BURSA TEKNİK ÜNİVERSİTESİ

Gerçek Zamanlı Sözdizimi Vurgulayıcı

23360859068 - Alihan Gündoğdu

Bu projede temel amaç, kullanıcı tarafından girilen C benzeri ifadelerin anlık olarak analiz edilmesi, sözdizimi hatalarının tespit edilmesi ve uygun şekilde renkli vurgulanmasıdır. Bu süreci mümkün kılan ana bileşenler; lexical analysis (sözcüksel analiz), syntax analysis (sözdizimi analizi) ve real-time UI update (gerçek zamanlı arayüz güncellemesi) olarak ayrılabilir.

1. Dil Seçimi ve Gramer Yapısı

Proje için C benzeri basit bir ifade dili seçildi. Bunun nedeni hem yaygınlığı hem de karmaşıklık açısından orta seviyede olmasıdır. Desteklenen gramer şu şekilde:

Kural	Açıklama	Örnek
stmt → assign ;	Atama	int x = 5;
stmt → decl ;	Tanımlama	char a;
decl → keyword identifier	Tip + isim	int x
expr → number	Sayılar	5
expr → identifier	Değişken	x
expr → expr op expr	İşlem	x + 3
expr → (expr)	Parantezli ifade	(x + 3)

2. Sözcüksel Analiz (Lexical Analysis)

Girilen ham metin, lexer (tokenizer) tarafından küçük anlamlı parçalara bölünür. Bu aşamada:

- Regex ve karakter bazlı kontrollerle, girdideki her parça uygun bir token türüne (keyword, identifier, number, operator vb.) dönüştürülür.
- Tanımsız karakterlerle karşılaşılırsa anında hata üretilir.
- Token listesi, sonraki sözdizimi analizine temel sağlar.

Tercih edilen yöntem, basit ve hızlı çalışan regex tabanlı tokenizer'dır. Bu sayede gerçek zamanlı uygulamalarda performans sorunu yaşanmaz.

3. Sözdizimi Analizi (Syntax Analysis)

Lexing aşamasında elde edilen token listesi, Bottom-Up Parsing yöntemiyle analiz edilir. Bu yaklaşımın seçilme nedenleri şunlardır:

- Bottom-Up Parsing, ifadeyi en küçük yapı taşlarından (tokenlar) başlayarak yukarı doğru, karmaşık yapılara (örneğin stmt) doğru inşa eder.
- Kullanılan Shift-Reduce algoritması, bir yığın (stack) kullanır.
    - Shift: Yeni bir token yığına eklenir.
    - Reduce: Stack'teki elemanlar bir gramer kuralına uyduğunda, bunlar tek bir sembole indirgenir.

Bu süreç, kullanıcının yazdığı kod her değiştiğinde tekrar çalışır ve gerçek zamanlı olarak gramer uyumluluğunu kontrol eder. Böylece anında geri bildirim sağlanır.

Örnek Shift-Reduce İşlemi

Aşağıda, int x = 5 ; ifadesinin işlenişini gösteren bir tablo yer alıyor:

Adım	Stack Durumu	İşlem	Açıklama
1	[]	shift *int*	`int` token'ı stack'e eklendi
2	[int]	shift x	`x` (identifier) eklendi
3	[int, x]	reduce	`int x → decl`
4	[decl]	shift =	Eşittir eklendi
5	[decl, =]	shift *5*	`5` (number) eklendi
6	[decl, =, 5]	reduce	`5 → expr`
7	[decl, =, expr]	reduce	`decl = expr → assign`
8	[assign]	shift ;	Noktalı virgül eklendi
9	[assign, ;]	reduce	`assign ; → stmt`

Sonuç olarak ifade başarılı şekilde stmt'e indirgenmiştir. Bu, geçerli bir C benzeri ifade olduğunu gösterir.

4. Gerçek Zamanlı Çalışma ve Arayüz Entegrasyonu

Projede, kullanıcının her klavye hareketinde:

1. Girilen metin lexer'a gönderilir, token listesi üretilir.
2. Token listesi parser'a gönderilir, sözdizimi kontrolü yapılır.
3. Sonuçlar (başarılı veya hatalı) hemen UI'da gösterilir.
4. Tokenlara göre renk vurgulamaları anında güncellenir.

Bu döngünün hızlı ve akıcı olması için:

- Parser ve lexer hafif.
- React'in state yönetimi ile arayüz güncelleniyor.
- Hata mesajları ve uyarılar anlık olarak kullanıcıya iletiliyor.

5. Öne Çıkan Tercihler ve Nedenleri

- Basit gramer seçimi: Hem öğrenmesi kolay hem de temel sözdizimi kontrolü için yeterli.
- Regex tabanlı lexer: Hızlı, kolay uygulanabilir ve gerçek zamanlı için uygun.
- Shift-Reduce Bottom-Up Parser: Kompleks olmadan güçlü analiz imkanı.
- Gerçek zamanlı UI güncelleme: Kullanıcı deneyimi için önemli, anında geri bildirim veriyor.
- Hata ayıklama kolaylığı: Lexer ve parser aşamalarında ayrı hata mesajları veriliyor, sorun tespiti kolay.

6. Sonuç

Bu çalışma, gerçek zamanlı sözdizimi vurgulama ihtiyacını karşılamak için tasarlanmış, performans ve kullanılabilirlik dengesi gözetilmiş bir sistemdir. Tercih edilen yöntemler, hem akademik olarak sağlam temellere dayanmakta, hem de pratikte hızlı ve kullanıcı dostu sonuçlar sunmaktadır.