
Bilkent University

Department of Computer Engineering

# CS 315

# Programming Languages

---

**Parser for a Programming Language for IoT Devices**

# *Antibiotic*

**Section 1**

**Group Members:**

Kerem Ayöz - 21501569

Muhammed Safa Aşkın - 21502860

# A. BNF

**STATEMENTS**

<program> → **begin** <statements> **end**

<statements> → <statement> | <statement><statements>

<statement> → <matched> | <unmatched>

<matched> → **if** <logic_expr> <matched> **else** < matched> | <non_if_statement>

<unmatched> → **if** <logic_expr> [ <statement> ] | **if** <logic_expr> [ <matched> ] **else** <unmatched>

<logic_expr> → <logic_value> <logic_operator> <logic_expr>

<logic_value> –> <arithmetic_logic> | <identifier>

<logic_operator> → && | \|\|

<arithmetic_logic> → <expression> <comparator> <expression>

<comparator> → = | < | > | >= | <= | !=

<non_if_statement> → <assignment> | <declaration> | <loop_statement> | <sensor_definition> | <send_function> | <switch_set> | <function_declaration> | <array_decleration> | <array_assignment>

<assignment> → <identifier> ← <expression>

<declaration> → <primitive_type> <identifier> | <primitive_type> <assignment>

<primitive_type> → <signed_integer> | <signed_float> | <bool> | <char> | <string>

<for_loop> → **for** <declaration> <for_seperator> <logic_expr> <for_seperator> <modifier> [<statements>]

<while_loop> → **while** <logic_expr> [<statements>]

<loop_statement> → <for_loop> | <while_loop>

<modifier> → <identifier> ++ | <identifier> --

**EXPRESSIONS**

<identifier> → <letters> | <letters> <numbers> <identifier>

<low_prec_operator> —> + | -

<high_prec_operator> —> * | /

<expression> → <expression> <low_prec_operator> <term> | <term>

<term> → <term> <high_prec_operator> <factor> | <factor>

<factor> → (<expression>) | <identifier> | <number> | <sensor_identifier> |
<function_call> | switch <digit>

<letters> → <letter> | <letters> <letter>

<letter> → a | b | c | d | … | z | _

**TYPES**

<number> → <signed_integer> | <signed_float>

<signed_float> → <sign> <float> | <float>

<float> → <integer> . <integer>

<signed_integer> → <sign> <integer> | <integer>

<integer> → <digit> | <digit> <integer>

<digit> --> 0 | 1 | 2 | … | 9

<sign> → + | -

<*bool*> → true | false

<character> → all ascii characters

<char> → ‘<character>’

<string> → “<text>”

<text> → <character> | <character> <text>

<capital_letters> → <capital_letter> | <capital_letters> <capital_letter>

<capital letter> → A | B | C | D | E | … |Z|

<for_seperator> → \|

**ARRAY**

<array_declaration> → <primitive_type> <identifier> \< <integer> \>

<array_assignment> → <array_element> ← <expression>

<array_element> → <identifier> \< <integer> \>

**FUNCTIONS**

<function_call> → <identifier> (<input_parameter_list>) (<output_parameter_list>) |
<identifier> (empty) (empty) | <identifier> (<input_parameter_list>) (empty) |
<identifier> (empty) (<output_parameter_list>) |
<send_function> | <receive_function> | <connect_function> | <timestamp>

<function_dec> → **function** <identifier> (<parameter_list>) (<parameter_list>)
[<statements>]

<parameter_list> → <primitive_type> <identifier> , <parameter_list>

| <primitive_type> <identifier> | empty

<input_parameter_list> → <expression>, <input_parameter_list> | <expression>

<output_parameter_list> → <identifier>, <output_parameter_list> | <identifier>

<print_statement> → <print_symbol> <expression>

| <print_symbol> <bool>

| <print_symbol> <char>

<print_symbol> → \>\>

<empty> —> _

**IoT FEATURES**

<connect_function> → **connect**(<url>),

<send_function> —> **send**(<identifier>, <integer>)

<receive_function> —> **receive**(<identifier>)

<url> → \'((http)|(ftp))s?:\/\/[a-zA-Z0-9]{2,}(\.[a-z]{2,})+(\/[a-zA-Z0-9+=?]*)*\'

<timestamp> → time()

<sensor_identifier> → _{capital_letters}

<sensor_definition> → **sensor** < sensor_identifier>  **port** <integer>

<switch> → **SWITCH_**<digit>

<switch_set> —> **SWITCH_**<digit> ← <bool>

# Definitions

## Program Statements

**<program>**: The scope where program is being executed. Corresponds to main() in C programming language. Program starts with begin and ends with end reserved words.

