CMPE230 SYSTEMS PROGRAMMING SPRING 2022

PROJECT 1 DOCUMENTATION

Bahadır Gezer – 2020400039
Simar Achmet Kechagia - 2020400378

Summary:

Our program behaves like a real compiler. The program reads the input file line by line. Every line is then tokenized, parsed and then translated.

We have implemented our own vector and stack header files. The vector is like the resizable vector in C++ and the stack is the standard stack data structure. Since we don't know how many Token structs there are the vector is used to hold tokens. This important Vector is called 'tokens'. Each token is a struct Token structure. This struct holds the type and value of the token. The value char array is the string which the token is named after, and the type is the type which we arbitrarily assigned. The lookup table for types are below.

For every line, after tokenization, tokens Vector is filled with Token structs and these structs does not change places while the current line being read.

Important Notes:

Both compilations must be done with the -lm option. This is to ensure that the <math.h> library is linked properly.

Important Functions:

void tokenizer();

This function tokenizes the expressions. No attribute is given to the tokens, that is done by the parser function.

void parser();

This function parses the expression. It gives meaning to the tokens. More explicitly it will assign types to tokens. It calls get_expression() to get the index of the expected expression. Then it calls expression() to check if the expression syntax is correct. This expression() function uses infix to postfix conversion to check the syntax. This is done with the help of our stack header file.

void output_generator(File *out);

This function takes the Vector tokens, which was attributed in the parser() function, and translates the tokens to appropriate C functions. It behaves just like the parser() in its recursive calls.

Token Types Lookup Table:

0 -> scalar keyword

1 -> vector keyword

2 -> matrix keyword

3 -> function keyword (tr, sqrt, choose)

4 -> for keyword

5 -> left paranthesis

6 -> right paranthesis

7 -> left square brace

8 -> right square brace

9 -> left curly brace

10 -> right curly brace

11 -> equals

12 -> multiplication

13 -> addition

14 -> subtraction

15 -> comma

16 -> colon

17 -> undetermined

18 -> comment

19 -> scalar variable

20 -> vector variable

21 -> matrix variable

22 -> for variable

23 -> print function keyword (print, printsep)

24 -> in keyword

25 -> indexed matrix

26 -> indexed vector

27 -> scalar function keyword (choose, sqrt, tr)

28 -> matrix function keyword (tr)