

# **Loogiline programmeerimine**

Tõnu Tamme

# 1. loeng

## Prolog: faktid, reeglid ja päringud

- ▶ Kursuse ülesehitus
- ▶ Mis on loogiline programmeerimine?
- ▶ Mis on Prolog?
- ▶ Faktid, reeglid ja päringud
- ▶ Tunnikontroll

# Kursuse ülesehitus



loeng	2	10
tunnikontroll	2	10
ülesanne	5	15
koduülesanne	5	15
eksam	50	50

# Imperatiivne ja deklarativne programmeerimine

- ▶ Imperatiivne: kuidas?
- ▶ Deklaratiivne: mida?
  - ▶ Funktsionaalne: funktsioon tagastab erinevaid väärtusi (n bitti)

128, -0.0053, 'abcd', , (1 + 2)/3, true

- ▶ Loogiline: lause on kas tõene või väär (1 bitt)

true, false

# Kus Prologi kasutatakse?

- ▶ keeletehnoloogia
- ▶ tehisintellekt
- ▶ Windows NT (*David Hovel*)
- ▶ ressursside jaotamine, paigutusülesanded
- ▶ ekspertsüsteemid
- ▶ kitsendustega programmeerimine
- ▶ paralleelarvutused

# Mis on loogiline programmeerimine?

- ▶ **Laias mõttes**

- ▶ iga programm sisaldab loogikat
- ▶ Robert Kowalski: *algorithm = logic + control*
- ▶ Niklaus Wirth: *program = algorithm + data structures*

- ▶ **Java näide:**





```
// Programm, mis kirjutab Gaudeamuse kaks esimest rida.  
class Gaudeamus {  
    public static void main(String[] args) {  
        System.out.println("Gaudeamus igitur,");  
        System.out.println("juvenes dum sumus!");  
    }  
}
```

- ▶ **Python:**

```
print "Gaudeamus igitur,"  
print "juvenes dum sumus!"
```



# Loogiline programmeerimine tavalises mõttes:

## 1. järku loogika

1)  on lind  $\Rightarrow$     lendab

2)  on lind  
part

Musta kasti võime asendada pardiga

 on lind  $\Rightarrow$   lendab  
part part

Järeldus:  lendab  
part

$\forall x (lind(x) \supset lendab(x))$

$lind(part)$

Järeldus:  $lendab(part)$

# Loogiline programmeerimine tavalises mõttes:

## 1. järku loogika (formaalsemalt)

- ▶ Funktsionaalsümbol:  $f, g, +, *$ .
- ▶ Term:
  - ▶ Konstant:  $a, b, true, 12, 'Tekst' \dots$
  - ▶ Muutuja:  $x, y, z, \dots$
  - ▶ Avaldised (funktsioonide rakendamine):  
Kui  $t_1, \dots, t_n$  on termid ja  $f$  on funktsionaalsümbol, siis  $f(t_1, \dots, t_n)$  on term.

$$f(x), x + 4, 2, (2 * y) - 1$$

- ▶ Predikaatsümbol:  $p, q, r$ .
- ▶ Atomaarne valem: saab olla tõene või väär:  
Kui  $t_1, \dots, t_n$  on termid ja  $p$  on predikaatsümbol, siis  $p(t_1, \dots, t_n)$  on atomaarne valem.

$$p(x), lendab(lind), ema(x, y)$$

- ▶ Tehted atomaarsete valemitega:  $\neg$  &  $\vee \implies \equiv$
- ▶ Kvantorid:  $\forall, \exists$



# Loogiline programmeerimine tavalises mõttes

- ▶ 1. järku loogika

tehted:  $\neg$  &  $\vee$   $\supset$   $\equiv$   $\forall$   $\exists$

$$\forall x (lind(x) \supset lendab(x))$$

$lind(\text{)$

- ▶ erijuhud

$$\neg lind(x) \vee lendab(x)$$

- ▶ laiendused

# Loogiline programmeerimine kitsas mõttes

- ▶ Horni loogika

$lendab(x) \leftarrow lind(x)$

$lind(\text{🦆}) \leftarrow$

*Kuni 1 järelalus*

$\leftarrow lendab(\text{🦆})$

- ▶ Prolog

$lendab(X) : \neg lind(X).$

$lind(part).$

$? \neg lendab(part).$

- ▶ Datalog

*Puuduvad funktsioonid*

# Prologi süntaks

- ▶ **aatom**: mittearvuline konstant, funktsionaalsümbol, predikaatsümbol  
*algab väiketähega*: lind, tweety, p2r  
*apostroofides*: 'Tweety'  
*kirjavahemärgid*: =?=:
  - ▶ +, \*, – on funktsionaalsümbolid
  - ▶ =, < on predikaatsümbolid
- ▶ **arv**: arvuline konstant  
*algab numbriga*: 25, 2pi, 2.73, 2.0E4
- ▶ **muutuja**  
*algab suurtähega*: X, Lind, \_Lind  
*nimetu muutuja*: \_
- ▶ **kommentaar**  
*algab protsendimärgiga*: %kuni rea lõpuni  
*sulgudes*: /\*kommentaar\*/

# Prologi süntaks (loogika)

eitus ( $\neg a$ )	$\backslash + a$
konjunktsioon ( $a \& b$ )	$a, b$
disjunktsioon ( $a \vee b$ )	$a ; b$
implikatsioon ( $a \implies b$ )	$b : -a$ $a -> b$
Valemi lõpp	.

$\equiv$  Pole kasutusel

$\forall$  Eraldi tähistust ei ole, kuid muutujad faktides ja reeglites on vaikimisi seotud üldisuskvantoriga. Järgmine väide kehtib iga  $X$  korral.

$$lendab(X) : -lind(X).$$

$\exists$  Eraldi tähistust ei ole, kuid muutujad päringutes on vaikimisi seotud olemasolukvantoriga. Järgmine päring küsib, kas leidub selline  $X$ , mis on lind.

$$? - lind(X).$$

# Herbrandi teoreem

## Teoreem

*Disjunktide hulk  $S$  on vastuoluline parajasti siis, kui eksisteerib lõplik vastuoluline muutujaid mittesisaldavate disjunktide hulk  $S_{\sigma_1} \cup \dots \cup S_{\sigma_n}$ , kus iga substitutsioon  $\sigma_i$  asendab kõik hulgas  $S$  esinevad muutujad termidega hulga  $S$  Herbrandi universumist.*

## Definitsioon

Disjunktide hulga  $S$  Herbrandi universumiks nimetatakse kõigi selliste muutujaid mittesisaldavate termide hulka, mida saab moodustada hulgas  $S$  sisalduvatest konstantidest ja funktsionaalsümbolitest.

