README

Literal Table and Expression Evaluator

CSC 354 - Systems Programming Assignment 2

Overview

This program is part of the CSC 354 Systems Programming course, designed to evaluate assembly language expressions and manage a literal table in the context of a SIC/XE assembler. The primary tasks include parsing expressions, building a literal table, and evaluating operands. It supports various addressing modes, arithmetic operations, and provides detailed error reporting.

Features

- Symbol Table Integration: Reads symbols from a file (`SYMS.DAT`) and uses them during expression evaluation.
- Literal Table Management: Inserts, updates, and displays literals using a linked list data structure.
- Expression Parsing: Parses assembly language expressions from an input file, handling different addressing modes and operand combinations.
- Error Handling: Logs detailed error messages for invalid literals, unsupported symbols, and expression errors.
- Expression Evaluation: Supports addition and subtraction of operands, and calculates relocatability flags (N-bit, I-bit, and X-bit).
- Pagination: Paginated display for long outputs, such as logs, literals, and expression results.
- Logging: Maintains logs of both errors and actions taken during execution.

Project Structure

Key Classes

1. LiteralData: Represents a literal's name, value, length, and address.
2. LiteralNode: Node structure for the linked list, containing `LiteralData`.
3. LiteralTableList: Manages the literal table, providing methods to insert, update, and display literals.
4. ErrorLogHandler: Manages logging for both actions and errors, with support for pagination.
5. ExpressionParser: Responsible for parsing expressions, handling literals, and validating operands.
6. ExpressionEvaluator: Evaluates the parsed expressions and handles operand calculations.

7. ExpressionResults: Formats and outputs evaluated expression results.
8. LiteralTableDriver: Coordinates the overall program, including building the symbol table, parsing and evaluating expressions, and displaying results.

Input/Output

Inputs:

- SYMS.DAT: A file containing the symbol table, with symbols, values, and relocatability flags.
- EXPR.DAT: A file containing assembly language expressions to be parsed and evaluated.

Outputs:

- Evaluated Expressions: Displays each expression's value, relocatability, and flags (N-bit, I-bit, and X-bit).
- Literal Table: Displays all literals with their name, value, length, and address.
- Logs: Provides detailed logs of all actions taken and errors encountered.

Usage

Running the Program

To run the program:

1. Navigate to the program directory:

   ```
   cd Systems_Programming_Projects_CSC_354/A2_Literal_Table
   ```

2. Execute the program with the expression file as an argument:

   ```
   python johnA2.py EXPR.DAT
   ```

   If no file is provided, it defaults to `EXPR.DAT`.

Commands:

- Expression File: The program accepts an expression file (`EXPR.DAT`) via the command line.
- Symbol Table: The symbol table is automatically read from `SYMS.DAT`.

Program Execution Flow:

1. **Symbol Table Construction:** Builds the symbol table from `SYMS.DAT`.
2. **Expression Parsing:** Parses the expressions from the provided file.
3. **Literal Management:** Handles insertion of literals and updates their addresses.
4. **Expression Evaluation:** Evaluates each expression's value and flags.
5. **Display Results:** Outputs the results and logs, with pagination for long output.

Error Handling

The program handles various types of errors:

- **Invalid Literals:** Errors related to invalid literal formats (e.g., illegal hexadecimal or character literals) are logged and displayed.
- **Undefined Symbols:** Expressions containing symbols that are not in the symbol table trigger an error.
- **Operand Validation:** Ensures proper number and types of operands, logging errors when incorrect.
  All errors are logged and displayed at the end of program execution.

