Gerçek Zamanlı Dilbilgisi Tabanlı Sözdizim Vurgulayıcı

Ahmet Furkan Öcel: 23360859729

Dil ve Gramer Seçimi

Bu projede, gerçek zamanlı sözdizimi vurgulaması (syntax highlighting) ve sözdizimi analizi (parsing) yapılabilmesi amacıyla PCAL adını verdiğim özgün bir programlama dili tasarlanmıştır. PCAL dili, temel matematiksel işlemleri gerçekleştirebilen, sade ve öğrenilmesi kolay bir yapıya sahiptir. Dil, kullanıcıların temel aritmetik ifadeler yazabilmesini, değişken tanımlamalarını ve basit kontrol yapılarının kullanımını desteklemektedir. PCAL dilinin dosya uzantısı ".pcal" olarak belirlenmiştir.

PCAL dilinde, sözdizimi kuralları ve gramer yapısı, context-free grammar biçiminde tanımlanmıştır. PCAL diline ait tanımlanmış 7 adet token vardır. Bunlar aşağıda verilmiştir:

- **IDENTIFIER:** Harflerden oluşan değişken veya fonksiyon isimleri (örneğin: x, sum)
- **NUMBER:** Sayısal değerler (örneğin: 123, 45)
- **OPERATOR:** Aritmetik operatörler (+, -, *, /)
- **EQUALS:** Atama operatörü (=)
- **QUESTION:** Sonuç için soru işareti (?)
- **DELIMITER:** Satır sonu için virgül işareti (,)
- **PAREN:** Parantezler ((,))

Örnek bir PCAL ifadesi aşağıdaki gibidir:

x = 444,

d = 66,

c = 110,

l = 40,

p = 20,

l / (x - c + d) * p = ?

Ekran çıktısı:
? = 2.0

Bu dil ve gramerin seçilmesindeki temel amaç, sözdizimi analizi ve vurgulama işlemlerinin daha sade ve anlaşılır olmasıdır. Hazır diller yerine özgün bir dil tasarlamak, projede kullanılan yöntemlerin (lexer ve parser) etkinliğini göstermek açısından avantaj sağlamaktadır. Ayrıca PCAL dili, gerçek zamanlı analiz için uygun yapıya sahip olduğundan, sözdizimi vurgulama uygulaması için ideal bir örnek teşkil etmektedir.

Söz Dizimi Analiz Süreci

Projede, PCAL adlı kendi oluşturduğum dilin sözdizimi analizi Top-Down (recursive descent) yöntemi ile yapılmıştır.

Parser, girdiyi token dizisine ayıran lexer'dan aldığı token listesini işler. PCAL dilinde temel yapılar şunlardır:

- **Değişken Atamaları:** IDENTIFIER = NUMBER, veya IDENTIFIER = EXPRESSION, şeklindedir. Virgül ile birden fazla atama art arda yapılabilir.
- **Matematiksel İfadeler:** Toplama (+), çıkarma (-), çarpma (*), bölme (/) operatörleri ve parantezler ("(" , ")") kullanılabilir.

- **Sonuç İsteği:** EXPRESSION = ? yapısı ile bir ifadenin sonucu istenir.

Parser, bu yapıları ayrıştırmak için üç temel fonksiyon kullanır:

- parse_expression(): Toplama ve çıkarma işlemlerini çözümler.
- parse_term(): Çarpma ve bölme işlemlerini çözümler.
- parse_factor(): Sayılar, değişkenler ve parantez içi ifadeleri çözümler.

Parser, değişkenlerin değerlerini bir sözlükte saklar ve bir ifadeyi çözümlerken bu değerleri kullanır. Tanımlanmamış değişken kullanımı veya beklenmedik token karşılaşıldığında ParseError fırlatılarak hata yönetimi sağlanır.

Sonuç olarak, Top-Down yaklaşımı ile yazılan bu parser, PCAL dilinin kurallarına uygun olarak ifadeleri doğru şekilde çözümler ve kullanıcıya geri bildirim verir.

