

# Deklaratív programozás

---

Hanák Péter

hanak@inf.bme.hu

Irányítástechnika és Informatika Tanszék

Szeredi Péter

szeredi@cs.bme.hu

Számítástudományi és Információelméleti Tanszék

# KÖVETELMÉNYEK, TUDNIVALÓK

---

# Deklaratív programozás: tudnivalók

---

## ● Honlap, levelezési lista

- Honlap: `<http://dp.iit.bme.hu>`

- Levlista: `<http://www.iit.bme.hu/mailman/listinfo/dp-1>`.

A listára automatikusan felvesszük a tárgy hallgatóit az ETS-beli címükkel. Címet módosítani csak az ETS-en keresztül lehet.

A listatagoknak szóló levelet a `<dp-1@iit.bme.hu>` címre kell küldeni.

Csak a levlistában szereplő címről küldött levelek jutnak el moderátori jóváhagyás nélkül a listatagokhoz.

## ● Jegyzetek

- Szeredi P., Benkő T.: Deklaratív programozás. Bevezetés a logikai programozásba (1000 Ft)

- Hanák D. Péter: Deklaratív programozás. Bevezetés a funkcionális programozásba (850 Ft)

- A nyomtatott változat korlátozott számban megvásárolható a SZIT tanszék V2 épületbeli titkárságán a V2.104 szobában, Bazsó Lászlónénál, 10:30-12:00 (hétfő-péntek) és 13:30-15:30 (hétfő-csütörtök)

- A jegyzetek elektronikus változatban letölthetők a tárgy honlapjáról.

## Deklaratív programozás: tudnivalók (folyt.)

---

### Fordító- és értelmezőprogramok

- SICStus Prolog — 3.12 verzió (licenz az ETS-en keresztül kérhető)
- Moscow SML (2.0, szabad szoftver)
- Mindkettő telepítve van a `kempelen.inf.bme.hu`-n
- Mindkettő letölthető a honlapról (linux, Win95/98/NT)
- Webes gyakorló felület az ETS-ben (ld. honlap)
- Kézikönyvek HTML-, ill. PDF-változatban
- Más programok: swiProlog, gnuProlog, poly/ML, smlnj
- emacs-szövegszerkesztő SML-, ill. Prolog-módban (linux, Win95/98/NT)

# Deklaratív programozás: félévközi követelmények

---

## Nagy házi feladat (NHF)

- Programozás mindkét nyelven (Prolog, SML)
- Mindenkinek önállóan kell kódolnia (programoznia)!
- Hatékony (időlimit!), jól dokumentált („kommentezett”) programok
- A két programhoz közös, 5–10 oldalas fejlesztői dokumentáció (PDF, PS, HTML, TXT,  $\LaTeX$ ,  $\TeX$ ; de nem DOC vagy RTF)
- Kiadás a 6. héten, a honlapon, letölthető keretprogrammal
- Beadás a 12. héten; elektronikus úton (ld. honlap)
- A beadáskor és a pontozáskor külön-külön tesztsorozatot használunk (nehézségben hasonlókat, de nem azonosakat)
- A minden tesztesetet hibátlanul megoldó programok *létraversenyen* vesznek részt (hatékonyság, gyorsaság plusz pontokért)

## Deklaratív programozás: félévközi követelmények (folyt.)

---

### Nagy házi feladat (folyt.)

- Nem kötelező, de *nagyon* ajánlott!
- Beadható csak az egyik nyelvből is
- A beadási határidőig többször is beadható, csak az utolsót értékeljük
- Pontozása mindkét nyelvből:
  - helyes és időkorláton belüli futás esetén a 10 tesztet mindegyikére 0,5-0,5 pont, összesen max. 5 pont, feltéve, hogy legalább 4 tesztet sikeres
  - a dokumentációra, a kód olvashatóságára, kommentezettségére max. 2,5 pont
  - tehát nyelvenként összesen max. 7,5 pont szerezhető
- A NHF súlya az osztályzatban: 15% (a 100 pontból 15)

