

Лексический анализ

Лекция №2

Книга “Syntactic Structures”, Avram Noam Chomsky

- Вышла в 1957 году
- Фундамент всех формальных языков

Книга “Syntactic Structures”, Avram Noam Chomsky

- Вышла в 1957 году
- Фундамент всех формальных языков

Что есть в книге:

1. **Порождающие грамматики** (Generative Grammars)
2. Гипотеза о врождённых способностях к усвоению языка через порождающие грамматики

Виды разбора в русском языке

Лексический разбор слов

Косил косой косой косой

Лексический разбор слов

Косил косой косой косой

Лексический разбор:

1. Определяем смысл каждого слова (по словарю)
2. Для многозначных слов определяем значение

Лексический разбор слов

Косил косой косой косой

Лексический разбор:

1. Определяем смысл каждого слова (по словарю)
2. Для многозначных слов определяем значение
 - Лемматизация — замена всех слов на леммы
 - Лемма — словарная форма слова

Лексический разбор слов

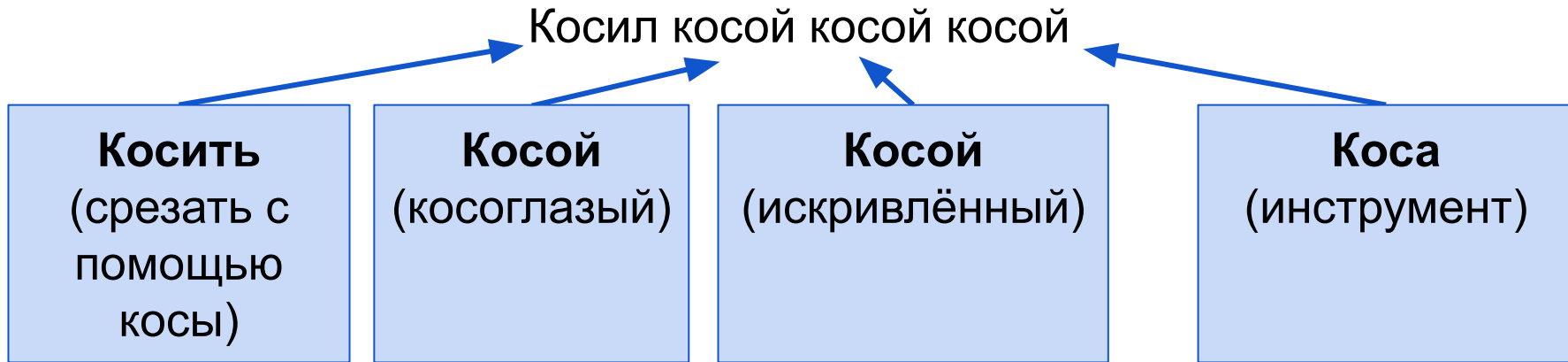
Косил косой косой косой

Косить
(срезать с
помощью
косы)

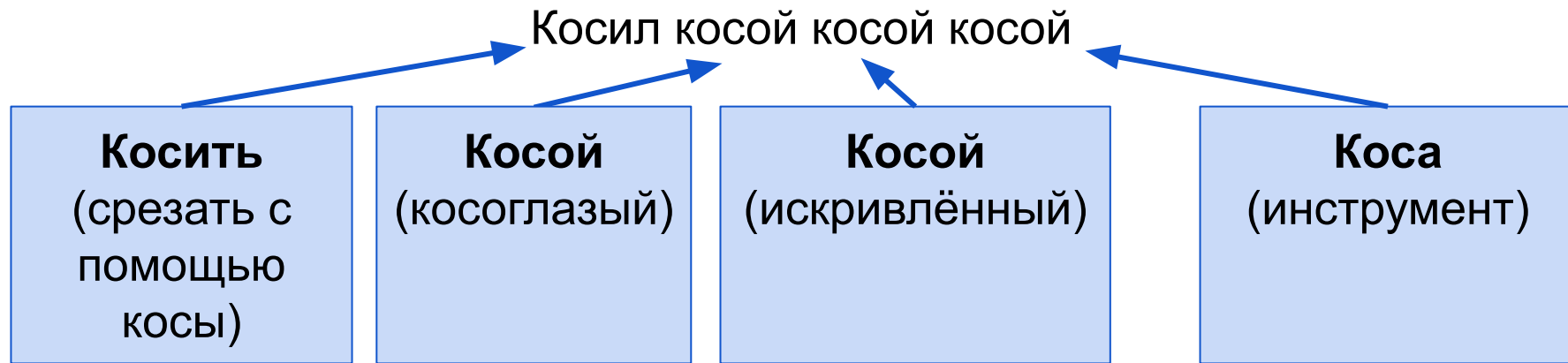
Косой
(косоглазый)

Косой
(искривлённый)

Коса
(инструмент)