Kui hulgas  $S$  konstante pole, lubatakse kasutada suvalist konstanti  $a$ .

## Näide

$\neg \text{lind}(\text{🦗}) \vee \text{lendab}(\text{🦗})$

$\text{lind}(\text{🦗})$

$\neg \text{lendab}(\text{🦗})$

# Loogilise programmeerimise ajalugu

- ▶ 1879 Frege
- ▶ 1929 Herbrand, Gödel
- ▶ 1960 Prawitz
- ▶ 1965 Alan Robinson
- ▶ 1970 Kowalski, Kuehner
- ▶ 1970 Alain Colmerauer Q-süsteemid
- ▶ 1972 Marseille Prolog 0
- ▶ 1973 *Prolog* 1
- ▶ 1977 Edinburghi Prolog
- ▶ 1981 Jaapani viienda põlvkonna arvuti projekt
- ▶ 1982 WAM

# Mis on Prolog?

- ▶ programmeerimine Horni loogikas
- ▶ programmeerimine Herbrandi termide universumis
- ▶ lisavahendid
  - ▶ loogilised
  - ▶ loogika välised

# Prologi plussid ja miinused

- + väike ressursivajadus
- + enamusel platvormidel
- + lihtne saada nõu
- eeldab matemaatilist taiplikkust
- erineb teistest keeltest
- täitmine raskesti jälgitav



# Kuidas Prologi nimetatakse?

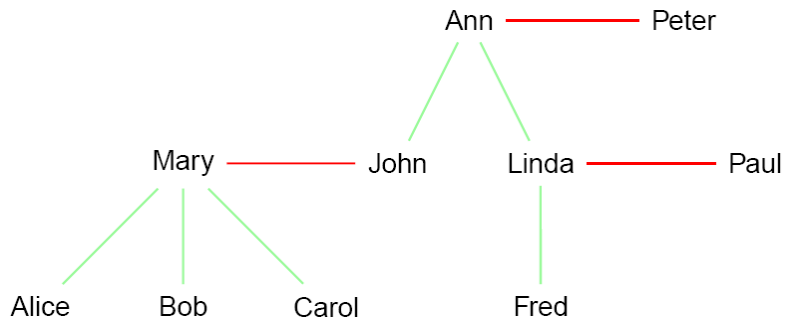
- ▶ SWI-Prolog
- ▶ GNU Prolog
- ▶ Sicstus Prolog
- ▶ ECLiPSe
- ▶ Visual Prolog ?
- ▶ *Edinburghi Prolog*
- ▶ *Marseille Prolog*
- ▶ *ISO Prolog*

# Kuidas Prolog töötab?

## ► Prologi käsud

```
?-halt.  
?-write(tere), nl.  
?-[tere].  
?-listing.  
?-listing(member/2).  
?-edit(tere).  
?-help.  
?-help(is).
```

# Joonis



# Faktid

```
married(mary, john) .  
married(linda, paul) .  
married(ann, peter) .
```

```
mother(john, ann) .      mother(linda, ann) .  
mother(alice, mary) .    mother(bob, mary) .  
mother(carol, mary) .  
mother(fred, linda) .
```

```
female(mary) .      female(alice) .      female(carol) .  
female(ann) .        female(linda) .  
male(john) .          male(bob) .          male(peter) .  
male(paul) .          male(fred) .
```

# Päringud

- ▶ `?-mother(alice,mary) .`

Yes

*Kas Alice ema on Mary?*

- ▶ `?-mother(alice,Kes) .`

Kes = mary

Yes

*Kes on Alice ema?*

Kes = mary ;

No

- ▶ `?-married(mary,_) .`

*Kas Mary on abielus?*

- ▶ `?-mother(Kelle,mary) .`

*Kelle ema on Mary?*

Kelle = alice ;

Kelle = bob ;

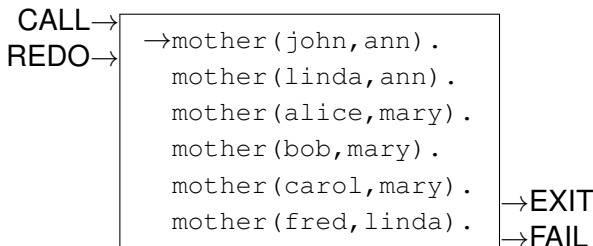
Kelle = carol ;

No

# Päringu täitmine

Byrdi kastimudel (*Byrd's box model*)

Näide: päringu `?-mother(alice, Kes) .` täitmine.



# Liitpäringud

```
?-mother(alice,Mother), married(Mother,Father).  
    {Mother=mary}                % unifitseerimine  
?-married(mary,Father).          % uus päring  
    {Father=john}                % unifitseerimine  
?-                                % tühipäring  
Father = john                    % arvutatud vastus
```

# Liitpäringu täitmine

CALL →  
REDO →

<pre>mother(john, ann) . mother(linda, ann) . →mother(alice, mary) . mother(bob, mary) . mother(carol, mary) . mother(fred, linda) .</pre>
--

→EXIT  
→FAIL

CALL →  
REDO →

<pre>→married(mary, john) . married(linda, paul) . married(ann, peter) .</pre>
--

→EXIT  
→FAIL



## Liitpäringu täitmine (2)

```
?-trace.  
?-trace, mother(alice,M), married(M,Father).  
Call: ( 7) mother(alice, _L119) ? creep  
Exit: ( 7) mother(alice, mary) ? creep  
Call: ( 7) married(mary, _G289) ? creep  
Exit: ( 7) married(mary, john) ? creep  
Father = john ;  
no
```

# Reeglid

```
father(Child,Father):-  
    mother(Child,Mother),  
    married(Mother,Father),  
    male(Father).
```

```
father(Father):-father(_Child,Father).
```

## 2. loeng

# Aritmeetika

- ▶ Sisseehitatud aritmeetika
- ▶ Unifitseerimine
- ▶ Peano aritmeetika
- ▶ Resolutsioon
- ▶ Täielikkus
- ▶ Tunnikontroll

# Sisseehitatud aritmeetika

Tehted:  $+$   $-$   $*$   $/$   $//$  *mod*  $**$   $^$

Võrdlused: *is*  $:=$   $<$   $=<$   $>$   $>=$   $=\backslash=$

## Erinevad võrdlused

- ▶ `?-2+3=3+2.`  
`false.`
- ▶ `?-2+3==3+2.`  
`false.`
- ▶ `?-2+3 is 3+2.`  
`false.`
- ▶ `?-2+3:=3+2.`  
`true.`
- ▶ `?-X=Y.`  
`X = Y.`
- ▶ `?-X==Y.`  
`false.`

# Summeerimine

```
sum(X,Y,Z):-Z is X+Y.
```

```
?-sum(1,2,X).
```

```
X = 3
```

```
?-sum(X,2,3).
```

```
ERROR: Arguments are not sufficiently instantiated
```

```
?-sum(X,Y,3).
```

```
ERROR: Arguments are not sufficiently instantiated
```

# Unifitseerimine

1) **X** lendab,  ja **Y** on linnud  
pingviin

2)  lendab, **Z** ja  on linnud  
part part

Mis väärtustustel 1) = 2) ?

