



# 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



## Operacje wejścia/wyjścia w Prologu

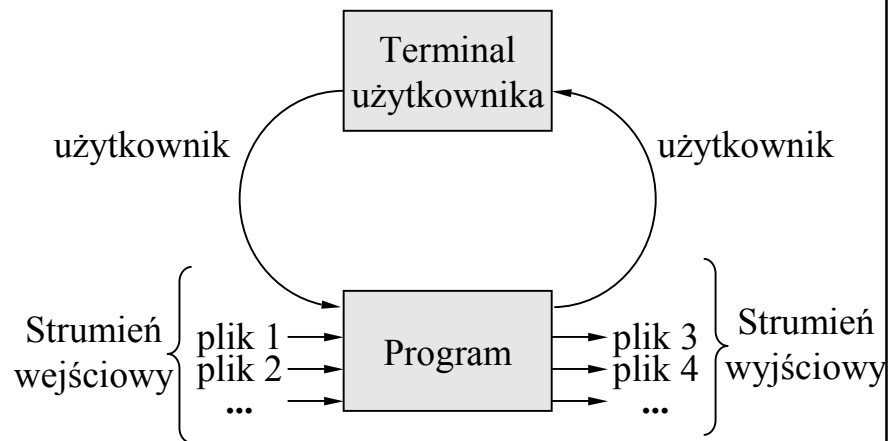
- Operacje na plikach
- Przetwarzanie plików termów
- Manipulowanie danymi znakowymi
- Kompozycja i dekompozycja atomów
- Wczytywanie programów prologowych:  
*consult* i *reconsult*



## Operacje na plikach

- W języku Prolog operacje na plikach opierają się na koncepcji *strumienia*
- Program prologowy czyta dane ze strumieni wejściowych i zapisuje dane do strumieni wyjściowych
- Strumieniem wejściowym i/lub wyjściowym może być dowolny plik
- Terminal użytkownika jest również traktowany jak strumień
- W trakcie wykonywania programu w danej chwili realizowana może być operacja odczytu i zapisu odpowiednio z/do jednego strumienia wejściowego i jednego strumienia wyjściowego
- Domyślnym strumieniem wejściowym i wyjściowym jest terminal użytkownika

## Operacje na plikach



## Operacje na plikach

### Otwieranie plików

Operacja zmiany aktualnego *strumienia wejściowego*

**see (<nazwa\_pliku>) .**

jeżeli plik jest już otwarty to nadal pozostanie w trybie odczytu  
(nie będzie błędu!)

Operacja zmiany aktualnego *strumienia wyjściowego*

**tell (<nazwa\_pliku>) .**

jeżeli plik jest już otwarty to nadal pozostanie w trybie zapisu  
(nie będzie błędu!)

## Operacje na plikach

### Zamykanie plików

Operacja zamknięcia aktualnego *strumienia wejściowego*

**seen .**

Cel ten jest zawsze spełniony. Po wykonaniu strumieniem wejściowym zostaje terminal.

Operacja zamknięcia aktualnego *strumienia wyjściowego*

**told .**

Cel ten jest zawsze spełniony. Po wykonaniu strumieniem wyjściowym zostaje terminal.

## Operacje na plikach

### Identyfikacja strumieni

Operacja identyfikacji aktualnego *strumienia wejściowego*

**seeing (Str) .**

Zmienna **Str** jest unifikowana z identyfikatorem strumienia (wygenerowanym automatycznie przez system).

Operacja identyfikacji aktualnego *strumienia wyjściowego*

**telling (Str) .**

Zmienna **Str** jest unifikowana z identyfikatorem strumienia (wygenerowanym automatycznie przez system).

## Operacje na plikach

Operacje otwarcia i zamknięcia (**see**, **tell**, **seen**, **told**) służą do przetwarzania plików sekwencyjnych (tekstowych).

Podczas przetwarzania plików sekwencyjnych każda operacja (zapis/odczyt) powoduje automatyczne przejście do następnej pozycji w pliku.

