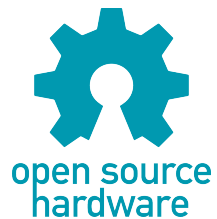




Скрипт-язык bl

© Дмитрий Понятов <dponyaton@gmail.com>



Оглавление

Цели	1
Установка	1
1 Учебник	3
Запуск	3
Комментарии	4
AST-тип	4
Атомы (литералы, скалярные объекты)	5
Списки	5
1.1 Заимствования из языка Forth	6
1.1.1 Стек данных	6
1.1.2 Forth-функции на стеке данных	8
2 Библиотеки	10
3 Синтаксис	11
4 Реализация	12

5	Мета-модель	13
	Литература	13

Цели

- универсальный язык описания входных данных для расчетных программ и файлов конфигурации
- язык-шаблонизатор для генерации выходных файлов на любых других ЯП
 - генерация проектов по набору шаблонных файлов
 - мульти-платформенная разработка ПО
 - формирование файлов конфигурации и управления работой кластерного ПО
- средство программирования пользовательского GUI (реализация APM)

Установка

- GitHub: <https://github.com/ponyatov/script.git>
- Ручная загрузка:
 - win32 .exe
<https://github.com/ponyatov/script/raw/master/bI.exe>
 - руководство .pdf
https://github.com/ponyatov/script/raw/master/manual_ru/manual.pdf
 - тестовый bl-скрипт
<https://github.com/ponyatov/script/raw/master/bI.bI>
- Сборка из исходных файлов:
 - Windows (win32):

- * пакет компилятора GNU GCC/G++
MinGW <http://www.mingw.org/download/installer?>
- * Система управления версиями Git
[git-scm https://git-scm.com/downloads](https://git-scm.com/downloads)
- * Текстовый редактор Vim
[gvim ftp://ftp.vim.org/pub/vim/pc/gvim74.exe](ftp://ftp.vim.org/pub/vim/pc/gvim74.exe)

```
git clone -o gh https://github.com/ponyatov/script.git bI_script
cd bI_script
mingw32-make EXE=.exe RES=res.res
```

– Linux:

```
git clone -o gh https://github.com/ponyatov/script.git bI_script
cd bI_script
make EXE=.elf RES=
```

Глава 1

Учебник

Запуск

Ядро системы реализовано в виде единственного .exe файла¹. При простом запуске открывается интерактивная сессия, для загрузки скрипта из файла используйте команду

```
bI.exe < script.bI > script.blog
```

В Linux исполняемым может быть сделан *любой* файл, добавьте к своим скриптам первой строкой полный путь к bl.elf:

```
#!/home/user/bI_script/bI.elf
```

и выполните команду

```
chmod +x script.bI
```

¹ консольная программа

Комментарии

Строчные комментарии начинаются с символа `#`:

Создайте файл `script.bl`, запишите в него

```
#!/home/user/bI_script/bI.elf
# syntax sample with numbers, symbols, [] lists and stack ops
# John McCarthy A Micro-Manual for Lisp - not the whole Truth.
```

и выполните через `bl`-интерпретатор:

```
bI.exe < script.bl
```

AST-тип

Язык `bl` реализован на базе операций с элементами данных, представленных в виде **символьного типа** AST:

```
class AST:sym
```

<code>string:tag</code>	тип данных (тэг класса)
<code>string:value</code>	значение
<code>sym:nest[]</code>	список вложенных элементов данных
<code>string:dump()</code>	вывод элемента данных в текстовом виде в виде дерева
<code>string:tagval()</code>	строковое представление только основной части <тэг:значение>
<code>sym:eval()</code>	вычисление (evaluate) элемента данных

Атомы (литералы, скалярные объекты)

- числа
 - целые:
`-01 -0 00 +0 +02222` → `<int:-1> <int:0> <int:0> <int:0> <int:2222>`
 - с плавающей точкой в простой и экспоненциальной форме:
`+02.30 -04E+05` → `<num:2.3> <num:-400000>`
 - шестнадцатеричные и бинарные машинные константы:
`0x12AF 0b1101` → `<hex:0x12AF> <bin:0b1101>`
- 'строки'
- символы: в простейшем виде просто задает уникальное имя
`symbol` → `<sym:symbol>`

Списки

Список — **компози́тная** структура данных, позволяющая рассматривать произвольный набор данных как один объект. Списки могут быть вложенными.