$\left\{ \begin{array}{l} \mathbf{X} / \text{part}, \mathbf{Y} / \text{part}, \mathbf{Z} / \text{pingviin} \end{array} \right\}$

 lendab,  ja  on linnud  
part pingviin part

# Unifitseerimine (formaalselt)

Termide võrdsustamine:  $f(x, g(b)) = f(a, y)$ .

Robinsoni determineeritud unifitseerimisalgoritm.

1.  $\theta := \{\}$ .
2. Leia termide  $s\theta$  ja  $t\theta$  vasakult esimene erinevus.
3. Konstrueeri erinevust põhjustavatest alamtermidest ebakõlapaar  $(w, u)$ .
4. Ebakõlapaar on lihtne siis, kui see on pööratav kujule  $(x, v)$ , kus  $x$  on muutuja ja  $v$  on term, mis ei sisalda muutujat  $x$ .
5.  $\theta := \theta\{x/v\}$ .
6. Kui  $s\theta = t\theta$ , siis termid on unifitseeritavad.
7. Kui ebakõlapaar pole lihtne, siis termid pole unifitseeritavad.
8. Korda samme 2–8.

# Unifitseerimine (järg)

Näide

$$f(x, g(b)) = f(a, y)$$

$$\{x/a\}$$

$$f(a, g(b)) = f(a, y)$$

$$\{y/g(b)\}$$

$$f(a, g(b)) = f(a, g(b))$$

Termide unifikaatori üldkuju (*mgu*)

$$\theta = \{x/a\}\{y/g(b)\} = \{x/a, y/g(b)\}$$

Küsimus: millised termid ei unifitseeru?



# Unifitseerimisteoreem

## Teoreem

*Olgu  $S$  termide lõplik hulk. Kui  $S$  on unifitseeritav, siis unifitseerimisalgoritm lõpetab töö ja annab  $S$  unifikaatori üldkuju.*

*Kui  $S$  pole unifitseeritav, siis unifitseerimisalgoritm lõpetab töö ja sedastab seda fakti.*

Saame, et unifitseerimisalgoritm tagastab alati kindla vastuse jah/ei.

# Kuidas kirjeldada naturaalarve?

Dedekindi lahendus:

$$1) \boxed{x} \text{ on naturaalarv} \implies \boxed{x} + 1 \text{ on naturaalarv}$$

$$2) 0 \text{ on naturaalarv}$$

$$\mathbb{N} := \{0, 0+1, 0+1+1, \dots\}$$

Siin tegelikult kasutatakse üheargumendilist funktsiooni '+1'. Kui elementide väärtusi ei arvutata, siis pole oluline, kuidas me funktsiooni '+1' tähistame.

Olgu  $s(X) := X + 1$  (Siin  $s$  tähendab 'successor').

$$1) \boxed{x} \text{ on naturaalarv} \implies s(\boxed{x}) \text{ on naturaalarv}$$

$$2) 0 \text{ on naturaalarv}$$

$$\mathbb{N} := \{0, s(0), s(s(0)), \dots\}$$

# Peano aritmeetika

## Sümbolarvud

$0, s(0), s(s(0)), s(s(s(0))), s(s(s(s(0)))) \dots$

## Tüübikirjeldus

`nat (0) .`

`nat (s (X) ) :- nat (X) .`

## Liitmine

`add (X, 0, X) .`

$\%x + 0 = x$

`add (X, s (Y) , s (Z) ) :- add (X, Y, Z) .`

$\%x + s(y) = s(x+y)$

`?-add (s (0) , s (s (0) ) , X) .`

`X = s (s (s (0) ) )`

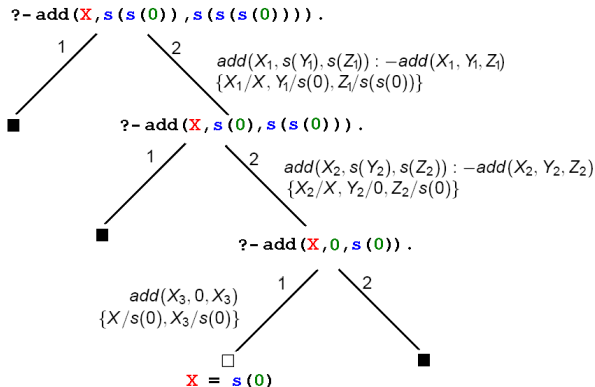
`?-add (X, s (s (0) ) , s (s (s (0) ) ) ) .`

`X = s (0)`

# Resolutsioon


1.  $\text{add}(X, 0, X)$  .
2.  $\text{add}(X, s(Y), s(Z)) : \neg \text{add}(X, Y, Z)$  .


Mis arvule X arvu 2 liitmine annab 3?



# Resolutsioon (järg)


## Frege küsimus:

?-add(X, s(s(0)), s(s(s())))).

X = s()

## Uus liitmine

add(X, 0, X) :- nat(X).  
add(X, s(Y), s(Z)) :- add(X, Y, Z).

?-add(X, s(s(0)), s(s(s())))).

false

?-add(X, Y, s(s(s(0)))).

X = s(s(s(0))) Y = 0 ;

X = s(s(0)) Y = s(0) ;

X = s(0) Y = s(s(0)) ;

X = 0 Y = s(s(s(0))) ;


false

# Peano aritmeetika

Tegelikult pole oluline, kuidas me nullelementi 0 ja järgmise arvu leidmise funktsiooni  $s$  tähistame.

Tähtis on, et see oleks programmis igal pool ühte moodi.

$$1) \text{ **X** }^{\text{on naturaalarv}} \implies s(\text{ **X** })^{\text{on naturaalarv}}$$

$$2) \text{  }^{\text{on naturaalarv}}$$

$$\mathbb{N} := \{ \text{  }, s(\text{  }), s(s(\text{  ))), \dots \}$$

Aga siis on võimalik, et programm muutub täiesti arusaamatuks!

