Создание собственного интерпретатора одного из диалектов языка программирования Lisp для написания пользовательских скриптов

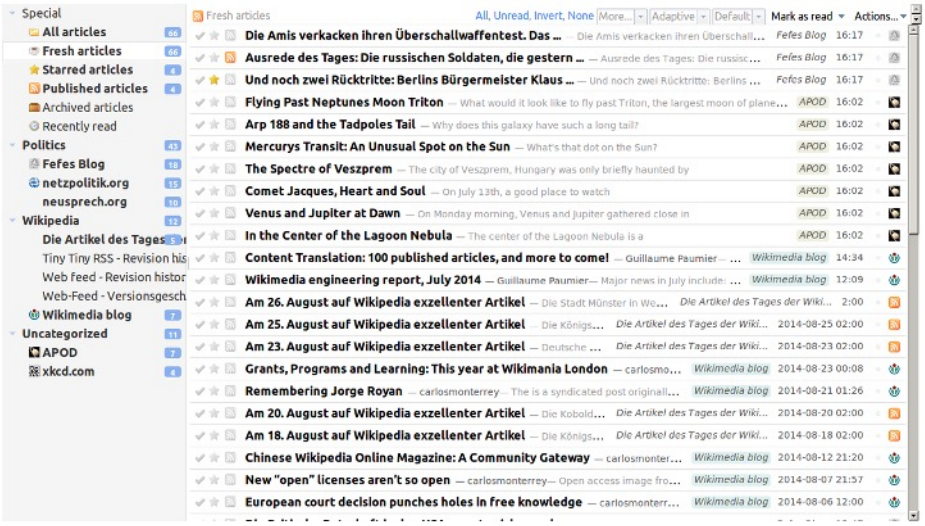
Выполнил: **Диникеев Аскар Маратович**

Город: **Уфа**

Преподаватели: **Гильдин Александр Григорьевич**
Винов Александр Сергеевич

Здравствуйте, меня зовут Диникеев Аскар. Тема моего проекта - Создание собственного интерпретатора одного из диалектов языка программирования Lisp для написания пользовательских скриптов.

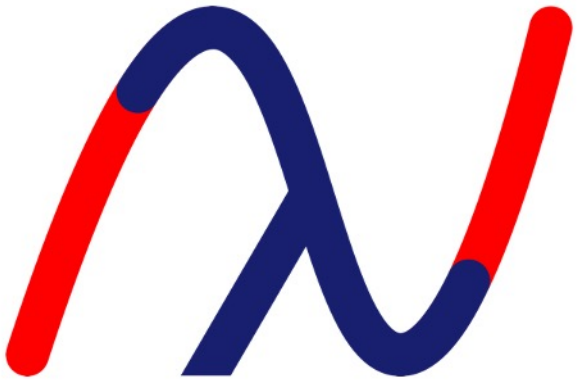
Я хочу рассказать, как мне пришла в голову идея, написать свой интерпретатор. Многие из нас любят читать новости. Задумываемся ли мы как работают новостные агрегаторы. Обычно они используют протоколы RSS и ATOM новостных сайтов. Сейчас на экране фрагмент RSS ленты новостного сайта. С этого сайта агрегатор легко возьмет новости.



RSS feed reader

А на этом слайде скриншот новостного агрегатора. Все удобно, все новости находятся в одном месте. Но многие сайты не поддерживают такие новостные форматы. Чтобы читать новости с этих сайтов, нужно заходить на каждый из них в отдельности.
Возникает противоречие: как например собирать новости с сайтов, не поддерживающих протоколы RSS и ATOM.

Создание приложения



Мое приложение позволяет читать любой сайт за счет написания пользовательских скриптов

4

Для решения данной проблемы я написал приложение которое позволяет читать любой сайт за счет написания пользовательских скриптов. А сами скрипты в моем приложении пишутся на языке Lisp. Этот язык всегда был мне интересен. Интерпретатор языка я тоже написал сам.

Моя работа состоит из двух разработок.

