Programming Languages Project Report

Group : 44 65 62 75 67 20 6D 65 20 67 65 6E 74 6C 79 (Debug me gently)

Member 1

- **Name:** Dilmina K.M.S
- **Student ID:** 220122X

Member 2

- **Name:** Abeyrathna A.H.M.R.T
- **Student ID:** 220008E

Project Structure and Overview

This project implements a simple interpreter for a functional programming language. The interpreter is modular, with each module handling a specific phase of the interpretation process. The main modules are:

- **myrpal.py**: Main driver script that orchestrates the entire process.
- **lexer.py**: Handles lexical analysis (tokenization).
- **grammar.py**: Handles parsing (syntax analysis).
- **nodes.py**: Defines the data structures for tokens and AST nodes.
- **standardizse.py**: Standardizes the AST for evaluation.
- **cse.py**: Implements the Control Stack Environment (CSE) machine for evaluation.

- **cse_structs.py**: Defines the data structures and built-in functions for the CSE machine.
- **vocabulary.py**: Provides utility functions for character and token classification.
- **parser.py** and **utils.py**: (Placeholders for additional parsing and utility functions.)

Main Program Flow

1. **Input Reading**:

   The program reads the source code from a file provided as a command-line argument.
2. **Lexical Analysis**:

   The scanner function tokenizes the input code, and screener cleans up the token list.
3. **Parsing**:

   The parser function builds an Abstract Syntax Tree (AST) from the tokens.
4. **AST Standardization**:

   The standardizer function transforms the AST into a standardized form suitable for evaluation.
5. **Evaluation**:

   The cse function evaluates the standardized AST using the CSE machine.
6. **Output**:

   Results are printed as specified by the program logic.

Function Prototypes and Structure

myrpal.py (Main Program)

```
def read_file_to_string(filename: str) → str:
    """Reads the entire content of a file as a string."""
    with open(filename, 'r') as file:
        return file.read()

def main():
    """Main entry point for the interpreter. Handles the workflow: input, lexing, par
    pass
```

lexer.py (Lexical Analysis)

```
def scanner(input_string):
    """Tokenizes the input string and populates the global tokens list."""

def screener():
    """Removes comments and whitespace tokens from the global tokens list."""
```

grammar.py (Parsing)

```
def parser(tokens: List[Token]) → Node:
    """Parses the token list and returns the root of the AST."""
```

nodes.py (AST and Token Structures)

```
class Token:
    def __init__(self, type_, value):
        """Represents a lexical token."""

class Node:
    def __init__(self, label, children=None):
        """Represents a node in the AST."""

def build_tree(label, num_args):
    """Builds an AST node with the given label and number of children."""

def print_ast():
    """Prints the AST."""

def print_tokens():
    """Prints the list of tokens."""

def print_tree():
    """Prints the derivation tree."""
```

standardizse.py (AST Standardization)

```
def standardizer():
    """Standardizes the AST in place."""

def standardize_tree(node):
    """Standardizes a subtree rooted at the given node."""

def standardize(node):
    """Standardizes a single node."""

def copy_node(dest, src):
    """Copies the contents of one node to another."""
```

cse_structs.py (CSE Machine Data Structures)

```
class Base:
    def __init__(self, type_: str, arg_str: Optional[str] = None, arg_int: Optional[int] =
        """Represents an element in the CSE machine."""
```

```
def add_in_built_to_env(env: Base):
    """Adds built-in functions and identifiers to the environment."""

def print_Base(env: Base):
    """Prints a Base object in a human-readable form."""

def in_built_functions(func: Base, func_args: Base):
    """Handles the execution of built-in functions."""

def clear_stacks():
    """Clears all global stacks and environments."""

def print_environments():
    """Prints all environments for debugging."""

def print_control_structures():
    """Prints all control structures for debugging."""
```

cse.py (CSE Machine Logic)

```
def add_func_to_control(prev, number):
    """Adds a function's control structure to the control stack."""

def pre_order_traversal(root, environment):
    """Traverses the AST in pre-order to build control structures."""

def rules(type_):
    """Implements the CSE machine rules for each control structure type."""

def cse():
    """Main entry point for the CSE machine evaluation."""
```

vocabulary.py (Token and Operator Classification)

```
def is_letter(char):
    """Checks if a character is a letter."""

def is_digit(char):
    """Checks if a character is a digit."""

def is_space(char):
    """Checks if a character is whitespace."""

def is_operator_char(char):
    """Checks if a character is an operator."""

def is_punctuation(char):
    """Checks if a character is punctuation."""

def is_reserved(token):
    """Checks if a token is a reserved word."""

def is_binary_operator(token):
    """Checks if a token is a binary operator."""

def is_unary_operator(token):
    """Checks if a token is a unary operator."""
```

Example Usage

- To run the interpreter, use the following command:

```
python .\myrpal.py input.rpal
```

- For more tests

```
python .\myrpal.py .\tests\ex7.rpal
```

- To print AST use following command

```
python .\myrpal.py input.rpal -ast
```

To print Standardized Tree use following command

```
python .\myrpal.py input.rpal -st
```

Notes

- The project is modular and each file is responsible for a specific phase of the interpretation process.
- Debugging utilities such as `print_environments()` and `print_control_structures()` are available for inspecting the internal state of the interpreter.
- The CSE machine is implemented in cse.py and cse_structs.py and is responsible for evaluating the standardized AST.

End of Report