Gerçek Zamanlı Renlendirme Yapan GUI

Programlama dilleri dersi proje ödevi kapsamında gerçekleştirdiğim bu çalışmada, kullanıcıların gerçek zamanlı olarak kod yazarken sözdizimi vurgulaması görebildiği bir **gramer tabanlı sözdizimi renklendirici** geliştirildi. Proje hem sözcük düzeyinde (lexical) hem de sözdizimsel (syntactic) analizleri gerçekleştirmektedir. Hazırlanan araç, 5 farklı token türünü ayırt etmekte ve kullanıcıya renklendirme ile birlikte anlık hata bilgisini metin şeklinde (Örneğin: ' let' sonrası bekleniyordu: 'variable' veya 'function', fakat 'var' bulundu.) sunmaktadır.

• Kullanılan Teknolojiler ve Yöntemler

1. Programlama Dili ve Platform

- **Dil:** JavaScript
- **Platform:** HTML + CSS + JavaScript (Vanilla)
- **Neden Seçildi?**
    - Tarayıcı destekli ve kurulumsuz çalışabilirlik
    - DOM manipülasyonu ile canlı renklendirme kolaylığı
    - contenteditable özelliğiyle kullanıcı dostu arayüz

2. Lexical Analyzer (Sözlük Analizi)

- **Yöntem:** Düzenli İfadeler (Regular Expressions)
- **Token Türleri:**
    - Anahtar kelimeler : if, else, for, while, function, return, let, const, var, class, variable
    - Operatörler : +, -, *, /, =, <, >, !
    - Literaller (sayılar, stringler)
    - Parantezler : (), [], {}
    - Tanımlayıcılar (geçerli isimler): Mutlaka bir harf ya da alt çizgi ile başlamalı, boşluk karakteri içermemeli. Keywordler ile çakışmamalı.
    - Aşağıdaki kod parçasında token çeşitleri gömülü olarak tanımlanmıştır.

```
const patterns = [
  { type: 'keyword', regex: /\b(if|else|for|while|function|return|let|const|var|class|variable)\b/g },
  { type: 'operator', regex: /[+\-*/=<>!]+/g },
  { type: 'literal', regex: /(["'`].*?["'`])|\b\d+(\.\d+)?\b/g },
  { type: 'parantez', regex: /[()[\]{}]/g },
  { type: 'tanımlayıcı', regex: /\b[a-zA-Z_]\w*\b/g },
];
```

- tokenize() fonksiyonu, yazılan kodu parçalara ayırarak her bir parçanın ne tür bir sözcük (token) olduğunu belirler.Bu işlem, **sözlük analizi (lexical analysis)** olarak adlandırılır ve sözdizimi denetiminin ilk adımıdır. Amaç; örneğin let, x, =, 5 gibi parçaların türlerini (anahtar kelime mi, operatör mü, sayı mı, değişken adı mı?) belirlemektir.

```
function tokenize(code) {
  let tokens = [];

  patterns.forEach(({ type, regex }) => {
    let match;
    while ((match = regex.exec(code)) !== null) {
      tokens.push({
        start: match.index,
        end: match.index + match[0].length,
        text: match[0],
        type
      });
    }
  });

  // Çakışan tokenları kaldır (en soldakini seç)
  tokens.sort((a, b) => a.start - b.start || b.end - a.end);
  const nonOverlapping = [];
  let lastEnd = 0;
  for (const token of tokens) {
    if (token.start >= lastEnd) {
      nonOverlapping.push(token);
      lastEnd = token.end;
    }
  }

  return nonOverlapping;
}
```

- Token'lar start ve end indeksleri ile takip edilerek çakışmalar önlendi.

3. Syntax Analyzer (Sözdizimi Analizi)

- **Yöntem:** Üstten aşağı (Top-down) analiz — *Recursive Descent Parser*
- Örneğin, değişken tanımı ve if bloklarının nasıl çözümlendiğini gösteren parça:

```
parseDeclaration() {
  this.consume('keyword', 'let');
  const nextToken = this.peek();
  if (nextToken.type === 'keyword' && nextToken.text === 'variable') {
    this.parseVariableDeclaration();
  } else if (nextToken.type === 'keyword' && nextToken.text === 'function') {
    this.parseFunctionDeclaration();
  } else {
    throw new ParseError(`'let' sonrası bekleniyordu: 'variable' veya 'function', fakat '${nextToken.text}' bulundu.`, nextToken);
  }
}
```

