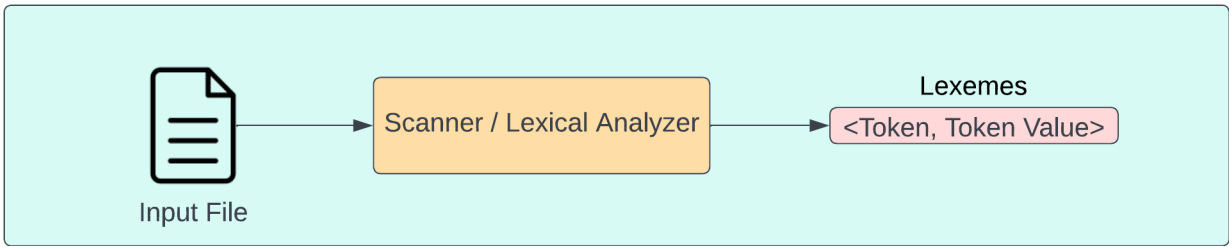
CS424
Compiler Construction

Assignment # 2
Report

Name: Syed Muhammad Ashhar Shah
Reg No: 2020478

Scanner Design:

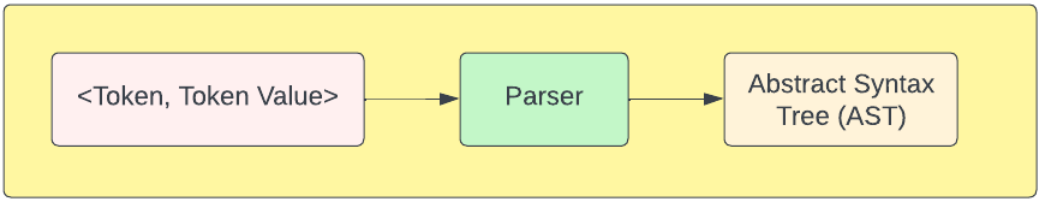


The scanner takes as input a source file that contains the contents that we want to read. The contents of the file are sent to our scanner also known as the lexical analyzer which then reads the contents of the file and breaks it down into lexemes. Each lexeme contains the token along with the detected tokens value.

Our tokens will be categorized into 7 different types shown below:

1. INTEGER
2. BOOLEAN
3. IDENTIFIER
4. OPERATOR = *, +, /, -, ; , =
5. KEYWORD = Print, If, Else, True, False
6. COMMENT = #
7. END_OF_FILE

Parser Details



A parser takes input tokens generated by lexical analysis and constructs an Abstract Syntax Tree (AST) by analyzing the syntactic structure of the code according to the rules defined by the programming language's grammar. The AST represents the hierarchical structure of the code, with nodes representing syntactic constructs and their relationships. This tree-like representation captures the essential syntax of the program, facilitating further analysis and transformation during compilation or interpretation processes.

The parser we have implemented iterates over the tokens in a iterative or recursive fashion to check if the provided stream of tokens is valid or not, if we encounter a token that is not supposed to be encountered means that there was a syntax error and we simply print that we have encountered a syntax error while parsing the stream of tokens.

Parser Context Free Grammar

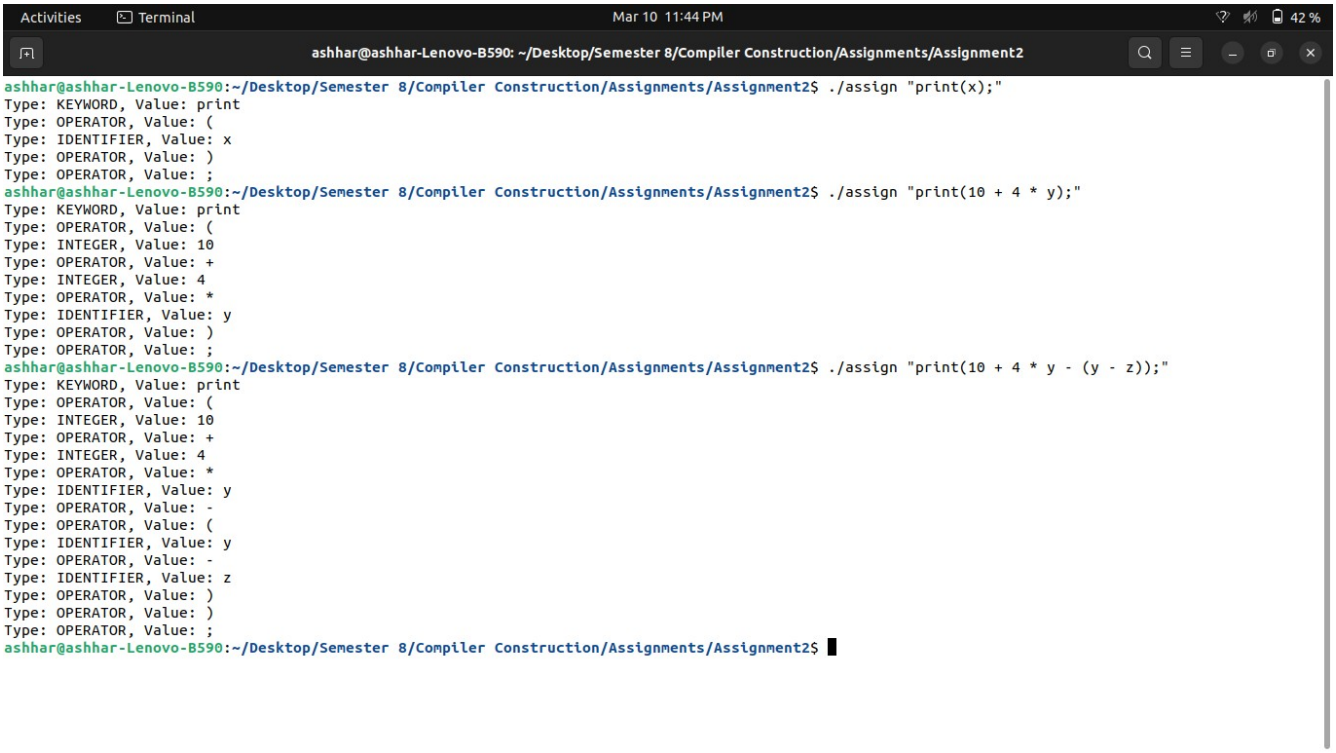
Below we can see all the production rules of our context free grammar that we have defined in our parser class:

- **program** → <statement> | END_OF_FILE
- **statement** → <assignment> | <conditional> | <printStatement>
- **assignment** → = <expression> ;
- **conditional** → if (<expression>) { <program> } | if (<expression>) { <program> } else { <program> }
- **printStatement**> → PRINT <expression> ;
- **expression** → <term> + <term> | <term> - <term> | <term>
- **term** → <factor> * <factor> | <factor> / <factor> | <factor>
- **factor** → INTEGER | IDENTIFIER | (<expression>)

The provided Parser class in the file 2020478_Assignment2.cpp implements a parser for a simple MiniLang programming language. It maintains a list of tokens received from the scanner and a pointer indicating the current token being processed. The parser defines several methods corresponding to different rules of the context-free grammar (CFG) that we have defined above, such as program(), statement(), assignment(), conditional(), printStatement(), expression(), term(), and factor().

Each method parses a specific syntactic construct according to the CFG rules, recursively calling other methods as needed. Error handling for syntax errors is included, with the parser attempting to recover and continue parsing after encountering an error. **The parse() method initiates the parsing process by starting with the program() rule.** Overall, the Parser class serves as a key component in analyzing and interpreting source code written in the defined language.

Valid Testcases



Invalid Testcases