

Gerçek Zamanlı Gramer Tabanlı Sözdizimi Vurgulayıcı (GUI Destekli)
(Real-Time Grammar-Based Syntax Highlighter with GUI)

MUHAMMED FATİH GÖRAL

22360859032

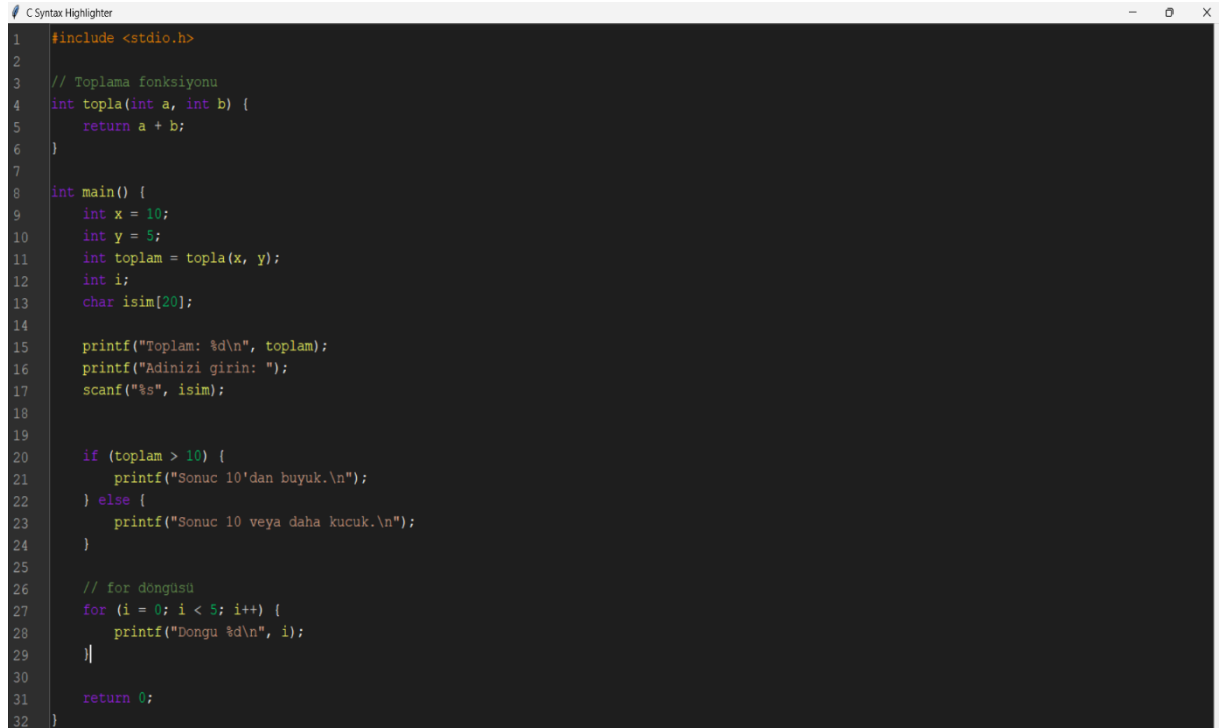
DERS:

Programlama Dilleri

Proje Özeti

Bu proje, yazılım geliştirme sürecinde kod yazımını hem görsel hem de yapısal olarak destekleyen, gerçek zamanlı bir sözdizimi vurgulayıcı (syntax highlighter) ve kullanıcı dostu bir grafiksel arayüz (GUI) geliştirmeyi amaçlamaktadır. Geliştirilen uygulama, kullanıcı tarafından yazılan kodu anlık olarak analiz eder; sözdizimsel yapıları tanıır ve önceden tanımlanmış bir renk şemasına göre renklendirerek kullanıcıya görsel geri bildirim sunar. Sistem; sözcük düzeyi analiz (lexical analysis), sözdizimi denetimi (syntax analysis), vurgulama mekanizması (highlighting) ve kullanıcı arayüzü (GUI) olmak üzere dört temel bileşen üzerine inşa edilmiştir.

