Типы данных

Лекция №10

Цель лекции

Познакомиться с системой типов:

1. В учебном языке Tiger
2. В промышленных языках: C#, C++, Go, JavaScript, …

Синтаксис и семантика языков программирования

Синтаксис

Способ разбора лексем в «предложения» языка

1. Правила КС-грамматики
2. Приоритет и ассоциативность операторов

Семантика

???

Синтаксис и семантика языков программирования

Синтаксис

Способ разбора лексем в «предложения» языка

1. Правила КС-грамматики
2. Приоритет и ассоциативность операторов

Семантика

Смысл и поведение языковых конструкций.

1. Правила работы с типами данных
2. Способ отображения имён на узлы AST
   - англ. name resolution / symbol linking
3. Контекстно-зависимые правила
   - Запрет break и continue вне циклов

Язык Tiger

Язык Tiger

- Учебный язык программирования
- Разработан Andrew Appel для книг и курсов в Princeton University

Язык Tiger

- Учебный язык программирования
- Разработан Andrew Appel для книг и курсов в Princeton University

```
let
    var a : int := 0
in
    for i := 0 to 100 do (
        a := a + 1;
        ()
    )
end
```

Типы данных в Tiger

Типы данных:

1. Строки — string
2. Целые числа — int
3. Структуры и массивы

Типы данных в Tiger

Типы данных:

1. Строки — string
2. Целые числа — int
3. Структуры и массивы

Особенности:

1. Выражения и функции могут ничего не возвращать
2. Нет приведений типов — как явных, так и неявных