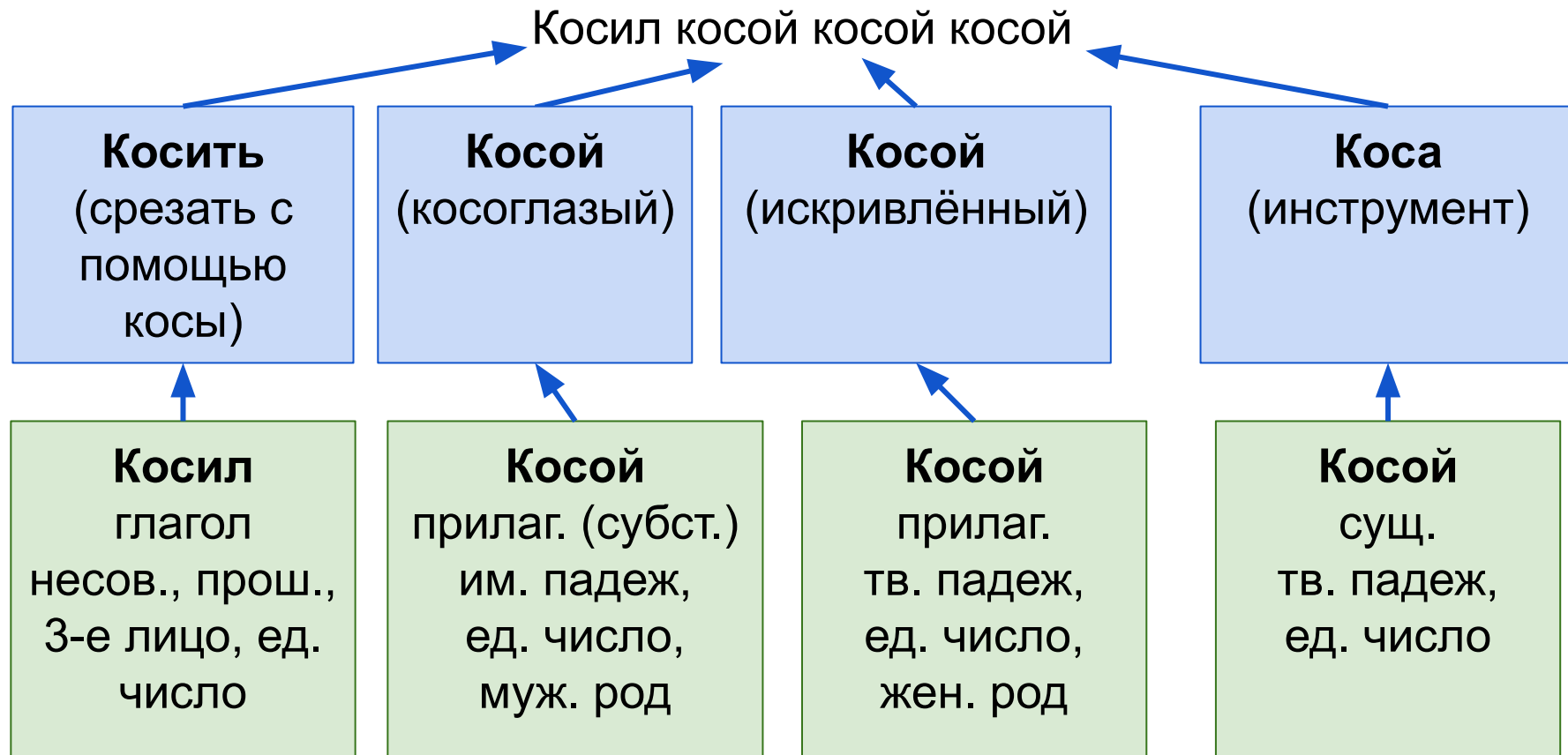
Морфологический разбор слов



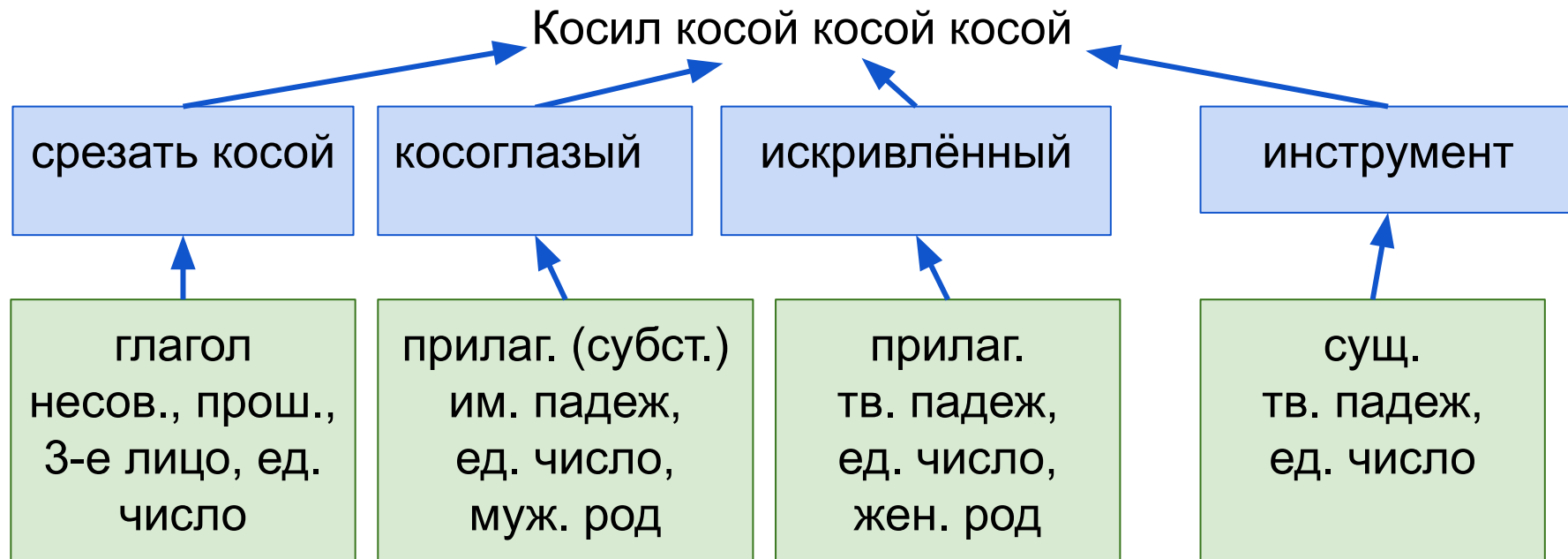
Морфологический разбор:

1. Определяем часть речи слова
2. Определяем признаки
 - Глаголы имеют вид, время, лицо, число
 - Прилагательные имеют род, падеж, число

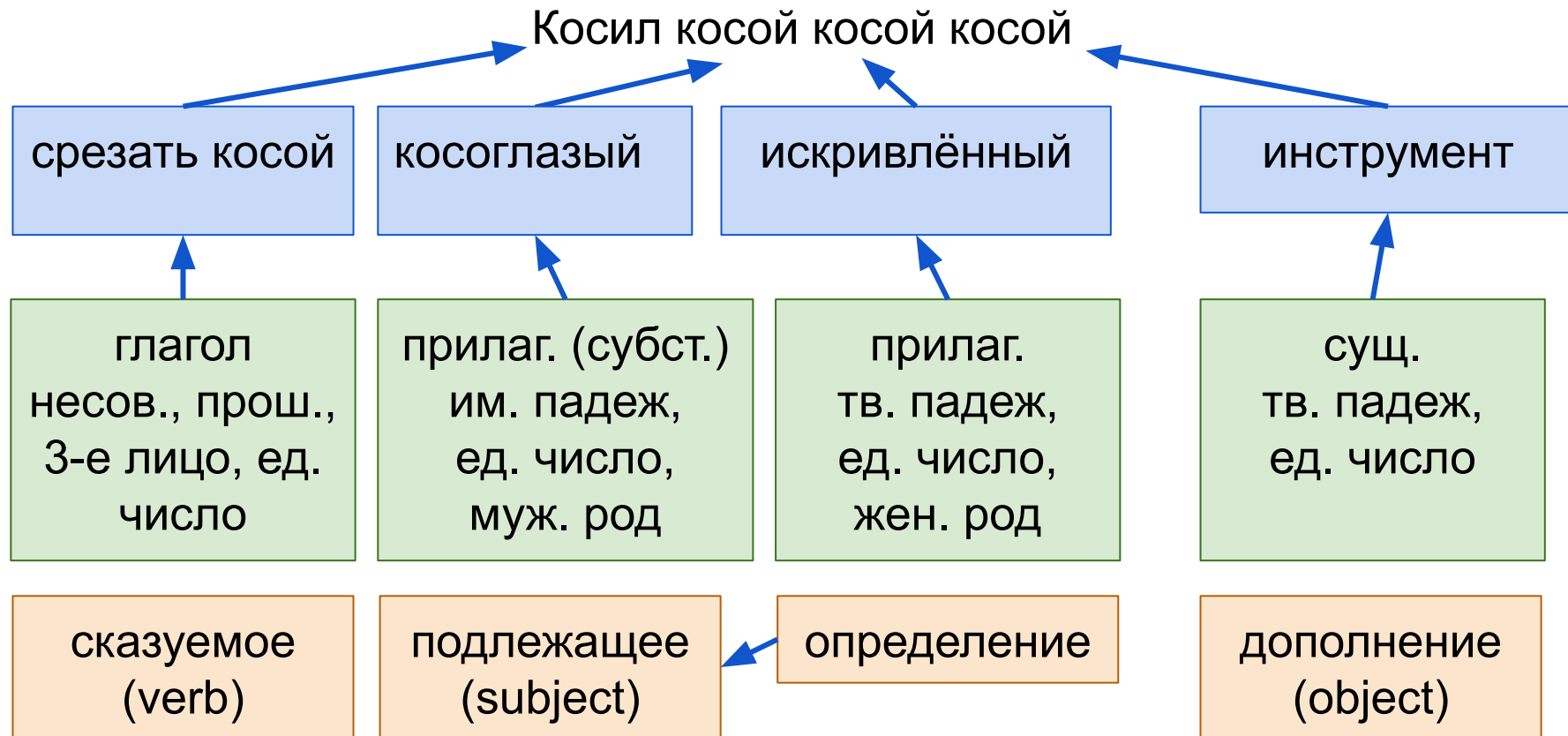
Морфологический разбор слов



Синтаксический разбор предложения



Синтаксический разбор предложения



Виды анализа для определения синтаксиса

Естественные языки

1. Лексический анализ
2. Морфологический анализ
3. Синтаксический анализ

Формальные языки

???

Виды анализа для определения синтаксиса

Естественные языки

1. Лексический анализ
2. Морфологический анализ
3. Синтаксический анализ

Формальные языки

1. Лексический анализ
2. Синтаксический анализ

Морфологический анализ не нужен, потому что лексемы однозначны.

Лексическая структура SQL

Примеры лексем в SQL

```
SELECT first_name, last_name FROM student;
```

Лексемы:

1. Ключевые слова — SELECT, FROM
2. Идентификаторы — first_name, last_name
3. Разделители — “;”, “,”

Примеры лексем в SQL

```
SELECT 2 * 3.14159 * radius FROM circle;
```

Лексемы:

???

Примеры лексем в SQL

```
SELECT 2 * 3.14159 * radius FROM circle;
```

Лексемы:

1. Ключевые слова — SELECT, FROM
2. Идентификатор — radius
3. Литералы чисел — 2, 3.14159
4. Разделитель — “;”
5. Оператор — “*”

Примеры лексем в SQL

-- Outputs squares for all circles

```
SELECT 2 * 3.14159 * radius FROM circle;
```

Лексемы: ???

