



# Programowanie deklaratywne

Artur Michalski  
Informatyka II rok



## Plan wykładu

- Wprowadzenie do języka Prolog
- Budowa składniowa i interpretacja programów prologowych
- Listy, operatory i operacje arytmetyczne
- Złożone/abstrakcyjne struktury danych
- Sterowanie mechanizmem nawrotów
- Operacje wejścia/wyjścia w Prologu
- Predefiniowane procedury prologowe
- Styl i technika programowania w Prologu

## Listy, operatory i operacje arytmetyczne

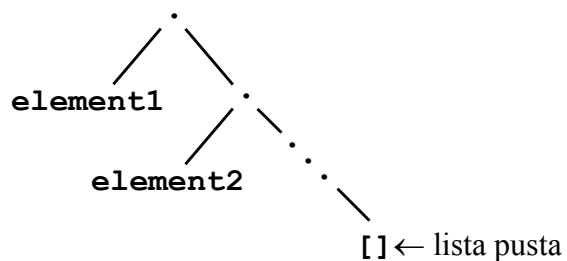
- Reprezentacja list w Prologu
- Wybrane operacje na listach w Prologu
- Notacja operatorów
- Operacje arytmetyczne w Prologu

## Reprezentacja list w Prologu

Lista to dowolnej długości ciąg obiektów zapisywany w postaci:

**[element1,element2,...]**

Reprezentacja wewnętrzna listy odpowiada strukturze drzewiastej:





## Reprezentacja list w Prologu

Budowa listy (niepustej):

- *głowa* - pierwszy element listy,
- *ogon* - pozostałe elementy listy

Własności:

- Głową listy może być dowolny obiekt języka Prolog np. inną listą, termem, zmienną
- Ogon listy jest zawsze listą i może być listą pustą
- Lista jest strukturą rekurencyjną - jeżeli ogon jest niepusty, to również on składa się z głowy i z ogona



## Reprezentacja list w Prologu

Przykładowe listy:

```
?-Hobby1=[muzyka,kuchnia],  
   Hobby2=[narty,taniec],  
   Lista=[tenis,Hobby1,film,Hobby2].
```

```
Hobby1=[muzyka,kuchnia],  
Hobby2=[narty,taniec],  
Lista=[tenis,[muzyka,kuchnia],film,[narty,  
      ,taniec]].
```

Elementem składowym listy może być inna lista.

## Reprezentacja list w Prologu

### Alternatywna notacja list

W prologu listy mogą być zapisywane w sposób odzwierciedlający ich budowę:

**[Head | Tail]**

gdzie:

**Head** - może być ciągiem dowolnych elementów (termów) oddzielonych przecinkiem, a **Tail** - dowolna listą elementów

*Przykład*

**[a,b,c] = [a|[b,c]] = [a,b|[c]] = [a,b,c|[]]**

## Wybrane operacje na listach w Prologu

### Operacja sprawdzania przynależności elementu do listy

Klauzula **member(X,L)** ma być prawdziwa, jeżeli element **X** należy do listy **L**. Przykładowo:

**member(b,[a,b,c])** - powinno być prawdziwe

**member(b,[a,[b,c]])** - powinno być fałszywe, ale

**member([b,c],[a,[b,c]])** - powinno być prawdziwe

Definicja

*X należy do listy L, o ile*

*X jest głową listy L lub*

*X należy do ogona listy L.*

Zapis w Prologu:

**member(X,[X|Tail]).**

**member(X,[Head|Tail]):- member(X,Tail).**

## Wybrane operacje na listach w Prologu

### Operacja łączenia (konkatenacji) list

Klauzula **conc (L1 , L2 , L3)** łączy listę **L1** z listą **L2** w listę wynikową **L3**.

#### *Przykłady*

**conc ([a,b] , [c,d] , [a,b,c,d])** - jest prawdziwe

**conc ([a,b] , [c,d] , [a,b,a,c,d])** - jest fałszywe

#### *Definicja*

*Jeżeli lista L1 jest pusta, to lista L3 jest taka sama jak lista L2.*

*Jeżeli lista L1 jest niepusta i ma postać [H|T1], to lista L3 ma postać [H|T2], gdzie T2 jest połączeniem list T1 i L2.*

