Utku **B**oran Torun 21901898 Section-01

Mehmet **O**nur Uysal 22002609 Section-01

**Y**arkın Sakıncı 22003439 Section-01

BOY

# BOY Language in BNF Form

<program> ::= <stmt\_list>

<stmt\_list> ::= <block\_stmt> | <stmt\_list> <func\_def>

<block\_stmt> ::= <stmt> | <block\_stmt> <stmt>

<stmt> ::= <matched> | <unmatched>

<matched\_list> ::= <matched> | <matched\_list> <unmatched>

<unmatched\_list> ::= <unmatched> | <unmatched\_list> <stmt>

<func\_def> ::= function <return\_type> <identifier>(<argument\_list>)

{ <block\_stmt> }

<matched> ::= <matched\_stmt> <matched\_list> } | <unconditional\_stmt>

<unmatched> ::= <matched\_first>

| if(<condition\_list>) { <unmatched\_list> }

| <matched\_stmt> <unmatched\_list> }

<matched\_first> ::= if(<condition\_list>) { <matched\_list> }

<matched\_stmt> ::= <matched\_first> else {

<condition\_list> ::= <and\_term>

| <condition\_list> || <and\_term>

| !( <condition\_list>) || <and\_term>

| !(<condition\_list>)||(!(<and\_term>))

| <condition\_list> || (!(<and\_term>))

<and\_term> ::= <condition>

| (<condition>)

| <and\_term> && <condition>

| <identifier>

<condition> ::= <bool\_const>

| <bool\_identifier>

| <identifier> <conditional\_op> <identifier>

| <numeric\_const> <conditional\_op> <identifier>

| <numeric\_const> <conditional\_op> <const>

| <identifier> <conditional\_op> <const>

<unconditional\_stmt> ::= <assign\_stmt>

| <declaration\_stmt>

| <func\_call>

| <loop>

| <reserved\_stmt>

| <increment\_stmt>

| <decrement\_stmt>

| <comment>

<increment\_stmt> ::= <identifier> ++ ;

<decrement\_stmt> ::= <identifier> -- ;

<declaration\_stmt> ::= <type\_id> <identifier\_list> ;

| <type\_id> <identifier\_list> = <block\_expression> ;

| <type\_id> <identifier\_list> = <func\_call>

<assign\_stmt> ::= <identifier> = <block\_expression> ;

| <identifier> = <func\_call> ;

<block\_expression> ::= <non\_numeric\_const> | <arith\_op>

<type\_def> ::= <type\_id> | void

<expression> ::= <non\_numeric\_const> | <arith\_op> | <func\_call>

<func\_call> ::= <identifier>(<arguments>);

| <primitive\_function\_call>

<arguments> ::= <identifier>

| <const>

| <identifer> , <arguments>

| ε

| <const> , <arguments>

<primitive\_function\_call> ::= display( <block\_expression>);

| display();

| readData( <int\_const\_unsigned>);

| readData( <identifier>);

| getTimestamp(<int\_const>);

| getTimestamp(<identifier>);

| connectURL(<string\_const>);

| connectURL(<identifier>);

| sendIntToConnection(<string\_const>, <identifier>);

| sendIntToConnection(<string\_const>, <int\_const>);

| sendIntToConnection(<identifier>, <identifier>);

| sendIntToConnection(<identifier>, <int\_const>);

|receiveIntFromConnection(<string\_const>);

|receiveIntFromConnection(<identifier>);

| setSwitch( <bool\_const>,<digit>);

| setSwitch( <identifier>, <digit>);

| setSwitch( <bool\_const>, <identifier>);

| setSwitch( <identifier>, <identifier>);

<return\_type> ::= <type\_id> | void

<argument\_list> ::= <type\_id> <identifier>

| ε

| <type\_id> <identifier> , <argument\_list>

<loop> ::= <do\_while> | <while\_loop> | <for\_loop>

<reserved\_stmt> ::= return;

| return <block\_expression>;

| return <func\_call>;

| break;

| continue;

<for\_update> ::= <identifier> = <expression>

| <identifier> (<arguments>)

| <identifier>++

| <identifier>--

<do\_while> ::= do { <block\_stmt> } while (<condition\_list>);

| do { <block\_stmt> } while (!(<condition\_list>));