Projede herhangi bir hazır sözdizimi motoru veya editör kütüphanesi kullanılmaksızın, tüm analiz ve vurgulama işlemleri tamamen manuel olarak geliştirilmiştir. Bu yaklaşım, teknik özgünlüğü artırırken aynı zamanda algoritmik düşünme, dil işleme (text processing) ve editör mantığı gibi konularda önemli bir mühendislik pratiği sunmaktadır. Uygulama, programlama eğitimi ve editör geliştirme alanlarında etkili bir öğrenme aracıdır. Kullanıcılar hem doğru sözdizimiyle kod yazmayı hem de dilin yapısal kurallarını sezgisel olarak kavrama becerisi kazanı



```
1 #include <stdio.h>
2
3 // Toplama fonksiyonu
4 int toplama(int a, int b) {
5     return a + b;
6 }
7
8 int main() {
9     int x = 10;
10    int y = 5;
11    int toplam = toplama(x, y);
12    int i;
13    char isim[20];
14
15    printf("Toplam: %d\n", toplam);
16    printf("Adınızı girin: ");
17    scanf("%s", isim);
18
19
20    if (toplam > 10) {
21        printf("Sonuc 10'dan büyük.\n");
22    } else {
23        printf("Sonuc 10 veya daha küçük.\n");
24    }
25
26    // for döngüsü
27    for (i = 0; i < 5; i++) {
28        printf("Döngü %d\n", i);
29    }
30
31    return 0;
32 }
```

Resim1

Resim1: Uygulamanın başlangıç ekranı. Gerçek zamanlı sözdizimi vurgulayıcı, C dilinde yazılan kodu anlık olarak analiz ederek farklı öğeleri renklendirmekte ve kullanıcıya görsel geri bildirim sunmaktadır

1. Dil ve Gramer Seçimi

Bu proje, belirli bir programlama diline bağlı kalmadan, özelleştirilebilir bir gerçek zamanlı sözdizimi vurgulayıcı geliştirmeyi amaçlar. Ancak, örnek bir uygulama olarak C programlama dilinin temel yapıları (anahtar kelimeler: int, if, return; fonksiyon çağrıları: printf, scanf; sayısal/metinsel sabitler, yorum satırları, önışlemci direktifleri: #include, operatörler) analiz birimi olarak ele alınmıştır. C dilinin statik ve yapılandırılmış sözdizimi, sözdizimi renklendirme ve ayrıştırma işlemleri için uygun bir temel sunar. Gramer tasarımı, "ifade odaklı" (statement-oriented) bir yaklaşımla yapılandırılmıştır. Bu yaklaşım, programlama dillerinin temel yapı taşlarını (kontrol ifadeleri, değişken tanımlamaları, fonksiyonlar) analiz birimi olarak ele alır ve esnek bir altyapı sunar. Kullanıcı dostu bir tasarım hedefiyle, gramer yapısı hem yorumlanabilir hem de görsel olarak ayırt edilebilir şekilde düzenlenmiştir; böylece kullanıcılar kod yapısını hızlıca anlayabilir ve hataları erken fark edebilir.

Proje, Python programlama dili kullanılarak geliştirilmiştir. Python'un seçilmesinin temel nedenleri arasında hızlı prototipleme imkanı, Tkinter gibi yerleşik bir GUI kütüphanesine sahip olması ve düzenli ifadeler (regex) ile metin işleme yeteneklerinin güçlü olması yer alır. Python, hem metin işleme hem de arayüz geliştirme işlemlerini aynı yapı içinde kolayca gerçekleştirmeyi mümkün kılarak projenin taşınabilir, sade ve modüler bir yapıda olmasını sağlamıştır. Ayrıca, Python'un farklı programlama dillerine uyulanabilirliği destekleyen esnek yapısı, projenin gelecekte diğer diller için genişletilmesine olanak tanır.

2. Sözdizimi Analizi Süreci

Sözdizimi analizi, kullanıcı tarafından yazılan kodun dilbilgisel açıdan geçerli olup olmadığını kontrol eden temel bir süreçtir. Bu projede, recursive descent (yinelemeli iniş) yaklaşımına dayalı bir Top-Down parser sistemi kullanılmıştır. Bu yöntem, her bir ifadeyi adım adım analiz ederek önceden tanımlanmış gramer kurallarına göre doğruluğunu kontrol eder ve hem anlaşılır bir yapı sunar hem de karmaşık ifadelerin analizini kolaylaştırır.