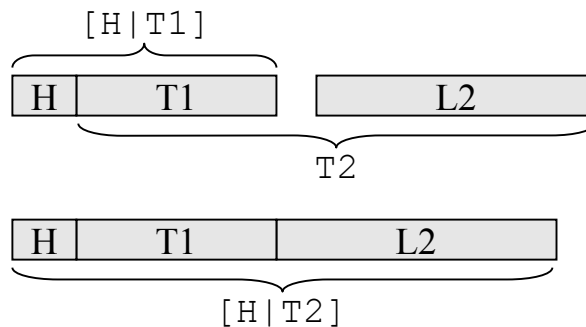
## Wybrane operacje na listach w Prologu

### Operacja łączenia (konkatenacji) list c.d.

Zapis konkatenacji w Prologu:

**conc ([ ] , L , L) .**

**conc ([H|T1] , L2 , [H|T2]) :- conc (T1 , L2 , T2) .**



## Wybrane operacje na listach w Prologu

### Operacja łączenia (konkatenacji) list c.d

Zastosowanie operacji konkatenacji - *dekompozycja listy*:

```
?- conc(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=[];
```

```
No
```

## Wybrane operacje na listach w Prologu

### Operacja łączenia (konkatenacji) list c.d.

Zastosowanie operacji konkatenacji - *szukanie podlist*:

```
?- conc(Przed,[sr|Po],  
        [pon,wt,sr,czw,pt,sob,nd]).
```

```
Przed=[pon,wt]
```

```
Po=[czw,pt,sob,nd]
```

Zastosowanie operacji konkatenacji - *szukanie poprzednika i następnika*:

```
?- conc(_, [Przed,sr,Po|_],  
        [pon,wt,sr,czw,pt,sob,nd]).
```

```
Przed=wt
```

```
Po=czw
```

## Wybrane operacje na listach w Prologu

Operacja łączenia (konkatenacji) list c.d.

Zastosowanie operacji konkatenacji - *usuwanie podlisty*:

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

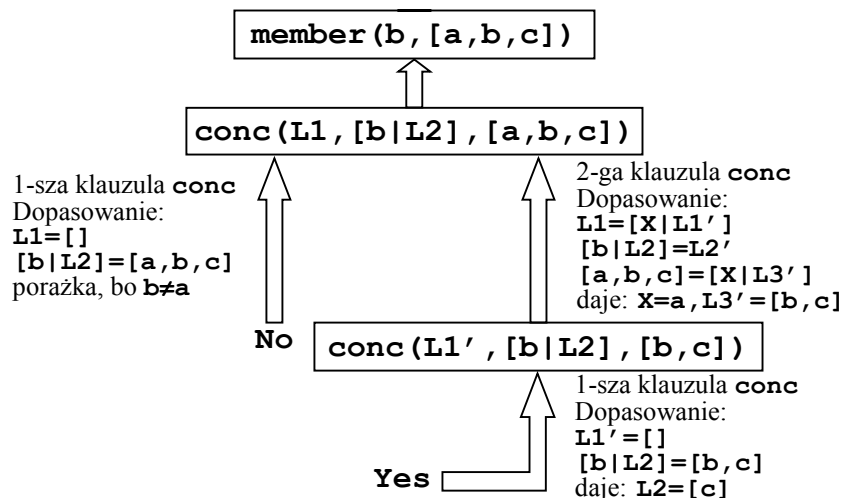
```
L1=[a,b,z,z,c,z,z,z,d,e]
L2=[a,b,z,z,c]
```

Zastosowanie operacji konkatenacji - *inna wersja member*:

```
member(X,L) :- conc(L1,[X|L2],L).
albo:
member(X,L) :- conc(_,[X|_],L).
```

## Wybrane operacje na listach w Prologu

Zastosowanie operacji konkatenacji - *inna wersja member* :



## Wybrane operacje na listach w Prologu

### Operacja dodawania elementu do listy

Najprostszy wariant - dodać element na początku listy (nowa głowa listy).

Zapis w Prologu:

```
add(X,L,[X|L]).
```

## Wybrane operacje na listach w Prologu

### Operacja usuwania elementu z listy

Klauzula **del(X,L,L1)** ma być prawdziwa, jeżeli lista **L1** jest równa liście **L** pomniejszonej o element **X**. Położenie elementu **X** jest dowolne.

#### Definicja

*Jeżeli X jest głową listy L, to lista L1 jest ogonem listy L.*

*Jeżeli X należy do ogona listy L, to usuń stamtąd X.*

Zapis w Prologu:

```
del(X,[X|Tail],Tail).
```

```
del(X,[H|Tail],[H|Tail1]):-
```

```
del(X,Tail,Tail1).
```



## Wybrane operacje na listach w Prologu

Operacja usuwania elementu z listy c.d.