Примеры лексем в SQL

-- Outputs squares for all circles

```
SELECT 2 * 3.14159 * radius FROM circle;
```

Комментарий — не лексема, потому что не важен для синтаксиса

Примеры лексем в SQL

-- Outputs squares for all circles

```
SELECT 2 * 3.14159 * radius FROM circle;
```

Комментарий — не лексема, потому что не важен для синтаксиса

Лексемы:

1. Ключевые слова — SELECT, FROM
2. Идентификатор — radius
3. Литералы чисел — 2, 3.14159
4. Разделитель — “;”
5. Оператор — “*”

BNF и EBNF

Идентификатор

Идентификатор в языке SQL:

1. Начинается с буквы или “_”
2. Далее могут идти буквы, цифры или “_”

BNF — форма Бэкуса-Найра (Backus-Naur)

identifier ::= head | head tail

head ::= letter | "_"

tail ::= non_head | non_head tail

non_head ::= letter | digit | "_"

BNF — форма Бэкуса-Наура (Backus-Naur)

`identifier ::= head | head tail`

`head ::= letter | "_"`

`tail ::= non_head | non_head tail`

`non_head ::= letter | digit | "_"`

В BNF правило содержит:

- альтернативы, соединённые через “|”
- символы в двойных кавычках (*терминалы*)
- список других правил (*нетерминалы*)

EBNF — расширенная форма Бэкуса-Наура

```
identifier = head, { non_head } ;
```

```
head = letter | "_" ;
```

```
non_head = letter | digit | "_";
```

EBNF — расширенная форма Бэкуса-Наура

```
identifier = head, { non_head } ;
```

```
head = letter | "_" ;
```

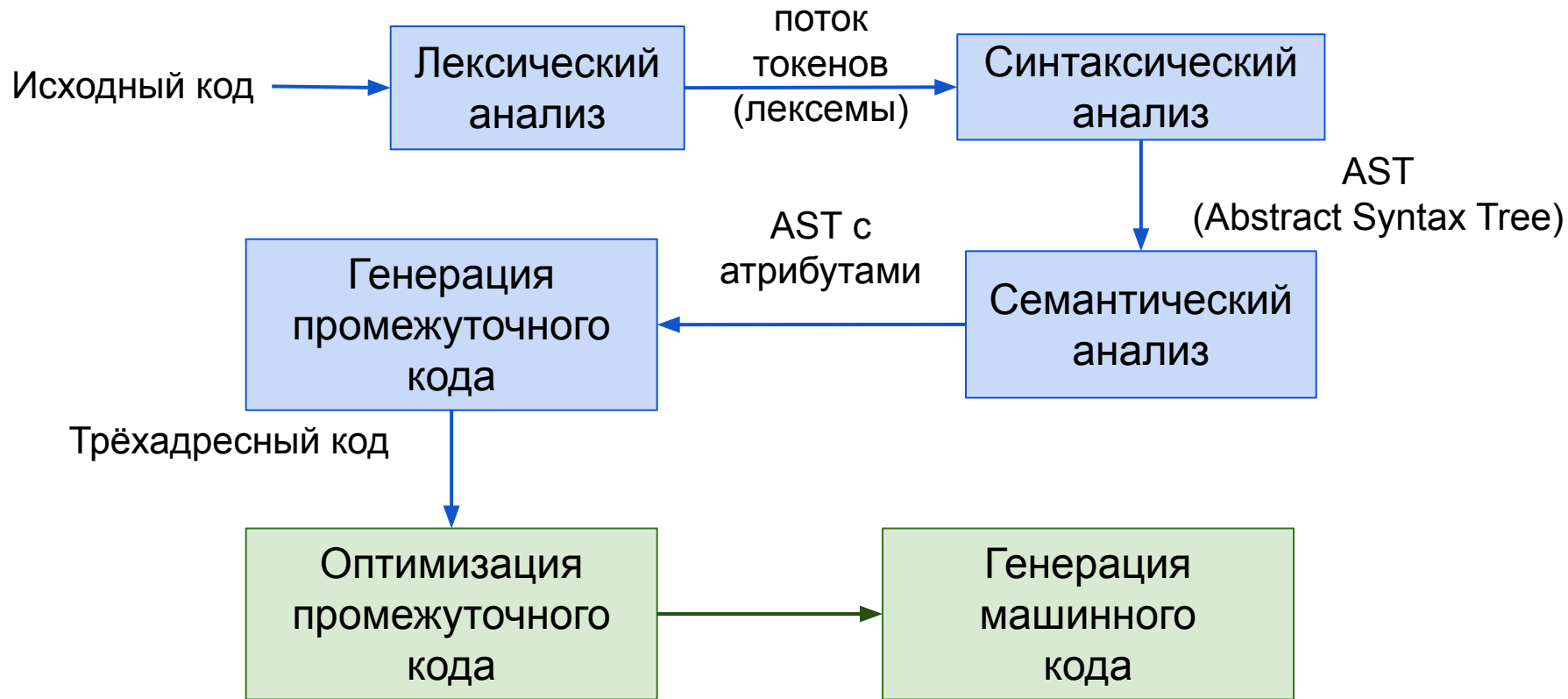
```
non_head = letter | digit | "_";
```

Что добавляет EBNF:

- { symbol_1 symbol_2 } для повтора 0, 1 или много раз
- [symbol_1 symbol_2] для опциональных частей
- (symbol_1 symbol_2) для группировки

Лексический анализатор в разработке компилятора

Архитектура компилятора



Фронтенд компилятора — слои и функции

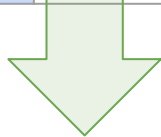
	Выражения	Блоки кода	Ветвления	Циклы	Функции
Лексика	✗	✗	✗	✗	✗
Синтаксис	✗	✗	✗	✗	✗
Семантика	✗	✗	✗	✗	✗
Генерация IR	✗	✗	✗	✗	✗

