LEX FILE DESCRIPTION

```
%{
#include <stdio.h>
int lineno;
%}
```

This is the definitions part for the lex file. It is not mandatory. I've included standard input/output

```
%%
START                   return START;
END                     return END;
\;                      return SEMICOLON;
integer                 return INT;
float                   return FLOAT;
string                  return STR;
and                     return AND_OPERATOR;
or                      return OR_OPERATOR;
\=                      return ASSIGNMENT;
is                      return EQUAL;
if                      return IF;
else                    return ELSE;
while                   return WHILE;
\(                      return LEFTPAR;
\)                      return RIGHTPAR;
\{                      return LEFT_CURLY;
\}                      return RIGHT_CURLY;
\+                      return PLUS;
\-                      return MINUS;
\*                      return MULTIPLICATE;
\/                      return DIVIDE;
\<                      return LESS_THAN;
\>                      return GREATER_THAN;
\!=                     return NOT_EQUAL;
\\n                     {extern int lineno; lineno++; return NEWLINE;}
[a-zA-Z]([a-zA-Z]|[0-9])*   return IDENTIFIER;
[+-]?[0-9]+             return INTNUMBER;
[+-]?[0-9]*(\.)?[0-9]+  return FLOATNUMBER;
\"(.)*\"                return STRING;
[ \t\n]                 ; //ignoring whitespace
```

In this part I've defined my tokens according to lex syntax to use in yacc file. This part is mandatory for a lex file.

```
%%

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

I created yywrap function to call when an end of file indication is encountered.

YACC FILE DESCRIPTION

```
%{
#include <stdio.h>
#include <math.h>
int yylex(void);
void yyerror()
{
        printf("THIS IS A CORRECT FORM OF YOUR PROGRAMMING LANGUAGE, ALI BORAN YILMAZ!\n");
}
%}
```

In the description part of my yacc file, i have defined yylex function which is my token generator function according to my grammars. And also i have defined yyerror function which is my error reporting function. If everything is OK according to the output of the yyparse function, yyerror function will print the statement in the parentheses.

```
%token INTNUMBER FLOATNUMBER STRING IDENTIFIER PLUS MINUS MULTIPLICATE DIVIDE
%token INT FLOAT STR
%token AND_OPERATOR OR_OPERATOR ASSIGNMENT EQUAL
%token IF ELSE WHILE
%token LEFTPAR RIGHTPAR
%token LEFT_CURLY RIGHT_CURLY
%token GREATER_THAN LESS_THAN NOT_EQUAL
%token SEMICOLON
%token NEWLINE START END
```

I've introduced my tokens to my yacc file here.

```
%%
program: START SEMICOLON stmts END SEMICOLON;

stmts:  stmts stmt | stmt;

stmt: non_block_stmt SEMICOLON | block_stmt;

non_block_stmt: assignment_stmt | var_declaration;

block_stmt: if_else_stmt | loop_stmt;

if_else_stmt: IF LEFTPAR conditional_expression RIGHTPAR body | IF LEFTPAR conditional_expression RIGHTPAR body ELSE body;

loop_stmt: while_stmt;

while_stmt: WHILE LEFTPAR conditional_expression RIGHTPAR body;

expression : conditional_expression;

conditional_expression: or_expression;

or_expression: and_expression | or_expression OR_OPERATOR and_expression;

and_expression: equality_expression | and_expression AND_OPERATOR equality_expression;
```

I've started to create my own grammer rules for my programming language using 'Backus-Naur Form' notation.

```
and_expression: equality_expression | and_expression AND_OPERATOR equality_expression;

equality_expression: relational_expression | equality_expression equality_operator relational_expression;

relational_expression: additive_expression | relational_expression relational_operator additive_expression;

additive_expression: multiplication_expression | additive_expression addition_operator multiplication_expression;

multiplication_expression: primary_expression | multiplication_expression multipication_operator primary_expression;

primary_expression: IDENTIFIER | primitive_type | LEFTPAR expression RIGHTPAR;

assignment_stmt: var_declaration ASSIGNMENT expression SEMICOLON | IDENTIFIER ASSIGNMENT expression SEMICOLON;

var_declaration: type_name IDENTIFIER SEMICOLON;

multipication_operator: MULTIPLICATE | DIVIDE;

addition_operator: PLUS | MINUS;

relational_operator:  LESS_THAN | GREATER_THAN;

equality_operator: EQUAL | NOT_EQUAL;

body: LEFT_CURLY stmts RIGHT_CURLY | LEFT_CURLY RIGHT_CURLY;

type_name: INT | FLOAT;

primitive_type: INTNUMBER | FLOATNUMBER;

%%
```

Defining grammar with BNF notation.

```
#include "lex.yy.c"

int main()
{
yyparse();

return 0;
}
```

In this part I've included the 'lex.yy.c' file which is generated after compiling my lex file. And finally, in main function I've called yyparse function. This function returns '0' if the input it parses is valid according to the given grammar rules. Otherwise, it returns '1'.