Operacja **del** (**X**, **L**, **L1**) usuwa tylko jedno, dowolne wystąpienie **X** z listy **L**.

Przykład

```
?- del(a, [a,b,a,a], L) .  
L=[b,a,a] ;  
L=[a,b,a] ;  
L=[a,b,a] ;  
No
```

## Wybrane operacje na listach w Prologu

Operacja usuwania elementu z listy c.d.

Zastosowanie operacji **del** (**X**, **L**, **L1**) - *wstawianie elementu do listy*:

```
?- del(a, L, [1,2,3]) .  
L=[a,1,2,3] ;  
L=[1,a,2,3] ;  
L=[1,2,a,3] ;  
L=[1,2,3,a] ;  
No
```

Definicja w Prologu:

```
insert(X,L,BiggerL) :- del(X,BiggerL,L) .
```

## Wybrane operacje na listach w Prologu

### Operacje na podlistach

Klauzula **sublist**(**S**,**L**) jest prawdziwa, jeśli lista **S** zawiera się w liście **L**.

#### *Przykłady*

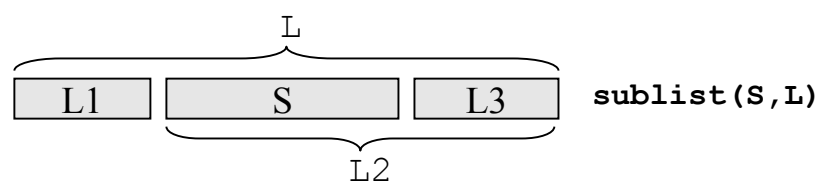
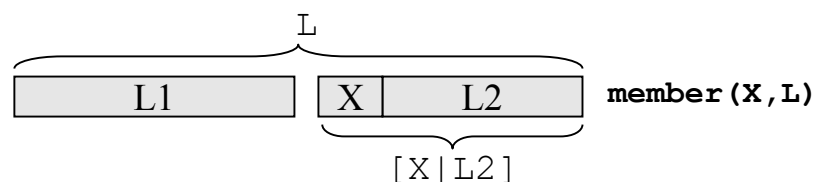
**sublist**(**[c,d,e]**, **[a,b,c,d,e,f]**) - jest prawdziwe

**sublist**(**[c,e]**, **[a,b,c,d,e,f]**) - jest fałszywe

## Wybrane operacje na listach w Prologu

### Operacje na podlistach c.d.

Analogia między klauzulą **member**(**X**,**L**) (w wersji z **conc**) a klauzulą **sublist**(**S**,**L**)



## Wybrane operacje na listach w Prologu

### Operacje na podlistach c.d.

#### Definicja

*Lista S należy do listy L, o ile lista L składa się z dwóch list L1 i L2, a lista L2 jest połączeniem list S i L3.*

#### Zapis w Prologu

```
sublist(S,L) :- conc(L1,L2,L) , conc(S,L3,L2) .
```

#### Przykład

```
?- sublist(S,[a,b,c]) .
```

```
S=[] ;
```

```
S=[a] ;
```

```
S=[a,b] ;
```

```
S=[a,b,c] ;
```

```
S=[b] ;
```

```
...
```

## Wybrane operacje na listach w Prologu

### Operacja generowania permutacji listy

Klauzula **permut**(L1,L2) jest prawdziwa, jeśli lista L2 jest permutacją listy L1.

#### Przykład

```
?- permut([a,b,c],P) .
```

```
P=[a,b,c] ;
```

```
P=[a,c,b] ;
```

```
P=[b,a,c] ;
```

```
P=[b,c,a] ;
```

```
...
```

## Wybrane operacje na listach w Prologu

### Operacja generowania permutacji listy c.d.

Klauzula **permut (L1 , L2)** jest prawdziwa, jeśli lista **L2** jest permutacją listy **L1**.

