

# TÖBB MEGOLDÁS ELŐÁLLÍTÁSA VISSZALÉPÉSSSEL

## *n* vezér a sakktáblán

- Hányféleképpen rakható *n* vezér a sakktáblára úgy, hogy ne üssék egymást?
- A vezéreket tartalmazó mezők sorának *j* sorszámát az egyes oszlopokon belül egy *n* hosszú sorvektor adott oszlophoz rendelt mezőjébe írt szám adja meg, ahol  $j \leq s < n$ .  
Példa *n*=4 esetén:
- A sorvektort (egy egyre bővülő listával valósítjuk meg. Egy listához balról könnyű új elemeket fűzni, ezért a táblát és a vezérek helyzetét leíró listát hossztengelye mentén tükrözzük.

```

+---+---+---+---+
| 3 | 1 | 4 | 2 |
+---+---+---+---+

      0   ---->  n-1
+---+---+---+---+
0 |   | q |   |   |
+---+---+---+---+
| |   |   |   | q |
| +---+---+---+---+
V | q |   |   |   |
+---+---+---+---+
n-1 |   |   | q |   |
+---+---+---+---+

```

```

...+---+---+---+
      | 4 | 1 | 3 |
...+---+---+---+

n-1 <----  0
...+---+---+---+
0   |   | q |   |
...+---+---+---+
|   |   |   |   |
| ...+---+---+---+
V   |   |   | q |
...+---+---+---+
n-1 | q |   |   |
...+---+---+---+

```

## *n* vezér a sakktáblán (folyt.)

Azt, hogy az új vezért üti-e egy korábban a táblára rakott másik vezér, a sorvektor vizsgálatával dönthetjük el: a sorvektor azt adja meg, hogy a listaelem indexe által meghatározott oszlopban és a listaelem értéke által meghatározott sorban vezér van.

1. Az új vezér sorának sorszáma, azaz az új listaelem értéke nem fordulhat elő a lista már felépített részében.
2. Az új vezér átlós irányban sem lehet egy vonalban más vezérrel a táblán. Ez azt jelenti, hogy ha a sorvektort jelentő lista elejére az  $s$  sorindexet akarjuk rakni, akkor az  $i$ -edik elemének az értéke, ha van ilyen eleme, nem lehet  $s - (i + 1)$ , ill.  $s + (i + 1)$ .
3. A következő példa segít megvilágítani az esetet.

Ha a 2-es oszlopba és az  $s=1$ -es sorba akarjuk lerakni az új vezért, akkor az  $x$ -szel jelölt mezőket kell megvizsgálnunk. Az eddig létrehozott listának (sorvektornak) két eleme van, ahol a lista fejének az indexe 0. A listafej értéke nem lehet  $s-1$ , sem  $s+1$ . A lista rekurzív algoritmussal dolgozható fel.

```

...-+----+----+
   s |   |   |
...-+----+----+

n-1 <--- 1   0
...-+----+----+
0   |   | x |   |
...-+----+----+
1   | q |   |   |
...-+----+----+
|   |   | x |   |
V ...-+----+----+
n-1 |   |   | x |
...-+----+----+

```

## *n* vezér a sakktáblán: „ütésben van”-vizsgálat

```

(* utesbenVan : int list -> bool
    utesbenVan zs = igaz, ha a (hd zs) vezért legalább egy
        (tl zs)-beli vezér üti
*)
fun utesbenVan [] = false
  | utesbenVan (z::zs) =
    let (* uV : int -> int -> int list -> bool
        uV s1 s2 rs = igaz, ha a z vezért s1, s2 vagy r, vagy
            egy másik rs-beli vezér közül legalább egy üti
        *)
        fun uV _ _ [] = false
          | uV s1 s2 (r::rs) = z = r orelse
                                s1 = r orelse
                                s2 = r orelse
                                uV (s1-1) (s2+1) rs

        in
            uV (z-1) (z+1) zs
        end

```

## *n* vezér a sakktáblán: egy megoldás előállítása

---

```

exception Zsakutca

(* vezerek0 : int -> int list
   vezerek0 n = a feladvány egy megoldása n vezér esetén
*)
fun vezerek0 n =
  let (* vez0: int -> int list -> int list
       vez0: egy megoldás n vezér esetén
       *)
      fun vez0 z zs =
          if z = 0 andalso utesbenVan zs orelse z = n
          then raise Zsakutca
          else if length zs = n
          then rev zs
          else vez0 0 (z::zs) handle Zsakutca => vez0 (z+1) zs
      in
          vez0 0 []
      end
  end

```

## *n* vezér a sakktáblán: több megoldás előállítása visszalépéssel

---

```

(* vezerek : int -> int list list
   vezerek n = a feladvány összes megoldásának listája
               n vezér esetén
*)
fun vezerek n =
  let (* vez0: int -> int list -> int list list
       vez0: az összes megoldás listája n vezér esetén
       *)
      fun vez0 z zs =
          if z = 0 andalso utesbenVan zs orelse z = n
          then raise Zsakutca
          else if length zs = n
          then [rev zs]
          else (vez0 0 (z::zs) handle Zsakutca => []) @
               (vez0 (z+1) zs handle Zsakutca => [])
      in
          vez0 0 []
      end
  end

```

***n* vezér a sakktáblán: több megoldás előállításával listák listájával**

```
(* vezerek : int -> int list list
   vezerek n = a feladvány összes megoldásának listája
               n vezér esetén
*)
fun vezerek n =
  let (* vez0: int -> int list -> int list list
       vez0: az összes megoldás listája n vezér esetén
     *)
      fun vez0 z zs =
          if z = 0 andalso utesbenVan zs orelse z = n
          then []
          else if length zs = n
          then [rev zs]
          else vez0 0 (z::zs) @ vez0 (z+1) zs
    in
      vez0 0 []
    end
```