**<statements>**: Consists of a single statement or statement followed by <statements> recursively. This non terminal shows how <statement> can be combined with other statements.

**int a <-- 10 + 6**

**if a > 5 [statement]**

**<statement>**: Describes the all possible code blocks. They can be either <matched> or <unmatched>.

**<matched>**: This non terminal contains all type of statements that can be written instead of <unmatched> (<unmatched> can be described as the if statements with less number of else statements) conditional statements.

**<unmatched>**: This non-terminal contains a conditional statement with only one if, or a recursive rule for creating additional conditional statements.

**<logic_expr>**: This non terminal defines the logical expressions that could be created in Antibiotic programming language. These expressions could be arithmetic logical expressions or identifiers. Truth value of identifier is different for each primitive type for instance bool variable used directly whereas int variables are evaluated 0 as false and other values as true. Also empty string is false while non-empty string is true. These logical expressions are separated with logical operators.

**3 < 5 && flag && (6 = 7 || x > 8)**

**<logic_operator>**: Describes usual logical operators used in every language: AND, OR

**&& and ||**

**<arithmetic_logic>:** This non terminal designates how an arithmetic logic can be defined. It takes two <expression> and a <comparator> between these expressions. The result can be either true or false.

**<comparator>:** This non terminal defines which character sets can be used for comparison. The list of comparison symbols is "=" or "<" or ">", ">=" , "<=" or " !="

**<non_if_statement>:** Defines all statements instead of if statement, can be listed as expression, assignment, declaration, loop statements, sensor definition, send function, switch setting, function declaration, array declaration or array assignment functions.

**<assignment>:** Defines how a variable can be assigned. Starts with an identifier followed by the reserved symbol for assignment which is "<--" and ends with an expression.

**prime <-- 11**

**<declaration> :** Defines how a variable can be declared. Starts with a primitive type followed by an identifier or identifier followed by an assignment operation.

**int prime <-- 7**

**int prime**

**<primitive_type>:** Defines all primitive types that language contains. They can be listed as bool, string, character, signed float, signed integer.

**<for_loop>:** This non terminal defines the for loop statement that starts with reserved word **for** followed by variable declaration followed by a logical expression and a modifier at the end. This for statement uses <for_sepeator> to separate all elements from each other. It can simply shown as below.

**for int i = 0 | i < 10 | i++ [ statements ]**

**<while_loop>:** This non terminal defines the while loop statement, which starts with reserved word **while** followed by one or more logic expressions. It ends with statements inside of box parenthesis.

**while a < 5 [statements]**

**<loop_statement>:** Loop statement consists of <while_loop >and <for_loop>.

**<modifier>:** Increment or decrement operation done in for loop. It has the same structure with C++ signature operation ++ and - -

## Expressions

**<identifier>:** This non terminal specifies how an identifier name can be defined. It can be named by <letters> or <letters> followed by <numbers> and <identifier> recursively.

**frequency, function1, relayModule**

**<low_prec_operator>:** Represents low precedence operators such as minus and plus.

**<high_prec_operator>:** Represents high precedence operators such as multiplication and divison.

**<expr>:** This non terminal specifies how an expression can be defined. It can be composed of expression + | - <term> or single <term>.

**<term>:** Defines how multiplication and division operations take into place. In order to eliminate ambiguity it has precedence on other expressions like “+” and “-”

**<factor>:** Defines all possible elements that can be used in expression, which are identifier, number, sensor identifier, function or switch.

## Types

**<letters>:** Defines the block of <letter>s or single <letter>.

**<letter>:** This non terminal is the collection of all lower case letters. It contains all English letters and underscore sign.

**[a-z] + “_”**

**<number>:** Defines what is defined as number. It can be either signed floating point number or signed integer.

**<signed_float>:** Defines the signed floating point number, which is a floating point number with a plus or minus sign in front of it.

**<float>:** Defines how a floating point number can be defined. It is the composition of integers with a dot sign at the middle.

**<integer> . <integer>**

**<signed_integer>:** Defines the signed integer, which is an integer value with a <sign> in front of it.

**<integer>:** Defines what is counted as integer, which is single digit number or digit that is followed by <integer> recursively.

**<digit>:** Defines what is counted as digit. Contains all digits [0-9]

**<sign>:** This non terminal indicates the acceptable sign symbols. Which are “+” or “-”

**<*bool*>:** Binary logic value which could be either true or false. Could be used in logical expressions.

**<character>:** This non terminal defines what counted as character, which are all ascii characters.

**<char>:** This non terminal specifies how to declare a char primitive type. It starts with (‘) and ends with another (‘) .

