Final Compiler Project Report

An Tran, Ameya Dighe,
Kelvin Lopez,
Mohammad Naqvi

December 18, 2023
CS 153
Team AKA

Language: AKA

Our language called AKA is based on Java, Python, and some of our own ideas. Syntactically it is most similar to Java while some of the semantics are handled similarly to Python. In the end, the language we created was a mashup of different components of languages that we liked, either because we were familiar with them or because they were convenient when writing a program.

Language Constructs:

Assignment

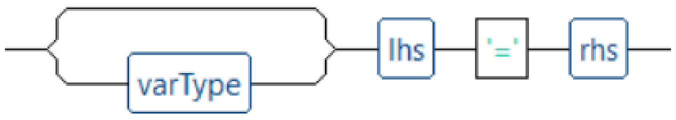
- Assignment statement types are now explicitly declared when creating a new variable. If the type identifier is absent, this means the variable is already declared and already has a type.

assignment Top

Text notation:

```
assignment : (varType)? lhs '=' rhs ;
```

Visual notation:



If Statement

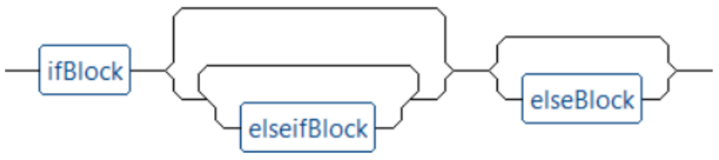
- An If statement has three segments, an if block, an if-else block, and an else block. The if block is mandatory. There can be zero up to as many if-else blocks as the programmer needs. Lastly, the else block is optional.

ifStatement Top

Text notation:

```
ifStatement : ifBlock elseifBlock* elseBlock? ;
```

Visual notation:



While Statement

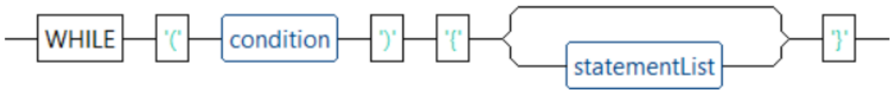
- The while statement will execute the statements within the statement until the condition for the while loop is false.

whileStatement Top

Text notation:

```
whileStatement : WHILE '(' condition ')' '{' (statementList)? '}' ;
```

Visual notation:



Guard

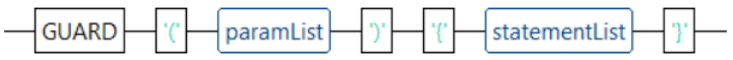
- As the name implies, the Guard statement will guard whatever expressions are given in its parameters and will immediately break if any of those expressions become false during the inside of the Guard statement.

GUARD Top

Text notation:

```
GUARD : G U A R D ;
```

Visual notation:



Display

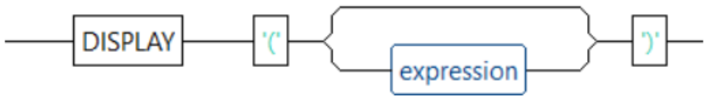
- The display statement prints out the arguments inside of the parentheses which are in the form of an expression.

display Top

Text notation:

```
display : (DISPLAY) '(' (expression)? ')' ;
```

Visual notation:



Definition Call

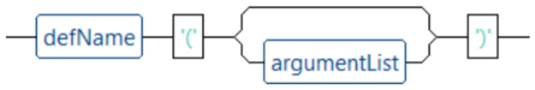
- A definition call calls either a function or procedure which is identified by the definition name.

defCall Top

Text notation:

```
defCall : defName '(' argumentList? ')' ;
```

Visual notation:



Definitions (Functions / Procedures)

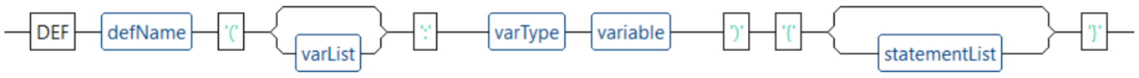
- Definitions are similar to functions and procedures in Pascal. In this language, they are referred to as the same thing but are differentiated by whether or not they return a value. A definition that returns a value will have this at end of their parameter list: : “returnType returnVariable”

definition Top

Text notation:

```
definition : DEF defName '(' varList? ':' (varType variable) ')' '{' (statementList)? '}' ;
```

