RPAL Interpreter

A Python implementation of an interpreter for the RPAL (Recursive Programming Algorithmic Language) programming language. This project implements a complete RPAL interpreter that follows the standard RPAL language specification and includes features like lexical analysis, parsing, AST standardization, and CSE machine execution.

Features

- Complete RPAL language support
- Lexical Analysis (Tokenization)
- Syntax Analysis (Parsing)
- Abstract Syntax Tree (AST) generation
- AST Standardization
- CSE Machine implementation for program execution
- Support for various RPAL language constructs including:
    - Function definitions and applications
    - Control structures (if-then-else)
    - Lists and tuples
    - Recursive functions
    - Pattern matching
    - And more

Project Structure

```
.
├── src/                                # Source code directory
│   ├── lexer.py                        # Lexical analyzer
│   ├── parser.py                       # Parser implementation
│   ├── utils.py                        # Utility functions
│   ├── rpal_ast.py                     # AST node definitions
│   ├── nary_to_lcrs_convertor.py       # N-ary to LCRS tree conversion
│   ├── lcrs_to_nary_convertor.py       # LCRS to N-ary tree conversion
│   ├── __init__.py
│   ├── standerizer/                    # AST standardization
│   │   ├── node.py                     # AST node implementations
│   │   ├── ast.py                      # AST standardization logic
│   │   └── ast_factory.py             # Factory for creating AST nodes
│   └── cse_machine/                    # CSE machine implementation
│       ├── machine.py                  # Main CSE machine implementation
│       ├── error_handler.py            # Error handling utilities
│       ├── cse_error_handler.py        # CSE-specific error handling
│       ├── __init__.py
│       ├── apply_operations/           # Operation implementations
│       │   ├── apply_binary_operations.py
│       │   ├── apply_unary_operations.py
│       │   └── __init__.py
│       ├── utils/                      # Utility functions
│       │   ├── tokens.py               # Token definitions
│       │   ├── util.py                 # General utilities
│       │   ├── control_structure_element.py
│       │   ├── stack.py                # Stack implementation
│       │   ├── STlinearizer.py         # Standard tree linearizer
│       │   └── __init__.py
│       └── data_structures/            # Data structure implementations
│           ├── stack.py                # Stack data structure
│           ├── enviroment.py           # Environment management
│           ├── control_structure.py    # Control structure implementation
│           └── __init__.py
├── test-programs/                      # Sample RPAL programs for testing
├── tests/                              # Test suite
│   ├── test_lexer.py                   # Lexer tests
│   ├── test_parser.py                  # Parser tests
│   └── test_parser_basics.py           # Basic parser tests
├── myrpal.py                           # Main interpreter script
```

```
├── test_interpreter.py              # Interpreter tests
├── Makefile                         # Build and test automation
└── .gitignore                       # Git ignore file
```

Requirements

- Python 3.x
- No external dependencies required

Installation

1. Clone the repository:

```
git clone https://github.com/yourusername/rpal-interpreter.git
cd rpal-interpreter
```

2. No additional installation steps are required as the project uses only Python standard library.

Usage

The interpreter can be run in several modes:

1. Basic execution:

```
python myrpal.py program.rpal
```

2. Print the original AST:

```
python myrpal.py program.rpal -ast
```

3. Print the standardized AST:

```
python myrpal.py program.rpal -st
```

Example Programs

The `test-programs/` directory contains various example RPAL programs demonstrating different language features:

- Basic arithmetic (`add.rpal` , `sum.rpal`)
- Function definitions (`defns.rpal` , `defns1.rpal`)
- Control structures (`if1.rpal`)
- List operations (`Innerprod.rpal` , `reverse.rpal`)
- Recursive functions (`recurs1.rpal`)
- And many more...

Testing

The project includes a comprehensive test suite. To run the tests:

```
make test
```

Or directly using Python:

```
python test_interpreter.py
```

Implementation Details

The interpreter follows these main steps:

1. **Lexical Analysis**: Converts source code into tokens
2. **Syntax Analysis**: Parses tokens into an Abstract Syntax Tree (AST)
3. **AST Standardization**: Transforms the AST into a standardized form
4. **CSE Machine Execution**: Executes the standardized AST using the CSE machine

Acknowledgments

- Based on the RPAL language specification
- Inspired by the original RPAL implementation