Первая - научная: я написал собственный интерпретатор диалекта языка Lisp.

Вторая - практическая я написал Android приложение, которое позволяет писать пользовательские скрипты и выполнять их на моем интерпретаторе.

Таким образом я с одной стороны решил задачу с чтением новостей с сайтов не поддерживающих протокол RSS, а с другой стороны у меня теперь есть свой собственный интерпретатор открывающий горизонты

для новых исследований.

Почему Lisp (Scheme)

- Язык очень прост и минималистичен. Особенно в плане синтаксиса.
- Язык поддерживает макросы.
- Большие возможности макросов в Lisp языках.

```
(defmacro (cond body)
  (fold nil
 (lambda (acc e)
  `(if ,(car e) ,(cdr e) ,acc))
    (reverse body)))

(cond (((= 1 2) (+ 1 2 3))
       ((= 1 1) (- 5 2))))

;; → 3
```

Пример создания макроса в моем языке

5

Но почему же я выбрал именно Lisp в качестве языка программирования?

Первая и самая главная причина это то, что все Lisp языки очень просты в плане синтаксиса.

Это позволяет мне больше сконцентрироваться на написании интерпретатора, а не на написании парсера.

Вторая причина это очень мощная система Lisp макросов, позволяющая продвинутому пользователю увеличивать возможности интерпретатора не меняя его исходного кода.

Все это достигается за счет гомоиконичности Lisp языков. Это свойство означает, что код и данные в языке взаимозаменяемы.

Мой диалект Lisp (Scheme)

- Поддержка процедур первого класса
- Поддержка рекурсии
- Закрытия (Closures)
- Цитирование и квазицитирование выражений (Пример: ‘exp `(1 ,a 3))
- Поддержка макросов

Итак я самостоятельно написал на языке программирования Kotlin свой диалект языка программирования Lisp «с нуля», не используя сторонние библиотеки и реализации.

Среди возможностей, поддерживаемых моим языком и интерпретатором есть:

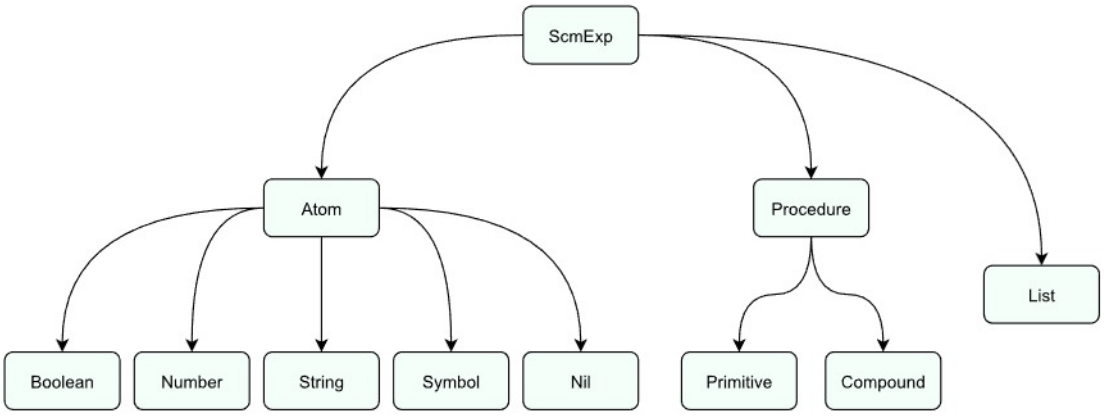
Поддержка процедур первого класса. Это означает, что процедура может быть переданна как аргумент.

Поддержка закрытий (Closures).

Поддержка цитирования и квазицитирования.

Поддержка макросов

Выражения в языке



7

На экране предствленна иерархия типов выражений языка.

Выражение может быть атомом, процедурой, или списком.

Хочу обратить ваше внимание на то, что вместо традиционных в Lisp языках пар в моем диалекте используются списки.