#### Definicja

*Jeżeli pierwsza lista (L1) jest pusta, to druga lista (L2) również jest pusta.*

*Jeżeli lista L1 jest niepusta, wtedy ma postać [X|L], a jej permutacja powstaje w wyniku permutacji na L i wstawienia do niej X (w dowolne miejsce).*

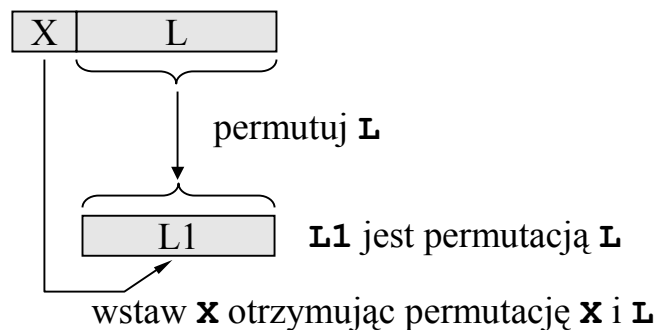
Zapis w Prologu:

```
permut ( [] , [] ) .
```

```
permut ( [X|L] , P ) :- permut ( L , L1 ) ,  
                        insert ( X , L1 , P ) .
```

## Wybrane operacje na listach w Prologu

### Operacja generowania permutacji listy c.d.



## Wybrane operacje na listach w Prologu

### Operacja generowania permutacji listy c.d.

Inna wersja klauzuli **permut (L1, L2)** - z zastosowaniem klauzuli **del**: najpierw usuwamy element, na pozostałej reszcie dokonujemy permutacji i wstawiamy element na początek spermutowanej listy.

Zapis w Prologu:

```
permut ([], []) .  
permut (L, [X|P]) :- del (X, L, L1) ,  
                     permut (L1, P) .
```

Przykład

```
?- permut ([red,blue,green], P) .  
P=[red,blue,green] ;  
P=[red,green,blue] ;  
P=[blue,red,green] ;  
...
```

## Wybrane operacje na listach w Prologu

### Operacja generowania permutacji listy c.d.

**Uwaga!** Operacja permutacji nie jest doskonała (obie wersje) i w pewnych przypadkach może prowadzić do nieskończonych pętli.

*Przykład (prowadzący do błędów)*

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

- dla pierwszej wersji klauzuli **permut** (z **insert**) doprowadzi do zapętlenia, jeśli zażądamy dalszych odpowiedzi po wygenerowaniu wszystkich możliwych permutacji
- dla drugiej wersji (z **del**) - podana zostanie tylko jedna permutacja (pierwotna lista), a próba kontynuacji zakończy się zapętleniem

## Notacja operatorów

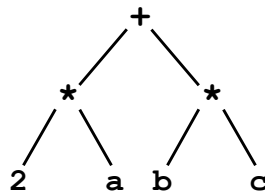
### Standardowa reprezentacja operatorów w Prologu

W Prologu operatory reprezentowane są w notacji *prefiksowej*.

*Przykład*

**$+( * (2, a), * (b, c) )$**

Wyrażenia tworzą w reprezentacji wewnętrznej struktury drzewiaste (są termami), a operatory pełnią rolę funktorów:



## Notacja operatorów

### Standardowa reprezentacja operatorów w Prologu

Dopuszczalna jest alternatywna reprezentacja operatorów w notacji *infiksowej*.

*Przykład*

**$2 * a + b * c$**

Wyrażenia w notacji *infiksowej* przekształcane są automatycznie do postaci *prefiksowej*.

Pierwszeństwo operatorów decyduje o interpretacji wyrażeń w notacji infiksowej.

## Notacja operatorów

### Definiowanie operatorów w Prologu

Prolog dopuszcza możliwość definiowania nowych operatorów. W celu zdefiniowania operatora korzystamy z specjalnej klauzuli systemowej zwanej również *dyrektywą*.

Postać dyrektywy:

**op(<pierwszeństwo>,<składnia>,<symbol>)**

gdzie:

**<pierwszeństwo>** - wartość pierwszeństwa operatora,

**<składnia>** - budowa operatora (prefix, infix, suffix),

**<symbol>** - oznaczenie operatora.

*Przykład*

**?- op(600,xfx,has) .**

Można teraz zdefiniować fakt np.: **piotr has auto.**

## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

Zasady:

- Definicja operatora nie określa żadnej operacji, która będzie wykonywana na argumentach operatora
- Operatory definiowane w Prologu są tylko funktorami, służącymi do konstruowania bardziej złożonych struktur
- Pierwszeństwo operatora może przyjmować wartości z zakresu od **1** do **1200**
- Oznaczenie operatora musi być nazwą

## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

Budowa składniowa operatora - notacje:

- prefiksowa:

**$f x \ f y$**

- infiksowa:

**$x f x \ x f y \ y f x$**

- postfiksowa:

**$x f \ y f$**

gdzie:

**$f$**  - to operator

**$x$**  - to argument o wartości pierwszeństwa  $< f$

**$y$**  - to argument o wartości pierwszeństwa  $\leq f$

## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

Zasady pierwszeństwa argumentów i operatora:

