SPARKY

SER 502 - Spring20 - Team 7 Presentation

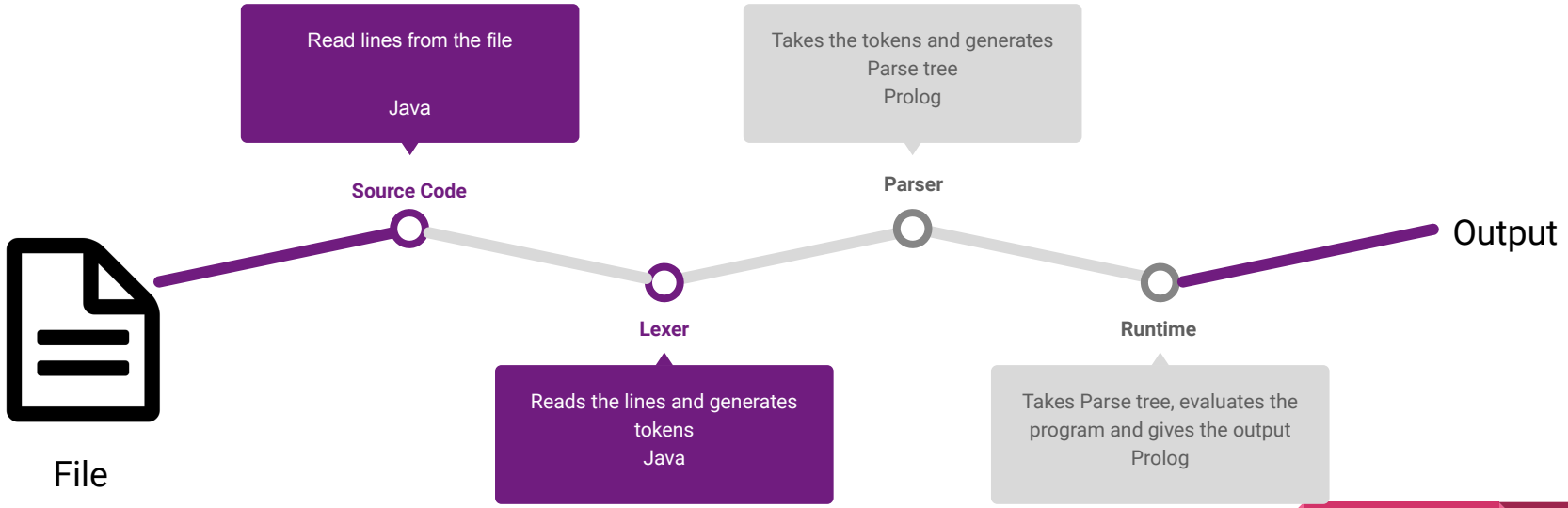
Team Members: Chandan Kiragadalu Javaregowda
Parikshith Kedilaya Mallar
Ashutosh Dey
Nagarjun Nama Aswath

Introduction To Sparky

To help implement algorithms and encode solutions to complex problems.

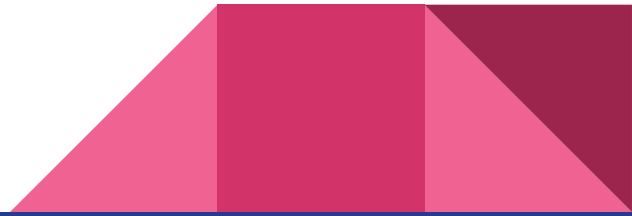
We also want to exhibit the ability of declarative languages in implementing imperative languages.