Структура интерпретатора

8

Процесс интерпретации делится на 4 стадии: токенизация, парсинг, расширение и выполнение.

На стадиях токенизации и парсинга интерпретатор сначала разделяет скрипт на токены,
а затем переводит эти токены в выражения.

Теперт про стадии Раскрытия и Выполнения.

Раскрытие

```
(define (a x y z) ... )   ⟶   (define a (lambda (x y z) ... ))
          (if test a)     ⟶   (if test a nil)
            `(1 ,a 2)     ⟶   (list 1 a 2)
(cond (((= 1 2) (+ 1 2 3))  ⟶   (if (= 1 2) (+ 1 2 3) (if (= 1 1) (- 5 2) nil))
      ((= 1 1) (- 5 2))))
```

```
(defmacro (cond body)
  (fold nil
 (lambda (acc e)
  `(if ,(car e) ,(cdr e) ,acc))
    (reverse body)))

(cond (((= 1 2) (+ 1 2 3))
       ((= 1 1) (- 5 2))))

;; → 3
```

9

Перед выполнением выражений интерпретатор сначала «Раскрывает» некоторые специальные выражения.

Слева представлены изначальные выражения, которые не могут быть выполнены без раскрытия.
А справа представлены те же выражения, но уже после процесса раскрытия.

Также на этом этапе объявляются и выполняются все макросы в программе.

Выполнение

10

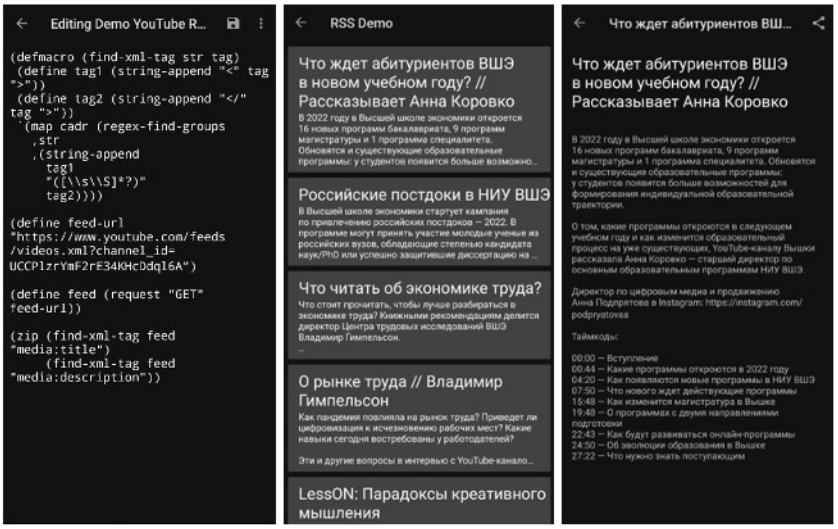
Следующий, после раскрытия, этап - это выполнение.

Процесс выполнения скрипта можно описать как взаимодействие двух функций: Eval и Apply.

Eval принимает выражение и окружение.
Она построена как разбор случаев в зависимости от синтаксического типа выражения.

Apply принимает процедуру и аргументы.
Она применяет процедуру к вычисленным аргументам составляющим тело функции.

Работа с пользовательскими скриптами

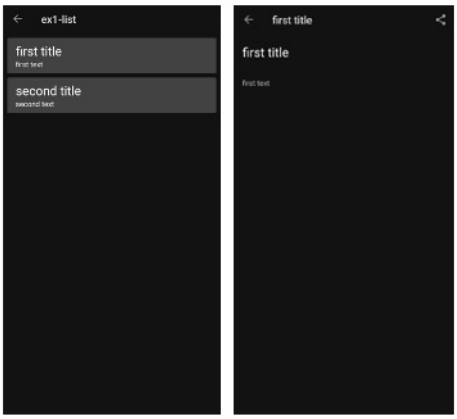


11

Теперь про вторую практическую часть.
Я написал приложение для Android на языке программирования Java.
В нем можно создавать и выполнять скрипты для моего интерпретатора.

