SOFA

COMPILER CONSTRUCTION

DESCRIPTION

Taking into consideration the amount of widely daily used programming languages in the past years, we have taken as an inspiration the lesser known R language where (as a simple example) you can print out by simply calling .cat.

Hence, by creating the SOFA programming language, we want to let the beginner to intermediate programmers know that whatever inspiration you would have, you could create your own language, express yourself and invent something new anytime, anywhere, whether it would be at a coffee place, at a friend's place, in the park, or just by sitting home on your comfy sofa.

REQUIREMENTS

Name: SOFA

Modifier: sofa

Simple types:

- *String =* ***dust***
- *integer =* ***nop***
- *boolean =* ***spot***

Expression statement:

- ***sofa dust*** *under = "There is dust under the sofa";*
- ***sofa nop*** *pillows = 3;*
- ***sofa spot*** *isDirty = false;*

Declaration statement:

- ***sofa dust*** *under;*
- ***sofa nop*** *pillow;*
- ***sofa spot*** *isDirty;*

Assignment statement:

- *under = "There is dust under the sofa";*
- *pillow = 3;*
- *isDirty = false;*

Natural operations for your types:

- *+*
- *=*
- *-*
- */*
- ***
- *dust.compare*
- *dust.length*

Structured Data Type:

- *sofa* ***list<dust>*** *name = "item", "another item"****;***
- *sofa* ***list<nop>*** *number = 1, 2, 3****;***

Statements for selection and repetition:

- ***if*** *statements* → ***if :*** ***:: then {*** ***; }***
  elseif : ***:: then {*** ***; }***
  else ***{*** ***; }***
  fi;
- ***foreach*** *statement* → ***foreach*** *:* *::* ***do*** *{* *; }* ***fe****;*
- ***while*** *loops* → ***while*** *:* *::* ***do*** *{* *; }* ***od****;*

Subroutines:

- *[access level] [return value] [name] : [parameters]*

  e.g. sofa dust *MethodName* ***: dust*** *name,* ***nop*** *2 { ; }*

Simple console input / output:

- ***tv.input***
- ***tv.output***

  (tv will be our console)

GRAMMAR

Program	*::=* ***SOFA***
SOFA	*::=* ***sofa*** *Declarations* ***AssignOperator*** *value*
Declaration	*::=* ***OneDeclaration****
OneDeclaration	*::=* ***Identifier*** *name*
	| ***sofa Identifier*** *name* ***out*** *Expression;*
	| ***sofa Identifier*** *name* **=>** *Expression;*
	| ***sofa void*** *name :* ***Identifier*** *name* **{** *Expression;* **}**
	| ***sofa void*** *name :* ***Identifier*** *name,* ***Identifier*** *name* **{** *Expression;* **}**
Statements	*::=* ***OneStatement****
OneStatement	*::= Expression*
	| ***if*** *: Expression ::* ***then*** *{ Statements }*
	fi*;*
	| ***if*** *: Expression ::* ***then*** *{ Statements }*
	else *{* ***OneStatement*** *}*
	fi*;*
	| ***if*** *: Expression ::* ***then*** *{ Statements }*
	elseif : *Expression ::* ***then*** **{** *Statements }*
	else *{* ***OneStatement*** *}*
	fi*;*
	| ***while*** *: Expression ::* ***do*** *{ Statements }* ***od;***
	| ***foreach*** *: Expression ::* ***do*** *{ Statements }* ***fe****;*
AssignOperator	*::=* **=**
AddOperator	*::=* **+**
SubOperator	*::=* **-**
MulOperator	*::=* *****
DivOperator	*::=* **/**

TOKENS:

Identifier	*::= dust, nop, spot*
Operator	*::= + | - | * | / | =*
dust	*::=* a | b | … |z | A | B | … | Z
nop	*::=* 0|1|2|3|4|5|6|7|8|9
spot	*::= true | false*

compare

→ *property of dust*

→ *returns spot true if the given objects are the same or spot false if the objects are different*

e.g. *spot isEqual = dust.**compare** name to variable;*

do

→ *part of **do while loops***

e.g.
```
while : true ::
do { tv.output "Hello World"; }
od;
```

→ *part of **foreach statements***

e.g.
```
foreach : dust name in names ::
do { tv.output name; }
fe;
```

else

→ *end part of **if statements***

e.g.
```
if : pillowNumber >= 1 :: then { tv.output "We have pillows"; }
else { tv.output "We need more pillows"; }
fi;
```

elseif

→ *part of **if statements***

→ *situated between **if** and **else** declarations*

e.g.
```
if : pillowNumbers = 2 :: then { tv.output 2; }
elseif : pillowNumber = 5 :: then { tv.output 5; }
elseif :pillowNumbers=10 :: then {tv.output "Too many pillows";}
```

		else { tv.output "No pillows were found"; }
		fi;
fe	→ *closing part for the foreach statements*	
	e.g.	*foreach : dust name in names ::*
		do { tv.output name; }
		fe*;*
fi	→ *ends an **if statement***	
	e.g.	*if : pillowNumbers = 2 :: then { tv.output 2; }*
		elseif : pillowNumber = 5 :: then { tv.output 5; }
		else { tv.output "No pillows were found"; }
		fi;
foreach	→ ***foreach statemet***	
	e.g.	***foreach** : dust name in names ::*
		do { tv.output name; }
		fe;
if	→ ***if statements --** checks the validity of a given statement*	
	e.g.	***if** : pillowNumber = 3 :: then { tv.out pillowNumber; }*
		fi;
input	→ *takes an input through the "tv"*	
	e.g.	*sofa dust newWord = tv.**input;***
length	→ *property of dust--returns an integer of characters that exist in an object*	
	e.g.	*tv.output newWord**.length**;*