# Korrektus ja täielikkus

```
nat (0) .
```

```
nat (s (X)) :- nat (X) .
```

Kas programm *nat* tõepoolest genereerib naturaalarve?

Taotletud mudel:  $\mathbf{N} = \{0, s(0), s^2(0), s^3(0), \dots\}$

Programm *nat* on **korrektne**: iga tuletatav term on naturaalarv.

$\{X \mid ?-nat(X)\} \subseteq \mathbf{N}$

```
?-nat (X) .
```

```
X = 0 ;
```

```
X = s (0) ;
```

```
X = s (s (0)) ;
```

```
...
```

Programm *nat* on **täielik**: iga naturaalarv on tuletatav.

$\mathbf{N} \subseteq \{X \mid ?-nat(X)\}$

```
?-nat (sn (0)) .
```

```
true
```

# Resolutsiooni korrektsus (formaalselt)




## Teoreem

*Olgu  $P$  määrav programm ja  $G$  määrav eesmärk. Siis  $P \cup \{G\}$  iga arvutatud vastus on  $P \cup \{G\}$  korrektne vastus.*

Programm peab alati vastama korrektselt.

Kui Prolog saab tuletada positiivse vastuse oma andmetest, siis ta vastab "jah".

Kui vastus pole nendest andmetest tuletatav, siis on vastus "ei".

1)  on lind  $\Rightarrow$    lendab

2)  on lind  
part

---

?-  on lind  
pingviin

false!  
programmist  
pole see tuletatav

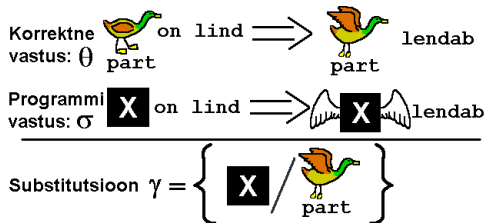


# Resolutsiooni täielikkus (formaalselt)

## Teoreem

*Olgu  $P$  määrav programm ja  $G$  määrav eesmärk.  $P \cup \{G\}$  iga korrektse vastuse  $\theta$  jaoks leidub  $P \cup \{G\}$  arvutatud vastus  $\sigma$  ja substitutsioon  $\gamma$ , nii et  $\theta = \sigma\gamma$ .*

Programm võib arvutada meile mingi vastuse  $\sigma$ , mis tegelikult ei ole päris see, mida me tahtsime. Siis on vaja rakendada  $\sigma$ -le veel substitutsiooni  $\gamma$ , mis asendab vabad muutujad korrektsete väärtustega. Saame kokkuvõtteks korrektse vastuse  $\theta$ .



## 3. loeng

### Tehted

- ▶ Loogikatehted
- ▶ Süsteemsed tehted
- ▶ Süntaksianalüüs
- ▶ Tehte muutmine ja lisamine
- ▶ Tüübikirjeldused
- ▶ Sümbolteisendused
- ▶ Tunnikontroll

# Loogikatehted

- ▶ tõene *true*
- ▶ väär *fail*, *false*
- ▶ konjunktsioon *a, b*
- ▶ disjunktsioon *a; b*
- ▶ eitus  $\neg a$   
(varem *not a*)
- ▶ implikatsioon *b: - a*  
if-then-else *a -> b; c*  
if-then *a -> b; true*
- ▶ võrdus (mittevõrdus)  
unifitseerimine *a = b* ( $a \setminus = b$ )  
objektide võrdus *a == b* ( $a \setminus == b$ )

# Süsteemsed tehted

## Tehete tüübid:

prefikstehe: *fx*, *fy*

lendab



infikstehe: *xfx*, *xfy*, *yfx*



ja



postfikstehe: *xf*, *yf*



lendab

# Süsteemsete tehete prioriteedid

Tehete prioriteedid on vahemikus  $1-1200$ .  
Mida väiksem arv, seda suurem prioriteet.

$:-\text{current\_op}(P, T, Op)$

$?\text{-current\_op}(P, T, +)$  .

$P = 200 \quad T = \text{fy} ;$

$P = 500 \quad T = \text{yfx} ;$

$?\text{-current\_op}(P, T, *)$  .

$P = 400 \quad T = \text{yfx}$

Näide:  $1 + 2 * 3 + 4 * 5 + (6 + 7)$

# Süntaksianalüüs

```
aunt(Person,Aunt):-  
    father(Person,A),  
    sister(A,Aunt).
```

```
?-aunt(alice,X).  
X = linda
```

```
aunt(Person,Aunt):-  
    father(Person,A) ; mother(Person,A),  
    sister(A,Aunt).
```

```
?-aunt(alice,X).  
X = _G245
```

# Süntaksianalüüs (järg)

```
?-listing(aunt/2).
```

```
aunt(A, B) :-  
    (    father(A, C)  
      ;    mother(A, C),  
        sister(C, B)  
    ).
```

```
?-display(aunt(A, B) :- (    father(A, C)  
                          ;    mother(A, C),  
                          sister(C, B) ) ).
```

```
:- (aunt(A, B), ; (father(A, C), , (mother(A, C), sister(C, B))))
```

```
?-current_op(P, T, ;).
```

```
P = 1100    T = xfy
```

```
?-current_op(P, T, ,).
```

```
P = 1000    T = xfy
```

# Tehte muutmine

$:-op(P, T, Op)$

```
?-X is 2+3*4.
```

```
X = 14
```

```
?-op(600,yfx,*).
```

```
?-X is 2+3*4.
```

```
X = 20
```

```
?-display(2+3*4).
```

```
* (+ (2, 3), 4)
```

```
?-op(400,yfx,*).
```

```
?-display(2+3*4).
```

```
+ (2, * (3, 4))
```



# Tehte lisamine

```
kõik linnud lendavad.
```

```
part on lind.
```

```
?-Kes lendab.
```

```
Kes = part
```



```
:-op(500,xf,[lendavad,lind,lendab]).
```

```
:-op(400,fx,kõik).
```

```
:-op(450,xf,on).
```

```
X lendab :- X on lind.
```

```
?-display(kõik linnud lendavad).
```

```
lendavad(kõik(linnud))
```

```
?-display(part on lind).
```

```
lind(on(part))
```

```
?-display(Kes lendab).
```

```
lendab(Kes)
```

# Tüübikirjeldused

- ▶ `atom(X)` –  $X$  on aatom
- ▶ `atomic(X)` –  $X$  on aatom või arv
- ▶ `integer(X)` –  $X$  on täisarv
- ▶ `number(X)` –  $X$  on arv
- ▶ `float(X)` –  $X$  on ujukoma-arv
- ▶ `var(X)` –  $X$  on väärtustamata muutuja
- ▶ `nonvar(X)` –  $X$  ei ole väärtustamata muutuja
- ▶ `compound(X)` –  $X$  on liitterm
- ▶ `ground(X)` –  $X$  on kinnine term

