CS 315 Programming Languages
Project 1 Report
Mamba Language Design for IoT Devices and its Lexical Analyzer

Hazal Buluş (21903435) Section 01
Süleyman Gökhan Tekin (21902512) Section 01
Kürşad Güzelkaya (21902103) Section 02

1) INTRODUCTION

This report contains features of the language Mamba that we created considering the readability, writability, and reliability criteria. The Mamba language was designed to be used for IoT nodes. In the rest of the report, BNF grammar explanations of our language are written. At the end of the report, there is a list of reserved words in the language.

2) BNF DESCRIPTION

Program

<program> ::= @BEGIN <stmt_list> @END

<stmt_list> ::= <stmt> | <stmt><stmt_list> | <comment> | <comment><stmt_list>

<comment> ::= /* <str_value> */

Statements

<stmt> ::= <if_stmt> | <comment> | <declaration> | <assignment> |
<declaration_assignment> | <loop_stmt> | <call_function> | <expression>

<declaration> ::= <type_name><identifier>

<declaration_assignment> ::= <declaration_assignment_const> |
<declaration><assign_op><expression>

<declaration_assignment_const> ::= const <declaration><assign_op><expression>

<call_function> ::= <identifier>() | <identifier>(<arguments>)

<loop_stmt> ::= <for> | <while>

<assignment> ::= <identifier><assign_op><expression> |
<identifier><assign_op><call_function>

<case_special_call_functions> ::= ADD_SENSOR(<str>)
| CHANGE_SENSOR_NAME(<str>, <str>)
| READ_SENSOR(<str>)
| REMOVE_SENSOR(<str>)

| SEE_ALL_SENSORS()
| GET_CURR_TIMESTAMP()
| SET_SWITCH_ON(<int>)
| SET_SWITCH_OFF(<int>)
| GET_SWITCH_STATUS(<int>)
| CREATE_URL_CONNECTION(<str>)
| DISCONNECT_URL(<str>)
| EDIT_CONNECTION_URL(<str>, <str>)
| RESET_ALL_CONNECTIONS()
| SEE_ALL_CONNECTIONS()
| SEND_DATA(<str>, <int>)
| RECIEVE_DATA(<str>)

Expressions

<expression> ::= <identifier> | <any_type> | <arithm_expression> |
<cond_expression> | <operation_expression> | <call_function>

<arithm_expression> ::= <arithm_expression> + <term> |
<arithm_expression> - <term> |
<term>

<term> ::= <term> * <factor> | <term> / <factor> | <factor>

<factor> ::= (<arithm_expression>) | <factor> ^ <factor> | <factor> % <factor> |
<identifier> | <any_type>

<cond_expression> ::= <bool> | <any_type><relate_op><any_type>

<operation_expression> ::= <increment> | <decrement>

<increment> ::= ++<identifier> | <identifier>++

<decrement> ::= --<identifier> | <identifier>--

Functions

<non_void_func> ::= <type_name> <identifier>(<params>)
{<stmt_list> return (<return_stmt>)}

<void_func> ::= void <identifier>(<params>) {<stmt_list>}

<params> ::= <space> | <param> | <param>, <params>

<param> ::= <declaration>

<arguments> ::= <identifier_list> | <identifier_list>, <arguments> |
<any_type> | <any_type>, <arguments>

<return_stmt> ::= <expression> | <return_stmt><arithm_op><expression>

If Statements

<if_stmt> ::= <matched> | <unmatched>

<unmatched> ::= if (<cond_expression>) {<stmt_list>} | if (<cond_expression>) {<matched>} else {<unmatched>} | <cond_expression> ? <assignment> : <assignment>

<matched> ::= if (<cond_expression>) {<matched>} else {<matched>} | <non_if_stmt>

<non_if_stmt> ::= <stmt>;<non_if_stmt> | <stmt>;