Flowchart


Read lines from the file
Java
Takes the tokens and generates
Parse tree
Prolog
Source Code
Parser
Output
File
Lexer
Runtime
Reads the lines and generates
tokens
Java
Takes Parse tree, evaluates the
program and gives the output
Prolog

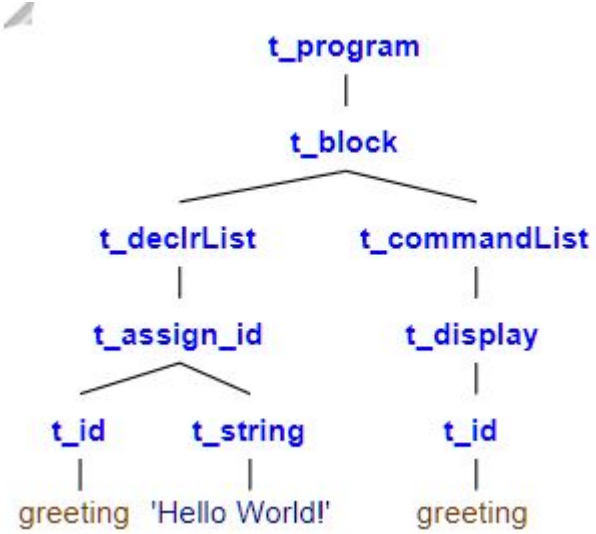
Tools

- Java 8
- SWI-SH Prolog
- Maven
- JPL



Example Parse Tree

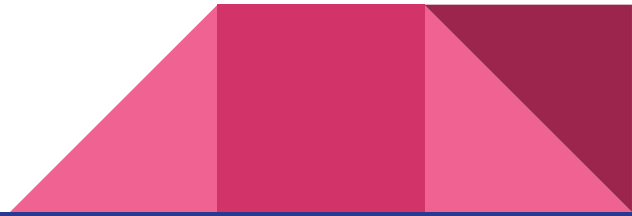
```
1  begin
2      var greeting := "Hello World!";
3      print(greeting);
4  end
```



Hello World!

Grammar

- Variables of type Boolean, Int, Float and String
- Variable Initialization: var <var_name> := <value>;
- Operations supported for Int and Float: Addition, Subtraction, Multiplication, Division
- String concatenation
- Boolean: and, or and ! (not)
- Conditional operators: <,>,=<,>=,==,!=
- If construct: if-then-else-endif

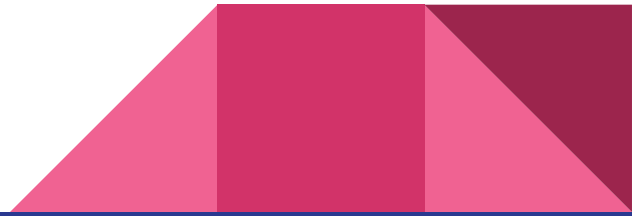


Grammar

- While: while-do-endwhile
- For: Traditional for loop similar to Java
- For: Range for loop similar to Python
- List: pushFirst, pushLast, popFirst, popLast, isEmpty
- Print: print(...);
- Ternary Operator: ‘? :’
- Increment and decrement operators: ++, - -

Lexer - Tokens

- Take input code in the file.
- Read the file and generate tokens.
- Using java.util.StringTokenizer to create tokens.
- Validation checks to have tokens in Prolog format.
- Tokens as a List are sent to Parser by the java code.



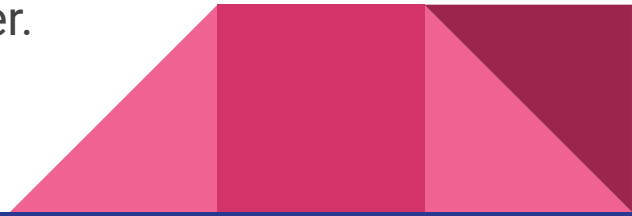
Tokens - Example

```
1  begin
2      var greeting := "Hello World!";
3      print(greeting);
4  end
```

begin,var,greeting,:,=,"",'Hello World!',"",;,print,'(',greeting,')',;,end

Parser

- Prolog DCG is used to build Parser.
- Parser takes a list of tokens as inputs.
- Parser validates all the tokens and generates parse tree.
- Parse tree is then given as return value to java code.
- The parse tree will be then sent as input to Interpreter.