## Tüübikirjeldused: näide

$$26 * X^2 + 15 * X * Y + a * X + 1 + b$$

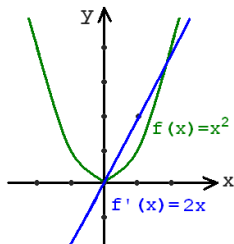
(Siin  $X^2 := X * X$ )

```
% polynomial(X) on tõene, kui X on polünoom
polynomial(a) .
polynomial(b) .
polynomial(c) .
polynomial(X):-integer(X) .
polynomial(X+Y):-polynomial(X), polynomial(Y) .
polynomial(X-Y):-polynomial(X), polynomial(Y) .
polynomial(X*Y):-polynomial(X), polynomial(Y) .
```

```
?- polynomial(1+2) .
?- polynomial(1+a) .
?- polynomial(1+x) .
?- polynomial(1+X) .
?- polynomial(2*(a+b)) .
```

# Sümbolteisendused: näide

## Diferentseerimine



Mõned reeglid:

$$x' = 1$$

$$c' = 0 \quad (c \text{ on konstant})$$

$$(f + g)'(x) = f'(x) + g'(x)$$

$$(f * g)'(x) = f'(x) * g(x) + f(x) * g'(x)$$

$$(\sin x)' = \cos x$$

# Sümbolteisendused: programm

```
% diff(P,X,DP) on tõene
% kui DP on avaldise P tuletis X järgi
diff(X,X,1):-atom(X).
diff(N,_X,0):-number(N).
diff(F+G,X,DF+DG):-
    diff(F,X,DF),
    diff(G,X,DG).
diff(F*G,X,DF*G+F*DG):-
    diff(F,X,DF),
    diff(G,X,DG).
diff(X^2,X,2*X).
diff(sin(X),X,cos(X)).
```

?-diff(2,x,D).

?-diff(2 + x,x,D).

?-diff(sin(x),x,D).

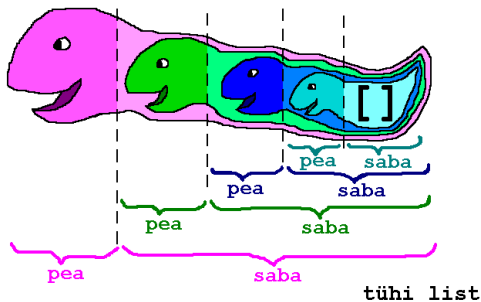
## 4. loeng

### Listid

- ▶ Mis on list?
- ▶ Tehted listidega
- ▶ Järjestamine
- ▶ Diferentslist
- ▶ Tunnikontroll

# Mis on list?

[ , , ,  ] neljaelemendiline list



= [  | [  | [  | [  |  ] ] ] ] ]

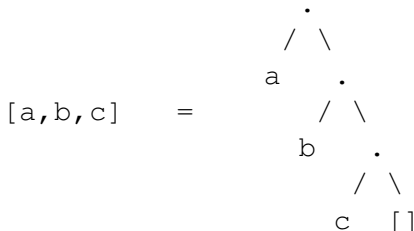
List on rekursiivne struktuur.

# Listi esitused

- |                                 |  |
|---------------------------------|--|
| <code>[a,b,c]</code>            | – standardne väline                    |
| <code>.(a,.(b,.(c,[ ])))</code> | – sisekuju (muudetud SWI-Prolog Ver.7) |
| <code>[a   [b,c]]</code>        | – Lispi laadne (cons-märgiga)          |

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

Sellise esituse järgi võib listi vaadelda puuna.





## Milliseid liste saab teha?

[a,b,c]  
[1,2,3]  
['Adam','Eeva']  
[[a,b],f(2,1),[[ ]]]


Matriks

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = [[1,0,0],[0,1,0],[0,0,1]]$$

## Listi konstrueerimine

Olgu programmis loetletud järgmised faktid:

```
lind().
```

```
lind().
```

```
lind().
```

Leiame üles kõik linnud ja paneme need ühe listi.

```
?- findall(X, lind(X), List).
```

```
List = [, , ].
```

Sarnased predikaadid: [bagof/3](#), [setof/3](#), [aggregate\\_all/3](#),  
[aggregate/3](#).

```
?- aggregate_all(count, lind(X), Count).
```

```
Count = 3.
```

```
?- aggregate_all(sum(1), lind(X), Count).
```

```
Count = 3.
```


```
?- aggregate_all(sum(X), lind(X), Y).
```


```
ERROR: is/2: Arithmetic: '/0' is not a function
```

## Erinevused: findall vs bagof, setof

ujub().

lendab().

ujub().

lendab().

ujub().






**setof** ja **bagof** tagastavad vastuse puudumisel false, **findall** moodustab tühja listi.

```
?-findall(X, (lendab(X), \+ujub(X)), List).  
List = [].
```

```
?-bagof(X, (lendab(X), \+ujub(X)), List).  
false.
```






```
?-setof(X, (lendab(X), \+ujub(X)), List).  
false.
```

## Erinevused: findall, bagof vs setof






ujub(  ) .	lendab(  ) .
ujub(  ) .	lendab(  ) .
ujub(  ) .	

**findall** ja **bagof** panevad listi sisse ka korduvaid väärtusi, **setof** vaid unikaalseid.

```
?-findall(X, (lendab(X) ; ujub(X)), List) .
```

```
List = [ , , , ,  ] .
```









```
?-bagof(X, (lendab(X) ; ujub(X)), List) .
```




```
List = [ , , , ,  ] .
```

```
?-setof(X, (lendab(X) ; ujub(X)), List) .
```

```
List = [ , ,  ] .
```

# Standardised tähistused

$X$	üksik element	
$X_S$	ühemõõtmeline list	[  ,  ,  , ... ]
$X_{SS}$	kahemõõtmeline list	[ [  ,  , ... ] , [  ,  , ... ] , ... ]

Siin iga  ,  ,  võib tegelikult olla ka omakorda list, kuid tavaliselt, kui mingis faktis või reeglis seda tähistatakse  $X$ , see tähendab, et selle sisemine struktuur meid ei huvita. Kui on kasutatud tähistust  $X_S$ , siis on tegemist kindlasti listiga, kuid pole oluline, kas selle listi elemendid on ise ka listid.

