**RULES**

*Variable Declaration*: Variable names must start with a letter or an underscore and it can continue with any number of alphanumeric characters or underscores. There is only one type of variable in my programming language (number) and therefor there is no keyword to specify variable type. Variables can be assigned to numbers and to each others using ‘=’ (assign) operator. Ex: number = 5$

*Arithmatic Operators*: There are 4 arithmatic operators in my programming language. They are ‘+’ (add), ‘-‘ (subtract), ‘\*’ (multiply), ‘/’ (division).

*Logical Operators*: There are 2 logical operators: ‘and’ which returns true if both logical statements are true and ‘or’ which returns true if either or both of the logical statements are true.

*Comparison Operators*: There are 2 comparison operators: ‘==’ which returns if 2 number or variables are equal and ‘!=’ which returns if 2 number of variables are not equal.

*Operator Precedence*: '\*' > '/' > '+' > '-' > '==' > '!=' > ‘and’ > ‘or’

*Conditional Statement*: There is only one conditional structure. It can be called using the keyword ‘check (logical\_statement){}’. It executes the codeblock if the conditon holds true inside the parentheses.

*Loop Structure*: There is only one loop structures. It can be called using the keyword ‘do (number or variable){ statements }’. It executes the codeblock the number of times specified inside the parentheses.

*Program Structure*: It starts with declarations and continues with conditional, loop and arithmatic statements. Every assignment or aritmatic operation in a declaration must and with the character ‘$’. The program MUST end with the keyword ‘FINISH’. If there are no statements using

‘+’ -> ADD, ‘-‘ -> SUBTRACT, ‘\*’ -> MULTIPLY, ‘/’ -> DIVIDE, ‘and’ -> AND, ‘or’ -> OR, ‘==’ -> ISEQUAL, ‘!=’ -> NOTEQUAL, ‘=’ -> ASSIGN, ‘FINISH’ -> FINISH, ‘(‘ -> OPENP, ‘)’ -> CLOSEP, ‘{‘ -> OPENBLOCK, ‘}’ -> CLOSEBLOCK, ‘$’ -> ENDLINE

BNF RULES:

<program> ::= <statements> FINISH | FINISH | ε

<statements> ::= <statement> | <statements> <statement>

<statement> ::= OP | CHECK LOGIC OPENBLOCK <statements> CLOSEBLOCK | DO OPENP E CLOSEP OPENBLOCK <statements> CLOSEBLOCK

<OP> ::= VARIABLE ASSIGN E ENDLINE

<LOGIC> ::= <LOGIC> AND <LOGIC> | <LOGIC> OR <LOGIC> | E ISEQUAL T | E NOTEQUAL T | OPENP LOGIC CLOSEP

<E> ::= <E> ADD <T> | <E> SUBTRACT <T> | <T>

<T> ::= <T> MULTIPLY <F> | <T> DIVIDE <F> | <F>

<F> ::= OPENP <E> CLOSEP | SUBTRACT <F> | NUMBER | VARIABLE

EBNF RULES:

program = statements "FINISH" | FINISH | ;

statements = statement { statement } ;

statement = OP | "CHECK" LOGIC OPENBLOCK statements CLOSEBLOCK | "DO" OPENP E CLOSEP OPENBLOCK statements CLOSEBLOCK ;

OP = VARIABLE "ASSIGN" E "ENDLINE" ;

LOGIC = LOGIC AND LOGIC | LOGIC OR LOGIC | E ISEQUAL T | E NOTEQUAL T | OPENP LOGIC CLOSEP ;

E = E ADD T | E SUBTRACT T | T ;

T = T MULTIPLY F | T DIVIDE F | F ;

F = OPENP E CLOSEP | SUBTRACT F | NUMBER | VARIABLE ;

*Clarification*: I made this assignment with my own efforts. First i watched a lot of videos about lex and yacc, why they are used and examples of how they are used. Then i checked the lecture presentations about bnf and ebnf notations. After that i installed ‘flex’ and ‘bison’ on my computer and i wrote the lex and yacc files considering only the number datatype. After solving the contradiction problems i wrote a simple example program in my language and tested it. Originally i was planning to add ‘text’ datatype too but i couldn’t make it work with operators and i did not want to add it half-baked.

*Commands for windows 11*:

to compile:  
-> flex mpl.l  
-> bison -d mpl.y  
-> gcc -o mpl-syntax-checker lex.yy.c mpl.tab.c

to test:  
-> mpl-syntax-checker.exe < myprog.mpl