***n* vezér a sakktáblán: több megoldás előállításával listák listájával (folyt.)**

Akkumulátor alkalmazásával:

```
(* vezerek : int -> int list list
   vezerek n = a feladvány összes megoldásának listája
               n vezér esetén
*)
fun vezerek n =
  let (* vez0: int -> int list -> int list list -> int list list
       vez0: az összes megoldás listája n vezér esetén
     *)
      fun vez0 z zs zss =
          if z = 0 andalso utesbenVan zs orelse z = n
          then zss
          else if length zs = n
          then rev zs :: zss
          else vez0 0 (z::zs) (vez0 (z+1) zs zss)
    in
      vez0 0 [] []
    end
```

# FÜGGVÉNYEK KOMPOZÍCIÓJA

## Függvények kompozíciója

- Az  $f \circ g$  függvénykompozíció az SML-ben

(\*  $f \circ g$  = az  $f$  és  $g$  függvények kompozíciója \*)

```
infix 2 o;
```

```
fun (f o g) = fn x => f(g x); vagy
```

```
fun (f o g) x = f(g x);
```

- Az  $\circ$  típusa  $? * ? \rightarrow ?$  szerkezetű. Mit írjunk a  $?$ -ek helyébe? Vezessük le!

- A függvénydefiníció jobb oldalán álló kifejezés elemzésével kezdjük.

$x : 'a$        $g : 'a \rightarrow 'b$        $(g\ x) : 'b$        $f : 'b \rightarrow 'c$

- A `fun (f o g) x = f(g x)` függvénydefinícióban az egyenlőségjel (=) bal és jobb oldalán álló kifejezéseknek azonos értéket kell eredményül adniuk, ezért  $f \circ g$  és  $f$  eredményének azonos a típusa (azaz  $'c$ ).

$(f \circ g) : 'a \rightarrow 'c$      $\circ : ('b \rightarrow 'c) * ('a \rightarrow 'b) \rightarrow ('a \rightarrow 'c)$

- Példa: `round : real -> int, chr : int -> char`  
`chr o round : real -> char`

# ÖSSZEFOGLALÓ: TÍPUSOK, ÁLLANDÓK, NEVEK, KÖTÉSEK

## Típusok az SML-ben (összefoglaló)

- Típusok és programozási nyelvek
  - Típus nélküli nyelvek, pl. assembly, LISP, Prolog
  - Gyengén típusos nyelvek, pl. Fortran, Algol, BASIC, C, C++, Pascal
  - Erősen típusos nyelvek, pl. Ada, **SML**, clean, haskell
  - Erős típus: a típusok ( $\sim$  halmazok) diszjunktak, azaz nincs közös elemük
- Egyszerű SML-típusok
  - `int` – előjeles egész szám, a  $Z$  egy részhalmaza
  - `word`, `word8` – előjel nélküli nemnegatív egész szám, az  $N_0$  egy részhalmaza
  - `real` – előjeles racionális (valós?!) szám, a  $Q$  egy részhalmaza
  - `bool`, `char`, `order`, `unit`, `string`, `substring`, `Time.time`, `Date.date`, ...
  - felsorolásos típus, pl. `datatype color = Red | White | Green`
- Polimorf összetett SML-típusok
  - rekord, ennes, pl. `type ('a, 'b) par = 'a * 'b`
  - lista, vektor (nem frissíthető), tömb (frissíthető), pl. `'a list`, `'b vector`, `'c array`
  - fák és más rekurzív típusok, pl. `datatype 'a t = L | N of 'a t * 'a t * 'a`

## Különleges állandók az SML-ben (összefoglaló)

---

- Előjeles egész állandó

Példák: 0 ~0 4 ~04 999999 0xFFFF ~0x1ff

Ellenpéldák: 0.0 ~0.0 4.0 1E0 -317 0XFFFF -0x1ff

- Valós állandó

Példák: 0.7 ~0.7 3.32E5 3E~7 ~3E~7 3e~7 ~3e~7

Ellenpéldák: 23 .3 4.E5 1E2.0 1E+7 1E-7

- Előjel nélküli egész állandó

Példák: 0w0 0w4 0w999999 0wxFFFF 0wx1ff

Ellenpéldák: 0w0.0 ~0w4 -0w4 0w1E0 0wXFFFF 0WXXXX

- Füzérállandó: "-ek között álló nulla vagy több nyomtatható karakter, szóköz vagy \ jellel kezdődő *escape-szekvencia* (l. a táblázatot a következő lapon).

- Karakterállandó: # jelet közvetlenül követő, egykarakteres füzérállandó.

Példák: # "a" # "\n" # "\^Z" # "\255" # "\"

Ellenpéldák: # "a" # c # ""

## Escape-szekvenciák az SML-ben (összefoglaló)

---

- Escape-szekvenciák

\a Csengőjel (BEL, ASCII 7).

\b Visszalépés (BS, ASCII 8).

\t Vízszintes tabulátor (HT, ASCII 9).

\n Újsor, soremelés (LF, ASCII 10).

\v Függőleges tabulátor (VT, ASCII 11).

\f Lapdobás (FF, ASCII 12).

\r Kocsi-vissza (CR, ASCII 13).

\^c Vezérlő karakter, ahol  $64 \leq c \leq 95$  (@ ... \_), és \^c ASCII-kódja 64-gyel kevesebb c ASCII-kódjánál.

\ddd A ddd kódú karakter (d decimális számjegy).

\uxxxx A xxxx kódú karakter (x hexadecimális számjegy).

\" Idézőjel (").

\\ Hátratórt-vonal (\).

\f...f Figyelmen kívül hagyott sorozat. f...f nulla vagy több formázókaraktert (szóköz, HT, LF, VT, FF, CR) jelent.