<b>Składnik wyrażenia</b>	<b>Wartość pierwszeństwa</b>
argument prosty	0
argument w nawiasach	0
operator	wg definicji (dyrektywa <b>op</b> )
argument złożony	wg pierwszeństwa operatora (funktora) głównego



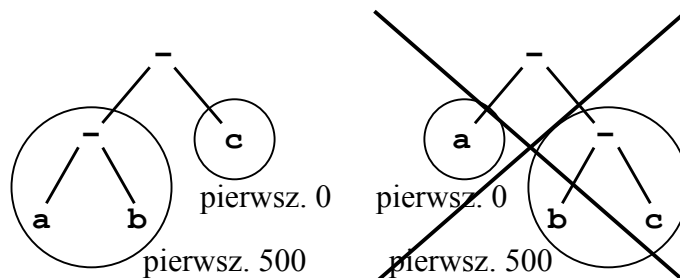
## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

#### *Przykład*

Założmy, że **op**(500, **yfx**, -).

Reprezentacja wyrażenia **a-b-c** to **(a-b)-c**, a nie **a-(b-c)**.



## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

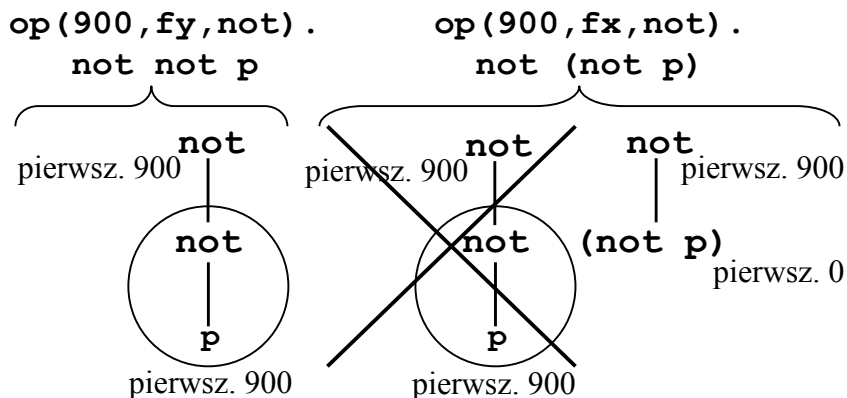
Zasady pierwszeństwa a łączność operatorów:

Notacja	Lewostronna łączność	Prawostronna łączność
prefiksowa	<b>fy</b>	<b>fx</b>
infiksowa	<b>yfx</b>	<b>xfy</b>
postfiksowa	<b>yf</b>	<b>xf</b>

## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

#### *Przykład*



## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

#### *Przykład*

Wyrażenie logiki predykatów (prawo de Morgana'a):

$$\sim (A \ \& \ B) \iff \sim A \vee \sim B$$

Zapis w Prologu (standardowy, prefiksowy):

**equivalence(not (and (A, B) ), or (not (A) , not (B) ) ) .**

Zapis w Prologu (nowe operatory):

?- **op(800, xfx, <=>) .**

?- **op(700, xfy, v) .**

?- **op(600, xfy, &) .**

?- **op(500, fy, ~) .**

Można teraz zdefiniować term: **~(A & B) <=> ~A v ~B.**

## Notacja operatorów

### Definiowanie operatorów w Prologu c.d.

#### *Uwagi*

- Definicje nowych operatorów nie określają żadnych nowych działań na argumentach, lecz wzbogacają notację o nowe sposoby tworzenia złożonych form reprezentacji
- Operator o najwyższej wartości pierwszeństwa jest głównym operatorem wyrażenia złożonego; operatory o niższej wartości pierwszeństwa mają natomiast silniejsze wiązanie
- Specyfika operatora zależy zarówno od jego położenia względem argumentów, jak i od zasad pierwszeństwa operatora oraz jego argumentów

## Operacje arytmetyczne w Prologu

### Operacja przypisania w Prologu

= (znak równości) operacja dopasowania (unifikacji) termów

**is** operacja przypisania (ewaluacji wyrażenia)

#### *Przykład*

?- **X=1+2.**

Odpowiedź:

**X=1+2**

term z funktorem +  
oraz argumentami 1 i 2

?- **X is 1+2.**

Odpowiedź:

**X=3**

predefiniowana procedura  
ewaluacji wyrażenia **is**

## Operacje arytmetyczne w Prologu

### Podstawowe operacje arytmetyczne w Prologu

Oznaczenie operatora	Operacja arytmetyczna
<b>+</b>	dodawanie
<b>-</b>	odejmowanie
<b>*</b>	mnożenie
<b>/</b>	dzielenie rzeczywiste
<b>//</b>	część całkowita z dzielenia
<b>mod</b>	reszta całkowita z dzielenia
<b>rem</b>	część ułamkowa z dzielenia
<b>**</b> lub <b>^</b>	potęgowanie

## Operacje arytmetyczne w Prologu

### Podstawowe operacje porównania w Prologu

Oznaczenie operatora	Operacja porównania
<b>&gt;</b>	wiekszy
<b>&lt;</b>	mniejszy
<b>&gt;=</b>	wiekszy lub równy
<b>=&lt;</b>	mniejszy lub równy
<b>=:=</b>	równy
<b>=\=</b>	różny
<b>\=</b>	termy się nie unifikują

## Operacje arytmetyczne w Prologu

### Operacja równości ( $=$ : $=$ ) a unifikacja ( $=$ )

Operator  $=$  sprawdza dopasowanie dwóch obiektów i jeśli jest ono możliwe, prowadzi do wiązania zmiennych w tych obiektach (bez obliczania wartości wyrażeń!).

Operator  $=$  :  $=$  powoduje obliczenie wartości argumentów bez wiązania zmiennych.

#### *Przykład*

?-  $1+2=$  :  $=2+1$  .

**Yes**

?-  $1+2=2+1$  .

**No**

?-  $1+A=B+2$  .

**A=2**

**B=1**

## Operacje arytmetyczne w Prologu

### Zastosowanie operacji arytmetycznych

#### *Przykład*

Największy wspólny dzielnik dwóch liczb: dla dwóch liczb całkowitych, dodatnich  $X$  i  $Y$  , największy wspólny dzielnik  $D$ :

- równa się  $X$ , jeżeli  $X$  i  $Y$  są równe,
- równa się największemu wspólnemu dzielnikowi  $X$  i  $Y-X$ , jeżeli  $X < Y$ ,
- równa się największemu wspólnemu dzielnikowi  $Y$  i  $X-Y$ , jeżeli  $Y < X$ .

Zapis w Prologu:

**nwd( $X$ , $X$ , $X$ ) .**

**nwd( $X$ , $Y$ , $D$ ) :-  $X < Y$ ,  $Y1$  is  $Y-X$ , nwd( $X$ , $Y1$ , $D$ ) .**

**nwd( $X$ , $Y$ , $D$ ) :-  $Y < X$ , nwd( $Y$ , $X$ , $D$ ) . lub**

**nwd( $X$ , $Y$ , $D$ ) :-  $Y < X$ ,  $X1$  is  $X-Y$ , nwd( $X1$ , $Y$ , $D$ ) .**

## Operacje arytmetyczne w Prologu

Zastosowanie operacji arytmetycznych c.d.

*Przykład*

Wyznaczyć długość listy termów. Długość listy:

- równa się 0, jeżeli lista jest pusta,
- równa się długości jej ogona powiększonej o 1, jeżeli lista jest niepusta.

Zapis w Prologu:

```
length([],0).  
length([_|Tail],N):- length(Tail,N1),  
                      N is 1+N1.
```

## Operacje arytmetyczne w Prologu

Zastosowanie operacji arytmetycznych c.d.

*ciąg dalszy przykładu* - inny (błędny) zapis w Prologu:

```
length([],0).  
length([_|Tail],N):- length(Tail,N1),  
                      N = 1+N1.
```

↑  
unifikacja!

Zapytanie: ?- length([a,b,[c,d],e],N).

N = (1+(1+(1+(1+0))))

Właściwe zapytanie (w tej wersji length):

?- length([a,b,[c,d],e],N), L is N.

N=(1+(1+(1+(1+0))))

L=4



## Operacje arytmetyczne w Prologu

Zastosowanie operacji arytmetycznych c.d.

### Podsumowanie

- Operacje z użyciem operatorów arytmetycznych wymagają zastosowania predefiniowanych procedur ewaluacji wyrażenia
- Wykonanie operacji arytmetycznej jest możliwe jedynie po zastosowaniu procedury **is**
- W trakcie ewaluacji wyrażenia wszystkie argumenty muszą mieć przypisaną (związaną) wartość liczbową
- Operatory porównania również prowadzą do ewaluacji porównywanych wyrażeń