Фронтенд компилятора — разработка по слоям

	Выражения	Блоки кода	Ветвления	Циклы	Функции
Лексика	✓	✓	✓	✓	✓
Синтаксис	✗	✗	✗	✗	✗
Семантика	✗	✗	✗	✗	✗
Генерация IR	✗	✗	✗	✗	✗

Фронтенд компилятора — разработка по функциям

	Выражения	Блоки кода	Ветвления	Циклы	Функции
Лексика	✓	✗	✗	✗	✗
Синтаксис	✓	✗	✗	✗	✗
Семантика	✓	✗	✗	✗	✗
Генерация IR	✓	✗	✗	✗	✗



Фронтенд компилятора — разработка по функциям

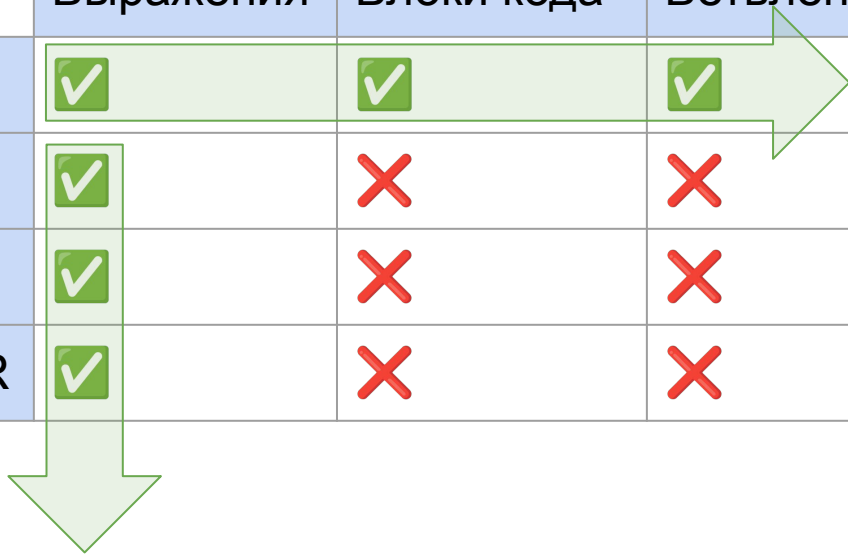
	Выражения	Блоки кода	Ветвления	Циклы	Функции
Лексика	✓	✗	✗	✗	✗
Синтаксис	✓	✗	✗	✗	✗
Семантика	✓	✗	✗	✗	✗
Генерация IR	✓	✗	✗	✗	✗

«Представьте, что история – это тонкий вертикальный срез, проходящий сквозь горизонтальные слои проекта.»

— Роберт Мартин, «Чистый Agile»

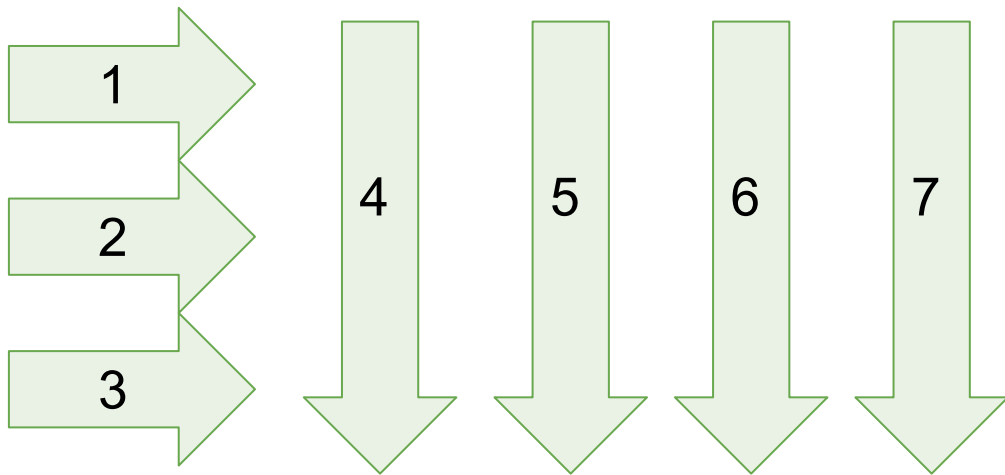
Фронтенд компилятора — смешанный подход

	Выражения	Блоки кода	Ветвления	Циклы	Функции
Лексика	✓	✓	✓	✗	✗
Синтаксис	✓	✗	✗	✗	✗
Семантика	✓	✗	✗	✗	✗
Генерация IR	✓	✗	✗	✗	✗



Когда предметная область незнакома

1. Лучше разрабатывать по функциям, а не слоям
2. ... но в начале проекта строим каркас по слоям



Каноничный TDD

Что такое модульный тест?

- Модульный тест — проверяет модуль через внутренние интерфейсы

Что такое модуль?