## Név képzése az SML-ben (összefoglaló)

- Alfanumerikus (*alphanumeric*) név: kis- és nagybetűk, számjegyek, percjelek (') és aláhúzás-jelek (\_) olyan sorozata, amely betűvel vagy perccel kezdődik

● Példák: `tothGyorgy Toth_3_Gyorgy toth'gyorgy`

- Írásjelekből álló (*symbolic*) név: az alábbi írásjelek tetszőleges, nem üres sorozata

! % & \$ # + - / : < = > ? @ \ ~ ' ^ | \*

● Példák: `++ <-> ||| ## |=|`

- Speciális a szerepe az alábbi *fenntartott írásjeleknek*

( ) [ ] { } , ; . ...

- Más jelentés nem rendelhető az alábbi *fenntartott nevekhez*

```
abstype and andalso as case do datatype else end eqtype exception
fn fun functor handle if in include infix infixr let local nonfix
of op open orelse raise rec sharing sig signature struct structure
then type val where with withtype while : :: :> _ | = => -> #
```

## Értékdeklaráció – név kötése értékhez – az SML-ben (összefoglaló)

- Tetszőleges típusú érték köthető tetszőleges névhez:

<code>val harom = 2 + 1</code>	<code>: int</code>	
<code>val MHz = 94.5</code>	<code>: real</code>	
<code>val veege = true</code>	<code>: bool</code>	true, false
<code>val kisA = #"a"</code>	<code>: char</code>	
<code>val palindrom = "ABBA"</code>	<code>: string</code>	
<code>val kisebb = LESS</code>	<code>: order</code>	LESS, EQUAL, GREATER
<code>val ezNemSemmi = ()</code>	<code>: unit</code>	Egyetlen értéke a (), azaz nulla
<code>val rat = {num = 3, den = 4}</code>	<code>: {den : int, num : int}</code>	Mezőnevek ábécében.
<code>val bLista = [2,3,4] @ [3,2]</code>	<code>: int list</code>	
<code>val telenek = [0w123, 0wxcd]</code>	<code>: word list</code>	

- Függvényérték is köthető tetszőleges névhez:

<code>val incMod = fn i =&gt; fn n =&gt; (i + 1) mod n</code>	<code>: int -&gt; int -&gt; int</code>	λ-jelöléssel
<code>fun incMod i n = (i + 1) mod n</code>	<code>: int -&gt; int -&gt; int</code>	A szokásos alakba

- Típusmegkötés:

<code>val id = fn (n : int) =&gt; n</code>	<code>: int -&gt; int</code>	Pl. <code>id 3</code> , de <code>id 4.5</code> nem!
<code>val telenek = [0w65, 0wx41 : word8]</code>	<code>: word8 list</code>	

## Függvénynév helyzete és kötése

- Függvénynév (másnéven függvényjel) helyzete és kötése általában
  - Egy függvénynév *prefix*, *infix* vagy *postfix* helyzetű lehet.
  - Az infix helyzetű függvénynevet gyakran *operátornak* nevezik.
  - Egy (infix helyzetű) operátor lehet *asszociatív* vagy *nem-asszociatív*; köthet balra vagy jobbra, vagy semerre. Asszociatív operátor esetén a kötési iránynak nincs jelentősége.
- Infix Prolog-operátor kötése
  - $xfx = f$  mindkét oldalán  $f$  csak zárójelben ismétlődhet ( $f$  „nem köt”),
  - $yfx = f$  bal oldalán  $f$  zárójelezés nélkül ismétlődhet ( $f$  „balra köt”),
  - $xfy = f$  jobb oldalán  $f$  zárójelezés nélkül ismétlődhet ( $f$  „jobbra köt”).

## Függvénynév helyzete és kötése az SML-ben

- Kifejezések és típuskifejezések az SML-ben
  - Az SML-ben a szokásos kifejezések mellett vannak *típuskifejezések* is.
  - A függvények *értékekre*, a típusfüggvények *típusokra* alkalmazhatók.
- Függvénynév és típusfüggvénynév helyzete és kötése az SML-ben
  - Függvénynév: *prefix* vagy *infix*.
  - Típusfüggvénynév: *infix* vagy *postfix*.
  - Az *infix* helyzetű függvénynév és típusfüggvénynév (szokásos nevén operátor, ill. típusoperátor) balra vagy jobbra köt, vagy semerre nem köt.
- Típusoperátorok
  - A két infix helyzetű beépített típusoperátor közül a  $\rightarrow$  jobbra, a  $*$  semerre nem köt.
  - A  $*$  operátornak magasabb a precedenciája, mint a  $\rightarrow$  operátornak.
  - A típusoperátoroknak magasabb a precedenciájuk a többi operátorénál.
  - Példák:
 

<code>int * real * string</code>	Egy hármas
<code>int * (real * string)</code>	Egy pár, amelynek a második tagja is egy pár
<code>(int * real) * string</code>	Egy pár, amelynek az első tagja is egy pár
<code>int * real -&gt; string</code>	Függvény, amelynek egy pár az argumentuma
<code>int * (real -&gt; string)</code>	Egy pár, amelynek a második tagja egy függvény

## Függvénynév helyzete és kötése az SML-ben

- Tetszőleges kétargumentumú függvénynévet lehet meghatározott preferenciájú (infix helyzetű) operátorként deklarálni az `infix` vagy az `infixr` deklaratívával.
- Az `infix` balra, az `infixr` jobbra kötő operátort deklarál.
- Az `op` deklaratíva egy (esetleg *infix* helyzetű) operátort átmenetileg *prefix* helyzetűvé alakít.
- A `nonfix` deklaratíva egy (esetleg *infix* helyzetű) operátort tartósan *prefix* helyzetűvé alakítja.
- Egy minősített nevet, vagy egy olyan nevet, amelyet az `op` deklaratíva előz meg, csak *prefix* helyzetben lehet alkalmazni.
- Táblázatban összefoglalva:
 

<code>infix &lt;d&gt; id<sub>1</sub>...id<sub>n</sub></code>	balra köt	binds to the left
<code>infixr &lt;d&gt; id<sub>1</sub>...id<sub>n</sub></code>	jobbra köt	binds to the right
<code>nonfix id<sub>1</sub>...id<sub>n</sub></code>	prefix	prefix
- A táblázatban `idi` tetszőleges nevet jelöl ( $n \geq 1$ ). A `d` 0 és 9 közötti számjegy, az operátor precedenciája (opcionális, alapértelmezés szerinti értéke 0). Nagyobb szám nagyobb precedenciát jelent (éppen fordítva, mint a Prologban!).