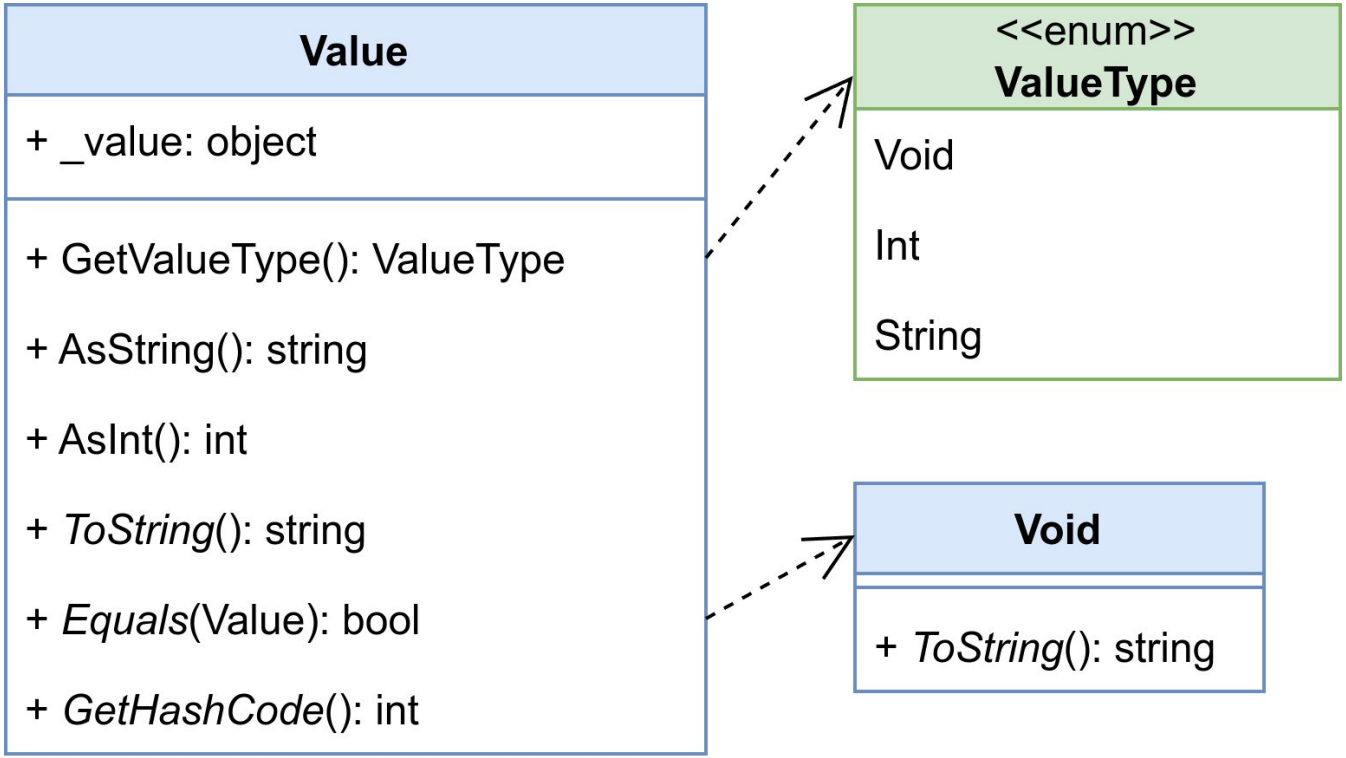
Типы данных в интерпретаторе Tiger

Цель

Реализовать интерпретатор Tiger

- с поддержкой int и string
- без поддержки массивов и структур

Класс Value


Value
+ _value: object
+ GetValueType(): ValueType
+ AsString(): string
+ AsInt(): int
+ ToString(): string
+ Equals(Value): bool
+ GetHashCode(): int
<<enum>>
ValueType
Void
Int
String
Void
+ ToString(): string

Сопоставление по шаблону в C#

```
public string AsString() {
    return _value switch {
        string s => s,
        _ => throw new InvalidOperationException($"Value
{_value} is not a string"),
    };
}
```

Сопоставление по шаблону в C#

```
public ValueType GetValueType() {
    return _value switch  {
        string => ValueType.String,
        int => ValueType.Int,
        Void => ValueType.Void,
        _ => throw new InvalidOperationException($"Unexpected
value {_value} of type {_value.GetType()}"),
    };
}
```

Сложность вычислений при динамической типизации

В языке Tiger:

1. Операторы `+, -, *, /, &, |` работают с числами
2. Операторы `<, >, <=, >=` работают с числами и строками
3. Операторы `=, <>` работают с числами, строками и ссылочными типами

Проблемы:

1. Бинарные операции — двойная диспетчеризация по типам
2. Много проверок на типы

Решение №1: разделить проверки и вычисления

Сделаем две реализации IAstVisitor:

1. `AstTypeChecker` — проверяет соответствие типов
    - не вычисляет результаты
    - относится к семантическим проверкам
2. `AstEvaluator` — выполняет интерпретацию по AST
    - предполагает, что типы уже проверены

Решение №2: ситуативный рефакторинг

```
public void Visit(BinaryOperationExpression e) {
    e.Left.Accept(this);
    e.Right.Accept(this);
    Value right = _values.Pop(), left = _values.Pop();
    _values.Push(EvaluationUtil.ApplyBinaryOperation(
        e.Operation, left, right
    ));
}
```

Решение №2: ситуативный рефакторинг

```
public void Visit(BinaryOperationExpression e) {
    e.Left.Accept(this);
    e.Right.Accept(this);
    Value right = _values.Pop(), left = _values.Pop();
    _values.Push(EvaluationUtil.ApplyBinaryOperation(
        e.Operation, left, right
    ));
}
```

switch-case внутри вызываемого метода

Числовые типы данных в C++, C#, Go

Проблема №1 — размеры типов

Типы целых чисел со знаком

Тип	Размер в C++	Размер в C#	Размер в Go
`short`	`2 ≤ short`	2	–

Проблема №1 — размеры типов

Типы целых чисел со знаком

Тип	Размер в C++	Размер в C#	Размер в Go
`short`	`2 ≤ short`	`2`	`–`
`int`	`2 ≤ int` `short ≤ int`	`4`	`4 на 32-битных,` `8 на 64-битных`

