Таблицы символов

Лекция №7

Цель

Добавить в язык поддержку пошаговых вычислений

1. Объявление переменных и констант
2. Последовательное выполнение инструкций
3. Ввод / вывод

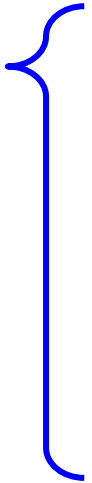
Структура типичных ЯП

Терминология в грамматиках

1. Программа — program, translation unit
2. Определение — definition (def)
3. Объявление — declaration (decl)
4. Инструкция — statement (stmt)
5. Выражение — expression (expr)

Терминология в грамматиках

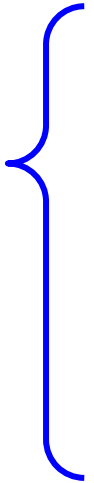
1. Программа — program, translation unit
2. Определение — definition (def)
3. Объявление — declaration (decl)
4. Инструкция — statement (stmt)
5. Выражение — expression (expr)



- Function def
- Class def
- Type def
- Constant def
- Variable def

Терминология в грамматиках

1. Программа — program, translation unit
2. Определение — definition (def)
3. Объявление — declaration (decl)
4. Инструкция — statement (stmt)
5. Выражение — expression (expr)



- Function decl
- Class decl
- Type decl
- Constant decl
- Variable decl

Терминология в грамматиках

1. Программа — program, translation unit
2. Определение — definition (def)
3. Объявление — declaration (decl)
4. Инструкция — statement (stmt)
5. Выражение — expression (expr)

Символы: именованные типы, классы, функции, переменные

Терминология в грамматиках

1. Программа — program, translation unit
2. Определение — definition (def)
3. Объявление — declaration (decl)
4. Инструкция — statement (stmt)
5. Выражение — expression (expr)

Символы: именованные типы, классы, функции, переменные

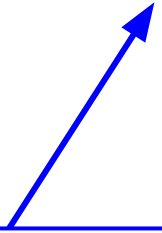


Таблица символов — любая структура данных, позволяющая:

1. Определить символ с именем
2. Получить символ по имени

Терминология в грамматиках

1. Программа — program, translation unit
2. Определение — definition (def)
3. Объявление — declaration (decl)
4. Инструкция — statement (stmt)
5. Выражение — expression (expr)

- Assignment stmt
- Expression stmt
- Return stmt
- Import stmt

Обычно выполняются последовательно

Три парадигмы

Три парадигмы

1. Императивная — инструкции выполняются последовательно

Три парадигмы

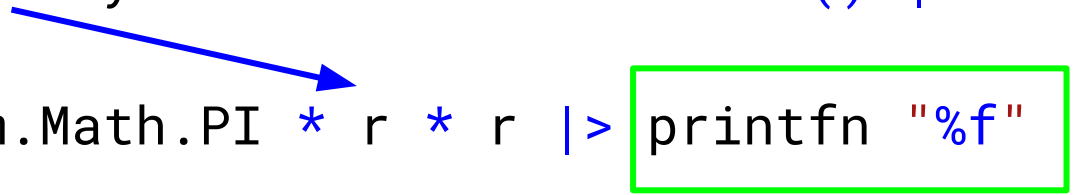
1. Императивная — инструкции выполняются последовательно
2. Функциональная — порядок вычислений произвольный
    - Только зависимости фиксируют порядок
    - «Ленивые вычисления»

```
let r = System.Console.ReadLine() |> float

System.Math.PI * r * r |> printfn "%f"
```

Три парадигмы

1. Императивная — инструкции выполняются последовательно
2. Функциональная — порядок вычислений произвольный
   - Только зависимости фиксируют порядок
   - «Ленивые вычисления»

```
let r = System.Console.ReadLine() |> float

System.Math.PI * r * r |> printfn "%f"
```



Три парадигмы

1. Императивная — инструкции выполняются последовательно
2. Функциональная — порядок вычислений произвольный
3. Объектно-ориентированная — всё рассматривается как объекты

```
class Program {
    static void Main() {
        double r = double.Parse(Console.ReadLine());
        Console.WriteLine(Math.PI * r * r);
    }
}
```

