Язык программирования Тривиль

Алексей Недоря

12.02.2023

Содержание

1. Назначение

Язык программирования Тривиль разработан в рамках работы на семейством языков программирования Языки выходного дня (ЯВД) проекта Интенсивное программирования.

Тривиль является нулевым языком семейства ЯВД, предназначенным для реализации компиляторов и экосистемы других языков семейства. В рамках классификации языков, принятом в проекте Интенсивное программирования, это язык L2.

Основными требованиями к языку при разработке были поставлены

- Язык должен быть минимально достаточным для удобной разработки компиляторов. Требование это во многом субъективно, так как компиляторы можно писать существенно по разному.
- Язык должен быть русскоязычным и с синтаксисом минимизирующим переключение на латиницу в процессе разработки программ.

Название языка происходит от слова "тривиальный что означает, что при разработке языка практически везде использовались решения, проверенные в других современных языках программирования, в первую очередь "донорами"являются Go, Swift, Kotlin и Oberon.

Несмотря на узкую направленность на разработку компиляторов, Тривиль является языком программирования общего назначения, пригодным для решения широкого круга задач.

Язык (и экосистема) обладает существенными предпосылками для использование его в качестве учебного языка для обучения студентов разработке компиляторов, библиотек, средств разработки, алгоритмов оптимизации и так далее, в первую очередь это:

- Простота языка
- Современный вид и набор конструкций языка
- Простота компилятора
- Открытая лицензия.

2. Обзор языка

Тривиль - это модульный язык с явным экспортом и импортом, автоматическим управлением памятью (сборка мусора), с поддержкой ООП.

Программа на языке Тривиль состоит из модулей (единиц компиляции), исходный текст каждого модуля расположен в одном или нескольких исходных файлов.

Пример программы:

```
1  модуль х
2
3  импорт "стд/вывод"
4
5  вход {
6      вывод.ф("Привет!\n")
7  }
```

Для описания языка используется EBNF в формате, близком к формату ANTLR4.
Операции:

()	группировка
X*	повторение 0 и более раз
X+	повторение 1 и более раз
X?	опциональность X (0 или 1 раз)
X | Y	X или Y

Пример:

```
Список-операторов: Оператор (Разделитель Оператор)*
```

3. Лексика

Исходный текст есть последовательность лексем: идентификаторов, ключевых слов, литералов, знаков операций и знаков препинания. Каждая лексема состоит из последовательности Unicode символов (unicode code point) в кодировке UTF-8.

Пробелы (U+0020), символы табуляции (U+0009) и символы завершения строки (U+000D, U+000A) разделяют лексемы, и, игнорируются, кроме следующих случаев:

- Символы завершения строк могут использоваться как разделители синтаксических конструкций (§3.2).
- Пробелы являются значащими символами в идентификаторах, состоящих из нескольких слов (§3.3).
- Пробелы являются значащими в строковых и символьных литералах (§3.3).

Несколько таких разделителей трактуются, как один.
Исходный текст может содержать *комментарии*.

3.1. Комментарии

Есть две формы комментариев:

- Строчный комментарий начинается с последовательности символов '//' и заканчивается в конце строки.
- Блочный комментарий начинается с последовательности символов '/*' и заканчивается последовательностью символов '*/'. Блочные комментарии могут быть вложенные.

```
Комментарий
    : '//' (любой символ, кроме завершения строки)*
    | '/*'  ( любой символ)* '*/
```

3.2. Разделители синтаксических конструкций

Некоторые синтаксические правила используют нетерминал *Разделитель* для разделения двух подряд идущих синтаксических конструкций, например:

```
Список-операторов: Оператор (Разделитель Оператор)*
```

В качестве разделителя может использоваться символ ';' или символ завершения строки.

```
Разделитель: ';'  | символ-завершения-строки
```

Пример:

```
1  а := 1; б := 2
2  в := 1
```

В строке 1 операторы разделены символом ';', а оператор в строке 2 отделен от операторов строки 1 символом завершения строки.