Visual notation:

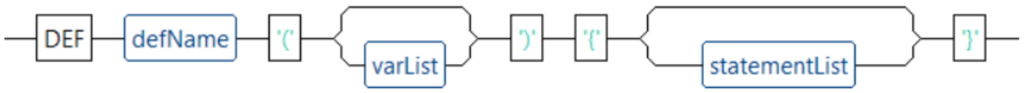


definitionnoreturn Top

Text notation:

```
definitionnoreturn : DEF defName '(' varList? ')' '{' (statementList)? '}' ;
```

Visual notation:



Expressions

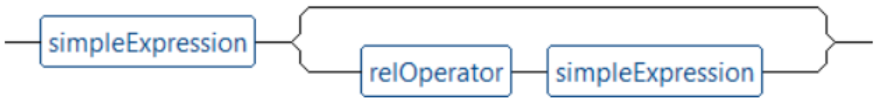
- Expressions have the same utility as the original Pascal language with some new functionality. Booleans were added in our language and we can handle true and false booleans. Numbers if combined with booleans are also treated like booleans, similar to Python (i.e. 5 == true). Secondly, strings can be concatenated with numbers like in Java (i.e. “hi “ + 5 = “hi 5”).

expression Top

Text notation:

```
expression : simpleExpression (relOperator simpleExpression)? ;
```

Visual notation:

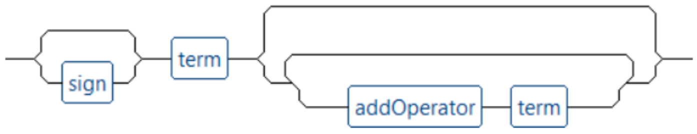


`simpleExpression` Top

Text notation:

```
simpleExpression : sign? term (addOperator term)* ;
```

Visual notation:

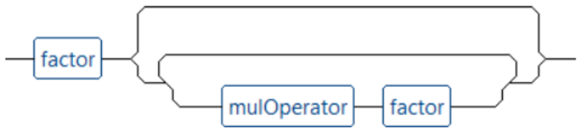


`term` Top

Text notation:

```
term : factor (mulOperator factor)* ;
```

Visual notation:

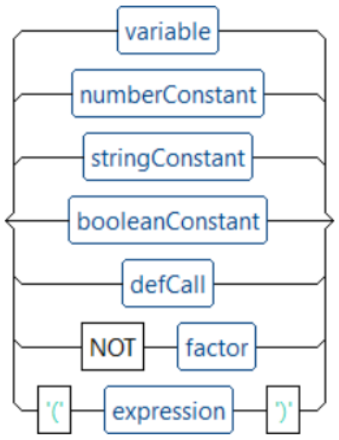


`factor` Top

Text notation:

```
factor : variable # variableFactor
notFactor | '(' expression ')' # pa
```

Visual notation:



Jasmin Code Templates

If Statement

```
        fconst_0
        putstatic       prog2/t F
        fconst_0
        putstatic       prog2/i F
        getstatic       prog2/i F
        ldc     3.0
        fcmpg
        iflt    L002
        iconst_0
        goto    L003
L002:
        iconst_1
L003:
        ifeq    L004
        fconst_1
        putstatic       prog2/t F
        goto    L001
L004:
        getstatic       prog2/i F
        ldc     5.0
        fcmpg
        iflt    L005
        iconst_0
        goto    L006
L005:
        iconst_1
L006:
        ifeq    L007
        fconst_2
        putstatic       prog2/t F
        goto    L001
L007:
        ldc     3.0
        putstatic       prog2/t F
L001:
```

```
number t = 0;
number i = 0;
if (i < 3) {
      t = 1;
} elseif (i < 5) {
      t = 2;
} else {
      t = 3;
}
```

While Statement

```
        fconst_0
        putstatic     prog2/i F
L001:
        getstatic     prog2/i F
        ldc    3.0
        fcmpg
        iflt   L003
        iconst_0
        goto   L004
L003:
        iconst_1
L004:
        ifeq   L002
        getstatic     prog2/i F
        fconst_1
        fadd
        putstatic     prog2/i F
        goto   L001
L002:
```

```
        number i = 0;
        while (i < 3) {
                i = i + 1;
        }
```

