*Лекция 4*

# *Списки. Встроенные предикаты.*

## Содержание



## 4.1 Списки

**Списки** - такая же важная структура данных в Прологе, как и в Лиспе.

Список в Лиспе **(a b c d) (1 2 (3))**
записывается на Прологе **[a,b,c,d] [1,2,[3]]**

т.е. элементы записываются в квадратных скобках через запятую.

**Элементами списка** могут быть любые термы.

**Пустой список** - не **nil** , а **[ ]**.

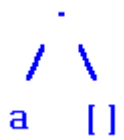
### 4.1.1 Представление списка диаграммой.

Список в лиспе можно представить через функцию **cons**

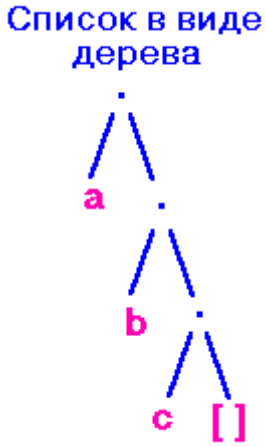
```
(cons 'a ()) => (a)

или          [ | ]
             / \
            a   ()

 или          a.()
```

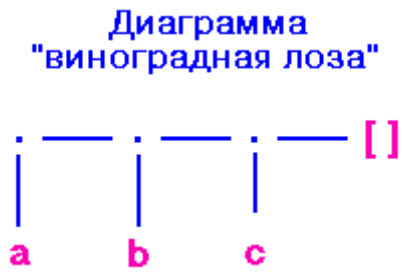
В Прологе функции **cons** соответствует функтор **"." (точка)**.

**.(a,[])** соответствует **[a]** , это другая форма записи
или

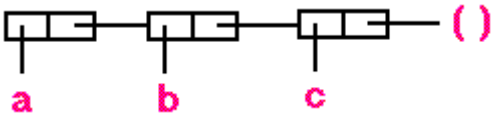



Ответственно список **[a,b,c]** представляется как структура **.(a ,.(b,.(c,[])**
или в виде дерева, или диаграммой "виноградная лоза"

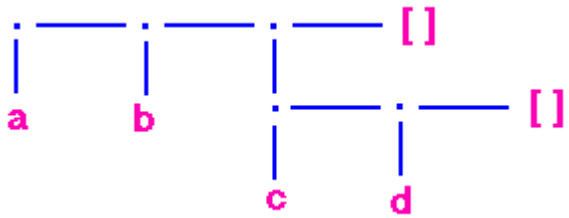



В лиспе :



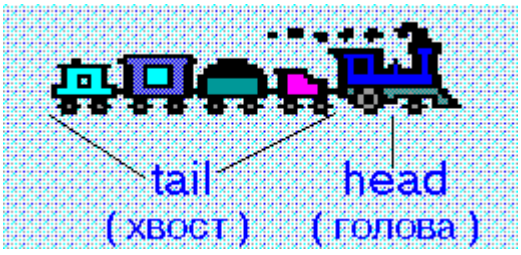

Для вложенных списков **[a,b,[c,d]]**




- на верхнем уровне три элемента

- на втором уровне два элемента.

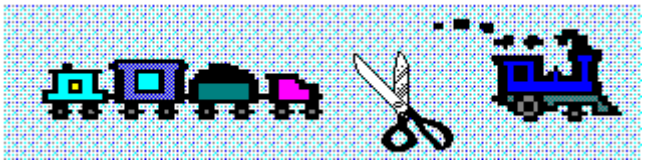
## 4.1.2 Выделение головы и хвоста списка




Первый элемент списка - голова.

Список без головы - хвост.

Главной операцией при работе со списками является **расщепление списка на голову и хвост**.



В Лиспе для этого используются функции **car** и **cdr.**

В Прологе имеется специальная форма представления списка, называемая **cons**-формой записи:

**[Head|Tail] или [H|T] [a|[]] = [a]**

При конкретизации формы списком **H** сопоставляется с головой списка, а **T** - с хвостом.

Например

**p([a,b,c]).**

**?-p([X|Y]).**
**yes**
**X=a**
**Y=[b,c]**

Таким образом выделяются одновременно голова списка и хвост.