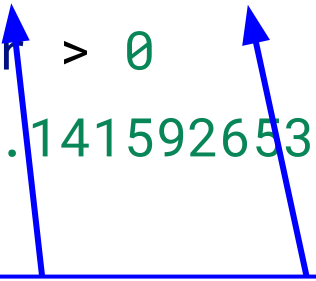
Обзор особенностей ЯП

Язык Python 2 — площадь круга

```
r = float(raw_input())
assert r > 0
print 3.141592653589793 * r * r
```

Язык Python 2 — площадь круга

```
r = float(raw_input())
assert r > 0
print 3.141592653589793 * r * r
```



В Python есть встроенные (builtin) символы

1. Тип `float`
2. Функция `raw_input`

Язык Python 2 — площадь круга

```
r = float(raw_input())
assert r > 0
print 3.141592653589793 * r * r
```

В Python 2 это отдельные инструкции:

```
assert_stmt = "assert", expr, [ ",", expr ] ;
print_stmt = "print", expr, { ",", expr } ;
```

Язык Pascal — площадь круга

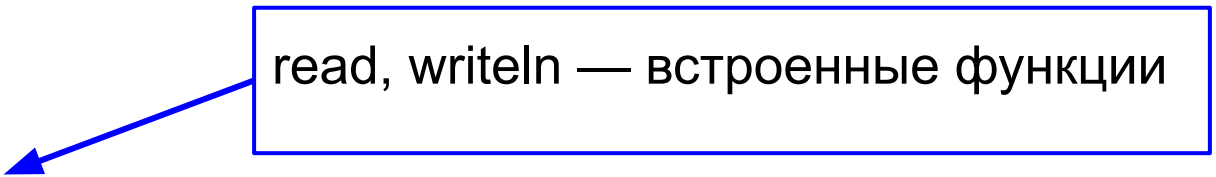
```
program CircleSquare;
var r: real;
begin
   read(r);
   writeln(3.141592 * r * r)
end.
```

Язык Pascal — площадь круга

```
program CircleSquare;
var r: real;
begin
   read(r);
   writeln(3.141592 * r * r)
end.
```

Язык Pascal — площадь круга

```
program CircleSquare;
var r: real;
begin
   read(r);
   writeln(3.141592 * r * r)
end.
```



Язык Pascal — площадь круга

```
program CircleSquare;
var r: real;
begin
   read(r);
   writeln(3.141592 * r * r)
end.
```

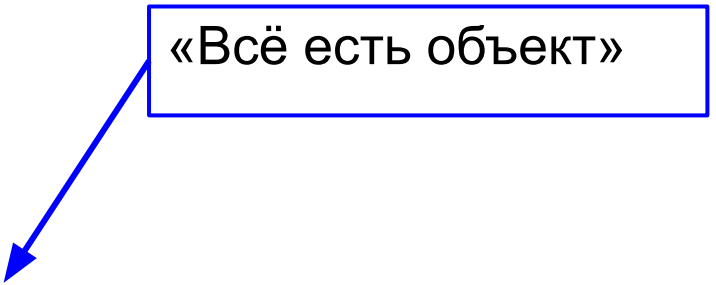
- ";" — разделитель инструкций
- "." — завершает программу

Язык Java — площадь круга

```java
import java.util.Scanner;

public class CircleSquare {
    public static void main(String[] args) {
        double r = new Scanner(System.in).nextDouble();
        System.out.println(Math.PI * r * r);
    }
}
```

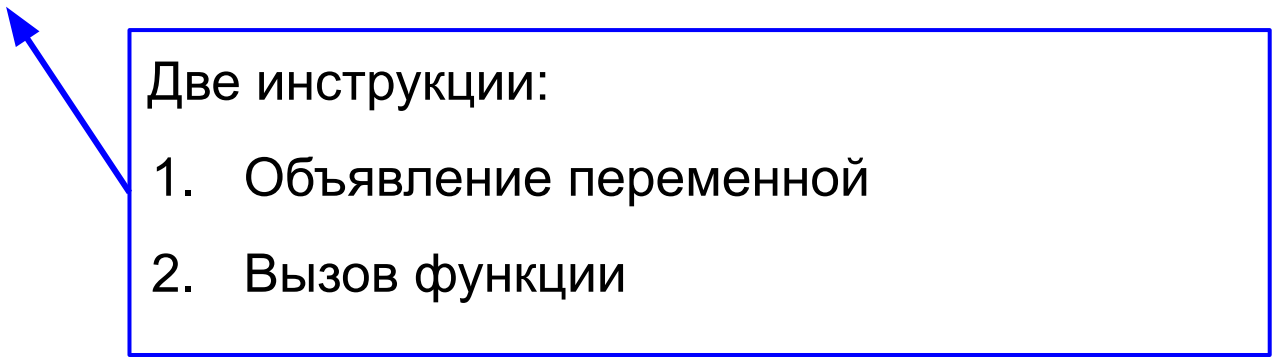
Язык Java — площадь круга

```
import java.util.Scanner;


public class CircleSquare {
    public static void main(String[] args) {
        double r = new Scanner(System.in).nextDouble();
        System.out.println(Math.PI * r * r);
    }
}
```



