MileStone2 Documentation (Team 8)

Shubhangi Gupta : 1219474243
Arjun Borkhatariya : 1220063169
Paresh Pandit : 1220832314
Nitish Tripathi : 1219500269

Name of the Language: Impetus

Context Free Grammar (BNF)

```
<Program> ::= main, <Block>.

<Block> ::= { <Statements>}

<Statements> --> <Declarations>, <Statements>
<Statements> --> < Commands>, <Statements>
<Statements> --> <Declarations>
<Statements> --> <Commands>

<Declarations> ::= <Datatype> <Identifier> = <Number> ;
<Declarations> ::= <Datatype> <Identifier> = " <String> " ;
<Declarations> ::= <Datatype> <Identifier> ;
<Declarations> ::= <Datatype> <Identifier>=<Boolean>[?]< Expression>[:] <Expression> ;

<Commands> ::= <Identifier> = <Expression> ;
<Commands> ::= for (<Expression> ; < Boolean > ; <Expression>) < Commands >
<Commands> ::= for <Identifier> in range (<Expression > : < Expression >) < Commands >
<Commands> ::= for <Identifier> in range (<Expression >)< Commands >
<Commands> ::= if (<Boolean>) <Commands> else < Commands >
<Commands> ::= if (<Boolean>) <Commands>
<Commands> ::= while(<Boolean>) < Commands >
<Commands> ::= <Identifier> += <Expression> ;
<Commands> ::= <Identifier> -= <Expression>;
<Commands> ::= <Identifier> *= <Expression>;
<Commands> ::= <Identifier> /= <Expression> ;
<Commands> ::= <Identifier>++ ;
<Commands> ::= print << 'String (N)' | print << <identifier> | print << 'String (N)'<< 'String (N)';
```

<Commands>::= < Block>
<Commands>::= <Identifier> =<Boolean>[?]< Expression>[:] <Expression> ;

<Boolean> ::= true
<Boolean> ::= false
<Boolean> ::= <Expression> <Comparator> <Expression>
<Boolean> ::= [!]<Boolean>

<Comparator> ::= ==
<Comparator> ::= >
<Comparator> ::= <
<Comparator> ::= >=
<Comparator> ::= <=
<Comparator> ::= !=

<Operator> ::= =
<Operator> ::= *
<Operator> ::= /
<Operator> ::= +
<Operator> ::= -

<Expression> ::= <Datatype> < Identifier>[=]< Expression>
<Expression> ::= <Expr_increment>
<Expression> ::= <Expr_minus>
<Expr_increment >::= <Expr_syntactic_sugar_add>
<Expr_increment> ::= <Identifier >[++].
<Expr_syntactic_sugar_add>::= <Identifier> [+=]<Expression>
<Expr_syntactic_sugar_add>::=<Expr_syntactic_sugar_minus>
<Expr_syntactic_sugar_minus> ::= <Identifier> [-=] <Expression>
<Expr_syntactic_sugar_minus> ::=<Expr_syntactic_sugar_multiply>
<Expr_syntactic_sugar_multiply>::= <Identifier> [*=]<Expression>
<Expr_syntactic_sugar_multiply >::= <Expr_syntactic_sugar_divide>
<Expr_syntactic_sugar_divide> ::= <Identifier> [/=]<Expression>
<Expr_syntactic_sugar_divide> ::= <Expr_minus>
<Expr_minus>::=< Expr_minus> [-] <Term>
<Expr_minus>::=<Term>

<Term>::= <Term> [+] <Multiply>
<Term>::= <Multiply>

<Multiply> ::= <Multiply>[*]<Division>
<Multiply> ::= <Division>
<Division> ::= <Division> [/]<Number>
<Division>::= <Brackets>
<Brackets> ::= ['(']< Expression > [')'].
<Brackets>::= <Number>
<Brackets>::= < Identifier >

<Datatype> ::= string | float

<Number> ::= ['(']<Expression> [')'].
<Number> ::= [N], { number(N) }.

<Identifier> ::= atom()

<String> ::= [a-zA-Z_][a-zA-Z_][a-zA-Z0-9_]

Design of Language :

Program : Program can have a block consisting of declarations and commands.

Block : Block can have multiple declarations and commands.

Syntax for block :

```
block
{
// declarations
// commands
//print statement
}
```

Declarations: The language supports declaration of variables.

Declaration support two data types: string and float

for eg.

```
y = "test"
float = 5.5
```

Commands: Commands in this language can be multiple conditional constructs such as 'for', 'while' or 'if else'. Initialization is also a part of commands.

Control Structures: Language supports if-else control statements

Syntax:

```
if(Conditional expression)
{
//Declaration
//Commands
}
else
{
//Declarations
//Commands
}
```

Loop Structures : Language supports - for loop, while loop and for in range loops.

Syntax of while loop :-

```
while(Condition expression)
{
//block
}
```

Syntax of for loop :-

```
for(Condition expression)
{
//block
}
```

Syntax of for i in range loop:-

```
for (identifier in range (number , number) )
{
//block
}

 for <Identifier> in range (<Expression >))
{
 //block
}
```

Ternary operator:
Identifier = Condition ? (Expression evaluated if condition is true) : (Expression evaluated if condition is false)

Expressions : arithmetic expressions can be evaluated using arithmetic operators.
Ex : x=2+3 or y=6*3.

Arithmetic operators : Arithmetic operations supported are :
Addition (+) , Subtraction (-), Multiplication (*), Division (/)
Conditional and logical operators : Language supports the following comparisons :
== , <=, >=, >, < , != , &&, ||

Print Statement : The print statement in the language is ‘print’.
ex: print << (String (N)).

New Line : “endl” keyword is used to represent nextline. This keyword will be used to create line breaks for printing console outputs

Code comments(Documentation) : “#” will be used for single line comments in the code and can be used to describe the code. As a future prospect it can be used to generate documentation for the code.

Choice of tools for Language:

lexical analyser: Python
Parser: using Prolog
Evaluator : using Prolog

GIT Repository : Github Link

YouTube Video : Youtube Link