Programming Languages 2'nd Homework Report

A. interpreter.lisp

The code begins by loading the formatted first assignment, lexer.lisp, assumed to contain lexical analysis functions. It then defines several functions for parsing and performing basic arithmetic operations on fractions. The valuef_parser function takes an input string and converts it into an integer value, interpreting each character as a digit. The remaining functions (valuef_summation, valuef_subtraction, valuef_multiplication, and valuef_division) perform addition, subtraction, multiplication, and division on fractions, respectively. These functions internally use valuef_parser to extract numerator and denominator values and handle the operations accordingly.

The expression_evaluation function evaluates an expression represented as a list with an operator and two operands. It recursively evaluates sub-expressions and performs the corresponding arithmetic operation. The keyword_split function assists in splitting a keyword list into operands and remaining parts, supporting nested expressions.

Finally, the main code block uses the keyword_split function to split a given tokenType into an expression and remaining parts. It then evaluates the expression using expression_evaluation and prints the result to the standard output.

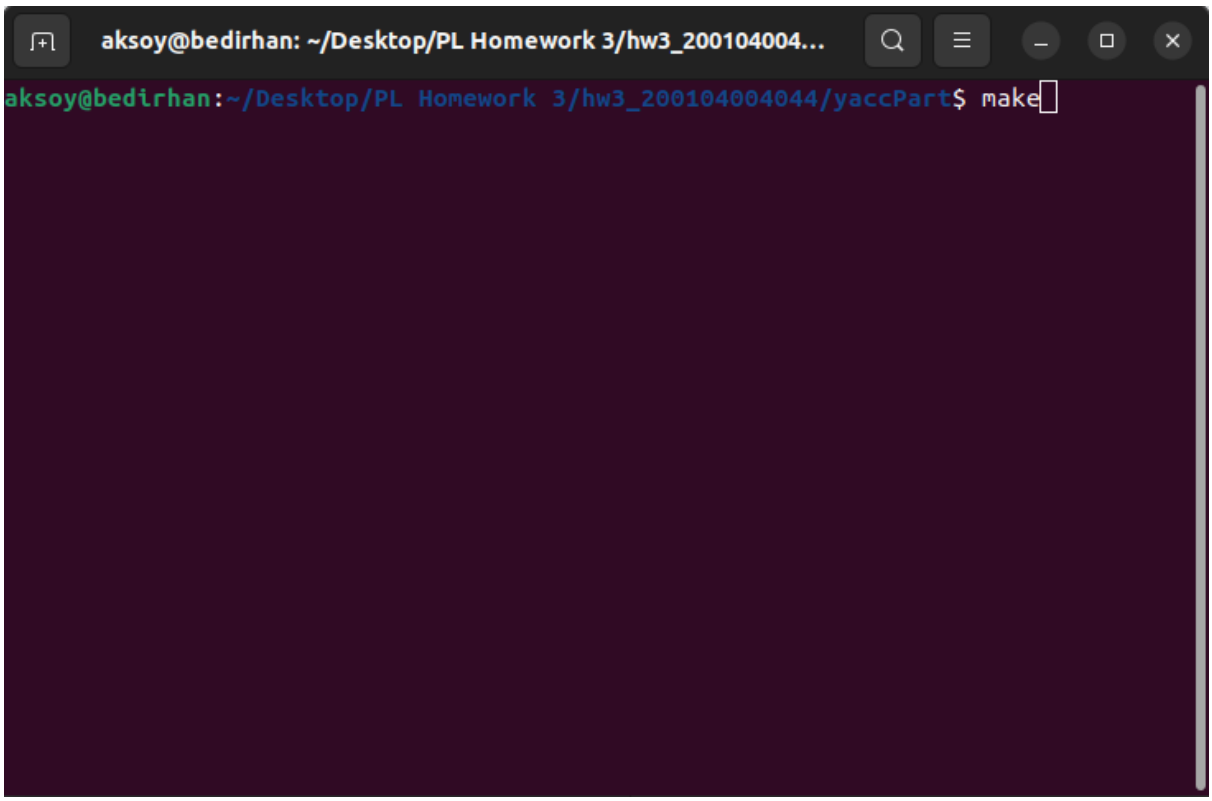
B. interpreter.y

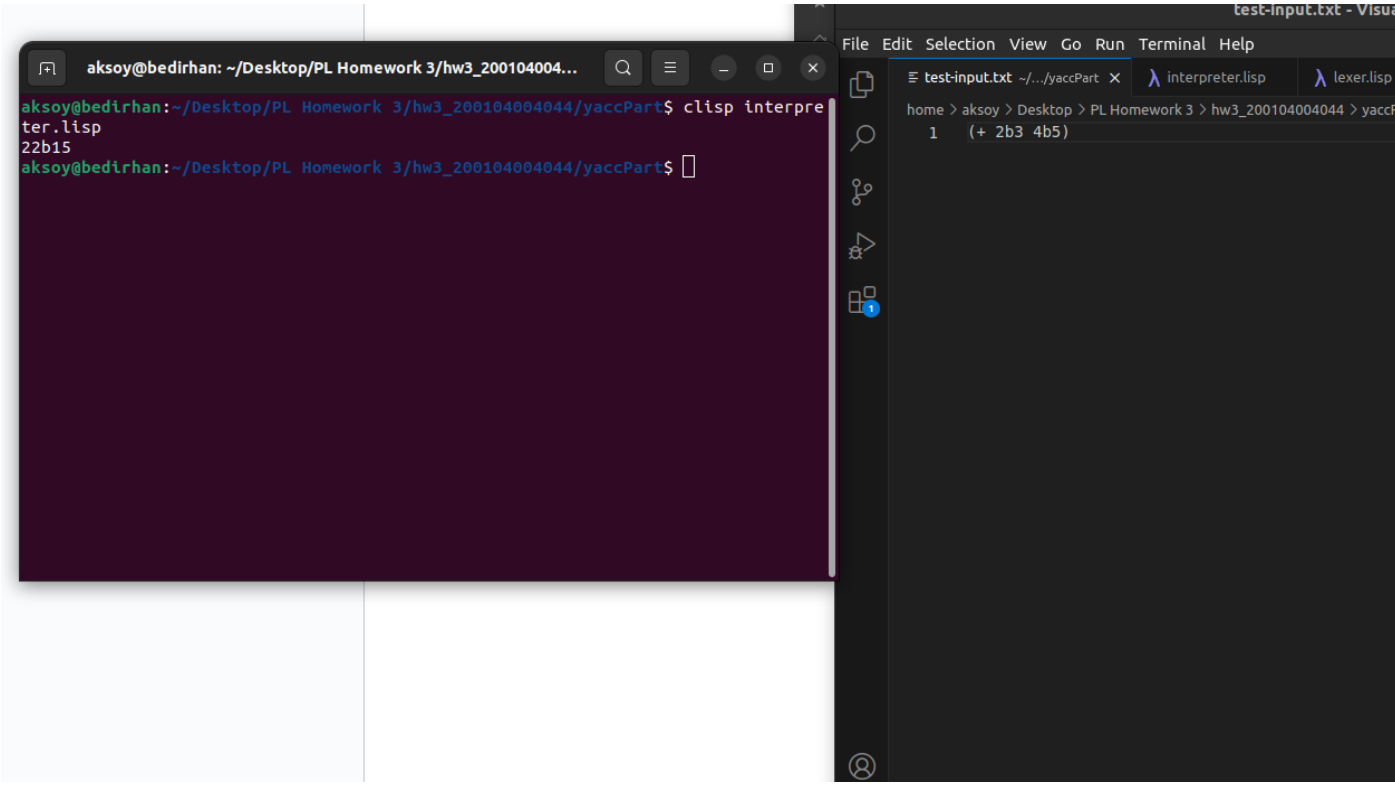
This program defines a simple language for arithmetic expressions and function definitions and implements an interpreter for it. The language supports basic arithmetic operations (+, -, *, /), function definitions, and function calls. The program uses a symbolic table to store function definitions and evaluates expressions based on an abstract syntax tree.