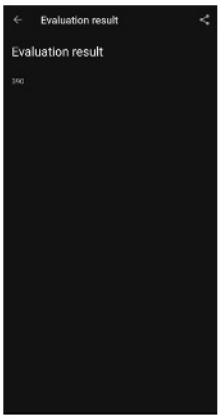
Работа с пользовательскими скриптами

```
(list (list “first title” “first text”)
      (list “second title” “second text”))
```



Скрипт вернул результат выполнения в виде списка пар объектов

390



Иначе

В результате своей работы скрипт возвращает некое значение. Есть два способа отображения этого значения. Они определяются автоматически и зависят от его формата.
Слева способ отображения в виде постов. Этот формат используется, если скрипт вернул результат выполнения в виде списка пар объектов. В ином случае выбирается формат изображенный справа.

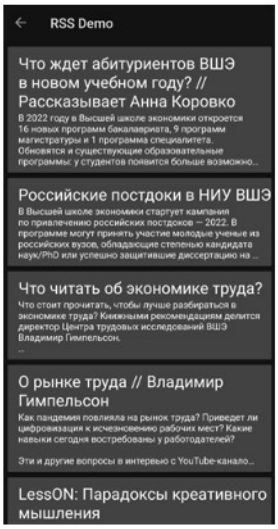
Пример пользовательского скрипта для приложения

```
(defmacro (find-xml-tag str tag)
 (define tag1 (string-append "<" tag ">"))
 (define tag2 (string-append "</" tag ">"))
 `(map cadr (regex-find-groups
   ,str
   ,(string-append
     tag1
     "([\\s\\S]*?)"
     tag2))))

(define feed-url
 "https://youtube.com/feeds/videos.xml?channel_id=UCCPlzrYmF2rE34KHcDdql6A")

(define feed (request "GET" feed-url))

(zip (find-xml-tag feed "media:title")
     (find-xml-tag feed "media:description"))
```



13

На этих слайдах я покажу примеры скриптов для моего приложения.

Этот скрипт позволяет читать публикацию новых видео на Youtube канале, указанном в переменной feed-url.
Данный скрипт получает данные с вебсайта в RSS формате.

Пример пользовательского скрипта для приложения

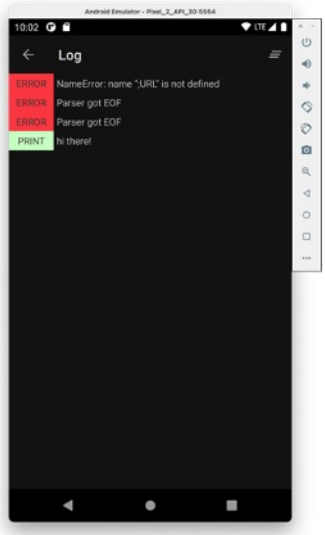
```
(define url "https://hse.ru/news")

(map cdr
  (regex-find-groups
  (request "GET" url)
  "alt=\"(.*?)\"(?:.*?) \"><p class=\"first_child last_child \">(.*?)<\\/p>"))
```

14

А этот скрипт не использует ни протокол RSS ни Atom. Вместо этого он получает всю веб страницу и выделяет из нее заголовки и тексты постов. При помощи этого скрипта можно получить новостную ленту Высшей Школы Экономики.

Отладка пользовательских скриптов



15

Для упрощения отладки пользовательских скриптов, ошибки в выполнении скриптов и сообщения можно смотреть в самом приложении

Вывод

В ходе проведенной работы был написан интерпретатор собственного диалекта языка Lisp, написано приложение на Android, позволяющее просматривать новости. Созданный интерпретатор позволяет проводить дальнейшие исследования интерпретируемых языков программирования.

16

В ходе проведенной работы был написан интерпретатор собственного диалекта языка Lisp, написано приложение на Android, позволяющее просматривать новости. Созданный интерпретатор позволяет проводить дальнейшие исследования интерпретируемых языков программирования.

Спасибо за внимание