- Модульный тест — проверяет модуль через внутренние интерфейсы
- Модуль — класс, функция или файл
 - имеет интерфейс и реализацию

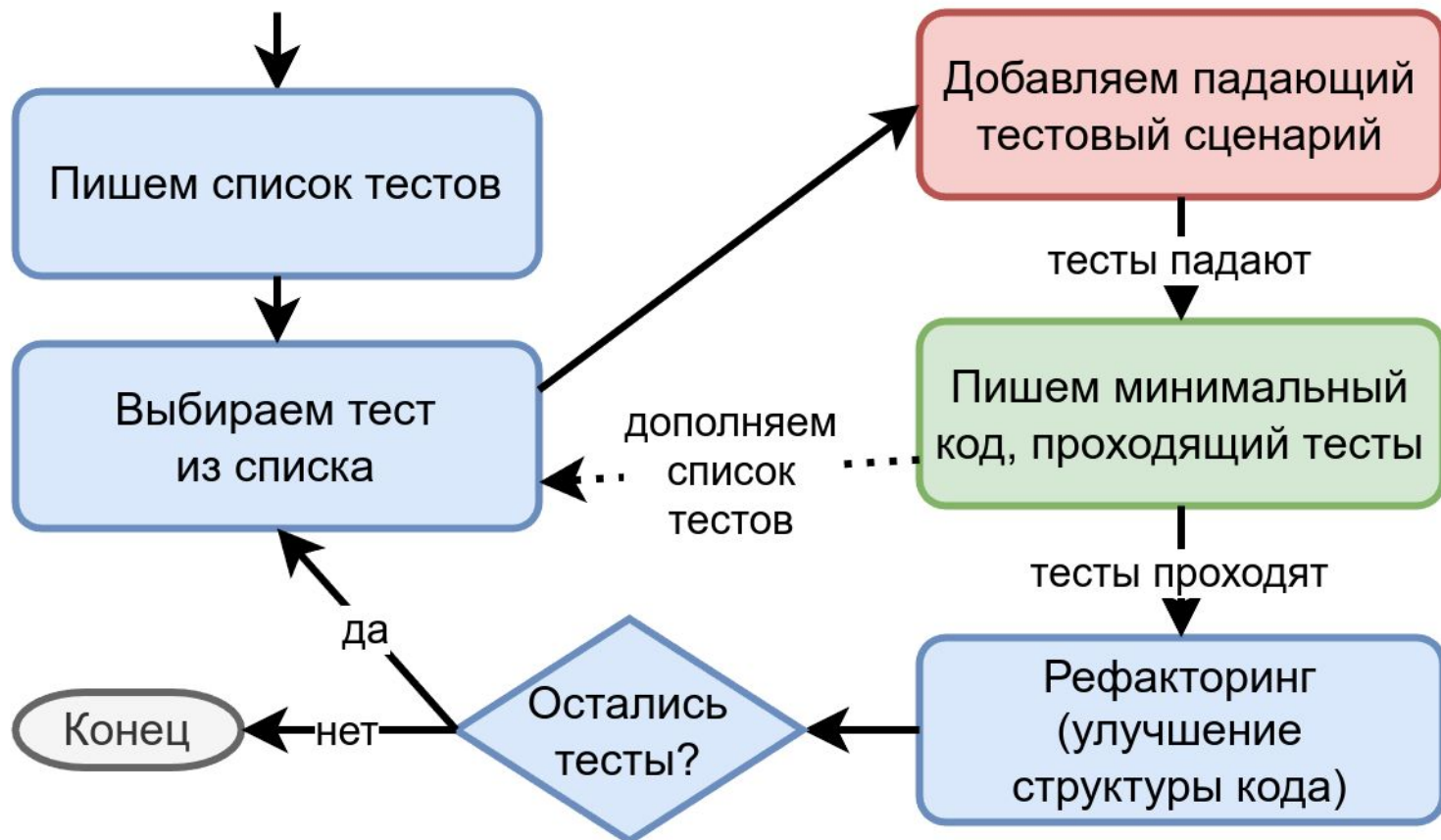
Что такое TDD?

- Модульный тест — проверяет модуль через внутренние интерфейсы
- Модуль — класс, функция или файл
 - имеет интерфейс и реализацию
- TDD (Test Driven Development) — тесты ведут кодирование
 - тесты определяют дизайн кода
 - тесты задают спецификацию поведения
 - тесты создают качество

Что такое TDD?

- Модульный тест — проверяет модуль через внутренние интерфейсы
- Модуль — класс, функция или файл
 - имеет интерфейс и реализацию
- TDD (Test Driven Development) — тесты ведут кодирование
 - тесты определяют дизайн кода
 - тесты задают спецификацию поведения
 - тесты создают качество
- Можно иметь модульные тесты и не иметь TDD