**<string>:** Describes how a string variable can be defined, which is a <text> between quotation symbols.

**“Hello Antibiotic”**

**<text>:** Describes what text is. It can be either a single character or a character followed by another <text> recursively.

**<character> | <character> <text>**

**<capital_letters>:** Allows left recursive definitions of <capital_letter>

**<capital_letter>:** Defines letters in uppercase form. [A-Z]

**<for_seperator>:** Defines the symbol that separates for loop elements from each other, which is “|”

## Arrays

**<array_declaration>:** This non terminal defines how an array can be declared. Starts with <primitive_type> followed by an identifier and ends with the length of array <5>.

**int animals <5>**

**<array_assignment>:** Defines how an <array_element> can be assigned. Starts with specifying array element followed by reserved word for assignment “<--” and ends with expression.

**<array_element>:** Defines how to reach a specific element of an array. It starts with <identifier> and continues with an integer between “< >” symbols.

**animals<5>**

## Functions

**<function_call>:** All possible types of function calls with or without parameters. It contains send function call, connect and time function call or receive function call.

**addition(x,y) (z)**

**<function_dec>:** Declaration of user defined functions. Starts with function reserved word. It could take undefined number of inputs and outputs as parameter.

```
function addition (int x, int y) (int z) [
    z = x + y
]
```

**<parameter_list>:** This is used in function definition where developers specify the input and output parameters.

**int var1, float var2, string var3**

**<input_parameter_list>:** This non-terminal specifies the input list for a function call. It could be single or multiple expressions separated by comma (,).

**x, 3+6, y, "Program"**

**<output_parameter_list>:** This non-terminal could be an identifier since function assigns a value to an output parameter in the end of its execution.

**z, x, w**

**<print_statement>:** This specifies the reserved word printing operation which starts with >> and continues with the printed element.

```
>> "CS-315 Project-1 Report"
>> (x+9) % 4
```

**<print_symbol>:** This non terminal denotes the reserved symbol for print statement which is >> in Antibiotic.

## IoT Related Features

**<connect_function>:** Uses connect reserved word and takes a url as parameter. It assigns connection to an identifier.

**connection <-- connect(www.google.com)**

**<send_funciton>:** This non terminal denotes how to define a send function. Starts with send reserved word followed by two parameters, connection and an integer value.

**send(connection, 100)**

**<receive_funtion>:** This non terminal denotes how to define a receive function. Starts with receive reserved word followed by a connection parameter. It returns an integer value.

**int value <--receive(connection)**

**<url>:** This variable declares which characters are acceptable for url definiton.

**<timestamp>:** Terminal variable that contains the time() function call.

**<sensor_definition**>: This nonterminal variable is used for defining a new sensor. It uses sensor reserved word followed by <sensor_identifier> and ends with port reserved word with port number.

**sensor _HUMIDITY port 10**

**sensor _TEMPRATURE port 20**

**<sensor_identifier>:** Sensor identifier that must start with underscore character(_) and continue with capital letters. This is used to define the name of the sensor that is created.

**<switch>:** Non-terminal that implies the switch in IoT node. In our project we have 10 different switches thus switch could take 10 different values.

**SWITCH_1, SWITCH_2, … , SWITCH_9**

**<switch_set>:** Non-terminal to set a switch on and off. It starts with particular switch followed by assignment operator and a boolean at the end. True indicates on and false indicates off.