Добавьте в `script.bl` или выполните в интерактивной сессии² код:

```
# пример синтаксиса с числами, символами, и вложенными списками:  
[ [-01 -0 00 +0 +02222] [ 0x12AF 0b1101 ] [ +02.30 -04E+05 ] ]  
'string'  
symbol
```

² запустив `bl.exe` без параметров


```
? # вывести дамп стека
~ # dropall: полностью очистить стек данных
? # еще раз вывести стек
```

1.1 Заимствования из языка Forth

В языке Forth программа представляет собой последовательность **слов**-команд, разделенных пробелами³. Когда интерпретатор Forth встречается **слово**, он ищет его в **словаре**, и если оно найдено — исполняет.

Если слово не найдено, Forth пытается его прочесть как целое число, при успехе кладет на стек, иначе выводит ошибку. Плавающей точки нет вообще.

Строки обрабатываются особо: слово "."⁴ читает входной поток, пока не встретит символ "\n", и кладет на стек адрес строки.

Язык bI расширяет синтаксис Forth'a инфиксными выражениями, встроенными типами данных, высокоуровневой библиотекой символьных вычислений и ООП⁵.

1.1.1 Стек данных

Каждое выражение языка bI после вычисления кладется на стек данных.

Стек — структура данных, поддерживающая операции:

³ сам пробел, табуляция и концы строк DOS/UNIX

⁴ с обязательным за ним пробелом

⁵ рассматривая все данные как обобщенный AST-тип1

push(объект)	(A B C – A B C obj)	поместить объект на вершину стека
pop()→объект	(A B C obj – A B C)	взять объект с вершины
стековые функции 1.1.2		выполняют вычисления на стеке

Для записи состояния стека используется строчная Forth-нотация: элементы записываются слева направо, вершина — самый правый элемент. Для встроенных Forth-функций 1.1.2 используется нотация вида A B C -- A C B, в левой части состояние стека до выполнения функции, справа — после того, как функция выполнила **вычисления на стеке данных**.

? вывести стек (вершина внизу)
 ~ очистить стек

Команда ? выводит стек в полной текстовой форме в виде дерева, для составных объектов включаются все вложенные элементы. Стек выводится сверху вниз, вершина стека — самый нижний элемент.

При выполнении примера 1 будет выведено (список списков чисел, строка и символ):

```
===== stack =====
<[:]>
  <[:]>
    <int:-1>
    <int:0>
    <int:0>
    <int:0>
    <int:2222>
  <[:]>
    <hex:0x12AF>
```

```
<bin:0b1101>
<[:]>
<num:2.3>
<num:-400000>
<str:'string'>
<sym:symbol>
=====

===== stack =====
=====
```

1.1.2 Forth-функции на стеке данных

Стек данных и Forth-функции были введены в язык bl как упрощение, позволяющее реализовать лямбды и определение функций.

Явное использование стека данных позволяет определять лямбды как простой список имен ранее определенных и встроенных функций, выполняющих операции на стеке.

Это позволяет обойтись без подстановки параметров в традиционном стиле реализации лямбда-функций.

Forth-функции выполняют вычисления на стеке данных, явно манипулируя элементами на вершине:

push(объект)	(A B C – A B C obj)	поместить объект на вершину стека
pop()→объект	(A B C obj – A B C)	взять объект с вершины
?	аналог .S	вывести полный дамп стека
~	аналог DROPALL	очистить стек
print	аналог . (A B – A)	вычислить и вывести символьное представление
dup	(A B C – A B C C)	продублировать верхний элемент
drop	(A B C – A B)	убрать один элемент с вершины
swap	(A B C – A C B)	обмен 2х объектов на стеке
over	(A B C – A B C B)	вытащить копию -1-ого элемента на вершину

Глава 2

Библиотеки

Глава 3

Синтаксис

Глава 4

Реализация

Глава 5

Мета-модель

Литература

- [1] A micro-manual for LISP Implemented in C
<http://nakkaya.com/2010/08/24/a-micro-manual-for-lisp-implemented-in-c/>
- [2] John McCarthy
A Micro-Manual for Lisp — not the whole Truth