Parse Tree - Example

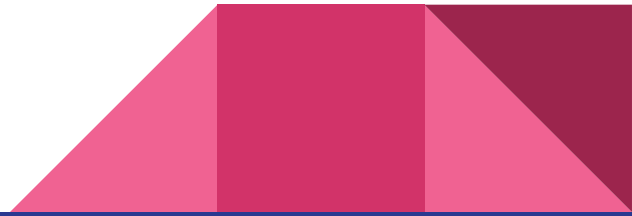
```
1  begin
2      var greeting := "Hello World!";
3      print(greeting);
4  end
```

t_program(t_block(t_declrList(t_assign_id(t_id(greeting), t_string('Hello World!'))), t_commandList(t_display(t_id(greeting)))))

Parser - Language Basic Structure

program(t_program(X)) --> block(X).

block(t_block(X,Y)) --> begin, declrList(X),commandList(Y),end.



Parser - Declaration Section

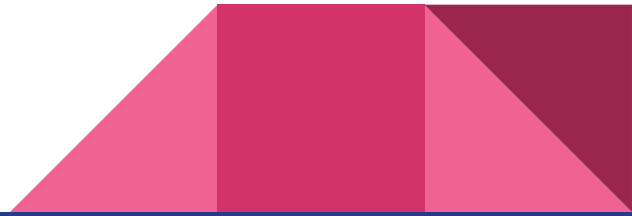
declrList(t_declrList(X,Y)) --> declR(X), endLine,declrList(Y).

declrList(t_declrList(X)) --> declR(X), endLine.

declR(t_assign_id(X,Y)) --> var, identifier(X),[:,=],expr(Y).

declR(X) --> var, identifierList(X).

declR(t_init_list(X)) --> [list], identifier(X).



Parser - Commands Section

commandList(t_commandList(X,Y)) --> commandI(X),endLine,commandList(Y).

commandList(t_commandList(X)) --> commandI(X),endLine.

commandI(X) --> display(X).

commandI(X) --> commandInitialize(X).

commandI(X) --> ifEval(X).

commandI(X) --> forEval(X).

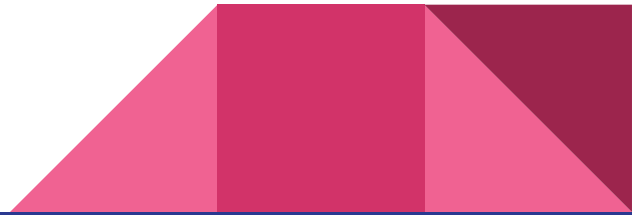
commandI(X) --> whileEval(X).

commandI(X) --> ternaryEval(X).

commandI(X) --> list_push(X).

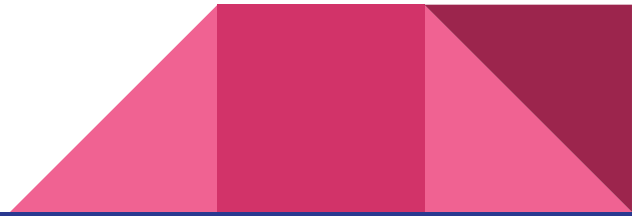
commandI(X) --> list_pop(X).

commandI(X) --> list_isEmpty(X).



Interpreter - A glimpse

```
eval_program(t_program(X),FinalEnv) :-

                                        eval_block(X,[],FinalEnv).

eval_block(t_block(X,Y), EnvIn, EnvOut) :-

                                        eval_declrList(X,EnvIn, Env1), eval_commandList(Y, Env1, EnvOut).
```



Runtime

- Runtime takes the intermediate code generated by the parser
- Input will be in the form of a parse tree to the runtime
- We are using prolog to evaluate the parse-tree nodes
- Output will be the final environment after evaluation is finished for all predicates

Runtime

eval_program(t_program(X),FinalEnv) :- eval_block(X,[],FinalEnv).

eval_block(t_block(X,Y), EnvIn, EnvOut) :- eval_declrList(X,EnvIn, Env1), eval_commandList(Y, Env1, EnvOut).

eval_declrList(t_declrList(X,Y),EnvIn, EnvOut) :- eval_declR(X,EnvIn, Env1), eval_declrList(Y, Env1, EnvOut).

eval_declrList(t_declrList(X),EnvIn, EnvOut):- eval_declR(X,EnvIn, EnvOut).