Guard

```
ldc    5.0
putstatic     Test4/x F
iconst_1
putstatic     Test4/boolean Z

getstatic     Test4/x F
ldc    5.0
fcmpg
ifeq   L003
iconst_0
goto   L004
L003:
iconst_1
L004:
ifeq   L002
getstatic     Test4/boolean Z
ifeq   L002

getstatic java/lang/System/out
```

```
number x = 5;
bool boolean = True;
guard(x == 5, boolean){
     display(“Hello World”)
     boolean = False;
     display(“!!!”)
}
```

```
Ljava/io/PrintStream;
ldc "%s\n"
iconst_1
anewarray java/lang/Object
dup
iconst_0
ldc "Hello World"
aastore
invokevirtual
java/io/PrintStream/printf(Ljava/lang/
String;[Ljava/lang/Object;)Ljava/io/Pr
intStream;
pop

iconst_0
putstatic Test4/boolean Z
```

Display

```
        getstatic    java/lang/System/out
Ljava/io/PrintStream;
        ldc    "%s\n"
        iconst_1
        anewarray    java/lang/Object
        dup
        iconst_0
        ldc    "hi"
        new    java/lang/StringBuilder
        dup_x1
        swap
        invokestatic
java/lang/String/valueOf(Ljava/lang/Obje
ct;)Ljava/lang/String;
        invokespecial
java/lang/StringBuilder/<init>(Ljava/lan
g/String;)V
        ldc    5.0
        invokevirtual
java/lang/StringBuilder/append(F)Ljava/l
ang/StringBuilder;
        invokevirtual
java/lang/StringBuilder/toString()Ljava/
lang/String;
```

```
display("hi" + 5);
```

```
      aastore
      invokevirtual
java/io/PrintStream/printf(Ljava/lang/St
ring;[Ljava/lang/Object;)Ljava/io/PrintS
tream;
      pop
```

Definition Call

```
      ldc    8.0
      ldc    "hello"
      invokestatic
prog2/bye(FLjava/lang/String;)Ljava/lang
/String;
      putstatic    prog2/a
Ljava/lang/String;

      ldc    "hey"
      invokestatic
prog2/hey(Ljava/lang/String;)V
```

```
string a = bye(8, "hello");
hey("hey");
```

Definitions (Functions / Procedures)

```
.method private static
bye(FLjava/lang/String;)Ljava/lang/Strin
g;
.var 3 is bye Ljava/lang/String;
.var 2 is t Ljava/lang/String;
.var 1 is y Ljava/lang/String;
.var 0 is z F
      fload 0
      new    java/lang/StringBuilder
      dup_x1
      swap
      invokestatic
java/lang/String/valueOf(F)Ljava/lang/St
ring;
      invokespecial
java/lang/StringBuilder/<init>(Ljava/lan
g/String;)V
      aload 1
      invokevirtual
java/lang/StringBuilder/append(Ljava/lan
g/String;)Ljava/lang/StringBuilder;
      invokevirtual
```

```
def bye (number z, string y : string t)
{
      t = z + y;
}
def hey (string y) {
      display(y);
}
```

```
java/lang/StringBuilder/toString()Ljava/
lang/String;
        astore_2
        aload_2
        astore_3
        aload_3
        areturn
.limit locals 16
.limit stack 16
.end method

.method private static
hey(Ljava/lang/String;)V
.var 0 is y Ljava/lang/String;
        getstatic    java/lang/System/out
Ljava/io/PrintStream;
        ldc    "%s\n"
        iconst_1
        anewarray    java/lang/Object
        dup
        iconst_0
        aload_0
        aastore
        invokevirtual
java/io/PrintStream/printf(Ljava/lang/St
ring;[Ljava/lang/Object;)Ljava/io/PrintS
tream;
        pop
        return
.limit locals 16
.limit stack 16
.end method
```

How to build and run our compiler

Prerequisites:

- A program file written in the AKA language, for example, filename.AKA

Steps:

1. Run the AKA.java file with the following argument parameters: -compile filename.AKA
2. A filename.j file will be generated if the program does not run into any syntactical or grammatical errors.
3. Open a terminal window and execute the following command: ‘java -jar jasmin.jar filename.j’ then run the generated .class file with ‘java classfile’
4. The output will appear in the terminal window.