Ömer Faruk Ünal
2019400048

Eren Dönmez
2019400021

MatLang Translator

Usage

- Firstly, in order to create an executable for the project, execute the command `make` in the root directory of the project. This creates an executable named `matlang2c`.
- Afterwards, to translate a `.mat` file to C, execute `./matlang2c file.mat` (specifying the directory of the `.mat` file). During translation, if an error is encountered in the `.mat` file, then the line number specifying the location of the error is output to the console. Otherwise, the translated version of the `.mat` file is created in the same directory as the `.mat` file. The name of the translated file is the same as the `.mat` file except for its extension, which is `.c`.
- To compile and execute the translated version of the `.mat` file, execute `gcc file.c -lm -o prog && ./prog` (specifying the directory of the `.c` file). Here, the `-lm` flag must be used because the .c file makes use of functions from the math library.

Description

- This program converts a file in the MatLang language to the C language and outputs the result in a C source file.
- The program takes a file with `.mat` extension that is written in the MatLang language as its input. It examines this file by performing syntax checks and type checks. If there is a syntax error in the `.mat` file, the program will raise an error. Otherwise, it will output the C language equivalent of this file.

Project Outline

This project was written in the C language and it consists of five `.c` files and one `.h` file. These files are:

- `main.c`: Reads the MatLang language from the input file line by line and writes the corresponding C equivalent to the out file.
- `eval.c`: Evaluates and translates the line and also performs type checking operations.
- `matlang_functions.c`: Includes the declarations of necessary MatLang operations and functions, which are addition, subtraction, multiplication, `print`, `printSep`, `for` loop, `sqrt`, `choose`, and `tr`.
- `assignment_declaration.c`: Includes the declarations of scalar and matrix assignment and declaration functions.
- `globals.c`: Includes the declarations and initializations of the global variables used in the program.
- `structs.h`: Includes the definitions of scalar and matrix structs used in the project.

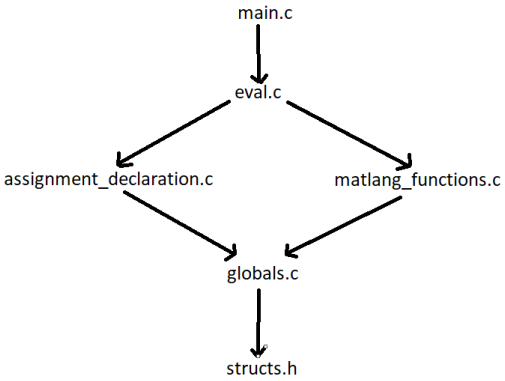

Figure 1

The project folder also includes a `Makefile`, which allows compiling and linking all source files and creating an executable with a single command. While writing the `Makefile`, the dependencies of the source files were taken into consideration. The dependency tree of the project can be seen in Figure 1.

Ömer Faruk Ünal
2019400048

Eren Dönmez
2019400021

Journey of a MatLang Code

- First of all, the input `.mat` file and the output `.c` file are opened.
- At the beginning of the `.c` file, function definitions that can be used throughout the program such as `choose` and `tr` are written.
- The given input is examined line by line and is separated into tokens.
- By looking at the first token of a line, the type of statement on that line (i.e., declaration, assignment, `print`, `for` loop beginning, or `for` loop ending) is decided.
- After understanding the classification of a line, the translation operations are performed according to the expressions using the functions we have defined before.
- If there is a mathematical expression inside the line, then:
    - Firstly, the expression is converted into postfix notation using a modified version of Dijkstra's Shunting-yard Algorithm. The algorithm is modified in a way that allows multiple-argument functions (in our case, matrix indexing, where a matrix can be considered a function that can have either 1 or 2 arguments) to be present.
    - Afterwards, type checking and translation is simultaneously completed. During this process, two stacks are utilized, where one keeps track of the dimensions of expressions while the other stores the translated version of each expression. In our implementation, scalar variables are considered 0x0 matrices, in order to increase consistency and conciseness in the code.
- Statements whose translations to the C language have been completed are simultaneously written to the output `.c` file.
- If, at any point, a compile-time error is encountered, then a message specifying the line number of the error is output to the console. Then, the output `.c` file is closed and deleted.

Used Technologies and Structures

As an IDE, both members of the group used CLion, since it has many functionalities that make debugging easier. Since this is a group project, we used Git to be able to work on the same files asynchronously and have the opportunity to review and comment on each other's code while staying up to date about the changelog. As C libraries; <stdio.h>, <stdlib.h>, <string.h>, and <ctype.h> were used. Naturally, pointers and arrays were used extensively throughout the program. In addition, the struct structure was used in order to access the properties of and process matrices easily.