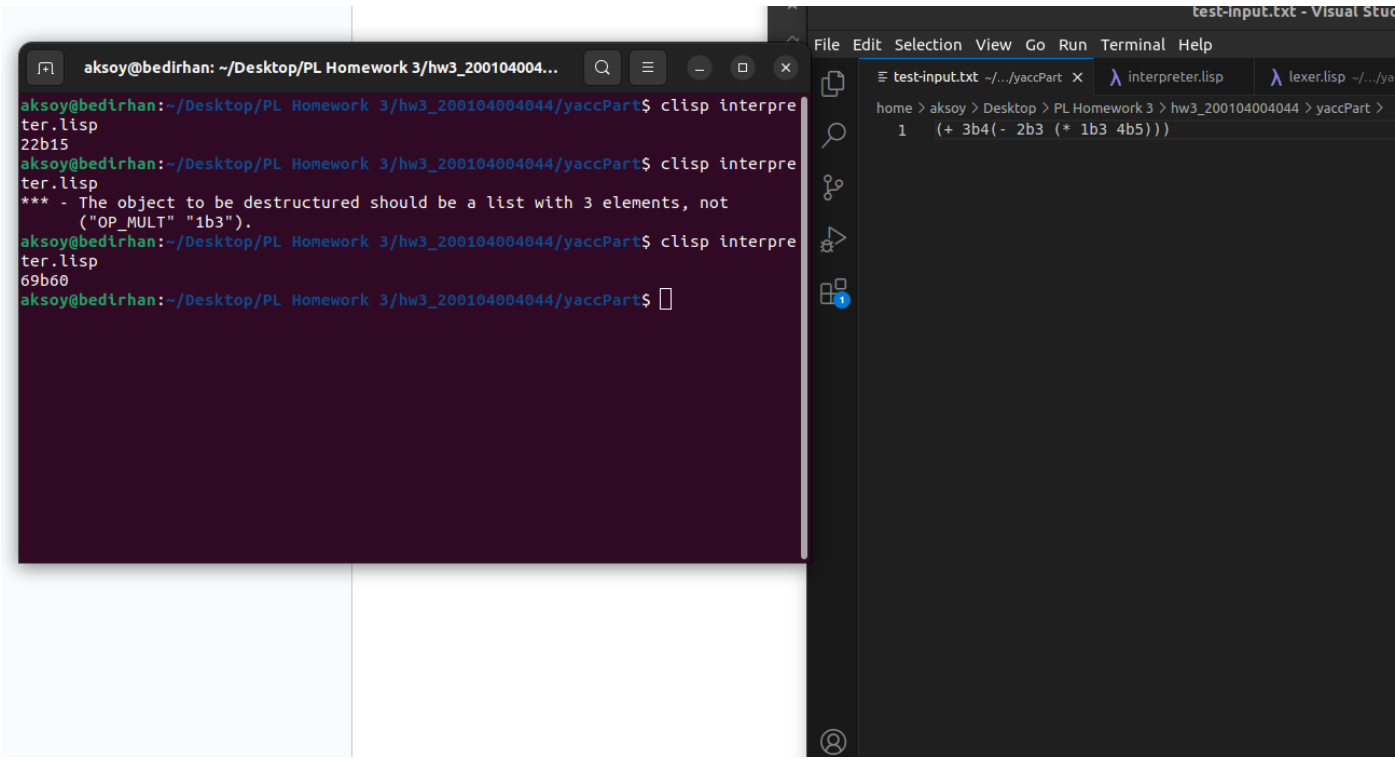
The abstract syntax tree (AST) is defined using a structure abs_stx_tree_node, which represents different types of nodes in the AST. The main types include VALUEF_NODE for numeric values, ID_NODE for identifiers, and various binary operation nodes (ADD_NODE, SUBTRACT_NODE, MULTIPLY_NODE, DIVIDE_NODE). Function calls are represented by FUNC_CALL_NODE. The program also defines structures for function definitions (funct_def) and a symbol table (symbol_table) to store functions.