Sözcüksel Analiz Detayları

Projemde sözcüksel analiz (lexical analysis) işlemi, kullanıcıdan alınan PCAL diline ait kaynak kodunu küçük anlamlı parçalara, yani tokenlara ayırmak için tasarlanmıştır.

Kullanılan Yöntem

Sözcüksel analiz için **State Diagram & Program** Implementation yöntemi seçilmiştir. Burada, programlama ile doğrudan düzenli ifadeler (regex) kullanılarak tokenlar tanımlanmış ve kod üzerinde adım adım ilerlenmiştir.

Kendi tasarımım olan PCAL dilinin sözcüksel (lexikal) yapısı, aşağıdaki token türlerinden oluşmaktadır:

```
TOKEN_TYPES = {
    'IDENTIFIER': r'[a-zA-Z]+',
    'NUMBER': r'\d+',
    'OPERATOR': r'[+\-*/]',
    'EQUALS': r'=',
    'QUESTION': r'\?',
    'DELIMITER': r',',
    'PAREN': r'[\(\)]',
}
```

IDENTIFIER: Harflerden oluşan değişken isimleri, matematiksel işlemlerde kullanılıyor.
Örneğin: “x” , “number” , “degisken” , ... , vb.

NUMBER: Tam sayı olan sayısal değerler. Değişken isimlerine atanarak veya daha basit işlemlerde bu tür atamalara dahi gerek kalmadan kullanılabilen ifadeler.
Örneğin: “7” , “104” , “0” , ... , vb.

OPERATOR: Matematiksel operatörler. Temel matematiksel işlemlerde kullanılıyor.
Örneğin: “+” , “-” , “*” , “/” , ... , vb.

EQUALS: Eşittir işareti. Değer atamalarında kullanılır. ? ile birlikte kullanıldığında sonucu verir.
Örneğin: “=”

QUESTION: Soru işareti. Sonucu döndürür.
Örneğin: “?”

DELIMITER: C dilindeki “;” gibi satır sonunda kullanılır. O satırın bittiğini ifade eder.
Örneğin: “,”

PAREN: Parantezler. Matematiksel işlemlerde; çarpma ve bölmenin, toplama ve çıkarmaya göre önceliği vardır. Toplama veya çıkarma işlemlerini daha önce yapmak istediğimizde parantezleri kullanabiliriz. Parantezler; çarpma, bölme, toplama ve çıkarma operatörlerine göre daha önceliklidir.
Örneğin: “(” , “)”

Sözcüksel Analiz Fonksiyonu

lexer fonksiyonu, verilen kaynak kodunu baştan sona tarayarak her karakteri uygun token türüyle eşleştirmeye çalışır. İşlem, aşağıdaki önemli adımları içerir:

- Kodun her pozisyonunda TOKEN_TYPES sözlüğündeki regex desenleri ile eşleşme aranır.
- Eğer bir eşleşme bulunursa, eşleşen ifade bir token olarak kaydedilir.
- Boşluk ve boş karakterler token olarak işlenmez ve atlanır.
- Tanımlanamayan karakterlerle karşılaşılırsa hata (exception) fırlatılır.

```
def lexer(code):
    tokens = []
    index = 0
    while index < len(code):
        match = None
        for token_type, pattern in TOKEN_TYPES.items():
            regex = re.compile(pattern)
            match = regex.match(code, index)
            if match:
                value = match.group(0)
                tokens.append({
```

```
                    'type': token_type,
                    'value': value,  # Burada value "(" veya ")" olacak
                    'start': index,
                    'end': index + len(value)
                })
                index += len(value)
                break
        if not match:
            if code[index].isspace():
                index += 1
                continue
            raise ValueError(f"Invalid character: {code[index]}")
    return tokens
```

Bu yöntem sayesinde PCAL diline ait kaynak kodu, sonraki aşamalarda işlenmek üzere doğru ve anlamlı tokenlar halinde ayrıştırılmış olur.

