

## *Logikai programozás*

8.

### ISMÉTLÉS: ADATBÁZISKEZELÉS

Adatok: a tényállítások

A külső adatok a consult hatására bekerülnek a memóriába

Lekérdezés:

Ahogy eddig – pl.:

szereti(jani, sör).

szereti(Ki, Mit).

mernek(Hol, bor).

## ISMÉTLÉS: ADATBÁZISKEZELÉS

Beszúrás: `assert/1`

pl.: `assert(szereti(jani, bor)).`

Törlés: `retract/1, retractall/1`

pl.: `retract(szereti(jani, bor)).`

`retract(szereti(jani, _)).`

Ha nincs törendő:

eredmény: `false`

`retract(szereti(_, _)).`

`retractall(szereti(jani, _)).`

Ha nincs törendő:

eredmény: `true`

## ISMÉTLÉS: ADATBÁZISKEZELÉS

A beszúrás, törlés hatására hibaüzenetet kaphatunk:

```
assert/1: No permission to modify static_procedure
`szereti/2'
```

A predikátumok alapértelmezetten statikusként fordulnak.

Dinamikussá tétel:

Figyelem!

`:- dynamic szereti/2.`

Ez deklaráció, nem predikátum,  
vagyis nincs zárójel!

## ISMÉTLÉS: ADATBÁZISKEZELÉS

Összes adat kilistázása: listing

pl.: listing(szereti/2).

Az adatok addig „élnek”, amíg ki nem lépünk a futtató-környezetből.

Fájlba mentés mint eddig, de a listing-gel egyszerűsíthető:

pl.: fajlba:- tell('fajlnev'), listing(szereti/2), told.

Hatására a dynamic deklaráció is a fájlba kerül – saját adatfájlnál is célszerű ide írni.

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

assert/1: nem csak adatokat, hanem szabályokat is be tudunk szűrni. Szintaktikája: assert(a:-b.)

Pl.: szereti(jani, bor).  
szereti(eva, tej).  
arulnak(sarok, bor).  
arulnak(tejbolt, tej).

egy :- assert(ember(X) :- szereti(X,\_)).

ketto :-assert(ital(X):-szereti(\_,X)).

harom :- assert(vasarol(X,Hely) :- (szereti(X,Y),  
arulnak(Hely,Y))).

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

„Szakértői rendszer” :

A felhasználónak feltett választól függően a kocsmázáshoz vagy kell pénz, vagy sem, vagyis a választól függően a két szabály valamelyike használható:

jar(Diak, Kocsma) :- szereti(Diak,Ital),  
mernek(Kocsma,Ital).

jar(Diak, Kocsma) :- szereti(Diak,Ital),  
mernek(Kocsma,Ital),  
gazdag(Diak).

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

Vagyis a választól függően:

retractall(jar(Diak,Kocsma):- (szereti(Diak,Ital),  
mernek(Kocsma,Ital),  
gazdag(Diak))),

assert(jar(Diak,Kocsma):- (szereti(Diak,Ital),  
mernek(Kocsma,Ital))),

vagy fordítva az assert és a retractall

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

Fibonacci sorozat

fib(0, 1).

fib(1, 1).

fib(N, F) :- N >= 2, N1 is N - 1, N2 is N - 2,  
fib(N1, F1), fib(N2, F2), F is F1 + F2.

?- fib(5,F).

Nyomkövetés: 86 lépés.  
pontosvessző után újabb 40.  
újrakéretés 86 lépés.

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

Fibonacci sorozat - módosítás

:- dynamic fib/2.

fib(0, 1). fib(1, 1).

fib(N, F) :- N >= 2, N1 is N - 1, N2 is N - 2,  
fib(N1, F1), fib(N2, F2), F is F1 + F2,  
asserta(fib(N,F)).

?- fib(5,F).

Nyomkövetés: 58 lépés.  
pontosvessző után újabb 21.  
újrakéretés 2 lépés.

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

Fibonacci sorozat – újabb módosítás

`:- dynamic fib/2.`

`fib(0, 1). fib(1, 1).`

`fib(N, F) :- N >= 2, N1 is N - 1, N2 is N - 2,  
fib(N1, F1), fib(N2, F2), F is F1 + F2,  
asserta(fib(N,F) :- !).`

?- fib(5,F).

Nyomkövetés: 58 lépés.

pontosvessző után újabb 16.

újraküldés 2 lépés.

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

Fibonacci sorozat – még hatékonyabb változat

`fib(0,1).`

`fib(N,F) :- N > 1, fib(N,F,_).`

`fib(1,1,1).`

`fib(N,F,FElozo) :- N > 1, N1 is N-1,  
fib(N1, FElozo, FEE),  
F is FElozo + FEE.`

?- fib(5,F).

Nyomkövetés: 38 lépés.

pontosvessző után újabb 9.

újraküldés 38 lépés.

## ASSERT/RETRACT TOVÁBBI ALKALMAZÁSA

Fibonacci sorozat – módosított változat

:-dynamic fib/3.

fib(0,1).

fib(N,F) :- N > 1, fib(N,F,\_).

fib(1,1,1).

fib(N,F,FElozo) :- N > 1, N1 is N-1,  
fib(N1, FElozo, FEE),  
F is FElozo + FEE,  
asserta(fib(N,F,FElozo):-!).

