

Programowanie
w logice

PROLOG

Struktury danych - listy

- **Lista** – ciąg uporządkowanych elementów o dowolnej długości.
- Elementy listy mogą być dowolnymi termami: stałymi, zmiennymi, strukturami (w tym listami).
- Lista jest albo **listą pustą**, nie zawierającą żadnych elementów, albo jest strukturą z dwiema składowymi: **głową** i **ogonem**.

Listy

- Lista jest **strukturą rekurencyjną** (do jej konstrukcji użyto funktora **.** (kropka))
- Listę pustą zapisuje się: `[]`
- Głowa i ogon listy są argumentami funktora **.** (kropka)
- Przykłady:
 - `.(a,[])` lista jednoelementowa
 - `.(a,.(b,[]))` lista o elementach a, b
 - `.(a,.(b,.(c,[])))` lista o elementach a, b, c

Przykłady list

Wygodniejszy zapis listy: elementy oddziela się przecinkami i umieszcza między nawiasami `[` oraz `]`
Zamiast: `.(5,.(8,.(3,[])))`
pisze się: `[5,8,3]`
Przykłady list:
`[wydział, informatyki]`
`[X, posiada, Y]`
`[autor(adam, mickiewicz), "Pan Tadeusz"]`
`[[2,3],[5,6,7],[2,8]]`

Lista z głową X i ogonem Y `[X|Y]`

lista	głowa	ogon
<code>[]</code>	niezdefiniowane	niezdefiniowane
<code>[a]</code>	<code>a</code>	<code>[]</code>
<code>[a,b]</code>	<code>a</code>	<code>[b]</code>
<code>[a,b,c]</code>	<code>a</code>	<code>[b,c]</code>
<code>[[1,2],[3,4],5]</code>	<code>[1,2]</code>	<code>[[3,4],5]</code>

Unifikacja list (sprawdzić, czy zakończy się sukcesem)

- `.(a,.(b,.(c,[]))) = [a,b,c].`
- `.(a,.(B,.(C,[]))) = [a,b,c].`
- `[a,V,1,[c,s],p(X)] = [A,B,C,D,E].`
- `[1,2,3,4] = [1|[2,3,4]].`
- `[1,2,3,4] = [1,2|[3,4]].`
- `[1,2,3,4] = [1,2,3|[4]].`
- `[1,2,3,4] = [1,2,3,4,[]].`
- `[1,2,3,4] = [1|2,3,4].`
- `[Head|Tail] = [1,2,3,4].`
- `[Head|Tail] = [[1,1,ala],2,3,4].`
- `[Head|Tail] = [X,y].`

Przetwarzanie list

- Listy są **strukturami rekurencyjnymi**, do ich przetwarzania służą procedury rekurencyjne.
- Procedura** – zbiór klauzul zbudowany w oparciu o ten sam predykat.
- Procedura rekurencyjna** składa się z klauzul:
 - Faktu opisującego sytuację, która powoduje zakończenie rekurencji, np. napotkanie listy pustej,
 - Reguły, która przedstawia sposób przetwarzania listy.
W jej ciele znajduje się ten sam predykat, co w nagłówku, tylko z innymi argumentami.

Przykład procedury rekurencyjnej

Wypisanie na ekranie elementów listy:

```
pisz([]).
pisz([X|Y]):-write(X),nl,pisz(Y).
```

Fakt mówi, że w przypadku napotkania listy pustej (końca listy) nie należy nic robić.

Reguła mówi: podziel listę i ogon, wydrukuj głowę listy, następnie ją pomiń i zastosuj tę samą metodę do powstałego ogona.

nl ozn. przejście do nowej linii

Przykłady predykatów wbudowanych działających na listach

- is_list (L)** - sprawdza, czy L jest listą

Przykład.

```
?- is_list([1,2,a,b]).
true.
```

- append (L1,L2,L3)** – łączy listy L1 i L2 w listę L3

Przykład.

```
?- append([1,2],[3,4],X).
X=[1,2,3,4].
```

- member(E,L)** – sprawdza, czy element E należy do listy L lub wypisuje elementy listy L

Przykład.

```
?- member(5,[3,6,5,7,6]).
true
?- member(X,[2,3,4,9]).
X = 2 ;
X = 3 ;
X = 4 ;
X = 9 ;
false.
```

- memberchk(E,L)** - równoważny predykatowi member, ale podaje tylko jedno rozwiązanie (pierwsze)

- nextto(X,Y,L)** – predykat spełniony, gdy Y występuje bezpośrednio po X

Przykład.

```
?- nextto(X,Y,[2,3,4,5]).
X = 2,
Y = 3 ;
X = 3,
Y = 4 ;
X = 4,
Y = 5 .
?- nextto(3,Y,[2,3,4,5]).
Y = 4
?- nextto(X,4,[2,3,4,5]).
X = 3
```

- delete(L1,E,L2)** – z listy L1 usuwa **wszystkie** wystąpienia elementu E, wynik uzgadnia z listą L2

Przykład.

```
?- delete([1,2,3,2,5,3],3,X).
X = [1, 2, 2, 5].
```

- select(E,L,R)** – lista R jest uzgadniana z listą, która powstaje z L po usunięciu wybranego (jednego) elementu.