Каноничный TDD

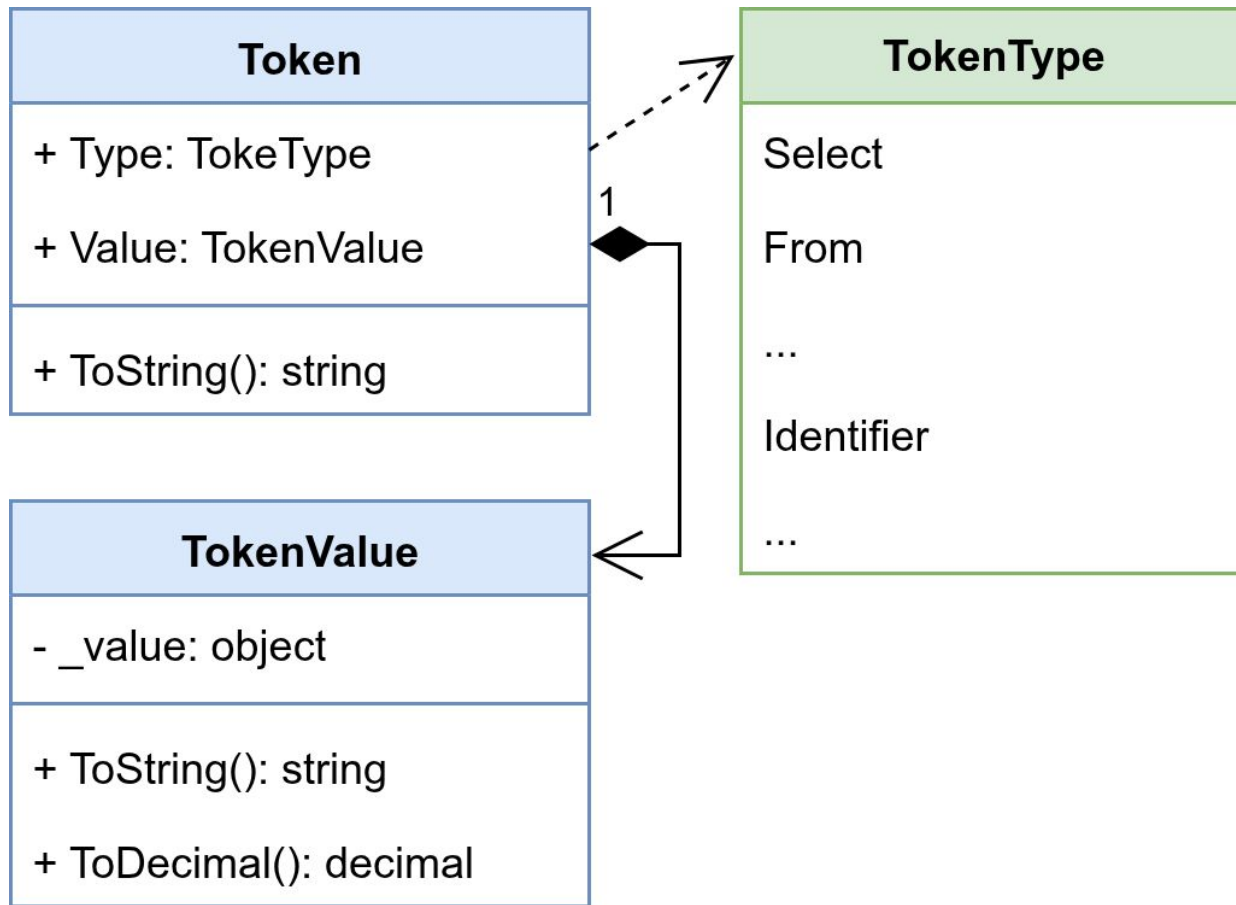


Проектирование Lexer

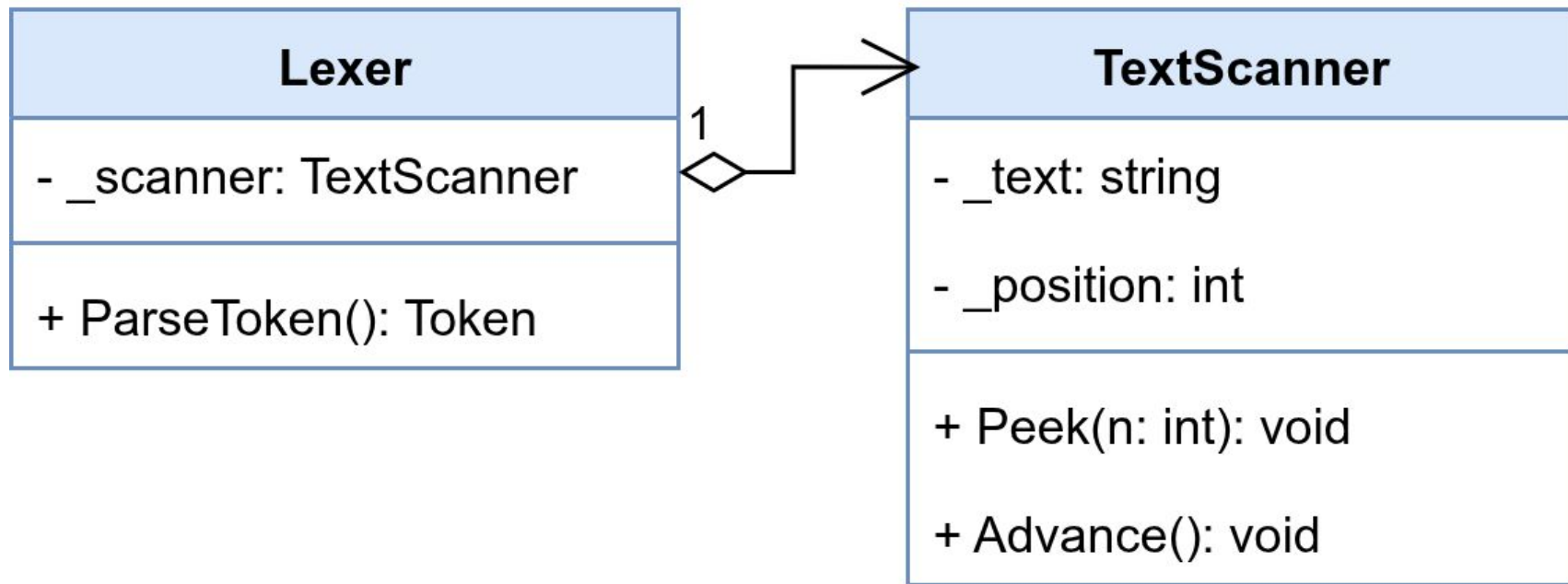
Ключевые решения

1. Целиком читаем исходный код в память
2. Разделяем на лексемы вручную
3. Выделяем абстракции

Token — представляет данные лексемы



TextScanner — абстракция над строкой



Элементы реализации

Совет: пишите правило в комментариях

```
/// <summary>
```

```
///  Распознаёт идентификатор по правилам:
```

```
///  identifier = [letter | '_' ] { letter | digit | '_' }
```

```
///  letter = "a" | "b" | ... | "z" | unicode_letter
```

```
///  digit = "0" | "1" | ... | "9"
```

```
///  unicode_letter – любая буква Unicode.
```

```
/// </summary>
```

```
private Token ParseIdentifierOrKeyword()
```

