YACC:

It works by reading from the file and receiving input from the terminal.

Compilation is done with Makefile.

run:

yacc -d -o gpp_interpreter.c gpp_interpreter.y

flex -o gpp_lexer.c gpp_lexer.l

gcc gpp_lexer.c gpp_interpreter.c -o main -ll -w

./main

input:

./main input.gpp

output:

./main input.gpp > output.txt

1) make run takes input from the terminal

2) make input reads from the file.

3) make output reads from the input file and writes the output to the output file.

Function struct was used to define functions.

The Valueb struct was used to keep the number of Valuebs.

4 processes are working correctly.

In the function definition, the function without parameters returns the incoming expression.

In a single-parameter function, for example (def sum x (+ x 3b2)), the number 3b2 was kept in memory (with the temp2 value). Then, when the function was called, the value entered with 3b2 was added.

The two-parameter function processed the entered values according to the type of the operator.

Comment token and exit token were added to the input grammar.

Gpp.h and gpp.c files contain the necessary functions and necessary structures.

For example, the valueb structure holds num and denom values. We split the incoming string and get num and denom values.

The function struct contains the name of the function and what action it will perform.

For Valueb, there are add, sub, mul, div, converting, gcd and simplify functions.

Converting turns the string into valueb form.

There are get and set functions for the function struct.

Set function makes definition. The get function returns the function if it exists.

input:

;; helloworld.g++

(+1b1 2b1)

(def sum x y (+ x y))

(sum 44b1 6b1)

(*4b1 6b1)

(exit)

Output:

Result: COMMENT

Result: 3b1

#function

Result: 50b1

Result: 24b1

LISP:

We compile the Lips code in 2 ways.

1) clisp gpp_interpreter.lisp => reading from terminal

2) clisp gpp_interpreter.lisp input.gpp => reading from file

For Lisp code, 4 operations run successfully.

Cfg rules were made. Start input exit exp. Function part is missing.

There are two files. Lexer file contains token list.

This token list is used in Interpreter.

The keyword_find function searches the keywordList.

The operator_find function searches the operatorList.