<while\_loop> ::= while (<condition\_list>) { <block\_stmt> }

| while ( !( <condition\_list>) ) { <block\_stmt> }

<for\_loop> ::= for(<assign\_stmt> <condition\_list>; <for\_update>)

{<block\_stmt>}

| for ( <assign\_stmt> !(<condition\_list>) ;

<for\_update>) {<block\_stmt>}

<const> ::= <numeric\_const> | <non\_numeric\_const>

<numeric\_const> ::= <int\_const> | <float\_const>

<non\_numeric\_const> ::= <bool\_const>

| <string\_const>

| <char\_const>

<bool\_const> ::= true | false

<int\_const\_unsigned> ::= <digit> | <digit> <int\_const\_unsigned>

<int\_const> ::= <sign> <int\_const\_unsigned> | <int\_const\_unsigned>

<float\_const> ::= <int\_const> . <int\_const\_unsigned>

| . <int\_const\_unsigned>

| <sign> . <int\_const\_unsigned>

| <int\_const>

<string\_const> ::= “<string>”

<string> ::= <char> | ε | <char> <string>

<char> ::= any character on keyboard

<char\_const> ::= ‘<char>’

<conditional\_op> ::= == | != | < | <= | > | >=

<arith\_op> ::= <arith\_op> + <term>

| <arith\_op> - <term>

| <term>

<term> ::= <term> \* <factor>

| <term> / <factor>

| <factor>

<factor> ::= <numeric\_const> | <identifier> | ( <arith\_op> )

<comment> ::= /\* <string> \*/

<identifier\_list> ::= <identifier> | <identifier>,<identifier\_list>

<identifier> ::= <letter>|<identifier><digit>|<identifier><letter>

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

<digit> ::= 0|1|2|3|4|5|6|7|8|9

<sign> ::= + | -

<type\_id> ::= int | float | char | bool | string

<reserved\_words> ::= int | float | char | string |bool | string | function | if | else if | else | for | do | while | continue | break

| display | read | void

# Explanation List

<program> : It is the combination of statements and it is the start variable of the language.

<int\_const\_unsigned> : Any combination of the digits 0 to 9.

<int\_const> : Any combination of digits with an optional sign before the number.

<float\_const> : Includes all <int\_const> values and also floating point numbers with an optional sign.

<char> : Represents any character

<char\_const> : Any single character that is enclosed in single quotation marks.

<string> : Any combination of characters.

<string\_const> : Any <string> that is enclosed in double quotation marks.

<bool\_const> : Represents boolean constants. Takes values true or false.

<numeric\_const> : Integer or float constant.

<non\_numeric const> : Boolean, string or char constant.

<const> : General definition of constant values. Can be numeric or non numeric literal values.

<arith\_op> : Represents arithmetic operations. In the language there are standart operations for addition, subtraction, multiplication, division and modulo. This variable is also used as the left hand side of addition, subtractions operations and can also be a single <term>.

<term> : Used as the left hand side of multiplication and division operations and right hand side of addition and subtraction operations. Can also be a single <factor>.

<factor> : Can be a numeric constant, identifier or an arithmetic operations enclosed in paranthesis. Represents the final step of operations.

<term> and <factor> variables are implemented and used seperately in the operations in order to prevent ambiguity due to precedence. All the steps of the operations that have different precedences take place in different layers. Therefore proper operation precedence is enforced to compilers without any ambiguity.

<identifier> : Used for naming variables and functions. Can be an instance of a string but has additional restrictions. First character must be a letter, ‘\_’ character or ‘$’ character. Other characters can be one of those characters, or digits. Another restriction for identifiers is that the reserved words cannot be used. All the reserved words used in the language are listed in the <reserved\_words> variable.

<type\_id> : Can be any of the type names used in the language. Primitive types of the language are int, float, char, bool and string. Variable types are written in the declaration of variables so each variable strictly has a type.

<reserved\_words>: This variable is about the reserved words that we have in our language. These words cannot be used as identifier.

<primitive\_function\_call>: This variable is for calling functions that are already in language.