Osiągnięcie końca pliku sygnalizowane jest specjalną wartością.

Istnieją dwa warianty plików tekstowych:

- pliki znakowe (plik składa się z pojedynczych bajtów/znaków)
- pliki termów (podstawowym składnikiem pliku jest term)

## Operacje na plikach

Operacje odczytu i zapisu dla plików znakowych:

**get (X)** - odczyt pojedynczego, niepustego znaku z akt. str.

**get\_byte (X)** - odczyt jednego bajtu z akt. strumienia

**put (X)** - zapis pojedynczego znaku do akt. strumienia

Osiągnięcie końca pliku sygnalizowane jest wartością -1.

Operacje odczytu i zapisu dla plików termów:

**read (X)** - odczyt pojedynczego termu z akt. strumienia

**write (X)** - zapis pojedynczego termu do akt. strumienia

Osiągnięcie końca pliku sygnalizowane jest predefiniowanym atomem **end\_of\_file**.

Pliki termów muszą spełniać wymogi składni termów języka Prolog.



## Przetwarzanie plików termów

### Operacja odczytu termu ze strumienia wejściowego

Cel **read (X)** oznacza odczyt pojedynczego termu z aktualnego strumienia wejściowego i unifikację termu ze zmienną **X**.

Brak dopasowania dla argumentu **X** klauzuli **read**, nie będącego zmienną, doprowadzi do błędu (nie nastąpi nawrót w celu odczytania następnego termu).

Termy zawarte w pliku muszą być oddzielone znakiem kropki, odstępu lub przejścia do nowej linii.

Argument **X** przyjmie wartość **end\_of\_file** kiedy zostanie osiągnięty koniec pliku.

Pełna wersja operacji odczytu pozwala również wskazywać jakiego strumienia (**Str**) dotyczy operacja: **read (Str, X)**.



## Przetwarzanie plików termów

### Operacja zapisu termu do strumienia wyjściowego

Cel **write (X)** oznacza zapis pojedynczego termu **X** do aktualnego strumienia wyjściowego.

Term **X** zostanie zapisany do pliku skojarzonego ze strumieniem wyjściowym w formie identycznej z wykorzystywaną dla domyślnego strumienia wyjściowego (terminala).

Termy zapisywane do pliku mogą mieć dowolny stopień złożoności (zagnieżdżenia).

Pełna wersja operacji zapisu pozwala również wskazywać jakiego strumienia (**Str**) dotyczy operacja: **write (Str, X)**.

## Przetwarzanie plików termów

### Pozostałe operacje zapisu do strumienia wyjściowego

Predykat **append (X)** otwiera plik **X** w trybie dopisywania i wiąże go z aktualnym strumieniem wyjściowym.

Cel **tab (N)** oznacza zapis **N** znaków odstępu do aktualnego strumienia wyjściowego, przy czym **N** musi być większe od 0.

Predykat **n1** (bezargumentowy) powoduje zapis znaku nowego wiersza (przejścia do nowej linii) do pliku skojarzonego z aktualnym strumieniem wyjściowym.

## Przetwarzanie plików termów

### Typowy schemat przetwarzania plików tekstowych

Odczyt:

```
... ,  
seeing (Old) ,  
see (' dane.txt' ) ,  
... ,  
read (X) ,  
... ,  
seen ,  
see (Old) ,  
... ,
```

Zapis:

```
... ,  
telling (Old) ,  
tell (' wyniki.txt' ) ,  
... ,  
write (X) ,  
... ,  
told ,  
tell (Old) ,  
... ,
```

## Przetwarzanie plików termów

### Przykład przetwarzania plików tekstowych - standardowe we/wy

Mamy daną klauzulę obliczanie sześcianu liczb:

```
cube(X,N):- N is X*X*X.
```

Interaktywne wykonanie tej operacji dla ciągu liczb tzn. bez konieczności jawnego podawania celu dla kolejnej wartości:

```
cube:- read(X) , process(X) .
```

```
process(stop):- !.
```

```
process(X):- N is X*X*X, write(N) , cube.
```

```
?- cube.
```

```
|:2.
```

```
8
```

```
|:5.
```

```
125
```

```
|:stop.
```

```
Yes
```

