Bilkent University

CS 315 Programming Languages Project Final

BAM Programming Language for IoT Devices

Barış Tan Ünal | 22003617
Ahmet Şahin | 21902702
Mert Ünlü | 22003747
(GROUP #09)

28.10.2022

1. BNF of the BAM Language

1.1 Program

```
<program>          →    BEGIN <stmt_list> END

<stmt_list>        →    <stmt> SC
                        | <stmt_list> <stmt> SC
```

1.2 Statements

```
<stmt>             →    <if_stmt>
                        | <while_loop>
                        | <for_loop>
                        | <id_decleration>
                        | <const_id_decleration>
                        | <assign_stmt>
                        | <increment_stmt>
                        | <decrement_stmt>
                        | <return_stmt>
                        | <io>
                        | <function_stmt>
                        | <comment>
                        | <con_sw_stmt>
```

1.3 Loops

```
<while_loop>       →    WHILE LP <cond_expr> RP
                        LB <stmt_list> RB

<for_loop>         →    FOR LP <id_decleration> SC <cond_expr>
                        SC <assign_stmt> RP LB <stmt_list> RB
                        | FOR LP <id_decleration> SC <cond_expr>
                        SC <increment_stmt> RP LB <stmt_list> RB
                        | FOR LP <id_decleration> SC <cond_expr>
                        SC <decrement_stmt> RP LB <stmt_list> RB
```

1.4 Types

```
<primitive_type>   →    int
                        | float
                        | string
                        | char
                        | boolean

<literal>          →    <int>
                        | <float>
                        | <char>
                        | <string>
                        | <boolean>
```

```
<int>                   →   <digit>
                            | <int> <digit>

<float>                 →   <int> DOT <int>
                            | DOT <int>

<char>                  →   SQT <letter> SQT
                            | SQT WHITE_SPACE SQT

<char_list>             →   <alphanumeric>
                            | <char_list> <alphanumeric>
                            | WHITE_SPACE
                            | <char_list> WHITE_SPACE
                            | <symbol>
                            | <char_list> <symbol>

<string>                →   DQT <char_list> DQT

<boolean>               →   TRUE
                            | FALSE
```

1.5 Expressions

```
<expression>            →   <arith_expr>

                            | <cond_expr>

                            | STRING

                            | CHAR

<increment_stmt>        →   <identifier> INCREMENT

<decrement_stmt>        →   <identifier> DECREMENT

<arith_expr>            →   <arith_expr> PLUS <term>
                            | <arith_expr> MINUS <term>
                            | <term>

 <term>                 →   <term> MUL <factor>
                            | <term> DIV <factor>
                            | <term> MOD <factor>
                            | <factor>

<factor>                →   LP <arith_expr> RP
                            | <identifier>
                            | <int>
                            | <float>
```

```
<cond_expr>                  →    <cond_expr> OR_OP
                                  <cond_expr_and>
                                  | <cond_expr_and>

<cond_expr_and>              →    <cond_expr_and> AND_OP
                                  <cond_expr_element>
                                  | <cond_expr_element>

<cond_expr_element>          →    LP <cond_expr> RP
                                  | <relational_expr>
                                  | <func_call>
                                  | BOOLEAN
                                  | NOT_OP <cond_expr>

<relational_expr>            →    <relational_expr_element>
                                  <relational_op>
                                  <relational_expr_element>

<relational_expr_element>    →    <int>
                                  | <float>
                                  | <identifier>
                                  | <func_call>
                                  | <boolean>
```

1.6 Operands

```
<relational_op>         →    LT
                             | LEQ
                             | GT
                             | GEQ
                             | EE
                             | NEQ
```

1.7 Variables

```
<identifier>            →    <letter>
                             | <identifier> <alphanumeric>
                             | <identifier> UNDER_SCORE
```

1.8 Declaration and Assignment

```
<id_decleration>        →    <primitive_type> <identifier>
                             ASSIGN_OP <expression>

<const_id_decleration>  →    BAMCONST <primitive_type> <identifier>
                             ASSIGN_OP <expression>
<assign_stmt>           →    <identifier> ASSIGN_OP <expression>
```

1.9 Connections, Sensors and Switches

```
<con_sw_stmt>          →    <connection_stmt>
                            | <switch_stmt>
                            | <sensor_select>

<connection_stmt>      →    <connection> <ARROW_OP> <URL>

<URL>                  →    HASHTAG <char_list>

<connection>           →    AT <char_list>

<sensor>               →    DOLLAR_SIGN <int>

<sensor_select>        →    BAMSEL LP <sensor> RP

<switch_stmt>          →    SW* ASSIGN_OP <boolean>
```

