Типы данных

Лекция №11

Цель лекции

Понять нюансы, связанные с типами данных:

1. Целочисленные типы
2. Типы с плавающей точкой
3. Символьные типы
4. Тип boolean

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

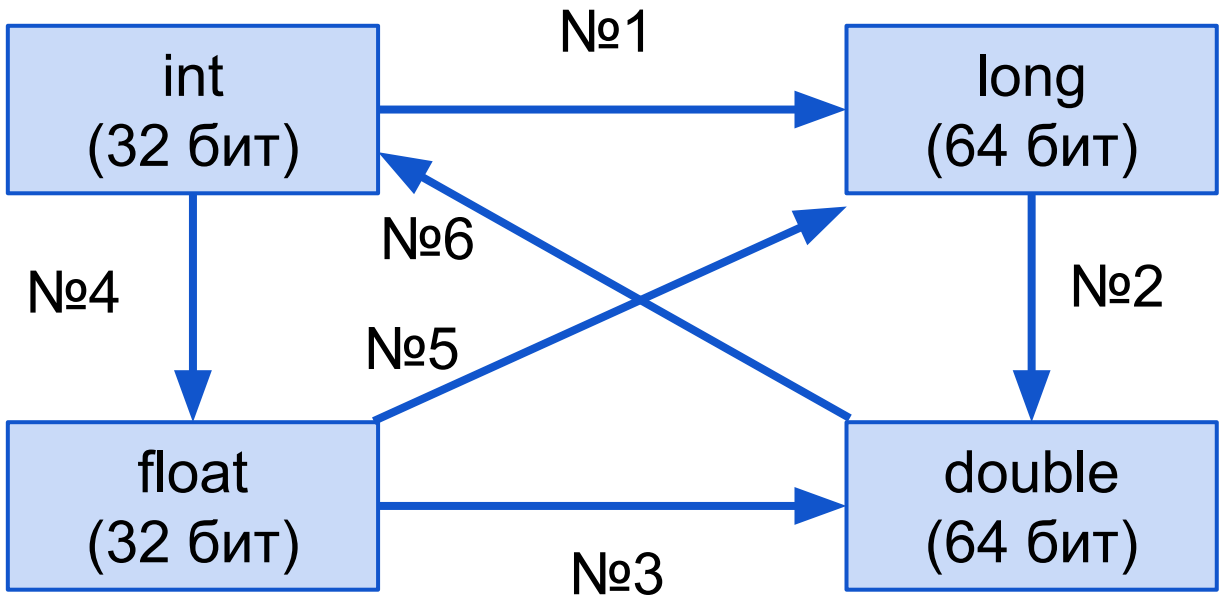
Проблема №2: потери при преобразовании типа

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

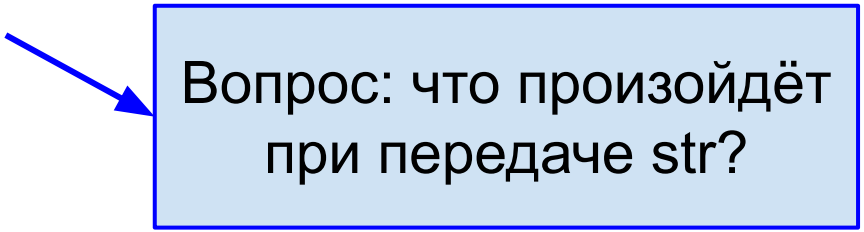
1. Типы указываются
    - для функций — параметры, возвращаемое значение
    - для классов — типы полей класса
2. Не проверяются интерпретатором
    - предназначены для инструментов (линтеры и т.д.)
    - добавляют читаемость

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?

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

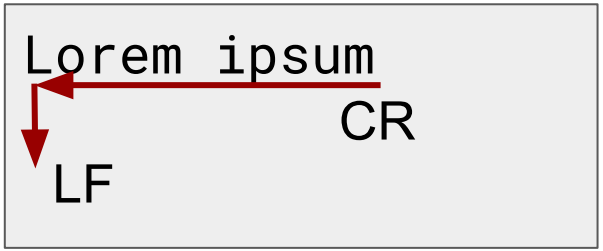
Язык	Способ	Результат
C++	`std::format("{}", v)`	`"true" / "false"`
	`std::count << v`	`"0" / "1"` или `"true" / "false"`
C#	`v.ToString()`	`"True" / "False"`
Go	`strconv.FormatBool(v)`	`"true" / "false"`
PHP	`(string)v`	`"1" / ""`

Текст и Unicode

Концы строк

Два символа:

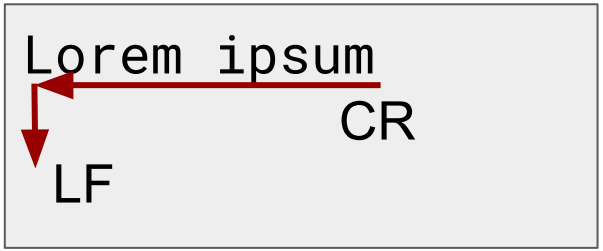
1. LF (line feed, `'\n'`) — перевод на следующую строку
2. CR (carriage return, `'\r'`) — возврат в начало строки



Концы строк

Два символа:

1. LF (line feed, `'\n'`) — перевод на следующую строку
2. CR (carriage return, `'\r'`) — возврат в начало строки



Три типа концов строки в файлах:

- `'\n'` (LF) — стиль Unix и Linux
- `'\r\n'` (CRLF) — стиль MS DOS и Windows
- `'\r'` (CR) — стиль старой Mac OS

Стандарт Unicode

- UTF-8, UTF-16, UTF-32 — кодировки стандарта Unicode

Стандарт Unicode

- UTF-8, UTF-16, UTF-32 — кодировки стандарта Unicode
- Символ Unicode — ровно 21 бит, от 0x000000 до 0x10FFFF

Стандарт Unicode

- UTF-8, UTF-16, UTF-32 — кодировки стандарта Unicode
- Символ Unicode — ровно 21 бит, от 0x000000 до 0x10FFFF
  - 17 плоскостей по 65536 символов = 1 114 112 символов
  - Плоскость BMP — символы с номерами от 0 до 0x00FFFF
  - Символы с 0x000000 по 0x00007F совпадают с ASCII

Кодировки Unicode

Кодировка	Размер codepoint	Размер символа
UTF-8	1 байт	1, 2, 3 или 4 байта
UTF-16	2 байта	2 или 4 байта
UTF-32	4 байта	4 байта

Кодировки Unicode

Кодировка	Размер codepoint	Размер символа
UTF-8	1 байт	1, 2, 3 или 4 байта
UTF-16	2 байта	2 или 4 байта
UTF-32	4 байта	4 байта

UTF-16 и UTF-32 имеют версии BE / LE (меняется порядок байт)

Число	Байты в Big Endian	Байты в Little Endian
0x0410	0x04 | 0x10	0x10 | 0x04

Кодировка UTF-32

Один символ — 4 байта (32 бита)

- номер символа помещается в 21 бит
- остальные биты заполняются нулями

Кодировка UTF-8

Номер символа	Байт №1	Байт №2	Байт №3	Байт №4
0x0000...0x007F	0xxxxxxx	—	—	—
0x0080...0x07FF	110xxxxx	10xxxxxx	—	—
0x0800...0xFFFF	1110xxxx	10xxxxxx	10xxxxxx	—
0x10000...0x10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8 совместим с ASCII

Кодировка UTF-16

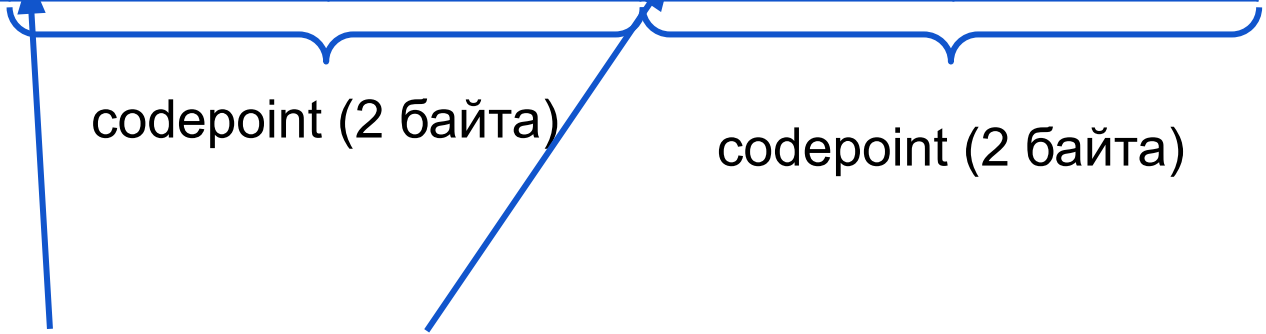
Номер символа	Байт №1	Байт №2	Байт №3	Байт №4
U+0000...FFFF	xxxxxxxx	xxxxxxxx	—	—
U+FFFF...10FFFF	110110xx	xxxxxxxx	110111xx	xxxxxxxx

codepoint (2 байта)

codepoint (2 байта)