Рассмотрим сопоставление двух списков :

| Список 1 | Список 2 [H\|T] H | T |
|---|---|---|
| [[a,b],c] | [a,b] | [c] |
| [1,[2,3]] | 1 | [[2,3]] |
| [a(c),b(d)] | a(c) | [b(d)] |
| [A is 2+3, B is 1+1] | A is 2+3 | [B is 1+1] |
| [ ] | нет решений | |

| Список 1 | Список 2 | |
|---|---|---|
| [a, b, c, d] | [X, H \| T] | X = a<br>H = b<br>T = [c, d] |
| [a, X \| b] | [Y, a \| _] | X = a<br>Y = a |
| [a, b, c] | [X, Y] | нет решений |

## 4.1.3 Шаблоны списков.

**Шаблон (образец) списка** -это форма описания множества (семейства)списков, обладающих определенными свойствами.

Например:

**Шаблон списка [X|Y]** описывает любой список,состоящий не менее чем из одного элемента.

**Шаблон [X,Y|Z]** - список, состоящий не менее чем из двух элементов.

**Шаблон [b|Z]** - список, первым элементом которого является b.

**Шаблон [Y,X,Z]** - список из трех элементов.

Шаблоны списка используются при описании процедур работы со списками.

### 4.1.4 Определения отношений через cons форму списка

**Задача 1:** Определить отношение **replace_first**, которое заменяет первый элемент списка новым

Например

**?-replace_first([a,b,c],w,X).**
**X=[w,b,c]**

Это отношение:

**replace_first([H|T],A,[A|T]).** или
**replace_first([_|T],A,[A|T]).**

Что будет ответом для следующего вопроса ?

**?-replace_first([_|T],A,[a,b,c]).**

## 4.2 Процедуры обработки списков

**Процедура в прологе** - это совокупность предложений с головами, представленными одинаковыми термами.

Для обработки списков используются типовые процедуры, аналогичные функциям лиспа.

### 4.2.1 member

Проверяет принадлежность элемента списку.

**member(X, L)**

Если **X** принадлежит **L**, то истина и ложь в противном случае.

С точки зрения декларативного смысла:

1. **X** принадлежит списку, если **X** совпадает с головой списка.
2. **X** принадлежит списку, если **X** принадлежит хвосту списка.

Можно записать:

```
member(X, [X|T]).
member(X, [H|T]) :-member(X, T).
```

С точки зрения процедурного смысла - это **рекурсивная процедура.**

Первое предложение терминальное условие.
Когда хвост будет равен **[]** проверка остановиться.

Второе предложение рекурсивное.
Сокращение списка происходит за счет взятия хвоста **(cdr-рекурсия)**.

Примеры применения

```
?-member(a, [a, b, c]).
Yes
?-member(X, [a, b, c]).
Yes
X = a
X = b
X = c
```

Ответьте на вопрос:

```
?-member(a, X).
```

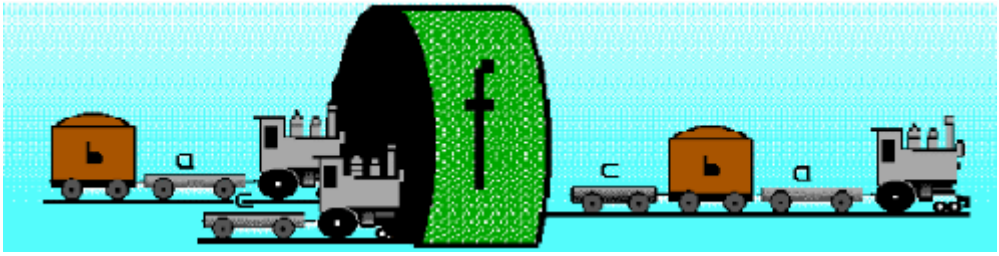
## 4.2.2 append

Используется для соединения двух списков. т.е

**append (L1, L2, L3)**

**L1** и **L2** - списки, а **L3** - их соединение.

```
?-append ([a, b], [c], [a,b,c]).
Yes
```



Для определения процедуры **append** используем два предложения:

1. Если присоединить пустой список **[]** к списку **L**, то получим список **L**.
   **append([], L, L).**
   **append([X|L1], L2, [X|L3]):- append(L1, L2, L3).**
2. Если присоединить не пустой список **[X|L1]** к списку **L2**, то результатом будет список **[X|L3]**, где **L3** получается соединением **L1** к **L2** :

```
    |X|      L1       |      L2       |

        ----------------------------------------

    |X|              L3                |
```

С точки зрения процедурной семантики
первое предложение - **терминальное условие**,
второе - **рекурсивное с хвостовой рекурсией**.

Рассмотрим примеры применения

```
?-append([a], [b, c], L).
L=[a, b, c]

?-append([a], L, [a, b, c]).
L=[b, c]

?-append(L, [b, c], [a, b, c]).
L=[a]
```

Можно использовать для разбиения

```
?-append(L1, L2, [a, b, c]).
L1=[]
L2=[a, b, c];
L1=[a]
L2=[b, c];
L1=[a, b]
L2=[c];
L1=[a, b, c]
L2=[]
```

## 4.2.3 Применение append

Процедуру **append** можно использовать для поиска комбинаций элементов.
Например можно выделить списки слева и справа от элемента

```
?-append(L, [3|R], [1, 2, 3, 4, 5]).
L=[1, 2]
R=[4, 5]
```

Можно удалить все, что следует за данным элементом и этот элемент тоже:

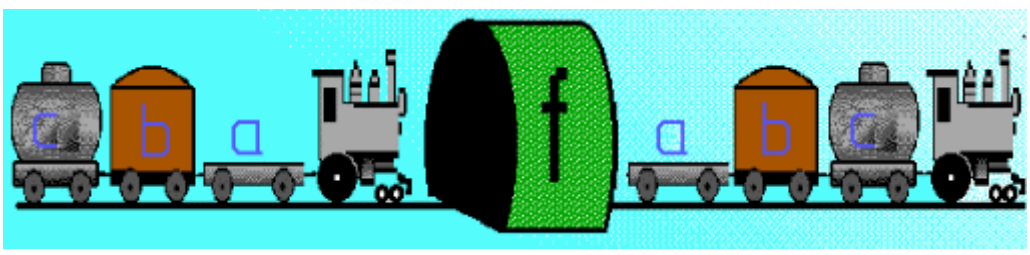
```
?-L1=[a, b, c, d, e], append(L2, [c|_], L1).
L1=[a, b, c, d, e]
L2=[a, b]
```

Можно определить процедуру ,выделяющую последний элемент в списке:

```
last(X, L):-append(_, [X], L).
```

## 4.2.4 reverse

Процедура **reverse** обращает список.



1. Пустой список после обращения - пустой.
   **reverse([], []).**
2. Обратить список **[X|L1]** и получить список **L2** можно, если обратить список **L1** в **L3** и в хвост ему добавить **X**

**reverse([X|L1], L2):-reverse(L1, L3),**
**append( L3, [X], L2).**

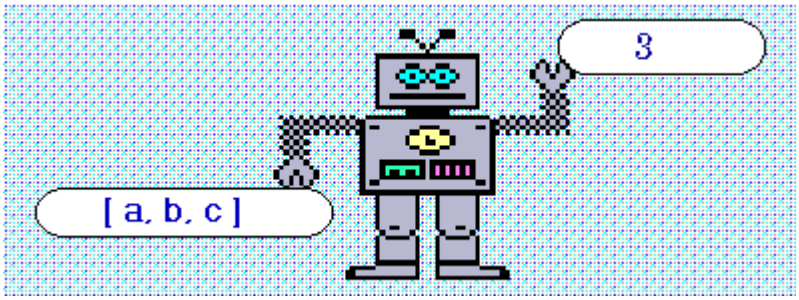
**reverse([X|L1], L2):-reverse(L1, L3),**
**append( L3, [X], L2).**

### 4.2.5 Длина списка

Можно, используя рекурсивный вызов, легко посчитать длину списка:

**length([], 0).**
**length([X|L], N):-length(L, M), N is M+1.**

**?-length([a, b, c], N).**
**N=3**




## 4.3 Встроенные предикаты

До сих пор мы получали ответы в форме предлагаемой прологом:

1. печатались значения переменных присутствующих в вопросе;
2. вопросы полностью записывались после запроса "**?-**".

