BBM301

1st_Project Report

Collaborators:

HAZEM ALABIAD 21403802

M.MALEK BABA 21403447

ABEDELJALIL JARJANAZY 21501041

Mohammed Ali 21403227

Subject: **Lex & Yacc**

- **Introduction:**

  - 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.

  Using makefile:

  Makefile is a program building tool which runs on Unix, Linux, and their flavors. It aids in simplifying building program executables that may need various modules. To determine how the modules need to be compiled or recompiled together, make takes the help of user-defined makefiles. This tutorial should enhance your knowledge about the structure and utility of makefile.

  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.

- **BNF grammar:**

```
program : statement_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
                  ;

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
                        | error
                        ;
```

```
factor_list : factor | factor_list COMMA factor;
RHS : arithmetic_expression
        | function_call
        | boolean_expression
        ;

function_call      :  BLTIN_PRINT LEFT_PARANTHESIS identifier
RIGHT_PARANTHESIS
                              | BLTIN_LIST_CONTENTS
LEFT_PARANTHESIS identifier RIGHT_PARANTHESIS
                              | BLTIN_GET_SIZE LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                              | BLTIN_CREATE LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                              | BLTIN_COPY LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                              | BLTIN_COMPARE LEFT_PARANTHESIS
identifier COMMA identifier RIGHT_PARANTHESIS
                              | BLTIN_CONNECT_TO
LEFT_PARANTHESIS identifier RIGHT_PARANTHESIS
                              | BLTIN_DELETE LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                              | BLTIN_RENAME LEFT_PARANTHESIS
identifier COMMA identifier RIGHT_PARANTHESIS
                              | BLTIN_MOVE LEFT_PARANTHESIS
identifier COMMA identifier RIGHT_PARANTHESIS
                              | BLTIN_SORT LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                              | BLTIN_FILTRE_FILES
LEFT_PARANTHESIS identifier RIGHT_PARANTHESIS
                              | BLTIN_BACK_UP LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                              | BLTIN_SYNCHRONIZE
LEFT_PARANTHESIS identifier RIGHT_PARANTHESIS
                              | BLTIN_SEARCH LEFT_PARANTHESIS
identifier COMMA identifier RIGHT_PARANTHESIS
                              | BLTIN_CD LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
```

```
                        | BLTIN_DOWNLOAD
LEFT_PARANTHESIS identifier RIGHT_PARANTHESIS
                        | BLTIN_UPLOAD LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                        | BLTIN_OPEN LEFT_PARANTHESIS
identifier RIGHT_PARANTHESIS
                        | BLTIN_PROPERTIES
LEFT_PARANTHESIS identifier RIGHT_PARANTHESIS
                        | BLTIN_MOVE_BACK
LEFT_PARANTHESIS empty RIGHT_PARANTHESIS
                        | BLTIN_MOVE_FORWARD
LEFT_PARANTHESIS empty RIGHT_PARANTHESIS
                        | error LEFT_PARANTHESIS IDNTF
RIGHT_PARANTHESIS
      | error LEFT_PARANTHESIS empty RIGHT_PARANTHESIS
                        ;

identifier :             IDNTF
                        |STR_LTRL
                        | error
                        ;

arithmetic_expression       : operand
                                    | arithmetic_expression ADD_OPT
operand
                                    | arithmetic_expression SUB_OPT
operand
                                    | arithmetic_expression
MULTIPLY_OPT operand
                                    | arithmetic_expression
DIVIDE_OPT operand
                                    | arithmetic_expression POW_OPT
operand
                                    | arithmetic_expression
MODE_OPT operand
                                    | arithmetic_expression
BITWISE_LEFTSHIFT operand
                                    | arithmetic_expression
BITWISE_RIGHTSHIFT operand
```

```
                                        | arithmetic_expression
BITWISE_AND operand
                                        | arithmetic_expression
BITWISE_OR operand
                                        | arithmetic_expression
BITWISE_XOR operand
                                        | arithmetic_expression
BITWISE_NOR operand
                                        | BITWISE_NOT
arithmetic_expression
                                       ;

operand : factor
    | IDNTF
         ;

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 PARANTHESIS block ;

do_while_loop : do_statement WHILE PARANTHESIS
SEMICOLON ;

do_statement : DO block ;

for_loop : FOR PARANTHESIS block ;

PARANTHESIS : LEFT_PARANTHESIS for_statement
RIGHT_PARANTHESIS
           | LEFT_PARANTHESIS boolean_expression
RIGHT_PARANTHESIS
                        | LEFT_PARANTHESIS function_call
RIGHT_PARANTHESIS
                        | LEFT_PARANTHESIS IDNTF
RIGHT_PARANTHESIS
                        | LEFT_PARANTHESIS error
                        ;

for_statement : declaration SEMICOLON boolean_expression
SEMICOLON assignment
                          ;


condition   : if_statement
                        | switch_statement
                        ;

if_statement      : IF PARANTHESIS block
                                | IFNOT PARANTHESIS block
                                | if_statement ELIF PARANTHESIS block
                                | if_statement ELSE block
                          ;

boolean_expression    : comparison
                              | BLN_FALSE
                              | BLN_TRUE
                              ;

comparison        : boolean_expression boolean_operators compared
                  | factor relational_operators factor
                  | IDNTF relational_operators factor
                  | factor relational_operators IDNTF
                  | IDNTF relational_operators IDNTF
                  ;

compared  : IDNTF
                  | BLN_FALSE
                  | BLN_TRUE
                  | NOT_OPT IDNTF
                  ;

relational_operators   : LESSEQ_OPT
                                  | GREATEREQ_OPT
                                  | NEQ_OPT
                                  | EQ_OPT
                                  | LESS_OPT
                                  | GREATER_OPT
                                  ;

boolean_operators      : AND_OPT
                              | OR_OPT
                              ;

switch_statement : SWITCH 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
                | FE
                | DIRECTORY
                | LONG_INT
                | SHORT_INT
                ;

flag :          UNDER_SCORE UNDER_SCORE IDNTF
UNDER_SCORE UNDER_SCORE
    | error
        ;
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.

Some difficulties we encountered:

The first problem we faced was how to represent boolean and byte types, as the language we are building on top, which is C, does not define those types, so we had to improvise and use the (char) type as byte type because they both are one byte long and (char) can thus replace byte. Moreover, the boolean type was also made by treating it as an (int), where the value zero is refereed to as false and the other values are true.

Another problem we faced was the error handling and reporting part. Adding the error option to the statement list didn't do the trick because some errors were still not caught, so we added the error option to the end of each statement and function usage, which gave us the ability to caught every error and name it based on the place it happened. The error handling issue made us rethink the whole process and consider the error to be an essential element in the language structure, that needs to be considered.

Resources:

http://www.delorie.com/gnu/docs/bison/bison_92.html
https://piazza.com/hacettepe.edu.tr/fall2017/bbm301/resources
https://www.youtube.com/watch?v=__-wUHG2rfM