**CSE2260**

**Principles of Programming Languages**

**Project 2 – Parser Project Report**

**Group Members:**

Feyzullah Asıllıoğlu – 150121021

Mohamad Nael Ayoubi – 150120997

Kadir Bat – 150120012

We used the java language in this program as we did in the first project. The purpose of this project is to read characters from an input file, perform lexical analysis and then apply the parser to write to an output file.

Our program reads characters from a file using FileReader and BufferedReader classes. It then reads each character one by one and performs lexical analysis on it. The lexical analysis is done using a set of nested if statements. If the character is a letter or digit, it is added to the token string. If the character is a space or newline character, the token string is checked against keywords and identifiers. If it matches a keyword, it is added to the tokens list as a keyword. If it matches an identifier, it is added to the tokens list as an identifier. If it matches neither, it is added to the tokens list as a lexical error.

We applied the tokens we used in the previous project in this project. Lexical analyzer we made in the previous project creates tokens.txt file in this project and we use it

We used some of the methods we used in the previous project in this project while determining the tokens. We used most of the isString isNumber isBracket and main method that we did in the first project.

As we process the tokens created by the Lexical Analyzer, we compare their order and structure to the grammar rules provided for the programming language. Our implementation employs a recursive-descent parser.

We generated a parse tree. We have assigned each of our parsing functions to a specific non-terminal in the language. To facilitate visualizing the parse tree's topology, we have implemented the printParse() method. By utilizing the PrintWriter class, we are able to save the generated parse tree to an output file named "outputParse.txt." This file provides an organized representation of the tokens, complete with appropriate indentation, enabling us to observe the nesting and hierarchical structure of the parse tree.

Our parsing process begins with the program() function, which serves as the grammar's beginning symbol. As the Syntax Analyzer, we begin the parsing process by invoking other parsing procedures to handle various program components.

topLevelForm() is one such parsing function, in which we handle each top-level form of the program, whether it be a function call or a definition. We compare the predicted tokens with the Lexical Analyzer's nextToken variable. If a token matches the anticipated one, we consume it and continue processing it. If a token does not match the anticipated one, a syntax error message stating the expected token is displayed.

We follow the grammar rules provided for the programming language throughout the parsing process. Our parsing functions are built to handle a variety of non-terminals and their associated rules. For example, we have definition() and definitionRight() functions to handle variable and function definitions, respectively, and definition() to process the right-hand side of a definition, which can be an expression or a list of arguments and statements.

Running the program: program prompts user for input file name.

metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

After the program is run, both output is printed to the terminal and a file named outputParse.txt. The program also creates tokens.txt for tokens.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Sample inputs and outputs:

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Output in the terminal:

metin, ekran görüntüsü, yazı tipi, menü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

outputParse.txt file:

metin, ekran görüntüsü, menü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, menü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, menü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tokens.txt file:

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Another input:

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Output in the terminal:

metin, ekran görüntüsü, yazı tipi, menü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

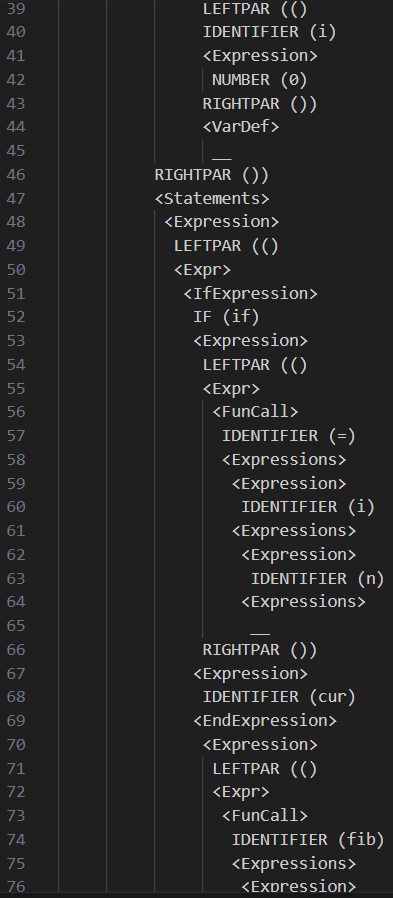
metin, yazı tipi, ekran görüntüsü, menü içeren bir resim

Açıklama otomatik olarak oluşturuldu

outputParse.txt file:

metin, ekran görüntüsü, menü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu



metin, ekran görüntüsü, menü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tokens.txt file:

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu