

# Функциональное программирование

Введение в язык LISP

# Современные языки функционального программирования

[Лисп](#) — ([Джон Маккарти, 1958](#)) и его диалекты, наиболее современные из которых:

[Scheme](#)

[Clojure](#)

[Common Lisp](#)

[Erlang](#) — ([Joe Armstrong, 1986](#)) для создания распределенных приложений.

[APL](#) — предшественник современных научных вычислительных сред, таких как [MATLAB](#).

[ML](#) (Meta Language) ([Робин Милнер, 1979](#), из ныне используемых диалектов известны [Standard ML](#) и [Objective CAML](#)).

[F#](#) — функциональный язык семейства ML для платформы [.NET](#)

[Scala](#)

[Miranda](#) ([Дэвид Тёрнер, 1985](#), который впоследствии дал развитие языку Haskell).

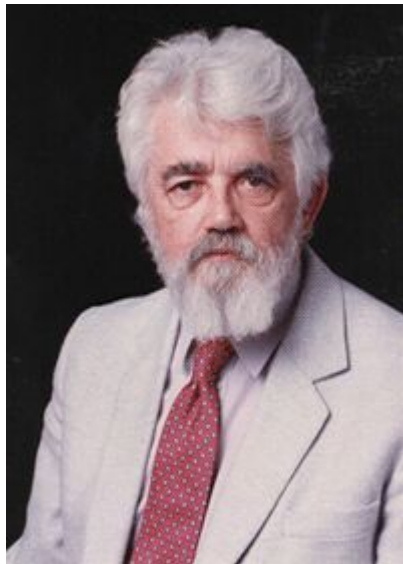
[Nemerle](#) — гибридный функционально/императивный язык.

[XSLT](#)<sup>[3][4]</sup> и [XQuery](#)

[Haskell](#) — [чистый](#) функциональный. Назван в честь [Хаскелла Карри](#).

# Создание языка Lisp

Начало 1930 –ых А. Church - формализация функций в lambda-исчислении



Джон Маккарти (John McCarthy)  
(4.09.1927 - 24.10.2011)  
профессор Станфордского  
университета с 1962 г.

Цитата из "*Lisp 1.5 Programmers Manual*",  
опубликованного в 1960 году, гласит: "это  
был очень специализированный язык, в  
котором программный код всегда  
представлялся в виде данных, а данные  
могли служить кодом."

# История развития языка Lisp

## Основные диалекты

### MacLisp

В 1964 году была создана первая реализация Маклиспа для PDP-6  
первый компилятор, библиотека математических функций

### Interlisp

Начало 1970-ых , IDE, хорошо документирован

### Franz Lisp

в конце 1970-х годов для новых компьютеров VAX

### Scheme

в 1976 году в MIT в рамках проекта по созданию лисп-машины

### Zetalisp

создан в MIT во второй половине 1970-х годов, графический интерфейс

# Стандарт COMMON LISP

[1984 г.](#) Steele G.L. Common Lisp: The Language. - Digital Press, Burlington, MA.: макросы, функционалы, замыкания; операторы императивных языков – циклы, условия

[1990 г.](#) Steele G.L. Common Lisp: The Language. 2nd Edition - Digital Press, 1030p.: включена объектная система CLOS

В [1995 году](#) Common Lisp был стандартизован [ANSI](#). Стандарт практически повторил спецификацию 1990 года

# Современные LISP-системы

## Поставщики коммерческих Лисп-систем

### [LispWorks LLC](#)

LispWorks - наиболее сбалансированная система по соотношению цена-качество. Цена профессиональная версии — \$1500. Библиотека графического интерфейса СAPI портативна между платформами Linux, Windows и MacOS. Есть 64-разрядные версии.

### [Franz Inc.](#)

Allegro Common Lisp : охват различных платформ, хорошие возможности среды разработчика, дополнительных библиотек (платных и с открытым кодом). Есть 64-разрядные версии

[Corman Technologies](#) Corman Lisp - одна из "молодых" систем, под Windows

# Современные LISP-системы

## Общедоступные (freeware) Коммон Лисп системы



[CLISP](#) Один из недостатков - компиляция в байт-код



[CCL](#) (Clozure Common Lisp) Работает на платформах Mac OS X, Linux, FreeBSD и Windows

[MUCL](#) (Carnegie Mellon University Common Lisp)

Полная реализация Коммон Лисп с компиляцией в машинный код, работающая на самых разных Unix-платформах.

[SBCL](#) (Steel Bank CL) Основные платформы: Linux, FreeBSD и MacOS.

CMUCL

# Особенности языка Лиспа

- **Одинаковая форма данных и программ**
- **Хранение данных, не зависящее от места**
- **Автоматическое и динамическое управление памятью**
- **Функциональная направленность**
- **Динамическая проверка типов**
- **Интерпретирующий и компилирующий режимы работы**
- **Пошаговое программирование**
- **Единый системный и прикладной язык программирования**



# Введение в язык LISP

## Атомы и списки.

Основная структура данных в Лиспе - символьные или S-выражения, которые определяются как атомы или списки.

Атомы: это символы и числа. Они представляют собой те объекты Лиспа, из которых строятся остальные структуры.

Символ - это имя, состоящее из букв, цифр и специальных знаков (+ - / @ \$ % ~ & \ > < \_ ).

В Лиспе символы обозначают числа, другие символы или более сложные структуры, программы (функции) и другие лисповские объекты.

Пример: x, г-1997, символ, function.

В большинстве ЛИСП-систем прописные и строчные буквы отождествляются:

zzz ⇔ ZZZ

Числа не являются символами, так как число не может представлять иные лисповские объекты, кроме самого себя, или своего числового значения. В Лиспе используется большое количество различных типов чисел (целые, десятичные и т. д.) - 24, 35.6, 6.3 e5.

Символы T и NIL имеют в Лиспе специальное назначение:

T обозначает логическое значение истина,

NIL - логическое значение ложь.

Символы T и NIL имеют всегда одно и тоже фиксированное значение.

Их нельзя использовать в качестве имен других лисповских объектов.

Символ NIL обозначает также и пустой список.

Числа и логические значения T и NIL являются константами, остальные символы - переменными, которые используются для обозначения других лисповских объектов.

Кроме того, существуют глобальные специальные переменные, имеющие встроенные значения: напр., PI – значение числа π.

Для превращения любого символа в константу используется директива (DEFCONSTANT **символ значение**)

Символ, определенный как константа, не может изменять предписанного ему таким определением значения.

Напр., (DEFCONSTANT z1 10) – значение z1 теперь нельзя переопределить.

Список - это упорядоченная последовательность, элементами которой являются атомы или списки (подсписки).

- Списки заключаются в круглые скобки, элементы списка разделяются пробелами.

- Открывающие и закрывающие скобки находятся в строгом соответствии. Список всегда начинается с открывающей и заканчивается закрывающей скобкой.

- Список, в котором нет элементов, называют пустым и обозначают **()** или **NIL**.

Пустой список - это атом.

Например: (a в (c o) p) - в списке 4 элемента; (+ 3 6) – 3 элемента.

В виде списка в ЛИСПе представляются как текст программы, так и данные:

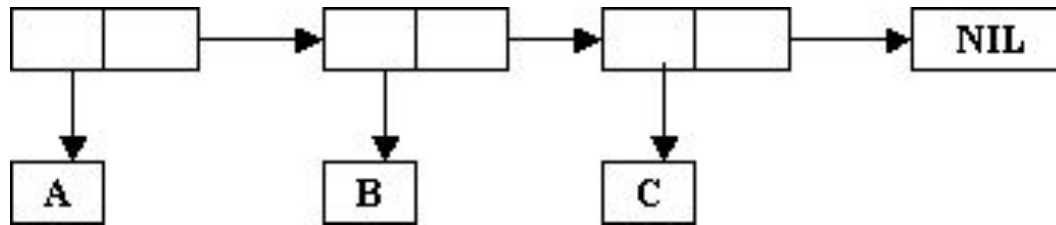
Напр., (+ 2 3) - интерпретируется как функция, результат **5**;

'(+ 2 3) – интерпретируется как список из трех элементов, результат (+ 2 3).

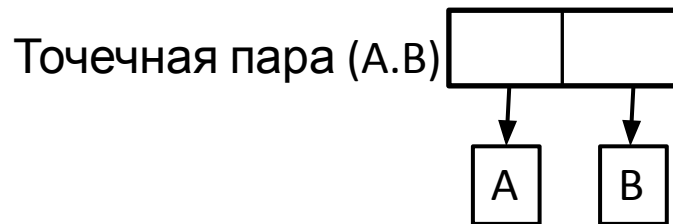
Представление списков в памяти компьютера

Графическая нотация

Рассмотрим представление списка (A B C) в графической нотации:



Nil выполняет роль **пустого списка**



## Запись функций в ЛИСПе.

В процедурных языках программирования для вызова функции используется префиксная нотация, т.е. имя функции стоит перед скобками, окружающими аргументы:

Напр.,  $f(x)$ ,  $fun(x, y)$ ,  $h(x, g(y, z))$

В арифметических выражениях используется инфиксная запись, в которой имя функции, т.е. действия, располагается между аргументами:

Напр.,  $x+y$ ,  $x-y$ ,  $x*(y+z)$

Для записи функций и выражений в ЛИСПе используется списочная форма записи, при которой имя функции или действия, а также аргументы записываются внутри скобок:

$(+ x y)$ ,  $(- x y)$ ,  $(* x (+ y z))$

В ЛИСПе для построения, разбора и анализа списков существуют базовые функции.

Все символьные вычисления сводятся к этой системе базовых функций:

**CAR, CDR, CONST, ATOM, EQ.**

Функции **ATOM** и **EQ** являются базовыми предикатами.

Предикаты – это функции, которые проверяют выполнение некоторого условия и возвращают в качестве результата логическое значение **T** или **NIL**.

По принципу использования все базовые функции делятся на функции *разбора, создания и проверки*.

## Базовые функции языка

<b>CONS</b>	Функция, которая строит списки из «головы» и «хвоста»
<b>CAR</b>	Функция, обеспечивающая доступ к первому элементу списка - «голове»
<b>CDR</b>	Функция, укорачивающая список на один элемент. Обеспечивает доступ к «хвосту» списка, т.е. к остатку списка после удаления его головы.
<b>ATOM</b>	Функция, различающая составные и атомарные объекты. На атомах ее значение «истина», а на более сложных структурах данных – «ложь».
<b>EQ</b>	Функция, которая проверяет атомарные объекты на равенство

## Функции *разбора* CAR и CDR.

Функция **CAR** возвращает в качестве значения первый элемент списка.

**(CAR список)**  $\Rightarrow$  *S - выражение* (атом либо список).

Функция CAR имеет смысл только для аргументов, являющихся списками.

(CAR 'a)  $\Rightarrow$  Error

\_(CAR '(a b c d))  $\Rightarrow$  a

\_(CAR '((a b) c d))  $\Rightarrow$  (a b)

\_(CAR '(a))  $\Rightarrow$  a

\_(CAR NIL)  $\Rightarrow$  NIL

«Голова пустого списка - пустой

список.»

Вызов функции CAR с аргументом (a b c d) без апострофа был бы проинтерпретирован как вызов функции «a» с аргументом «b c d», и было бы получено сообщение об ошибке.



Функция **CDR** - возвращает в качестве значения хвостовую часть списка, т. е. список, получаемый из исходного списка после удаления из него головного элемента:

**(CDR список)**                   ⇒ **список**

Функция CDR определена только для списков.

\_(CDR 'a) ⇒ **Error**

\_(CDR '(a b c d)) ⇒ (b c d)

\_(CDR '((a b) c d)) ⇒ (c d)

\_(CDR '(a (b c d))) ⇒ ((b c d))

\_(CDR '(a)) ⇒ **NIL**

\_(CDR **NIL**) ⇒ **NIL**

## Функция *создания списка* CONS.

Функция CONS строит новый список из переданных ей в качестве аргументов головы и хвоста.

### **(CONS голова хвост)**

Для того чтобы можно было включить первый элемент функции CONS в качестве первого элемента значения второго аргумента этой функции, второй аргумент должен быть списком. Значением функции CONS всегда будет список:

### **(CONS s-выражение список)**

⇒ **список**

\_(CONS 'a '(b c)) ⇒ (a b c)

\_(CONS '(a b) '(c d)) ⇒ ((a b) c d)

\_(CONS (+ 1 2) '(+ 3)) ⇒ (3 + 3)

\_(CONS '(a b c) NIL) ⇒ ((a b c))

\_(CONS NIL '(a b c)) ⇒ (NIL a b c)

Функция	Аргументы	Результат
CONS	A и Nil	(A )
CONS	(A B) и Nil	((A B) )
CONS	A и (B)	(A B)
CONS	A и (B C)	(A B C)
CAR	(A B C)	A
CAR	((A B) C)	(A B)
CAR	A	ошибка
CDR	(A )	Nil
CDR	(A B C D)	(B C D)
CDR	((A B) C)	( C )
CDR	A	ошибка

Функция	Аргументы	Результат
ATOM	VeryLongStringOfLetters	T
ATOM	( A B )	Nil
ATOM	Nil	T
ATOM	()	T
EQ	A A	T
EQ	A B	Nil
EQ	A (A B)	Nil
EQ	(A B) (A B)	Не определено
EQ	Nil и ()	T
CDR	((A B) C)	( C )
CDR	A	ошибка

## Композиции CAR-CDR

Вычисляются в порядке, обратном записи:

(Caar '((A) B C)) -> A

(Cadr '(A B C)) -> B

(Caddr '(A B C)) -> C

(Cadadr '(A (B C) D)) -> C

# Другие простейшие встроенные функции

## Лиспа

Предикат `EQL` сравнивает числа одинаковых типов:

`(EQL число число)`

`(EQL 2.0 2.0) => T`

`(EQL 2 2.0) => NIL` (не годится для разных типов)

Предикат `=` сравнивает числа разных типов: `(= число число)`

`(= 2 2.0) => T`

Предикат `EQUAL` проверяет идентичность записей: `(EQUAL список список)`

`_(EQUAL 'a 'a) => T`

`_(EQUAL '(a b c) '(a b c)) => T`

`_(EQUAL '(a b c) '(CONS 'a '(b c))) => T`

`_(EQUAL 1.0 1) => NIL`

Предикат `EQUALP` проверяет наиболее общее логическое равенство:

`(EQUALP s-выражение s-выражение)`

*Функция `NULL` проверяет, является ли аргумент пустым списком:*

`(NULL s-выражение)`

`(NULL '()) => T`

`(NULL '(1 2 3)) => NIL`

`(NOT (NULL NIL)) => NIL`

`(NULL x) ⇔ (EQ NIL x)`

**(first список)**  $\Leftrightarrow$  **(car список)**  
**(second список)**  $\Leftrightarrow$  **(cadr список)**  
**(third список)**  $\Leftrightarrow$  **(caddr список)**  
**(fourth список)**  $\Leftrightarrow$  **(caddrdr список)**

...

(THIRD (CONS a (CONS b (CONS c NIL)))) => c  
(FOURTH '(a b c)) => NIL

**(NTH n список)**

(NTH 2 '(A B C)) => C

**(LAST список)**

(LAST '(1 2 3 4 5)) => 5

**(LIST arg.1 arg.2 arg.3 ...)** => **(arg.1 arg.2 arg.3 ...)**

(LIST 'a 'b 'c) => (a b c)

(LIST 'a 'b (+ 1 2)) => (a b 3)

(cons 1 (cons 2 (cons 3 NIL))) => (1 2 3)  $\Leftrightarrow$  (list 1 2 3)

## **Выводы:**

- Список – это перечень произвольного числа элементов, разделенных пробелами, заключенный в круглые скобки.
- Элементы списка могут быть любой природы.
- S-выражение - это или атом или список .
- Любое S-выражение может быть построено из атомов с помощью CONS и любая его часть может быть выделена с помощью CAR-CDR.
- Для изображения S-выражений используют различные нотации: графическую, точечную и списочную.
- Базис Лиспа содержит элементарные функции CAR, CDR, CONS, EQ, АТОМ



## Запись Лисп- программ

1) Самая простая **форма выражения - символ.**

*Примеры:*

X

n

Variable1

Переменная2

LongSong

ДолгаяПесня

2) **Имена функций**, лучше всего изображать с помощью символов, для наглядности можно предпочитать заглавные буквы:

*Примеры:*

**CONS**

**CAR**

**CDR**

**ATOM**

**EQ**

3) Все более сложные выражения понимают как применение функции к ее аргументам. Аргументом функции может быть любое s-выражение. Список, первый элемент которого – представление функции, остальные элементы – аргументы функции, – это основная конструкция в Лисп-программе:  
**(функция аргумент1 аргумент2 ... )**

4) **Композиции функций** естественно строить с помощью вложенных скобок:  
**(функция1 (функция2 аргумент21 аргумент22 ... ) аргумент2 ... )**

**(CAR (CONS 'x '(y))) -> x**

**(CDR (CONS 'x '(y))) -> (y)**

**(ATOM (CONS 'x '(y))) -> Nil**

**(CONS (CAR x) (CDR x)) = x для неатомарных x.**

5) **Специальные функции**

**QUOTE - блокировка вычислений**

**(QUOTE '(a b c)) -> (A B C)**

**EVAL – запускает интерпретатор, позволяет вычислять значения выражений, представленных в виде списков,**

<http://lisp.ystok.ru/ru/lispworks/>