Язык Java — площадь круга

```
import java.util.Scanner;

public class CircleSquare {
    public static void main(String[] args) {
        double r = new Scanner(System.in).nextDouble();
        System.out.println(Math.PI * r * r);
    }
}
```



Язык C — площадь круга

```c
#include <stdio.h>

int main() {
    double r;
    scanf("%lf", &r);
    printf("%.6f\n", 3.14158 * r * r);
    return 0;
}
```

Язык C — площадь круга

```c
#include <stdio.h>

int main() {
    double r;
    scanf("%lf", &r);
    printf("%.6f\n", 3.14158 * r * r);
    return 0;
}
```

1. Есть препроцессор
2. Лексер взаимодействует с процессором
3. Парсер получает готовые лексемы

Язык C — площадь круга

```c
#include <stdio.h>

int main() {
    double r;
    scanf("%lf", &r);
    printf("%.6f\n", 3.14158 * r * r);
    return 0;
}
```

Внешние функции scanf / printf.

Реализации обеспечит компоновщик (linker)

Язык C — площадь круга (без #include)

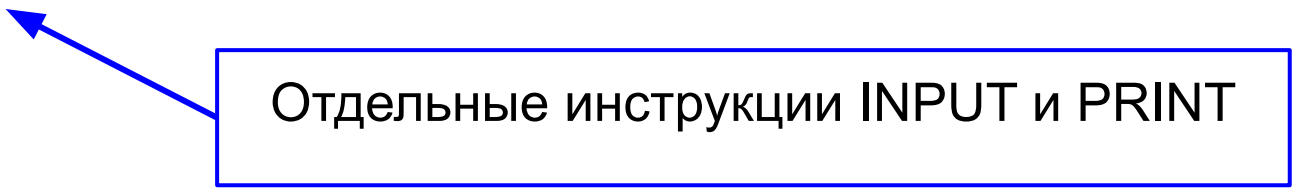
```c
extern int scanf(const char *format, ...);
extern int printf(const char* format, ...);

int main() {
    double r;
    scanf("%lf", &r);
    printf("%.6f\n", 3.14158 * r * r);
    return 0;
}
```

Внешние функции scanf / printf.

Реализации обеспечит компоновщик (linker)

Язык BASIC — площадь треугольника

```
INPUT X1, Y1, X2, Y2, X3, Y3
A = SQR((X1-X2)^2 + (Y1-Y2)^2)
B = SQR((X2-X3)^2 + (Y2-Y3)^2)
C = SQR((X3-X1)^2 + (Y3-Y1)^2)
P = (A + B + C) / 2
S = SQR(P * (P-A) * (P-B) * (P-C))
PRINT S
```



Проектирование языка

Выбор парадигмы

Императивный стиль

1. Последовательное выполнение
2. Есть переменные

Функциональный стиль:

1. Произвольный порядок — но с учётом зависимостей
2. Неизменяемые переменные
3. Минимум побочных эффектов

Проблема неиспользуемых выражений

```
r = float(raw_input())
r > 0
3.141592653589793 * r * r
```

Проблема неиспользуемых выражений

```
r = float(raw_input())
r > 0
3.141592653589793 * r * r
```

Решения:

1. Принять «как есть»
2. Запрет на уровне правил грамматики
3. Выводить предупреждения

Нужны ли top level инструкции?

```
public class Program {
    public static void main(String[] args) {
        double r = new Scanner(System.in).nextDouble();
        System.out.println(Math.PI * r * r);
    }
}
```

Решения на уровне грамматики:

1. Разрешить top level инструкции
2. Требовать точку входа

Переменные

1. Изменяемые или неизменяемые
2. Явное определение или неявное?
    - var x = 10 или x = 10
3. Разрешено ли переопределение?
4. Есть ли области видимости?

