Содержание

1 Описание синтаксиса

Yet Another LISP–Like

```
<code> ::= {<statement>}
<statement> ::= "(", <function_name>, <parameters>,
    ")" | <number> | <identificator>
<parameters> ::= <statement>, {<statement>}
<identifier> ::= <symbol>, {<symbol> | <digit>}
<digit> ::= "0" | ... | "9"
<symbol> ::= "a" | ... | "z" | "A" | ... | "Z" | "_"
<number> ::= [sign], <digit>, {digit}
<sign> ::= "–" | "+"
<function_name> ::= <identifier> | "+" | "–" | "*" |
    "/" | "?"
```

2 Основные принципы

Код состоит из выражений, которые вычисляются последовательно. Выражение состоит из фукнции, которая принимает некоторое, больше или равное 1, число параметров, которые могут являться константами, переменными или другими выражениями, в этом случае для вычисления функции требуется вычислить это выражение, это делается последовательно слева направо. Исключениями являются **any** и **do**, их поведение описано далее. При работе с переменными они должны быть определены.

3 Функции языка

3.1 list

Параметры: принимает неограниченное число параметров любого типа
Возвращает: значение последнего параметра
Использование: используется главным образом для вычисления последовательности функций
Пример:

```
(list
    (list 1)
    (list 1 2)
)
```

3.2 any

Параметры: принимает 2 параметра любого типа
Возвращает: равновероятно выбирает 1 или 2 параметр, вычисляет его и возвращает его значение
Пример:

```
(any 1
    (any 2
        (any 3 4)
))
```

3.3 do

Параметры: принимает 2 параметра любого типа
Возвращает: вычисляет первый параметр $n \in [0, \text{inf}]$ раз и возвращает его, если $n = 0$ возвращает значение второго параметра
Пример:

```
(do
    (def
        (* n 2)
        n
    )
    (def
        -1
        n
    )
)
```

3.4 Арифметические и логические: +, -, /, *, leq, geq, le, gr

Параметры: принимает 2 целых параметра
Возвращает: вычисляет арифметическое или логическое выражение, в случае логического 0 это false, не 0 - true. При делении на ноль функция может принимать любое целое значение.
Пример:

```
(def
    (+
        (*  4
```

```
            5
        )
        (/  8
            2
        )
    )
    n
)
```

3.5 array

Параметры: принимает 2 параметра любого типа
Возвращает: объект типа массив, отображение из первого типа данных во второй с уже заданным соответствием между параметрами. (отображение ключ → значение)
Пример: Вернется массив с уже заданным 1337 → 42

```
(array
    1337
    42
)
```

3.6 get

Параметры: принимает 2 параметра: параметр типа ключа и массив
Возвращает: значение, соответствующее заданому ключу, если его нет в массиве, то все что угодно
Пример: Вернется 42

```
(get
    1337
    (array
        1337
        42
    )
)
```

3.7 put

Параметры: принимает 3 параметра: параметр типа ключа, параметр типа значение и идентификатор переменной-массив

Возвращает: массив-копию с измененным отображением ключ → значение, где добавлена переданная как параметры пара
Пример: Вернется массив с уже заданными 1337 → 42, 42 → 1337, 0 → -1

```
(list
    (def
        (array
            1337
            42
        )
        var
    )
    (def
        (put
            0
            -1
            var
        )
        var
    )
)
```

3.8 ?

Параметры: принимает 1 параметр - целое число
Возвращает: 0, в случае если значение параметра равно 1, иначе значение полагается неопределенным и данное выражение высчитывается заново после вычислениях всех остальных предшедствующих параметров в функции, которой потребовалось вычислить это выражение. Если этой функции нет, т.е оно вычисляет на правах выражения в коде программы, то аналогично все выражения будут вычисленны заново.
Пример: После завершения вычислений n=42

```
(list
    (def 1 n)
    (do
        (def
            (+ n 1)
            n
        )
        0
```

```
    )
    (? (eq n 42))
)
```

3.9 def

Параметры: принимает 2 параметра одинакового типа, второй должен быть идентификатором переменной или ссылкой на нее
Возвращает: ссылку на второй параметр, создается копия первого параметра и устанавливается соответствие идентификатор второго параметра ↔ эта переменная.
Пример:

```
(def
    (array
        1337
        42
    )
    myArray
)
```

4 Реализация сортировки массива

```
(list
    (def 5 n)

# ARRAY DEFINITION
    (def
        (array 1 0)
        p
    )
    (def 0 i)

    (do
        (list
            (def (+ i 1) i)
            (def
                (put
                    i
                    (+ 1337 i)
```

```
                p
            )
            p
        )
        (? (leq i n))
    )
    0
)

(? (eq n i))

# BUBBLE SORT HERE

(def 1 i)

(do
    (list
        (def 1 j)
        (do
            (list
                (? (le j n))
                (def (get j p) this)
                (def (get (+ j 1) p) next)

                (any
                    (list
                        (? (le this next))
                        (def (put (+ j 1) this p)
                            p)
                        (def (put j next p) p)
                    )
                    (? (geq this next))
                )

                (def (+ 1 j) j)
            )
            0
        )
        (? (eq j n))

        (def (+ 1 i) i)
```

```
            (? (leq i n))
        )
        0
    )
    (? (eq i n))

# END SORT

)
```