PROJECT MILESTONE-1

VENKATA SAI TARUN NUKA(ntarun@asu.edu)
SRI VIKAS GANUGU(sganugu@asu.edu)
NITIN SURYA MOTURU (nmoturu@asu.edu)

Name: Striger

Design:
Striger is a simple programming language that consists of 2 variable types i.e. String and Integer [as str and int respectively] 3 components that make up a program.

1. Initialization: The initialization part of the program consists of defining new variables[a unit that helps store and access values]. The variables are defined with a name and a type(mandatory)[Variable type is a generalization of the values assigned to the variable].
   Syntax: int x = 23
   int is type
   X is name
   23 is the value assigned to the variable x
2. Computation: The computation part of the program is the main part of the program where all calculations and manipulations happen. This part consists of conditionals and loops:
   a. Conditionals: These are conditions as the name suggests that compute values if a condition is met(True or False).
   Example: “if” sky is light blue then it is day, “else” it is night.
   ```
   Syntax: if (boolean_expression):
                   Arithmetic_expression
           Else:
                   Arithmetic_expression
   ```
   b. Loops: These are lines of the program that repeat a part of the program when a condition is met until the condition fails.
   Example: while time>9am and time<5pm, you have to work
   Syntax:
   ```
   while(boolean_expression):
           arithmetic_expression
   ```
3. Conclusion: This part of the program is an optional part of the program where the results from the computation are displayed or returned for further computation.
   Example:
   print(x) → 23 (x is the variable defined in a previous example for initialization).
   print(‘2312312asafg’)
   print(231)
   print(2<5)--> true.

The programming language also promotes different user-friendly commands like:

1. If_then_else
2. For i in range
3. Auto increment operators
4. Ternary operator

For better readability of the code addition of tab spaces after conditions and loops is made mandatory.

Grammar:

```
program : components

components: initialization  '\n' conclude* computation '\n' conclude*

"""PRE-DEFINITIONS"""
"""
asnmt_op: '='

variable_type: str | int

lowercase: 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' |
'w' | 'x' | 'y' | 'z'

digit: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

variable_name: lowercase (lowercase | digit)*

int: digit+

str: "'" (lowercase*)(digit*) "'"
   | '"' (lowercase*)(digit*) '"'

"""




"""CODE BLOCK"""
"""
initialization: initialization '\n' initialization
              | variable_type variable_name asnmt_op expression
              | variable_type variable_name
```

```
computation: computation '\n' computation
            | conditionals
            | loops

conclude: print_statement
"""


"""Print Statement"""
"""
print_statement: 'print' variable_name
                | 'print' '(' variable_name ')'
                | 'print' '(' str ')'
                | 'print' '(' int ')'
                | 'print' '(' boolexpr ')'
"""




"""Conditions"""
"""
conditionals: if_condition
            | if_then_else
            | terinary

if_condition: 'if' boolexpr ':' '\n' computation
            | 'if' boolexpr ':' '\n' computation '\n' 'else' ':' '\n' computation

if_then_else: 'if' boolexpr 'then' computation 'else' computation

terinary: boolexpr '?' arthexpr ':' arthexpr
"""



"""LOOPS"""
"""
loops: for_loop
     | while_loop
     | for_inrange

for_loop: 'for' '(' initialization ';' arthexpr ';' arthexpr ')' '\n' '\t' computation
```

```
for_inrange: 'for' variable_name 'in' 'range' '('  ',' ')'

while_loop: 'while' boolexpr ':' '\n' '\t' computation
"""




"""EXPRESSIONS"""
"""
expression: boolexpr
           | arthexpr

boolexpr: 'true'
        | 'false'
        | arthexpr '==' arthexpr
        | 'not' boolexpr
        | arthexpr 'and' arthexpr
        | arthexpr 'or' arthexpr
        | arthexpr '>' arthexpr
        | arthexpr '<' arthexpr
        | arthexpr '>=' arthexpr
        | arthexpr '<=' arthexpr
        | arthexpr '!=' arthexpr
        | '(' boolexpr ')'

arthexpr: arthexpr '+' arthexpr
        | arthexpr '-' arthexpr
        | arthexpr '*' arthexpr
        | arthexpr '/' arthexpr
        | arthexpr '++'
        | arthexpr '--'
        | '++' arthexpr
        | '--' arthexpr
        | variable_name
        | int
"""
```

