Lexer and Parser Code Documentation

A comprehensive explanation and breakdown of the lexer and parser implementation.

Author: Mahyar Mohammadian
Date: January 25, 2025

Introduction

This document explains the implementation of the lexer and parser files, breaking them into key components and describing their functionality in detail. It provides insights into the code, enhancing understanding for readers and developers.

Lexer Code Breakdown

The lexer (‘lexer.l‘) is responsible for tokenizing the input. Below are its key parts:

Header Section

The header section includes necessary libraries and definitions:

```
%{
#include "parser.tab.h"
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>
%}
```

Listing 1: Header Section of Lexer

This section imports the parser header file and standard libraries used in lexical analysis.

Definitions

Token patterns are defined here:

```
DIGITS  [0-9]+
IDENT   [a-zA-Z_][a-zA-Z_0-9]*
```

Listing 2: Definitions in Lexer

Explanation:

- `DIGITS`: Matches sequences of digits (e.g., ”123”).
- `IDENT`: Matches identifiers starting with a letter or underscore.

Rules Section

The rules map patterns to actions:

```
{DIGITS} {
    yylval.attributes.value = strtol(yytext, NULL, 10);
    yylval.attributes.temp_var = -1;
    return NUMBER;
}

{IDENT} {
    yylval.identifier = strdup(yytext);
```

```
    return IDENTIFIER;
}

"+" { return PLUS; }
"-" { return MINUS; }
"*" { return MULTIPLY; }
"/" { return DIVIDE; }
"=" { return ASSIGN; }
\n { /* Ignore newlines */ }
[ \t]+ { /* Ignore whitespace */ }
. { printf("Unknown character: %s\n", yytext); }
```

Listing 3: Rules Section in Lexer

Explanation:

- `DIGITS`: Converts digit strings to integers and sets attributes.
- `IDENT`: Copies the identifier text for further processing.
- `Operators (+, -, *, /)`: Recognized as tokens for arithmetic operations.
- `Whitespace and Newlines`: Ignored to allow flexible input formatting.
- `Unknown Characters`: Printed as errors to help debugging.

Parser Code Breakdown

The parser (‘parser.y‘) defines the grammar and constructs the syntax tree. Below are its key parts:

Header Section

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

extern int yylex();
extern int yyparse();
extern FILE *yyin;

void yyerror(const char *msg);

int temp_var_counter = 1; // Counter for temporary variables
int final_result = 0;     // Stores the final computed result
```

Listing 4: Header Section of Parser

Explanation: This section declares external functions, global variables, and includes libraries needed for parsing.

Grammar Rules

The grammar rules define how tokens are combined:

```
stmt: expr ';' { printf("Result: %d\n", $1); }
    | error ';' { yyerror("Syntax error"); };

expr: expr '+' term { $$ = $1 + $3; }
    | expr '-' term { $$ = $1 - $3; }
    | term;

term: term '*' factor { $$ = $1 * $3; }
    | term '/' factor { if ($3 != 0) $$ = $1 / $3; else
       yyerror("Division by zero"); }
    | factor;

factor: '(' expr ')' { $$ = $2; }
      | NUMBER { $$ = $1; }
      | IDENTIFIER { /* Handle variable references */ };
```

Listing 5: Grammar Rules in Parser

Explanation:

- `stmt`: Handles expressions and reports syntax errors.
- `expr`: Defines addition and subtraction operations.
- `term`: Defines multiplication and division operations with error handling for division by zero.
- `factor`: Handles grouping with parentheses and uses numbers or identifiers.

Error Handling

The parser uses `yyerror` to report errors:

```
void yyerror(const char *msg) {
    fprintf(stderr, "Error: %s\n", msg);
}
```

Listing 6: Error Handling in Parser

This function prints an error message to `stderr`, making it easier to debug syntax issues.