Mitme muutuja korral tähistatakse analoogiliselt:

$Y_S, Y_{SS}, Z_S, Z_{SS}, \dots$

# Listide liigid

## ► Täielik list:

```
?-[a, b, c, X] = [a, b, c, d].  
X = d.
```

## ► Osaline list:

```
?-[a, b, c|Xs] = [a, b, c, d, e].  
Xs = [d, e].
```

## ► Tsükliline list

```
?- Xs = [a, b, c |Xs].  
Xs = [a, b, c|Xs].
```

Predikaadid [cyclic\\_term/1](#) ja [acyclic\\_term/1](#) lubavad kontrollida, kas tegemist on tsüklilise termiga.

```
?- Xs = [a, b, c |Xs], cyclic_term(Xs).  
Xs = [a, b, c|Xs].
```

```
?- Xs = [a, b, c |Xs], acyclic_term(Xs).  
false.
```

# Kuidas kontrollida, kas tegemist on listiga?

Süsteemne predikaat `is_list(Xs)` kontrollib, kas `Xs` on list.

```
is_list([]).  
is_list(_|Xs):-  
    is_list(Xs).
```

Töötab vaid lõpliku listiga. Tsüklilise listi korral saame `false`.

► **Täielik** list:

```
?-is_list([a, b, c, X]).  
true.
```

► **Osaline** list:

```
?-is_list([a, b, c|Xs]).  
true. %false!
```

► **Tsükliline** list

```
?-Xs = [a, b ,c |Xs], is_list([a, b, c|Xs]).  
false.
```

# Tüübikirjeldus

- ▶ Võime defineerida listi elementide jaoks tüübikontrolli:

% `integer_list(Xs)` on tõene, kui `Xs` on arvude list

```
integer_list([]).  
integer_list([X|Xs]):-  
    integer(X),  
    integer_list(Xs).
```

- ▶ Võime kasutada enda poolt defineeritud predikaate.

```
lindude_list([]).  
lindude_list([X|Xs]):-  
    lind(X),  
    lindude_list(Xs).
```

- ▶ Võime täpsutada täisarvude listi.

```
kolmekohaliste_list([]).  
kolmekohaliste_list([X|Xs]):-  
    integer(X),  
    X >= 100, X <= 999,  
    kolmekohaliste_list(Xs).
```



# Tehted listidega (eelharjutused)

Lisamine, muutmine, kustutamine

?-Xs=[a,b,c], Ys=[d|Xs].

Ys=[d,a,b,c]

?-Xs=[a,b,X], X=c.

Xs = [a,b,c]

?-Xs=[a,b,c], Xs=[A,B,C], Ys=[A,B,C,d].

Ys=[a,b,c,d]

?-Ys=[a,b,c|Zs], Zs=[d|Ws], Ws=[e].

Ys=[a,b,c,d,e]

?-Xs=[a,b,c], Xs=[A,B,C], Ys=[A,d,C].

# Tehted listidega

otsimine	– member/2
lisamine	– (add), [a   Xs]
eemaldamine	– select/3, delete/3
ühendamine	– append/3
pööramine	– reverse/2
tasandamine	– flatten/2
pikkus	– length/2
indekseerimine	– nth1/3, nth0/3
summeerimine	– sumlist/2
järjestamine	– msort/2, sort/2, predsort, keysort

# Listist otsimine

% **member(X,Xs)** on tõene, kui **X** on listi **Xs** element

```
member (X, [X|_] ) .
```

```
member (X, [_|Ys] ) :-
```

```
    member (X, Ys) .
```

Kasutused:

1. Element kuulub listi

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

2. Võta listist element

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

3. Genereeri list

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

# Listist eemaldamine

% **select**(X,Xs,Ys) on tõene, kui eemaldades elemendi X

% listist Xs saame listi Ys

```
select (X, [X|Xs], Xs) .
```

```
select (X, [Y|Ys], [Y|Zs]) :-
```

```
    select (X, Ys, Zs) .
```

```
?-select (b, [a,b,c], Xs) .
```

```
Xs=[a,c]
```

```
?-select (b, Xs, [a,c]) .
```

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

**member** select'i kaudu:

```
member (X, Xs) :-select (X, Xs, _) .
```

## Listist eemaldamine: delete

% **delete(Xs,X,Ys)** on tõene, kui eemaldades listist **Xs**

% elemendi **X** kõik esinemised saame listi **Ys**

```
delete([],_,[]).
```

```
delete([X|Xs],X,Ys):-delete(Xs,X,Ys).
```

```
delete([X|Xs],Y,[X|Zs]):-
```

```
    delete(Xs,Y,Zs).
```

**select** ja **delete** erinevus:

- ▶ **select** annab mitu vastust, eemaldab vaid ühe elemendi.

```
?-select(a,[a, b, a, c],Xs).
```

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

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

```
false.
```

- ▶ **delete** eemaldab antud elemendi kõik esinemised.

```
?-delete([a, b, a, c],a,Xs).
```

```
Xs = [b, c].
```

# Listide ühendamine

% `append(Xs,Ys,Zs)` on tõene, kui listide `Xs` ja `Ys` ühend on `Zs`

```
append([ ], Xs, Xs) .
```

```
append([X|Xs], Ys, [X|Zs]) :-
```

```
    append(Xs, Ys, Zs) .
```

Kasutused:

1. `?-append([a,b,c],[d],Xs) .`

`Xs=[a,b,c,d]`

2. `?-append(Xs,Ys,[a,b,c,d]) .`

Richard O'Keefe:

`?-append([ ], X, X) .`  $\neq$  `?-append(X, [ ], X) .`

`member` `append`'i kaudu:

```
member(X, Xs) :- ...
```

# Listi pikkus

% `length(Xs,N)` on tõene, kui listi `Xs` pikkus on `N`

```
length([],0).
```

```
length([X|Xs],N):-
```

```
    length(Xs,N1),
```

```
    N is N1+1.
```

Kasutused:

1. `?-length([a,b,c],N).`

`N=3`

2. `?-length(X,3).`

`X=[_,_,_]`

# Summeerimine

## Akumulaator:

```
sumlist(List, Sum) :- sumlist(List, 0, Sum) .
```

```
sumlist([], Sum, Sum) .
```

```
sumlist([X|Xs], Acc, Sum) :-
```

```
    Acc2 is Acc+X,
```

```
    sumlist(Xs, Acc2, Sum) .
```

```
?-sumlist([1,2,3], X) .
```

```
?-sumlist([1,2,3], 0, X) .
```

```
?-sumlist([2,3], 1, X) .
```

```
?-sumlist([3], 3, X) .
```

```
?-sumlist([], 6, X) .
```

```
X=6
```



