Лексер обробляє JavaScript код, що може містити:

* **Класичні конструкції:** змінні, функції, цикли
* **ES6+ конструкції:** стрілочні функції, template literals, деструктуризація
* **Сучасні оператори:** optional chaining, nullish coalescing, spread operator
* **Модульна система:** import/export statements
* **Асинхронний код:** async/await конструкції

**Реалізовані нетривіальні конструкції:**

|  |  |  |
| --- | --- | --- |
| **Токен** | **Синтаксис** | **Реалізація через FA** |
| **ARROW\_FUNCTION** | => | Двостадійний автомат: = → > |
| **TEMPLATE\_LITERAL** | `Hello ${name}` | Складний FA з обробкою ${} інтерполяції |
| **SPREAD\_OPERATOR** | ...args | Тристадійний автомат: . → . → . |
| **OPTIONAL\_CHAINING** | obj?.prop | Двостадійний автомат: ? → . |
| **NULLISH\_COALESCING** | a ?? b | Двостадійний автомат: ? → ? |
| **ASYNC\_AWAIT** | async/await | Розпізнавання через keywords |
| **CLASS** | class MyClass | Розпізнавання через keywords |
| **IMPORT\_EXPORT** | import/export | Розпізнавання через keywords |
| **DESTRUCTURING** | {a, b} = obj | Постобробка через аналіз послідовності |

**Методи скінченних автоматів (FA)**

**Template Literals FA:**

State 0 --'`'--> State 1 (in\_template)

State 1 --char--> State 1

State 1 --'$'--> State 2

State 2 --'{'--> State 3 (in\_expression)

State 3 --char--> State 3

State 3 --'}'--> State 1

State 1 --'`'--> State 4 (ACCEPT)

**Spread Operator FA:**

State 0 --'.'--> State 1

State 1 --'.'--> State 2

State 2 --'.'--> State 3 (ACCEPT)

State 1/2 --[not '.']--> REJECT

**Методи:**

* **currentChar()** - отримання поточного символу
* **peekChar(offset)** - перегляд символу з зміщенням (FA transitions)
* **advance()** - просування позиції з оновленням рядка/стовпця
* **readString(char quote)** - FA для рядкових літералів
* **readTemplateLiteral()** - складний FA для template strings
* **readNumber()** - FA для числових літералів
* **readIdentifier()** - FA для ідентифікаторів та ключових слів
* **readComment()** - FA для одно/багаторядкових коментарів
* **tokenize()** - головний метод токенізації
* **hasDestructuring()** - виявлення деструктуризації
* **readFileContent()** - статичний метод читання файлів

**Основний цикл токенізації:**

1. **Ініціалізація:** встановлення позиції, рядка, стовпця
2. **Головний цикл:**

while (position < input.length()) {

char ch = currentChar();

if (isWhitespace(ch)) → skipWhitespace()

else if (ch == '/') → readComment()

else if (ch == '`') → readTemplateLiteral()

else if (ch == '"' || ch == '\'') → readString()

else if (isDigit(ch)) → readNumber()

else if (isAlpha(ch)) → readIdentifier()

else → processOperators() // FA для операторів

}

1. **Постобробка:** додавання END\_OF\_FILE токена

**Обробка нетривіальних операторів:**

// Arrow function FA

if (ch == '=' && peekChar() == '>') {

token = ARROW\_FUNCTION("=>")

advance(2)

}

// Spread operator FA

if (ch == '.' && peekChar() == '.' && peekChar(2) == '.') {

token = SPREAD\_OPERATOR("...")

advance(3)

}