2022 Spring

CmpE 230 – Project 1

Prepared by: **Zeynep Buse Aydın** 2019400066 & **Asude Ebrar Kızıloğlu** 2019400009

Content:

NOTE: We used **sqrt()** function in .c file. Please execute file with **–lm.**

ABSTRACT

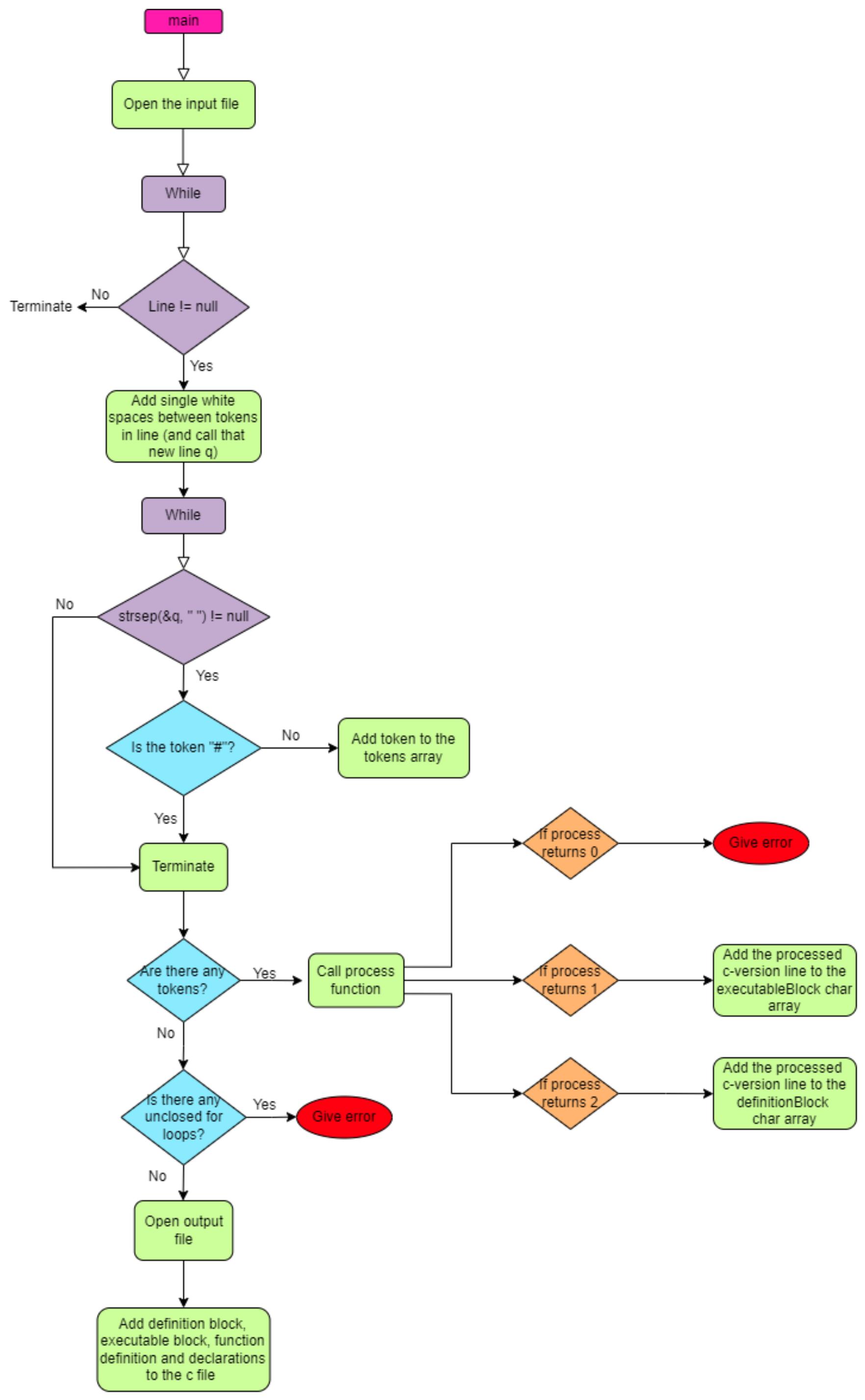
In this project, we implemented a translator of a sort to convert the given matlang code to C code to be able to execute it. Our code (main.c) reads lines from the input file matlang, whose path is given as argument, and process the code line by line while converting it to the C language. To achieve this, the lines, read from the input file, are first expanded by adding extra white spaces, with the function separateLine(). Then the edited line coming from this function is separated into tokens and these tokens are added to 2-dimensional char array **tokens** one-by-one. Then process() function is called to process these tokens. This function determines the type of the line (assignment / definition / print / printsep / for) and call the relevant functions accordingly.

