Содержание

Введение

Анализ текущего состояния в разработке языков программирования показал, что существует необходимость в языке программирования, ориентированном на быстрой разработке сценариев (скриптов), первоначальном обучении программированию и исследовательском программировании.

Для этих целей, на наш взгляд, должен существовать функциональный язык с «дружелюбным» синтаксисом, который подразумевает инфиксную нотацию в записи арифметических выражений.

В настоящее время существует много скриптовых языков программирования, однако функциональных среди них мало. Функциональных языков тоже много. Но скриптовых среди них мало.

1 Языки-прототипы

Для описания синтаксиса языка используются расширенная форма Бэкуса-Наура (РБНФ). Альтернатива обозначается символом '|'. '*' после выражения означает, что оно может быть включено 0 и более раз, '+' - 1 и более, '?' - 0 или 1 разНетерминальные символы начинаются с заглавной буквы. Терминальные либо начинаются малой буквой, либо состоят целиком из заглавных букв.

Язык является регистрозависимым.

1.1 Словарь и представление

Существуют следующие виды токенов: идентификатор, число, символ, строка, операторы и ключевые слова. Пробельные символы игнорируются, если они не существенны для разделения двух последовательных токенов.

- Идентификаторы – последовательности символов, исключающие пробельные символы, точки, запятые, скобки, кавычки, вертикальную черту. Второй вариант записи идентификаторов – внутри двух вертикальных черт – подразумевает возможность использовать любые символы.

```
1 ident ::= [a-zA-z0№-9\!\\%\:\;\-\/\?\~\\];
```

- Число – целочисленная или вещественная константа. Типом числа считается минимальный тип, содержащий значение этого числа. Если константа начинается с $'0x'$, то число рассматривается, как записанное в шестнадцатеричной системе счисления. Иначе - в десятичной.

```
fdigit  ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'.
digit   ::= '0' | fdigit.
hexd    ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'a' | 'b' | 'c' |
    'd' | 'e' | 'f'.
dec     ::= '0' | (fdigit digit*).
hex     ::= '0x' ('0' | (fdigit | hexd) (digit | hexd)*).
pow     ::= 'e' ( '+' | '-' )? dec.
real    ::= dec '.' digit+ pow?.
number  ::= '+inf' | '-inf' | dec | hex | real.
```

- Строки – последовательности символов, заключённые в одинарные (') или двойные(") кавычки.

- Операторы и ключевые слова – специальные символы, пары символов и слова, зарезервированные системой.

  Список таких символов: $+- <> */ = ,1. : |[]() \ -- >< -$ if scheme map filter reduce eval

2 Способы реализации языка программирования

3 Примеры кода

```
day-of-week day month year <-
    a <- (14 - month) mod 12
    b <- year - a
    m <- month + (a * 12) - 2

    (7000 + day + y + (y mod 4) + (y mod 400) + (31 * m mod 12)
        - (y mod 100)) div 7
```

```
8
9  count x []         <- 0
10 count x [ x : xs ] <- 1 + count x xs
11 count x [ y : xs ] <- count x xs
12
13 count 'a ['a 'b 'c 'a]
14
15 replace pred? proc []         <- []
16 replace pred? proc [ x : xs ] <-
17     if | pred? x [ proc x : replace pred? proc xs ]
18        |         [      x : replace pred? proc xs ]
19
20 replace zero?
21         \ x -> x + 1
22         [ 0 1 2 3 0 ]
23
24 replace odd?
25         \ x -> x * 2
26         [ 1 2 3 4 5 6 ]
27
28 replace \ x -> 0 > x
29         exp
30         []
31
32 replicate x 0 <- []
33 replicate x n <- [ x : replicate x ( n - 1 ) ]
34
35 replicate 'a 5
36 replicate [ 'a 'b ] 3
37 replicate 'a 0
38
39 cycle xs 0 <- []
40 cycle xs n <- append xs cycle xs ( n - 1 )
```

```
41
42 cycle [ 0 1 ] 3
43 cycle [ 'a 'b 'c ] 5
44 cycle [] 0
45
46 and-fold        <- #t
47 and-fold x : xs <- x && and-fold : xs
48
49 and-fold #f #f #f
50 and-fold #f #f #t
51 and-fold #f #t #t
52 and-fold #t #t #t
53 and-fold
54
55 or-fold        <- #f
56 or-fold x : xs <- x || or-fold : xs
57
58 o        <- \ y -> y
59 o x : xs <- \ y -> ( x o : xs ) y
60
61 f x <- x * 2
62 g x <- x * 3
63 h x <- -x
64
65 (o f g h) 1
66 o . [ 1 ]
67
68 find-number a b c <-
69     a <= b &&
70     (
71         (zero? a div c) && a ||
72         find-number (a + 1) b c
73     )
```

```
74
75 find-number 0 5 2
76 find-number 7 9 3
77 find-number 3 7 9
78
79 eval 'a y <- \ x -> y + x\n' +
80      'a f'
81      { f <- 4 }
82
83 scheme (define (b x) (+ 1 x))
84 scheme (+ 5 (b 7))
85
86 c <- [ a.[5  3] b.[6] 8 10 [1 2 3] "sova" ]
87
88 func . args
```

Список литературы

[1] Скоробогатов С.Ю. Лекции по курсу 'Компиляторы', 2014.