**SWITCH_10 <--true**

# B. LEX

```
digit  [0-9]
integer {digit}+
float {digit}*(\.)?{digit}+
signed_integer ([+-]?{integer})
signed_float ([+-]?{float})
letter [a-z]
capital_letter [A-Z]
letters {letter}+
capital_letters {capital_letter}+
alphabetic ({letter}|{capital_letter}|$|_)
strings \"(\\.|[^"\\])*\"
chr \'{character}\'
character ({digit}|{alphabetic})
identifier {alphabetic}+{character}*
url \'((http)|(ftp))s?:\/\/[a-zA-Z0-9]{2,}(\.[a-z]{2,})+(\/[a-zA-Z0-9+=?]*)*\'
sensor_identifier \_{capital_letters}
comment  \#.*

%%

\+                      return( PLUS );
\-                      return( MINUS );
\*                      return( MULTIPLY );
\/                      return( DIVIDE );
\%                      return( MODULUS );
\;                      return( SEMICOLON );
\&\&                    return( AND );
\|\|                    return( OR );
\>                      return( GREATER );
\<                      return( SMALLER );
\>\=                    return( GREATER_EQUAL );
\<\=                    return( SMALLER_EQUAL );
\,                      return( COMMA );
\.                      return( DOT );
\<\-\-                  return( ASSIGN );
\!\=                    return( NOT_EQUAL );
\=                      return( EQUALITY_CHECK );
\{                      return( LEFT_BRACE );
\}                      return( RIGHT_BRACE );
\(                      return( LEFT_PARANT );
\)                      return( RIGHT_PARANT );
\[                      return( LEFT_BOX );
\]                      return( RIGHT_BOX );
\>\>                    return( PRINT );
\|                      return( FOR_SEPERATOR );
\+\+                    return( INCREMENTOR );
\-\-                    return( DECREMENTOR );
if                      return( IF );
elseif                  return( ELSEIF );
else                    return( ELSE );
for                     return( FOR );
while                   return( WHILE );
int                     return( INT );
float                   return( FLT );
string                  return( STRING );
char                    return( CHAR );
bool                    return( BOOL );
function                return( FUNCTION );
sensor                  return( SENSOR );
time                    return( TIME );
connect                 return( CONNECT );
send                    return( SEND );
receive                 return( RECEIVE );
port                    return( PORT );
true                    return( TRUE );
false                   return( FALSE );
switch                  return( SWITCH );

{sensor_identifier}     return( SENSOR_IDENTIFIER );
{identifier}            return( IDENTIFIER );
{integer}               return( INTEGER );
{float}                 return( FLOAT );
{comment}               return( COMMENT );
{strings}               return( TEXT );
{url}                   return( URL );
{chr}                   return( CHARACTER );

%%

int yywrap() { return 1; }
```

# C. YACC

```
/* Antibiotic.y */

%{
    #include <stdio.h>
    #include <stdlib.h>
    void yyerror(char *);
    int yylex(void);
%}

%token
    PLUS
    MINUS
    MULTIPLY
    DIVIDE
    MODULUS
    SEMICOLON
    GREATER
    SMALLER
    GREATER_EQUAL
    SMALLER_EQUAL
    COMMA
    AND
    OR
    DOT
    ASSIGN
    NOT_EQUAL
    EQUALITY_CHECK
    LEFT_BRACE
    RIGHT_BRACE
    LEFT_PARANT
    RIGHT_PARANT
    LEFT_BOX
    RIGHT_BOX
    NEW_LINE
    PRINT
    FOR_SEPERATOR
    INCREMENTOR
    DECREMENTOR
    IF
    ELSEIF
    ELSE
    FOR
    WHILE
    INT
    FLT
    STRING
    CHAR
    BOOL
    FUNCTION
    SENSOR
    TIME
    CONNECT
    SEND
    RECEIVE
    PORT
    SENSOR_IDENTIFIER
    IDENTIFIER
    INTEGER
    FLOAT
    CHARACTER
    COMMENT
    TEXT
    SWITCH
    TRUE
    FALSE
    URL
%%
```

```
start: program {
            printf("Input is valid!");
            return 0;
        }

program: statements ;

statements: statement statements | statement ;

statement:  IF logic_expr LEFT_BOX matched RIGHT_BOX unmatched | non_if_statement ;

matched
:  IF logic_expr LEFT_BOX matched RIGHT_BOX ELSE LEFT_BOX matched RIGHT_BOX | non_if_state
ment ;

unmatched: ELSE LEFT_BOX single_if RIGHT_BOX | ;

single_if:  IF logic_expr LEFT_BOX single_if RIGHT_BOX | non_if_statement;

identifier: IDENTIFIER;

logic_expr: logic_value logic_operator logic_expr | logic_value;

logic_value: arithmetic_logic | identifier ;

logic_operator: AND | OR ;

arithmetic_logic: expression comparator expression ;

comparator
: EQUALITY_CHECK | GREATER | SMALLER | SMALLER_EQUAL | GREATER_EQUAL | NOT_EQUAL ;

non_if_statement:  assignment | declaration | loop_statement | print_statement
| sensor_definition | send_function | switch_set | function_declaration
| array_decleration | array_assignment ;

assignment: identifier ASSIGN expression ;

declaration: primitive_type identifier | primitive_type assignment ;

primitive_type: INT | FLT | BOOL | CHAR | STRING  ;

for_loop
: FOR declaration for_seperator logic_expr for_seperator modifier LEFT_BOX statements RIGH
T_BOX ;

while_loop: WHILE logic_expr LEFT_BOX statements RIGHT_BOX ;

loop_statement: for_loop | while_loop ;

modifier: identifier INCREMENTOR | identifier DECREMENTOR ;

expression: expression low_prec_operator term | term ;

term: term high_prec_operator factor | factor ;

factor: LEFT_PARANT expression RIGHT_PARANT | identifier | number
| function_call | sensor_identifier | SWITCH INTEGER | TEXT ;

low_prec_operator: PLUS | MINUS ;

high_prec_operator: MULTIPLY | DIVIDE ;

number: signed_integer | signed_float ;

signed_float: sign float | float ;

float: FLOAT ;

signed_integer: sign integer | integer ;

integer: INTEGER ;

sign: PLUS | MINUS ;

bool: TRUE | FALSE ;

for_seperator: FOR_SEPERATOR ;
```

```
array_decleration: primitive_type identifier SMALLER integer GREATER ;

array_assignment: array_element ASSIGN expression ;

array_element: identifier SMALLER integer GREATER ;

function_call
: identifier  LEFT_PARANT input_parameter_list RIGHT_PARANT  LEFT_PARANT output_parameter_
list RIGHT_PARANT
| identifier LEFT_PARANT RIGHT_PARANT LEFT_PARANT RIGHT_PARANT
| identifier LEFT_PARANT input_parameter_list RIGHT_PARANT LEFT_PARANT RIGHT_PARANT
| identifier LEFT_PARANT RIGHT_PARANT LEFT_PARANT output_parameter_list RIGHT_PARANT
| receive_function | connection | timestamp ;

input_parameter_list: expression COMMA input_parameter_list | expression ;

output_parameter_list: identifier COMMA output_parameter_list | identifier ;

function_declaration
: FUNCTION identifier LEFT_PARANT parameter_list RIGHT_PARANT LEFT_PARANT parameter_list R
IGHT_PARANT LEFT_BOX statements RIGHT_BOX ;

parameter_list: primitive_type identifier COMMA parameter_list
| primitive_type identifier | ;

print_statement: PRINT expression | PRINT bool | PRINT CHARACTER ;

sensor_identifier: SENSOR_IDENTIFIER ;

switch_set: SWITCH INTEGER ASSIGN bool ;

sensor_definition: SENSOR sensor_identifier PORT INTEGER ;

timestamp: TIME LEFT_PARANT RIGHT_PARANT ;

connection: CONNECT LEFT_PARANT URL RIGHT_PARANT ;

send_function: SEND LEFT_PARANT identifier COMMA INTEGER RIGHT_PARANT ;

receive_function: RECEIVE LEFT_PARANT URL RIGHT_PARANT ;


%%
#include "lex.yy.c"

int lineNo;
int state = 0;

int main() {
    yyparse();
    if (state == 0) {
        fprintf(stderr, "Parsing is successfully completed.\n");
    }
    return 0;

}

void yyerror( char *s ) {
    state = -1;
    fprintf(stderr, "%s in line %d\n", s, (lineNo+1));
}
```

