МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МАТЕРИАЛЫ

VII Международной молодежной научной конференции

«МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ, ТЕХНИЧЕСКИХ И ЭКОНОМИЧЕСКИХ СИСТЕМ»

Томск, 23–25 мая 2019 г.

Под общей редакцией
кандидата технических наук И.С. Шмырина

Томск
Издательский Дом Томского государственного университета
2019

результатах, представленных в таблице 1 и таблице 2, а также на рис. 3 и рис. 4, можно рекомендовать для фалов .txt алгоритм сжатия PPMd для наилучшей компрессии, но не лучшей скорости, и метод LZMA2, как лидер среди сжатия данных с таблицами, документами и презентациями. Как самый быстрый алгоритм, чаще всего показывает себя Deflate.

ЛИТЕРАТУРА

1. Академик [Электронный ресурс] : Дельта-кодирование – 2014. – URL: https://dic.academic.ru/dic.nsf/ruwiki/41976.
2. Habr [Электронный ресурс] : Алгоритмы сжатия данных без потерь, часть 2 – 2014. – URL: https://habr.com/post/235553/.
3. Habr [Электронный ресурс] : Простейшие алгоритмы сжатия: RLE и LZ77 – 2012. – URL: https://habr.com/post/141827/.
4. Wikipedia [Электронный ресурс] : Алгоритм сжатия PPM – 2018. – URL: https://ru.wikipedia.org/wiki/Алгоритм_сжатия_PPM.

ВИРТУАЛЬНАЯ МАШИНА EMERAVM

Е.П. Чалых, С.И. Самохина
Томский государственный университет

Введение

На данный момент в мире существует большое число языков программирования, каждый из них создан с определённой целью. Изначальной целью проекта было создание мощного и быстрого языка программирования с поддержкой математических вычислений, таких как дифференцирование, интегрирование, работа с матрицами и другие. Но первая проблема нового языка возникла при выборе его типа: интерпретируемый или компилируемый. Очевидный минус первого типа заключается в том, что процесс выполнения кода будет довольно долгим, особенно на очень больших файлах исходного кода. Но главное достоинство – интерпретатор языка можно сделать кроссплатформенным. Компилируемый язык программирования будет работать гораздо быстрей, т.к. из исходного кода мы получим исполняемый файл. А минус такого языка, естественно, платформозависимость.

Очевидно, для получения кроссплатформенного и быстрого языка, нужно использовать промежуточное состояние исходного кода и исполняемого файла. Такое состояние называется байт-кодом[1]. Байт-код – это последовательность байтов, каждый из которых характеризует конкретную инструкцию выполнения. Чтобы сформировать байт-код, нужен определенный транслятор, а для выполнения полученного байт-кода – определенный интерпретатор, называемый виртуальной машиной[2].

1. Постановка задачи

Первый этап создания языка программирования – это создание работоспособной, желательно кроссплатформенной, виртуальной машины. И поскольку язык будет использоваться в математических целях, виртуальная машина должна поддерживать большие математические вычисления.

Название проекта происходит от названия языка и сокращения от virtual machine – EmeraVM, или сокращенно EVM.

Кроссплатформенность можно достичь с помощью другого кроссплатформенного языка, в данном случае используется язык Kotlin[3], работающий на JVM[4] – виртуальной машине Java. Соответственно, EVM будет работать на любой операционной системе, где установлена Java. Инструментом создания EVM является мощная интегрированная среда разработки IntelliJ IDEA[5].

2. Архитектура EVM

Реализация EVM разделена на 3 основных модуля: lang, vm, asm (рис. 1).

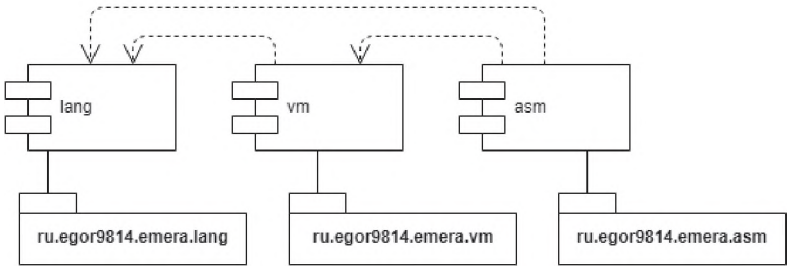

Рис. 1. Модули EVM и их зависимость

Модуль «lang» является главным, т. к. он отвечает за работу с типами данных. Рассмотрим его структуру (рис. 2).