Ошибка компиляции - нет разделителя:

```
1  а := 1 б := 2
```

3.3. Идентификаторы

Идентификатор - это последовательность *Слов*, разделенных пробелами или символами дефис '-' с опционально завершающим знаком препинания:

Каждое слово состоит из *Букв* и *Цифр*, и начинается с Буквы. Буквой считается любой Unicode символ, имеющий признак *Letter*, и, дополнительно, символы '№' и '_'.

```
Идентификатор: Слово (' ' | '-') Слово)* Знак-препинания?
Слово: Буква (Буква | Цифра)*
Буква: Unicode-letter | '_' | '№'
Цифра: '0' .. '9'
Знак-препинания: '?' | '!'
```

Примеры идентификаторов:

```
1  буква
2  буква-или-цифра
3  №-символа
4  Цифра?
5  Пора паниковать!
```

3.4. Ключевые слова

Следующие ключевые слова зарезервированы и не могут быть использованы, как идентификаторы:

авария	есть	когда	надо	прервать
вернуть	иначе	конст	осторожно	пусть
вход	импорт	мб	пока	тип
если	класс	модуль	позже	фн

3.5. Знаки операций и знаки препинания

Следующие последовательности символов обозначают знаки операций и знаки препинания:

```
+    -    *    /    %
&    |    ~
=    #    <    <=   >    >=
:=   ++   --
```

```
(   )   [   ]   {   }
(:  .   ^   ,   :   ;
```

3.6. Целочисленные литералы

```
Целочисленный-литерал: Цифра+  | '0x' Цифра16+
Цифра16: '0'..'9' | 'a'..'f' | 'A'..'F'
```

3.7. Вещественные литералы

В текущей реализации есть только одна форма записи вещественных литералов, без экспоненты.

```
Вещественный-литерал: Цифра+ '.' Цифра*
```

3.8. Строковые литералы

Строковый литерал - это последовательность символов, заключенные в двойные кавычки. Строковый литерал может содержать символы, закодированные с помощью escape-последовательности, которая начинается с символа `'\'`.

```
Строковый литерал
    : '"'
    (~('"' | '\\' | '\n' | '\r' | '\t') | Escape)*
    '"'
Escape
    : '\\'
    ( 'u' Цифра16 Цифра16 Цифра16 Цифра16
    | 'n' | 'r' | 't'
    | '"'
    | "'"
    )
```

3.9. Символьные литералы

Символьный литерал задает значение для Unicode code point, - это последовательность символов, заключенные в двойные кавычки. Он записывается как один или несколько символов, заключенных в одинарные кавычки. Символьный литерал может быть закодирован с помощью escape-последовательности, которая начинается с символа `'\'`.

```
Символьный литерал
    : "'"
    ~("'" | '\\' | '\n' | '\r' | '\t') | escape_value)
    "'"
```

4. Описания и области действия

Каждый идентификатор, встречающийся в программе, должен быть описан, если только это не предопределенный идентификатор (§4.1). Идентификатор может быть описан как тип, константа, переменная, функция или как поле класса.

```
Описание
        : Описание-типов
        | Описание-констант
        | Описание-переменных
        | Описание-функций
```

Описанный идентификатор используется для ссылки на связанный объект, в тех частях программы, которые попадают в *область действия* описания. Идентификатор не может обозначать более одного объекта в пределах заданной области действия. Область действия может содержать внутри себя другие области действия, в которых идентификатор может быть переопределен.

Область действия, которая содержит в себе все исходные тексты на языке Тривиль называется *Универсум*.

Области видимости:

- Областью действия предопределенного идентификатора является Универсум
- Областью действия идентификатора, описанного на верхнем уровне (вне какой-либо функции), является весь модуль (§8).
- Областью действия имени импортируемого модуля является файл (часть модуля), содержащего импорт (§8.1).
- Областью действия идентификатора, обозначающего параметр функции, является тело функции (§4.8).
- Областью действия идентификатора, описанного в теле функции (§4.8) или теле входа (§8.2), является часть *блока* (§6.1), в котором описан идентификатор, от точки завершения описания и до завершения этого блока.

