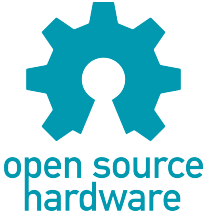
Азбука ARMатурщика

Скрипт-язык bl

© Дмитрий Понятов <dponyatov@gmail.com>



Оглавление

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 АРМ)

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/bI_manual/raw/master/manual.pdf`
  - тестовый Ы-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`
  * Текстовый редактор Vim
    gvim `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 единстеного .exe файла[1]. 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 bI.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.bI
```

AST-тип

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

class AST:sym

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

Скалярные типы (атомы, литералы)

- 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>

1.1 Композитные типы

1.1.1 Список

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

Добавьте в script.bl или выполните в интерактивной сессии[2] код:

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

[2] запустив bl.exe без параметров

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

1.1.2 Пара

Через пары вида `A:B` реализуются деревья, списки в Lisp-стиле, и описание ООП-структур.

1.2 Заимствования из языка Forth

В языке Forth программа представляет собой последовательность слов-команд, разделенных пробелами[3]. Когда интерпретатор Forth встречает слово, он ищет его в словаре, и если оно найдено — исполняет.

Если слово не найдно, Forth пытается его прочитать 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 `"`, и кладет на стек адрес строки.

Язык bl расширяет синтаксис Fortha инфиксными выражениями, встроенными типами данных, высокоуровневой библиотекой символьных вычислений и ООП[5].

1.2.1 Стек данных

Каждое выражение языка bl после вычисления кладется на стек данных.

---

[3] сам пробел, табуляция и концы строк DOS/UNIX

[4] с обязательным за ним пробелом

[5] рассматривая все данные как обобщенный AST-тип1

Стек — структура данных, поддерживающая операции:

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 функции1.2.2		выполняют вычисления на стеке

Для записи состояния стека используется строчная Forth-нотация: элементы записываются слева направо, вершина — самый правый элемент. Для встроенных Forth-функций1.2.2 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 вполнении примера 1.1.1 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 ==========
=========================
```

1.2.2 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 лябда-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

1.3 ООП

Глава 2

Библиотеки

2.1 Математика

2.1.1 Константы

сивольная	численная		
E	e	основание натурального логарифма	e
PI	pi	число	π
	c	скорость света в вакууме	c
	h	постоянная Планка	h
	G	гравитационная постоянная	G
	g	ускорение свободного падения	g
	e	элементарный электрический заряд	e
	k	постоянная Больцмана	k
	Na	постоянная Авогадро	N_a
E0	e0	(ди)электрическая постоянная	ε_0
U0	u0	магнитная постоянная	μ_0

2.1.2 Стандартные функции

sqrt	$x \to \sqrt{x}$	квадратный корень
sqrn	$x\ n \to \sqrt[n]{x}$	
exp	$x \to e^x$	
pow	$x\ y \to x^y$	
ln	$x \to \ln x$	натуральный логарифм
log10	$x \to \log_{10} x$	десятичный логарифм

2.1.3 Тригонометрия

sin	$x \to sin(x)$	синус
cos	$x \to cos(x)$	косинус
tan	$x \to tan(x)$	тангенс
ctg	$x \to ctg(x)$	котангенс
asin	$x \to arcsin(x)$	арксинус
acos	$x \to arccos(x)$	
atan	$x \to atctan(x)$	

2.2 Операции с файлами и каталогами

2.2.1 Файловые классы

```
class:fileio
fileio:dir
fileio:file
```

2.2.2 Типовые операции командной оболочки

mkdir str $\rightarrow$ dir создать каталог

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