Gerçek Zamanlı, Dilbilgisi Tabanlı Syntax Highlighting Aracı

Bu doküman, geliştirilen sözdizimi renklendirme (syntax highlighting) aracının teknik detaylarını, kullanılan yöntemleri ve uygulama mimarisini açıklamaktadır.

1. Dil ve Gramer Seçimi

Proje için C-ailesi dillerinin (C++, Java, C#) temel özelliklerini barındıran, statik tipli ve prosedürel bir dil tasarlanmıştır. Bu dilin seçilmesinin temel nedeni; değişken tanımlamaları, kontrol yapıları, fonksiyonlar ve operatörler gibi derleyici tasarımının temel kavramlarını kapsamlı bir şekilde içermesidir.

Dilin Temel Özellikleri:

- **Veri Tipleri:** `int, float, string, bool` -- gibi temel veri tipleri bulunur.
- **Anahtar Kelimeler:** `if, else, while, for, return, print` -- gibi kontrol ve işlem anahtar kelimeleri mevcuttur. true ve false boolean sabitleri için kullanılır.
- **Kontrol Yapıları:** C-tarzı `if-else` koşul blokları, `for` ve `while` döngüleri desteklenmektedir.
- **Fonksiyonlar:** Dönüş tipi belirtilen, parametre alabilen ve `{}` (süslü parantez) ile gövdesi tanımlanan fonksiyonlar yazılabilir.

  Örnek: `int topla(int a, int b) { ... }.`
- **Diziler (Arrays):** tip[] formatında dizi tanımlama `(int[] dizi;)` ve `dizi[indeks]` formatında eleman erişimi desteklenir. Ayrıca `.length` gibi özelliklere erişim de ayrıştırıcı tarafından tanınır.
- **Operatörler:** Aritmetik `(+, -, *, /)`, mantıksal `(&&, ||, !)` ve karşılaştırma `(==, !=, <, >, <=, >=)` operatörleri tanımlanmıştır.
- **Yorumlar:** Tek satırlık `//` ve çok satırlı `/* ... */` yorum biçimleri desteklenir.
- **Sözdizimi Kuralları:** Her ifadenin (statement) sonunda noktalı virgül (;) bulunması zorunludur.

2. Sözdizimi Analiz Süreci

Uygulama, metin üzerinde gerçek zamanlı analiz yapmak için iki aşamalı bir süreç kullanır: **Sözcüksel Analiz (Lexical Analysis)** ve **Sözdizimsel Analiz (Syntactic Analysis/Parsing)**.

Bu süreç, kullanıcı arayüzündeki bir olaya (event) bağlı olarak tetiklenir:

1. **Tetikleme:** Kullanıcı klavyeden bir tuşu bıraktığında `(<KeyRelease>)` veya metin yapıştırdığında `(<<Paste>>)`, gui.py içerisindeki highlight metodu çağrılır.
2. **Sözcüksel Analiz:** highlight metodu, metin alanındaki tüm kodu alır ve lexer.py içerisindeki tokenize fonksiyonuna gönderir. Bu fonksiyon, metni Token adı verilen

anlamlı birimlere (anahtar kelime, operatör, sayı vb.) ayırır ve bir token listesi döndürür.

3. **Lexer Tabanlı Renklendirme:** GUI, bu token listesini döngüye alarak her bir token'ın tipine göre önceden tanımlanmış bir renk etiketi (tag) uygular. Bu, temel seviye renklendirmeyi sağlar.
4. **Sözdizimsel Analiz (Parsing):** Aynı token listesi, bu kez parser.py dosyasındaki Parser sınıfının bir örneğine verilir. `parser.parse()` metodu çağrılarak token'ların dilin gramer kurallarına uyup uymadığı kontrol edilir.
5. **Hata Tespiti ve Raporlama:**
    - Eğer kod, gramere tam olarak uyuyorsa, `parse()` metodu sessizce tamamlanır.
    - Eğer bir sözdizimi hatası varsa, parser bir ParseError istisnası fırlatır. Bu istisna, hatanın nerede ve neden olduğunu belirten bir mesaj ve hataya neden olan Token'ı içerir.
    - gui.py'deki `try...except` bloğu bu hatayı yakalar. Hata mesajı arayüzün altındaki etikette gösterilir ve hataya sebep olan token, özel bir hata rengiyle (altı çizili kırmızı arka plan) işaretlenir.

