metin, yazı tipi, ekran görüntüsü, grafik içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

PROGRAMLAMA DİLLERİ

C Syntax Highlighter

22360859056

AHMET YUMUTKAN

1. GİRİŞ

Bu projede, **C programlama dili** için gerçek zamanlı sözcüksel analiz (lexical analysis), sözdizimsel analiz (syntax analysis) ve görsel vurgulama (syntax highlighting) işlemlerini gerçekleştiren bir masaüstü uygulama geliştirilmiştir. Python’un Tkinter kütüphanesiyle geliştirilen bu uygulama, kod yazımını kolaylaştırma, hataları gerçek zamanlı gösterme ve dil yapılarını görselleştirme görevi görmektedir.

Uygulama; lexer, parser ve GUI olmak üzere üç ana bileşenden oluşmaktadır. Projede State Diagram tabanlı sözcüksel analiz ve Top-Down Parsing yöntemleri kullanılmıştır. Kullanıcılar, yazdıkları kodları anında analiz edebilir, token listesini görebilir, parse ağacını inceleyebilir ve sözdizimsel hataları canlı olarak takip edebilir.

1. DİL VE DİL BİLGİSİ SEÇİMİ
   1. HEDEF DİL: C

C dili; prosedürel yapısı, düşük seviyeli sistemlere yakınlığı ve eğitimdeki yaygın kullanımı nedeniyle tercih edilmiştir. Dilin analiz edilecek temel öğeleri şunlardır:

* Anahtar kelimeler (int, if, return, while vb.)
* Operatörler (+, \*, ==, !=, && vb.)
* Veri türleri (int, float, char, double, void)
* Denetim yapıları (if, while, for, switch-case)
* Fonksiyon tanımları ve çağrıları
  1. ANALİZ DİLİ: PYTHON

Python, hızlı prototipleme ve güçlü metin işleme yetenekleri sayesinde analiz ve GUI işlemleri için tercih edilmiştir. Ayrıca ‘Tkinter’ kütüphanesinin sunduğu esneklik sayesinde kullanışlı bir arayüz sağlanmıştır.

* 1. DİL BİLGİSİ

Dilbilgisi, bağlamdan bağımsız gramer (Context-Free Grammar - CFG) kurallarına göre modellenmiştir. Bu CFG yapısı içerisinde her yapı için parser içerisinde özel fonksiyon tanımlanmıştır. Bu yapılar;

program listprogram → statement\_list

statement → declaration | function\_definition | assignment | if\_blockif\_stmt | while\_stmt | for\_stmt | expression\_stmt | block

declaration → type IDENTIFIER [= expression] ;

function\_blockfunction\_definition → type IDENTIFIER ( [params] ) block

assignment → IDENTIFIER = expression ;

expression → simple\_expression [comparison\_op simple\_expression]

simple\_expression → term ((+|-) term)\*

term → factor ((\*|/) factor)\*

factor → NUMBER | IDENTIFIER | STRING | CHAR | function\_call | ( expression )

1. SÖZ DİZİMİ ANALİZ SÜRECİ

Proje, gerçek zamanlı (real-time) analiz mantığına sahiptir. Kullanıcı arayüzüne girilen kod her tuş vuruşunda analiz edilerek sonuçlar anında güncellenir. Sözdizimi analiz süreci şu adımları içerir:

**Girdi Alımı:** Kullanıcının yazdığı C kodu alınır.

**Lexical Analyzer:** Kod karakter karakter taranır ve Token nesnelerine dönüştürülür.

**Parser:** Token dizisi ayrıştırılarak soyut sözdizimi ağacı (AST) oluşturulur**.**  Token dizisi oluştuktan sonra, Top-Down Recursive Descent yöntemi ile sözdizimi çözümlemesi yapılır.

**Vurgulayıcı:** Token’lar renklendirilir, AST ağaç şeklinde görselleştirilir.

**Hata Analizi:** Hem lexer hem parser seviyesinde tespit edilen hatalar kullanıcıya bildirilir.

1. SÖZCÜKSEL ANALİZ SÜRECİ
   1. Seçilen Yaklaşım: State Diagram & Program Implementation

Lexical Analyzer, State Diagram & Program Implementation yaklaşımı ile geliştirilmiştir. Bu seçimin nedenleri şunlardır:

• Esneklik: Karmaşık token yapılarını kolayca işleyebilme

• Hata Yönetimi: Detaylı hata raporlama imkanı

• Performans: Doğrudan kod implementasyonu ile yüksek performans

• Genişletilebilirlik: Yeni token türlerinin kolayca eklenmesi

* 1. State Diagram Yapısı

Lexical Analyzer aşağıdaki durumları kullanmaktadır:

• START: Başlangıç durumu

• IDENTIFIER: Tanımlayıcı okuma

• NUMBER: Sayı okuma

• STRING: String literal okuma

• CHAR: Karakter literal okuma

• COMMENT\_SINGLE: Tek satır yorum

• COMMENT\_MULTI: Çok satır yorum

• OPERATOR: Operatör işleme

• PREPROCESSOR: Preprocessor direktifi

• ERROR: Hata durumu

* 1. Token Türleri ve Tanımlama

Proje 12 farklı token türünü desteklemektedir:

• KEYWORD:

Önceden tanımlı anahtar kelime listesi ile karşılaştırma

Örnek: int, if, while, return

• IDENTIFIER:

Regex: [a-zA-Z\_][a-zA-Z0-9\_]\*

• NUMBER:

Desteklenen formatlar: 123, 0, 3.14, 0.5

Ondalık nokta isteğe bağlı

• STRING:

Çift tırnak içinde: "Hello World"

Escape desteği: \n, \t, \"

• CHAR:

Tek tırnak içinde: 'A', '\n'

Escape desteği: \\', \\

• OPERATOR:

Tek karakter: + - \* / % = < >

İki karakter: == != <= >= && || ++ --

• SEPARATOR:

Parantezler: () {} []

Noktalama: ; , . : ?

• COMMENT:

Tek satır: // yorum

Çok satır: /\* yorum \*/

• PREPROCESSOR:

# ile başlayan satırlar

Örnek: #include <stdio.h>

* 1. Token Sınıfı Yapısı

Her token nesnesi aşağıdaki alanları içerir:

- type: Token türü (örneğin KEYWORD, IDENTIFIER)

- value: Token'ın metinsel değeri

- line: Satır numarası

- column: Başlangıç sütunu

- position: Kaynak metindeki pozisyon

* 1. Lexical Analysis Algoritması

Algoritma adımları şu şekildedir:

* Initialization: Input text ve pozisyon bilgilerini ayarla
* Character Reading: Karakter karakter okuma
* State Transition: Mevcut karakter ve duruma göre state geçişi
* Token Recognition: Duruma göre token oluşturma
* Position Tracking: Satır ve sütun bilgilerini güncelleme
* Error Handling: Tanınmayan karakterler için ERROR token'ı oluşturma

1. AYRIŞTIRMA METODOLOJİSİ

Bu projede sözdizimsel analiz işlemi, Top-Down Recursive Descent Parsing (Yukarıdan Aşağıya Özyinelemeli Ayrıştırma) yöntemiyle gerçekleştirilmiştir. Bu yaklaşım, her dil yapısına karşılık gelen bir fonksiyon ile token dizisini analiz eden modüler ve okunabilir bir tekniktir.

* 1. Seçilen Yöntem: Top-Down Parse

Top-Down yaklaşımı, gramerin başlangıç sembolünden başlar ve kaynak kodu yukarıdan aşağıya doğru çözümleyerek bir Abstract Syntax Tree (AST) oluşturur. Her non-terminal sembol için bir parse\_X() fonksiyonu tanımlanır.

Bu yöntem şunları sağlar:

* **Modülerlik**: Her yapı için ayrı fonksiyon vardır
* **Hata Ayıklama Kolaylığı**: Hatalı yapılar tespit edilip, detaylı mesajlarla kullanıcıya iletilir.
* **Genişletilebilirlik**: Yeni gramer kuralları kolaylıkla eklenebilir
  1. Parser'ın Genel Yapısı

Parser, Token listesini sırayla okur ve belirli gramer kurallarına göre yapıları tanımaya çalışır. Başlıca işlevi, bir dizi token'ı, dilbilgisel olarak geçerli bir program yapısına dönüştürmek ve bir ağaç (AST) halinde modellemektir. Buradan tüm C yapıları tanınır: değişkenler, fonksiyonlar, kontrol ifadeleri, ifadeler, bloklar.

* 1. Hata Yönetimi

Parser, hataları aşağıdaki şekilde işler:

* **Beklenen yapının eksikliği**: Örneğin; eksikse kullanıcıya satır numarasıyla hata verilir.
* **Parantez uyumsuzlukları**: Açılan ama kapanmayan “(, {” gibi yapılarda işaretleme yapılır.
* **Tanımsız yapı**: Geçersiz token veya sıradışı ifadeler ERROR olarak işlenir.
* **Parser Hata Tespit**: Hatalı yapı tespit edilse bile analiz devam eder ve diğer hatalar da listelenir.
  1. AST (Abstract Syntax Tree) Yapısı

AST, programın mantıksal yapısını temsil eder. Her düğüm (ParseNode) bir dil yapısını temsil eder.

Örneğin;

metin, yazılım, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

(Görsel 1)

1. VURGULAMA ŞEMASI

Bu projede, sözdizimsel analiz sonucunda elde edilen token'lar gerçek zamanlı olarak görsel olarak vurgulanmaktadır. Vurgulama işlemi, kullanıcı arayüzünde kullanılan **Tkinter.Text** bileşeni üzerinde her token türü için farklı renk ve stil ayarlarıyla gerçekleştirilir.