# D. EXAMPLE PROGRAM

```
float temperature <-- _TEMPRATURE
float humidity <-- _HUMIDITY
int soundLevel <-- _SOUND

string a <-- "Hava Soguk"
string b <-- "Hava Sicak"
string c <-- "Hava Ortalama"


if temperature < 10 [
>> a
]
else [
>> c
]

for int b <-- 0 | b < 10 | b++ [
        if b = 0 [
            >> time()
        ]
        else [
            >> "Antibiotic is still running great"
        ]
]


int switchNumber <-- 0
while  temperature > 10 && humidity > 5 && sound > 20 [
        >> "It is so boring inside"
        if switchNumber < 10 [
                switchNumber <-- switchNumber + 1
        ]
        switch 5 <-- false
]


connection <--  connect('http://www.cs.bilkent.edu.tr/guvenir/')
send(connection, 100)

function calculateAvarage (int note1, int note2, int note3, int note4)(float average)[
        int total <-- note1 + note2 + note3 + note4
        average <-- total / 4
]
```

```
int projectNote1 <--  receive('http://www.cs.bilkent.edu.tr/guvenir/notes/21502960')
int projectNote2 <--  receive('http://www.cs.bilkent.edu.tr/guvenir/notes/21502360')
int projectNote3 <--  receive('http://www.cs.bilkent.edu.tr/guvenir/notes/21505460')
int projectNote4 <-- receive('http://www.cs.bilkent.edu.tr/guvenir/notes/21501260')


float average <-- calculateAvarage(quizNote1, quizNote2, quizNote3, quizNote4)()

if projectNote1 < average [
        >> "My note is below the project average"
]
else [
        >> "THANKS FOR MY NEW NOTE"
]
```