?- fib(5,F).

Nyomkövetés: 46 lépés.

pontosvessző után újabb 9.

újrafuttatás 6 lépés és nincs alternatíva.

## ADATBÁZISKEZELÉS – PÉLDÁK

1. Egy szokatlan alkalmazás: „procedurális” összegzés

start1:- **retractall(ossz(\_))**, **assert(ossz(0))**, számol.

számol :- write('van még szám? i/n '), read(V), V = i,  
write(' a szám értéke: '), read(Sz),  
**ossz(Eddig)**, **retract(ossz(Eddig))**,  
Uj is Eddig + Sz, **assert(ossz(Uj))**,  
számol.

számol :- **ossz(S)**, writef('A számok összege: %w\n', [S]).

## ADATBÁZISKEZELETÉS – PÉLDÁK

2. Adatbázisban lévő adatok (pl.: adat(kati,3).) összegzése:

szamol(Eddig) :- adat(Nev,Db), Uj is Eddig + Db,

**retract(adat(Nev,Db)),**

**assert(temp(Nev,Db)),** szamol(Uj).

szamol(Veg) :- writef('Az adatok összege: %w', [Veg]),

visszair.

visszair :- temp(Nev,Db), retract(temp(Nev,Db)),

assert(adat(Nev,Db)),visszair.

visszair.

start :- szamol(0), ...

## ADATBÁZISKEZELETÉS – PÉLDÁK

3. Ugyanaz másképp (ügyesebben):

start :- retractall(osszeg(\_)), assert(osszeg(0)),

forall(adat(\_,Db),

retract(osszeg(Eddig)),

Uj is Eddig + Db,

assert(osszeg(Uj))

)

),

osszeg(Veg),

writef('Az adatok összege: %w', [Veg]).



## ADATBÁZISKEZELÉS – PÉLDÁK

4. Max-keresés adatbázison (pl. adat(kati,3).) (a):

start :- adat(Nev,Szam), max(Nev,Szam).

max(\_,Eddigimax) :- adat(Nev,Szam),  
Szam>Eddigimax,                   Hol van az  
max(Nev,Szam).                   „else” ág?

max(Nev,Max) :- writef('Legjobb %w %w értékkel',  
[Nev,Max]).

Hátrányok:

1. csak kiírja, de nem tárolja a legjobbat
2. nem enged holtversenyt
3. rossz hatásfok

## ADATBÁZISKEZELÉS – PÉLDÁK

5. Max-keresés adatbázison (pl. adat(kati,3).) (b):

start :- adat(\_,Szam), max2(Szam,Vegmax),  
writef('A legjobbak %w értékkel\n', [Vegmax]),  
forall(adat(Nev,Vegmax), writeln(Nev)).

max2(Eddigimax,Vegmax) :- adat(\_,Szam),  
Szam>Eddigimax,  
max2(Szam,Vegmax).

max2(Vegmax,Vegmax).

Ez sem őrzi meg az adatokat, de kezeli a holtversenyt.

## ADATBÁZISKEZELÉS – PÉLDÁK

6. Max-keresés adatbázison (pl. adat(kati,3).) (c):

```
start :- adat(_,Szam), max2(Szam,Vegmax),  
        writef('A legjobb %w értékkel\n', [Vegmax]),  
        findall(Nev, adat(Nev,Vegmax), Nevsor), ki(Nevsor).
```

```
ki([Nev|_]) :- writeln(Nev), ki(_).  
ki([]).
```

Most a Nevsor lista őrzi is a legjobb névsorát.  
(Ez persze, nem biztos, hogy fontos.)

Esetleg még: sort(Nevsor, Rendezett).

## ADATBÁZISKEZELÉS – PÉLDÁK

De persze így is lehet:

```
findall(Szam, adat(_,Szam), Szamok)  
majd kikeresni a Szamok lista maximumát, és megkeresni,  
melyik nevekhez tartozik ez az érték.
```

## ADATBÁZISKEZELÉS – PÉLDÁK

Hogyan rendez?

```
findall((Nev,Szam), adat(Nev,Szam), Adatok),  
sort(Adatok, Rendezve),...
```

```
findall((Szam,Nev), adat(Nev,Szam), Adatok),  
sort(Adatok, Rendezve),...
```

Megjegyzés:

Attól még, hogy rendezéshez esetleg (Szam,Nev) alakban adjuk meg az adatokat, kiíratni Nev, Szam sorrendben illik!

## ADATBÁZISKEZELÉS – PÉLDÁK

A nevek.dat fájlban nevek vannak felsorolva. Be kell olvasni őket, és létrehozni egy diak(Nev,0) szerkezetű adatokból álló adatbázist.

```
beolvas:- retractall(diak(_,_)),  
           see('nevek.dat'), be, seen.
```

```
be :- read(Nev), Nev \== end_of_file,  
      assert(diak(Nev,0)), be.
```

```
be.
```

## ADATBÁZISKEZELÉS – PÉLDÁK

Ugyanez másképp:

```
be :- repeat,  
    read(Nev),  
    (  
        Nev == end_of_file,!;  
        assert(diak(Nev,0)), fail  
    ).
```