In assignment and for lines, whenever an expression (expr) is encountered, the infix form of it is converted to the postfix format using expr – term – moreterms – factor – morefactors; and the resultant postfix format is read and processed and converted to the C language inside processStack() function.

In all the levels, generated C language lines are added to one of the following 4 char arrays: **definitionBlock**, **executableBlock**, **funcDefinitionBlock**, **funcDeclarationBlock**. These global variables store the C code.

If any invalid entry is found at any instant, the functions terminate with the returning value of 0, and in the main function, "Error" message is printed as a standard output, and no output C file is created.

If the input file does not contain any errors, at the very end of the main() function, the path of the output C file is generated from the path of the input file; and the content of the 4 global char arrays mentioned above is printed to the output file in order.


main
Open the input file
While
Line != null
No
Terminate
Yes
Add single white spaces between tokens in line (and call that new line q)
While
strsep(&q, " ") != null
No
Yes
Is the token "#"?
No
Add token to the tokens array
Yes
Terminate
Are there any tokens?
Yes
Call process function
If process returns 0
Give error
If process returns 1
Add the processed c-version line to the executableBlock char array
If process returns 2
Add the processed c-version line to the definitionBlock char array
No
Is there any unclosed for loops?
Yes
Give error
No
Open output file
Add definition block, executable block, function definition and declarations to the c file

FUNCTION EXPLANATIONS

*Below the detailed explanation of the mechanisms of the functions can be found, in the order of their apperances in **main.c** file.*

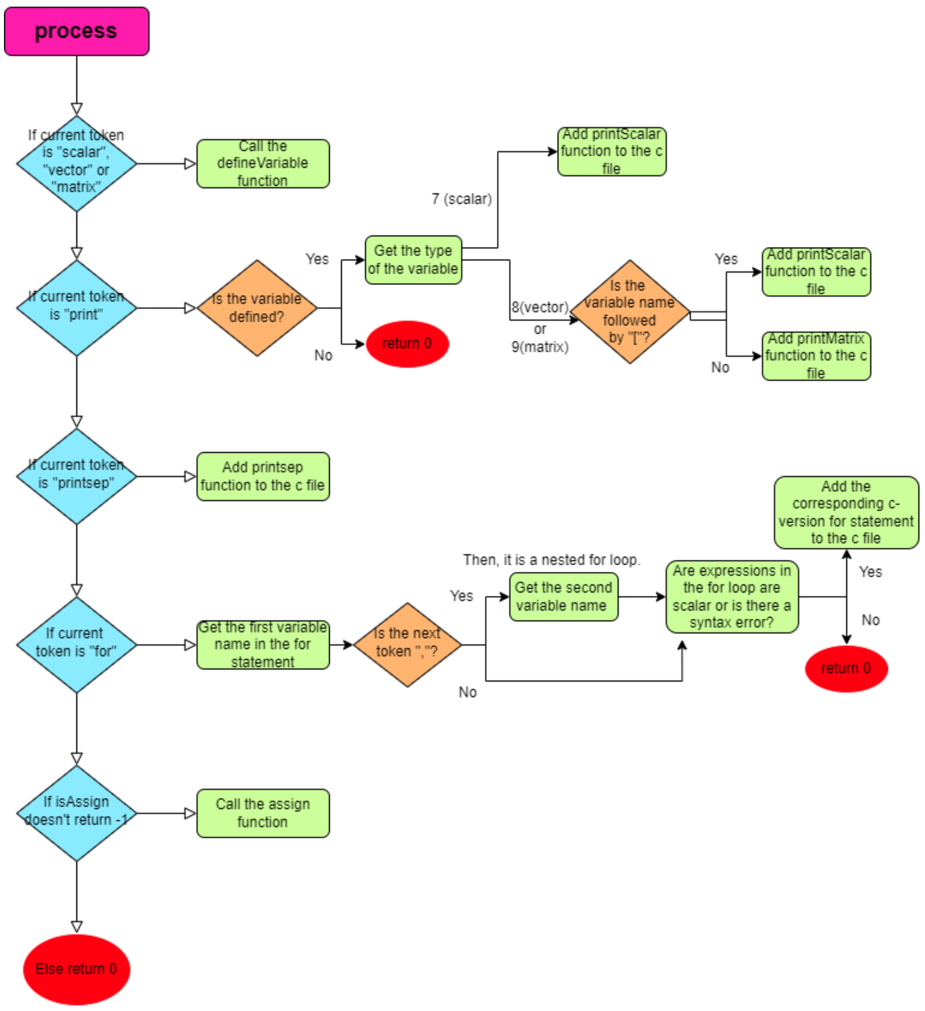
char *separateLine(char line1[], int length, bool nw):