## A beépített operátorok és precedenciájuk az SML-ben

Az alábbi táblázatban `wordint`, `num` és `numtxt` az alábbi típusnevek helyett állnak.

`wordint` = `int`, `word`, `word8`.      `num` = `int`, `real`, `word`, `word8`.  
`numtxt` = `int`, `real`, `word`, `word8`, `char`, `string`.

Prec.	Operátor	Típus	Eredmény	Kivétel
<b>7</b>	*	<code>num * num -&gt; num</code>	szorzat	Overflow
	/	<code>real * real -&gt; real</code>	hányados	Div, Overflow
	<code>div</code> , <code>mod</code>	<code>wordint * wordint -&gt; wordint</code>	hányados, maradék	Div, Overflow
	<code>quot</code> , <code>rem</code>	<code>int * int -&gt; int</code>	hányados, maradék	Div, Overflow
<b>6</b>	<code>+</code> , <code>-</code>	<code>num * num -&gt; num</code>	összeg, különbség	Overflow
	<code>^</code>	<code>string * string -&gt; string</code>	egybeírt szöveg	Size
<b>5</b>	<code>::</code>	<code>'a * 'a list -&gt; 'a list</code>	elemmel bővített lista (jobbra köt)	
	<code>@</code>	<code>'a list * 'a list -&gt; 'a list</code>	összefűzött lista (jobbra köt)	
<b>4</b>	<code>=</code> , <code>&lt;&gt;</code>	<code>'a * 'a -&gt; bool</code>	egyenlő, nem egyenlő	
	<code>&lt;</code> , <code>&lt;=</code>	<code>numtxt * numtxt -&gt; bool</code>	kisebb, kisebb-egyenlő	
	<code>&gt;</code> , <code>&gt;=</code>	<code>numtxt * numtxt -&gt; bool</code>	nagyobb, nagyobb-egyenlő	
<b>3</b>	<code>:=</code>	<code>'a ref * 'a -&gt; unit</code>	értékkadás	
	<code>o</code>	<code>('b -&gt; 'c) * ('a -&gt; 'b)</code> <code> -&gt; ('a -&gt; 'c)</code>	a két függvény kompozíciója	
<b>0</b>	<code>before</code>	<code>'a * 'b -&gt; 'a</code>	a bal oldali argumentum	

# EGYSZERŰSÍTETT SML-SZINTAXIS

## SML-szintaxis: szintaktikai kategóriák (egyszerűsítve)

- A nevek *szintaktikai kategóriákba* sorolhatók
  - vid** értéknév value identifier long
  - tyvar** típusváltozó type variable
  - tycon** típuskonstruktor type constructor long
  - lab** mezőnév record label
  - strid** struktúranév structure identifier long
  - sigid** szignatúranév signature identifier
  - unitid** állománynév unit identifier
- Az *értéknév* tetszőleges név; jelölhet állandó értéket, függvényértéket, adatkonstruktor, kivételkonstruktor. Példák: pi, +, sin, nil, true, Match
- A *típusváltozó* perccel kezdődő alfanumerikus név. Példa: ' a.
- A *típuskonstruktor* tetszőleges név; jelölhet típusállandót vagy típusfüggvényt. Példák: int, order, \*, ->, list
- A *mezőnév* tetszőleges név vagy (nem 0-val kezdődő) pozitív egész szám. Példák: num, 2
- A *strid* tetszőleges név: SML-kódot tartalmazó „implementációs” modult (ún. SML-struktúrát) azonosít. A fájlnev-kiterjesztés célszerűen .sml, de ez nem kötelező. Példák: Int.sml, Rat.ml

## SML-szintaxis: szintaktikai kategóriák (folyt.)

- A *sigid* tetszőleges név: SML-kódot tartalmazó „interfész”- modul (ún. SML-szignatúrát) azonosít. A fájlnev-kiterjesztés kötelezően `.sig`. Példák: `Int.sig`, `Rat.sig`.
- A *unitid* egy struktúra, ill. szignatúra lefordításával létrejövő, tárgykódot tartalmazó fájl neve. A fájlnev-kiterjesztés szignatúra esetén `.ui`, struktúra esetén `.uo`. Példák: `Int.ui`, `Rat.uo`.
- Minden, az előző felsorolásban „long”-gal megjelölt  $X$  szintaktikai kategóriának van egy *longX* párja. A *longX* szintaktikai kategóriába tartozó nevek rövid és hosszú (ún. minősített) alakban is felírhatók. A *rövid alak* csak egy névből, a *hosszú alak* egy hosszú struktúranévből, egy pontból és egy névből áll:

$$\begin{array}{l|l|l} \textit{longx} ::= x & \text{név} & \text{identifier} \\ & \textit{longstrid.x} \text{ teljes név} & \text{qualified identifier} \end{array}$$

Példák:

- `explode, Misc.explode`
- `Real.toString`
- `Int.+`
- `List.filter`

## SML-szintaxis: nemterminális szimbólumok, nyelvtani jelölések

- Minden nemterminális szimbólumot *változatok* sorozataként definiálunk, soronként egy változattal. Üres sor üres változatot jelent.
- A `<` és a `>` csúcsos zárójelpárok opcionális kifejezést fognak közre.
- Bármely  $X$  nemterminális szimbólumra az alábbiak szerint definiáljuk az *Xseq* nemterminális szimbólumot:

$$\begin{array}{l|l|l} \textit{Xseq} ::= X & \text{egyelemű sorozat} & \text{singleton sequence} \\ & \text{üres sorozat} & \text{empty sequence} \\ & X_1, \dots, X_n \text{ sorozat, } n \geq 1 & \text{sequence, } n \geq 1 \end{array}$$

- A változatokat csökkenő prioritási sorrendben soroljuk föl.
- A változatokat számozzuk, a példákban utalunk az alkalmazott változatra.
- A függvénynevek és operátorok általában balra kötnek, az eltérést jelezzük.
- Minden ismétlődő konstrukció (pl. a *klózsorozat*) a lehető legmesszebb terjeszkedik jobbra. Ezért pl. egy *case*-kifejezést egy másik *case*- vagy *fn*-kifejezésen, valamint egy *fun*-definíción belül zárójelbe kell tenni.

## SML-szintaxis: kifejezések és klózsorozatok (egyszerűsítve)

### • Kifejezés (*exp*: expression)

(1) $exp ::= infexp$		
(2) $exp : ty$	típusmegkötés	type constraint
(3) $raise\ exp$	kivételjelzés	raise exception
(4) $case\ exp\ of\ match$	esetszétválasztás	case analysis
(5) $fn\ match$	függvénykifejezés	function expression

### • Példák:

```
fn (n : int) => n;                                vö. (2), (5)
case c of 00 => 01 | 01 => 11 | 11 => 10 | 10 => 00; vö. (4), (19)
fn 00 => 01 | 01 => 11 | 11 => 10 | 10 => 00;      vö. (5), (19)
fn 00 => 01 | 01 => 11 | 11 => 10 | 10 => 00
    | _ => raise Domain; vö. (3), (5), (19)
```

## SML-szintaxis: kifejezések és klózsorozatok (folyt.)

### • Infix kifejezés (*infexp*: infix expression)

(6) $infexp ::= appexp$		
(7) $infexp_1\ id\ infexp_2$	infix alkalmazás	infix application

### • Applikatív kifejezés (*appexp*: applicative expression)

(8) $appexp ::= atexp$		
(9) $appexp\ atexp$	(prefix) alkalmazás	(prefixed) application

### • Példák:

```
3 + 4;                                vö. (7)
Real.toString 3.56;                    vö. (9)
Int.toString(round 3.56); vö. (9), (17)
```

## SML-szintaxis: kifejezések és klózsorozatok (folyt.)

### • Atomi kifejezés (*atexp*: atomic expression)

(10)	$atexp ::= scon$	különleges állandó	special constant
(11)	$<op> longvid$	értéknév	value identifier
(12)	$\{<exprow>\}$	rekord	record
(13)	$\# lab$	rekordszelektor	record selector
(14)	$(exp_1, exp_2)$	pár	pair
(15)	$()$	nullas	0-tuple
(16)	$[exp_1, \dots, exp_n]$	lista, $n \geq 0$	list, $n \geq 0$
(17)	$(exp)$	kifejezés zárójelben	parenthesized expr.

### • Példák:

<code>1.12, #"Z", 0w123</code>	vö. (10)
<code>Math.pi, false, Math.sin, sin</code>	vö. (11)
<code>#den {num=1, den=2}</code>	vö. (12), (13), (18)
<code>(2, 3.5), (), [1, 2, 3]</code>	vö. (14), (15), (16)

## SML-szintaxis: kifejezések és klózsorozatok (folyt.)

### • Kifejezéssor (*exprow*: expression row)

(18)  $exprow ::= lab = exp <, exprow>$

### • Klózsorozat (*match*)

(19)  $match ::= mrule <| match>$

### • Klóz (*mrule*: match rule)

(20)  $mrule ::= pat => exp$

### • Példák:

<code>num=1, den=2</code>	vö. (18)
<code>00 =&gt; 01   01 =&gt; 11   11 =&gt; 10   10 =&gt; 00</code>	vö. (19), (20)

## SML-szintaxis: deklarációk és kötések

### • Deklaráció (*dec*: declaration)

(20)	$dec ::= \text{val } tyvarseq \text{ valbind}$	értékdeklaráció	value declaration
(21)	$\text{fun } tyvarseq \text{ fvalbind}$	függvénydeklaráció	function declaration
(22)	$\text{type } tyvarseq \text{ typbind}$	típusdeklaráció	type declaration
(23)		üres deklaráció	empty declaration
(24)	$dec_1 <;> dec_2$	deklaráció-sorozat	sequential declaration
(25)	$\text{infix } <d> id_1 \dots id_n$	infix-direktíva, $n \geq 1$	infix (left) directive
(26)	$\text{infixr } <d> id_1 \dots id_n$	infixr-direktíva, $n \geq 1$	infix (right) directive
(27)	$\text{nonfix } id_1 \dots id_n$	nonfix-direktíva, $n \geq 1$	nonfix directive

### • Példák:

```
val xy = "XY"; fun ++ x y = x ^ y   vö. (20), (21), (24)
type Rat = {num : int, den : int}   vö. (22)
infixr 4 ++; fun x ++ y = x ^ y    vö. (21), (26)
```

## SML-szintaxis: deklarációk és kötések (folyt.)

### • Értékkötés (*valbind*: value binding)

(28)	$valbind ::= pat = exp <\text{and } valbind>$	értékkötés	value binding
(29)	$\text{rec } valbind$	rekurzív kötés	recursive binding

### • Függvényérték-kötés (*fvalbind*: function value binding)

(30)	$fvalbind ::=$	$<op> \text{var } atpat_{11} \dots atpat_{1n} <:ty> = exp_1$ $  <op> \text{var } atpat_{21} \dots atpat_{2n} <:ty> = exp_2$ $  \dots$ $  <op> \text{var } atpat_{m1} \dots atpat_{mn} <:ty> = exp_m$ $<\text{and } fvalbind>$	$m, n \geq 1$
------	----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------

**Megjegyzés:** Ha *var* infix, akkor egy *fvalbind* definícióban vagy infix helyzetben kell használni, vagy elé kell írni az *op* direktívát; azaz a definícióban a bal oldalon (*atpat var atpat'*) vagy *op var (atpat, atpat')* írható. A zárójelk elhagyhatók, ha *atpat'* után közvetlenül *:ty* vagy *=* áll.

