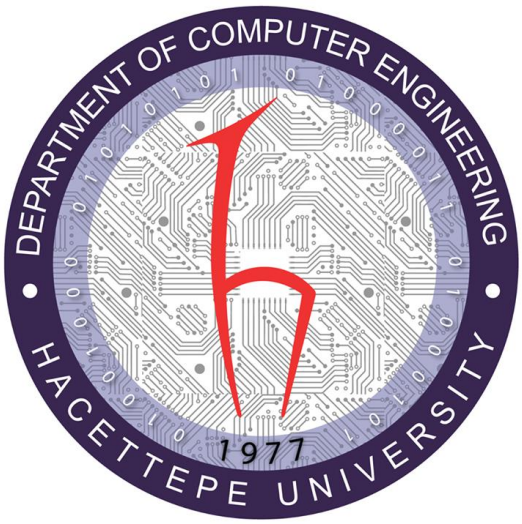


**BBM301**

# 1st_Project Report

Collaborators:
HAZEM ALABIAD 21403802
M.MALEK BABA 21403447
ABEDELJALIL JARJANAZY 21501041
Mohammed Ali 21403227

Subject: **Lex & Yacc**

- **<u>Introduction</u>:**
  - The idea behind this project is to design our own programming language based on Lex & Yacc tools. First we define our BNF grammars then, using these grammars we create our Lex & Yacc files. Lex: is a tool for writing lexical analyzers, Yacc: is a tool for constructing parsers.
  Lex reads our input character sets and converts them into tokens that we feed to the Yacc file in order to parse them and do the wanted actions. Since we are designing our own language we can choose the relevant action we wish.

  **Solution steps:**

  a) Setting up BNF grammars with our imagination and functions that may used to ease the job.
  b) Creating Lex file and specifying the patterns (regex) that are going to be passed to the Yacc file as tokens.
  c) Constructing Yacc file by defining the tokens, the data types, the grammar rules, and the C functions.
  d) Code debugging time.
  e) Error checking & handling through the right functions e.g: yyerror() .. etc.

- **<u>BNF grammar:</u>**

```
<program> ::= <statment_list>
                ;
<statement_list> ::= <statement> | <statement_list> <statement>
                ;
<statement> ::= <assignment> SEMICOLON
                | <declaration> SEMICOLON
                | <loop>
                | <condition>
                | GOTO COLON <flag> SEMICOLON
                | <comment>
                | <function_call> SEMICOLON
                | BREAK SEMICOLON
                | CONTINUE SEMICOLON
                | RETURN SEMICOLON
                | RETURN IDNTF SEMICOLON
                | RETURN <factor> SEMICOLON
                ;
```

<function> ::= FUNCTION <return_type> IDNTF LEFT_PARANTHESIS <parameter_list> RIGHT_PARANTHESIS <block>
;
<return_type> ::= <data_type>
;
<parameter_list> ::= <empty>
| <data_type> IDNTF
| <parameter_list> COMMA <data_type> IDNTF
| VOID
;
<block> ::= LEFT_BRACKET <statement_list> RIGHT_BRACKET
| LEFT_BRACKET <empty> RIGHT_BRACKET
;
<declaration> ::= <data_type> IDNTF
| <declaration> <assignment_operator> <RHS>
| ARRAY <data_type> IDNTF LEFT_SQ_BRACKET INT_LTRL RIGHT_SQ_BRACKET ASSIGNMENT_OPT LEFT_BRACKET <factor_list> RIGHT_BRACKET
;
<factor_list> ::= <factor>
| <factor_list> COMMA <factor>
;
<RHS> ::= <arithmetic_expression>
| <function_call>
| <boolean_expression>
;
<function_call> ::= IDNTF LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_PRINT LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_LIST_CONTENTS LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_GET_SIZE LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_CREATE LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_COPY LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_COMPARE LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_CONNECT_TO LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS
| BLTIN_DELETE LEFT_PARANTHESIS <identifier_list> RIGHT_PARANTHESIS