} Przykład zastosowania

## Przetwarzanie plików termów

### Przykład przetwarzania plików tekstowych c.d. - standardowe we/wy

Próbna wersja „skrótowa”:

```
cube:- read(stop),!.
```

```
cube:- read(X) , N is X*X*X, write(N) , cube.
```

Wyniki:

```
?- cube.
```

```
|:2. ←
```

Dane są tracone, bo  
brak unifikacji dla **read(stop)!**

```
|:stop. ←
```

Po unifikacji **read(X)** błąd typu  
w **N is X\*X\*X** dla **stop**

```
ERROR: Arithmetic:'stop/0' is not a function
```



## Przetwarzanie plików termów

### Przykład przetwarzania plików tekstowych c.d.

Wersja „user-friendly”:

```
cube:- write('Podaj liczbę: '),
        read(X), process(X).
process(stop):- !.
process(X):- N is X*X*X,
              write('Sześćcian '),write(X),
              write(' wynosi '), write(N),nl,
              cube.
```

```
?- cube.
```

```
Podaj liczbę: 2.
```

```
Sześćcian 2 wynosi 8
```

```
Podaj liczbę: stop.
```

```
Yes
```

} Przykład zastosowania

## Przetwarzanie plików termów

### Przykład przetwarzania plików tekstowych dla list

Klauzula **writelist(L)** wyświetla listę **L** na ekranie w taki sposób, że każdy term pojawia się w kolejnym wierszu:

```
writelist([]).
writelist([H|T]):- write(H), nl,
                   writelist(T).
```

Jeżeli składowymi listy są *tylko* listy, to wszystkie elementy jednej listy powinny być w jednym wierszu:

```
writelist2([]).
writelist2([H|L]):-
                    doline(H),nl,writelist2(L).
doline([]).
doline([H|L]):- write(H), tab(1),doline(L).
```

## Przetwarzanie plików termów

### Przykład przetwarzania plików tekstowych dla list

Klauzula **bar (L)** wyświetla listę **L** zawierającą tylko liczby całkowite jako poziomy „wykres słupkowy” .

```
bars([]) .  
bars([H|T]) :- stars(H),nl,bars(T) .  
stars(N) :- N>0,write(*),N1 is N-1,stars(N1) .  
stars(N) :- N=<0 .
```

Przykładowe wyniki:

```
?- bars([3,4,6,5]) .  
***  
****  
*****  
*****
```

## Przetwarzanie plików termów

### Formatowanie napisów dla termów

Złożone (zagnieżdżone) termy mogą być nieczytelne.

```
family(person(tom,fox,date(7,may,56)),  
        person(ann,fox,date(9,may,60)),  
        [person(pat,fox,date(4,june,80)),  
         person(jim,fox,date(9,july,81))]) .
```

Klauzula **wfamily(F)** wyświetla dane z termu **F** w czytelny sposób.

```
wfamily(family(H,W,Ch)) :-  
    nl,write(parents),nl,nl,  
    writeperson(H),nl,  
    writeperson(W),nl,nl,  
    write(children),nl,nl, writepersonlist(Ch)
```

## Przetwarzanie plików termów

### Formatowanie napisów dla termów c.d.

Kontynuacja klauzuli **wfamily(F)** :

```
writeperson(person(N,SN,date(D,M,Y))):-  
    tab(4),write(N), tab(1),write(SN),  
    write(',born '),  
    write(D),tab(1),  
    write(M),tab(1),  
    write(Y).
```

```
writepersonlist([]).  
writepersonlist([H|T]):-  
    writeperson(H),nl,  
    writepersonlist(T).
```

## Przetwarzanie plików termów

### Formatowanie napisów dla termów c.d.

Rezultat klauzuli **wfamily(F)** :

