’, ‘*’, and ‘/’ respectively.
    b. The GIAA language supports basic arithmetic comparisons such as less than, greater than, equal to, or not equal to.
    c. *Associativity:* All the operators are left associative and the assigned operators must be right associative.
    d. *Precedence:* The language gives higher precedence to the multiplication and division operations as compared to the addition/subtraction operations.
    e. Ternary operator is also handled by the GIAA language.

9. **Print**
    In order to print any string to the console, the GIAA language supports the functionality of “print” keyword followed by a space and a string to be printed.

GIAA Programming Language Grammar

The below demonstrates the proposed set of tentative grammar rules for the GIAA language.

Terminal Rules

```
INTEGER       := /^[0-9]+$/
BOOL          := /‘True’ | ‘False’/
STRING        := /\”[\x00-\x7F]*\”/
CHAR          := /'[\x00-\x7F]'/
FLOAT         :=  /([0-9]*[.])?[0-9]+/
DATATYPE      := ‘int’ | ‘bool’ | ‘string’ | ‘float’  | ‘char’
IDENTIFIER    := /^[a-zA-Z_$][a-zA-Z_$0-9]*$/

ADD           := ‘+’
SUB           := ‘-’
MUL           := ‘*’
DIV           := ‘/’
AND           := ‘and’
OR            := ‘or’
NOT           := ‘not’
ASSIGN        := ‘=’
COMPARE       := ‘>’ | ‘<’ | ‘<=’ | ‘>=’ | ‘==’ | ‘!=’
CONDOP        := and | or | not

IF            := ‘if’
ELSE          := ‘else’
ELSEIF        := ‘elseif’

WHILE         := ‘while’
FOR           := ‘for’
IN            := ‘in’
RANGE         := ‘range’

COMMENT       :=  ‘@’

SBLOCK        := ‘{‘
FBLOCK        := ‘}’
DELIMITER     := ‘;’
COMMA         := ‘,’
SQUOTE        := ‘’’
EQUOTE        := ‘’’
SPACE         := //
BEGIN         := ‘begin’
END           := ‘end’

OPENPARENTHESIS    := ‘(‘
CLOSEPARENTHESIS  := ‘)’
PRINT         := ‘printout’
```

Non Terminal Rules

program := BEGIN statements DELIMITER END | comment DELIMITER BEGIN statements DELIMITER END.

statements := allstatements DELIMITER statements | statements

allstatements := print | declaration | assign | ifelse | while | for

declaration := DATATYPE SPACE IDENTIFIER ASSIGN data | DATATYPE SPACE IDENTIFIER

assign := IDENTIFIER ASSIGN expression

print := PRINT SPACE SQUOTE STRING EQUOTE | PRINT SPACE IDENTIFIER | PRINT SQUOTE STRING EQUOTE | PRINT IDENTIFIER

ifelse := IF OPENPARENTHESIS condition CLOSEPARENTHESIS SBLOCK statements DELIMITER FBLOCK | IF OPENPARENTHESIS condition CLOSEPARENTHESIS SBLOCK statements DELIMITER FBLOCK DELIMITER, elseifLoop DELIMITER ELSE SBLOCK statements DELIMITER FBLOCK | IF OPENPARENTHESIS condition CLOSEPARENTHESIS SBLOCK statements DELIMITER FBLOCK DELIMITER ELSE OPENPARENTHESIS statements DELIMITER CLOSEPARENTHESIS

elseifLoop := elseifLoop1 DELIMITER elseifLoop | elseifLoop1
elseifLoop1 := ELSEIF OPENPARENTHESIS condition CLOSEPARENTHESIS SBLOCK statements DELIMITER FBLOCK

while := WHILE OPENPARANTHESIS condition CLOSEPARENTHESIS SBLOCK statements FBLOCK
for := FOR forRange SBLOCK statements FBLOCK

forRange := OPENPARENTHESIS IDENTIFIER ASSIGN expression DELIMETER IDENTIFIER COMPARE expression DELIMETER CLOSEPARANTHESIS | OPENPARENTHESIS IDENTIFIER ASSIGN expression DELIMETER IDENTIFIER COMPARE expression DELIMETER expression CLOSEPARANTHESIS | IDENTIFIER IN RANGE OPENPARENTHESIS expression COMMA expression CLOSEPARANTHESIS

condition := IDENTIFIER SPACE COMPARE SPACE expression | IDENTIFIER SPACE COMPARE expression CONDOP condition | BOOL

comment := COMMENT STRING
expression := term ADD expression | term SUB expression | term
term := factor MUL term | factor DIV term | factor
factor := OPENPARENTHESIS expression CLOSEPARENTHESIS | data | IDENTIFIER
data := INTEGER | BOOL | STRING | FLOAT | CHAR