| BLTIN_RENAME LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_MOVE LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_SORT LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_FILTRE_FILES LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_BACK_UP LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_SYNCHRONIZE LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_SEARCH LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_CD LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_DOWNLOAD LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_UPLOAD LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_OPEN LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_PROPERTIES LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_MOVE_BACK LEFT_PARANTHESIS <identifier_list>
RIGHT_PARANTHESIS
| BLTIN_MOVE_FORWARD LEFT_PARANTHESIS
<identifier_list> RIGHT_PARANTHESIS
;
<identifier_list> ::= <empty>
| IDNTF
| <identifier_list> COMMA IDNTF
| <factor>
| <identifier_list> COMMA <factor>
;
<arithmetic_expression> ::= <term>
| <arithmetic_expression> ADD_OPT <term>
| <arithmetic_expression> SUB_OPT <term>
;
<term> ::= <factor>
| <term> MULTIPLY_OPT <factor>
| <term> DIVIDE_OPT <factor>
| <term> POW_OPT <factor>
| <term> MODE_OPT <factor>

;

<factor> ::= INT_LTRL
| FLT_LTRL
| STR_LTRL
| CHR_LTRL
| DOUBLE_LTRL
;

<assignment_operator> ::= ASSIGNMENT_OPT
| MULTIPLY_ASSIGNMENT_OPT
| DIVIDE_ASSIGNMENT_OPT
| ADD_ASSIGNMENT_OPT
| SUB_ASSIGNMENT_OPT
| MODE_ASSIGNMENT_OPT
| POW_ASSIGNMENT_OPT
;

<assignment> ::= <LHS> <assignment_operator> <RHS>
| <LHS> INCREMENT_OPT
| <LHS> DECREMENT_OPT
;

<LHS> ::= IDNTF
| IDNTF LEFT_SQ_BRACKET INT_LTRL RIGHT_SQ_BRACKET
;

loop
: <while_loop>
| <do_while_loop>
| <for_loop>
;

<while_loop> ::= WHILE LEFT_PARANTHESIS <boolean_expression> RIGHT_PARANTHESIS <block>
;

<do_while_loop> ::= <do_statement> WHILE LEFT_PARANTHESIS <boolean_expression> RIGHT_PARANTHESIS SEMICOLON
;

<do_statement> ::= DO <block>
;

<for_loop> ::= FOR LEFT_PARANTHESIS <for_statement> RIGHT_PARANTHESIS <block>
;

<for_statement> ::= <initialize> SEMICOLON <boolean_expression> SEMICOLON <assignment>
;

<initialize> ::= <declaration>
| <assignment>
;

<condition> ::= <if_statement>
| <switch_statement>
;
<if_statement> ::= IF LEFT_PARANTHESIS <boolean_expression>
RIGHT_PARANTHESIS <block>
| IF LEFT_PARANTHESIS <function_call>
RIGHT_PARANTHESIS <block>
| <if_statement> ELIF LEFT_PARANTHESIS
<boolean_expression> RIGHT_PARANTHESIS <block>
| <if_statement> ELSE <block>
| IFNOT LEFT_PARANTHESIS <boolean_expression>
RIGHT_PARANTHESIS <block>
;
<boolean_expression> ::= <comparison>
| IDNTF
| BLN_FALSE
| BLN_TRUE
| NOT_OPT IDNTF
;
<comparison> ::= <boolean_expression> <relational_operators>
<compared>
| <boolean_expression> <boolean_operators> <compared>
| <function_call> <relational_operators> <compared>
;
<compared> ::= IDNTF
| BLN_FALSE
| BLN_TRUE
| NOT_OPT IDNTF
| <factor>
;
<relational_operators> ::= LESSEQ_OPT
| GREATEREQ_OPT
| NEQ_OPT
| EQ_OPT
| LESS_OPT
| GREATER_OPT
;
<boolean_operators> ::= AND_OPT
| OR_OPT
| XOR_OPT
| NOR_OPT
;