### • Példák:

```
val even = fn 0 => true | x => not(odd(x-1))
and odd = fn 0 => false | y => not(even(y-1)); vö. (28)
fun (f o g) x = g(f x); vö. (30)
```

# LISTÁK RENDEZÉSE

Listák rendezése FP-226

## Listák rendezése (folyt.)

- `inssort` (beszúró rendezés),
- `selSort` (kiválasztó rendezés),
- `quicksort` (gyorsrendezés),
- `tmsort` (felülről lefelé haladó összefésülő rendezés),
- **`bmsort` (alulról felfelé haladó összefésülő rendezés),**
- **`smsort` (simarendezés).**

## Összefésülő rendezések (folyt.)

- Az összefésülő rendezéshez kell egy olyan függvény, amely két listát növekvő sorrendben egyesít. Korábban már definiáltuk a `merge` függvényt.

```
(* merge(xs, ys) = xs és ys elemeinek <= szerint
    egyesített listája
   merge : int list * int list -> int list
*)
fun merge (xxs as x::xs, yys as y::ys)=
  if x <= y
  then x::merge(xs, yys)
  else y::merge(xxs, ys)
| merge ([], ys) = ys
| merge (xs, []) = xs
```

- Korlátot jelent, ha a részeredményeket a veremben tároljuk.
- Akkumulátor használata esetén meg kell fordítani az eredménylistát.

## Alulról fölfelé haladó összefésülő rendezés

- Az alulról fölfelé haladó összefésülő rendezés (*bottom-up merge sort*) legegyszerűbb változata az eredeti  $k$  hosszúságú listát  $k$  darab egyelemű listára bontja, majd a szomszédos listákat összefuttatja, így 2, 4, 8, 16 stb. elemű listákat állít elő.
- R. O'Keefe algoritmus (1982) lépésről lépésre futtatja össze az egyforma hosszú részlistákat, de csak az utolsó lépésben rendezi az összeset. Az alábbi példában az összefuttatott részlistákat *egymás mellé írással* jelöljük:

```
A B C D E F G H I J K
AB  C D E F G H I J K
AB  CD E F G H I J K
ABCD  E F G H I J K
ABCD  EF  G H I J K
ABCD  EF  GH  I J K
ABCD  EFGH  I J K
ABCDEF  GH  I J K
ABCDEF  GH  IJ  K
...
```

## Alulról fölfelé haladó összefésülő rendezés (folyt.)

---

- `bmsort` a `sorting` segédfüggvényt használja, amelynek
  - első argumentuma a rendezendő lista,
  - második argumentuma a már rendezett részlisták akkumulátora,
  - harmadik argumentuma az adott lépésben összefuttatandó elem sorszama.

```
(* bmsort xs = az xs elemeinek a <= reláció szerint
      rendezett listája
   bmsort : int list -> int list
*)
fun bmsort xs = sorting(xs, [], 0)
```

## Alulról fölfelé haladó összefésülő rendezés (folyt.)

---

- Ha a rendezendő lista (`xs`) még nem fogyott el, soron következő eleméből `sorting` egyelemű listát (`[x]`) képez, és ezt a már rendezett részlisták listája (`lss`) elé fűzve meghívja a `mergepairs` segédfüggvényt. `mergepairs` az argumentumként átadott lista két azonos hosszúságú bal oldali részlistáját fűzi egybe, feltéve persze, hogy vannak ilyenek. `k` az éppen átadott elem sorszama. Ha a rendezendő lista kiürült, `sorting` a kétszintű lista egyetlen elemét, a rendezett listát adja eredményül.

```
(* sorting(xs, lss, k) = a még rendezetlen xs lista elemeit
      berakja a rendezett részlisták összesen
      már k elemet tartalmazó lss listájába
   sorting : int list * int list list * int -> int list
   PRE: k >= 0
*)
fun sorting (x::xs, lss, k) =
      sorting(xs, mergepairs([x]::lss, k+1), k+1)
  | sorting ([], lss, k) = hd(mergepairs(lss, 0))
```

## Alulról fölfelé haladó összefésülő rendezés (folyt.)

---

- `mergepairs` egyetlen listában gyűjti a már összefuttatott részlistákat. Az éppen átadott elem  $k$  sorszámából dönti el, hogy mit kell csinálnia a következő részlistával.

```
(* mergepairs(llss, n)= az n elemet tartalmazó, már
    rendezett llss lista első két részlistáját,
    ha egyforma a hosszuk, összefuttatja
mergepairs : int list list -> int list list
PRE: n >= 0
```

\*)

```
fun mergepairs (llss as ls1::ls2::lss, n) =
  (* legalább kételemű a lista *)
  if n mod 2 = 1
  then llss
  else mergepairs(merge(ls1, ls2)::lss, n div 2)
| mergepairs (lss, _) = lss (* egyelemű a lista *)
```

- Ha  $n$  páratlan, `mergepairs` a listát változtatás nélkül adja vissza, ha páros, akkor az `llss` lista elején álló két, egyforma hosszú listát egyetlen rendezett listává futtatja össze.  $n=0$ -ra `mergepairs` az összes listák listáját olyan listává futtatja össze, amelynek egyetlen eleme maga is lista.

