SER 502 - Project

Milestone 1

Team Members:

Abhishek Haksar,
Rohit Kumar Singh,
Sarvansh Prasher,
Surya Chatterjee

Github Repository URL -: https://github.com/sarvanshprasher/SER502-Spring2020-Team20

Language Details

Name: *LitePiler*

Design:

1. The **Program** for our coding language needs to be encapsulated by a **Structure**.

2. **Structure Block**

- The **Structure** will comprise of the following parts-
    a. The structure code is required to start with **[enter]** keyword and end with a **[exit]** keyword.
    b. Within the [enter] and [exit] block the code will have **Declaration block** and **Operation block.**

    Example-
    structure → **[enter]**, declaration, operation, **[exit].**

3. **Declaration block**

- In this block, we mainly deal with various ways to declare variables for the code and how to assign values to them. These are the following ways we can declare the variable in this programming language-
    a. Either we can just declare a variable without assigning any values.
    b. Or we can also declare variables and assign values to it.
    c. Various types of variables that are accepted by this programming paradigm are- **int, bool, string.**
    d. To assign values to a variable, the user must use the **equal (=)** operator.

- Every keyword must be separated by a **comma(,)**.

- Every statement must be separated by a **semi colon(;)**.

- The variable can be assigned a **numerical** value or **value of another variable.**

- **Example-**
  Variable_type, variable_name => every keyword separated by a comma(,)

  Variable_type, variable_name; Variable_type, variable_name, =, name; In this example, two statements are shown which are separated by a semi colon(;).

4. Once the variables are declared, we can perform various tasks in the **Operation block.** In this block, we can perform various loops, condition checking, and expression evaluations.

5. **Operation Block**

- In the operation block as well we must have provision to assign values to variables as often during expression evaluation we need to store values for further calculation.

- Hence, we have provisions to assign values to variables here as well.

- Variables can not only be assigned numerical values, but they can also be assigned an **expression**.

- **Expression-** Any PEMDAS equation can be considered as an expression. For our use, we have divided the expression into following types-
  a. **Horizontal expressions**
     i. Summation operation can be performed by the **“+” operator.**
     ii. Subtraction operation can be performed by the **“-” operator.**

  b. **Vertical expressions**
     i. Multiplication can be performed by the **“ * ” operator.**
     ii. The division can be performed by the **“ / ” operator.**

  c. **Boolean expressions**
     i. Variables can be assigned boolean values such as **“True”** or **“False”.**
     ii. To show if two expressions or values are the same we use the following operator **“ :=: ”.**
     iii. To show if two expressions or values are **NOT the same** we use the following operator “ **~= “.**
     iv. To show if a value is greater than another value we use the **greater than operator (>).**

v. To show if a value is less than another value we use the **less than operator (<).**
vi. To show if a value is greater than or equal to another value we use the **greater than and equal operator separated by a comma (>,=).**
vii. To show if a value is less than or equal to another value we use the **less than operator and equal operator separated by a comma(<, =).**

- Our program also has the provision for loops and condition checkings. These are the following techniques that are available-

a. IF condition

- Used to check if a condition is satisfied or not.

Syntax- [if], condition, [then], operation, [else], operation, [endif].

- The **condition** can be any boolean expression mentioned in the previous step.

- The If block must have an else block attached to it which mentions what to do in case the expression fails.

b. While Do loop

- Used to check if a condition and if it is satisfied the loop continues and performs the tasks mentioned in the **DO block.**

Syntax- [while], condition, [do], operation, [endwhile].

- The loop must have the following keywords- **"While", "do" and "endwhile".**

c. When loop

The when loop has two variations-

a. Without Range()

- This loop takes into consideration a condition and as long as the condition satisfies, the loop continues.
- **[when], condition, [repeat], operation, [endrepeat].**

b. With Range()

- This loop takes into consideration a variable and continues the iteration as long as the value of the variable is in **Range().**
- Range is a function that is used for the iteration which specifies the starting and ending value of the loop.
- **[when], word, [in], [range],["("],number,number,[")"], [repeat],operation,[endrepeat].**

- The Language also has provision for Printing anything the user wants.
  Syntax-
  [display], exp, [;].

