Programlama Dilleri Proje Ödevi

Syntax Highlighter

Beyza Kahraman

22360859067

1. GİRİŞ

Bu proje, kullanıcı tarafından yazılan basit C benzeri program kodlarının **söz dizimi analizini (parsing)** yaparak geçerli veya hatalı olduğunu belirleyen, aynı zamanda söz dizimi elemanlarını renklendirerek kullanıcıya görsel geri bildirim sağlayan bir **söz dizimi renklendirici (syntax highlighter) ve parser** uygulamasıdır.

Grafik arayüzü Python'un **Tkinter** kütüphanesi kullanılarak geliştirilmiştir. Projenin temel amacı kodun söz dizimsel doğruluğunu anlık olarak kontrol edebilmektir. Proje, **C benzeri programlama dilinin basit bir alt kümesi** üzerinde çalışmaktadır.

2. DİL VE GRAMER SEÇİMİ

2.1 Hedef Dil

Proje kapsamında kullanılan hedef dil, C diline benzer bir yapıya sahip olup aşağıdaki temel özelliklere sahiptir:

- Temel veri tipleri: int, float, double
- Değişken bildirimi ve ataması
- Print fonksiyonu (string ifadeler için)
- Koşul ifadeleri: if ve isteğe bağlı else
- Döngüler: while ve for
- Fonksiyon tanımlamaları (parametreli ve gövdeli)
- Basit operatörler ve ifadeler
- Tek satırlık yorum satırları

2.2 Dilin Söz Dizimi Kuralları (CFG)

Projenin parser kısmında kullanılan basit bağlam serbest gramer (CFG) kuralları aşağıdaki gibidir:

```
# CFG
# statement -> print_stmt|if_else_stmt|assign_stmt|typed_var_decl|while_stmt|for_stmt;
# print_stmt -> 'print' '(' string ')' ';'
# if_stmt -> 'if' '(' condition ')' '{' statement '}' ['else' '{' statement'}']
# condition -> identifier operator number ';'
# assign_stmt -> identifier '=' number ';'
#statement -> type identifier = number ;
#type -> 'int' | 'float' | 'double';
#function_decl -> type identifier '(' parameters ')' '{' statement '}';
#parameters -> [ type identifier { ',' type identifier } ];
#while_stmt -> 'while' '(' condition ')' '{' statement '}';
#for_stmt -> 'for' '(' init ';' condition ';' update ')' '{' statement '}';
#init -> typed_variable_decl;
#update -> identifier '=' expression;
```

3. Sözcüksel Analiz Süreci (Lexical Analysis)

3.1 Amaç ve İşleyiş

Sözcüksel analiz (lexer), kullanıcı tarafından girilen kaynak kodu anlamlı **token** parçalarına (kelime, sayı, operatör vb.) ayıran ilk aşamadır. Projede kullanılan lexer, Python'un re modülü ile yazılmış bir **düzenli ifadeler (regex)** tabanlı token ayırıcıdır.

3.2 Token Türleri ve Düzenli İfadeler

Projede tanımlı token türleri ve örnek regex desenleri:

Token Türü	Açıklama	Regex Örneği
Single_comment	Tek satırlık yorum	//.*
Float_number	Ondalıklı sayı	\d+\.\d+
Number	Tam sayı	\d+
String	Çift tırnak içinde metin	"[^"]*"

identifier	Değişken/fonksiyon isimleri	[a-zA-Z_]\w*
Operatör	Atama, karşılaştırma operatörleri	==|!=|<=|>=|[+\-*/=<>]
Delimiter	Parantez, süslü parantez, noktalı virgül	[();{}]
Whitespace	Boşluk, tab, yeni satır	\s+

3.3 Anahtar Kelimeler

Lexer, identifier tokenlarını ayrıca keyword (anahtar kelime) olarak sınıflandırmakta, önceden tanımlı anahtar kelimeleri (if, else, print, int, float, double, for, while) tespit etmektedir.

3.4 Fonksiyon Tanıma

Bir identifier tokenının hemen ardından "(" karakteri geliyorsa, bu token fonksiyon adı olarak işaretlenir.