Analiz süreci, kodun her bir satırını ve yapısını detaylı bir şekilde inceler. Örneğin, bir if yapısının analizi şu adımlarla gerçekleştirilir:

1. if anahtar kelimesinin varlığı kontrol edilir.
2. Açma parantezi (ve ardından gelen ifade (Expression) değerlendirilir; bu ifade bir karşılaştırma (örneğin, $x > 5$) veya bir değişken olabilir.
3. Kapatma parantezi) kontrol edilir.
4. Yapının gövdesi (BlockOrStatement) analiz edilir; bu, bir blok ({ ... }) veya tek bir ifade olabilir.
5. Eğer bir else yapısı varsa, onun da doğruluğu kontrol edilir.

Tanımlı gramer kuralları, temel yapı taşlarını kapsayacak şekilde şu şekilde belirlenmiştir:

- **Statement** → if (Expression) BlockOrStatement [else BlockOrStatement]
- **Statement** → while (Expression) BlockOrStatement
- **Statement** → return [Expression] ;
- **Statement** → (int | string | void) Identifier [= Expression] ;
- **Statement** → Identifier = Expression ;
- **Expression** → Term { (+ | - | == | != | > | < | >= | <=) Term }
- **Term** → Factor { (* | /) Factor }
- **Factor** → Number | Identifier | String | (Expression)
- **BlockOrStatement** → { Statement } | Statement

Her yapı, parse_if(), parse_while(), parse_return() gibi özel tanımlı fonksiyonlarla analiz edilir. Geçerli olarak algılanan yapılar, mark_tokens() fonksiyonu ile "valid_syntax" etiketi altında işaretlenir. Bu etiket, kullanıcıya görsel olarak yansıtılmasa da, yapının derlenebilirliğine dair içsel bir doğrulama mekanizması sağlar. Hatalı yapılar bu etiketi almaz, ancak kullanıcıya doğrudan bir hata mesajı gösterilmez. Bu yaklaşım, kullanıcı deneyimini kesintiye uğratmadan yalnızca doğru yapıların vurgulanmasını sağlar ve hatalı yapıların fark edilmesini kolaylaştırır.

3. Sözcük Düzeyi Analiz (Lexical Analysis)

Sözcük düzeyi analiz (lexical analysis), kullanıcı tarafından girilen kaynak kodun satır bazında taranarak her yazı parçasının (token) hangi kategoriye ait olduğunun belirlenmesini sağlayan süreçtir. Bu işlem, tokenize() fonksiyonu aracılığıyla gerçekleştirilir. Fonksiyon, kodda yer alan öğeleri tanımlar ve her birini belirli bir sınıfa (etikete) atar. Bu sınıflandırma, hem renklendirme hem de sözdizimsel doğrulama işlemleri için temel bir altyapı sunar.

Projede tanımlanan başlıca token türleri şunlardır:

- **Anahtar Kelime (keyword):** int, if, return gibi programlama diline özel sözcükler.
- **Tanımlayıcı (identifier):** Değişken veya fonksiyon adları, örneğin x, toplam, hesapla.
- **Sayı (number):** Tam sayılar veya ondalıklı sayılar, örneğin 42, 3.14.
- **Metin (string):** Çift tırnak içinde yazılan sabit metinler, örneğin "Merhaba".
- **Yorum (comment):** // veya /* ... */ şeklindeki açıklama satırları.
- **Önişlemci (preprocessor):** Derleyici talimatları, örneğin #include <stdio.h>.
- **Operatör (operator):** Matematiksel ve mantıksal semboller, örneğin +, -, ==, >.
- **Noktalama (punctuation):** Yapısal karakterler, örneğin ;, {, }, (,).
- **Standart Fonksiyon (stdfunc):** Yerleşik fonksiyonlar, örneğin printf, scanf.

tokenize() fonksiyonu, her satırı tarayarak bu türlere karşılık gelen öğeleri belirler. Her token, türü, içeriği ve konum bilgisiyle birlikte kaydedilir. Örneğin int x = 10; ifadesi şu şekilde ayrıştırılır:

- int → keyword
- x → identifier

- = → operator
- 10 → number
- ; → punctuation

Bu token'lar, metin kutusu üzerinde uygun renklerle vurgulanır ve daha sonra gramer kontrolü (parsing) için kullanılır.

```

74 def tokenize(code):
75     tokens = []
76     lines = code.splitlines()
77     for line_num, line in enumerate(lines):
78         pos = 0
79         while pos < len(line):
80             if line[pos].isspace():
81                 pos += 1
82                 continue
83             matched = False
84             for token_type, pattern in TOKEN_PATTERNS:
85                 match = re.match(pattern, line[pos:])
86                 if match:
87                     value = match.group()
88                     tokens.append((token_type, value, line_num, pos))
89                     pos += len(value)
90                     matched = True
91                     break
92             if not matched:
93                 pos += 1
94     return tokens
95
96 def parse(tokens):

```

Resim2

Resim2 : tokenize() fonksiyonu, kullanıcı kodunu satır satır analiz ederek sözcük türlerine ayırır ve token listesine ekler.

4. Ayırıştırma Yöntemi (Parsing Methodology)

Ayırıştırma (parsing) süreci, leksik analizle elde edilen token'ların gramer kurallarına uygunluğunu kontrol eder. Bu işlem, tamamen manuel olarak geliştirilmiş ve Top-Down (yukarıdan aşağı) yaklaşımıyla yapılandırılmıştır. Parser, parse() fonksiyonu ve ona bağlı yardımcı fonksiyonlar (parse_if(), parse_while(), parse_block() gibi) aracılığıyla çalışır. Her bir token dizisi, beklenen gramer yapısıyla karşılaştırılır.

Ayırıştırma sürecinin işleyişi şu şekildedir:

1. Token'lar sırayla okunur ve bir indeks ile takip edilir.
2. Her token, ilgili gramer kuralıyla eşleştirilmeye çalışılır. Örneğin, bir if yapısı için şu sıra kontrol edilir: if, (, bir ifade,), ve ardından bir blok veya ifade.
3. Eğer yapı gramer kurallarına uygunsa, mark_tokens() fonksiyonu çağrılır ve ilgili token'lar valid_syntax etiketiyle işaretlenir.

4. Hatalı yapılar ise görmezden gelinir ve herhangi bir işaretleme yapılmaz.

Örneğin, `parse_if()` fonksiyonu şu adımları izler:

- `if` anahtar kelimesini kontrol eder.
- Açma parantezi (`var` mı diye bakar.
- Parantez içindeki ifadeyi (Expression) analiz eder.
- Kapatma parantezi `)` kontrol edilir.
- Son olarak, bir blok `{ ... }` veya tek bir ifade kontrol edilir.

Başarılı eşleşmeler, `valid_syntax` etiketiyle işaretlenir ve renklendirme sürecinde bu doğrulama dikkate alınır. Hatalı yapılar (örneğin, eksik parantez) işaretlenmez, ancak analiz akışı kesintiye uğramadan devam eder. Bu, kullanıcı deneyimini kesintiye uğratmamak için bilinçli bir tasarım tercihidir.

```
151 def parse_if():
152     start = index
153     if match("keyword", "if") and match("punctuation", "(") and parse_expression() and match("punctuation", ")"):
154         if parse_block():
155             mark_tokens(start, index)
156             return True
157     index = start
158     return False
```

Resim3

Resim3: `parse_if()` fonksiyonu, `if` yapılarının gramer kurallarına uygunluğunu denetleyen ayrıştırma adımlarını göstermektedir.

5. Vurgulama Sistemi (Highlighting Scheme)

Vurgulama sistemi, kullanıcı deneyimini artırmak ve kodun okunabilirliğini kolaylaştırmak amacıyla tasarlanmıştır. Renk şeması, **THEME** adında bir sözlük yapısında tanımlanmıştır ve koyu tema (dark theme) esas alınarak geliştirilmiştir. Koyu tema, hem estetik bir görünüm sağlar hem de uzun süreli kullanımlarda göz yorgunluğunu azaltır. Kullanılan renk paleti, kodun farklı bileşenlerini net bir şekilde ayırt edecek şekilde seçilmiştir:

- **Anahtar Kelime (keyword):** Mor (`#a020f0`) – Anlamlı kelimeler için dikkat çekici bir renk.
- **Tanımlayıcı (identifier):** Parlak Sarı (`#ffff66`) – Değişken ve fonksiyon adlarını belirginleştirir.
- **Sayı (number):** Canlı Yeşil (`#00cc66`) – Sayısal değerlerin kolayca fark edilmesini sağlar.
- **Metin (string):** Bej-Turuncu (`#d69d85`) – Metin sabitlerini sıcak bir tonla vurgular.
- **Yorum (comment):** Açık Yeşil (`#6a9955`) – Yorum satırlarını kodun geri kalanından ayırır.
- **Önişlemci (preprocessor):** Turuncu (`#ff8800`) – Derleyici direktiflerini öne çıkarır.