**parents**

```
tom fox born, 7 may 56  
ann fox born, 9 may 60
```

**children**

```
pat fox born, 4 june 80  
jim fox born, 9 july 81
```

## Przetwarzanie plików termów

### Przetwarzanie plików z termami

Typowy schemat przetwarzania plików termów:

```
procfile:- read(Term) , process(Term) .  
process(end_of_file):- !.  
process(Term):- treat(Term) ,procfile.
```

Przykład przetwarzania pliku termów - wyświetlanie zawartości pliku z numerowaniem termów

```
showfile(N):- read(Term) , show(Term,N) .  
show(end_of_file,_):- !.  
show(Term,N):- write(N) , tab(2) ,  
                write(Term) ,nl,  
                N1 is N+1 , showfile(N1) .
```

## Przetwarzanie plików termów

### Przetwarzanie plików z termami - przykład

Katalog towarów zawarty jest w pliku składającym się z termów postaci:

```
item(numer, opis, cena, dostawca)
```

Chcemy wygenerować nowy plik zawierający tylko towary od jednego wyznaczonego dostawcy **Sup**. Przetwarzanie wymaga odczytu z jednego pliku np. **dane** i zapisu do drugiego pliku np. **wyniki**.

Zapytanie celu głównego:

```
?- see(dane) , tell(wyniki) , Sup=lloyd,  
   makefile(Sup) , told, seen.
```

## Przetwarzanie plików termów

### Przetwarzanie plików z termami

Klauzula **makefile(Sup)** wymaga podania nazwy dostawcy:

```
makefile(Sup) :- write(Sup), write(' '),  
                 nl, makerest(Sup).  
makerest(Sup) :- read(Item), proc(Item, Sup).  
proc(end_of_file, _) :- !.  
proc(item(N, D, P, Sup), Sup) :- !,  
    write(item(N, D, P)), write(' '), nl,  
    makerest(Sup).  
proc(_, Sup) :- makerest(Sup).
```

Znak kropki jest dopisywany na końcu każdego termu w pliku wynikowym w celu umożliwienia ich ewentualnego odczytu za pomocą predykatu **read**.

## Manipulowanie danymi znakowymi

### Operacje odczytu i zapisu znaku do strumienia wyjściowego

Cel **get0(X)** oznacza odczyt pojedynczego znaku z aktualnego strumienia wejściowego i unifikację jego kodu ASCII ze zmienną **X**.

Cel **get(X)** jest wariantem klauzuli **get** przeznaczonym do odczytu niepustych znaków kodu ASCII - znaki tzw. białe są pomijane.

Cel **put(X)** oznacza zapis pojedynczego kodu ASCII znaku **X** do aktualnego strumienia wyjściowego.

## Manipulowanie danymi znakowymi

### Przykład przetwarzania plików znakowych

Usuwanie nadmiarowych znaków odstepu z wczytanego zdania wejściowego - predykat **squeeze(X)**. Zdanie musi być zakończone znakiem kropki.

Przykład zastosowania

```
?- Robot wylał wodę na parapet.  
Robot wylał wodę na parapet  
Yes
```

Definicja:

```
squeeze:- get0(C),put(C),do_rest(C).  
do_rest(46):- !. %znak kropki-koniec  
do_rest(32):- !,get(C),put(C),do_rest(C).  
do_rest(C):- squeeze.
```

## Kompozycja i dekompozycja atomów

Predykat systemowy **name(A,S)** służy do przekształcania wczytanej informacji reprezentowanej w postaci listy kodów znaków (**S**) w stałą symboliczną lub liczbę (**A**). Konwersji można dokonywać w obie strony.

