Bilkent University

Department of Computer Engineering

SOUFFLE

…programming language for IoT devices

-Language Design Report-

Project Group Members:

Doğukan KÖSE	ID# 21602375	Section 1
Selen UYSAL	ID# 21702292	Section 1
Meryem Binnaz EFE	ID# 21601779	Section 1

NAME OF LANGUAGE: Souffle

About Souffle

Souffle is a simple language that can control IoT devices. While creating this language, we tried to focus on three main specialties which are readability, writability, and reliability.

Readability

Our programming language has English terms since it is global language. Furthermore, the most of the reserved words are similar with the other languages so that it is easy to understand and analyze the code. Also, some structures are more readable from the other languages. For example, we changed for loop.

Writability

While creating the Souffle, we tried to make it easily writable. In our language, we have not put any reserved word which is beginning with uppercase, because we thought that upper case reserved words decrease writability.

Reliability

While planning the Souffle, we also think some details about its reliability, because the language must be reliable in order to be used. Therefore, we tried to be careful about structure of the language.

BNF of Souffle

<program> → <main>

|<function list> <main>

<function list> → <function declaration>

|<function list> <function declaration>

<main> → go{<statements>} stop;

<statements> → <statement>

| <statement> <statements>

<statement> → <matched> | <unmatched>

<matched list> → <matched list> <matched>

| <matched>

<matched> → if (<logic expr>) <matched> else <matched>

| if (<logic expr>) {<matched list>} else {<matched list>}

| if (<logic expr>) <matched> else {<matched list>}

| if (<logic expr>) {<matched list>} else <matched>

|<non if statements>

<unmatched list> → <unmatched list> <unmatched>

| <unmatched>

<unmatched> → if (<logic expr>) <statement>

| if (<logic expr>) <matched> else <unmatched>

| if (<logic expr>) {<matched list>} else <unmatched>

| if (<logic expr>) {<unmatched list>} else statement

| if (<logic expr>) {<unmatched list>} else {<unmatched list>}

| if (<logic expr>) {<matched list>} else {<unmatched list>}

| if (<logic expr>) {<unmatched list>} else {<matched list>}

<non if statements> → <loops>

| <input>;

| <output>;

| <function call>;

| <declaration>;

| <dec assign>;

| <assignment>;

|<primitive function call>;

|<comment>

<loops> → <for loop> | <while loop>

<for loop> → loop <loop data type> <identifier> from <const number> to <const number> by <arithmetic op> <const number> {<statements>?}

<while loop> → until (<logic expr>) {<statements>?}

<function declaration> → <function header> <function body>

<function header> → <data type> <identifier> (<parameter list>?)

<function body> → {<statements>? return <expression>;}

| {<statements>?}

| {return <expression>;}

<function call> → <identifier> (<expression list>?)

<primitive function call> → readTemp()

| readHum()

|readAirPress()

|readAirQuality()

|readLight()

|readSoundLevel()

| send (<identifier>, <const number>)

| send (<identifier>, <identifier>)

| receive (<identifier>)

| receive(<url const>)

| turnOnSwitch(<expression>)

| turnOffSwitch(<expression>)

| connect(<identifier>)

| connect(<url const>)

<declaration> → <data type> <identifier list>

<expression> → <logic expr>

| <arithmetic expression>

| <string const>

| <url const>

<expression list> →

| <expression>

|<expression>, <expression list>

<arithmetic expression> → <arithmetic expression> + <term>

|<arithmetic expression> - <term>

|<term>

<term> → <term> * <factor>

| <term> / <factor>

| <term> % <factor>

| <factor>

<factor> → (<arithmetic expression>)

| <identifier>

| <const number>

| <function call>

| <primitive function call>

<identifier list> →

| <identifier>

| <identifier>, <identifier list>

<parameter list> →

|<parameter>

| <parameter list>, <parameter>

<parameter> → <data type> <identifier>

<assignment> → <postincrement expr>

|<postdecrement expr>

| <identifier> <assignment op> <expression>

<dec_assign> → <data type> <identifier> <assignment op> <expression>

<postincrement expr> → <identifier> <increment op>

<postdecrement expr> → <identifier> <decrement op>

<data type> → int

| double

| string

| boolean

| conn

| time

| url

| void

<loop_data_type> → int

| double

<const number> → <integer const>

| <double const>

| <negative number>

<arithmetic op> → +

| -

| *

| /

<logic expr> → <logic expr> <logic op> <comp expr>

| <logic expr> <logic op> <sub logic expr>

| <comp expr>

| <sub logic expr>

<sub logic expr> → <boolean>

| (<logic expr>)

<comp expr>→ <comp item> <comp op> <comp item>

<comp item> → <arithmetic expression>

| <boolean>

<logic op> → <or>

| <and>

<comp op> →<equal comp>

| <non equal comp>

| <less comp>

| <greater comp>

| <less equal comp>

| <greater equal comp>

<input> → scan (<identifier>)

<output> → print (<expression list>)

<negative number> → MINUS_SIGN INTEGER_CONST

| MINUS_SIGN DOUBLE_CONST

<boolean> → true|false

<comment> → #

|#* *#

<increment op> → ++

<decrement op> → --

Explanation Part

<program>: Program is the combination of functions and main function

<function list>: It is a list of functions

<main>: It is the main function, it starts with go token and left curly bracket, then it takes statements, lastly it stops with right curly bracket stop semicolon tokens.