INTERPRETOR: JAVA

Java is a programming language created in 1995. It's famous for being able to run on many different types of computers without needing to change the code. Java is organized around the idea of objects, making it easier to write and manage big programs. It automatically takes care of cleaning up unused computer memory, so programmers don't have to worry about it. Java comes with lots of ready-made tools to help programmers do common tasks, like making graphics or connecting to the internet. It's often used to build big software systems for businesses and other organizations. Many people use Java, and there are lots of helpful resources available for learning and using it. It's also known for being safe and secure, protecting computers from harmful software.JAVA uses the following data structures:

1. Lists:
   - Lists in Java are ordered collections of elements.
   - They allow duplicate elements and maintain the insertion order of elements.
   - Common implementations of lists in Java include ArrayList and LinkedList.
   - ArrayList: Implements a dynamic array that can grow as needed. Offers fast access to elements by index.
   - LinkedList: Implements a doubly-linked list where elements are stored as nodes. Offers fast insertion and deletion at any position.

2. Maps:
   - Maps in Java are collections that store key-value pairs.
   - Each key in a map must be unique, and each key maps to exactly one value.
   - Common implementations of maps in Java include HashMap, TreeMap, and LinkedHashMap.
   - HashMap: Implements a hash table for storing key-value pairs. Provides fast lookup and insertion, but does not maintain any specific order of elements.
   - TreeMap: Implements a Red-Black tree for storing key-value pairs. Maintains elements in sorted order based on the natural ordering of keys or a custom Comparator.
   - LinkedHashMap: Maintains the insertion order of elements, in addition to providing key-value mappings like HashMap.

3. Sets:
   - Sets in Java are collections that store unique elements.
   - They do not allow duplicate elements.
   - Common implementations of sets in Java include HashSet, TreeSet, and LinkedHashSet.
   - HashSet: Implements a hash table for storing unique elements. Provides constant-time performance for basic operations.
   - TreeSet: Implements a Red-Black tree for storing unique elements. Maintains elements in sorted order based on the natural ordering of elements or a custom Comparator.
   - LinkedHashSet: Maintains the insertion order of elements, in addition to providing uniqueness like HashSet.

PARSER:ANTLR

ANTLR is a helpful tool for making parsers that understand different languages and formats. Here's a simpler way to explain it:

1. Easy Parsing: ANTLR makes it simple to create parsers for various languages and data types. You just describe the rules of your language, similar to writing down cooking instructions.

2. Writing Rules: To use ANTLR, you write down how your language should be structured. It's like creating a blueprint for understanding your language.

3. Lexer and Parser: ANTLR automatically creates two important parts: a lexer, which breaks text into small pieces, and a parser, which puts those pieces together according to your rules.

4.Tree Structure: ANTLR can also create a tree structure to organize the text it understands, making it easier to work with.

5. Works with Different Languages: ANTLR can generate parsers in various programming languages, so you can choose the one you're most comfortable with.

6. Helpful Tools: ANTLR comes with tools like ANTLR Works, which helps you write and test your rules visually, making the process easier to understand.

7. Handling Mistakes: ANTLR is good at dealing with errors in the input text, making it more forgiving if there are mistakes.

The data structure present in ANTLR are:

1. Parse Tree:
   - The parse tree shows how the input text is structured based on the rules you've defined.
   - It's like a family tree, where each node represents a part of the input text, following the rules you've set.

2. Abstract Syntax Tree (AST):
   - The abstract syntax tree shows the important parts of the input text, ignoring minor details.
   - It's like a simplified version of the parse tree, focusing on the key elements of the text.