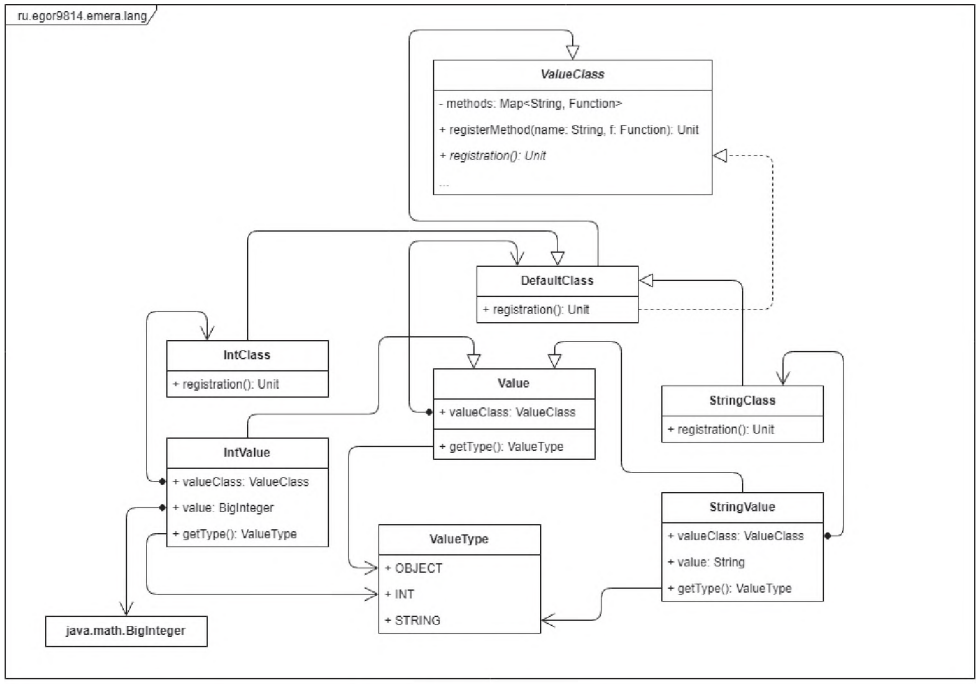

Рис. 2. Классы модуля «lang»

Класс «ValueType» определяет доступные типы данных: STRING – строка, INT – целое число, OBJECT – объект.

Класс «ValueClass» определяет доступные методы для конкретного класса. Класс «DefaultClass» регистрирует метод «toString()», который представляет типе в виде строки. Класс «StringClass» определяет методы работы со строками. Класс «IntClass» определяет методы работы с целыми числами.

Класс «Value» определяет тип данных и способ работы с ним. Класс «IntValue» содержит поле типа «BigInteger», что позволяет работать с большими целыми числами. Класс «StringValue» содержит поле типа «String», что позволяет работать со строками.

В данном модуле также есть класс для работы с объектами типа «Value» - EmeraEnvironment (рис. 3).

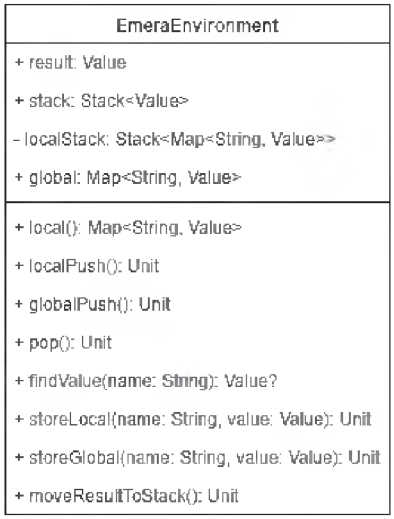

Рис. 3. Класс «EmeraEnvironment»

Этот класс позволяет управлять стеком значений, сохранять глобально или локально объекты по ключу, а также загружать в локальный стек. По умолчанию результат выполнения любой операции заносится в регистр результата «result», и если есть необходимость его использовать, то с помощью метода «moveResultToStack()» можно поместить результат в локальный стек.

Модуль «vm» (рис. 4, 5) позволяет прочитать байт-код и выполнить его. Интерфейс «IInstruction» предоставляет абстракцию, которую реализует конкретная инструкция, в зависимости от кода операции класса «ByteCodeV1». Класс «InstructionEnvironment» позволяет контролировать процесс добавления и выполнения инструкций. Класс «Interpreter» отвечает за чтение байт-кода и составление списка инструкций. Класс «ByteCodeLauncher» использует экземпляр класса «InstructionEnvironment», полученный из интерпретатора «Interpreter», для дальнейшего выполнения инструкций в методе «execute()». Если результат вызова этого метода окажется ложным (False), то с помощью метода «getError()» можно получить текст ошибки.