Przykład.

```
?- select(3,[1,2,3,2,5,3],X).
X = [1, 2, 2, 5, 3] ;
X = [1, 2, 3, 2, 5]
```

- **nth1(N,L,E)** – predykat spełniony, jeśli element listy L o numerze N daje się uzgodnić z elementem E

Przykład.

```
?-nth1(2,[a,b,c,d],Y).
Y = b.
?-nth1(X,[a,d,b,c,d],d).
X = 2 ;
X = 5.
```

- **last(L,E)** – ostatni element listy L

Przykład.

```
?-last([a,b,c,d],Y).
Y = d.
```

- **reverse(L1,L2)** – odwraca porządek elementów listy L1 i unifikuje rezultat z listą L2

Przykład.

```
?-reverse([a,b,c,d],Y).
Y = [d,c,b,a].
```

- **permutation(L1,L2)** – lista L1 jest permutacją listy L2

Przykład.

```
?- permutation([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1] ;
```

- **sumlist(L,S)** – suma listy liczbowej L

Przykład.

```
?-sumlist([1,4,7,9],S).
S=21.
```

- **length(L,N)** – liczba elementów listy L

Przykład.

```
?-length([b,2,a,0],N).
N=4.
```

Operacje na listach

Sprawdzenie, czy element jest na liście

Procedura: X jest elementem listy L, jeżeli X jest głową listy L lub X jest elementem ogona listy L.

```
element(X,[X|_]).
element(X,[_|Ogon]) :- element(X,Ogon).
```

„_” to zmienna anonimowa zastępująca głowę listy [_|Ogon], jej nazwa nie ma znaczenia

Przykład.

```
?-element(a,[w,s,d,a,e]).
true
```

Predykat wbudowany: **member**

Operacje na listach

Łączenie list

Procedura:

- Jeżeli pierwszy element listy jest pusty [], to drugi i trzeci element muszą być takie same (L = L).
- Jeżeli pierwszy element nie jest pusty, to głową listy L3 staje się głową listy L1, a ogonem listy L3 jest ogon listy L1 złączony z listą L2.

```
polacz([],L,L).
```

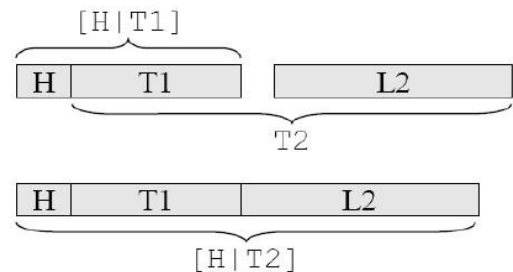
```
polacz([X|L1],L2,[X|L3]) :- polacz(L1,L2,L3).
```

Predykat wbudowany: **append**

Przykład.

```
?-polacz([1,2,3],[2,3,4],L).
```

```
L=[1,2,3,2,3,4].
```



Operacje na listach

Liczba elementów listy liczbowej

Procedura:

- długość listy pustej jest równa 0 (fakt)
- Długość listy, to długość jej ogona plus jeden (reguła)

```
dlugosc([],0).
dlugosc([G|O],N):-dlugosc(O,N1),N is N1+1.
```

Przykład.

```
?-dlugosc([a,s,d,f,g],K).
K=5.
```

Predykat wbudowany: **length**

Operacje na listach

Odwracanie kolejności elementów listy

Procedura:

- odwrotna do listy pustej jest lista pusta (fakt)
- odwrotnością listy jest połączenie odwróconego ogona listy z listą złożoną z głowy listy wejściowej (reguła)

```
odwracanie([],[]).
odwracanie([A|B],C):- odwracanie(B,D),
                        append(D,[A],C).
```

Predykat wbudowany: **reverse**

Literatura

- W. Clocksin, C. Mellish, „Prolog. Programowanie”
- E. Gatnar, K. Stąpor, „Prolog”
- G. Brzykcy, A. Meissner, „Programowanie w prologu i programowanie funkcyjne”
- M. Ben-Ari, „Logika matematyczna w informatyce”