Loops

<while> ::= while (<cond_ expression>) {<stmt_list>}

<for> ::= for (<for_start>:<cond_expression>:<for_operation>) {<stmt_list>}

<for_start> ::= <for_start>, <assignment> | <for_start>, <declaration> | <for_start>, <declaration_assignment> | <declaration_assignment> | <assignment> | <declaration>

<for_operation> ::= <assignment> | <operation_expression>

Operators

<relate_op> ::= == | != | < | <= | !< | > | >= | !> | AND | OR | !AND | !OR

<assign_op> ::= = | += | -= | *= | /= | %=

<arithm_op> ::= + | - | * | / | %

<set_op> ::= ++ | --

Symbols

<new_line> ::= \n

<char_letter> ::= [a-zA-Z]

<char_special> ::= ! | @ | # | \$ | % | ^ | & | * | (|) | + | = | / | * | - | ' | " | ; | '|' | { | } | [|] | . | ,

<digit> ::= [0-9]

<true> ::= true | 1

<false> ::= false | 0

<space> ::= “ ”

Types

<type_name> ::= int | string | bool | float | char | ptr

<any_type> ::= <int> | <str> | <bool> | <float> | <char>

<bool> ::= <true> | <false>

<digits> ::= <digit> | <digit><digits>

<int> ::= <positive_int> | <negative_int>

<positive_int> ::= <digits> | +<digit> | +<digits>

<negative_int> ::= -<digits>

<char_value> ::= <char_letter> | <char_special> | <digit>

<char> ::= ‘<char_value>’ | ‘ ’

<str_value> ::= <char> | <space> | <char><str_value> | <str_value><char>

<str> ::= “<str_value>” | “ ”

<float> ::= .<digits> |<int>.<digits>

<identifier_list> ::= <identifier> | <identifier>,<identifier_list>

<identifier> ::= <char_letter> | <char_letter><identifier> | <identifier><digit> | <identifier><char_letter> | <identifier>_<identifier>

3) EXPLANATION

<program> ::= @BEGIN <stmt_list> @END

This state is made of a statement list which is a combination of statements. It is the initial state to start a program. Users have to start typing with a special reserved word @BEGIN and end with the special reserved word @END.

<stmt_list> ::= <stmt> | <stmt><stmt_list> | <comment> | <comment><stmt_list>

Statement list can be a single statement, comment and a combination of statement, statement lists.

<comment> ::= /* <str_value> */

To write any comment, users have to type the comment between the special characters /* example comment */

<stmt> ::= <if_stmt> | <comment> | <declaration> | <assignment> | <declaration_assignment> | <loop_stmt> | <call_function> | <expression>

Statement may consist of lots of statement types including if statement, comment, declaration statement, assignment statement, declaration assignment statement, for loop, while loop, function call, or any expression.

<declaration> ::= <type_name><identifier>

Declaration simply declares various types such as bool, int, str and all the other types. These type names are reserved words in the Mamba language.

<declaration_assignment> ::= <declaration_assignment_const> | <declaration><assign_op><expression>

Declaration assignment statement is like the combination of declaration statement and assignment statement. It declares a normal and constant type and at the same time assigns a value to the declared identifier.

<declaration_assignment_const> ::= const <declaration><assign_op><expression>

Constant declaration assignment declares and assigns a not changeable value to the identifier that has reserved word const before its type name.

<call_function> ::= <identifier>() | <identifier>(<arguments>)

This structure is used for using functions at any place in the program by writing the identifier of the function and parentheses after it. The user must write arguments inside the parentheses if that function needs parameters.

<loop_stmt> ::= <for> | <while>

Loop statement only includes two types of loops in the Mamba language. These loops are for loop and while loop.

<assignment> ::= <identifier><assign_op><expression> | <identifier><assign_op><call_function>

Assignment expression assigns an expression or an output of a function to the beforehand declared identifier.

<type_name> ::= int | string | bool | float | char | ptr

The type name non terminal is used to provide the type of the data that will be stored in the identifier.

<any_type> ::= <int> | <str> | <bool> | <float> | <char>

Any type contains all types as int, string, bool, float, char.