Grammar:

```
:-table exp/2,verticalExp/2.

% Rule for the main function of language.
program -->structure,[.].

% Rule for structure inside the program
structure -->[enter],declaration,operation,[exit].

% Rule for declarations inside the structure
declaration -->[const],word,[=],number,[;],declaration.
declaration -->varType,word,[;],declaration.
declaration --> varType,word,[=],word,[;],declaration.
declaration--> [const],word,[=],number.
declaration--> varType,word.
declaration --> varType,word,[=],word.

% Rule for variable types in language.
varType --> [int].
varType --> [bool].
varType --> [string].

% Rule for assigning values to variable.
assignValue --> word, [=] ,exp, [;].
assignValue -->  word, [is], boolExp, [;].
assignValue -->  word, [=], ternary, [;].

% Rule for the operations done in between structure.
operation --> declaration,operation.
operation --> assignValue, operation.
operation --> routine, operation.
operation --> print, operation.
operation--> structure,[;],operation.
```

```
operation --> declaration.
operation --> assignValue.
operation --> routine.
operation --> print.

% Rule for the routines done in between operations.
routine --> word,[:=],exp,[;],routine.
routine --> structure,[;],routine.
routine --> word,[:=],exp.
routine --> [if], condition, [then], operation, [else], operation, [endif].
routine -->[while],condition,[do],operation,[endwhile]|structure.
routine --> [when], condition, [repeat], operation, [endrepeat].
routine --> [when], word, [in], [range],["("],number,number,[")"],
    [repeat],operation,[endrepeat].


% Rule for evaluating ternary expressions.
ternary --> ["("],boolExp,[")"],["?"],generalValue,[:],generalValue.

% Rule for conditions in routines.
condition --> boolExp, [and], boolExp.
condition --> boolExp, [or], boolExp.
condition --> [~], boolExp.
condition --> [not], boolExp.
condition --> boolExp.

% Rule for determining boolean expression.
boolExp --> [true].
boolExp --> [false].
boolExp --> [not], boolExp.
boolExp --> exp,[=],exp.
boolExp --> exp, [:=:], exp.
boolExp --> exp, [~=], exp.
boolExp --> exp, [<],[=], exp.
boolExp --> exp, [>],[=], exp.
boolExp --> exp, [<], exp.
boolExp --> exp, [>], exp.
boolExp --> exp, [:=:], boolExp.
boolExp --> exp, [~=], boolExp.

% Rule for evaluating the horizontal expression(includes addition &
difference).
```

```
exp --> exp,horizontal,verticalExp | verticalExp.
horizontal --> [+].
horizontal --> [-].


% Rule for evaluating the vertical expression(includes multiplication &
division).
verticalExp --> verticalExp,vertical,paranthesis | paranthesis.
vertical --> [*].
vertical --> [/].

% Rule for evaluating the expression inside paranthesis.
paranthesis --> ["("] , exp , [")"].
paranthesis --> generalValue.

% Rule for evaluating the expression inside paranthesis.
generalValue --> word|number.

% Rule for including word & numbers.
word --> [X],{atom(X)}.
number --> [X],{number(X)}.

% Rule for printing values.
print --> [display],exp,[;].
```

Implementation of language:

We are going to implement all the stages of language in the **PROLOG** programming language. We will be writing predicates and based on them we will be defining rules.

Information about the lexer:

Here we will be defining a table that will map the token identifiers with their values and check their consistency from the parser. We are working on implementing that token table as we are designing the context-free grammar which will help us in defining the lexer tokens.

Information about the Parser:

We are going to use the Top-Down parsing technique for the parser in which our parser will construct a syntax tree from the top and then give it to the interpreter for evaluating those tokens.

Information about the Interpreter:

We are going to implement the reduction machine. Here we are going to use environment variables that will keep an account of the key and value pairs in the list.

Data Structure used by the Interpreter:

As we are going to use the PROLOG programming language, so we will be using the List data structure most.