## Deklaratív programozás: félévközi követelmények (folyt.)

---

### Kis házi feladatok (KHF)

- 2-3 feladat Prologból is, SML-ből is
- Beadás elektronikus úton (ld. honlap)
- Nem kötelező, de *nagyon* ajánlott
- Minden feladat jó megoldásáért 1-1 jutalompont jár

### Gyakorló feladatok

- Nem kötelező, de a sikeres ZH-hoz, vizsgához *elengedhetetlen!*
- Gyakorlás az ETS rendszerben (lásd honlap)

### Tantermi gyakorlatok

- A keddi előadásokon általában gyakorló feladatokat oldunk meg a hallgatóság bevonásával, és lehetőséget adunk a közérdeklődésre számot tartó kérdések megbeszélésére.

### Laboratóriumi konzultációk

- Rendszeres – számítógép melletti – konzultációs lehetőség az órarendi órákon kívül

## Deklaratív programozás: félévközi követelmények (folyt.)

---

### Nagyzárthelyi, pótzárthelyi (NZH, PZH, PPZH)

- A zárthelyi kötelező, semmilyen jegyzet, segédlet nem használható!
- 40%-os szabály (nyelvenként a maximális részpontszám 40%-a kell az eredményességhez).  
Kivétel: a korábban aláírást szerzett hallgató zárthelyin szerzett pontszámát az alsó ponthatártól függetlenül beszámítjuk a félévvégi osztályzatba. A korábbi félévekben szerzett pontokat nem számítjuk be!
- Az NZH a 10. héten, a PZH az utolsó oktatási hetekben lesz
- A PPZH-ra indokolt esetben, ismétlővizsga-jelleggel a vizsgaidőszak első három hetében egyetlen alkalommal adunk lehetőséget
- Az NZH anyaga az első két blokk (nagyjából az 1.-8. hét) tananyaga
- A PZH, ill. a PPZH anyaga azonos az NZH anyagával
- A zárthelyi súlya az osztályzatban: 15% (a 100 pontból 15)
- Több zárthelyi megírása esetén a zárthelyikre kapott pontszámok közül a *legnagyobb*at vesszük figyelembe



# Deklaratív programozás: vizsga

---

## Vizsga

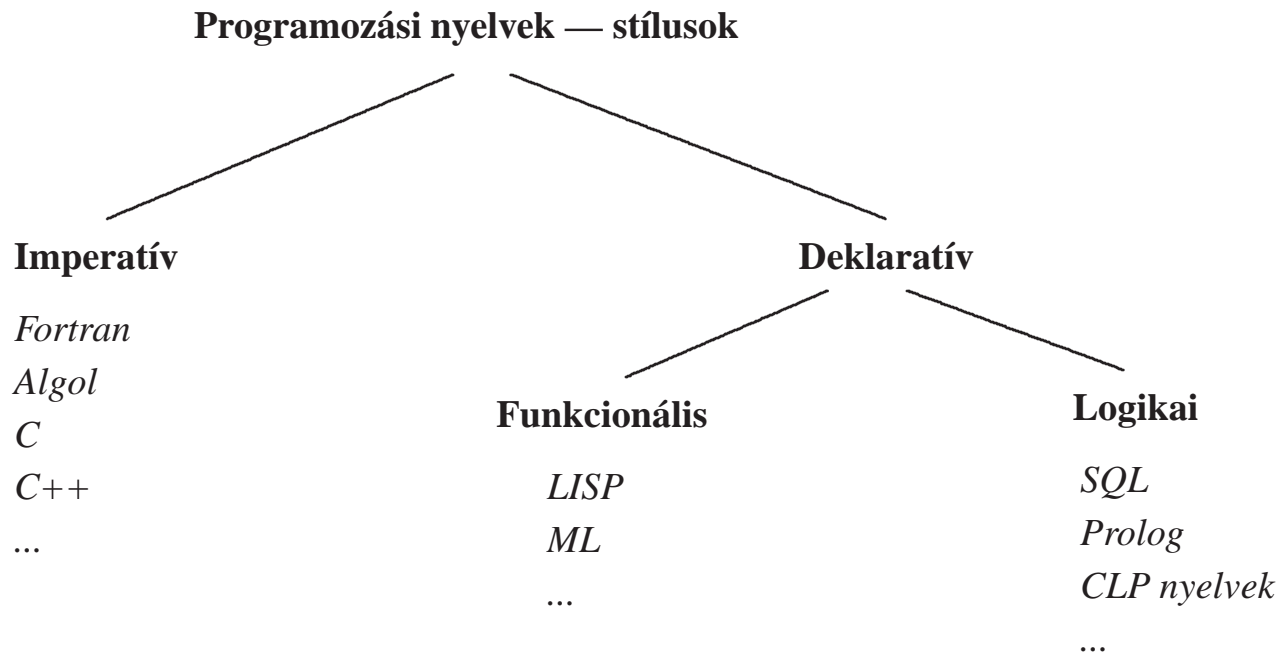
- Vizsgára az a hallgató bocsátható, aki aláírást szerzett a jelen félévben vagy a jelen félévet megelőző négy félévben
- A vizsga szóbeli, felkészülés írásban
- Prolog, SML: több kisebb feladat (programírás, -elemzés) kétszer 35 pontért
- A vizsgán szerezhető max. 70 ponthoz adjuk hozzá a **jelen** félévben félévközi munkával szerzett pontokat: ZH: max. 15 pont, NHF: max. 15 pont, továbbá a pluszpontokat (KHF, létraverseny)
- *Korábbi* félévben szerzett pontokat *nem* számítunk be!
- A vizsgán semmilyen jegyzet, segédlet nem használható, de lehet segítséget kérni
- Ellenőrizzük a nagy házi feladat és a zárthelyi „hitelességét”
- 40%-os szabály (nyelvenként a max. részpontszám 40%-a kell az eredményességhez)
- Korábbi vizsgakérdések a honlapon találhatóak