Однако можно задавать вопросы и получать ответы в произвольной форме.

Для этого достаточно использовать т.н. встроенные предикаты.

**Встроенные предикаты** - предикаты исходно определенные в прологе, для которых не существует процедур в базе данных.

Когда интерпретатор встречает цель, которая сравнивается с встроенным предикатом, он вызывает встроенную процедуру.

Встроенные предикаты обычно выполняют функции не связанные с логическим выводом.

При сопоставлении строенные предикаты обычно дают побочный эффект,который не устраняется при бэктрекинге.

## 4.3.1 Простые встроенные предикаты ввода-вывода.

Встроенные предикаты обеспечивают возможности ввода-вывода информации:

1. **write/1** - этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет вывод значения аргумента на экран. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.

2. **nl/0** - этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет перевод на следующую строку. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.

3. **tab/1** - этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет печать коли- чество пробелов заданное аргументом. Аргумент должен быть целым. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.

4. **read/1** - этот предикат читает терм , который вводится с клавиатуры и заканчивается точкой. Этот терм сопоста- вляется с аргументом. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.

Например,

**pr1:- read(X),nl,write('X='),tab(2),write(X).**

При вызове

**?-pr1.**

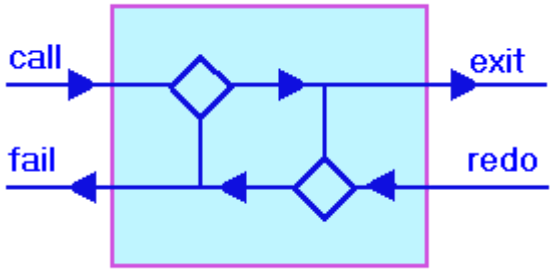
последовательность термов читает значение **X**,переводит строку,печатает **'X='**,пропускает два пробела и печатает значение **X**.

## 4.3.2 Процедурный смысл встроенных предикатов ввода-вывода.

Определяя встроенные предикаты мы писали: "этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет... При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект."

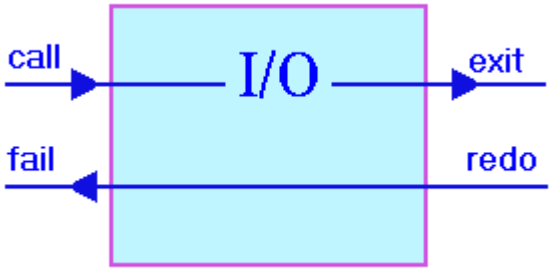
Обычно в прологе вход в цель возможен через **"call"** при вызове и через **"redo"** при бэктрекинге.

Цель имеет внутреннею структуру:




**Первый ромб** - решение при вызове **call**.
**Второй ромб** - решение при сбросе **redo**.

Для встроенных предикатов нет внутренних точек решения внутри цели.
Она представляется в виде:




Таким образом проход через встроенный предикат будет :
или **call - exit**,
или **redo - fail**.

# 4.4 Ввод-вывод списков.

Для ввода-вывода списков возможны следующие два способа.

## 4.4.1 Ввод-вывод списка как терма.

При этом способе список рассматривается как один терм.

Например, процедура:

```
pr:-write('Введите список L:'),nl,
        read(L),nl,
        write('Список L='),
        tab(2), write(L),nl.
```

При вызове цели

**?-pr.**

она будет выполнятся следующим образом:

```
Введите список L:
[a,b,c,d].
Список L= [a,b,c,d]
```

## 4.4.2 Поэлементный ввод-вывод списка.

Данный способ может быть организован с помощью рекурсивно определенных процедур.

```
read_list([X|T]):-write('Введите элемент: '),
                  read(X),
                  X\==end,!,
                  read_list(T).
```

**end** - терм , означающий конец списка.

```
read_list([]).
write_list([]):-nl.
```

```
write_list([H|T]):-write(H),
                   tab(2),
                   write_list(T).
```

Тогда после вызова цели:

**?-read_list(L),nl,write('Список='),nl, write_list(L).**

Возникает следующий диалог:

**Введите элемент: a.**
**Введите элемент: b.**
**Введите элемент: c.**
**Введите элемент: d.**
**Введите элемент: end.**

**Список= a b c d**

---