<bool> ::= <true> | <false>

Bool non terminal is the type used for true and false.

<digits> ::= <digit> | <digit><digits>

Digits type represents any single digit or more than one digit.

<int> ::= <positive_int> | <negative_int>

Integer type consists of two types of integer, positive and negative.

<positive_int> ::= <digits> | +<digits>

Positive integer consists of digits having plus sign before the digits or just digits itself without any sign.

<negative_int> ::= -<digits>

Negative integer type is the digits having minus sign before the digits.

<char_value> ::= <char_letter> | <char_special> | <digit>

Char value can be any single letter character, special character or any single digit.

<char> ::= '<char_value>' | ' '

Char type name is used to represent any char value type or just an empty space.

<str_value> ::= <char> | <space> | <char><str_value> | <str_value><char>

String value type represents any char type, empty space, or the combinations of char types and empty spaces.

<str> ::= "<str_value>" | " "
String type is used to represent the string values or empty strings.

<float> ::= .<digits> |<int>.<digits>

Floating numbers are not integer type numbers, can be positive and negative such as .9, 7.8, -5.7 etc.

<identifier_list> ::= <identifier> | <identifier>,<identifier_list>

Identifier list contains just a single identifier or a combination of identifiers.

<identifier> ::= <char_letter> | <char_letter><identifier> | <identifier><digit> |<identifier><char_letter> | <identifier>_<identifier>

A single identifier is used to represent char letters, digits or a combination of char letters and digits.

<expression> ::= <identifier> | <any_type> | <arithm_expression> | <cond_expression> | <operation_expression> | <call_function>

Expression is a common statement used to represent either an identifier, or any type name, or an arithmetic expression, or a conditional expression, or an operation expression, or function call expression.

<arithm_expression> ::= <arithm_expression> + <term> |
<arithm_expression> - <term> |
<term>

Arithmetic expression represents summation and subtraction operations. For the priority of different arithmetic operations, the multiplication, division, exponential and modular arithmetic operations separated into other expressions.

<term> ::= <term> * <factor> | <term> / <factor> | <factor>

Term is simply a multiplication operation or arithmetic operation or just a factor.

<factor> ::= (<arithm_expression>) | <factor> ^ <factor> | <factor> % <factor> | <identifier> | <any_type>

Factor represents an arithmetic operation or modular arithmetic operation or an exponentiation operation.

<cond_expression> ::= <bool> | <any_type><relate_op><any_type>

Conditional expression may be a bool type which has two outcomes as true and false, or a relation operation of any two types of variable, which again outcomes as true or false.

<operation_expression> ::= <increment> | <decrement>

Operation expression consists of two types of expressions: increment and decrement. It either used to decrease or increase the value of a variable.

<increment> ::= ++<identifier> | <identifier>++

Increment expression is used to increase the value of an identifier by one by putting two plus signs before or after the identifier name.

<decrement> ::= --<identifier> | <identifier>--

Decrement expression is used to decrease the value of an identifier by one by putting two minus signs before or after the identifier name.

<while> ::= while (<cond_ expression>) {<stmt_list>}

While is a loop statement that starts with the reserved word while, it has to contain a conditional expression inside parentheses and a statement list in a special format, curly braces.

<for> ::= for (<for_start>:<cond_expression>:<for_operation>) {<stmt_list>}

For is a loop statement starting with the reserved word for, then inside the parentheses, first there is an assignment operation list, and then the conditional expression is declared, and an operation that can be done in for loop as assignment or increment/decrement. Inside curly braces, statement lists show up.

<for_start> ::= <for_start>, <assignment> | <for_start>, <declaration> | <for_start>, <declaration_assignment> | <declaration_assignment> | <assignment> | <declaration>

For start is used for representing a single assignment expression or list of expressions, a single declaration expression or list of declarations and a single declaration assignment expression or list of declaration assignment expressions.