## Summeerimine (järg)

```
sumlist([], 0).  
sumlist([X|Xs], Sum) :-  
    sumlist(Xs, Sum2),  
    Sum is Sum2+X.
```

```
?-sumlist([1,2,3], X).
```

```
?-sumlist([2,3], X).
```

```
?-sumlist([3], X).
```

```
?-sumlist([], 0).
```

```
?-sumlist([3], 3).
```

```
?-sumlist([2,3], 5).
```

```
?-sumlist([1,2,3], 6).
```

```
X=6
```

## Muud tehted listi elementidega

- Kõigepealt defineerime predikaadi `add/3` tavalise aritmeetika jaoks.

```
add(X, Y, Z) :-  
    Z is X + Y.
```

- `maplist(P,Xs,Ys,Zs)` on tõene, kui `P(X,Y,Z)` on tõene iga  $X \in Xs$ ,  $Y \in Ys$ ,  $Z \in Zs$  korral, kus  $X,Y,Z$  asuvad samal positsioonil.

```
?-maplist(add, [1, 2, 3], [3, 4, 5], Zs).  
Zs = [4, 6, 8].
```

- Analoogiliselt defineeritakse `maplist/3` binaarse predikaadi `inverse/2` jaoks.

```
inverse(X, Y) :-  
    Y is -X.
```

```
?-maplist(inverse, [1, 2, -3], Ys).  
Zs = [-1, 2, 3].
```

# Järjestamine

- ▶ **sort**

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

- ▶ **msort**

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

- ▶ **predsort**

```
order(<,tartu,tallinn).  
order(<,elva,tallinn).  
order(<,elva,tartu).
```

```
?-predsort(order,[elva,tartu,tallinn],X).
```

- ▶ **keysort**

```
?-keysort([2-tartu,3-tallinn,1-elva],X).  
X = [1-elva, 2-tartu, 3-tallinn].
```

# Slowsort

% **slowsort(Xs,Ys)** on tõene, kui listi **Xs** järjestatud kuju on **Ys**

```
slowsort(List, SortedList) :-  
    integer_list(List),  
    permutation(List, SortedList),  
    ordered(SortedList).
```

```
?-slowsort([1,3,2,4,2],S).
```

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

# Permutatsioonid

% `permutation(Xs,Ys)` on tõene, kui `Ys` on listi `Xs` permutatsioon

```
permutation([], []).
```

```
permutation(List, [X|Xs]) :-
```

```
    select(X, List, List2),
```

```
    permutation(List2, Xs).
```

```
?-permutation([a,b,c], Xs).
```

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

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

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

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

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

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

```
false.
```

# Diferentslist

$$x - y$$

$$Xs - []$$

$$[1, 2, 3] = [1, 2, 3 | Xs] - Xs$$

`concat (Xs-Ys, Ys-Zs, Xs-Zs) .`

`?-concat ([1, 2, 3 | Xs] - Xs, [4, 5 | Ys] - Ys, List) .`

`List=[1, 2, 3, 4, 5 | Ys] - Ys`

1 samm

`?-append ([1, 2, 3], [4, 5], List) .`

`List=[1, 2, 3, 4, 5]`

4 sammu

## Diferentslistid: arvuväljendite parsimine

```
?-sajalistefraas([sada,viiskümmend,kuus],[ ]).
```

```
sajalistefraas(Xs,Ys):-
```

```
    sajalised(Xs,Zs), künnelistefraas(Zs,Ys).
```

```
künnelistefraas(Xs,Ys):-
```

```
    künnelised(Xs,Zs), ühelistefraas(Zs,Ys).
```

```
ühelistefraas(Xs,Ys):-ühelised(Xs,Ys).
```

```
sajalised([sada|Xs],Xs).
```

```
künnelised([viiskümmend|Xs],Xs).
```

```
künnelised(Xs,Xs).
```

```
ühelised([kuus|Xs],Xs).
```

```
ühelised(Xs,Xs).
```

## 5. loeng

### Keerdülesanded

- ▶ Mis nädalapäev täna on?
- ▶ Ahv ja Banaan
- ▶ Kolm sõpra
- ▶ Tunnikontroll



# Vahendid

- ▶ algorithm = logic + control
- ▶ genereeri ja testi
- ▶ andmestruktuurid
- ▶ eitus

# Eitus


Eitus on tõesuse puudumine (negation as failure)

lind(  ).

lendab(  ).

?-\+lendab(X), lind(X).  
false.

?-lind(X), \+lendab(X).

X= 

**Järeldus:** kasutage eitust võimalikult hilja.

# 1. Mis nädalapäev täna on?

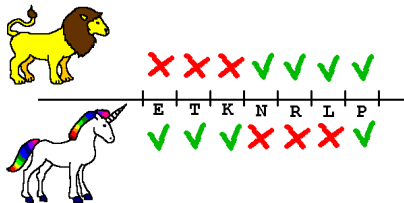
Alice kohtas metsas Lõvi ja Ükssarve.

On teada, et Lõvi valetab esmaspäeval, teisipäeval ja kolmapäeval ning räägib tõtt ülejäänud nädalapäevadel.

Ükssarv valetab neljapäeval, reedel ja laupäeval ning räägib tõtt ülejäänud päevadel.

Alice päris loomadelt, kas nad eile rääkisid tõtt või valetasid. Mõlemad loomad väitsid, et olid eile valetanud.

Mis nädalapäeval Alice neid küsitles?



# Faktid



```
lõvivaletab(esmapäev). ükssarvvaletab(neljapäev).  
lõvivaletab(teisipäev). ükssarvvaletab(reede).  
lõvivaletab(kolmapäev). ükssarvvaletab(laupäev).
```

```
lõvitõtt(X):- \+lõvivaletab(X).  
ükssarvtõtt(X):- \+ükssarvvaletab(X).
```

```
eile(esmapäev,pühapäev).
```

```
...
```

```
eile(pühapäev,laupäev).
```

# Reeglid

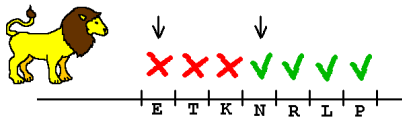


```
lõvi(X) :- eile(X,Y), lõvitõtt(X), lõvivaletab(Y).  
lõvi(X) :- eile(X,Y), lõvivaletab(X), lõvitõtt(Y).
```



```
ükssarv(X) :- eile(X,Y), ükssarvtõtt(X), ükssarvvaletab(Y).  
ükssarv(X) :- eile(X,Y), ükssarvvaletab(X), ükssarvtõtt(Y).
```

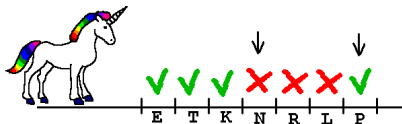
# Lahendus



?-lõvi(X).

