Интенсивное программирование: язык Тривиль

Алексей Недоря, апрель 2023

- ❏ Зачем?
- ❏ Первые примеры
- ❏ Основные конструкции
- ❏ Пример: Сборщик строки
- ❏ Пример: Форматная строка
- ❏ Что дальше?

Интенсивное программирование	http://digital-economy.ru/stati/интенсивное-программирование
Языки выходного дня	http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=419
Описание языка Тривиль	https://gitflic.ru/project/alekseinedoria/trivil-0/blob?file=doc%2Freport%2Freport.pdf
Публичный репозиторий	https://gitflic.ru/project/alekseinedoria/trivil-0
Ворчалки о программировании	http://алексейнедоря.рф/

Зачем делаю

Интенсивное программирование

Языки выходного дня:
L1: динамический язык
L2: статика + сборка мусора
+ компоненты
L3: статика + ARC (Swift)
L4: статика + системный (Rust)

Зачем рассказываю

Полигон для студентов:
- простота языка
- современный вид
- современные типы
- простота компилятора
- открытая лицензия

Тривиль

- язык для разработки компилятора и экосистемы
- современный, надежный, удобный
- простой и понятный
- русскоязычный
- минимально достаточный

Привет, Тривиль!

Тривиль - это (тривиальный) модульный язык с явным экспортом/импортом, с поддержкой ООП и сборкой мусора.

```
модуль привет

импорт "стд::вывод"

вход {
    вывод.ф("Привет, Тривиль!\n")
}
```

```
Привет, Тривиль!
```

```
модуль факториал

импорт "стд::вывод"

фн Факториал(n: Цел64): Цел64 {
    надо n > 1 иначе вернуть 1
    вернуть n * Факториал(n - 1)
}

вход {
    пусть № = 5
    вывод.ф("%v! = %v\n",  №, Факториал(№))
}
```

```
5! = 120
```

Компилятор

Классическая схема:

- AST
- Лексер
- Парсер
- Семантика
- Генерация (C99)

Размер в строках:

- Компилятор: 9846 (Go)
- Runtime: 930 (C)
- Библиотеки (5 шт): 561 (Тривиль)

Описание языка

45 страниц (PDF, latex)

Основные конструкции языка

Типы

- Байт, Цел64, Слово64
- Вещ64
- Лог
- Символ
- Строка, Строка8
- вектор: []Т
- класс: **класс** (база) {}
- может быть: **мб** Т

Описания

- **тип** Т = *тип*
- **конст** к (: Т)? = *знач*
  конст к = 1
  конст к: Байт = 1
- **пусть** п(: Т)? (= | :=) *знач*
  пусть № = 1 // *val*
  пусть №: Байт := 1 // *var*
- функции и методы

Операторы

- :=, ++, --
- **если** *усл* {} **иначе** {}
- **надо** *усл* **иначе** (завер | {})
- **когда** - оператор выбора
- **пока** *усл* {}
- **прервать**
- **вернуть** *знач*?
- **авария**(“описание”)

Функции, методы

из библиотеки “вывод”:
фн ф*(формат: Строка, список: ...*) {}
^ вариативный, полиморфный
из библиотеки “строки”:
тип Сборщик* = класс ...
фн (сб: Сборщик) добавить строку*(ст: Строка) {}

Выражения

- +, -, *, /, %
- =, #, <, <=, >, >=
- лог: &, |, ~ (not)
- бит: :&, :|, :\ (xor), :~, <<, >>
- (:, ^
- конструкторы

Пример: Сборщик строки (string builder)

```
1  модуль строки
2  осторожно
3
4  тип Байты = []Байт
5
6  тип Сборщик* = класс {
7      байты = Байты[]
8      число-символов := 0
9  }
10
11 фн (сб: Сборщик) добавить строку*(ст: Строка) {
12     пусть ст8 = ст(:осторожно Строка8)
13     сб.число-символов := сб.число-символов + длина(ст)
14     сб.байты.добавить(ст8...)
15 }
16
17 фн (сб: Сборщик) строка*(): Строка {
18     вернуть сб.байты(:Строка)
19 }
```

разрешает “осторожные” операции

тип: вектор байтов

‘*’ - экспорт
обязательная инициализаций полей
  байты - конструктор пустого вектора

идентификатор с пробелом
конверсия к байтовой строке (RO)

‘добавить’ - встроенный метод:
(вектор: []T) добавить(x: ...T)

преобразование байтов в строку

Пример: Форматная строка (sprintf)

```
1  модуль строки
2
3  тип Символы = []Символ
4
5  тип Разборщик = класс {
6      сб: Сборщик = позже
7      формат: Символы = позже
8      …
9  }
10 фн (сб: Сборщик) ф*(фс: Строка, аргументы: ...*) {
11     пусть р = Разборщик{сб: сб, формат: фс(:Символы) }
12     ...
13     пока р.следующий() {
14         надо №-арг < длина(аргументы)
15         иначе авария("не достаточно аргументов")
16         // обработка аргумента
17     }
18     ...
19 }
```

Конструктор - единственный способ создать вектор или экземпляр класса

- пусть Три = Символы[‘Т’, ‘р’, ‘и’]
- пусть ч = Человека{имя: “Вася”}

Поздняя инициализация (#6, 7):

- все такие поля должны быть указаны в конструкторе класса (#11)

Оператор **надо**

- **проверить** - Вир
- **guard** - Swift

Что дальше?

Что еще интересное?

- обобщенные модули
- безопасность ссылок (мб T)
- полиморфные параметры
- подход к русскоязычному языку

Ближайшие планы

Раскрутка компилятора (bootstrap), для этого:

- необходимый набор библиотек
- система модульного тестирования
- доработка runtime
- доработка языка
  - выбор по типу
  - foreach?
  - доработка конверсии?

Дальше

- проектирование L2 языка с компонентами

Студенческие задачи и/или вклад в ЯВД

- Дизайн языков
  - L1, L2, L3, L4
  - добавление конструкций
- Компилятор
  - оптимизации
  - генерация кода
  - IR
- Среда исполнения
  - управление памятью
- Библиотеки
  - минимальный набор
  - далее везде
- Интеграция
  - IDE
  - системы тестирования
- Бенчмарки