* 1. Vurgulama Mekanizması
* Lexical Analyzer tarafından her bir token türü belirlenir.
* Token'ların satır, sütun ve uzunluk bilgisi kullanılarak Text.tag\_add() fonksiyonu aracılığıyla ilgili stil uygulanır.
* Her token türü için önceden tanımlanmış bir tag\_configure stil etiketi kullanılır.
* Kod editörü içindeki karakterler, token türlerine göre renklendirilir.
* Hatalı yapılar ve eşleşmeyen parantezler özel arka plan renkleri ile kullanıcıya görsel olarak sunulur.
  1. Vurgulama Şeması Tablosu



1. GUI UYGULAMASI

Bu projede grafik kullanıcı arayüzü, Python’un standart GUI kütüphanesi olan Tkinter kullanılarak geliştirilmiştir. Arayüz, sözcüksel ve sözdizimsel analiz süreçlerini gerçek zamanlı olarak görselleştirmek ve kullanıcıya sezgisel bir editör deneyimi sunmak amacıyla yapılandırılmıştır.

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

(Görsel 2)

* 1. Genel Yapı ve Mimari

GUI kısmı kod editör paneli ve sekmeli sonuç panellerinden oluşmaktadır. Kodun her değişiminde analiz işlemi tetiklenerek sonuçlar eşzamanlı olarak kullanıcıya sunulur. Arayüz tepkiselliği after() fonksiyonu ile sağlanan gecikmeli analiz tekniği sayesinde korunmuştur.

* 1. Kod Editörü Paneli

Bu panel “Text” bileşeni ile inşa edilmiştir ve sözdizim renklendirme (syntax highlighting) özelliğine sahiptir. Editöre yazılan her karakter girdisi:

* Sözcüksel analiz modülüne iletilir
* Token türüne göre renklendirilir
* Satır/sütun konumuna göre işlenir
* Parantez dengesi kontrol edilir

Bu alan, sürekli güncellenen bir metin tamponu gibi davranır ve analiz sonucuna göre kendi içeriğini dinamik olarak yeniden biçimlendirir.

* 1. Sekmeler

Bu yapı, kullanıcıya farklı analiz çıktılarının sunulmasını sağlar. Uygulamada üç ana sekme bulunmaktadır:

* + 1. Tokenlar Sekmesi

Bu sekmede, sözcüksel analiz sonucu elde edilen tüm token’lar listelenir. Her token, şu bilgilerle birlikte gösterilir:

* Token türü (örneğin: KEYWORD, IDENTIFIER)
* Değeri (örneğin: int, main)
* Konumu (satır ve sütun bilgisi)

Tokenlar görsel olarak sıralanır ve hatalı token’lar (örneğin tanınmayan semboller) özel renk veya simgelerle vurgulanır. Bu sayede kullanıcı hangi yapının hangi token türüne ait olduğunu açıkça görebilir.

* + 1. Parse Tree (Ayrıştırma Ağacı) Sekmesi

Bu sekmede, sözdizimsel analiz (parsing) sonucunda oluşturulan Soyut Sözdizim Ağacı (AST) ağaç yapısı şeklinde sunulur. Her düğüm:

* Kodun bir yapısal bileşenini temsil eder (örneğin: function\_definition, if\_statement)
* Alt düğümler ise bu yapının içeriğini gösterir (örneğin parametre listesi, blok içeriği)

Bu yapı hiyerarşik olarak çizilir ve kullanıcı tarafından genişletilip daraltılabilir. Amaç, kullanıcıya programın yapısal bütünlüğünü görsel olarak anlamasını sağlamaktır.

* + 1. Hatalar Sekmesi

Bu sekme, analiz sürecinde tespit edilen tüm hataları toplar ve kullanıcıya açık biçimde sunar. Hatalar iki türdür:

* Lexical (sözcüksel) hatalar: Geçersiz semboller, kapatılmamış string/char, tanımsız karakterler.
* Syntax (sözdizimsel) hatalar: Eksik parantez, eksik noktalı virgül, hatalı ifade yapısı.

Her hata, ilgili satır ve sütun bilgisiyle birlikte kullanıcıya gösterilir. Bu sayede hata ayıklama süreci kolaylaşır.

1. SONUÇ

Bu proje kapsamında, C programlama diline yönelik çalışan, grafik arayüz tabanlı ve gerçek zamanlı sözdizim analiz yeteneğine sahip bir sistem geliştirilmiştir.

Teknik başarılar:

* Lexical Analyzer, duruma dayalı State Diagram modeliyle inşa edilerek hataya toleranslı, genişletilebilir ve modüler bir yapı sunmuştur.
* Parser, Top-Down Recursive Descent yaklaşımıyla tasarlanarak anlaşılır, modüler ve gramer kurallarına uygun bir AST üretimi gerçekleştirmiştir.
* Syntax Highlighting, 12’den fazla token türü için ayrı ayrı yapılandırılmış, renk ve font stilleriyle kullanıcıya sezgisel bir geri bildirim sağlamıştır.
* GUI, Tkinter kütüphanesiyle geliştirilen kullanıcı dostu arayüzü sayesinde analiz çıktılarının sekmeli yapıda sunulmasını, hata takibinin kolaylaşmasını ve anlık renklendirmeyle kodun daha anlaşılır hale gelmesini sağlamıştır.