Как устроены IDE

Лекция №10
Теория языков программирования

Вопросы этой лекции

1. Что такое IDE?
2. Что IDE знают о языках?
3. Как возможности IDE соотносятся с фронтендом компилятора?
4. Как поддержать разные языки в одной IDE?
5. Как поддержать одних язык в разных 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

Пример задачи

Пишем компилятор и IDE для Pascal.

Как научить IDE понимать язык?

Лексический анализ

Лексический анализ

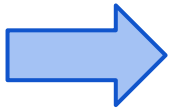
1. Основан на регулярной грамматике
    - Регулярная грамматика $\asymp$ ДКА/НКА
    - Регулярная грамматика $\asymp$ регулярное выражение
2. Разделяет текст на лексемы

Лексический анализ

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

Лексический анализ

```
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
```

Синтаксический анализ

Синтаксический анализ

1. Основан на контекстно-свободной грамматике
   - КС-грамматика $\asymp$ конечный автомат со стеком
   - Выражается в EBNF и других нотациях
2. Разбирает текст в дерево
   - Parse tree
   - Abstract Syntax Tree

Возможности IDE

Синтаксический анализ добавит фичи:

1. Отображение синтаксических ошибок
2. Outline — краткая структура файла

```
BEGIN
 ASSIGN(F, 'input.txt');
 RESET(F);
 READ(F, A, B)
 CLOSE(F);
END;
```

Проблемы синтаксического анализа в IDE

1. Код в IDE обычно не дописан
    - Нельзя парсить только до первой ошибки
    - *Что делать?*

Проблемы синтаксического анализа в IDE

1. Код в IDE обычно не дописан
    - Нельзя парсить только до первой ошибки
    - Решение: восстановление при ошибках
    - Пример: разбирать до “;” или “END” в Pascal

Проблемы синтаксического анализа в IDE

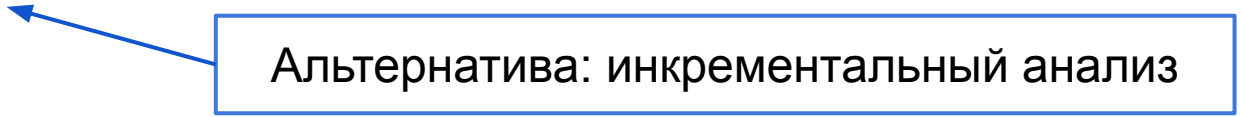
1. Код в IDE обычно не дописан
    - Нельзя парсить только до первой ошибки
    - Решение: восстановление при ошибках
    - Пример: разбирать до “;” или “END” в Pascal
2. Программист быстро печатает
    - Нельзя выполнять разбор на каждый символ
    - *Что делать?*

Проблемы синтаксического анализа в IDE

1. Код в IDE обычно не дописан
    - Нельзя парсить только до первой ошибки
    - Решение: восстановление при ошибках
    - Пример: разбирать до “;” или “END” в Pascal
2. Программист быстро печатает
    - Нельзя выполнять разбор на каждый символ
    - Решение: запускать через 0.3 секунды после последнего символа
    - Решение: выполнять в фоновом потоке, передавая версию текста (целое число)

Проблемы синтаксического анализа в IDE

1. Код в IDE обычно не дописан
   - Нельзя парсить только до первой ошибки
   - Решение: восстановление при ошибках
   - Пример: разбирать до “;” или “END” в Pascal
2. Программист быстро печатает
   - Нельзя выполнять разбор на каждый символ
   - Решение: запускать через 0.3 секунды после последнего символа
   - Решение: выполнять в фоновом потоке, передавая версию текста (целое число)



tree-sitter

синтаксический разбор
с разрешением имён

Инструменты на базе КС-грамматик

1. Есть генераторы парсеров — LL(k), LR(1), LALR, GLR
2. Можно добавить разрешение имён
    - Это часть семантического анализа
3. И применить результат в IDE

Инструменты на базе КС-грамматик

1. Есть генераторы парсеров — LL(k), LR(1), LALR, GLR
2. Можно добавить разрешение имён
    - Это часть семантического анализа
3. И применить результат в IDE

Реализации:

- tree-sitter — инкрементальный генератор парсеров, алгоритм GLR
- внутренние инструменты JetBrains

Возможности IDE

На основе tree-sitter можно сделать (для одного файла):

1. Отображение синтаксических ошибок
2. Outline — краткая структура файла
3. …

Возможности IDE

На основе tree-sitter можно сделать (для одного файла):