Ввод-вывод

Варианты реализации:

1. Отдельные инструкции:
    - `print_stmt = "print" expr`
2. Встроенные функции (объекты, модули)
3. Импорт функций (объектов, модулей)
4. Отдельные операторы
5. На уровне соглашений:
    - все вычисленные значения печаются в консоль

LLVM Kaleidoscope

LLVM Kaleidoscope — примеры

```
# С разделителями
def pi 3.14159;
pi * 4.0 * 4.0;

# Без разделителей
def pi 3.14159
pi * 4.0 * 4.0
```

LLVM Kaleidoscope — примеры

```
# С разделителями
def pi 3.14159;
pi * 4.0 * 4.0;

# Без разделителей
def pi 3.14159
pi * 4.0 * 4.0
```

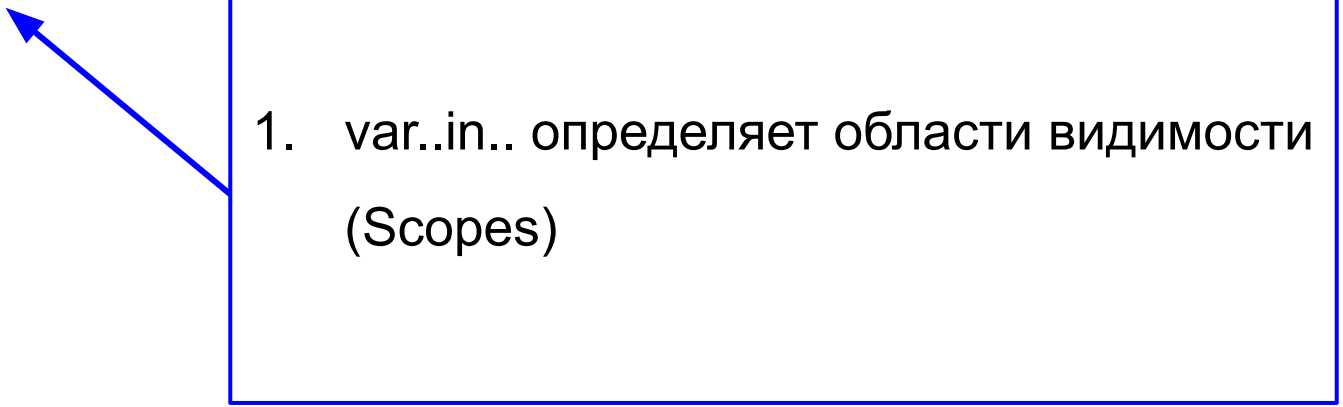
1. def определяет константу
2. Интерпретатор печатает результат каждой инструкции

LLVM Kaleidoscope — примеры

```
var
    x = 1,
    y = 2,
    z = 3
in x + y * z;
```

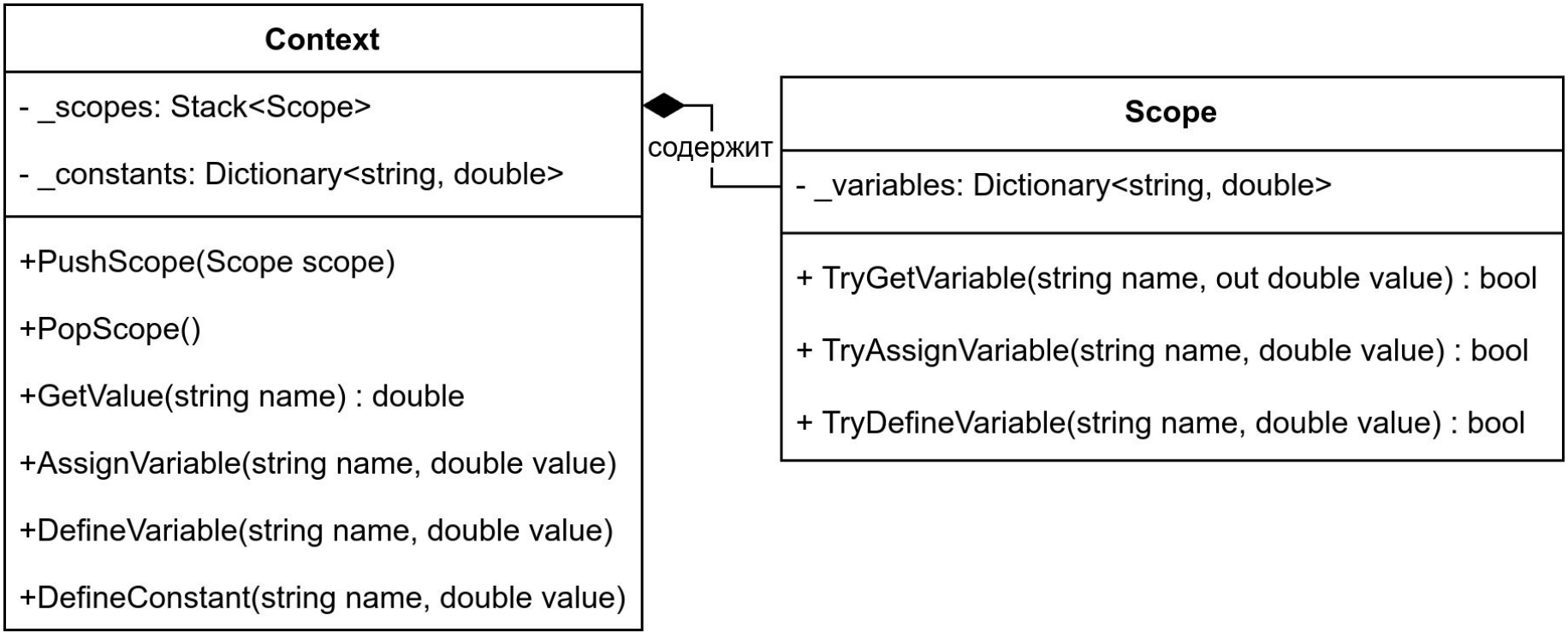
LLVM Kaleidoscope — примеры

```
var
    x = 1,
    y = 2,
    z = 3
in x + y * z;
```