This function is called in two locations. First is inside main() function with the parameter the line read from the input file. Second id inside processStack() function with the parameter the postfix format of an expression.
In either case, it takes char array as a parameter and reads it char-by-char. Also it creates a static char array **result**, to append the resultant string. Whenever it encounters an alphanumeric character, a point '.', or a dash '-', it directly appends to the **result**. If the character is something else, before adding it to **result**, it adds a white space, add returns **result** at the end. With this mechanism, this function facilitates to divide the line into tokens afterwards.

int process (int numTokens, char *res):

This function takes the number of tokens (numTokens) and an empty char pointer res.
process, checks whether the line is an assignment, definition, print, printsep or for line.

- The index of equal sign is found by **isAssign** function and stored in **assignment** integer. If the line is an assignment line, then **assignment** is not -1. Then the **assign** function is called with **numTokens**, **res** and **assignment**.
- If the first token in line is "scalar", "vector" or "matrix", then the line is a definition line. In that case **defineVariable** function is called.
- If the first token in line is "print", then we get the type of the id that will be printed. If its type is 7 (scalar) we add the "printScalar()" function with the id to **res.** If its type is 8 (vector) or 9 (matrix) we add the "printMatrix()" function with the id and its dimensions to **res**.
- If the first token in line is **"printsep"**, then we add "printsep()" function to **res.**
- If the first token in line is **"for"**, we first create the variables in for loop. Then we check if it is a nested for loop or not by looking at the current token. If the current token is "," then it is a nested for loop. Here we have two Boolean variables: **isInForLoop1** and **isInForLoop2**. These global variables are made true if it is a nested for loop. If it is a 1 dimensional for loop, then only isInForLoop is made true. The C version of for loop is added to **res**.


process
If current token is "scalar", "vector" or "matrix"
Call the defineVariable function
If current token is "print"
Is the variable defined?
Yes
No
Get the type of the variable
return 0
7 (scalar)
Add printScalar function to the c file
8(vector) or 9(matrix)
Is the variable name followed by "["?
Yes
No
Add printScalar function to the c file
Add printMatrix function to the c file
If current token is "printsep"
Add printsep function to the c file
If current token is "for"
Get the first variable name in the for statement
Is the next token ","?
Yes
No
Then, it is a nested for loop.
Get the second variable name
Are expressions in the for loop are scalar or is there a syntax error?
Yes
No
Add the corresponding c-version for statement to the c file
return 0
If isAssign doesn't return -1
Call the assign function
Else return 0

This function returns 1 after making all the necessary additions to char pointer **res** successfully. If an error is spotted, then the function directly returns 0 at that point.

int defineVariable (char *str):
First, gets the type of the variable from the first token in the line. Then checks if the variable name is valid by calling isValidVarName() function.
If the type is "vector", then we treat it as a n*1 matrix in C code. After making necessary checks, we add "float <var_name>[n][1];" to **str**.
If the type is "matrix", then we add "float <var_name>[m][n];" to **str**.
If the type is "scalar" we simply add "float <var_name>;" to **str**.
In the end we create an ID struct a and add it to **IDs** array.

int assign (int numTokens, char *res, int equalIndex):
First, gets the type of the left hand side (which is the tokens before the **equalIndex** index) of the assignment statement by calling **factor()** function. Then, gets the type of the right hand side of the statement by calling **expr()** function. Checks if these two values are equal, if they are not returns 0.

