MATTOC
MatLang to C Translator Documentation

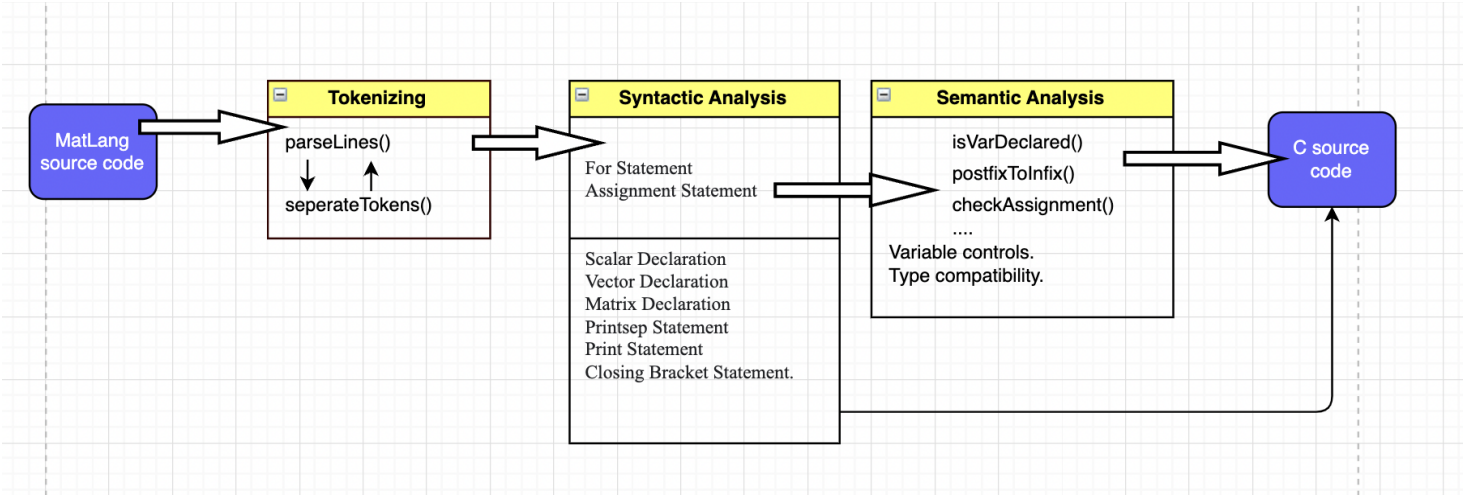
Ömer Faruk Çelik

What is MATTOC?

Mattoc is a translator that translates from MatLang to C. Mattoc reades the matlang file called file.mat, performs syntactic and semantic checks through that source code and generates the C code in file.c.

Processes

1-Tokenizing
2-Syntactic Analysis
3-Semantic Analysis
4-Generating Output
5-Error Handling



1- Tokenizing

While reading the input file line by line inside the main function, the line being read is sent to the parseLines function.

The string entering the parseLines function is sent to the seperateTokens function to add a space character between the tokens.

Tokens are separated by a space character by adding a space before and after the special characters : '[' ']' '{' '}' ',' '=' ':' '(' ')' '<' '>' '*' '-' '+'

String returned from seperateTokens function splitted into tokens by space demitters with the strtok function

Tokens added to the 'tokens' array are classified in the following sub-branches according to certain characteristics: Scalar Declaration, Vector Declaration, Matrix Declaration, Assignment Statement , Printsep Statement, For Statement , Print Statement , Closing Bracket Statement.

2-Syntactic Analysis

With the help of if else statements, the classified lines are checked to see if the syntax of the matlang language is complied with.

Expressions are checked in the expr_parser.c file with recursive descent parsing to see if the expression conforms to the bnf notation given below.

```
expr          →     term  moreterms

moreterms  →    + term moreterms
                |  - term moreterms
                |   null

term          →     factor  morefactors

morefactors →   *     factor  morefactors
                |  null

factor        →     ( expr )
                |  id[ expr, expr ]
                | id[ expr]
                | sqrt(expr)
                | choose(expr1,exr2,expr3,expr4)
                |  id
                |  num
                | tr(expr)
```

While checking the expression, the tokens are taken into the postfixTokens array in postfix notation.

Postfixed tokens are put in the correct output format after passing the necessary checks with the help of stack data structure in the postfixToInfix function.

3-Semantic Analysis

It is checked that the variables are already declared.

The compatibility of the expressions on the right and left side of the assignment is checked.

Type compatibility is checked in cases such as matrix multiplication, matrix sum, etc.

The type of variables in the relevant places in functions and statements is checked.

4-Generating Output

After the relevant line has passed the necessary checks, the c language equivalent of the line is printed to the output file.

5-Error Handling

When an error is encountered while reading a line, the next line is not passed and the error is printed to the terminal in Error (Line n) format.