CSE 341 PROGRAMMING LANGUAGES

HW3 DOCUMENTATION

BARIŞ BATUHAN BOLAT

210104004029

Definitions:

- **%union**: Defines a union for the semantic values of tokens. The union includes either a fraction value (**value**) or an identifier (**name**).
- **%token**: Defines tokens used in the grammar. For example, **VALUEF** represents a fraction value, **IDENTIFIER** represents an identifier, and so on.
- **%type**: Specifies the type of the semantic value for certain non-terminals in the grammar.

1. Data Structures:

    - **fraction**: A structure to represent fractions with numerator (**num**) and denominator (**den**).
    - **Entry**: Represents an entry in the symbol table with an identifier, type (variable or function), and a fraction value.
    - **Table**: A symbol table to store entries.
    - **Type**: An enumeration representing the types of entries (function or variable).

2. Functions:

    - **tableCreate**: Creates a new symbol table.
    - **tableFree**: Frees the memory allocated for the symbol table.
    - **increase**: Increases the capacity of the symbol table.
    - **contains**: Checks if an entry with a given type and identifier exists in the symbol table.
    - **tableGet**: Retrieves an entry from the symbol table.
    - **tableDef**: Defines a new entry in the symbol table.
    - **createFraction**: Creates a fraction with given numerator and denominator.
    - **simplify**: Simplifies a fraction by finding the greatest common divisor.
    - **add, subtract, multiply, divide**: Functions to perform basic arithmetic operations on fractions.

3. Grammar Rules:

    - **START, EXP, EXPLIST, FUNCTION, FCALL**: Non-terminals representing the structure of the input.

Resolutions:

- The language supports basic arithmetic operations on fractions.
- It allows users to define functions and variables.
- The code maintains a symbol table to store and retrieve function and variable information.
- The **yyparse()** function initiates the parsing process.
- The program reads input from either a file or standard input and writes output to a file named "output.txt".

Note 1 : Only the first part of homework is implemented (Flex and Yacc).
Note 2 : Function creation is working but Function Call is not working. It gives Syntax Error.

HOW TO RUN

- make
- ./gpp_interpreter input.txt (outputs will be in “output.txt”)
- /gpp_interpreter (read from shell)