# DEKLARATÍV ÉS IMPERATÍV PROGRAMOZÁS

---

# Programozási nyelvek osztályozása

---



# Imperatív és deklaratív programozási nyelvek

---

## ● Imperatív program

- felszólító módú, utasításokból áll
- változó: változtatható értékű memóriahely
- C nyelvű példa:

```
int pow(int a, int n) { // pow(a,n) = a ^ n
    int p = 1;         // Legyen p értéke 1!
    while (n > 0) {   // Amíg n>0 ismételd ezt:
        n = n-1;     // Csökkentsd n-et 1-gyel!
        p = p*a;     // Szorozd p-t a-val!
    }                // Add vissza p végértékét
    return p;
}
```

## ● Deklaratív program

- kijelentő módú, egyenletekből, állításokból áll
- változó: egyetlen rögzített értékkel bír, amely a programírás idején még ismeretlen
- SML példa:

```
fun pow(a, n) =
    if n > 0           (* Ha n > 0 *)
    then a*pow(a,n-1) (* akkor a^n = a*a^(n-1) *)
    else 1             (* egyébként a^n = 1 *)
```

## Deklaratív programozás imperatív nyelven

---

- Lehet pl. C-ben is deklaratívan programozni,
  - ha nem használunk: értékadó utasítást, ciklust, ugrást, stb.,
  - amit használhatunk: (rekurzív) függvények, if-then-else
- Példa (a `pow` függvény deklaratív változata a `powd`):

```
/* powd(a,n) = a^n */
int powd(int a, int n) {
    if (n > 0)                /* Ha n > 0 */
        return a*powd(a,n-1); /* akkor a^n = a*a^(n-1) */
    else
        return 1;            /* egyébként a^n = 1 */
}
```

- A (fenti típusú) rekurzió költséges, nem valósítható meg konstans tárigénnyel :- (.