<func\_def>: This is used for function definition. All function definitions have to be at the end of the program, due to design choice.

<func\_call>: Function call variable is basically rule for how to call function.

<arguments>: This is for arguments that we use for our functions and these arguments are separated with comma.

<return\_type>: When function returns there should be return type. This variable contains these types including void.

<comment>: In order to make program more readable programmers may choose to add comments to their code. This variable is for these comments.

<expression>: This is for the right-hand side of the assignment. It cannot be used as left-hand side. In addition to this, it can be used for declarations.

<assign\_stmt>: It is for assignment of identifier.

<declaration\_stmt>: Declaration statement is for creating objects with or without assigning to any expression.

<identifier\_list>: Identifier list is a list of identifiers separated with comma which used in order to have more than one declaration in one line.

<condition>: Condition is basically boolean expression.

<condition\_list>: This list is mainly used for conserving precedence while doing ‘or’ operations.

<and\_term>: This is used for logical operator precedence. (‘and’ and ‘or’ operators)

<stmt>: Statement is basically combination of matched and unmatched statements.

<unconditional\_stmt>: These statements are combination of assignment, declaration, function call, loops and reserved statements. Note that there is no conditional statement in this variable.

<reserved\_stmt>: These statements are used in functions and loops, in order to return or stop the function.

<for\_update>: This variable is for third statement of for loop.

<loop>: It is basically all loops in a programming language (for, while, do-while).

<stmt>: is any statement that is either <matched> or <unmatched>. If the number of if’s in the

<stmt> is equal to the number of else’s then it is <matched> if the number of if’s in the

<stmt> are bigger than number of else’s, the <stmt> is <unmatched>

<matched>: any statement or statements that have equal number of if and else’s. A <stmt> is

<matched> if it is an <unconditional\_stmt> or if it has the structure: if( <condition\_list> ){

<stmt> } else{<stmt>} where the <stmt>’s inside both the if’s statement body and else’s statement body are <matched>.

<unmatched>: any statement or statements that have more if’s than else’s. A <stmt> is

<unmatched> if it has the structure: if( <condition\_list> ){ <stmt> } where <stmt> is either

<matched> or <unmatched> or if it has the structure: if( <condition\_list> ){ <stmt> } else{<stmt>} where the <stmt>’s inside the if’s statement body is <matched> and else’s statement body is <unmatched>.

<block\_stmt>: It is basically any statement that can be used in blocks. There is no function definitions in blocks, so we do not have that variable in this variable.

<matched\_first>: It is used for avoiding conflicts between matched and unmatched statements.

<matched\_stmt>: To avoid conflict, this variable is used for matched statements.

<matched\_list>: List of matched statements.

<unmatched\_list>: List of unmatched statements.

<increment\_stmt>: To increment a variable by one, this variable is used.

<decrement\_stmt>: To decrement a variable by one, this variable is used.

<block\_expression>: When semi colon was used, there were confusions. In order to avoid this problem, this variable is used.

# Language Criteria

Our language has simplicity in terms of operations, data types, conditional statements, reserved words and primitive functions. BOY, has limited operators for basic operations and other complex operations must be defined explicitly. Which increases readability however it decreases writability because some operations implemented with more lines of code. Also, there are increment and decrement operators that increases the writability but decreases readability. Data types are self-explanatory and basic. Variables are defined with data types in

the declaration statements which prevents confusion and increases reliability. Having less data types which are distinct prevents confusion. This enables type checking which also increases reliability. There are only essential reserved words which decreases the restrictions for the identifier names and increases writability. This is also the case with the primitive functions.

Primitive functions are for displaying, connecting to URL, getting sensor data and getting the timestamp. All the other functions are not primitive and must be implemented with the keyword “function”. Use of this keyword increases readability while decreasing writability. Functions are defined with their return types which increases reliability.

In our language, use of curly brackets is mandatory for every loop, function and conditional statement bodies even when there is only one line of code. This may decrease writability but increases readability and reliability. Since, the language does not allow ambiguity in if-else statements and operator precedence, it is reliable in terms of the consistency of the output. If and else statements are matched and the standard operator precedence is enforced.