- Koşullu ifadelerin analizini gösteren parça:

```
parseIfStatement() {
  this.consume('keyword', 'if');
  this.consume('parantez', '(');
  this.parseExpression();
  this.consume('parantez', ')');

  this.consume('parantez', '{');
  while (this.peek() && !(this.peek().type === 'parantez' && this.peek().text === '}')) {
    this.parseStatement();
  }
  this.consume('parantez', '}');

  if (this.peek() && this.peek().type === 'keyword' && this.peek().text === 'else') {
    this.consume('keyword', 'else');
    this.consume('parantez', '{');
    while (this.peek() && !(this.peek().type === 'parantez' && this.peek().text === '}')) {
      this.parseStatement();
    }
    this.consume('parantez', '}');
  }
}
```

- **Özellikler:**
    - Her bir sözdizim kuralı için ayrı fonksiyon
    - Token tipi ve değeri kontrol edilerek ilerleme
    - ParseError sınıfı ile özel hata mesajları üretimi
- Desteklenen yapılar: Değişken tanımı, fonksiyon tanımı, if/else blokları, atamalar, return ifadeleri

4. GUI ve Renklendirme

- **highlight()** fonksiyonu, verilen kodu analiz ederek her token'ı türüne göre renklendirilmiş HTML çıktısına dönüştürür.

```
function highlight(code) {
  const tokens = tokenize(code);
  let result = '';
  let lastIndex = 0;

  for (const token of tokens) {
    // Önce aradaki boşlukları, kaçırılmış karakterleri ekle
    if (token.start > lastIndex) {
      result += escapeHTML(code.slice(lastIndex, token.start));
    }

    // Token'ı renklendir
    result += `<span class="${token.type}">${escapeHTML(token.text)}</span>`;
    lastIndex = token.end;
  }

  // Son kalan kısmı ekle (tokenlenmemiş son karakterler)
  result += escapeHTML(code.slice(lastIndex));

  return result;
}
```

- Her token HTML **** etiketi ile sarılır, Aralarda kalan renklendirilmemiş metin korunur. Sonuç olarak, innerHTML olarak kullanılabilecek renklendirilmiş bir içerik döndürülür. Bu fonksiyon, editörde yazılan kodun canlı olarak renklendirilmesini sağlar.

- imleç konumu korunarak kullanıcı deneyimi bozulmadan içerik güncellenir. İmleç konumunu koruyan fonksiyon aşağıda verilmiştir.

```
function saveCaretPosition(context) {
  const selection = window.getSelection();
  let charCount = -1, node;

  if (selection.rangeCount > 0) {
    const range = selection.getRangeAt(0);
    const preRange = range.cloneRange();
    preRange.selectNodeContents(context);
    preRange.setEnd(range.endContainer, range.endOffset);
    charCount = preRange.toString().length;
  }
  return charCount;
}
```

- Hatalı token, error sınıfı ile kırmızı arka planla vurgulanır. kodda bir sözdizimi hatası tespit edildiğinde, o hatalı token'ı özel bir CSS sınıfı **(.error)** ile vurgulayarak kullanıcıya görsel olarak gösterir. İşlemi yapan fonksiyon aşağıda verilmiştir.

```
function highlightWithError(code, tokens, errStart, errEnd) {
  let result = '';
  let lastIndex = 0;

  tokens.forEach(token => {
    if (token.start >= lastIndex) {
      if (token.start === errStart && token.end === errEnd) {
        result += escapeHTML(code.slice(lastIndex, token.start));
        result += `<span class="error">${escapeHTML(token.text)}</span>`;
        lastIndex = token.end;
      } else {
        result += escapeHTML(code.slice(lastIndex, token.start));
        result += `<span class="${token.type}">${escapeHTML(token.text)}</span>`;
        lastIndex = token.end;
      }
    }
  });

  result += escapeHTML(code.slice(lastIndex));

  const caretPos = saveCaretPosition(editor);

  editor.innerHTML = result;

  restoreCaretPosition(editor, caretPos);
}
```

Karşılaşılan Zorluklar

1. İmleç Konumunun Korunması

Kullanıcı yazdıkça DOM güncellendiği için imleç pozisyonu sürekli sıfırlanıyordu. Bunu çözmek için imleç karakter sayısıyla takip edildi ve Range API'si ile yeniden yerleştirildi.