- If it is a scalar assignment, it simply adds the assignment line with ";" added to the end, to **res**.
- If it is a matrix assignment, it first checks if the next token is "{".
    - If it is "{", adds **res** an assignment of temp matrix with the given values. After that it also adds **copyMatrixtoMatrix()** function with temp variable and the real variable that will be assigned.
    - If the next token is not "{", adds **res** the lines that get the row and column of the temp variable and call the **copyMatrixtoMatrix()** function with these values, temp variable and the original matrix that will be assigned.

int processStack(char str[N], char *line, char *lasttoken):
This function takes the postfix string (str) coming from the expr() function, along with empty char pointers line and lasttoken.
Initially, calling separateLine() function, str is divided into tokens and with the help of a while loop, these tokens are added to the **stacktokens** array.
Then, one while loop is processed until all of the tokens in the **stacktokens** array is processed. Since the number of the tokens is stored in the variable **numstacktokens**, and current number of tokens processed is stored in the variable **stackcur**, while loop terminates when (**stackcur** >= **numstacktokens**) and **stackcur** is incremented by 1 at the end of each iteration of the loop.

Variables used in this function are as follows:

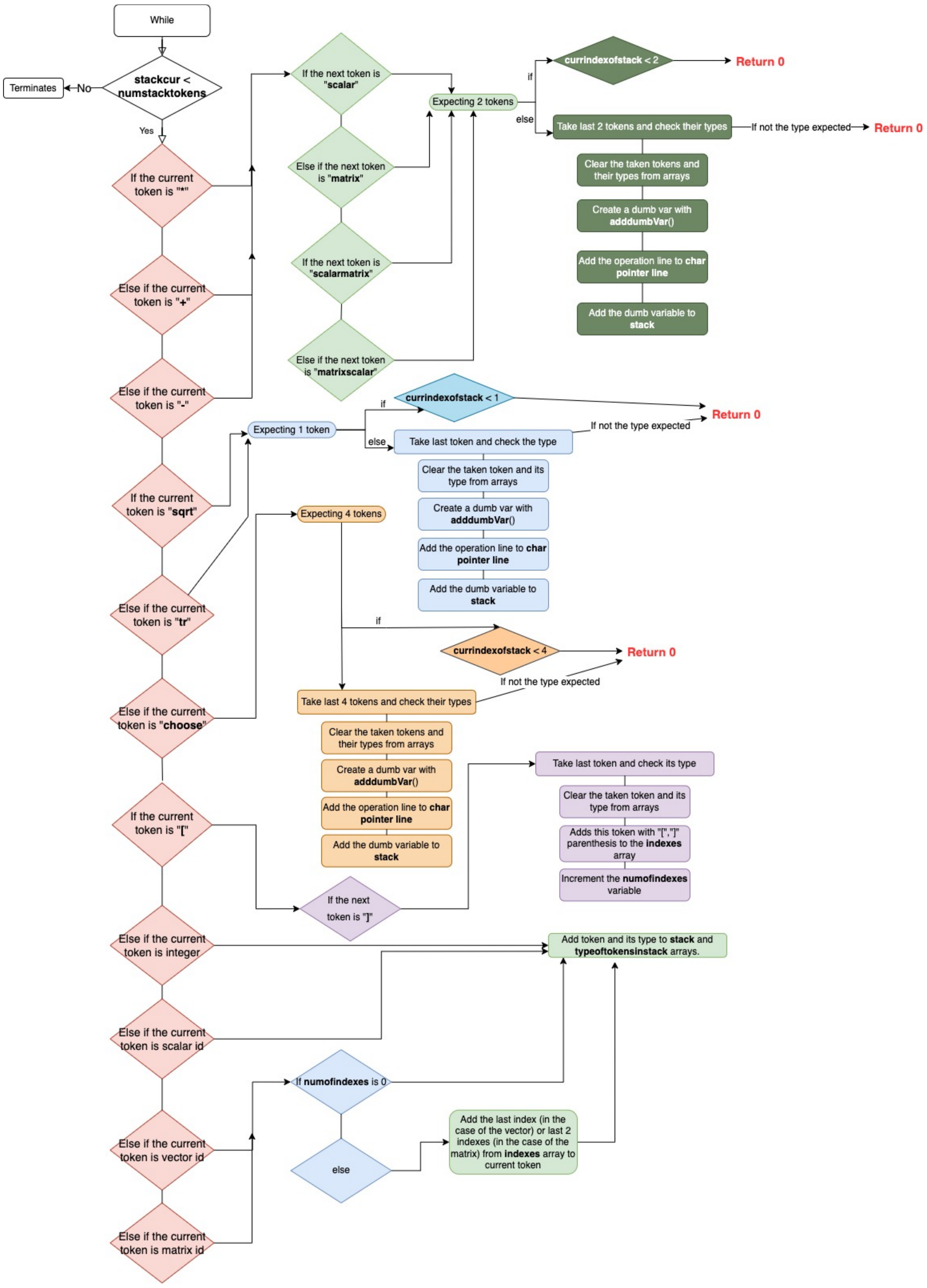
- In this function, char pointer line which is the parameter is filled with the corresponding code lines to be written to the output C file.
- Global 2-dimensional char array **stacktokens** is used to store the tokens of the postfix string.
- Global int **numstacktokens** stores the number of tokens in **stacktokens**.
- Global int **stackcur** keeps the current index of the token being processed.
- Global 2-dimensional char array **stack** is used to store scalars or matrices to be chosen as operands.
- Global int array **typeoftokensinstack** is used to store the types of tokens stores in **stack**.
- Global int **currentindexofstack** stores the current number of tokens in **stack** array at each instant.
- Global char arrays **token1**, **token2**, **token3**, **token4** are used to copy the tokens from **stack** to be added to char pointer line.
- Global 2-dimensional char array **indexes** is used to temporarily store the indexes until their id's token is processed. When the id token is read, according to the value of **numofindexes,** the content of **indexes** is added to the id's token, and they are together added to **stack** array.

