From: Vandana Agarwal . (via Nalanda)
Sent: 10 February 2023 21:36
To: RAMAKANT PANDURANG TALANKAR .
Subject: 11551001318_CS F363_JAN_2023: Stage 1: Modules

V

Stage 1: Modules
by Vandana Agarwal . - Friday, 10 February 2023, 9:06 PM

Develop the following modules of the compiler for implementing the given language.

Lexical Analyzer: This module takes as input the file containing user source code in the given language and produces the tokens. The lexical analyzer module scans the input only once and collects all relevant information required by the other modules of the compiler. The lexical analyzer ignores comments and white spaces while recognizing the useful lexemes as valid tokens. The lexical errors are reported by this module when it sees any symbol or pattern not belonging to the language. Your lexical analyzer must

- Tokenize lexemes appropriately
- Maintain all information collected during a single pass of the source code
- Be efficient with respect to time and space complexity
- Report all lexical errors (with line number)

Syntax Analyzer: This module takes as input the token stream from the lexical analyzer module and verifies syntactic correctness of the source code. This uses predictive parser (using parsing table) to establish the syntactic structure of the source code. As the parser sees next token, verifies its correctness, it uses the token information to build a tree node and inserts appropriately in the parse tree corresponding to the input source code. If the source code (in given language) is syntactically correct, a corresponding parse tree is produced as the output. If the input is syntactically wrong, errors are reported appropriately. Your syntax analyzer (Parser) must

- Ensure time and space efficiency and use a single pass of the token stream
- Use predictive parser using parsing table
- Produce as output the parse tree, if the source code is syntactically correct
- Produce a list of all syntax errors with appropriate messages and line numbers

Implementation Details

- Use C language (Linux/ Ubuntu based GCC) to implement the modules [exact version to be announced shortly].
- Use of any other high level language or lexer/ parser generator packages is NOT allowed.
- Test your code with the test cases given in the language specification document.
- Generate more test cases and verify the correctness of your code.
- You will be given more test cases later.

- An appropriate interface support will be provided to you as you are through with the ground work. Instead of starting coding right immediately, spend time designing the structure of your compiler code first.

[See this post in context]