X = neljapäev ;

X = esmaspäev



?-ükssarv(X).

X = pühapäev ;

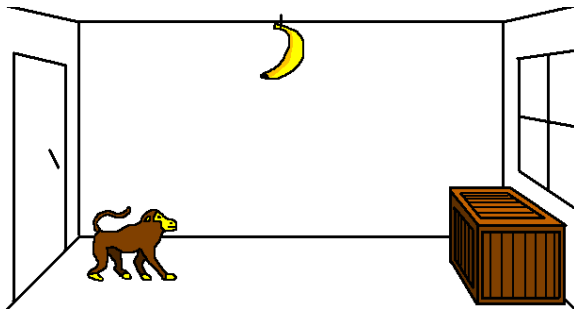
X = neljapäev

lahenda(X):- lõvi(X), ükssarv(X).

## 2. Ahv ja Banaan

Ahv on toas, mille keskel on lakke riputatud Banaan. Ahv asub vasakul ukse juures ja paremal akna juures on suur Kast. Ahv ei ulatu põrandalt Banaanini; küll on võimalik selleni ulatuda Kasti pealt.

Kuidas Ahv saab Banaani kätte?



# Võimalikud tegevused

► võta



► lükka



► roni



► liigu





# Seisukirjeldused

*seis(AhvHor, AhvVer, Kast, OmabBanaani)*

AhvHor: *uks, keskel, aken*

AhvVer: *põrand, kastil*

Kast:

OmabBanaani: *omab, eioma*

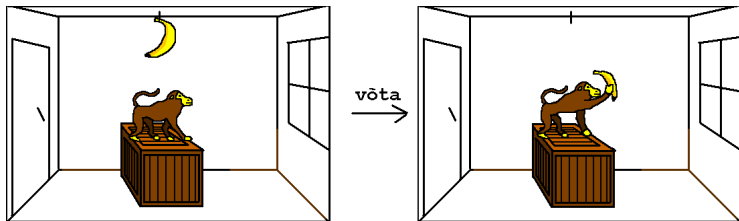
**algseis:** *seis(uks,põrand,aken,eioma)*

**lõppseis:** *seis(\_,\_,\_,omab)*

# Käigud:

## võta (1)

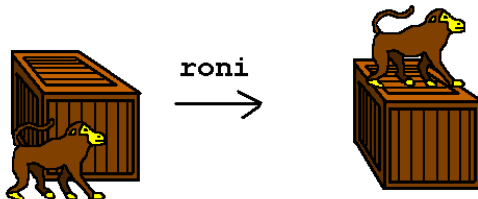
```
käik (seis (keskel, kastil, keskel, eioma),  
      võta,  
      seis (keskel, kastil, keskel, omab)) .
```



# Käigud:

roni (6)

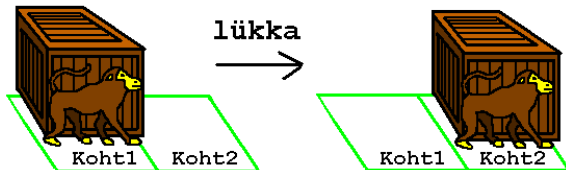
```
käik (seis (Koht, põrand, Koht, Omamine),  
      roni,  
      seis (Koht, kastil, Koht, Omamine)).
```



# Käigud:

## lükka

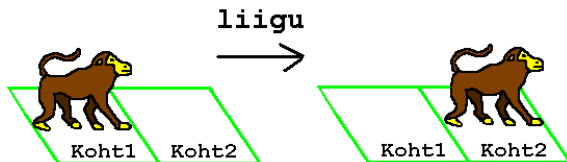
```
käik (seis (Koht1, põrand, Koht1, Omamine) ,  
      lükka (Koht1, Koht2) ,  
      seis (Koht2, põrand, Koht2, Omamine) ) .
```



# Käigud:

## liigu

```
käik (seis (Koht1, põrand, Kast, Omamine),  
      liigu (Koht1, Koht2),  
      seis (Koht2, põrand, Kast, Omamine)) .
```



# Liikumine

```
saab_kätte(seis(_,_,_,omab)).
```

```
saab_kätte(Seis1):-  
    käik(Seis1,Käik,Seis2),  
    write(Käik), nl,  
    saab_kätte(Seis2).
```

```
?-saab_kätte(seis(uks,põrand,aken,eioma)).
```

```
liigu(uks, _G461)  
roni  
lükka(aken, _G469)  
roni  
võta
```

### 3. Kolm sõpra

Kolm sõpra said programmeerimisvõistlusel esimese, teise ja kolmanda koha. Igaühel neist on erinev eesnimi, spordiala ja kodakondsus.

Teadada on faktid:

1. Michael mängib korvpalli ja ta oli parem kui ameeriklane.
2. Simon on israellane ja ta oli parem kui tennisemängija.
3. Kriketimängija tuli esimeseks.

Küsimused:

- ▶ Kes on austraallane?
- ▶ Millist sporti teeb Richard?

# Struktuur ja predikaadid

```
struktuur ([sõber (_, _, _),  
            sõber (_, _, _),  
            sõber (_, _, _)]).
```

## Predikaadid

```
nimi (sõber (A, _, _), A).  
kodakondsus (sõber (_, B, _), B).  
sport (sõber (_, _, C), C).
```

```
oli_parem (A, B, [A, B, _]).  
oli_parem (A, C, [A, _, C]).  
oli_parem (B, C, [_, B, C]).
```

```
first ([X|_], X).
```



# Programm

```
mõistatus:-  
    struktuur(Sõbrad),  
    oli_parem(S11,S12,Sõbrad),  
        nimi(S11,michael),  
        sport(S11,korvpall),  
        kodakondsus(S12,ameerika),  
    oli_parem(S21,S22,Sõbrad),  
        nimi(S21,simon),  
        kodakondsus(S21,iisraeli),  
        sport(S22,tennis),  
    first(Sõbrad,S3),  
        sport(S3,kriket),
```

# Lahendus

```
member(S4,Sõbrad),  
    nimi(S4,Nimi),  
    kodakondsus(S4,austraalia),  
member(S5,Sõbrad),  
    nimi(S5,richard),  
    sport(S5,Sport),  
write('Austraallane on '), write(Nimi), nl,  
write('Richard mängib '), write(Sport), nl.
```

?-mõistatus.

Austraallane on michael  
Richard mängib tennis