Ключевое слово — это зарезервированное имя

```
if (Keywords.TryGetValue(  
    value.ToUpper(CultureInfo.InvariantCulture),  
    out TokenType type)  
) {  
    return new Token(type); // Ключевое слово  
}  
  
return new Token(  
    TokenType.Identifier, new TokenValue(value)  
);
```

Разбор целых чисел

```
decimal value = GetDigitValue(_scanner.Peek());  
_scanner.Advance();  
for (char c = _scanner.Peek();  
     char.IsAsciiDigit(c);  
     c = _scanner.Peek())  
{  
    value = value * 10 + GetDigitValue(c);  
    _scanner.Advance();  
}
```

Пропуск комментариев

```
private void SkipWhiteSpacesAndComments() {  
    do {  
        SkipWhiteSpaces();  
    } while (  
        TryParseMultilineComment()  
        || TryParseSingleLineComment()  
    );  
}
```

Примеры

1. C#: sourcecraft.dev/sshambir-public/pstiger
 - ветка 02_lexical_analysis
2. C++: sourcecraft.dev/sshambir-public/pstiger/cpp
 - ветка 01_lexemes
3. Go: sourcecraft.dev/sshambir-public/minijava/go
 - ветка 02_lexer

Плагин для IDE

Термины

- IDE — Integrated Development Environment
 - Написание, сборка, отладка кода
- Text Editor
 - Написание кода

Термины

- IDE — Integrated Development Environment
 - Написание, сборка, отладка кода
- Text Editor
 - Написание кода

Вопрос: это IDE или текстовый редактор?

1. Visual Studio

Термины

- IDE — Integrated Development Environment
 - Написание, сборка, отладка кода
- Text Editor
 - Написание кода

Вопрос: это IDE или текстовый редактор?

1. Visual Studio
2. Notepad++

Термины

- IDE — Integrated Development Environment
 - Написание, сборка, отладка кода
- Text Editor
 - Написание кода

Вопрос: это IDE или текстовый редактор?

1. Visual Studio
2. Notepad++
3. VSCode

Как научить IDE понимать язык?

Как научить IDE понимать язык?

1. Вместо IDE — плагин VSCode

Как научить IDE понимать язык?

1. Вместо IDE — плагин VSCode
2. Сначала базовая поддержка
 - `*.tmLanguage.json` — лексика в формате TextMate
 - `language-configuration.json`

Как научить IDE понимать язык?

1. Вместо IDE — плагин VSCode
2. Сначала базовая поддержка
 - `*.tmLanguage.json` — лексика в формате TextMate
 - `language-configuration.json`
3. Затем расширенная поддержка
 - свой language server
 - интеграция в плагине VSCode

Лексический анализ

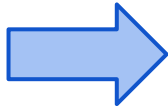
1. Основан на регулярной грамматике
 - Регулярная грамматика $\approx$ ДКА/НКА
 - Регулярная грамматика $\approx$ регулярное выражение
2. Разделяет текст на лексемы

Лексический анализ Pascal

```
PROCEDURE WriteIntToFile(  
    Result: INTEGER  
);  
VAR  
    F: TEXT;  
BEGIN  
    ASSIGN(F, 'output.txt');  
    REWRITE(F);  
    WRITELN(F, Result);  
    CLOSE(F);  
END;
```

Лексический анализ Pascal

```
PROCEDURE WriteIntToFile(  
    Result: INTEGER  
);  
VAR  
    F: TEXT;  
BEGIN  
    ASSIGN(F, 'output.txt');  
    REWRITE(F);  
    WRITELN(F, Result);  
    CLOSE(F);  
END;
```



```
PROCEDURE WriteResultToFile(  
    Result: INTEGER  
);  
VAR  
    F: TEXT;  
BEGIN  
    ASSIGN(F, 'output.txt');  
    REWRITE(F);  
    WRITELN(F, Result);  
    CLOSE(F);  
END;
```

Возможности IDE

Лексического анализа хватит для фич:

1. Подсветка лексем — ключевые слова, литералы
2. Простое автодополнение — ключевые слова
3. Закомментировать / раскомментировать код

Возможности IDE

Лексического анализа хватит для фич:

1. Подсветка лексем — ключевые слова, литералы
2. Простое автодополнение — ключевые слова
3. Закомментировать / раскомментировать код
4. Простые отступы
 - добавить отступ после BEGIN
 - убрать отступ после END
5. Подсветка пар скобок
6. Сворачивание областей — fold / unfold



немного больше чем
лексический анализ

Расширяемые редакторы

1. Декларативный формат TextMate
2. Декларативный формат Kate (KDE)

Расширяемые редакторы

1. Декларативный формат TextMate
2. Декларативный формат Kate (KDE)
3. Декларативный формат VSCode
 - `ISO-Pascal.tmLanguage.json` — грамматика TextMate
 - `language-configuration.json` — дополнительные параметры языка

Вайбкодим расширение VSCode

Промпт:

Напиши для VSCode плагин с описанием грамматики в формате TextMate для языка, лексика которого описана в приложенном файле.



1. Сохранить в каталог
2. Выполнить действие: “Developer: Install Extension from Location...”
3. Выполнить действие: “Developer: Reload Window”

Вайбкодим расширение VSCode

Подсветка для Tiger:

```
/* Программа вычисляет факториал числа. */  
let  
    function factorial(x: int): int =  
        if x <= 1 then 1 else x * factorial(x - 1)  
    in  
        printi(factorial(5))  
end
```

Конец
Вопросы?