1.10 Input Output

```
<io>                   →    <input_stmt>
                            | <output_stmt>

<input_stmt>           →    <connection_input>
                            | <url_input>
                            | <timer_input>
                            | <sensor_input>

<connection_input>     →    BAMCIN LP <identifier> COMMA
                            <connection> RP

<url_input>            →    BAMUIN LP <identifier> COMMA <url> RP

<timer_input>          →    BAMTIN LP <identifier> COMMA TIMER RP

<sensor_input>         →    BAMSIN LP <identifier> COMMA SENSOR RP

<output_stmt>          →    <connection_output>
                            | <url_output>

<connection_output>    →    BAMCOUT LP <int> COMMA <connection> RP

<url_output>           →    BAMUOUT LP <int> COMMA <url> RP
```

1.11 Comments

```
<comment>              →    DIV DIV <char_list> DIV DIV
```

1.12 If Statements

```
<if_stmt>              →    IF LP <cond_expr> RP LB <stmt_list> RB
                            | IF LP <cond_expr> RP LB <stmt_list> RB
                            ELSE LB <stmt_list> RB
```

1.12 Function Declarations and Calls

```
<function_stmt>        →    <func_dec>
                            | <func_call>

<func_dec>             →    FUNCT <identifier> LP <parameter_list>
                            RP LB <stmt_list> RB
                            | FUNCT <identifier> LP RP LB <stmt_list> RB

<parameter_list>       →    <parameter_list> COMMA <identifier>
                            | <identifier>

<func_call>            →    BAMCALL <identifier> LP <parameter_list> RP

<return_stmt>          →    RETURN <expression>
```

1.14 Symbols and Alphanumerics

```
<symbol>               →    NOT_OP | POW | PLUS | INCREMENT
                            | AND_OP | OR_OP
                            | DIV | LP | RP | ASSIGN_OP | EE | NEQ
                            | MINUS | DECREMENT | MOD
                            | GT | GEQ | LT | LEQ | LB | LSB | RSB | RB
                            | MUL | DOT | COMMA | UNDER_SCORE | AT

<alphanumeric>         →    <alphanumeric> <letter>
                            | <alphanumeric> <digit>
                            | <letter>
                            | <digit>
```