- Global int **numofindexes** stores the current number of indexes stored in **indexes**.

Inside the while loop, according to the current tokens content, the necessary action is taken as follows:

- If the current token is "*" or "+" or "-" or “sqrt” or “tr” or “choose” or “[”, the necessary number of tokens (according to the type of the operator) are taken from **stack** array, with their types from **typeoftokensinstack** array; and relative lines of code are added to the char pointer **line**.
- If the current token is either integer or a scalar id (this is checked by the function **isID()**), token is directly added to **stack** array, as well as its type being added to **typeoftokensinstack** array.
- If the current token is either vector or matrix id, with **numofindexes**=0, token is again directly added to **stack** array, as well as its type being added to **typeoftokensinstack** array.
- If the current token is either vector or matrix id, with **numofindexes** > 0, last index (if it is a vector) or last 2 indexes (if it is a matrix) copied from **indexes** array to the current token. Then, token is added to **stack** array, as well as its type being added to **typeoftokensinstack** array.

Below, the detailed chart of the inner mechanism of this while loops is placed.
While loop terminates when all of the tokens from **stacktokens** array is processed.
Finally, the last token remained in the **stack** array is copied to char pointer **lasttoken** parameter, end function returns with value 1.
During the process, if any error is encountered at any instant, the function immediately terminates with the return value 0 (this can be observed from the chart below).


While
stackcur < numstacktokens
No
Terminates
Yes
If the current token is "*"
Else if the current token is "+"
Else if the current token is "-"
If the current token is "sqrt"
Else if the current token is "tr"
Else if the current token is "choose"
If the current token is "["
Else if the current token is integer
Else if the current token is scalar id
Else if the current token is vector id
Else if the current token is matrix id
If the next token is "scalar"
Else if the next token is "matrix"
If the next token is "scalarmatrix"
Else if the next token is "matrixscalar"
Expecting 2 tokens
if
currindexofstack < 2
Return 0
else
Take last 2 tokens and check their types
If not the type expected
Return 0
Clear the taken tokens and their types from arrays
Create a dumb var with adddumbVar()
Add the operation line to char pointer line
Add the dumb variable to stack
Expecting 1 token
if
currindexofstack < 1
Return 0
else
Take last token and check the type
If not the type expected
Clear the taken token and its type from arrays
Create a dumb var with adddumbVar()
Add the operation line to char pointer line
Add the dumb variable to stack
Expecting 4 tokens
if
currindexofstack < 4
Return 0
If not the type expected
Take last 4 tokens and check their types
Clear the taken tokens and their types from arrays
Create a dumb var with adddumbVar()
Add the operation line to char pointer line
Add the dumb variable to stack
If the next token is "]"
Take last token and check its type
Clear the taken token and its type from arrays
Adds this token with "[","]" parenthesis to the indexes array
Increment the numofindexes variable
Add token and its type to stack and typeoftokensinstack arrays.
If numofindexes is 0
else
Add the last index (in the case of the vector) or last 2 indexes (in the case of the matrix) from indexes array to current token

int isAssign ():
This function looks for a "=" in the line. If it finds "=", it returns the index of it, else it returns -1. (Which means it is not an assignment line.)