<switch_statement> ::= SWITCH LEFT_PARANTHESIS IDNTF RIGHT_PARANTHESIS LEFT_BRACKET <case_statement> RIGHT_BRACKET
| SWITCH LEFT_PARANTHESIS IDNTF LEFT_SQ_BRACKET IDNTF RIGHT_SQ_BRACKET RIGHT_PARANTHESIS LEFT_BRACKET <case_statement> RIGHT_BRACKET
;
<case_statement> ::= CASE IDNTF COLON <statement_list>
| CASE <factor> COLON <statement_list>
| <case_statement> CASE IDNTF COLON <statement_list>
| <case_statement> CASE <factor> COLON <statement_list>
| <case_statement> DEFAULT COLON <statement_list>
;
<data_type> ::= CHAR
| INT
| FLOAT
| BOOL
| BYTE
| STRING
| DOUBLE
| FILE
| DIRECTORY
| LONG_INT
| SHORT_INT
| LONG_FLOAT
| SHORT_FLOAT
| LONG_DOUBLE
| SHORT_DOUBLE
;
<flag> ::= UNDER_SCORE UNDER_SCORE IDNTF UNDER_SCORE UNDER_SCORE
<comment> ::= BACK_SLASH ASTRIC COMMENT ASTRIC BACK_SLASH
<empty> ::= /* empty */
;

- **Prerequisites:**
- **Extended Features:**

Functions:

1) print : prints the string value of the object passed as an argument. Prints th string literal value passed as argument.
2) list_contents : lists the contents (files, directories) of the passed directory argument.
3) get_size : returns the size of the (file, directory) passed as argument.
4) create : creates a new (file, directory) in the current directory.

5) copy : makes a copy of the argument passed (file, directory) in the current directory.
6) compare : compares two given files to check for equality, returns false or true;
7) connect : connects to the given server as an argument.
8) delete : deletes the given (file, directory).
9) rename : renames a given (file, directory) to a name passed as an argument.
10) move : moves a given (file, directory) to the given new path.
11) sort : sorts the given directory's files according to name ordering.
12) filter: returns all files of the specified type from the given directory.
13) back_up: makes a back-up copy of your local files to the cloud.
14) synchronize: performs a synchronization between local files and files on the cloud to update the cloud files with any new changes.
15) search: search in the given directory for the given file.
16) cd : changes the current directory to the given one.
17) download: downloads the given file using it's URL.
18) upload: uploads the specified (file, directory) to the cloud.
19) open: opens the give file.
20) properties: returns the properties of the given (file, directory).
21) move_back: moves one directory back to the previously opened one.
22) move_forward: moves one directory forward, in case of moving back then the need to move forward again.

**Data types:**

The language has 9 primitive data types:
1) int: a 4 byte integer.
2) float: a 4 byte for single precision floating point number.
3) bool: boolean value representation.
4) void: it represents the absence of a value.
5) char: a 1 byte ASCII character.
6) byte: a byte value represented by a char data type (each of 1 byte size)
7) double: an 8 byte sized for double precision floating point number.
8) long_int: a 4 byte integer value.
9) short int: a 2 byte integer value.

The language also has 4 composite date types:
1) array: it represents a sequence of values of the same data type.
2) string: it represents an array of characters.
3) file: represents the file type which has many attributes such as file name, size, path, etc…
4) directory: represents the directory type which has many attributes such as directory name, path, size, contents, etc…

**The language's structure/statements:**

The language follows a simple, forward logic to operate. One of two cases is recognized in each statement, the first case is a statement that requires a semicolon to end it, such as assignment, declaration, function calling or a go_to command along with the break and continue commands.
The second case is a statement that uses a code block starting with a left bracket and ending with a right one, such as loop and condition statements. Moreover, the notion of multi-line commenting is present in the language, where it starts with a (/*) and end of with (*/).
Defining functions is not an option in our language because as it is a file operation language it has all the necessary function built in and ready to be used.
The code block is defined as any number of statements withing opening and closing brackets.
The language presents a new and useful type of statement, that reminds us of the assembly type languages, which is the (go_to) command that searches for the specific flag in the following code statements to move the execution to the corresponding statement, jumping over statements before it.

**Error handling:**

the language detects any statement that doesn't meet its specifications and consider it as an error, then it exits execution and print out to the console a message indicating the cause of the error.