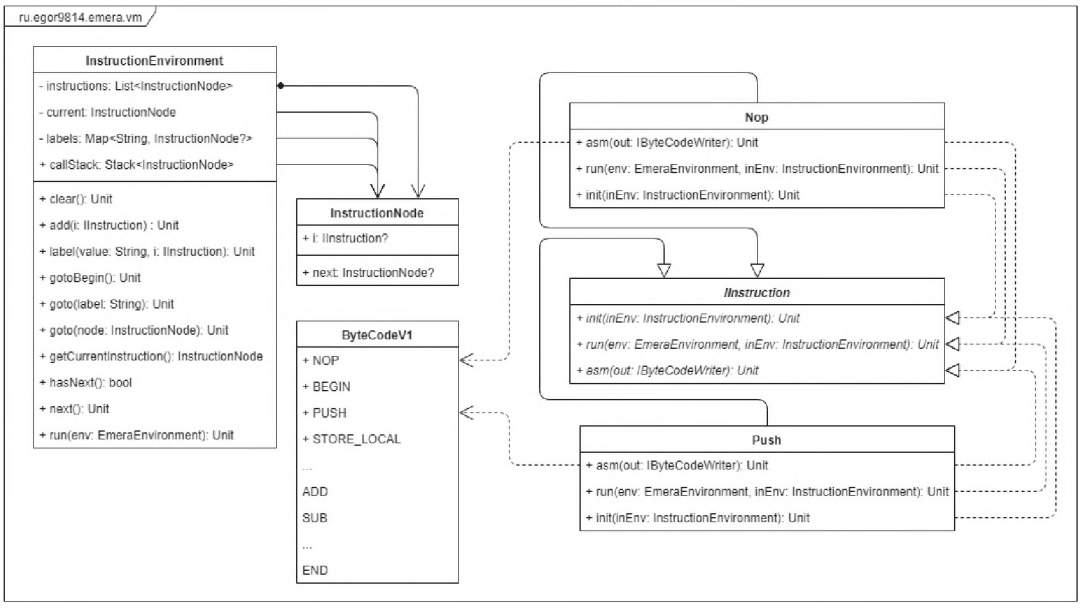

Рис. 4. Первая часть модуля «vm»

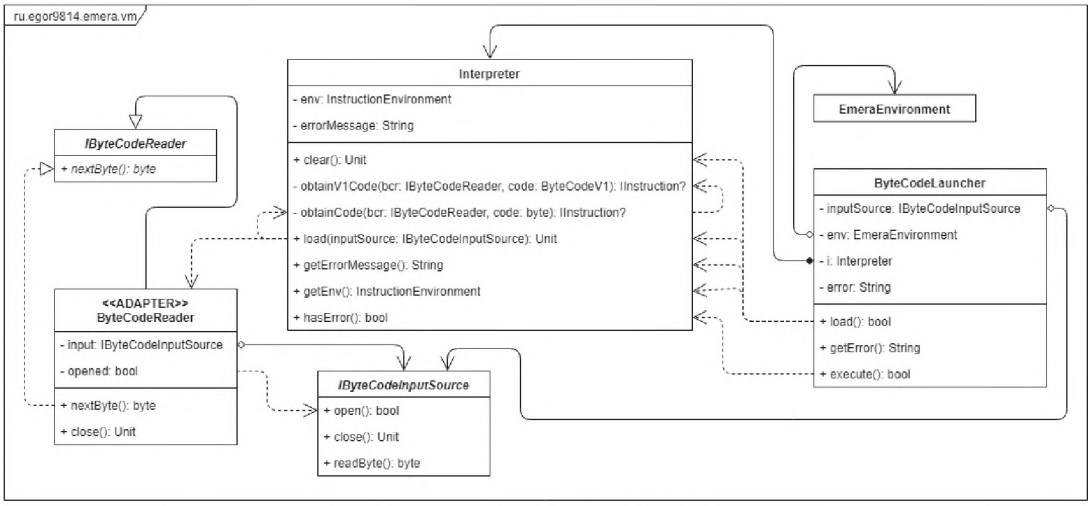

Рис. 5. Вторая часть модуля «vm»

На данный момент у нас есть способ запустить байт-код, но как сформировать его? Первый способ – это вручную прописывать байты в файле. Это очень неудобно, трудоёмко, и велика вероятность ошибки. Второй – использовать компилятор, который преобразует исходный код в байт-код. Это наилучший вариант, но языка, как и компилятора, ещё не существует. На этот случай был разработан модуль «asm» (рис. 6). Он позволяет преобразовать текстовые инструкции, понятные для человека, в байт-код. В качестве таких инструкций выступает низкоуровневый язык программирования похожий на Ассемблер (рис. 7).