<for_operation> ::= <assignment> | <operation_expression>

For operation can be two types of operations: either an assignment operation or operation expression that consists of decrement or increment statements.

<if_stmt> ::= <matched> | <unmatched>

If statement consists of matched or unmatched if statements.

<unmatched> ::= if (<cond_expression>) {<stmt_list>} | if (<cond_expression>) {<matched>} else {<unmatched>} | <cond_expression> ? <assignment> : <assignment>

Unmatched expression is basically an expression having a greater number of if statements than else statements. It can be just a single if statement not followed by any else statement, or just a combination of multiple if and else statements.

<matched> ::= if (<cond_expression>) {<matched>} else {<matched>} | <non_if_stmt>

Matched expression can be just any non if statement or, it can contain if statements followed by else statements afterwards. In matched expressions, either there are no if and else statements or the if statement and else statements are in equal numbers.

<non_if_stmt> ::= <stmt>;<non_if_stmt> | <stmt>;

A non if statement is a statement that does not contain any if,else,matched or unmatched expressions. It consists either of a single regular statement, or the combinations of regular non if statements.

<non_void_func> ::= <type_name> <identifier>(<params>)
{<stmt_list> return (<return_stmt>)}

A structure that represents a function that returns a type such as int, bool, string, float or char.

<void_func> ::= void <identifier>(<params>) {<stmt_list>}

A structure that represents a function that contains a statement list and returns nothing (void).

<params> ::= <space> | <param> | <param>, <params>

Parameters can be just an empty space, a single parameter or parameter followed by other parameters.

<param> ::= <declaration>

Parameter consists of declaration expressions.

<arguments> ::= <identifier_list> | <identifier_list>, <arguments> |
<any_type> | <any_type>, <arguments>

Arguments represent an identifier list or more than one identifier list, or any type variable, or combination of these.

<return_stmt> ::= <expression> | <return_stmt><arithm_op><expression>

Return statement is simply an expression or an arithmetic operation of a return statement and an expression.

<new_line> ::= \n

Simply represents a new line.

<char_letter> ::= [a-zA-Z]

Letter character represents all uppercase and lowercase letters in the English alphabet.

<char_special> ::= ! | @ | # | \$ | % | ^ | & | * | (|) | + | = | / | * | - | ' | " | ; | '|' | { | } | [|] | . | ,

Char special consist of all possible special character symbols that are not letters in alphabet or underscore or any number.

<digit> ::= [0-9]

Digit represents integer numbers from 0 to 9.

<true> ::= true | 1

True represents boolean type true or 1 as value.

<false> ::= false | 0

False represents boolean type false or 0 as value.

<space> ::= “ ”

Space represents an empty space in the program.

<relate_op> ::= === | !== | < | <= | !< | > | >= | !> | AND | OR | !AND | !OR

Relation operators in Mamba language are if equal, if not equal, greater, greater or equal, not, not greater, less than, less than or equal, not less than, and logical operators as and, or, not and, not or.

<assign_op> ::= = | += | -= | *= | /= | %=

Assign operators are used for assignment operations such as equals, or assignment arithmetic operators add equals, subtract equals, divide equals and mod equals.

<arithm_op> ::= + | - | * | / | %

Arithmetic operator is used to represent five arithmetic operations in the language Mamba as summation, subtraction, multiplication, division and modular arithmetic operation.

<set_op> ::= ++ | --

Set operators are used to increment or decrement the value by one.

<case_special_call_functions> ::= ADD_SENSOR(<str>)
| CHANGE_SENSOR_NAME(<str>, <str>)
| READ_SENSOR(<str>)
| REMOVE_SENSOR(<str>)
| SEE_ALL_SENSORS()
| SET_SWITCH_ON(<int>)
| SET_SWITCH_OFF(<int>)
| GET_SWITCH_STATUS(<int>)
| CREATE_URL_CONNECTION(<str>)
| EDIT_CONNECTION_URL(<str>, <str>)
| DISCONNECT_URL(<str>)
| SEND_DATA(<str>, <int>)
| RECIEVE_DATA(<str>)
| GET_CURR_TIMESTAMP()
| SEE_ALL_CONNECTIONS()
| RESET_ALL_CONNECTIONS()

The programming language Mamba is designed to program the IoT nodes produced by the hardware department. So, the Mamba language has some special functions to get things done faster.