Parsing Metodolojisi

Projemde sözcüksel analiz sonucunda elde edilen token akışını işleyip, anlamlı yapılar haline getirmek için **Top-Down Parsing** yaklaşımı uygulanmıştır. Bu yöntem, verilen token dizisini üstten aşağı doğru (öncelikle en dış yapıyı çözerek) analiz eder ve parse ağacını pre-order (önden) şeklinde izler.

Kullanılan Yöntem: Recursive Descent Parser

Top-down parsing için klasik bir yöntem olan ***Recursive Descent Parsing*** kullanılmıştır. Bu yöntem, dilin gramer yapısına göre her üretim kuralı için bir fonksiyon yazmayı gerektirir. Fonksiyonlar, tokenları sırasıyla kontrol eder, doğru yapıların olup olmadığını denetler ve uygun hata mesajlarını döndürür.

Parser Yapısı ve İşleyişi

Parse fonksiyonu, token listesini alır ve değişken atamaları ile matematiksel ifadeleri yorumlar. Fonksiyon içinde:

- expect fonksiyonu ile beklenen token tipi ve değer kontrol edilir. Uyum sağlanmazsa hata fırlatılır.
- parse_expression, parse_term ve parse_factor fonksiyonları ile matematiksel ifadeler sırasıyla işlem önceliğine göre ayrıştırılır. (Örneğin, parse_term çarpma ve bölme işlemlerini, parse_expression toplama ve çıkarma işlemlerini işler.)
- Parantezler ve değişkenler uygun şekilde işlenir.
- Değişken atamaları (x = 10,) ve değer sorgulamaları (x = ?) desteklenir.

```
    def parse_expression():
        nonlocal pos
        val = parse_term()
        while pos < len(tokens) and tokens[pos]['type'] == 'OPERATOR'
and tokens[pos]['value'] in ('+', '-'):
            op = tokens[pos]['value']
            pos += 1
            right = parse_term()
            if op == '+':
                val += right
            else:
                val -= right
        return val
```

Örneğin yukarıdaki fonksiyon, toplama ve çıkarma işlemlerini çözerek ifadeyi anlamlandırır.

Hata Yönetimi

Kodda **ParseError** özel istisnası tanımlanmıştır. Yanlış ya da beklenmedik token dizisi durumunda, kullanıcıya anlamlı hata mesajları döndürülür. Bu sayede sözdizimi hatalarının takibi kolaylaşır.

Vurgulama Şeması

Projemde, kullanıcı deneyimini artırmak ve kodun okunabilirliğini kolaylaştırmak amacıyla, yazılan PCAL kodlarının sözcüksel öğeleri (tokenlar) renklerle vurgulanmaktadır. Bu, özellikle karmaşık ifadelerde farklı türdeki kelimeleri hızlıca ayırt etmek için faydalıdır.

Kullanılan Token Türleri ve Renk Atamaları

Lexer aşamasında tanımlanan token türleri ve bu türlerin her biri için belirlenen renkler aşağıdaki gibidir:

```
def get_colors(is_dark_mode):
    if is_dark_mode:
        return {
            "IDENTIFIER": "sky blue",
            "NUMBER": "green",
            "OPERATOR": "orange",
            "EQUALS": "orange",
            "QUESTION": "purple",
            "DELIMITER": "purple",
            "PAREN": "magenta",
        }
    else:
        return {
            "IDENTIFIER": "blue",
            "NUMBER": "green",
            "OPERATOR": "orange",
            "EQUALS": "orange",
```

```
            "QUESTION": "purple",
            "DELIMITER": "purple",
            "PAREN": "magenta",
        }
```

Kullanıcı arayüzümde ekstradan light mode ve dark mode özelliklerini de tanımladım. Bu sayede kullanıcı deneyimini iyileştirmeyi hedefledim. Tokenlarıma renk atamaları aşamasında IDENTIFIER için “blue” rengini uygun gördüm fakat dark modda tam görünmüyordu. O nedenle dark mode için IDENTIFIER için tanımlamış olduğum bu “blue” rengini özel olarak daha açık renkte olan ve dark mode için daha uyumlu “sky blue” olarak değiştirdim. Bunu yukarıdaki fonksiyondan da görebilirsiniz.