1.15 Elementary Components for Literals and Identifiers

```
<letter>    →    a | b | c | d | e | f | g | h | i | j | k | l | m | n
                 | o | p | q | r | s | t | u | v | w | y | x | z | A | B
                 | C | D | E | F | G | H | I | J | K | L | M | N | O | P
                 | Q | R | S | T | U | V | W | Y | X | Z

<digit>     →    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

2. Description of Non-Terminals

<program>

The program non-terminal is the root of each code block. It includes a statement list. This is a conventional name.

<stmt_list>

Statement list consists of statements separated by semicolons. It has left associativity, therefore left recursion.

<stmt>

Statement could be any line or block of code that is associated within itself. This could be any one of a conditional block, loop block, a variable declaration, function call, function declaration, input-output operation or return statement.

<if_stmt>

This is a default conditional statement. There are two possibilities of conditional statements: standalone if and if-else block. Both if and if-else statements are included in this non-terminal. There is no possibility of ambiguity, therefore need for matched and unmatched versions since BAM language encourages the programmer to use curly brackets.

<while_loop>

This is a control flow statement. While loop continues to execute the statements given to it as long as the given boolean condition is true. When the conditions do not hold, the program stops iterating.

<for_loop>

This is a control flow statement. After stating the identifier, the user creates a condition and a statement inside the parentheses. As long as the condition holds, the for loop performs the code given to it repeatedly.

<id_decleration>

This is a variable declaration statement. The language does not allow the programmer to just declare the variable without assigning a value to it.

<const_id_decleration>

This is a variable declaration statement. The value of const variable should be initialised in the declaration. BAMCONST token is used to identify const variables.

<assign_stmt>

In BAM language, this statement type is used for assigning values to variables. Left hand side could be any variable and the right hand side can be a literal, expression or a function call.

<URL>

In BAM language, this non-terminal is used for symbolising URL links. In our language, a URL link consists of a hashtag sign followed by an arithmetic expression.

<connection_stmt>

Connection statement is a special operation which is only used when a URL is required to be linked with a specific CONNECTION. Arrow operator is used in between to assign the CONNECTION to the URL.

<increment_stmt>

This statement adds 1 to the target variable. A programmer can use this statement instead of an arithmetic add statement. This statement is a kind of shortcut in BAM language for ease of use for programmers.

<decrement_stmt>

This statement subtracts 1 from the target variable. A programmer can use this statement instead of an arithmetic subtraction statement. This statement is a kind of shortcut in BAM language for ease of use for programmers.

<return_stmt>

This non-terminal statement is used for returning values in functions. In BAM language, return statement also means the end of the execution of the function. Return statement returns the value to the calling function.

<function_stmt>

In BAM language, Function statements consist of 2 elements: function calls and function declarations.

<io>

In BAM language, İo non-terminal only consists of input statements and output statements.

<url_input>

In BAM language, URL inputs are taken with a special reserved word called “bamuin”. With the help of this keyword, we take the input from the connection that is linked to the given URL and put it in the variable which is specified with an identifier.

<timer_input>

In BAM language, this non-terminal is for taking the current timestamp from the timer hardware, which is symbolised with a keyword, and put it in a variable which is also specified with an identifier. Reserved word “bamtin” is used for taking the time input.

<sensor_input>

Inputs are taken from designated sensors which are chosen with the help of reserved word bamsel. After the program knows the target sensor, it uses the reserved word “bamsin” to take the input from the sensor and put it in a specified variable with the help of an identifier.

<output_stmt>

Output statements consist of 2 different kinds of outputs. An output statement is either type of connection output or a url output.

<connection_output>

These are statements used for sending an integer output to connections. The reserved word “bamcout” is used for connection output statements. Although there also is a statement for sending integer output to connections with the assigned URL to a connection, this statement is used for sending the output directly without using URLs.

<connection_input>

These input statements have the same functionality as URL input statements however, one shall pass the connection as the parameter directly instead of an URL. The reserved word “bamuin” is used for connection input statements.

<url_output>

These are statements used for sending a given integer to the connection which is linked to the given URL. The reserved word “bamuout” is used for URL output statements.

<con_sw_stmt>

This non-terminal stands for connection switch statement. In our language, connection switch statements can be 3 different kinds. They are either connection statements or switch statements or sensor selection statements.

<connection>

Connection is a type that has a name which starts with @ symbol followed by an alphanumeric character set. Multiple URLs can be assigned to connections to access them.

<sensor>

This non-terminal consists of 2 components: Dollar sign followed by an integer. All sensor non-terminal instances start with a dollar sign which indicates that the following integer symbolises a sensor. With the help of this non-terminal, correct target sensors are selected in the stage of sensor selection.

<sensor_select>

To determine the target sensor, reserved word “bamsel” and non-terminal <sensor> is used. With integer input taken from <sensor>, the correct sensor is selected to successfully reach the designated target sensor.

<switch_stmt>

In BAM language, switch statements are used for assigning boolean values to the current switches in the hardware. Since a switch can either be on or off, only boolean values are assigned to the switches.

<input_stmt>