Проблема №1 — размеры типов

Типы целых чисел со знаком

Тип	Размер в C++	Размер в C#	Размер в Go
`short`	`2 ≤ short`	`2`	`–`
`int`	`2 ≤ int` `short ≤ int`	`4`	`4 на 32-битных,` `8 на 64-битных`

`long`	`4 ≤ long`
	`int ≤ long`
	`long = ptr size(*)`
	
	** кроме Windows*
		`8`	`–`

Проблема №1 — размеры типов

Типы чисел с плавающей точкой (IEEE 754)

Тип	Размер (байт)	Экспонента (бит)	Мантисса (бит)
`float` *IEEE 754 binary32*	4	8	23
`double` *IEEE 754 binary64*	8 (*) ** иногда 4 в embedded*	11	52

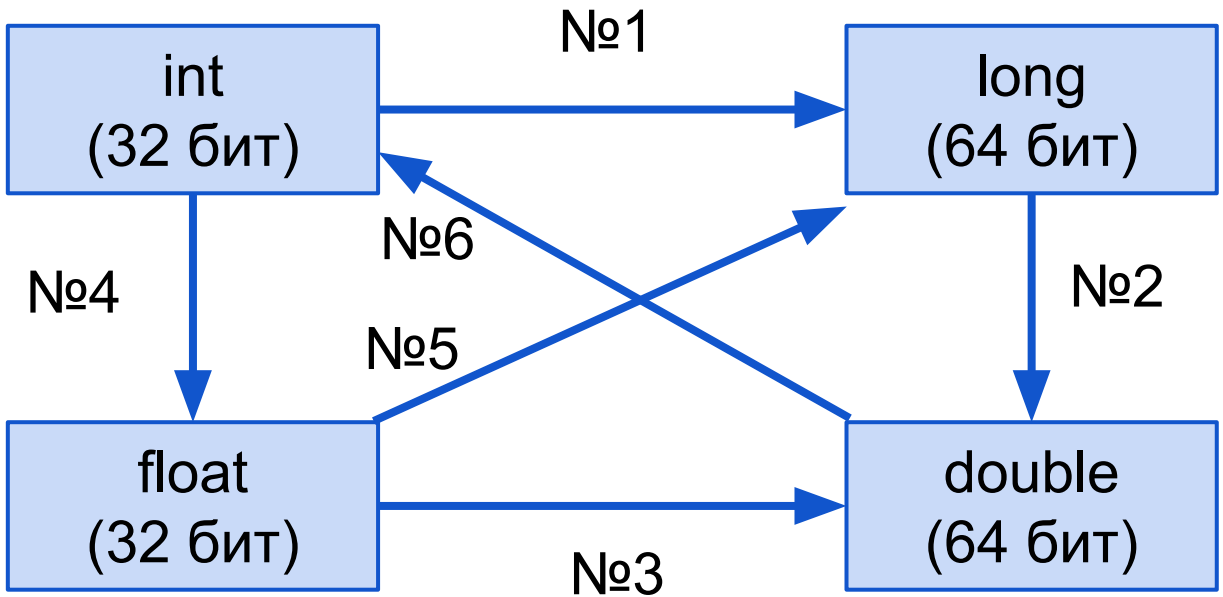
Проблема №2: потери при преобразовании типа

Преобразование меняет диапазон значений:

1. **Narrow cast** — преобразование из более широкого диапазона в более узкий
2. **Wide cast** — преобразование из более узкого диапазона в более широкий

Narrow cast — это риск потери данных.

Какие преобразования являются narrow cast?



Запреты неявных преобразований типов

Преобразование	C++	C#	Go
narrow cast	разрешён (warning)	запрещён	запрещён
wide cast	разрешён	разрешён	запрещён

Проблема №3: двоичные и десятичные дроби

Есть десятичная дробь:

$$1 / 5 = 0.2$$

В двоичном виде это:

$$0.2_{10} = 0.001100110011\ldots_{2} = 0.(0011)_{2}$$

Проблема: 0.2 нельзя представить точно числом с плавающей точкой.