- **Operatör (operator):** Açık Gri (#d4d4d4) – Matematiksel ve mantıksal sembolleri sade bir şekilde gösterir.
- **Noktalama (punctuation):** Beyaz (ffffff) – Yapısal karakterleri net bir şekilde vurgular.
- **Standart Fonksiyon (stdfunc):** Sarımsı (ffff66) – Tanımlayıcılarla aynı renk tonu kullanılarak tutarlılık sağlanır.
- **Geçerli Sözdizimi (valid_syntax):** Parlak Yeşil (#00ff00) – Sözdizimsel olarak doğru yapıları işaretlemek için kullanılır, ancak kullanıcıya görünmez.

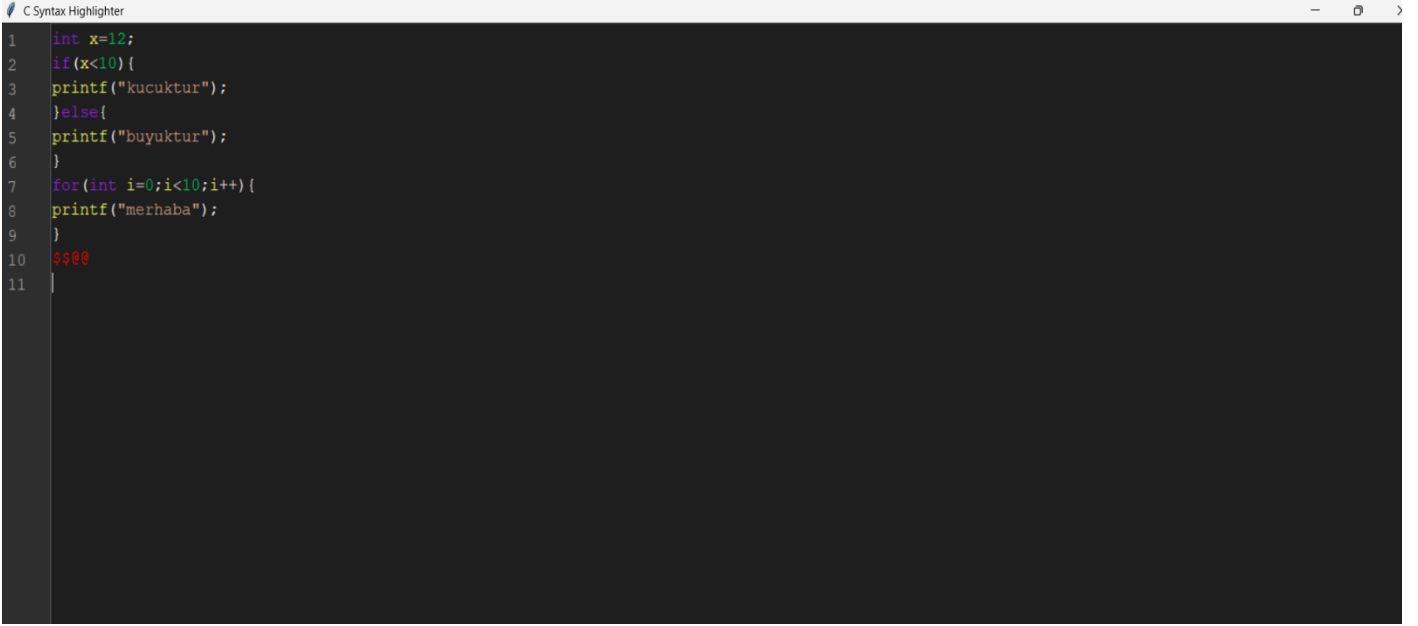
Renklendirme işlemi, `highlight_and_parse()` fonksiyonu tarafından gerçekleştirilir. Bu fonksiyon, her token'ı ilgili renk etiketiyle işaretler ve metin kutusunda görselleştirir. Örneğin, `int x = 10;` ifadesinde `int` mor, `x` parlak sarı, `=` açık gri, `10` canlı yeşil ve `;` beyaz renkte görüntülenir. Bu sistem, kodun hangi parçalarının ne anlama geldiğini kullanıcıların kolayca ayırt etmesini sağlar.



```
C Syntax Highlighter
1  int not = 80;
2
3  if (not >= 50) {
4      printf("Geçtiniz.\n");
5  } else {
6      printf("Kaldınız.\n");
7  }
8
```