## Alulról fölfelé haladó összefésülő rendezés (folyt.)

---

- A legrosszabb esetben  $O(n \cdot \log n)$  lépésre van szükség.
- A függvények működését egy példán is bemutatjuk. A kezdőhívás legyen

```
bmsort [1,2,3,4,5,6,7,8,9]
----> sorting ([1,2,3,4,5,6,7,8,9], [], 0)
```

- Amíg `sorting` első argumentuma a nem üres ( $x::xs$ ) lista, `sorting` saját magát hívja meg. A rekurzív hívás
  - első argumentuma a lépésenként egyre rövidülő  $xs$  lista,
  - második argumentuma a `mergepairs([x]::lss, k+1)` függvényalkalmazás eredménye, ahol kezdetben  $lss = []$ ,
  - harmadik argumentuma  $(k+1)$  a már feldolgozott listaelemek száma.

```
fun sorting (x::xs, lss, k) =
  sorting(xs, mergepairs([x]::lss, k+1), k+1)
| sorting ([], lss, k) = hd(mergepairs(lss, 0))
```

## Alulról fölfelé haladó összefésülő rendezés (folyt.)

- A következő táblázatos elrendezés
  - `mergepairs` mindkét argumentumát,
  - a rekurzív `sorting` hívás itt `j`-vel jelölt 3. argumentumát, `k+1`-et, és
  - bináris számként `k`-t mutatja lépcsőről lépésre.
- A `sorting` függvény hívja `mergepairs`-t azokban a sorokban, amelyekben a `j` új értéket vesz föl, a többi helyen `mergepairs` hívása rekurzív.
- Ne feledjük, hogy `mergepairs`-nek listák listája az első argumentuma!
- A táblázat utolsó oszlopa a vonatkozó magyarázatra hivatkozik.
- Vegyük észre, hogy kapcsolat van az `l1ss` első eleme utáni listaelemek hossza és a `k` bitjei között! Ha `k` valamelyik bitje 1, akkor (balról jobbra haladva) az `l1ss` megfelelő listaelemének a hossza az adott bit helyiértékével egyenlő. A 0 értékű biteknek megfelelő listaelemek „hiányoznak” `l1ss`-ből.

```
fun sorting (x::xs, lss, k) =
    sorting(xs, mergepairs([x]::lss, k+1), k+1)
  | sorting([], lss, k) = hd(mergepairs(lss, 0))
```

## Alulról fölfelé haladó összefésülő rendezés (folyt.)

<code>l1ss</code>	<code>n</code>	<code>j</code>	<code>k</code>		<code>fun sorting (x::xs, lss, k) =</code>
<code>[[1]]</code>	1	1	0	m1	<code>    sorting(</code>
<code>[[2], [1]]</code>	2	2	1	m2	<code>        xs,</code>
<code>[[1, 2]]</code>	1			m3	<code>        mergepairs([x]::lss, k+1),</code>
<code>[[3], [1, 2]]</code>	3	3	10	m3	<code>        k+1</code>
<code>[[4], [3], [1, 2]]</code>	4	4	11	m2	<code>    )</code>
<code>[[3, 4], [1, 2]]</code>	2			m2	<code>    sorting([], lss, k) =</code>
<code>[[1, 2, 3, 4]]</code>	1			m3	<code>    hd(mergepairs(lss, 0))</code>
<code>[[5], [1, 2, 3, 4]]</code>	5	5	100	m3	<b>m1:</b> Az argumentumként átadott listának egyetlen eleme van (maga is lista), ezért az argumentumot <code>mergepairs</code> második klóza változtatás nélkül visszaadja az őt hívó <code>sorting</code> -nak.
<code>[[6], [5], [1, 2, 3, 4]]</code>	6	6	101	m2	<b>m2:</b> <code>n</code> páros, ez azt jelzi, hogy az argumentumként átadott lista első két eleme egyforma hosszú lista, amelyeket <code>merge</code> egyetlen rendezett listává futtat össze, majd az eredménnyel <code>mergepairs</code> első klóza meghívja saját magát.
<code>[[5, 6], [1, 2, 3, 4]]</code>	3			m3	<b>m3:</b> <code>n</code> páratlan, ez azt jelzi, hogy az argumentumként átadott lista első két eleme nem egyforma hosszú lista, ezért az argumentumot <code>mergepairs</code> első klóza változtatás nélkül visszaadja az őt hívó <code>sorting</code> -nak.
<code>[[7], [5, 6], [1, 2, 3, 4]]</code>	7	7	110	m3	<b>m4:</b> <code>n=0</code> , az összes listák listáját olyan listává kell összefuttatni, amelynek egyetlen lista az eleme.
<code>[[8], [7], [5, 6], [1, 2, 3, 4]]</code>	8	8	111	m2	
<code>[[7, 8], [5, 6], [1, 2, 3, 4]]</code>	4			m2	
<code>[[5, 6, 7, 8], [1, 2, 3, 4]]</code>	2			m2	
<code>[[1, 2, 3, 4, 5, 6, 7, 8]]</code>	1			m3	
<code>[[9], [1, 2, 3, 4, 5, 6, 7, 8]]</code>	9	9	1000	m3	
<code>[[9], [1, 2, 3, 4, 5, 6, 7, 8]]</code>	0	0		m4	
<code>[[1, 2, 3, 4, 5, 6, 7, 8, 9]]</code>					

## Simarendezés

- Az applikatív simarendezés (*smooth sort*) algoritmus a O'Keefe alulról fölfelé haladó rendezéséhez hasonló, de nem egyelemű listákat, hanem növekvő *futamokat* állít elő.
- Ha a futamok száma  $n$ -től független, azaz a lista majdnem rendezve van, akkor az algoritmus végrehajtási ideje  $O(n)$ , és a legrosszabb esetben is legfeljebb csak  $O(n \cdot \log n)$ .