Выводы

1. Платформы имеют отличия
    - 16, 32, 64-разрядные
    - Linux и Windows
    - Разные архитектуры: Intel/AMD, ARM, RISK-V, …
2. Неявные преобразования — источник проблем
    - Narrow cast = потеря данных
    - Wide cast = ошибка выбора типа

Числовые типы данных в JavaScript, PHP, Python

Скриптовые языки

1. Предназначены для интерпретации
2. Имеют динамическую типизацию
3. Чем меньше базовых данных — тем лучше

Скриптовые языки

1. Предназначены для интерпретации
2. Имеют динамическую типизацию
3. **Чем меньше базовых данных — тем лучше**

Каждый новый тип — это:

1. N правил взаимодействия со старыми типами
2. Дилемма: неявные преобразования или ошибки в runtime

Числа в JavaScript

Два типа данных: Number и BigInt

Числа в JavaScript

Два типа данных: Number и BigInt

Number — это два способа представления внутри V8

Тип данных в V8	Нижележащий тип (32-битные системы)	Нижележащий тип (64-битные системы)
Small Integer	int (31 бит)	int (32 бита)
Heap Number	double	double

Числа в PHP

Два типа данных: int и float

Числа в PHP

Два типа данных: int и float

1. Размер int:
    - 32 бита на 32-битных платформах
    - 64 бита на 64-битных платформах
2. Размер float: 64 бита
3. Автоматическое преобразование
    - `int / int → float`
    - `int + float → float`
    - `… и так далее`

Type hints в PHP

1. Типы указываются
   - для функций — параметры, возвращаемое значение
   - для классов — типы полей и констант
2. Проверяются интерпретатором PHP
3. Два режима:
   - нестрогий
   - строгий (`strict_types=1`)

Type hints в PHP

```php
declare(strict_types=1);


function sphereVolume(float $radius): float {
    return (4/3) * M_PI * $radius ** 3;
}
$volume = sphereVolume(5.0);
$volume = sphereVolume(5);
```

Type hints в PHP

```php
declare(strict_types=1);


function sphereVolume(float $radius): float {
    return (4/3) * M_PI * $radius ** 3;
}
$volume = sphereVolume(5.0);
$volume = sphereVolume(5);
```

Вопрос: что произойдёт при передаче int?

Type hints в Python

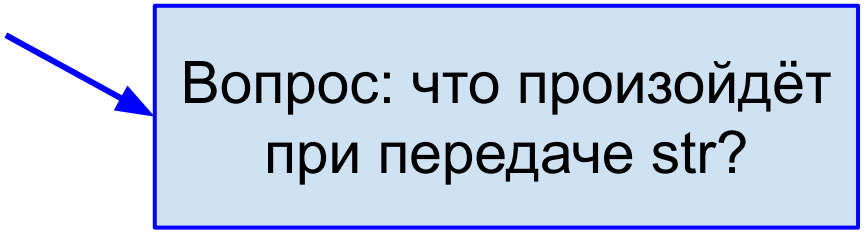
1. Типы указываются
    - для функций — параметры, возвращаемое значение
    - для классов — типы полей класса
2. Не проверяются интерпретатором
    - предназначены для инструментов (линтеры и т.д.)
    - добавляют читаемость

Type hints в Python

```python
import math


def sphere_volume(radius: float) -> float:
    return (4/3) * math.pi * radius ** 3


volume = sphere_volume(5.0)
volume = sphere_volume("5.0")
```

Type hints в Python

```python
import math


def sphere_volume(radius: float) -> float:
    return (4/3) * math.pi * radius ** 3


volume = sphere_volume(5.0)
volume = sphere_volume("5.0")
```



Выводы

В скриптовых языках — свои проблемы

1. Чем меньше типов, тем лучше
2. Сложно выбрать даже тип для чисел
    - Дилемма: один Number или два Int + Float?
    - Какие правила преобразования?