Кодировка UTF-16

Номер символа	Байт №1	Байт №2	Байт №3	Байт №4
U+0000...FFFF	xxxxxxxx	xxxxxxxx	—	—
U+FFFF...10FFFF	110110xx	xxxxxxxx	110111xx	xxxxxxxx



В Unicode символы 0xD800...0xDFFF зарезервированы для UTF-16

BOM (byte order mark)

Символ 0xFEFF — позволяет отличить порядок байт

UTF-8	EF BB BF
UTF-16 (BE)	FE FF
UTF-16 (LE)	FF FE
UTF-32 (BE)	00 00 FE FF
UTF-32 (LE)	FF FE 00 00

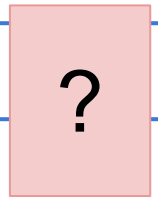
Что такое текстовый файл?

1. Не каждый набор байт является текстом в Unicode
    - Текстовый файл — последовательность байт, которая является текстом
    - Бинарный файл — последовательность байт, которая может и не быть текстом
2. В начале файла может быть BOM (byte order mark)
3. Концы строк могут быть разными — '\n', '\r\n', '\r'

Символьный тип данных

Символьный тип

Язык	8 бит (UTF-8, ASCII)	16 бит (UTF-16)	32 бита (UTF-32)
C++			
C#			
Go			
PHP			



Символьный тип

Язык	8 бит (UTF-8, ASCII)	16 бит (UTF-16)	32 бита (UTF-32)
C++	char char8_t		
C#	byte		
Go	byte		
PHP	—		

?

Символьный тип

Язык	8 бит (UTF-8, ASCII)	16 бит (UTF-16)	32 бита (UTF-32)
C++	char char8_t	`char16_t`	
C#	byte	char	
Go	byte	—	
PHP	—	—	

?

Символьный тип

Язык	8 бит (UTF-8, ASCII)	16 бит (UTF-16)	32 бита (UTF-32)
C++	char char8_t	`char16_t`	char32_t
C#	byte	char	System.Text.Rune
Go	byte	—	rune
PHP	—	—	—

Преобразование символа в число и обратно

Язык	**char → int**	**int → char**
C	`(int)ch`	`(char)ch`
C#	`(int)ch`	`(char)ch`
Go	`int(ch)`	`byte(ch)`
Pascal	`Ord(ch)`	`Chr(ch)`

Строковый тип данных

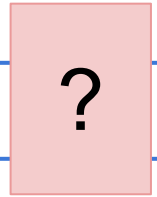
Строковый тип

Язык	UTF-8	UTF-16	UTF-32
C++			
C#			
Go			
PHP			

?

Строковый тип

Язык	UTF-8	UTF-16	UTF-32
C++	std::string std::u8string		
C#	—		
Go	string		
PHP	—		



Строковый тип

Язык	UTF-8	UTF-16	UTF-32
C++	std::string std::u8string	`std::u16string`	
C#	—	string	
Go	string	—	
PHP	—	string	

?

Строковый тип

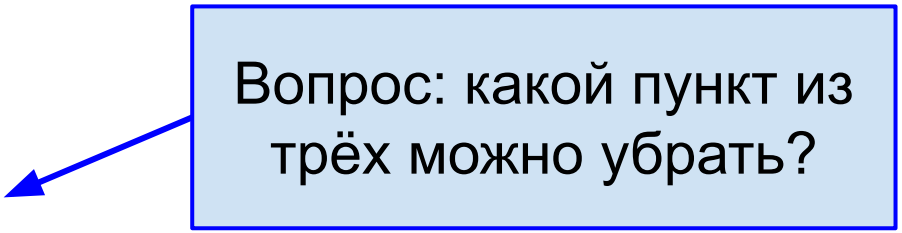
Язык	UTF-8	UTF-16	UTF-32
C++	std::string std::u8string	`std::u16string`	std::u32string
C#	—	string	—
Go	string	—	—
PHP	—	string	—

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)`

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

Подытожим

Составление спецификации языка

1. Чем меньше типов, тем проще

Составление спецификации языка

1. Чем меньше типов, тем проще
2. Чем меньше преобразований типов, тем проще

Составление спецификации языка

1. Чем меньше типов, тем проще
2. Чем меньше преобразований типов, тем проще
3. Для каждого типа набор возможностей должен быть полным

Составление спецификации языка

1. Чем меньше типов, тем проще
2. Чем меньше преобразований типов, тем проще
3. Для каждого типа набор возможностей должен быть полным
4. Автоматический вывод типов лучше делать только в пределах функции

Советы по реализации

1. Добавьте в модуль семантики проход по AST
    - вычисление типов + проверка типов
    - можно совместить с разрешением имён, если язык того требует
2. Используйте ATDD или Canon TDD

Спасибо за внимание!
Вопросы?