Input statements consist of 4 kinds. Connection inputs, url inputs, timer inputs and sensor inputs.

<cond_expr>

Conditional expression is a type of non-terminal expression in BAM language. Conditional expressions are built from 2 expressions and a logic operator in between them. Possible logic operators that can be used in a conditional expression are AND operator, OR operator, NOT operator. An example for a conditional expression would be: A & B.

<cond_expr_element>

Conditional expression element is a non-terminal which is mostly used in the structure of a conditional expression in the BAM language. There are 3 different possible structures for the conditional expression element. It can be the form of a conditional expression with parentheses, can be a relational expression or can be a function call.

<primitive_type>

In BAM language, primitive type non-terminal means the following types: integer, float, string, char and boolean.

<literal>

In BAM language, literal non-terminal are non-terminals which can represent 1 value opposed to the variables. Literals can be of the following types: integer, float, char, string, boolean.

<int>

In BAM language, the int non-terminal can be defined as a positive or negative whole number that also includes zero. Integer values can not have fractional values. While -4, 5, 17 are integers; 3.5, 6.61, -0.73 do not fall into the integer category.

<digit>

In BAM language, digit non-terminal can be thought of as the main building block for integers, floats and some strings. Valid digits in this language are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

<float>

Float non-terminal can be defined as the non-terminal type which stores the floating-point type. Floating-point type can be defined as the data type which includes fractional values. For example: 5.37, -6.21 and 0.89 fall into the float non-terminal type category.

<char>

Char, which stands for character, is any single one of a keyboard letter, symbol or digit. BAM language uses the Unicode standard. Characters are specified by single quotations.

<char_list>

Char lists or character list consists of any combination of characters

<letter>

In BAM language, the letter non-terminal can be thought of as the main building block for strings. Valid letters set for BAM language are stated above in the part (1.13).

<symbol>

In BAM language, symbol non-terminal is the set of characters that are not falling under the alphanumeric category. Parentheses, @, # are some examples of symbols. One can check all of the symbols in part (1.13).

<string>

In BAM language, string non-terminal can be defined as a set of characters. BAM language puts string non-terminals in between (“”). An example string would be: “Hello world!”

<boolean>

In BAM language, boolean non-terminal is a type of data which can have 2 possible values. A boolean is either 1 or 0. One can also think of it as true or false.

<expression>

This non-terminal has 2 subcategories in BAM language. An expression is either type of an arithmetical expression or conditional expression. Both types of expressions are discussed in their own non-terminal explanation parts.

<arith_expr>

In BAM language, this non-terminal symbolises arithmetical expressions. Arithmetical expressions can consist of another arithmetic expression, plus operator / minus operator, and a term. An example arithmetic expression would be: 5 + 7.

<term>

The reason behind not including multiplication division in arithmetic expression non-terminal is to give precedence to multiplication, division, and modulus operations over addition and subtraction. This non-terminal will always be under arithmetic expression non-terminal, therefore will be executed first.

<factor>

Factor is the general form of arithmetic expressions, identifiers, and numeric literals. In short, factors are all of the possible expressions that return a numeric value that will be used in another arithmetic expression.

<identifier>

This non-terminal basicly identifies anything in the BAM language. It identifies variables, functions. It differentiates a variable/function from another one. An example for identifier would be the x in the following: int x = 5;

<relational_op>

This non-terminal stands for relational operators in the BAM language. Relational operators are the operators which are used in relational expressions. These operators decide the relation between 2 relational expression elements. An example would be the > sign in the following expression: 5 < 6.

<relational_expr_element>

Relational expression elements are the one of the main building blocks for constructing a relational expression. In a relational expression, 2 relational expression elements are used. An example for relational expression elements would be number 5 in the following relational expression: 3 > -2.

<alphanumeric>

In BAM language, alphanumeric is a non-terminal character list which consists only of letters and digits. An example for an alphanumeric is: mert123. An example for a non alphanumeric is: mert_123.

<dec_exp>

This non-terminal stands for declaration expression. It is most commonly used when a variable is initially declared. It could be any one of a literal, char, string, expression or a function call.

<func_call>

Function calls are identified by the “bamcall” keyword in the language. The parameter list follows the keyword in parentheses. The programmer must follow the definition of the function when they want to call it. Also, they shall keep an eye to the return type if they are using the function’s output. Parameter list

<comment>

In BAM language, this non-terminal is for comments. To create a comment, the programmer must start the line with // and also must end the line with // again. Any supported character can be written in between the slashes. They will not be executed as a regular line since they are comments in this language.

<parameter_list>

This non-terminal is the parameters of a function. In BAM language, there is no upper limit for the number of parameters that a function can have. Multiple parameters are separated with commas in between them.

<symbol>

In BAM language, this non-terminal defines the characters which are not letters.

<alphanumeric>

In BAM language, this non-terminal defines the characters which are letters and digits.

3. Terminals

NOT_OP	→	!
POW	→	^
PLUS	→	+
INCREMENT	→	++
AND_OP	→	&&
OR_OP	→	||
DIV	→	/
LP	→	(
RP	→	)
ASSIGN_OP	→	=
EE	→	==
NEQ	→	!=
MINUS	→	-
DECREMENT	→	--
MOD	→	%
GT	→	>
GEQ	→	>=
LT	→	<
LEQ	→	<=
LB	→	{
LSB	→	[
RSB	→	]
RB	→	}
MUL	→	*
SQT	→	‘