В описании сразу за идентификатором может следовать признак экспорта '*', указывающий, что идентификатор *экспортирован* и может использоваться в другом модуле, *импортирующем* данный (§8.1).

```
Идент-оп:  Идентификатор '*'?
```

4.1. Предопределенные идентификаторы

Следующие идентификаторы неявно описаны в области действия *Универсум*.

Типы (§4.2):

```
Байт Цел64 Слово64 Вещ64 Лог Символ Строка
```

Константы типа Лог (§4.2):

```
ложь истина
```

Литерал nullable (§**??**):

```
пусто
```

Стандартные функции (§7):

```
длина тег нечто
```

Кроме того, для векторных типов определен набор встроенных методов (§7.1).

4.2. Предопределенные типы

Следующие типы обозначаются предопределенными идентификаторами, значениями данных типов являются:

Тип	**Множество значений**
Байт	множество целых числа от 0 до 255
Цел64	множество всех 64-битных знаковых целых
Слово64	множество всех 64-битных беззнаковых целых
Вещ64	множество всех 64-разрядных чисел с плавающей запятой стандарта IEEE-754
Лог	константы `ложь` и `истина`
Символ	множество всех Unicode символов
Строка	множество всех строковых литералов.

Операции над значениями этих типов определены в (§5.2).

4.3. Указание типа

Тривиль является языком со статической типизацией, что означает, что тип любого объекта языка явно или неявно указывается во время описания объекта. Неявное указание типа может быть использовано в описании констант (§4.4), переменных (§4.4) и полей класса ().

Для явного указании типа используется имя типа, перед которым может стоять ключевое слово мб (*может быть*).

```
Указ-типа: 'мб'? Квалидент
Квалидент: Идентификатор ('.' Идентификатор)?
```

Множество значений объекта с типом мб T состоит из значения, обозначенного предопределенным идентификатором пусто и значений типа T. Тип такого объекта называется *может быть T* (§4.7). В англоязычных языках программирования используется термин *nullable type*.

4.4. Описание констант

Описание константы связывает идентификатор с постоянным значением. Значение константы может быть задано явно или неявно, в случае группового описания констант.

```
Описание-констант: 'конст' (Константа | Группа-констант)
Константа: Идент-оп (':' Указ-типа)? '=' Выражение
```

Если тип константы не указан, то он устанавливается равным типу выражения (§5). Выражение для константы должно вычисляться во время компиляции (§5.3).

```
конст к1: Цел64 = 1 // тип Цел64
конст к2: Байт = 2 // тип Байт
конст к3 = 3 // тип Цел64
конст к4 = "Привет" // тип Строка
```

Групповое описание констант позволяет опускать тип и выражение для всех констант, кроме первой константы в группе и указать признак экспорт для всех констант группы.

```
Группа-констант:
    '*'? '('
    Константа (Разделитель След-константа)*
    ')'
След-константа: Идент-оп (':' Указ-типа)? '=' Выражение)?
```

Пример группового экспорта:

```
конст *(
    Счетчик = 1
    Имя = "Вася"
)
```

Пример неявного задания значения для констант:

```
конст (
    / / операции
    ПЛЮС = 1    // тип Цел64, значение = 1
    МИНУС        // тип Цел64, значение = 2
    ОСТАТОК    // тип Цел64, значение = 3
    // ключевые слова
    ЕСЛИ = 21    // тип Цел64, значение = 21
    ИНАЧЕ         // тип Цел64, значение = 22
    ПОКА           // тип Цел64, значение = 23
)
```

4.5. Описание переменных

4.6. Описание типов

4.7. Может быть типы

4.8. Описание функций

5. Выражения

5.1. Операнды

5.2. Операции

5.3. Константные выражения

6. Операторы

6.1. Блоки

7. Стандартные функции

7.1. Встроенные методы для векторов

8. Модули

Заголовок модуля не является описанием, имя модуля не принадлежит никакой области действия. Его цель - идентифицировать файлы, принадлежащие одному и тому же модулю.

8.1. Импорт

Ошибка компиляции, если знак экспорта указан для идентификатора, описанного не на уровне модуля.

8.2. Вход