## Hatékony deklaratív programozás

---

- A rekurzióknak van egy hatékonyan megvalósítható változata

- Példa: döntsük el, hogy egy  $a$  természetes szám előáll-e egy  $b$  szám hatványaként:

```

/* ispow(a,b) = 1 <=> exists i, such that bi = a. Precondition: a,b
> 0 */
int ispow(int a, int b) {
    /* again: */
    if (a == 1)          return 1;
    else if (a%b == 0)  return ispow(a/b, b);    /* a = a/b; goto again; */
    else                return 0;
}

```

- Itt a rekurzív hívás a kommentben jelzett értékadásokkal és ugrással helyettesíthető!
- Ez azért tehető meg, mert a rekurzióból való visszatérés után *azonnal* kilépünk az adott függvényhívásból.
- Az ilyen függvényhívást **jobbrekurzió**nak vagy **terminális rekurzió**nak nevezzük
- A Gnu C fordító megfelelő optimalizálási szint mellett (`gcc -O2`) a rekurzív definícióból is a nem-rekurzív (jobboldali) kóddal azonos kódot generál!

## Jobbrekurzív függvények

---

- Lehet-e jobbrekurzív kódot írni a hatványozási ( $\text{pow}(a, n)$ ) feladatra?
  - A gond az, hogy a rekurzióból „kifelé jövet” már nem csinálhatunk semmit,
  - tehát a végeredménynek az utolsó hívás belsejében elő kell állnia!
  - A megoldás: segédfüggvény definiálása, amelyben további, ún. gyűjtőargumentumokat helyezünk el.
- A  $\text{pow}(a, n)$  jobbrekurzív megvalósítása:

```
/* Segédfüggvény: powa(a, n, p) = p*a^n */
int powa(int a, int n, int p) {
    if (n > 0)
        return powa(a, n-1, p*a);
    else
        return p;
}

int powr(int a, int n){
    return powa(a, n, 1);
}
```

## Cékla: A „Cé” nyelv egy deKLAratív része

---

- **Megszorítások:**
  - **Típusok:** csak `int`
  - **Utasítások:** `if-then-else`, `return`, **blokk**
  - **Feltétel-rész:** ( `< kif>` `< hasonlító-op>` `< kif>` )
    - `< hasonlító-op>`: `<` | `>` | `==` | `\=` | `>=` | `<=`
  - **kifejezések:** változókból és számkonstansokból kétargumentumú operátorokkal és függvényhívásokkal épülnek fel
    - `< aritmetikai-op>`: `+` | `-` | `*` | `/` | `%` |
- Egy Cékla fordító letölthető a tárgy honlapjáról.



# A Cékla nyelv szintaxisa

---

- A szintaxist BNF jelöléssel adjuk meg, kiterjesztés:

- ismétlés (0, 1, vagy többszöri ismétlés): «ismétlendő»... vagy [«ismétlendő»]...

- A program szintaxisa

```

<program> ::=                <function_definition> ...
<function_definition> ::=   <head> <block>
<head> ::=                  <type> <identifier> (<formal_args>)
<type> ::=                  int
<formal_args> ::=           <formal_arg>[, <formal_arg>]... | <empty>
<formal_arg> ::=            <type> <identifier>
<block> ::=                 { <declaration>... <statement>...}
<declaration> ::=          <type> <declaration_elem> <declaration_elem>...;
<declaration_elem> ::=     <identifier> = <expression>

```

## Cékla szintaxis: folytatás

### ● Utasítások szintaxisa

```

<statement> ::=
    if <test> <statement> <optional_else_part>
    | <block>
    | return <expression> ;
    | ;
<optional_else_part> ::=
    else <statement> | <empty>
<test> ::=
    (<expression> <comparison_op> <expression>)
  
```

### ● Kifejezések szintaxisa

```

<expression> ::=
<term> ::=
<factor> ::=
    <term> [<additive_op> <term>]...
    <factor> [<multiplicative_op> <factor>]...
    <identifier>
    | <identifier> (<actual_args>)
    | <constant>
    | (<expression>)
<constant> ::=
    <integer>
<actual_args> ::=
    <expression> [, <expression>]... | <empty>
<comparison_op> ::=
    < | > | == | \= | >= | <=
<additive_op> ::=
    + | -
<multiplicative_op> ::=
    * | / | %
  
