Language Name: Sparky

General System Requirements:

Processor: Intel and AMD processors (Processors with instruction set capable of imperative paradigm)

Operating System on which compiler and runtime are built: Windows OS.

Type of Language: Imperative

Data Structure Used: Abstract Syntax Tree, Hash Map, Stack

Tools Used: Git, Eclipse, Antlr (https://www.antlr.org/), ANT

Parsing Technique Employed: Antlr (Feeding a grammar (.g4) file to generate an abstract syntax tree)

Steps to install and run Antlr on Windows:

- Download https://www.antlr.org/download/antlr-4.8-complete.jar.
- Add antlr4-complete.jar to CLASSPATH, either:
- Permanently: Using System Properties dialog > Environment variables > Create or append to CLASSPATH variable
- Temporarily, at command line:
- SET CLASSPATH=.;C:\Javalib\antlr4-complete.jar;%CLASSPATH%
- Create batch commands for ANTLR Tool, TestRig in dir in PATH
- antlr4.bat: java org.antlr.v4.Tool %*
- grun.bat: java org.antlr.v4.gui.TestRig %*

Alternative Steps using Dos Key commands to run Antlr:

- Doskey antlr4=java org.antlr.v4.Tool $*
- Doskey grun =java org.antlr.v4.gui.TestRig $*

Directions/instructions to install Sparky language

Follow the below steps to install via GitHub:

- Clone the git project from **https://github.com/MayankBatra005/SER502-Spring2020-Team25**
- Download this git project and Unzip the project in a new folder. **< Make sure there should be no spaces or invalid characters>**
- Open the project in Eclipse using following steps:
- Files >> Open Project From File System. Browser your project folder here upto extracted project directory.

Steps to build JARS for sparky

1. Right click on the project folder at the top.

2. Click on Export -> Under the Java Option, select Runnable JAR File option. -> Click Next.

3. Select the destination directory in which you want to export the jar.

4. Under Library handling chose "package required libraries into generated Jar"

6. Click on Finish

Note : Jar will be generated under selected destination folder mentioned in step 3

→Please refer to Installation steps as shown in YouTube video

How to run any program using Sparky language

Run via Eclipse:

1. Select the Compiler.java class under src>sparkyCompiler>Complier.java

2. Right click and select run as Run Configurations

3. Select Arguments tab

4. Provide the complete path of the file located on your disk with extension as sparky

**** Make sure the file should be stored on path containing no white spaces or invalid characters such as _ / - etc. ****

5. Click on Run

6. Output can be seen in Eclipse console

Run using compiler Jars:

Pre requisite:

1. Jar should be generated as illustrated in above steps
2. Source code with extension as ".sparky" is created and path to this file is known

Steps to run on console(Windows command prompt):

1. Navigate to the folder where compiler.jar was created

2. open command prompt (CMD) on this location

3. Type the following command Java - jar compiler.jar "path\Filename.sparky"

Path stands for the path to the file

Filename stands for the name of the file which contains the source code

4. Hit Enter

5. Code is executed on command prompt

Grammar Snippet:

```
grammar Sparky;
program: LIVE ball DIE;
ball: expression* | declare* expression*;

declare:
(datatype STUFF EQUALTO NUMBER SEMICOLON)|
(datatype STUFF SEMICOLON)|
(HAINA STUFF EQUALTO booleanvalue SEMICOLON)|
(HAINA STUFF SEMICOLON)| stringdatatype STUFF EQUALTO STRINGLITERAL SEMICOLON | stringdatatype STUFF SEMICOLON;

expression
 : assignment
 | ifte
 | loopum
 |ternary_operator
 |print;

assignment
 : STUFF EQUALTO expr SEMICOLON |
 STUFF EQUALTO yesnostatement SEMICOLON
 ;

ifte
 : IF yesnostatement in_loop ('warna' in_loop)? FI
 ;


loopum : loop_for|loop_while | loop_for_range;
loop_for: 'for' LSmoothBrace for_declare? ';' for_expression? ';' for_expr? RSmoothBrace in_loop;
loop_while
 : WHILE yesnostatement in_loop
 ;

 loop_for_range: 'for'  STUFF 'in' 'range' LSmoothBrace NUMBER COMMA NUMBER RSmoothBrace in_loop;

in_loop: LCurlyBrace ball RCurlyBrace| expression;

for_expr: STUFF EQUALTO expr;
for_expression :expr YESNOOPERATOR expr;
for_declare:datatype STUFF EQUALTO NUMBER;

term: NUMBER | STUFF | STUFF op=(MUL | DIV) term | NUMBER  op=(MUL | DIV) term;
expr: term | term op=(PLUS | MINUS) expr | NOT expr;
yesnostatement : booleanvalue | expr YESNOOPERATOR expr |yesnostatement ANDOROPERATOR yesnostatement;
ANDOROPERATOR: AND|OR;

AND: 'and';
OR: 'or';
NOT:'not';


ternary_operator: yesnostatement '?' in_loop ':' in_loop;

print:'print' LSmoothBrace expr RSmoothBrace SEMICOLON;
```

```
LIVE: 'Live';
DIE: 'Die';
FI: 'fi';


YESNOOPERATOR: ASSEQ| LESS_THAN| MORE_THAN | LESS_THAN_EQ | MORE_THAN_EQ ;
EQUALTO : '=';
ASSEQ : '==';
LESS_THAN : '<';
MORE_THAN: '>';
LESS_THAN_EQ : '<=';
MORE_THAN_EQ : '>=';

warna :'else';

PLUS : '+';
MINUS :'-';
MUL : '*';
DIV : '/';
SEMICOLON : ';';
COMMA : ',';

LSmoothBrace : '(';
RSmoothBrace : ')';
LCurlyBrace : '{';
RCurlyBrace : '}';
DQ: '"';

STRINGLITERAL: DQ (~["\\\r\n])* DQ;
HAINA: 'haina';
haina:'bool';
datatype: INTEGER| DOUBLE | HAINA;
stringdatatype: STRING;
INTEGER: 'int';
STRING: 'string';
DOUBLE: 'double';
IF : 'if';
WHILE : 'while';
STUFF:[a-zA-Z_] [a-zA-Z_0-9]*;
NUMBER:[0-9]+;
WS: [ \t\r\n] -> skip;
booleanvalue: 'yup' | 'nup';
yup:'true';
nup:'false';
```

Sample Codes: (https://github.com/MayankBatra005/SER502-Spring2020-Team25/tree/master/data)

Example1: arithmaticOps.sparky

```
Live
int a=40;
int b=8;
int result;
result=a+b;
print(result);
result=a-b;
print(result);
```

```
result=a*b;
print(result);
result=a/b;
print(result);
Die
```

Example 2: fibonacci.sparky

```
Live
int count = 7;
int counter =1;
int firstFib=1;
int secondFib=1;
int sum;
while counter<=count
{
print(firstFib);
sum=firstFib+secondFib;
firstFib = secondFib;
secondFib=sum;
counter=counter+1;
}
Die
```

Example 3: factorial.sparky

```
Live
int result = 1;
int n=5;
for (int i=2; i<=n; i=i+1)
{result=result*i;}
print(result);
Die
```