IDENTIFIER: light mode için *“blue”* , dark mode için *“sky blue”*

NUMBER: *“green”*

OPERATOR: *“orange”*

EQUALS: *“orange”*

QUESTION: *“purple”*

DELIMITER: *“purple”*

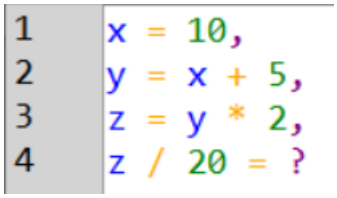
PAREN: *“magenta”*

Yukarıda 7 farklı token için ayarladığım renkleri görebilirsiniz. Burada bazı tokenlar için aynı renkleri kullanmaya karar verdim. 7 ayrı token için toplamda 5 ayrı rengin yeterli olduğunu düşündüm.

Bu renkler, grafik kullanıcı arayüzünde (GUI) kullanıcı tarafından yazılan PCAL kodunun her öğesine uygulanarak okunabilirliği artırır.

Vurgulama Yöntemi

Renk ataması, lexer tarafından üretilen token listesindeki her token türüne karşılık gelen renk kodlarıyla gerçekleştirilir. GUI katmanında, ilgili tokenların yazı tipi rengi değiştirilerek gösterilir. Örneğin, aşağıda bir PCAL kodunu ve nasıl renklendirildiğini görebilirsiniz:

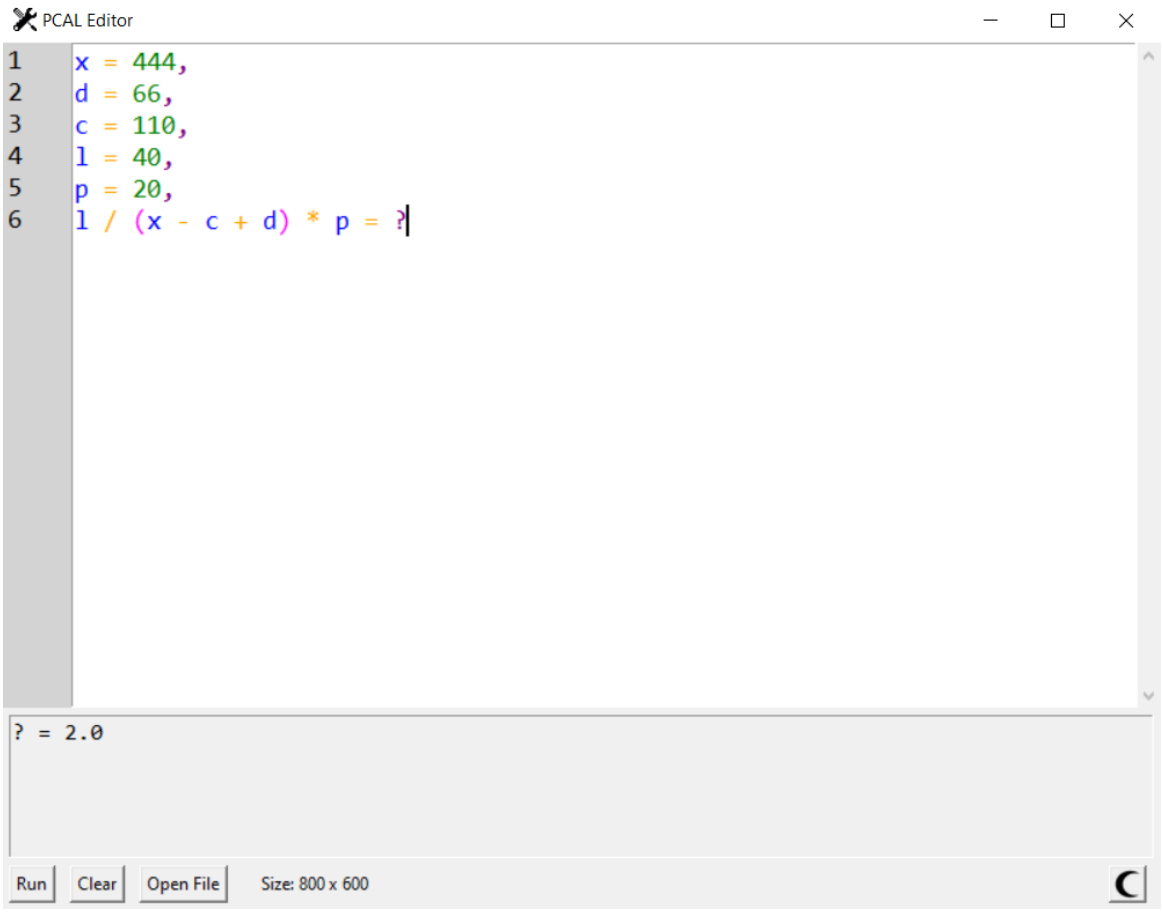


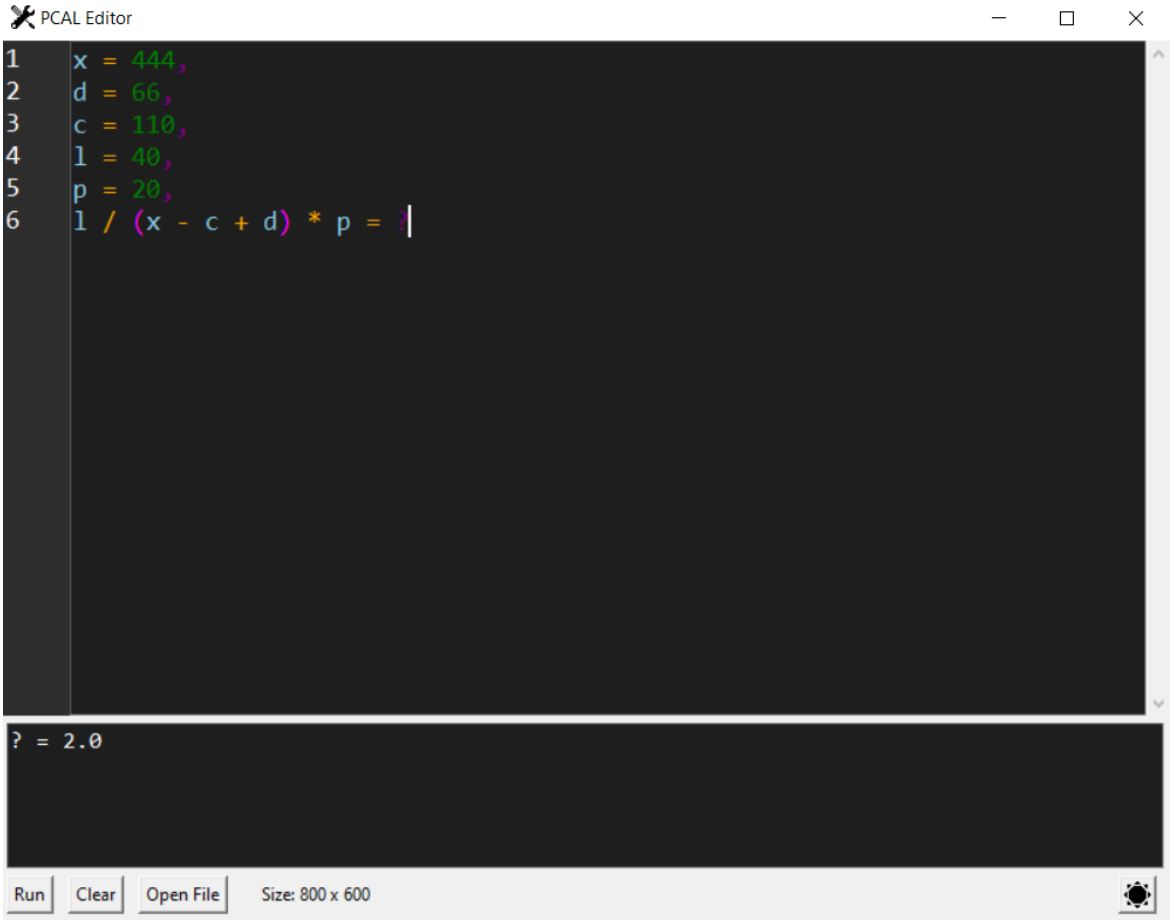
Bu görsel ayrım, kullanıcıların kodu daha hızlı anlamasını sağlar.