3. Sözcüksel Analiz Detayları (Lexical Analysis)

- **Yöntem:** Projede, ödev gereksinimlerinde belirtilen **"State Diagram & Program Implementation"** (Durum Diyagramı ve Programatik Uygulama) yöntemi kullanılmıştır. Lexer, herhangi bir kütüphane veya jeneratör kullanmadan, doğrudan Python kodu ile yazılmıştır.
- **Uygulama:** lexer.py dosyasındaki Lexer sınıfı, kaynak kodu karakter karakter işler.
    - `_current_char(), _peek_char() ve _advance()` gibi yardımcı metodlar ile kod üzerinde gezinir.
    - Ana tokenize döngüsü, mevcut karaktere göre hangi token'ın işleneceğine karar verir. Örneğin, " karakteri gördüğünde `handle_string` metodunu, bir rakam gördüğünde `handle_number` metodunu çağırır.
    - `handle_identifier` metodu bir kelimeyi okuduktan sonra, bu kelimenin keywords sözlüğünde olup olmadığını kontrol ederek anahtar kelime ve tanımlayıcı (identifier) ayrımını yapar.
    - Boşluklar ve yeni satır karakterleri `_skip_whitespace` metodu ile atlanır.
- **Token Yapısı:** Üretilen her bir Token, type, value, start_pos, end_pos, line ve column bilgilerini içeren bir veri sınıfıdır. Bu zengin yapı, hem renklendirme işleminin hem de hata raporlamanın hassas bir şekilde yapılmasını sağlar.

4. Ayrıştırma Metodolojisi (Parsing)

- **Yöntem:** Proje, **"Top-Down: Recursive Descent Parser"** (Yukarıdan Aşağıya: Özyinelemeli İniş Ayrıştırıcı) metodolojisini kullanır. Bu, parser.py dosyasındaki metodların birbirini çağırma yapısından açıkça görülmektedir.
- **Uygulama:** Parser sınıfı, dilin gramerindeki her bir kural (non-terminal) için bir metod içerir (örneğin if_statement, expression, term).
    - Ayrıştırma, en genel kural olan program ile başlar ve bu kural daha spesifik kuralları (declaration, statement vb.) çağırır. Bu özyinelemeli yapı, giriş token'ları için bir soyut sözdizimi ağacı (AST) oluşturur.
    - **Operatör Önceliği (Operator Precedence):** Parser, operatör önceliğini zarif bir şekilde yönetir. assignment -> logical_or -> equality -> comparison -> term -> factor -> unary şeklinde zincirleme metod çağrıları sayesinde, çarpma (*) gibi daha yüksek öncelikli işlemlerin toplama (+) gibi daha düşük öncelikli işlemlerden önce değerlendirilmesi sağlanır.
- **Hata Senkronizasyonu:** Parser, bir hata ile karşılaştığında çökmez. ParseError fırlatıldıktan sonra, `synchronize()` metodu devreye girer. Bu metod, bir sonraki `if`, `for` veya `;` gibi güvenilir bir başlangıç noktasına gelene kadar token'ları atlar. Bu, tek bir hatanın onlarca sahte hata üretmesini engeller.

5. Renklendirme Şeması (Highlighting Scheme)

Tüm renk ve stil tanımlamaları, modern bir "dark mode" estetiği sunmak amacıyla gui.py dosyasındaki ModernTheme sınıfı içinde merkezi olarak yönetilir.

- **Anahtar Kelimeler (`if, while` vb.):** Canlı ve okunaklı bir mavi tonu (#7aa2f7).
- **Özel Anahtar Kelime (`print`):** Diğerlerinden ayrışması için turuncu-sarı bir renk (#e0af68).
- **Veri Tipleri (`int, string` vb.):** Açık mavi/cyan (#7dcfff).
- **Sayısal Değerler (Literals):** Yeşil tonu (#9ece6a).
- **String Değerler (Literals):** Pembe/macenta tonu (#f7768e).
- **Operatörler:** Mor (#bb9af7).
- **Tanımlayıcılar ve Ayraçlar (`(, ), {, }`):** Yumuşak bir lavanta rengi (#c0caf5).
- **Yorumlar:** Koddan ayrışması için soluk ve koyu bir gri-mavi (#565f89).
- **Sözdizimi Hataları:** Hatalı token, pembe/macenta bir arka plan rengi (#f7768e) ve alt çizgi ile belirginleştirilir.

6. GUI Uygulaması

- **Çerçeve (Framework):** Arayüz, Python'ın standart tkinter kütüphanesi kullanılarak geliştirilmiştir.
- **Bileşenler:**
    - **Ana Pencere ve Metin Editörü:** tk.Tk ana penceresi ve içine yerleştirilmiş tk.Text bileşeni, uygulamanın temelini oluşturur. Tüm bileşenler ModernTheme sınıfındaki renklerle stilize edilmiştir.
    - **Satır Numaraları:** LineNumbers adında, tk.Text'ten türetilmiş özel bir sınıf yazılmıştır. Bu sınıf, ana metin editöründeki kaydırma, tuş basma ve pencere yapılandırma olaylarını dinleyerek kendi içeriğini senkronize bir şekilde günceller. Bu, satır numaralarının her zaman doğru kalmasını sağlar.
    - **Hata Mesajı Alanı:** Arayüzün altında bulunan bir ttk.Label, parser'dan gelen hata mesajlarını kullanıcıya anlık olarak göstermek için kullanılır.
- **Gerçek Zamanlı Güncelleme:** Uygulamanın en önemli özelliği, highlight metodunun metin bileşeninin `<KeyRelease>` ve `<<Paste>>` olaylarına bağlanmasıdır. Bu sayede kullanıcı metni her değiştirdiğinde analiz süreci otomatik olarak çalışır ve arayüz anında güncellenir.