4. Ayrıştırma Metodolojisi (Parsing)

4.1 Amaç ve Kullanılan Yöntem

Ayrıştırma (parsing) aşamasında, lexer tarafından üretilen token dizisinin, dilin sözdizimi kurallarına uygun olup olmadığı kontrol edilir. Bu proje için **top-down recursive descent parsing** yaklaşımı tercih edilmiştir. Her dil yapısı için (örneğin print ifadesi, if bloğu, while döngüsü) ayrı parse fonksiyonları yazılmıştır.

4.2 Parser Yapısı

- **Global İndeks:** Tokenlar arasında gezinmek için global bir indeks (i) kullanılır.
- **Match Fonksiyonu:** Beklenen tip ve değer ile mevcut token karşılaştırılır, uyuyorsa ilerlenir.
- **Try Parse Fonksiyonu:** Birden fazla olasılık denemek için fonksiyonlar sırasıyla çağrılır, başarısız olan durumlarda indeks geri alınır.

- **Her Bileşen İçin Ayrı Fonksiyon:** parse_print_stmt(), parse_if_stmt(), parse_assign_stmt() gibi.

4.3 Desteklenen Yapılar

- Print ifadeleri
- Basit atama işlemleri
- Tip bildirimleri (int a = 5;)
- Koşul ifadeleri (if-else)
- Döngüler (while, for)
- Fonksiyon tanımları ve parametre listeleri
- Tek satırlı yorum satırları

4.4 Hata Yönetimi

Eğer token akışı dil kurallarına uymuyorsa, parser False döner ve GUI üzerinde **"Syntax Error"** uyarısı gösterilir.

5. Vurgulama (Syntax Highlighting) Şeması

5.1 Amaç

Kod okunabilirliğini artırmak için farklı token türlerine farklı renkler atanarak kullanıcıya görsel destek sağlanır. Bu kısım yazdığımız lexer fonksiyonu ve belirlenen renklere göre çalışır.

Bu projede toplam 8 token tipi belirlenmiştir ve 8 ayrı renk ataması yapılmıştır. Token tiplerinden biri olan keywords ise 8 ayrı kelime grubunu içinde barındırır. Bunların kontrolleri de lexer fonksiyonu içerisinde yapılmıştır.

5.2 Renk Atamaları

Token Türü	Renk	Açıklama
keyword	Mavi	Anahtar kelimeler
identifier	Siyah	Değişken ve fonksiyon adları
number	Koyu Turuncu	Sayılar (int, float)
string	Koyu Cam Göbeği	String ifadeler
operator	Kırmızı	Atama ve karşılaştırma operatörleri
delimiter	Gri	Parantez, süslü parantez, noktalı virgül
function	Mor	Fonksiyon isimleri
single_comment	Yeşil	Tek satır yorumlar

5.3 Uygulama

Tkinter'ın Text widget'ında her token türü için ayrı bir **tag** oluşturulup, uygun renk atanır. Kod içerisinde token pozisyonlarına göre tag_add ile etiketler uygulanır.

6. Kullanıcı Arayüzü (GUI)

- Tkinter kullanılarak temel bir metin düzenleyici oluşturulmuştur.
- Kullanıcı kod yazarken 300 ms aralıklarla otomatik olarak lexing ve parsing işlemi yapılır.
- Parse sonucu başarılı ise "Syntax OK" yeşil renkle, hata varsa "Syntax Error" kırmızı renkle gösterilir.
- "Temizle" butonu metni ve vurgulamaları temizler.

7. Sonuç

Bu proje, C benzeri basit bir programlama dilinin temel sözdizimi kurallarını destekleyen ve anlık söz dizimi kontrolü yapabilen bir syntax highlighter ve parser uygulaması sunmaktadır.

Kullanıcı dostu arayüzü ile, yazılan kodun anlık olarak sözdizimsel doğruluğu kontrol edilmekte ve hatalar görsel olarak kullanıcıya bildirilmektedir.

Ek Notlar

- Proje sadece tek satırlık ya da tek statement içeren blokları desteklemektedir. Çoklu statement ve kapsamlı ifadeler genişletilebilir.
- Parser, hataları detaylı konum ve tür bazında raporlamamaktadır, geliştirilebilir.
- Proje yalnızca tek satırlı yorum satırlarını algılamaktadır.