<statements>: Combination of multiple statements or just one statement.

<statement>: Statement consists of unmatched and matched statements in order to get rid of unambiguous grammar of if-else structure.

<matched>: Matched indicates that there are pairs of if-else statements and it also includes non-if-statements. Because, non-if-statements are also balanced in terms of if-else.

<unmatched>: Unmatched indicates that there is unpaired if-else structure in the code.

(We know that matched and unmatched is a little bit longer but we tried to get rid of ambiguous structure of if-else which is written in book that's why we tried to put there all combinations of if-else structure)

<unmatched list>: List of unmatched statements

<matched list>: List of matched statements

<non if statements>: Non if statements include loops, input, output function call, declaration, dec assign, assignment, primitive function call, and assignment.

<loops>: It is combination of two iteration type which are loop and while loop.

<for loop>: For loop is start with "loop" then it takes data type of variable and takes an identifier. Then, it takes two parameter and it iterates according to given constant number from first parameter to second parameter. An example of loop of our code is "loop int i from 5 to 100 by +2". So, it iterates from 5 to 100 by adding 2 to 5.

<while loop>: While loop starts with “until”, then it takes a logic expression inside parenthesis. Then, it iterates all the statements inside the curly parenthesis.

<function declaration>: It declares a function. Function declaration consists of two parts. First part is function header and the second part of function declaration is function body.

<function header>: Function header takes data type and identifier. It also takes parameter list inside parenthesis.

<function body>: Function body includes statements into curly parenthesis and one return statement to exit from function.

<function call>: It needs one identifier and parameter list inside the parenthesis in order to call a function.

<primitive function call>: It includes functions that are already specified by program such as readSensor().

<declaration>: It is for variable declaration and it needs data type of variable and identifier list

<expression>: It consists of expressions that are arithmetic expression, logic expression, string_const, and url_const.

<arithmetic expression>: Arithmetic expressions includes addition, subtraction, multiplication and division.

<term>: This construct is a sub element of arithmetic expression.

<factor>: Factor is a sub element of term construct. It includes variables and constant numbers.

<identifier list>: List of identifiers

<parameter list>: It is set of parameters that are separated by comma.

<parameter>: A parameter consists of data type and variable

<assignment>: It assign expression to a variable. It needs variable, then left assign op and then expression.

<dec assign>: It assign expression to a variable when variable is declared.

<postincrement expr>: Post increment expr increase the value of variable by 1

<postdecrement expr>: Post decrement expr decrease the value of variable by 1

<identifier>: It is a name of functions or variables. It needs a letter then it can get letter or digit.

<data type>: It specify the type of variable. It can be int, double, boolean or string.

<loop data type>: It consists integer and double data type

<const number>: It includes integer_const double_const and negative_numbers

<assign op>: This construct is the assignment operator.

<arithmetic op>: List of 4 arithmetic operator

<logic expr>: This construction can be boolean or it can compare one expression with another.

<sub logic expr>: It is an sub logic expression for predecence of parenthesis in logical expressions

<comp expr>: This expression is contained from logic expression and it compares two comp item with comp operations

<comp item>: Comp items includes arithmetic expressions, booleans, string constant and url constant

<logic op>: List of logical operators (and, or)

<comp op>: List of comparator operators

<input>: This structure lets users to input.

<output>: Output is a print function and it can print expressions.

<negative number>: Negative number contains negative integer and negative double constant numbers

<boolean>: Boolean can be true or false.

<comment>: Comment contains single line comment or block comment.

TOKENS

GO STOP

READ_TEMP READ_HUM READ_AIR_PRESS READ_AIR_QUALITY READ_LIGHT READ_SOUND_LEVEL SEND RECEIVE TURN_ON_SWITCH TURN_OFF_SWITCH CONNECT

STRING INTEGER DOUBLE BOOLEAN CONN TIME URL VOID

LOOP FROM TO BY UNTIL

PRINT SCAN

IF ELIF ELSE

RETURN

PLUS_SIGN MINUS_SIGN DIV_SIGN MOD_SIGN MULT_SIGN

INCREMENT_OP DECREMENT_OP

OR AND

ASSIGMENT_OP

EQUAL_COMP LESS_COMP GREATER_COMP LESS_EQUAL_COMP GREATER_EQUAL_COMP NON_EQUAL_COMP

SEMICOLON DOT COMMA NL

END_BLOCK_COMMENT BLOCK_COMMENT SINGLE_COMMENT

LEFT_SQ_BRAC RIGHT_SQ_BRAC LEFT_CR_BRAC RIGHT_CR_BRAC LEFT_PARAN RIGHT_PARAN

STRING_CONST IDENTIFIER INTEGER_CONST DOUBLE_CONST TRUE FALSE URL_CONST