1. Отображение синтаксических ошибок
2. Outline — краткая структура файла
3. Отображение ошибок класса “undefined variable”
4. Улучшенная подсветка кода
5. Навигация
   - Go to definition
   - Find usages
6. Контекстное автодополнение
   - с учётом имён
   - без учёта типов и иных ограничений

```
BEGIN
  ASSIGN(F, 'input.txt');
  RESET(F);
  READ(FF, A, B);
  CLOSE(F);
END;
```

Утилита grep-ast

Идея:

1. Ищет в файле по подстрокам — как grep
2. Разбирает файл — с помощью tree-sitter
3. Расширяет вывод, чтобы показать блок

Вывод grep:

```
Temp := A MOD B;
```

Вывод grep-ast:

```
BEGIN
  Temp := A MOD B;
  A := B;
  B := Temp;
END;
```

Семантический анализ

Семантический анализ

Обходы дерева для анализа правил языка:

1. Разрешение имён
2. Вычисление типов
3. Проверка типов
4. Проверка контекстно-зависимых правил
5. Импорт модулей

AST → annotated AST

Возможности IDE

Полный семантический анализ нужен для фич:

1. Семантическая подсветка
2. Отображение ошибок семантики
3. Навигация по всему проекту
4. Контекстное автодополнение
   - `object.<методы>`
   - `function(<параметр>)`

```
BEGIN
 ASSIGN(F, 'input.txt');
 RESET(F);
 READ(F, A, B)
 CLOSE(A);
END;
```

Проблема расхождения компилятора и IDE

Ситуация:

1. Вышел Go 1.22, где можно выполнять range по целым числам
2. Компилятор поддерживает это
3. IDE — не поддерживает

```
n := 10
for i := range n {
    // перебор i от 0 до n-1
}
```

Проблема расхождения компилятора и IDE

Ситуация:

1. Вышел Go 1.22, где можно выполнять range по целым числам
2. Компилятор поддерживает это
3. IDE — не поддерживает

```
n := 10
for i := range n {
    // перебор i от 0 до n-1
}
```

Проблема расхождения компилятора и IDE

Ситуация:

1. Вышел Go 1.22, где можно выполнять range по целым числам
2. Компилятор поддерживает это
3. IDE — не поддерживает

```
n := 10
for i := range n {
    // перебор i от 0 до n-1
}
```

Как решить эту проблему?

Две парадигмы построения IDE

1. IDE со своим разбором
    - Лексика, синтаксис, семантика — для каждого языка
    - Автодополнение, автоформатирование — для каждого языка
    - Статический анализ — если получится

Две парадигмы построения IDE

1. IDE со своим разбором
    - Лексика, синтаксис, семантика — для каждого языка
    - Автодополнение, автоформатирование — для каждого языка
    - Статический анализ — если получится
2. Компилятор как библиотека
    - libclang (C API) — AST + семантика, автодополнение
    - libpython-dev (C API) — AST + семантика

Примеры

1. IDE со своим разбором
    - Все IDE JetBrains
    - Старые версии QtCreator, CodeBlocks
2. Компилятор как библиотека
    - Visual Studio для C++ и C#
    - XCode с libClang
    - QtCreator с libClang

Проблема: разные языки

IDE написана на одном языке:

1. QtCreator — C++ (Qt)
2. Visual Studio — C#
3. VSCode — JavaScript / TypeScript (Chromium)

Проблема: разные языки

IDE написана на одном языке:

1. QtCreator — C++ (Qt)
2. Visual Studio — C#
3. VSCode — JavaScript / TypeScript (Chromium)

Компилятор — на другом:

1. Компилятор Go — на Go
2. Компилятор C# — на C#

Проблема: разные языки

IDE написана на одном языке:

1. QtCreator — C++ (Qt)
2. Visual Studio — C#
3. VSCode — JavaScript / TypeScript (Chromium)

Компилятор — на другом:

1. Компилятор Go — на Go
2. Компилятор C# — на C#

Как решить эту проблему?