list<**identifier**>	*→ creating a list*
	*e.g. sofa **list<**dust**>** listOfStrings = “Hello”, “World”**;***
od	*→ marks the end of the do while loop, followed by a semicolon (“**;**”)*
	*e.g. while : true :: do { tv.output “Hello World”; } **od;***
out	*→ used instead of a return statement*
	→ see also “=>”
	*e.g. sofa dust GetName : dust name { **out** name; }*
output	*→ displays the output through the “tv”*
	*e.g. tv.**output** “Hello World”;*
sofa	*→ access level*
	→ used before declarations of variables and methods
	*e.g. **sofa** dust name;*
then	*→ part of the loops. When the Expression is true, the Statements after **then** keyword is being ran.*
	*e.g. if : pillowNumbers = 2 :: **then** { tv.output 2; } elseif : pillowNumber = 5 :: **then** { tv.output 5; } else { tv.output “No pillows were found”; } fi;*

to

→ part of the compare property

→ check compare token

*e.g. spot isEqual = dust.compare name **to** variable;*

tv

→ equivalent to Console

*e.g. **tv**.output "Hello World";*

void

→ void methods without return statements

e.g.
```
sofa void IncreaseNumberOfPillows : nop pillows
{
        pillowNumber += extraPillows;
}
```

while

*→ keyword used in **do while loops***

e.g.
```
while : true ::
do { tv.output "Hello World"; }
od;
```

:

→ used to declare parameters within methods

*e.g. sofa void Identifier **:** Identifier name, Identifier name { }*

→ used for statements for selection and repetition (if statements, while loops, foreach statements)

e.g.
```
while : true ::
do { tv.output "Hello World"; }
od;
```

::	*→ used in statements for selection and repetition at the end of the Expression*
	*e.g. while : true **::***
	do { tv.output "Hello World"; }
	od;
;	*→ marks the end of an assignment statement or blocks*
{	*→ marks the beginning of a method's body*
}	*→ marks the end of a method's body*
=>	*→ lambda equivalent of return statement*
	*→ same as **out***
	*e.g. sofa dust GetNumberOfPillows **=>** pillowNumber;*

SEMANTIC RULES

1. General rules

1.1 All tokens and keywords in SOFA programming language are written with small letters only.

1.2 There is no use for parenthesis "(" ")".

1.3 Variables and functions must be defined before usage.

1.4 All blocks are ended in a semicolon ";".

2. Classes

2.1 Classes are defined with the "sofa" access level in the beginning followed by the "container" token and a unique identifier.

2.2 [access level] [container] [identifier]

2.3 Objects are created using the "new" keyword followed by the name of the class that the object will be based on.

3. *Variables*

3.1 All variables are defined with "sofa" access level in the beginning, followed by the identifier.

3.2 [access level] [identifier] [name]

3.3 A name defined as a variable cannot be used as a function.

3.4 Variable names cannot start with any other character than a letter.

3.5 Inside the variable name can be used special character "_" or capital letters.

3.6 Variable name cannot end with a special character, but a letter or a number.

4. *Statements for repetition and selection*

4.1 Statements for repetition and selection can only be used within a method's body or the main method.

4.2 Each statement or loop must contain the right tokens. (if, then, elseif, else, while, do, foreach, in, fi, fe, od) in the right order.

5. *Methods*

5.1 Variables declared within a method can only be used inside that method.

5.2 Methods' bodies must be placed between curly brackets "{" "}".

5.3 A parameter to a function is treated as a variable inside the scope of that function.

5.4 A name defined as a method cannot be used as a variable.

5.5 Methods are declared by specifying:

- *The sofa access level*
- *The return value or void if the method has none*
- *The method name*
- *Any method parameters. Method parameters are enclosed in curly brackets and are separated by commas. Empty curly brackets indicate that the method requires no parameters.*

5.6 The method definition specifies the names and types of any parameters that are required.

5.7 When invoking the method, it needs to be provided concrete values, for each parameter that must be compatible with the parameter type.

```
//Class example
tag Hub;
tag Hub.Cluster.Generic;

sofa container MySofa <- IKia {

        sofa nop pillowNumber;
        sofa dust name;
        sofa spot isDirty;

        sofa list<dust> listOfPillow = "Hart", "Kony";


        sofa MySofa : nop pillowNumber, dust name, spot isDirty{
                this.pillowNumber = pillowNumber;
                this.name = name;
                this.isDirty = isDirty;
        }


        IKia void BuyFurniture{}

        IKia nop Purchases : dust personName{}

        sofa void IncreaseNumberOfPillows : nop pillows {
                pillowNumber += extraPillows;
        }

        sofa dust GetNumberOfPillows => pillowNumber;

        sofa dust GetName : dust name{
                out name;
        }

        sofa void ChangeName : dust newName, dust lastName {
                name = newName+lastName;
        }

        sofa void SetIsDirty : spot dirty {
                isDirty = dirty;
        }

        sofa spot isDirty{
                out isDirty;
        }

        sofa spot isDirty{
                get isDirty;
                set isDirty = value;
        }

}

sofa void Main(dust[] args){

        MySofa mySofa = new MySofa : 5,"Ivan",false;

                mySofa.GetName;
                mySofa.ChangeName : "Maria","Ivanich";
}
```