Real-Time Grammar-Based Syntax Highlighter

Proje Adı: Real-Time Grammar-Based Syntax Highlighter with GUI
Öğrenci: Sami Ramazan Erzincanlı
Numara: 23360859058
Ders: Programlama Dilleri

1. Giriş

Bu projede, JavaScript programlama dili için gerçek zamanlı syntax highlighting özelliği olan bir web tabanlı editör geliştirilmiştir. Proje, formal dilbilgisi kurallarına dayalı lexical analysis ve syntax analysis süreçlerini içermektedir. Sistem, kullanıcının yazdığı kodu anlık olarak analiz ederek syntax hatalarını tespit etmekte ve 8 farklı token tipini farklı renklerle vurgulamaktadır.

2. Programlama Dili ve Dilbilgisi Seçimi

2.1 Seçilen Programlama Dili: JavaScript

JavaScript seçilmesinin nedenleri:

- Yaygın kullanım alanı
- Zengin syntax yapısı (değişken tanımları, döngüler, koşullar, fonksiyonlar)
- Web tabanlı implementasyon için uygun

2.2 Tanımlanan Dilbilgisi

Projede JavaScript'in aşağıdaki yapıları için dilbilgisi tanımlanmıştır:

```
Program → Statement*

Statement → VarDeclaration | IfStatement | WhileStatement |
            DoWhileStatement | ForStatement | SwitchStatement |
            FunctionDeclaration | ReturnStatement | ExpressionStatement

VarDeclaration → ('var'|'let'|'const') IDENTIFIER '=' Expression ';'

IfStatement → 'if' '(' Expression ')' Block ['else' (Block | IfStatement)]

WhileStatement → 'while' '(' Expression ')' Block

DoWhileStatement → 'do' Block 'while' '(' Expression ')' ';'

ForStatement → 'for' '(' [VarDeclaration|Expression] ';' [Expression] ';'
               [Expression] ')' Block

SwitchStatement → 'switch' '(' Expression ')' '{' CaseClause* '}'

FunctionDeclaration → 'function' IDENTIFIER '(' Parameters ')' Block

ReturnStatement → 'return' [Expression] ';'

ExpressionStatement → Expression ';'

Expression → Assignment

Assignment → IDENTIFIER ('='|'+='|'-='|'*='|'/=') Assignment | LogicalOr

LogicalOr → LogicalAnd ('||' LogicalAnd)*

LogicalAnd → Equality ('&&' Equality)*

Equality → Relational (('=='|'!=') Relational)*

Relational → Additive (('<'|'>'|'<='|'>=') Additive)*

Additive → Term (('+'|'-') Term)*

Term → Factor (('*'|'/') Factor)*

Factor → NUMBER | STRING | IDENTIFIER | '(' Expression ')' |
         ArrayLiteral | FunctionCall
```

---

3. Lexical Analysis Detayları

3.1 Seçilen Yöntem

State Diagram & Program Implementation yöntemi seçilmiştir.

3.2 Token Tipleri

Sistem 8 farklı token tipini tanımaktadır:

1. KEYWORD - Anahtar kelimeler (var, let, const, if, else, function, return, while, do, for, switch, case, break, default)
2. NUMBER - Sayısal değerler (integer ve float)
3. IDENTIFIER - Değişken ve fonksiyon adları
4. OPERATOR - Operatörler (==, !=, <=, >=, &&, ||, ++, --, +=, -=, *=, /=, <, >, =, +, -, *, /)
5. SYMBOL - Semboller ({, }, ;, ', (,), !, ?, :, ., [,])
6. STRING - String literaller
7. COMMENT - Tek satır yorumlar (//)
8. WHITESPACE - Boşluk karakterleri

3.3 Tokenizer Implementasyonu

Regular expression tabanlı pattern matching kullanılarak tokenization gerçekleştirilmiştir.

```javascript
function tokenize(input) {
    const tokens = [];
    while (input.length > 0) {
        let matched = false;
        for (const pattern of patterns) {
            const match = input.match(pattern.regex);
            if (match) {
                tokens.push({ type: pattern.type, value: match[0] });
                input = input.slice(match[0].length);
                matched = true;
                break;
            }
        }
        if (!matched) {
            tokens.push({ type: "ERROR", value: input[0] });
            input = input.slice(1);
        }
    }
    return tokens;
}
```

4. Parsing

4.1 Seçilen Parser Tipi: Top-Down (Recursive Descent)

Recursive descent parser seçilmesinin nedenleri:

- Implementasyon kolaylığı
- Kod okunabilirliği
- Debug edilebilirlik

4.2 Parser Yapısı

Parser, her dilbilgisi kuralı için ayrı bir fonksiyon içeren recursive descent yaklaşımını kullanmaktadır:

- parseProgram() - Ana program yapısı
- parseStatement() - Statement türlerini yönlendirme
- parseVarDeclaration() - Değişken tanımları
- parseIfStatement() - If-else yapıları
- parseWhileStatement() - While döngüleri
- parseForStatement() - For döngüleri
- parseSwitchStatement() - Switch-case yapıları
- parseFunctionDeclaration() - Fonksiyon tanımları
- parseExpression() - İfade parsing'i (operator precedence ile)

5. Highlighting Scheme

5.1 Renk Şeması

Token Tipi	Renk	Stil
KEYWORD	Mavi	Kalın
NUMBER	Yeşil	Normal
IDENTIFIER	Mor	Normal
OPERATOR	Beyaz	Normal
SYMBOL	Turuncu	Normal
STRING	Kahverengi	Normal
COMMENT	Gri	İtalik
ERROR	Kırmızı arkaplan	Normal

6. GUI Implementation

6.1 Mimari Tasarım

GUI, overlay tekniği kullanılarak geliştirilmiştir:

- Textarea: Kullanıcı input'u için (şeffaf)
- Pre element: Renklendirilmiş kod gösterimi için
- Error box: Syntax hatalarını göstermek için

6.2 Özellikler

1. Real-time highlighting: Kullanıcı yazdıkça anlık renklendirme
2. Synchronized scrolling: Textarea ve pre element'i arasında senkronize scroll
3. Error display: Debounced error gösterimi (1 saniye gecikme ile)
4. Responsive design: Farklı ekran boyutlarına uyum

7. Sonuç

Bu proje, formal dilbilgisi kurallarına dayalı bir syntax highlighter'ın başarıyla geliştirildiğini göstermektedir. Lexical analysis ve syntax analysis süreçlerinin entegrasyonu ile real-time kod analizi gerçekleştirilmiştir. GUI, kullanıcı dostu bir interface sunmakta ve syntax hatalarını anlık olarak tespit etmektedir.