```

## Számkonverzió Cékla nyelven

- A feladat: Egy 0 és 1023 közé eső *num* egész számot egy olyan 10-jegyű decimális számmá kell konvertálni, amely csak a 0 és 1 jegyekből áll, úgy, hogy ha ezt a 0-1 sorozatot bináris számként olvassuk ki, akkor a *num* értéket kapjuk, pl.  $\text{bin}(5) = 101$ ,  $\text{bin}(37) = 100101$ .
- Megoldás (imperatív) C-ben és Céklában:

```

int bin(int num) {
    int bp = 512;
    int dp = 1000000000;
    int bin = 0;
    while (bp > 0) {
        if (num >= bp) {
            num = num-bp;
            bin = bin+dp;
        }
        bp = bp / 2;
        dp = dp / 10;
    }
    if (num > 0)
        return -1;
    else
        return bin;
}

int bina(int num,
         int bp,
         int dp,
         int bin) {
    if (bp > 0) {
        if (num >= bp)
            return bina(num-bp, bp/2, dp/10, bin+dp);
        else
            return bina(num, bp/2, dp/10, bin);
    }
    if (num > 0)
        return -1;
    else
        return bin;
}

int bind(int num) {
    return bina(num, 512, 1000000000, 0);
}

```

# Deklaratív programozási nyelvek — a Cékla tanulságai

---

- Mit veszítettünk?
  - a megváltoztatható változókat,
  - az értékadást, ciklus-utasítást stb.,
  - általánosan: a megváltoztatható állapotot
- Hogyan tudunk mégis állapotot kezelni deklaratív módon?
  - az állapotot a (segéd)függvény paraméterei tárolják,
  - az állapot változása (vagy helybenmaradása) explicit!
- Mit nyertünk?
  - Állapotmentes szemantika: egy nyelvi elem értelme nem függ a programállapottól
    - Hivatkozási átlátszóság (referential transparency) — pl. ha  $f(x) = x^2$ , akkor  $f(a)$  **helyettesíthető**  $a^2$ -tel.
    - Egyszeres értékadás (single assignment) — párhuzamos végrehajthatóság.
  - A deklaratív programok **dekomponálhatók**:
    - A program részei egymástól **függetlenül** megírhatók, tesztelhetők, verifikálhatók.
    - A programon könnyű következtetéseket végezni, pl. helyességét bizonyítani.

# Deklaratív programozási nyelvek — jelmondat

---

- MIT és nem HOGYAN (WHAT rather than HOW): a *megoldás módja* helyett inkább a megoldandó *feladat leírását* kell megadni
- A gyakorlatban mindkét szemponttal foglalkozni kell — kettős szemantika:
  - deklaratív szemantika — MIT (milyen feladatot) old meg a program;
  - procedurális szemantika — HOGYAN oldja meg a program a feladatot.

# Deklaratív programozás — miért tanítjuk?

---

- Új, magasszintű programozási elemek
  - rekurzió
  - mintaillesztés
  - visszalépéses keresés
- Új gondolkodási stílus
  - dekomponálható programok: a programrészek (relációk, függvények) önálló jelentéssel bírnak
  - verifikálható programok: a kód és a jelentés összevethető
- Új alkalmazási területek
  - szimbolikus alkalmazások
  - következtetési módszerekre épülő megoldások
  - nagyfokú megbízhatóságot igénylő rendszerek
  - párhuzamosság, pl. többmagú processzorok!

## Egy példa: párbeszéd egy 50 soros Prolog programmal

---

```
| ?- párbeszéd.  
|: Magyar legény vagyok én.  
Felfogtam.  
|: Ki vagyok én?  
Magyar legény  
|: Péter kicsoda?  
Nem tudom.  
|: Péter tanuló.  
Felfogtam.  
|: Péter jó tanuló.  
Felfogtam.  
|: Péter kicsoda?  
tanuló  
jó tanuló  
|: Boldog vagyok.  
Felfogtam.  
|: Te egy Prolog program vagy.  
Felfogtam.  
|: Ki vagyok én?  
Magyar legény  
Boldog  
|: Okos vagy.  
Felfogtam.  
|: Te vagy a világ közepe.  
Felfogtam.  
|: Ki vagy te?  
egy Prolog program  
Okos  
a világ közepe  
|: Valóban?  
Nem értem.  
|: Unlak.  
Én is.
```