Класс «Tokenizer» позволяет разбить одну строковую линю на специальные последовательности – токены (класс «Token»), а именно: ключевое слово, строка, число. Класс «Interpreter» проходит весь исходный код по строкам и с помощью «Tokenizer», определяет тип инструкции, после чего формирует список инструкций (класс «IInstruction»).

Класс «Assembler» – это генератор байт-кода. Первый метод «interpret()» анализирует исходный код с помощью класса «Interpreter». Если во время анализа произойдет ошибка, то результатом этой функции будет текст этой ошибки. Метод «asm()» проходит по каждой инструкции и вызывает метод «IInstruction:: asm(IByteCodeWriter)», который в свою очередь формирует и записывает фрагмент байт-кода в интерфейс вывода «IByteCodeWriter».

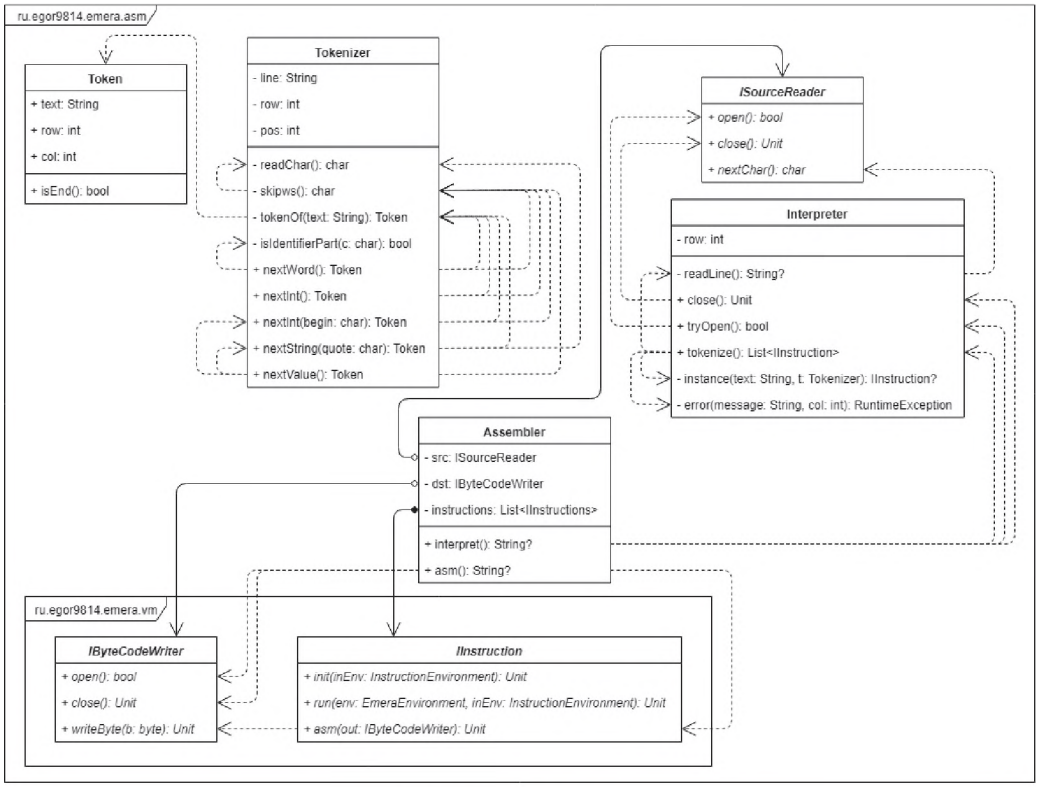

Рис. 6. Классы модуля «asm»

```
Open
 2 push 10
 3 global "a"
 4
 5 push 20
 6 global "b"
 7
 8 lbl __add
 9   load "b"
10   load "a"
11   add
12   mrs
13   global "add"
14
15 lbl __sub
16   load "b"
17   load "a"
18   sub
19   mrs
20   global "sub"
21
22 lbl __mul
23   load "b"
24   load "a"
25   mul
26   mrs
27   global "mul"
28
29 lbl __div
30   load "b"
31   load "a"
32   div
33   mrs
34   global "div"
35
36 lbl __pow
37   load "b"
38   load "a"
39   pow
40   mrs
41   global "pow"
```

Рис. 7. Пример исходного кода для EVM

3. Способ запуска виртуальной машины

Запустить EVM на данный момент можно через терминал с помощью скрипта «emera.bat» на Windows, и «./emera» на MacOS или Linux (рис. 8).