Example Output

Evaluated Expression Results:

EXPRESSION RESULTS	Value	Relocatable	N-Bit	I-Bit	X-Bit
@SUM+10	25	RELATIVE	1	0	0
#LENGTH	ERROR	-	-	-	-

Literal Table:

LITERAL TABLE	Value	Length	Address
=X'05A'	05A	2	1

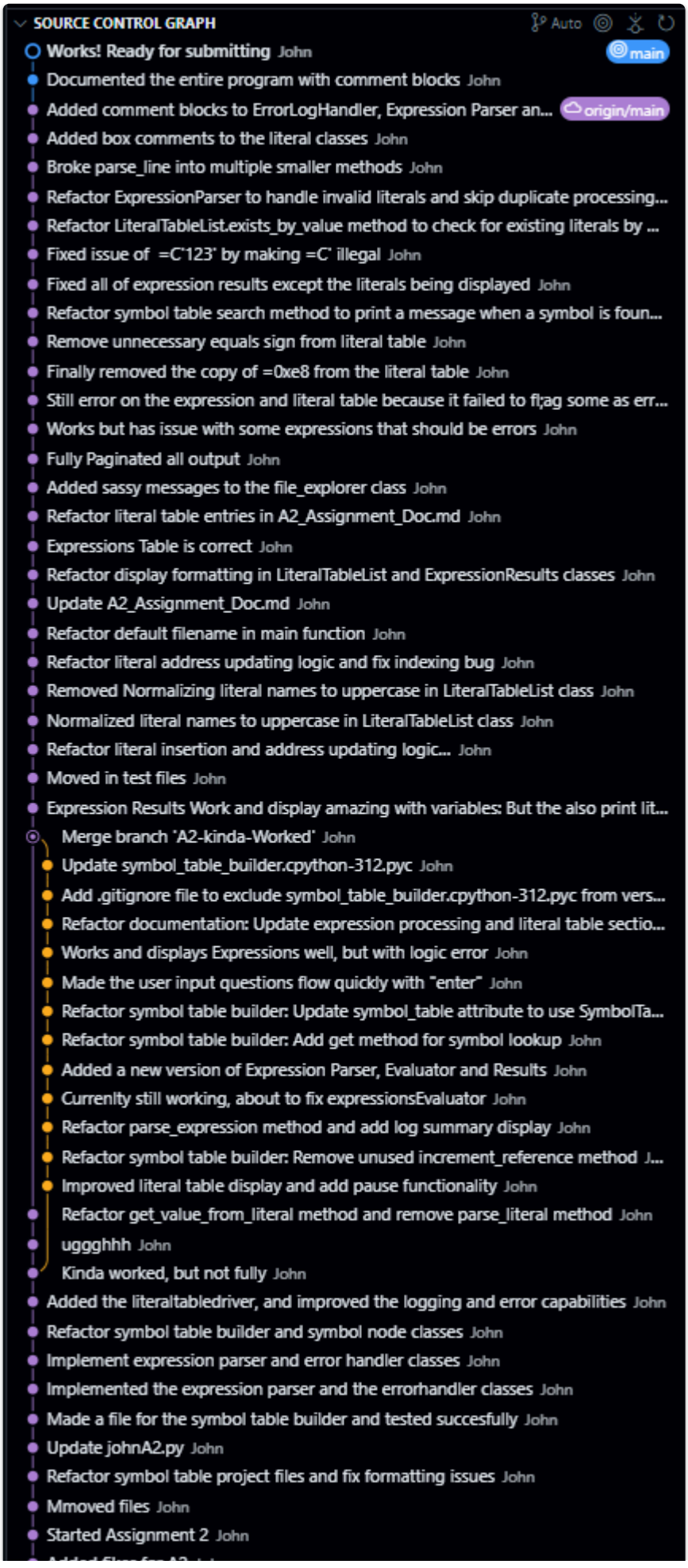
License

Author

- John Akujobi
- Systems Programming
- South Dakota State University
- Instructor: George Hamer
- October 2024

Development Commit History

Here is the graph showing the commit history of the project up to the creation of this README file


