گزارش پروژه کامپایلر

استاد : دکتر اقبال منصوری

تهیه کننده : امیرحسین تسلیمی - 9932099

1. Lex Code:

- Lex is used to define patterns (regular expressions) and corresponding actions.
- The patterns identify tokens (e.g., numbers, operators) in the input stream.
- The actions perform tasks when a pattern is matched (e.g., copying a numeric value).

Detail:

Lex Prologue:

- **Prologue:** This section is known as the Lex prologue. It allows you to include header files and perform other pre-processing tasks. In this case, it includes the standard I/O library (**stdio.h**), the string manipulation library (**string.h**), and a header file named "parser.tab.h." The "parser.tab.h" file typically contains definitions generated by Yacc and is required for communication between Lex and Yacc.

Lex Definitions:

- **Definitions Section:** Lex allows you to define patterns using regular expressions.
    - **delim**: Represents whitespace characters (space, tab, newline).
    - **ws**: Represents one or more whitespace characters.
    - **digit**: Represents a single digit (0-9).
    - **number**: Represents a numeric literal. It includes an optional fractional part and an optional exponent part.

Lex Rules:

- **Rules Section:** Lex rules consist of a regular expression pattern and an associated action.
    - **{ws}**: Matches whitespace characters and does nothing (**{ ; }**). This is used to skip over whitespace.
    - **{number}**: Matches a numeric literal. The action copies the matched text (**yytext**) to **yylval.num** and returns the token type **NUM**.
    - **"+"**, **"-"**, **"*"**, **"/"**: Matches the basic arithmetic operators. Each returns the respective token type (**ADD**, **SUB**, **MUL**, **DIV**).

- **[-=()]**: Matches any character from the set **[-=()]** and returns the character itself as the token type.
- **.**: Matches any character that hasn't been matched by the previous rules. It prints an error message indicating that an unexpected character has been encountered.

2. Yacc Code :

Yacc is defining a parser for a simple (specific) calculator language. It specifies the grammar rules that define the syntax of arithmetic expressions, including addition, subtraction, multiplication, and division operations. The code uses semantic actions associated with each grammar rule to generate intermediate code for these operations. The parser maintains a symbol table to store variables and their values, facilitating the processing of expressions involving variables. The external declarations indicate the integration of a lexical analyzer (Lex) and external file pointers for input and output. The main function initiates the parsing process, invoking the Yacc-generated parser to analyze and generate intermediate code for arithmetic expressions, considering both numeric literals and variables.

Detail:

Yacc Declarations:

- I define a union union to hold different types of tokens. I use %token to declare terminal symbols (NUM, ADD, SUB, MUL, DIV). %type is used to declare non-terminal symbols (expr, add, term, factor, start). I also specify the associativity of operators using %right and %left, it is nescery if grammar was ambiguous.

Yacc Rules:

- These are the Yacc rules that define the grammar and the corresponding actions to be taken when each rule is matched. The rules specify how to build the syntax tree and generate intermediate code.
- The start rule indicates the starting point for parsing and specifies that it should match an expr.
- The expr rule represents an expression and assigns its value to the left-hand side ($$).

- The add rule handles addition and subtraction operations, generating intermediate code and updating the result.
- The term rule handles multiplication and division operations, similar to the add rule.
- The factor rule handles parentheses, unary minus, and numeric literals.

External Declarations:

- External declarations like extern long long yylex(); indicate that the lexer function (yylex) is defined elsewhere.
- Declarations like extern FILE *yyin; and extern FILE *yyout; declare external file pointers used for input and output.

Error Handling and Main Function:

- The yyerror function is called on syntax errors, and it prints an error message to stderr. The yywrap function is called when the end of the input is reached. The main function calls yyparse to initiate the parsing process.

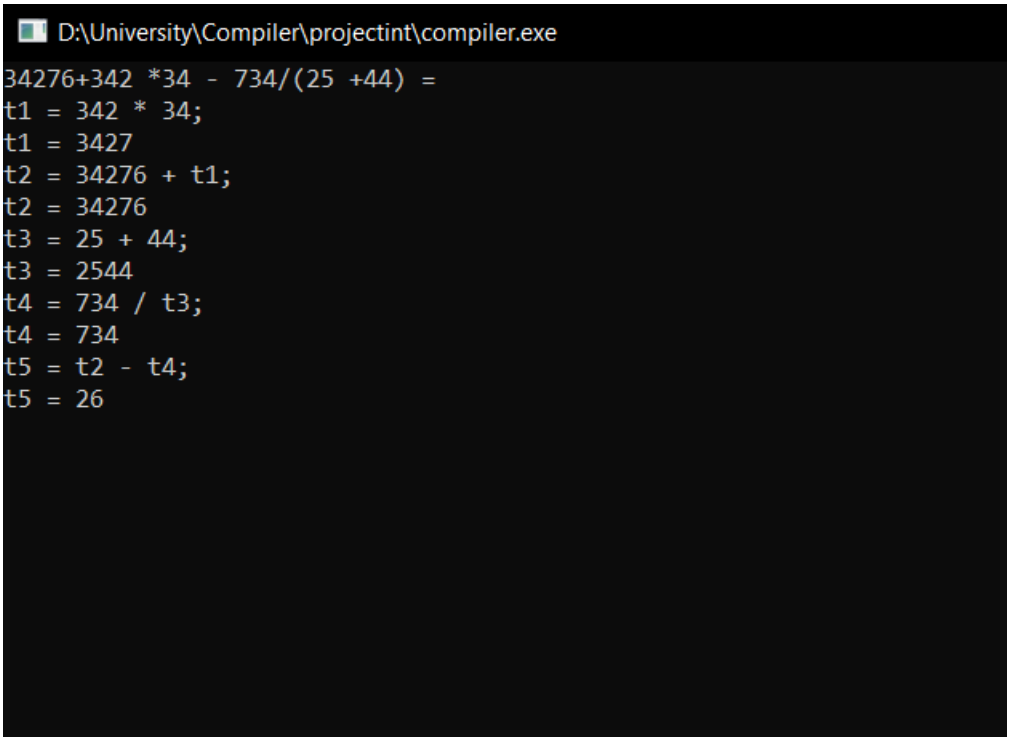
Symbol Table and Helper Functions:

I have defined a SymbolTable structure in C to store variables and their values, such as t1 = 50. The symbolTable is a struct that holds a variable and its corresponding value. Every time an operation is performed, the counter increases, and the result is stored in a new variable, for example, t1 -> t2.

The calculate function takes four parameters: result, op1, operator, and op2. For instance, if it receives the input t1 = 20 + 30, it calculates the result and stores t1 with the value 50 in the symbolTable. Subsequently, if it encounters t2 = t1 + 10, it retrieves the value of t1 from the symbolTable and computes the value of t2.

The performOperation function takes three parameters: op, a, and b. It operates on two numbers (a and b) based on the specified operation (op).

- a+b: Append the digits of number b that are not present in number a to the end of a.
- a-b: Remove the digits of number b that are present in number a from a.
- a*b: Append the digit obtained by adding the digits (or sum of the digits) of number b, which are not present in a, to the end of a.
- a/b: Remove the digit obtained by adding the digits (or sum of the digits) of number b, which are present in a, from a.


D:\University\Compiler\projectint\compiler.exe
34276+342 *34 - 734/(25 +44) =
t1 = 342 * 34;
t1 = 3427
t2 = 34276 + t1;
t2 = 34276
t3 = 25 + 44;
t3 = 2544
t4 = 734 / t3;
t4 = 734
t5 = t2 - t4;
t5 = 26