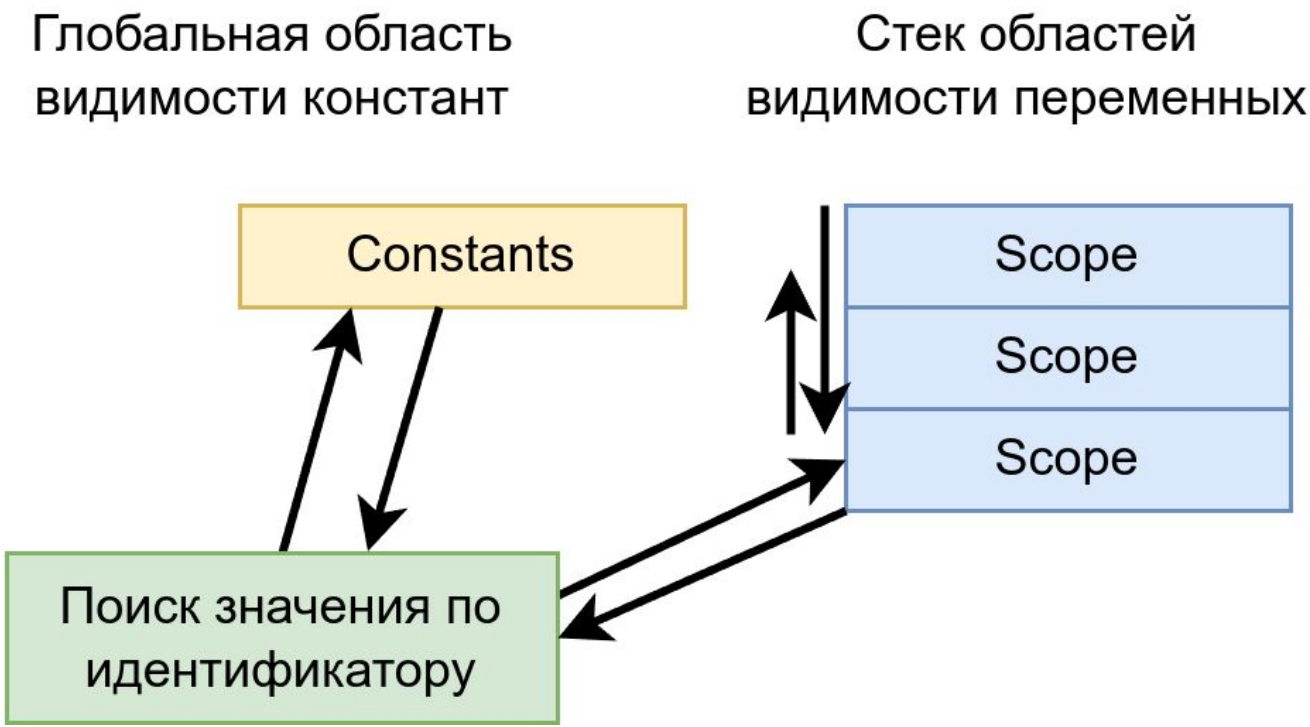


Реализация

Области видимости



Области видимости



Грамматика

```
var_def_scope =
  "var", var_def_list, "in", expression ;
var_def_list =  var_def, { ",", var_def } ;
var_def = identifier,
  [ "=", logical_or_expression ];
```