GUI Uygulaması

Proje kapsamında kullanıcı dostu ve işlevsel bir grafiksel kullanıcı arayüzü (GUI) geliştirilmiştir. Bu arayüz sayesinde kullanıcılar, PCAL dilinde yazdıkları kodları rahatça yazabilir, düzenleyebilir ve anlık olarak sonuçlarını görebilirler.

Aşağıda kullanıcı arayüzününün bir örnekle beraber hem light mode hem de dark mode görselleri verilmiştir:


PCAL Editor
1 x = 444,
2 d = 66,
3 c = 110,
4 l = 40,
5 p = 20,
6 l / (x - c + d) * p = ?
? = 2.0
Run
Clear
Open File
Size: 800 x 600


PCAL Editor
1 x = 444,
2 d = 66,
3 c = 110,
4 l = 40,
5 p = 20,
6 l / (x - c + d) * p = ?
? = 2.0
Run
Clear
Open File
Size: 800 x 600

Amaç

Kullanıcının .pcal uzantılı dosyalarla doğrudan çalışmasını kolaylaştırmak, ayrıca komut satırı yerine görsel bir ortam sunarak etkileşimli deneyimi artırmak amaçlanmıştır. GUI uygulaması, lexer ve parser bileşenleri ile entegre çalışarak kullanıcının yazdığı kodu analiz eder ve sonucu ekranda gösterir.

Teknolojik Alt Yapı

GUI uygulaması **Python** programlama dili ile yazılmıştır ve arayüz için **Tkinter** kütüphanesi kullanılmıştır.

Arayüz Bileşenleri

- **Kod Editörü Alanı:** Kullanıcının PCAL dilinde kod yazabileceği metin kutusudur. Burada yazılan kod lexer tarafından işlenerek renklendirilir.
- **Çalıştır Butonu:** Yazılan kodun derlenmesini ve analiz edilmesini sağlar. Butona basıldığında kod lexer ile token'lara ayrılır, ardından parser ile sözdizimi kontrolü ve hesaplamalar yapılır.
- **Sonuç Alanı:** Hesaplama sonucu burada görüntülenir. Örneğin "x = 10, y = 5, x + y = ?" gibi bir kodda ? işaretli ifade çözülerek sonucun gösterilmesi sağlanır.
- **Dosya Aç/Kaydet:** Kullanıcının .pcal uzantılı dosyaları açmasına ve oluşturduğu kodları kaydetmesine olanak tanır.
- **Vurgulama (Highlighting):** Kod editörü alanında yazılan her sözcük türü lexer’dan gelen token tipine göre renklendirilerek

kullanıcıya daha anlaşılır bir görünüm sunulur.

Görsel Geri Bildirim

Kod hatalıysa (örneğin sözdizimi kurallarına uymuyorsa) kullanıcıya uygun bir hata mesajı verilir: "Expected '=', but found ','" gibi. Bu sayede kullanıcı yazdığı kodda nerede hata olduğunu görebilir.

Github Linki:

https://github.com/AFurkanOcel/Realtime_Syntax_Highlighter_PCAL

Youtube Linki:

https://youtu.be/eCkWtttOFr0