2. Çakışan Token'lar

Örneğin bir sayı (123) hem literal hem tanımlayıcı olarak algılanabiliyordu. Bu nedenle tokenlar başlangıç-bitiş konumlarına göre sıralandı ve yalnızca çakışmayanlar tutuldu.

3. Hatalı Kodla Başa Çıkma

Eksik parantez, tanımsız değişken ya da yanlış anahtar kelime gibi hatalarda programın çökmesini önlemek için özel hata sınıfı yazıldı. Bu sınıf, kullanıcıya görsel olarak yanlış kısmı gösterebildi.

4. Tab ve Enter Tuşu Desteği

contenteditable alanında Tab ve Enter tuşları beklenildiği gibi çalışmıyordu. Bunun için klavye olayları özel olarak dinlenerek manuel içerik yerleştirildi (örneğin Tab yerine iki boşluk, Enter yerine
).

5. Gerçek Zamanlı Performans

Her tuş vuruşunda tokenizasyon, parsing ve renklendirme işlemleri yapıldığından performans kaygısı oluştu. Bu nedenle minimal yeniden işleme uygulandı: sadece değişen içerikler analiz edildi.

Arayüz Tasarımı

- **Kod Editörü (contenteditable div):** Kullanıcı girişini alır.
- **Renklendirme Alanı:** HTML olarak renklendirilmiş kodu gösterir.
- **Hata Paneli:** Anlık olarak hatalı ifadeyi ve açıklamasını gösterir.
- Arayüz örnek ekran görüntüleri aşağıda verilmiştir.

```
let variable x=5;
let variable y=17;
function topla()
{

}
```

```
Bilinmeyen anahtar kelime: function
```

```
let variable x=5;
let variable y=17;
let variable z=x+y;
let function topla()
{
```

```
Bilinmeyen hata: Bekleniyordu: parantez }, fakat dosya sonu geldi.
```

```
let variable x=5;
let variable y=17;
let function topla()
{
   |
}
```

```
Sözdizimi hatası yok.
```

- Proje Kodunun BNF Tanımı

<program> ::= <statement>*

<statement> ::= <declaration>

| <if-statement>

```
                | <assignment>
                | <return-statement>
<declaration>     ::= "let" <declaration-type>
<declaration-type> ::= "variable" <identifier> [ "=" <expression> ] [ ";" ]
                | "function" <identifier> "(" ")" "{" <statement>* "}"
<if-statement>    ::= "if" "(" <expression> ")" "{" <statement>* "}" [ "else" "{" <statement>* "}" ]
<assignment>      ::= <identifier> "=" <expression> [ ";" ]
<return-statement> ::= "return" <expression> ";"
<expression>      ::= <term> { ("+" | "-") <term> }*
<term>            ::= <factor> { ("*" | "/") <factor> }*
<factor>          ::= <literal>
                | <identifier>
                | "(" <expression> ")"
<literal>         ::= <number> | <string>
<identifier>      ::= <letter> { <letter> | <digit> }*
<number>          ::= <digit>+ [ "." <digit>+ ]
<string>          ::= "\"" <char>* "\"" | "'" <char>* "'"
<letter>          ::= "a" | "b" | ... | "z" | "A" | ... | "Z" | "_"
<digit>           ::= "0" | "1" | ... | "9"
<char>            ::= herhangi geçerli karakter (tırnak işareti hariç)
```

- ## Sonuç ve Kazanımlar

Bu proje sayesinde bir dilin sözdizimsel yapısının nasıl çözümlendiğini ve analiz edildiğini uygulamalı olarak öğrendik. Gerçek zamanlılık, kullanıcı deneyimi ve hata toleransı gibi yazılım mühendisliğinde önemli olan konularla pratik yaptık. Ayrıca, metin düzenleme, DOM yönetimi ve kullanıcı odaklı tasarım ilkeleri konusunda yetkinlik kazandık.

- ## Demo Videosu ve Kaynak Kodları
  - **Kaynak Kodu:** https://github.com/semaimre/realtime-GUI/blob/main/index.html
  - https://github.com/semaimre/realtime-GUI/blob/main/script.js
  - https://github.com/semaimre/realtime-GUI/blob/main/style.css

- **Demo Videosu:** https://youtu.be/kBZkmNas5B4?si=F7fYh9ewPWsepgAe