Resim4

Resim4: if-else yapısına ait örnek bir C kodunun uygulama üzerinde renklendirilmiş ve doğru sözdizim yapıları işaretlenmiş hali. Renkler ve geçerli blok vurguları, hem okunabilirliği hem de yapısal ayrımı desteklemektedir.



```
1 int x=12;
2 if(x<10){
3 printf("kucuktur");
4 }else{
5 printf("buyuktur");
6 }
7 for(int i=0;i<10;i++){
8 printf("merhaba");
9 }
10 $$$
11 |
```

RESİM5

RESİM5: Bu sistem, \$, @ gibi tanımsız sembolleri algılayarak yazım hatalarını anında gösterir ve programcının doğru sözdizimi kullanmasına yardımcı olur.

6. Grafiksel Kullanıcı Arayüzü (GUI) Tasarımı

Kullanıcı arayüzü, Python'un **Tkinter** kütüphanesi kullanılarak geliştirilmiştir ve kullanıcıların yazdığı kodu anında analiz edebilmesi için tasarlanmıştır. Arayüz, sade ve işlevsel bir yapı sunarak kullanıcı deneyimini ön planda tutar. Ana bileşenler şunlardır:

- **Metin Alanı (text):** Kullanıcının kod yazdığı ana metin kutusu. Courier New fontu ve 13 punto yazı boyutu kullanılarak sabit genişlikli bir yazı tipiyle yapılandırılmıştır. Bu sayede kodun hizalaması korunur ve okunabilirlik artar. Metin kutusu, koyu tema (#1e1e1e) arka plan rengine ve beyaz (#ffffff) ön plan rengine sahiptir. İmleç (cursor) beyaz renkte (insertbackground="white") ayarlanmıştır.
- **Satır Numaraları (line_numbers):** Sol kenarda yer alan ve satır numaralarını gösteren bir metin kutusu. Bu alan, kullanıcıya kodun hangi satırında olduğunu gösterir ve büyük kod bloklarında gezinmeyi kolaylaştırır. Satır numaraları, gri bir arka plan (#2e2e2e) ve açık gri bir yazı rengi (#aaaaaa) ile görüntülenir. Bu alan yalnızca görüntüleme amaçlıdır ve kullanıcı tarafından düzenlenemez (state="disabled").
- **Kaydırma Çubuğu (scrollbar):** Dikey kaydırmayı sağlayan bir bileşen. Metin kutusu ile senkronize çalışır ve uzun kodlarda gezinmeyi kolaylaştırır.
- **Etkinlik Bağlayıcılar:** <KeyRelease>, <MouseWheel>, <Button-4>, <Button-5> gibi olaylar, highlight_and_parse() fonksiyonunu tetikler ve gerçek zamanlı analiz sağlar.

Her karakter yazıldığında veya silindiğinde sistem yeniden analiz yapar, renklendirme uygular ve geçerli yapıları belirler. update_line_numbers() fonksiyonu, satır numaralarını dinamik olarak

günceller ve metin kutusu ile aynı kaydırma pozisyonunu korur. Bu yapı, kullanıcıya bir IDE deneyimi sunar ve kod yazımını hem görsel hem de işlevsel olarak destekler

Sonuç ve Değerlendirme

Bu proje, C diline yönelik gerçek zamanlı bir sözdizimi vurgulayıcı ve kullanıcı dostu bir arayüz sunarak kod yazım sürecini hem görsel hem de yapısal olarak desteklemiştir. Manuel olarak geliştirilen analiz ve vurgulama sistemleri, öğrenme sürecini kolaylaştıran özgün ve etkili bir araç ortaya koymuştur. Geliştirmeye açık yapısıyla farklı diller ve özelliklerle genişletilebilir.