Przykłady

```
?- name(kot,X).  
X=[107,111,116]  
?- name(Y,[112,105,101,115])  
Y=pies
```

Alternatywna reprezentacja list kodów znaków w Prologu - napis ograniczony znakami cudzysłowu

```
?- [112,105,101,115]="pies".  
Yes  
?- name(Y,"pies").  
Y=pies
```



## Kompozycja i dekompozycja atomów

Zastosowanie predykatu **name**

Przetwarzanie identyfikatorów numerowanych np. postaci:

**item1, item2, ..., itemk, ...**

Predykat **item(X)** ma służyć do sprawdzania, czy mamy do czynienia z właściwym identyfikatorem:

```
item(X) :- name(X,Xlist) ,  
            name(item,T) ,  
            conc(T,_,Xlist) .
```

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

```
conc([H|T],L,[H|T2]) :- conc(T,L,T2) .
```



## Kompozycja i dekompozycja atomów

Zastosowanie predykatu **name** c.d.:

Przekształcanie napisów języka naturalnego w następującą reprezentację wewnętrzną:

- każdy pojedynczy napis jest atomem
- całe zdanie jest listą atomów

Zdanie w języku naturalnym - dane wejściowe:

Tomek był zadowolony z postępów robota.

Rezultat - dane wyjściowe:

**[Tomek,był,zadowolony,z,postępów,robota]**



## Kompozycja i dekompozycja atomów

Zastosowanie predykatu **name** c.d.:

Program:

```
getsntc(Wlist):- get0(C),getrest(C,Wlist).
```

```
getrest(46,[]):- !. %znak końcowy - kropka
```

```
getrest(32,Wlist):- !,getsntc(Wlist).
```

```
getrest(L,[W|Wlist]):- getlttrs(L,Ls,Next),  
    name(W,Ls),getrest(Next,Wlist).
```

```
getlttrs(46,[],46):- !.
```

```
getlttrs(32,[],32):- !.
```

```
getlttrs(L,[L|Ls],Next):- get0(C),  
    getlttrs(C,Ls,Next).
```



## Kompozycja i dekompozycja atomów

Zastosowanie predykatu **name** c.d.:

*Komentarz*

W predykanie **getrest** rozważamy trzy przypadki:

**L** - jest znakiem końca (kropką)

**L** - jest znakiem białym: pomijamy go i **getsntc** dla reszty

**L** - jest literą: najpierw odczyt całego słowa **W**,  
rozpoczynającego się od **L**, potem reszta zdania **Wlist** za  
pomocą **getsntc**, a wynik skumulowany w **[W|Wlist]**

```
getrest(46,[]):- !. %znak końcowy - kropka
```

```
getrest(32,Wlist):- !,getsntc(Wlist).
```

```
getrest(L,[W|Wlist]):- getlttrs(L,Ls,Next),  
    name(W,Ls),getrest(Next,Wlist).
```



## Kompozycja i dekompozycja atomów

Zastosowanie predykatu **name** c.d.:

### *Komentarz*

W predykanie **getlttrs**(**L**,**Ls**,**Nc**) argumenty oznaczają:

**L** - aktualny znak (już wczytany) czytanego słowa

**Ls** - lista znaków (zaczynająca się od **L**) do końca słowa

**Nc** - znak, który znajduje się bezpośrednio za wczytanym słowem; musi być znakiem białym

```
getlttrs(46, [], 46) :- !.
```

```
getlttrs(32, [], 32) :- !.
```

```
getlttrs(L, [L|Ls], Next) :- get0(C),  
    getlttrs(C, Ls, Next).
```

## Wczytywanie programów prologowych: *consult* i *reconsult*

W języku Prolog wczytywanie plików tekstowych zawierających program odbywa się na dwa sposoby:

- predykat **consult**(**F**) łąduje wszystkie klauzule z pliku **F**, które następnie są wykorzystywane do osiągnięcia zadanego celu; ponowne wykonanie tej klauzuli spowoduje dopisanie nowych klauzul do już istniejących
- predykat **reconsult**(**F**) działa podobnie; jedynie w przypadku wystąpienia w pliku **F** klauzul relacji zawartych już w pamięci nastąpi ich redefinicja; pozostałe klauzule pozostaną nie zmienione