Assignment 2 – Syntax Analyzer

1. Problem Statement

The objective of Assignment 2 is to design a Syntax Analyzer using a top-down parser, a predictive recursive descent parser or a table-driven predictive parser.

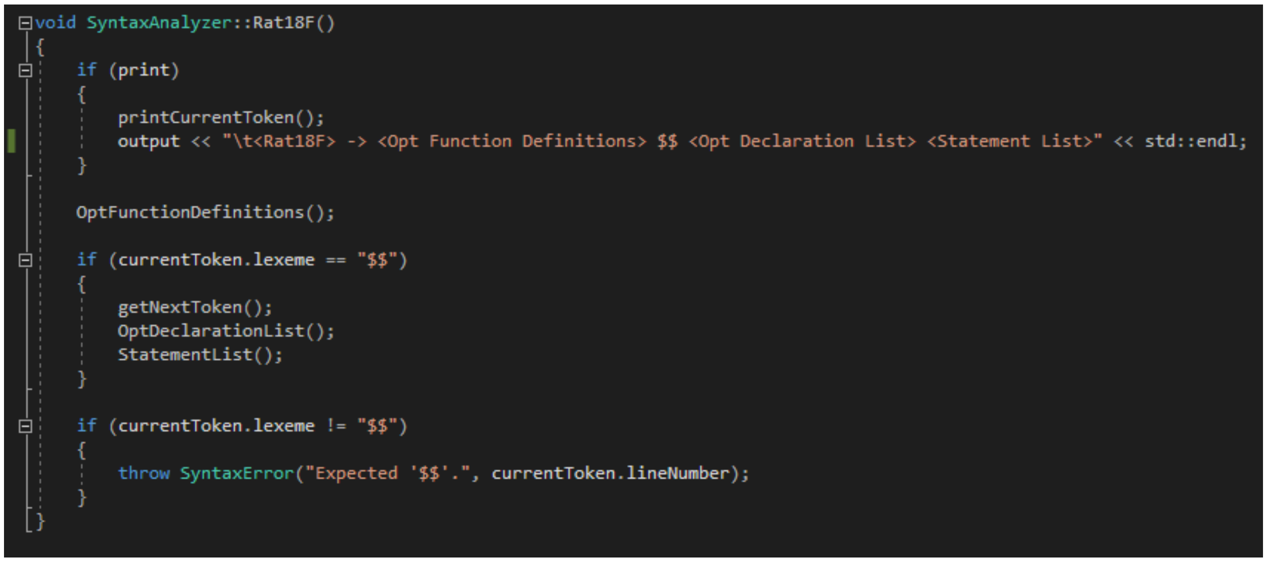
The Syntax Analyzer will take its input from our Lexer, analyze it, and output each token and lexeme and the production rules used to analyze them. If an error is encountered, the Syntax Analyzer will display a meaningful message to the output file and then terminate.

2. How to use your program

To use the program, first insure that the test files are in the same directory relative to the executable file "`SyntaxAnalyzer.exe`". Launch the executable. Once it has finished, output will be located in a newly created file, local to the executable file, called "`output.txt`".

3. Design of your program

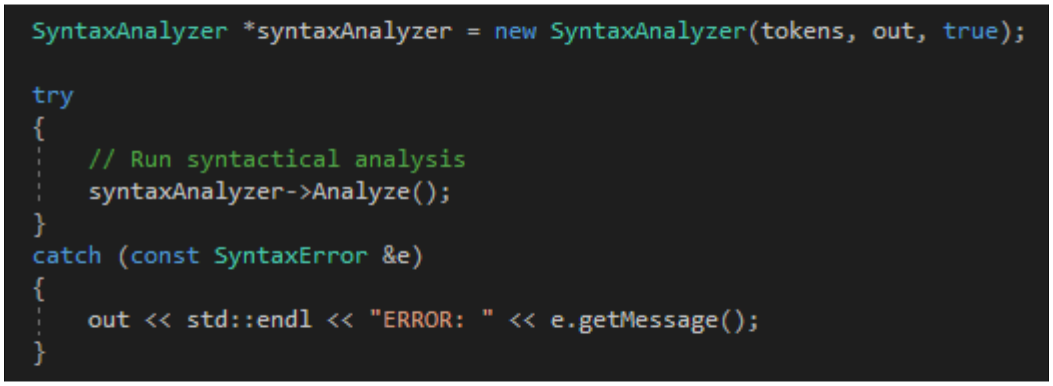
We chose the top-down parser architecture for the Syntax Analyzer. The program begins in a root `Rat18F()` function, and depending on the tokens it encounters and production rules used, will enter into several other functions. Each function prints the production rules used, creating a log of syntax analysis in the output file.



```
void SyntaxAnalyzer::Rat18F()
{
    if (print)
    {
        printCurrentToken();
        output << "\t<Rat18F> -> <Opt Function Definitions> $$ <Opt Declaration List> <Statement List>" << std::endl;
    }

    OptFunctionDefinitions();

    if (currentToken.lexeme == "$$")
    {
        getNextToken();
        OptDeclarationList();
        StatementList();
    }

    if (currentToken.lexeme != "$$")
    {
        throw SyntaxError("Expected '$$'.", currentToken.lineNumber);
    }
}
```

Once a terminating node is reached, we get the next token from the list of tokens provided by the Lexer and print the new token to the output. If at any time improper syntax or an illegal token is encountered, the Syntax Analyzer will throw a `SyntaxError` *containing a meaningful error message and line number on which it occurred. The analyzer will then terminate.*

4. **Any Limitation**
   One limitation of our Syntax Analyzer is how we handle errors. Currently, each function that checks for required symbols such as '{', '(' or ',', will throw a `SyntaxError` *which is handled by the driver program. The* `getNextToken()` *function will also throw a* `SyntaxError` *if it finds an Illegal token.*



```
SyntaxAnalyzer *syntaxAnalyzer = new SyntaxAnalyzer(tokens, out, true);

try
{
    // Run syntactical analysis
    syntaxAnalyzer->Analyze();
}
catch (const SyntaxError &e)
{
    out << std::endl << "ERROR: " << e.getMessage();
}
```

5. **Any shortcomings**
   No known shortcomings.