```
DQT         →     “
DOT         →     .
COMMA       →     ,
UNDER_SCORE →     _
AT          →     @
HASHTAG     →     #
DOLLAR_SIGN →     $
ARROW_OP    →     ->
```

4. Non-Trivial Tokens

4.1 Comments

BAM programming language encourages the programmers to use comments in order to clarify their code and algorithm. Comments help the language to increase its readability. Comments start and end with double forwards slashes (//). The motivation behind this pattern is to both allow the programmers to write multi-line comments and standardise the form of comments throughout all the programs. However, unlike the C programming language, one should put semicolons at the end of each comment just like any other statement in BAM.

4.2 Identifiers

Identifiers could potentially be the name of a function or a variable. The fundamental thing that distinguishes variables and functions in between themselves is the identifier. They can be anything starting with a letter and continuing with alphanumeric characters or symbols. The programmer can name a variable or function in the way that it defines its use or functionality.

Examples:

- actuator_0
- actuator_1234
- sensorRefrigrator98

4.3 Literals

4.3.1 Char: Char (which stands for character) is a single Unicode element. It is specified in single quotation marks.

4.3.2 String: Strings are literals which contain a set of characters, including the white space.

4.3.3 Int: Int (which stands for integer) can be defined as a positive or negative whole number that also includes zero. Integers can not have fractional values.

4.3.4 Float: Float is a data type which has a floating-point value. Floats can be defined as data types which include fractional values.

4.3.5 Boolean: Boolean is a 1-bit data type which can only have 2 possible values. It is either 1 or 0.

4.4 Reserved Words

`IF`	`Used in if blocks. It does not require an else block presence.`
`ELSE`	`Used in else blocks after the presence of if block.`

WHILE	This reserved word declares a while loop. Used for iterative loops in BAM language.
FOR	This reserved word declares a for loop. Used for iterative loops in BAM language.
RETURN	Used for returning values in functions.
FUNCT	Used for declaring functions.
TRUE	This reserved word is a boolean value that represents logical truth.
FALSE	This reserved word is a boolean value that represents logical fallacy.
BAMCIN	This reserved word specifies taking input from a connection in BAM language.
BAMTIN	This reserved word specifies taking input which is a timestamp from a timer in BAM language.
BAMSIN	This reserved word specifies taking input from sensor in BAM language.
BAMUIN	This reserved word specifies taking input from URL in BAM Language.
BAMCOUT	This reserved specifies sending an integer output to a connection in BAM language.
BAMSEL	This reserved word selects the target sensor to take input from.
BAMCONST	This reserved word means the following variable is a constant in BAM language.
BAMCALL	This reserved word indicates that the following identifier will be a function name followed by its parameters and that function will be called.
TIMER	This reserved word specifies a timer that will generate timestamps. Timer fetches the UTC.
SW*	This set of reserved words is used for pointing to switches. Since there are 10 switches in total, * can take any digit [0-9].

4. Evaluation of Language Design Criteria

4.1 Readability

In BAM programming language, any programmer can understand what a program does at the first glance thanks to its similarity with the C language family. The layout of arithmetic operations and use of variables in the terms of declarations and assignments are almost exactly the same with C. However there are some dissimilarities like function calls. When the programmer sees the reserved word “bamcall”, they can easily tell that a function was called in that line of code. This might decrease the writability of the language a bit since the programmer must type another word each time they would like to call a function but it increases the readability which makes it a beneficial tradeoff. Also, reserved words such as “bamcin”, “bamcout”, “bamuin” and “bamuout” states the programmer’s intention efficiently. For example “bamuin” is a self explanatory reserved word for taking input from a connection through the URL that was assigned to that connection. Finally, one cannot write a conditional statement like “if (bool1)” where bool1 is a boolean variable. Instead, “(bool1 == true)” works fine, which decreases the writability but increases both reliability and readability of the language because every conditional expression is standardised this way.

4.2 Writability

Functions are easier to declare compared to C language because BAM does not require the return type of the function just as in Python. This does not cause any reliability problem either because the function can know the return type by itself. The structure of the reserved words for input output statements such as “bamuin”, “bamuout”, “bamcin”, “bamcout”, “bamsin” and “bamtin” are very similar to each other and easy to keep in mind. Logic expressions are compatible with mathematical presentations, so any programmer with basic algebra knowledge can adapt to BAM without any additional effort.

4.3 Reliability

BAM is a quite reliable language since the programmer cannot leave a variable declaration without initialising it. Therefore the programmer does not have to worry about what the default value of any primitive type is. Another reliable aspect is the absence of the void functions in BAM. The programmer is obliged to put a return statement even though the function does not have to return a value. This provides a standard in all function declarations and therefore simplicity. Last but not least, every non-terminal derivation of BAM has left recursion which makes the language even more consistent. BAM does not have type checking nor exception handling features, and it might cause some reliability issues. However, BAM is a programming language specialised for IoT devices with its reserved words for taking input-output from those IoT devices.

5. Conflicts

There are no possible conflicts of any kind in the BAM programming language.