CS3513 Programming Languages
Project Report
Group 1

Basic Information

- Group Members:
  1. Perera M.D.B. 210465P
  2. Weerasinghe M.C. 210688C

Project Overview

- The project comprises three primary modules:
  1. **Lexer**: Responsible for the lexical analysis of the input program.
  2. **Parser**: Converts the token stream produced by the lexer into an Abstract Syntax Tree (AST).
  3. **CSE Machine**: Executes the program by converting the AST into a Standard Tree (ST) and evaluating it.

Function Prototypes and Descriptions

Lexer Module

- The lexer module tokenizes the input program based on predefined lexical rules.
- Here are the key functions and their descriptions:

  TokenStream *lex(char *input):
  - The main entry point for lexical analysis. It takes an input string (the source code) and processes it to produce a stream of tokens.

  static regex_t *compile_regex(const char *pattern):
  - Compiles a regular expression pattern and returns a pointer to the compiled regex.

  static bool does_match(const regex_t *regex, const char *input):
  - Checks if a regex matches the beginning of an input string.

  static Match *match_regex(const regex_t *regex, const char *input):
  - Matches a regex against the input string and returns a Match structure if successful.

  static bool is_digit(char *input),
  static bool is_letter(char *input),
  static bool is_operator(char *input),
  static bool is_punctuation(char *input),
  static bool is_whitespace(char *input):
  - Checks if the input matches the respective regex.

  static char *ignore_whitespace(char *input),
  static char *ignore_comment(char *input):
  - Skips over whitespace and comments in the input.

static char identify_identifier(char input, TokenStream* stream),
static char identify_integer(char input, TokenStream* stream),
static char identify_operator(char input, TokenStream* stream),
static char identify_punctuation(char input, TokenStream* stream),
static char identify_string(char input, TokenStream* stream):**

- Identifies specific token types and adds them to the token stream.

Parser Module

- The parser module constructs an Abstract Syntax Tree (AST) from the token stream provided by the lexer, adhering to the grammar rules defined.
- These rules allow the parser to break down and analyze complex expressions, tuples, boolean expressions, arithmetic expressions, rators and rands, definitions, and variables according to RPAL's syntax.
- Each rule is implemented as a recursive function, enabling the parser to handle nested structures and various levels of precedence effectively.

- The parser implements a recursive descent parser for each of the grammar rules defined for the RPAL language.

CSE Machine Module

- The CSE machine module evaluates the Standard Tree (ST). Here are the key functions and their descriptions:
  void init_cse_machine(Vertex *vertex):
  - Initializes the CSE machine with a given vertex.
  - Allocates control cells for the current cell and environment.
  - Creates a new environment control cell, initializes its contents, and links it to the current environment.
  - Generates control structures for the given vertex, linking them appropriately.
  - Updates the current cell to point to the appropriate control structure.

  static CtrlCell *generate_ctrl_structs(Vertex *vertex, bool selfish):
  - Generates control structures recursively based on the given vertex.
  - Handles different vertex types (T_AUG, T_COND, T_TAU, E_LAMBDA, etc.) and creates corresponding control cells.
  - **For T_AUG**: Deals with left and right children to form TAU control cells, managing expressions and sibling links.
  - **For T_COND**: Creates control cells for conditionals (_then, _else, beta), linking them to the branches (B, T, E).
  - **For T_TAU**: Creates TAU cells, managing multiple expressions.
  - **For E_LAMBDA**: Manages parameters, allocating and initializing LAMBDA control cells.
  - **For other vertex types**: Creates simple control cells, setting types and content accordingly, and links them to children and siblings as necessary.

  void eval_cse_machine(void):
  - Evaluates the control structures in the CSE machine, following predefined rules.

- **Identifiers**: Searches for identifiers in the environment, replacing them with bound values if found.
- **Primitive Types**: For integers, strings, and doubles, moves to the previous cell.
- **Boolean Operations**: Handles B_OR, B_AND, B_NOT operations by evaluating and updating truth values.
- **Comparison Operations**: Evaluates comparison operators (B_GR, B_GE, B_LS, B_LE, B_EQ, B_NE) and updates the control cell with the result.
- **Arithmetic Operations**: Performs arithmetic operations (A_ADD, A_SUB, A_MUL, A_DIV, A_EXP) and updates the control cell with the result.
- **Negation**: For A_NEG, negates the content of the next cell.
- **R_GAMMA**: Handles function application, including built-in functions (Order, Print) and applying lambdas.
- **Environment Management**: Manages entering and exiting environments, updating links and cleaning up as necessary.
- **Y Combinator**: Handles the YSTAR operation by manipulating lambda expressions to implement recursion.

Additional Modules

1. **Tree Module**

- This module processes the Abstract Syntax Tree (AST) generated by the Parser module and produces a Standard Tree (ST).
- It conforms to an extended set of rules for the CSE machine, which includes 11 rules, compared to the basic set of 5.
- The tree module also contains functions to draw the tree structures to the console.

2. **Hash Table Module**

- This module implements a custom hash table to store renaming rules for a given environment in the CSE Machine module.
- The hash table uses a division hash algorithm and resolves collisions using linear probing.

3. **Linked List Module**

- This custom module stores the tokens generated by the lexer. Additionally, it can print the tokens to the console if the -lex switch is provided.

Summary

- The project integrates several sophisticated modules to achieve a comprehensive evaluation and execution of RPAL programs.

- The Lexer module performs lexical analysis, transforming source code into tokens.
- The Parser module constructs an AST from these tokens, adhering to RPAL's syntax rules.
- The CSE Machine module then standardizes and evaluates this tree, applying a wide array of rules and control structures to dynamically execute the program.
- Supporting modules like the tree module, hash table module, and linked list module further enhance the system's functionality and efficiency, making this project a robust implementation of the RPAL language.