## For the following 5 functions: they return 0 if any error occers, 1 if the type is scalar, 2 if the type is matrix (including vectors here, with adding the second dimension as [1]).
int expr (char * str):
With respect to the BNF Grammar of the expressions (we used the Grammar the Professor provided), term() and moreterms() functions are called, with parameters **str1** and **str2** (char arrays) respectively. Now, **str1** includes the postfix format coming from term() and **str2** includes the postfix format coming from moreterms(). Also, these functions' returning values are checked. If either is 0, 0 is returned. If moreterms() return 1, the value of term() is returned. Otherwise, if moreterms()'s returned value differs from term()'s returned value, due to the type-incompatibility, 0 is returned. **str2** is appended to **str1**, and **str1** is appended to **str**, which is the parameter of the function.

int term (char *):
Very similar to expr(), this time factor() and morefactors() functions are called respectively, and according to their returning values, the necessary action (similar to expr()) is taken.

int moreterms (char *):
If the next token is not "-" or "+", function returns integer 1 without appending anything to the parameter.
Otherwise, very similar to expr(), term() and moreterms() functions are called respectively, and according to their returning values, the necessary action is taken.

int factor (char *str):
First checks the current token.

- If the current token is an integer:
  - Adds **str** the current token.
- If the current token is **"("**:
  - Calls expr() function. Adds **str** the char array that was modified by expr().
- If the current token is **"tr"**:
  - Calls expr() function for the expression inside. Adds **str** the post fix version of **tr**.
- If the current token is **"sqrt"**:
  - Calls expr() function for the expression inside. Adds **str** the post fix version of **sqrt**.
- If the current token is "**choose**":
  - Calls expr() function 4 times for the 4 expressions inside. Adds **str** the post fix version of **choose**.
- If the current token is a variable name:
  - If it is a scalar adds directly the name to **str**.
  - If it is a vector followed by [<int>], adds str "<int> [] name".
    - If it is not followed by "[", adds directly the name to **str**.
  - If it is matrix and followed by [<int1>, <int2>], adds str "<int1> [],<int2> [] name".
    - If it is not followed by "[", adds directly the name to **str**.

int morefactors (char *):
If the next token is not "*", function returns integer 1 without appending anything to the parameter.
Otherwise, very similar to term(), factor() and morefactors() functions are called respectively, and according to their returning values, the necessary action is taken.

int is_integer (char *):
Returns a non-zero integer if given char pointer points to an array of integer characters, returns 0 otherwise.

int isNumber (char s[]):
Returns 0 if s is not a number (float and integer), 1 otherwise.

int isID (char *):
Checks whether a char array is a variable name or not by looking for it in the IDs array. Returns -1 if it doesn't exist, return the index of the name otherwise.

int findComma():
Finds and returns the index of the "," token in the tokens array.

int findColon():
Finds and returns the index of the ":" token in the tokens array.

void editVarName (char *old, char *new):
Gets the **old** name in the post fix notation and changes it in a way that can be used in a .c file. The new notation is stored in the new **char** array.

int isValidVarName (char *name):
This function checks if the given name is a valid variable name or not. Returns 1 if valid, 0 otherwise.
If the name is scalar, vector, matrix, tr, sqrt, choose, for, print or printsep then the name Is not valid.
A valid name can start with "_" or an alphabetic character, the rest of the name should contain only alphanumerical characters.

int adddumbvar (int typeofdumbvar, int varRow, int varColumn):
Creates a dumb variable named "buse<int>" to be used in the .c file. The variable has **varRow** number of rows and **varColumn** number of columns.

void addFunctionDefinitions():
Adds function definitions to **funcDefinitionBlock**.

void addFunctionDeclarations():
Adds function declarations to **funcDeclarationBlock**.

C FILE FUNCTIONS

The matrix functions in .c file are all starts by calling **allocateMatrix()** function. **allocateMatrix()** takes an array of pointers, number of rows and number of columns as parameter. It allocates a part in the memory with the given sizes (row*column). And makes the pointers in the given array to the rows of matrix. Since this operation is done with calloc, the initial values are 0.
The results of the matrix functions are then assigned to the wanted matrix by copying the values from the dumb matrix with **copyMatrixtoMatrix()** function.