Команда «./emera –help» позволит увидеть доступные параметры для запуска EVM.

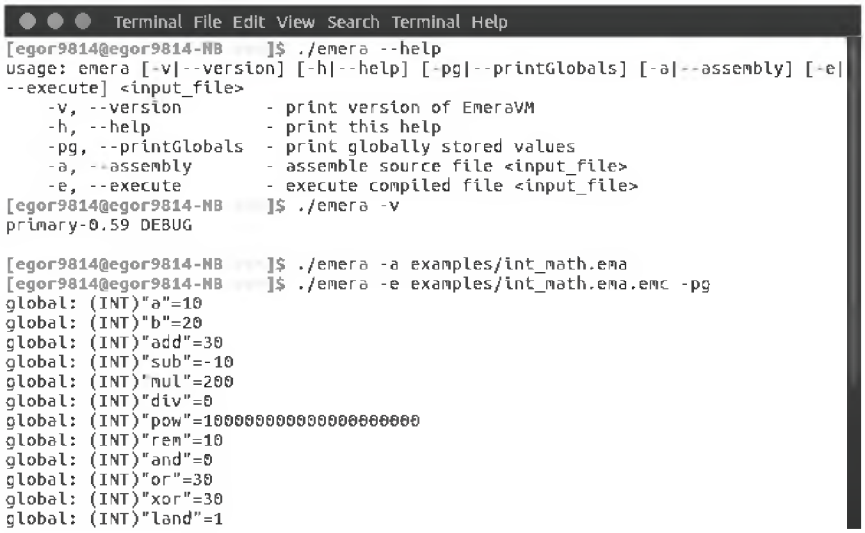

Рис. 8. Пример запуска EVM

Заключение

Данный проект был создан как компонент языка программирования «EmeraLang», в нём реализована поддержка больших целых чисел, работа со строками, условные и безусловные переходы и другое. Т. к. язык направлен на математические вычисления, то в виртуальную машину также будет добавлена поддержка больших чисел с плавающей точкой, комплексных чисел, матриц, а также булева алгебра.

ЛИТЕРАТУРА

1. Байт-код [Электронный ресурс]: сайт / Википедия – URL: https://ru.wikipedia.org/wiki/Байт-код (дата обращения: 21.05.2019).

2. Виртуальная машина [Электронный ресурс]: сайт / Википедия – URL: https://ru.wikipedia.org/wiki/Виртуальная_машина (дата обращения: 21.05.2019).

3. Язык программирования Kotlin [Электронный ресурс]: сайт / KotlinLang – URL: https://kotlinlang.org/ (дата обращения: 21.05.2019).

4. JVM [Электронный ресурс]: сайт / Википедия – URL: https://ru.wikipedia.org/wiki/Java_Virtual_Machine (дата обращения: 21.05.2019).

5. IntelliJ IDEA [Электронный ресурс]: сайт / JetBrains – URL: https://www.jetbrains.com/idea/, (дата обращения: 21.05.2019).

ВЛИЯНИЕ ВИБРАТО НА КАЧЕСТВО РАСПОЗНАВАНИЯ НОТ В ВОКАЛЬНОМ ИСПОЛНЕНИИ

А.Ю. Якимук

Томский государственный университет систем управления и радиоэлектроники

Введение

Применение программных средств в задачах обучения пользователей распространено во многих областях знаний. Точность работы таких программ крайне важна, поскольку ошибочно работающий алгоритм способен ввести в заблуждение ученика и сформировать компетенцию некорректно. Особенно сильно это проявляется в сфере речевых технологий. В случае, если пользователь неправильно выучит произношение слов при изучении языка или не сформирует музыкальный слух корректно, полученные в программе навыки окажутся бесполезными, а в некоторых ситуациях даже вредными. В данной работе рассматривается вопрос определения причин в получаемом проценте ошибок при идентификации спетой диктором ноты.