Runtime

Declaration Evaluation

eval_declR(t_identifierList(X,Y), EnvIn, EnvOut) :- eval_declR(X, EnvIn, Env1), eval_declR(Y,Env1,EnvOut).

eval_declR(t_id(X), EnvIn, EnvOut) :- update(X,0,EnvIn,EnvOut).

eval_declR(t_init_list(t_id(X)),EnvIn,EnvOut) :- update(X,([]),EnvIn,EnvOut).

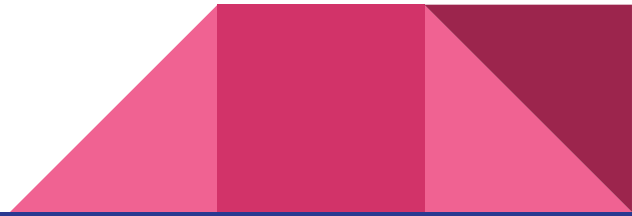
Print Evaluation

eval_commandI(t_display(X),EnvIn,EnvOut) :- eval_expr(X, EnvIn,EnvOut, Val),write(Val),nl.

Runtime

If then else evaluation

```
eval_commandI(t_ifteEval(X,Y,_Z),EnvIn,EnvOut):- eval_bool(X,EnvIn,EnvOut1,true),
                                                  eval_commandList(Y,EnvOut1,EnvOut).
eval_commandI(t_ifteEval(X,_Y,Z),EnvIn,EnvOut):- eval_bool(X,EnvIn,EnvOut1,false),
                                                  eval_commandList(Z,EnvOut1,EnvOut).
```



Runtime

While Evaluation

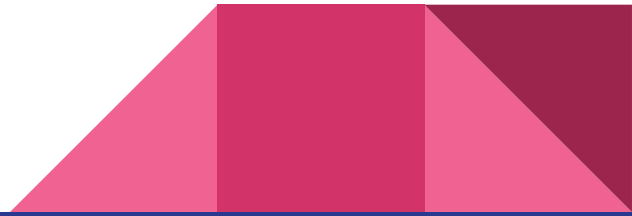
```
eval_commandI(t_whileEval(B,C),EnvIn,EnvOut):-eval_bool(B,EnvIn,EnvIn,true),
                                        eval_commandList(C,EnvIn,Env2),
                                        eval_commandI(t_whileEval(B,C),Env2,EnvOut).
eval_commandI(t_whileEval(B,_C),Env,Env):-eval_bool(B,Env,Env,false).
```

Sample Program - Declarations

```
begin
    var teamName := "Team7";
    var teamMembers := 4;
    var emptyInitalize ;
    list listExample ;

    print(teamName);
    print(teamMembers);
    print(emptyInitalize);
    print(listExample);
end
```

Sample Program - String Operations

```
begin
    var var1 := "Hello";
    var var2 := " World!";
    print(var1 + var2 );
    print("Hello" + var2);
end
```



Sample Program - If Else

```
begin
    var a := 5;
    var b := 6;
    var c := 7;


    print("Test for if");

    if (a == b and b != c)
    then
    print("Inside If");
    else
    print("Inside Else");
    endif;
end
```

Sample Program - While Loop

```
begin
    var a := 5;
    var b := 6;
    var c := 7;

    while ( a < 10)
    do
        print(a);
        a++;
    endwhile;

end
```