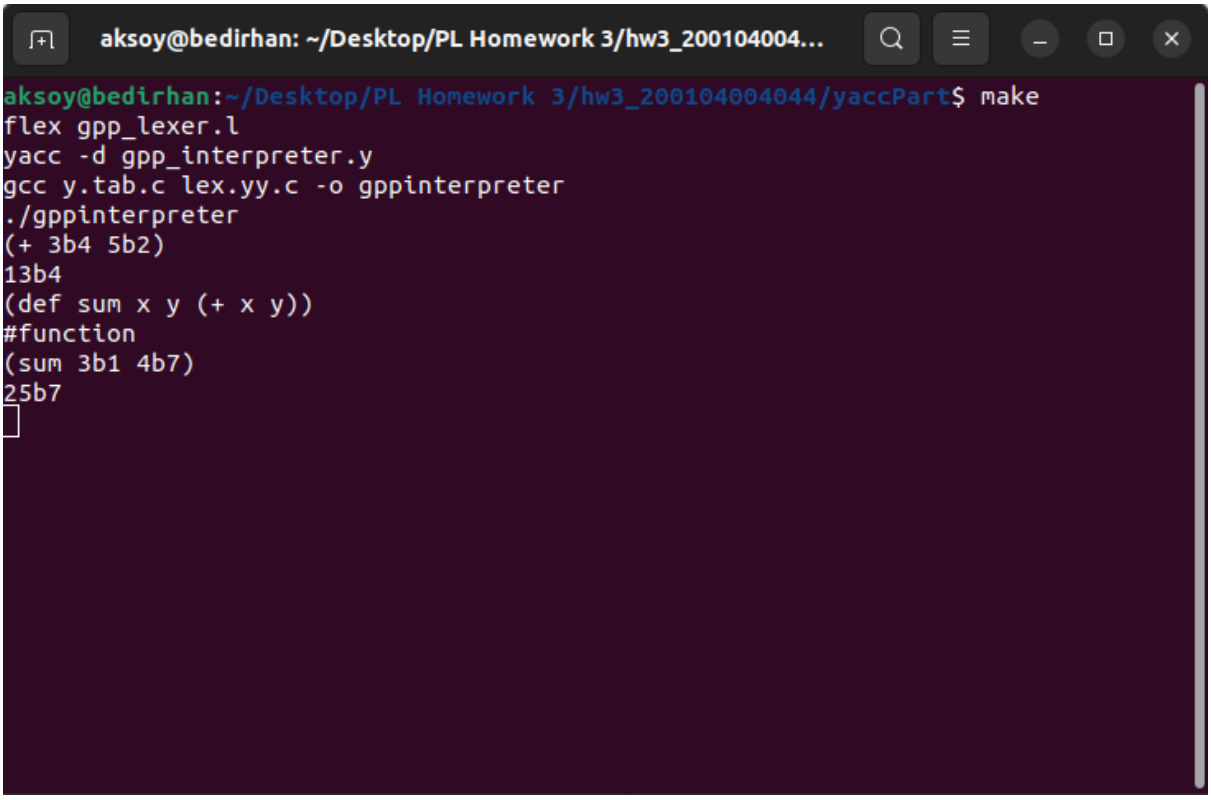
The evaluate function recursively traverses the AST and computes the result of expressions. It handles numeric values, identifiers, binary operations, and function calls. The program uses a separate function, VALUEF_operations, to perform arithmetic operations on valuef values. Additionally, the program ensures proper memory management for dynamically allocated structures and values.

The program uses the lexer from the previous homework to tokenize input and a parser to generate the abstract syntax tree.


aksoy@bedirhan: ~/Desktop/PL Homework 3/hw3_200104004...
aksoy@bedirhan:~/Desktop/PL Homework 3/hw3_200104004044/yaccPart$ make


aksoy@bedirhan: ~/Desktop/PL Homework 3/hw3_200104004...
aksoy@bedirhan:~/Desktop/PL Homework 3/hw3_200104004044/yaccPart$ clisp interpre
ter.lisp
22b15
aksoy@bedirhan:~/Desktop/PL Homework 3/hw3_200104004044/yaccPart$
File Edit Selection View Go Run Terminal Help
test-input.txt ~/.../yaccPart
interpreter.lisp
lexer.lisp
home > aksoy > Desktop > PL Homework 3 > hw3_200104004044 > yaccP
1 (+ 2b3 4b5)





Bedirhan Ömer Aksoy
200104004074