Пример:

```
var
    x = 1,
    y = 2,
    z = 3
in x + y * z;
```

Правило разбора

```
private double ParseVariableDefinitionScope() {
    Match(TokenType.Var);
    _context.PushScope(new Scope());
    try {
        ParseVariableDefinitionList();
        Match(TokenType.In);
        return ParseExpression();
    } finally {
        _context.PopScope();
    }
}
```

Пример:

```
var
    x = 1,
    y = 2,
    z = 3
in x + y * z;
```

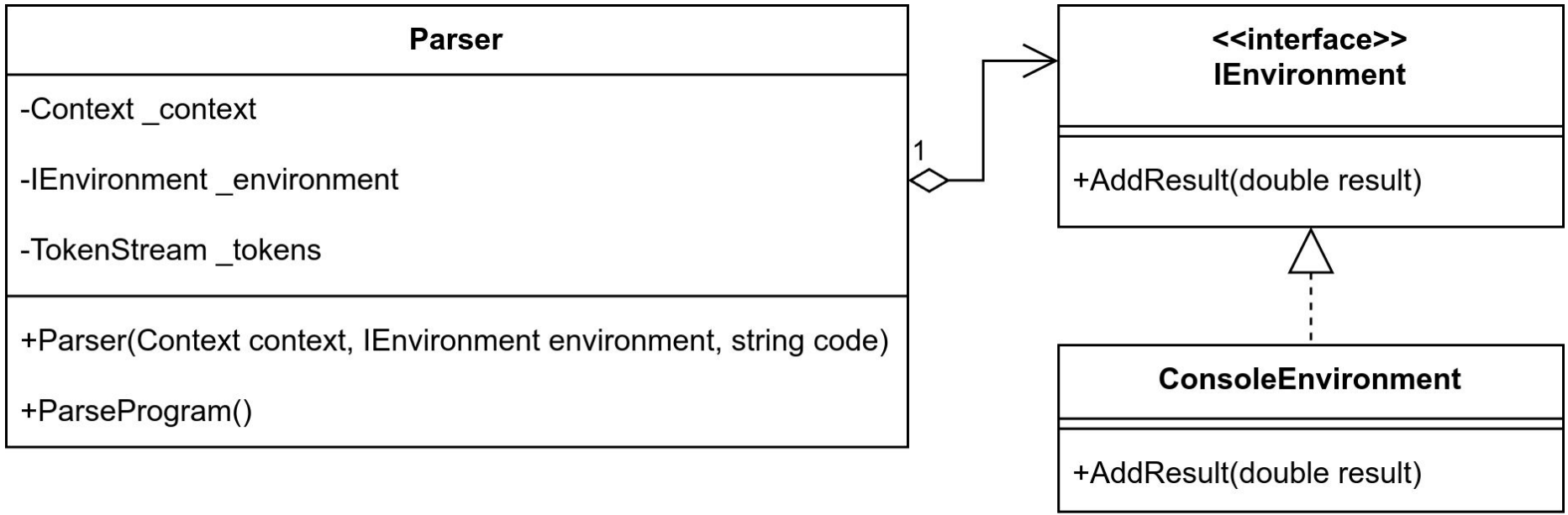
Правило разбора

```
private void ParseVariableDefinition() {
    string name = Match(TokenType.Identifier).Value!.ToString();
    double value = 0;
    if (_tokens.Peek().Type == TokenType.Assign) {
        _tokens.Advance();
        value = ParseRelationalExpression();
    }
    _context.DefineVariable(name, value);
}
```

Пример:

```
var
    x = 1,
    y = 2,
    z = 3
in x + y * z;
```

Абстракция для ввода-вывода



Пример PsKaleidoscope

Пример к лекции:

https://sourcecraft.dev/sshambir-public/pskaleidoscope

Конец.
Вопросы?