Sample Program For and Advanced For

```
begin
    var a := 5;
    var b := 6;
    var c := 7;

    print("Test for traditional for loop");
    for(i := 0; i <10; i++)
    do
        print(i);
    endfor;

    print("Test for advanced for loop");
    for(j := 10; j >0; j--)
    do
        print(j);
    endfor;
    ;

end
```

Runtime

Boolean Evaluation logic:

```
eval_bool(t_booleanExprCond(X,and,Y),EnvIn,EnvOut,Val):- eval_bool(X,EnvIn,Env1,Val1),
                                                         eval_bool(Y,Env1,EnvOut,Val2),
                                                         equalForAnd(Val1,Val2,Val).

eval_bool(t_booleanExprCond(X,or,_Y),EnvIn,EnvOut,true):- eval_bool(X,EnvIn,EnvOut,true).

eval_bool(t_booleanExprCond(X,or,Y),EnvIn,EnvOut,true):- eval_bool(X,EnvIn,Env1,false),
                                                         eval_bool(Y,Env1,EnvOut,true).

eval_bool(t_booleanExprCond(X,or,Y),EnvIn,EnvOut,false):- eval_bool(X,EnvIn,Env1,false),
                                                          eval_bool(Y,Env1,EnvOut,false).

eval_bool(t_booleanNegate(B),EnvIn,EnvOut,Val):-eval_bool(B,EnvIn,EnvOut,Val1),
                                                not(Val1,Val).
eval_bool(t_booleanNegate(B),EnvIn,EnvOut,Val):-eval_id(B,EnvIn,EnvOut,Val1),
                                                not(Val1,Val).

eval_bool(t_booleanExprCond(E1,==,E2),Env,NewEnv,Val):-eval_expr(E1,Env,Env1,Val1),
                                                       eval_expr(E2,Env1,NewEnv,Val2),
                                                       equal(Val1,Val2,Val).
```

Runtime

Expression Evaluation Logic:

```
eval_expr(t_add(X,Y),EnvIn, EnvOut, Val) :-
    eval_expr(X,EnvIn,EnvOut1,Val1),eval_expr(Y,EnvOut1,EnvOut,Val2),
    number(Val1),number(Val2),Val is Val1 + Val2.


%Evaluate expression when t_sub tree node is encountered
eval_expr(t_sub(X,Y),EnvIn, EnvOut, Val) :- eval_expr(X,EnvIn,EnvOut1,Val1),
    eval_expr(Y,EnvOut1,EnvOut,Val2),
    number(Val1),number(Val2),Val is Val1 - Val2.

%Evaluate expression when t_mul tree node is encountered
eval_expr(t_mul(X,Y),EnvIn,EnvOut, Val) :- eval_expr(X,EnvIn,EnvOut1,Val1),
    eval_expr(Y,EnvOut1,EnvOut,Val2),
    number(Val1),number(Val2),Val is Val1 * Val2.

%Evaluate expression when t_div tree node is encountered
eval_expr(t_div(X,Y),EnvIn,EnvOut, Val) :- eval_expr(X,EnvIn,EnvOut1,Val1),
    eval_expr(Y,EnvOut1,EnvOut,Val2),
    number(Val1),number(Val2),Val is Val1 / Val2.


eval_expr(t_add(X,Y),EnvIn, EnvOut, Val) :-
    eval_expr(X,EnvIn,EnvOut1,Val1),eval_expr(Y,EnvOut1,EnvOut,Val2),
    atom(Val1),atom(Val2),concat(Val1,Val2,Val).

eval_expr(t_id_expr_equality(X,Y),EnvIn,EnvOut,Result):-eval_expr(Y,EnvIn,EnvOut1,Result),
                                                        update(X,Result,EnvOut1,EnvOut).
```

Sample Program - Boolean

```
begin
    var tr :="TRUE";
    var fl := "FALSE";
    var x := 8;
    var y := 5;
    var z := 6;
    var t := 6;
    if (x != y and t == z  or z != t and !t == z and true)
    then
        print(tr);
    else
        print(fl);
    endif;
end
```