ADD_SENSOR(str sensorName) function adds a sensor according to the given name by string as parameter ,

CHANGE_SENSOR_NAME(str oldSensorName, str newSensorName) function changes the name of a sensor that is already existing ,

READ_SENSOR(str sensorName) function reads the value on the sensor that the user indicated in the parameter,

REMOVE_SENSOR(str sensorName) function removes a sensor with the given name in the parameter from the system,

SEE_ALL_SENSORS() function shows all of the sensors in the system,

SET_SWITCH_ON(int switchNo) function turns on the switch with the given id by parameter,

SET_SWITCH_OFF(int switchNO) function turns off the switch with the given id by parameter,

GET_SWITCH_STATUS(int switchNO) function gets the status of the switch with the given id by parameter.

CREATE_URL_CONNECTION(str URLName) function creates an URL connection with the string given by the parameter,

EDIT_CONNECTION_URL(str URLName) function changes the connection URL with the given string by parameter.

DISCONNECT_URL(str URLName) function disconnects the URL connection with the string given by the parameter,

SEND_DATA(str URLName, int data) function sends data as an integer given by parameters to the given URL as a string given by the parameter,

RECIEVE_DATA(str URLName) function receives the data from the URL with the string given by the parameter,

GET_CURR_TIMESTAMP() function returns the current timestamp, the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970, not counting leap seconds, and

SEE_ALL_CONNECTIONS() function shows all the connection in the system to the user.

RESET_ALL_CONNECTIONS() function removes all of the connections in the system.

Reserved Words in Mamba

@BEGIN starts the program

@END ends the program

for for loop

if if statement

else used for else statements

while while loop

true represents true value of a boolean type

false represents false value of a boolean type

return states return statements

const constant variables

int represents integer type of numbers

bool represents the type of boolean

float represents floating numbers

char character type

ptr pointer type

AND logical and operator

OR logical or operator

str represents string type of variables

4) EVALUATION

a - Readability

Mamba language is carefully designed by considering the prioritization of readability. Our first goal is for users who are not computer engineers to understand our language after looking at it for a few minutes. We assumed that the users had taken at least a few simple programming courses when designing this language so we tried to make it similar to imperative languages, which is still hard to understand by people far from software and programming. We consider the overall simplicity while designing the language. In the language, there are some feature multiplicities - such as instead of “a= a+1” the expressions a++ and ++a. Although this increases the writability of the language and decreases the readability of the language a little, we have created a language of fairly low complexity. The language has a manageable set of features and constructs. Data types are basic and very simple to use. Additionally, the syntax of the language includes simple reserved and special words. Although the special functions we have written for the case seem complicated, we have written the explanation of all of them, including all expressions and statements in the language, in an understandable way. By the explanations and understandable names, we tried to keep the readability rate of our language at the highest level.

b - Writability

While designing Mamba language, with the aim of increasing writability, we added feature multiplicities for all five arithmetic operations and make assignment operators (+=, -=, *= /= %=). In addition to feature multiplicity, we kept the syntax and name of data types very basic for making the language easily writable. In order to keep punctuation simple, we used a minimum level of punctuation in the language we

designed, as we thought that using a semicolon at the end of each line would make it difficult to write and compile, like java, which is the most used language among imperative languages.

c - Reliability

There is currently no type checking feature in our language. Likewise, there is no exception handling mechanism. These two shortcomings reduce the reliability of our programming language. On the other hand, since the writability and readability of the programming language greatly affect its reliability, paying attention to its readability and writability while designing our language also increases the reliability of the language to a certain extent. A programming language is simpler to develop and alter the easier it is to read. Similar to this, a programming language's writability raises its success rate.