Вывод типов
(type inference)

Вывод типов в Go

- Для функции типы указываются всегда
- Для переменных — доступен **вывод типа** (type inference)

Вывод типов в Go

- Для функции типы указываются всегда
- Для переменных — доступен **вывод типа** (type inference)

```
func sphereVolume(radius float64) float64 {
    cube := radius * radius * radius
    volume := (4.0 / 3.0) * math.Pi * cube
    return volume
}
```

Вывод типов в C++23

- Вывод типов доступен везде
- Последствия — рост времени компиляции

```
auto sphereVolume(auto radius) {
    auto cube = radius * radius * radius;
    auto volume = (4.0 / 3.0) * M_PI * cube;
    return volume;
}
```

Выводы

Дилемма: где вывод типов, а где обязательное объявление?

Решения:

- вывод типов внутри функции → простая реализация
- глобальный вывод типов → сложная реализация, сложно оптимизировать

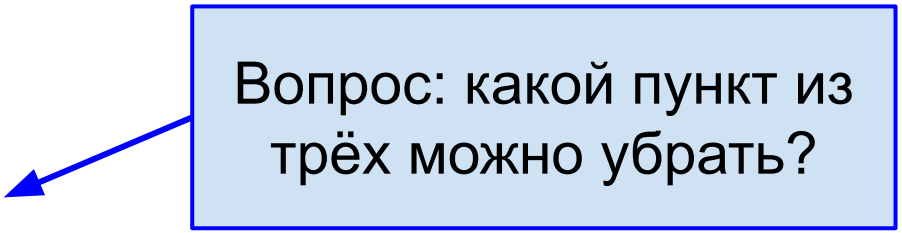
Строковый тип данных

Минимум для работы со строками

1. Конкатенация строк
2. Декомпозиция строки
    - Получение размера
    - Получение подстроки
    - Обход строки по символам

Минимум для работы со строками

1. Конкатенация строк
2. Декомпозиция строки
   - Получение размера
   - Получение подстроки
   - Обход строки по символам



Конкатенация строк

Язык	Способ конкатенации
Python	`"Hello, " + "World!"`
JavaScript	`"Hello, " + "World!"`
PHP	`"Hello, " . "World!"`
Tiger (*учебный*)	`concat("Hello, ", "World!")`

Получение подстроки

Язык	Получение подстроки
Python	`str[i:i+n]`
JavaScript	`str.substring(from, from + count)`
PHP	`substr(str, from, count)`
Tiger (*учебный*)	`substring(str, from, count)`

Булев тип данных

Короткая схема вычислений (Short-circuit evaluation)

Если левый операнд определяет результат выражения, то правый операнд **не вычисляется**.

Короткая схема вычислений (Short-circuit evaluation)

Если левый операнд определяет результат выражения, то правый операнд **не вычисляется**.

Примеры:

1. `true  && foo()` — функция foo будет вызвана
2. `false && foo()` — функция foo **не будет** вызвана
3. `true  || foo()` — ???
4. `false || foo()` — ???

Преобразование bool в строку

Язык	Способ	Результат
C++	`std::format("{}", v)`	`"true" / "false"`
	`std::count << v`	`"0" / "1"` или `"true" / "false"`
C#	`v.ToString()`	`"True" / "False"`
Go	`strconv.FormatBool(v)`	`"true" / "false"`
PHP	`(string)v`	`"1" / ""`

Подытожим

Выводы для аналитика

1. Чем меньше типов — тем проще
2. Чем меньше преобразований типов — тем проще
3. Для каждого типа есть минимальные ожидания
4. Автоматический вывод типов — только в пределах функции

Выводы для разработчика

1. Лучше разделить проверку типов и вычисления
2. Пишите по TDD
3. Соблюдайте минимализм
4. Используйте ситуативный рефакторинг

Спасибо за внимание!
Вопросы?