Sample Program - Expression

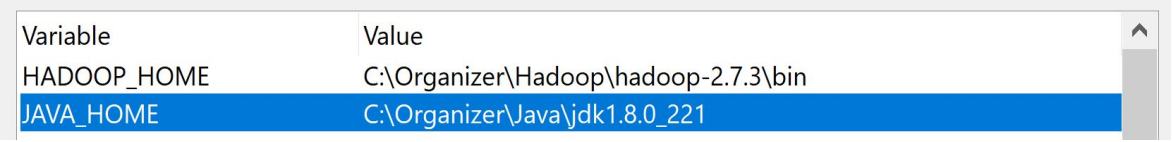
```
begin

var num1 := 1.1;
var num2 := 2.2;
var str1 := "SER";
var str2 := "502";
var str,add,sub,mul,div;

add := num1+num2;
sub := num1-num2;
mul := num1*num2;
div := num1/num2;
str := str1+str2;

print("ExpressipnEvaluation");
print(add);
print(sub);
print(mul);
print(div);
print("StringConcatination:");
print("Str1:");
print(str1);
print("Str2:");
print(str2);
print("Str1+Str2:");
print(str);

end
```

Installation Guide

- As explained by teammates, we need two software to run a program in our language. Prolog and Java. In order to improve our language and create a new version, we would require maven to build and deploy the new version.
- Prolog Installation:
  - Download the SWI Prolog **8.0.3-1** for microsoft windows from https://www.swi-prolog.org/download/stable
  - During the installation of SWI Prolog ensure to select the “Update Path Variables for All Users” options.
  - Check for the path variable and ensure that it has the below path appended.

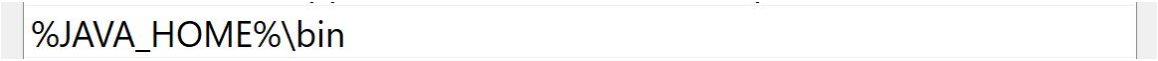
Installation Guide Continued..

- Java Installation:
  - Download the Java **8** for microsoft windows from https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
  - Ensure that JAVA_HOME variable is set as follows:



Variable	Value
HADOOP_HOME	C:\Organizer\Hadoop\hadoop-2.7.3\bin
JAVA_HOME	C:\Organizer\Java\jdk1.8.0_221

  - Once JAVA_HOME is set, update the value of Path Variable as follows:



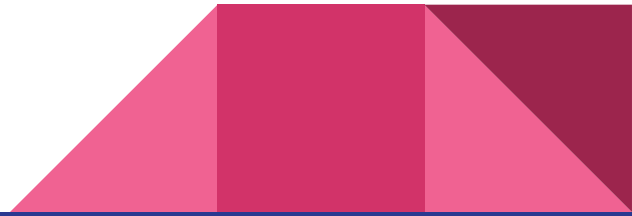
```
%JAVA_HOME%\bin
```

Installation Guide Continued..

- Maven Installation:
  - Maven can be downloaded from: https://maven.apache.org/download.cgi
  - Complete guide to install maven can be found here: https://maven.apache.org/install.html

Execution Steps:

- In command prompt, navigate to the source code folder.
- Use the command : **mvn clean compile assembly:single** to create the jar.
- To execute the we can use the below command with or without program arguments.
  - Command with program argument will pickup the program file specified in the path.
    Ex: **java -jar target\sparky-0.0.1-SNAPSHOT-jar-with-dependencies.jar C:\SampleProg.spk**
  - Command without program argument will pickup the sample program present in resources.
    Ex: **java -jar target\sparky-0.0.1-SNAPSHOT-jar-with-dependencies.jar**



Challenges Faced and Future Scope:

Challenges Faced:

- We had difficulties in printing String and Number together.
- We had tough time in converting int data type to float and vice-versa.

Future Scope:

- Recursion and function methods can be implemented in subsequent release
- Class and Objects can also be implemented in the subsequent release.