SOURCE CONTROL GRAPH
Auto
Works! Ready for submitting John
main
Documented the entire program with comment blocks John
Added comment blocks to ErrorLogHandler, Expression Parser an...
origin/main
Added box comments to the literal classes John
Broke parse_line into multiple smaller methods John
Refactor ExpressionParser to handle invalid literals and skip duplicate processing...
Refactor LiteralTableList.exists_by_value method to check for existing literals by ...
Fixed issue of =C'123' by making =C' illegal John
Fixed all of expression results except the literals being displayed John
Refactor symbol table search method to print a message when a symbol is foun...
Remove unnecessary equals sign from literal table John
Finally removed the copy of =0xe8 from the literal table John
Still error on the expression and literal table because it failed to fl;ag some as err...
Works but has issue with some expressions that should be errors John
Fully Paginated all output John
Added sassy messages to the file_explorer class John
Refactor literal table entries in A2_Assignment_Doc.md John
Expressions Table is correct John
Refactor display formatting in LiteralTableList and ExpressionResults classes John
Update A2_Assignment_Doc.md John
Refactor default filename in main function John
Refactor literal address updating logic and fix indexing bug John
Removed Normalizing literal names to uppercase in LiteralTableList class John
Normalized literal names to uppercase in LiteralTableList class John
Refactor literal insertion and address updating logic... John
Moved in test files John
Expression Results Work and display amazing with variables: But the also print lit...
Merge branch 'A2-kinda-Worked' John
Update symbol_table_builder.cpython-312.pyc John
Add .gitignore file to exclude symbol_table_builder.cpython-312.pyc from vers...
Refactor documentation: Update expression processing and literal table sectio...
Works and displays Expressions well, but with logic error John
Made the user input questions flow quickly with "enter" John
Refactor symbol table builder: Update symbol_table attribute to use SymbolTa...
Refactor symbol table builder: Add get method for symbol lookup John
Added a new version of Expression Parser, Evaluator and Results John
Currenlty still working, about to fix expressionsEvaluator John
Refactor parse_expression method and add log summary display John
Refactor symbol table builder: Remove unused increment_reference method J...
Improved literal table display and add pause functionality John
Refactor get_value_from_literal method and remove parse_literal method John
uggghhh John
Kinda worked, but not fully John
Added the literaltabledriver, and improved the logging and error capabilities John
Refactor symbol table builder and symbol node classes John
Implement expression parser and error handler classes John
Implemented the expression parser and the errorhandler classes John
Made a file for the symbol table builder and tested succesfully John
Update johnA2.py John
Refactor symbol table project files and fix formatting issues John
Mmoved files John
Started Assignment 2 John

Artificial Intelligence Usage and Disclaimer

Asides these forementioned cases, the project was created and built by John Akujobi, and the materials provided by the instructor, George Hamer.

README

Based on a draft manually written by John Akujobi, this README was recreated and improved using a finetuned opensource large language model that was run locally on his personal computer.

- Model used: qwen2-1_5b-instruct-q8_0.gguf
- Interface: LM Studio

A2_Assignment_Doc.md

Using his handwritten notes, and a copy of the assignment document provided by the instructor, George Hamer, John Akujobi used an opensource local llm to break the requirements down into a form he better understood. Then he made further manual edits to it.

- Model used: Monah-8b-Uncensored-v0.2-gguf-Q6_K.gguf
- Interface: LM Studio

Comment Blocks in johnA2.py

After John had completed the program, he used a prototype of an app he is developing to generate the comment blocks. The application uses a locally run opensource llm to create the descriptions written in the comment blocks.

- Model used : Meta Llama3 8B
- Interface: Ollama

Debugging

John used a locally run opensource llm, and GitHub copilot to debug snippets of code in johnA2.py. The llm only debugged, and suggested improvements to sections of code in the program.

After completing the program, John also used GitHub Copilot to review the program and suggest ways of further modularizing the code.

- Local Model used: Qwen2.5-Coder-7B-Instruct-Q8_0.gguf
- Interface: LM Studio
- Model Used: GitHub Copilot
- Interface: GitHub Copilot Extension in Visual Studio Code