Проблема: разные языки

IDE написана на одном языке:

1. QtCreator — C++ (Qt)
2. Visual Studio — C#
3. VSCode — JavaScript / TypeScript (Chromium)

Компилятор — на другом:

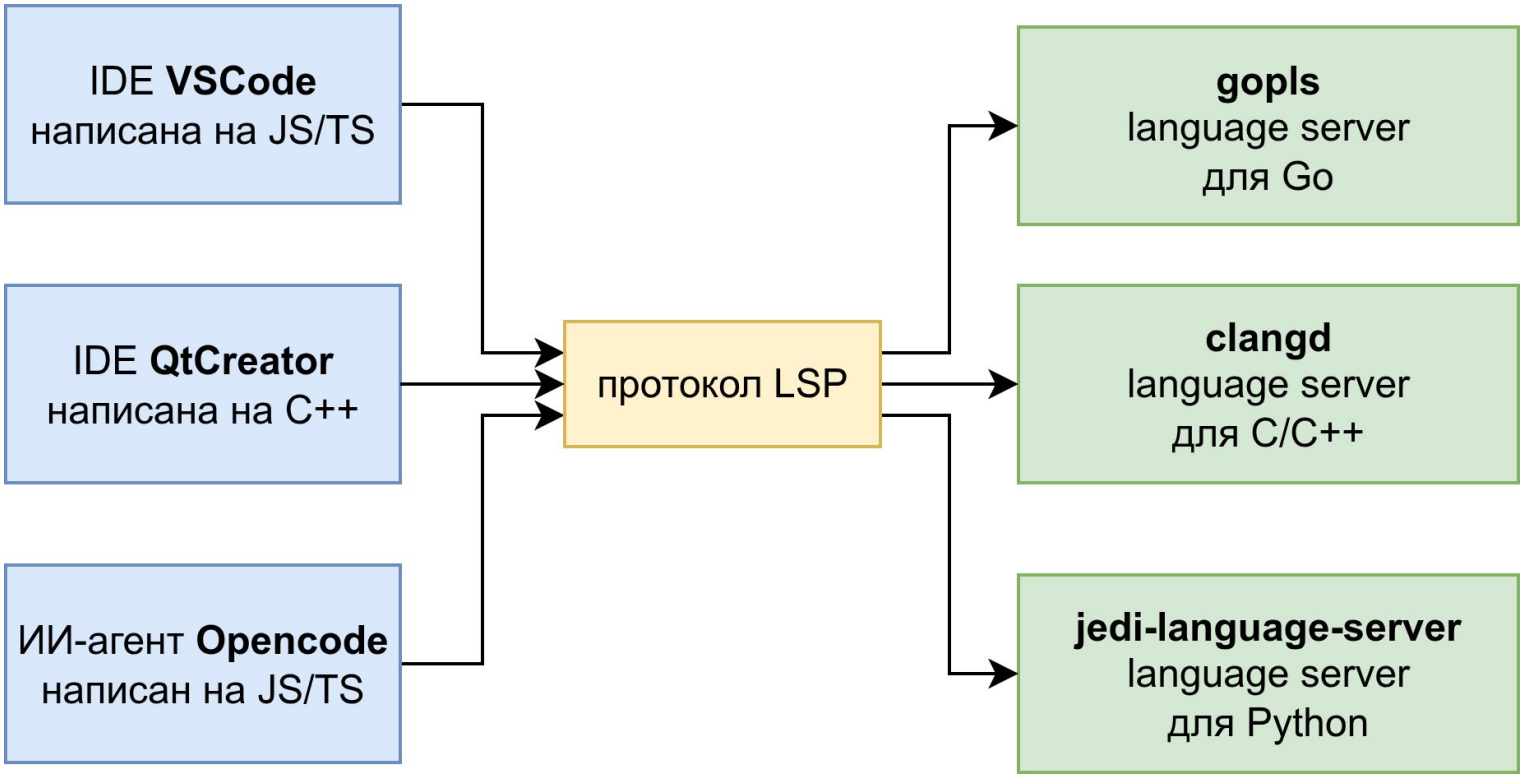
1. Компилятор Go — на Go
2. Компилятор C# — на C#

Как решить эту проблему?

1. FFI
2. LSP

Language Server Protocol

Клиент-серверный протокол от Microsoft



Особенности LSP

1. Основан на JSON-RPC
    - Запросы
    - Уведомления о событиях

Особенности LSP

1. Основан на JSON-RPC
    - Запросы
    - Уведомления о событиях
2. Фокус на IDE
    - IDE запускает language-server как дочерний процесс
    - … и перезапускает, если он упал
    - IDE передаёт несохранённые файлы

Особенности LSP

1. Основан на JSON-RPC
    - Запросы
    - Уведомления о событиях
2. Фокус на IDE
    - IDE запускает language-server как дочерний процесс
    - … и перезапускает, если он упал
    - IDE передаёт несохранённые файлы
3. Расширяемость
    - Есть обнаружение возможностей
    - Все методы опциональны

Подытожим

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

Конец!