(\* nextrun (run, xs) = olyan pár, amelynek első tagja xs egy növekvő sorrendű futama, második tagja pedig xs maradéka

```
nextrun : int list * int list -> int list * int list
```

\*)

```
fun nextrun (run, x::xs) =
  if x < hd run
  then (rev run, x::xs)
  else nextrun(x::run, xs)
| nextrun (run, []) = (rev run, [])
```

- nextrun eredménye egy pár, amelynek első tagja a futam (egy növekvő számsorozat), a második tagja pedig a rendezendő lista maradéka.
- A futam csökkenő sorrendben bővül, kilépéskor a futamot meg kell fordítani.

## Simarendezés (folyt.)

- smsorting a futamokat ismételten előállítja és összefuttatja:

(\* smsorting(xs, lss, k) = a még rendezetlen xs lista elemeit berakja a rendezett részlisták összesen már k elemet tartalmazó lss listájába

```
smsorting : int list * int list list * int -> int list
```

```
PRE: k >= 0
```

\*)

```
fun smsorting (x::xs, lss, k) =
  let val (run, tail) = nextrun([x], xs)
  in smsorting(tail, mergepairs(run::lss, k+1), k+1)
  end
| smsorting ([], lss, k) = hd(mergepairs(lss, 0))
```

- (\* smsort xs = az xs elemeinek  $\leq$  szerint rendezett listája

```
smsort : int list -> int list
```

\*)

```
fun smsort xs = smsorting(xs, [], 0)
```

- A simarendezés egy változata sort néven megtalálható a Listsort könyvtárban.

## A futási idők összehasonlítása

---

```

fun futIdo2 (sort, sortFn) (xs, kind) =
  let val starttime = Timer.startCPUTimer()
      val zs = sort xs
      val usr=tim,... = Timer.checkCPUTimer starttime
  in "Int sort with " ^ sortFn ^ ", length = " ^ Int.toString(length xs) ^
    " (" ^kind ^ "), time = " ^ Time.fmt 2 tim ^ " sec\n"
  end

val t101 = futIdo2 (tmsort, "tmsort")
  ((Random.rangelist (1, 100000) (100000, Random.newgen())), "random");
val t102 = futIdo2 (bmsort, "bmsort")
  ((Random.rangelist (1, 100000) (100000, Random.newgen())), "random");
val t103 = futIdo2 (smsort, "smsort")
  ((Random.rangelist (1, 100000) (100000, Random.newgen())), "random")

Int sort with tmsort, length = 100000 (random), time = 10.96 sec
Int sort with bmsort, length = 100000 (random), time = 7.69 sec
Int sort with smsort, length = 100000 (random), time = 7.70 sec
Int sort with quicksort2, Int.compare,
      length = 100000 (random), time = 11.98 sec
Int sort with Listsort.sort, Int.compare,
      length = 100000 (random), time = 14.17 sec

```

## SML-szintaxis: típuskifejezések

### • Típus (*ty*: type)

(31)	$ty ::= tyvar$	típusváltozó	type variable
(32)	$tycon$	típuskonstruktor	type constructor
(33)	$\{ \langle tyrow \rangle \}$	rekordtípus-kifejezés	record type expression
(34)	$ty_1 * ty_2$	pár-típus	pair type
(35)	$ty_1 \rightarrow ty_2$	függvénytípus-kifejezés	function type expression
(36)	$( ty )$	típus zárójelben	parenthesized type

### • Típuskifejezés-sor (*tyrow*: type-expression row)

(37)  $tyrow ::= lab : ty \langle , tyrow \rangle$

### • Példák:

'a, 'c, 'gamma vö. (31)  
 int, real, word, word8, char, bool, string, order vö. (32)  
 int \* int -> int, unit -> unit vö. (34), (35)  
 ('a -> 'b) -> ('a list -> 'b list) vö. (35), (36)  
 {num : int, den : int}, num : int, den : int vö. (33), (37)

## SML-szintaxis: minták

### • Atomi minta (*atpat*: atomic pattern)

(38)	$atpat ::= \_$	mindenesjel	wildcard
(39)	$scon$	különleges állandó	special constant
(40)	$\langle op \rangle longid$	értéknév	value identifier
(41)	$\{ \langle patrow \rangle \}$	rekord	record
(42)	$( pat_1 * pat_2 )$	pár	pair
(43)	$( ), \{ \}$	nullas	0-tuple
(44)	$[ pat_1, \dots, pat_n ]$	lista, $n \geq 0$	list, $n \geq 0$
(45)	$( pat )$	minta zárójelben	parenthesized pattern

### • Példák:

fun le GREATER = false | le EQUAL = true | le LESS = true; vö. (40)  
 fun le GREATER = false | le \_ = true; vö. (38), (40)  
 fun neg Bool.false = true | neg (true) = Bool.false; vö. (40), (45)  
 fun prod [a, b] = a\*b | prod [a, b, c] = a\*b\*c  
 | prod [a] = a | prod () = 1; vö. (43), (44)

## SML-szintaxis: minták (folyt.)

### • Mintasor (*patrow*: pattern row)

(46) <i>patrow</i> ::= ...	mindenesjel	wildcard
(47) <i>lab</i> = <i>pat</i> <, <i>patrow</i> >	mintasor	pattern row
(48) <i>lab</i> <: <i>ty</i> > <, <i>patrow</i> >	mezőnév mint változó	label as variable

### • Példák:

```

fun // {den = 0, ...} = raise Domain
  | // {num = n, den = d} = (real n) / (real d);   vö. (46), (47)
fun // {den = 0, ...} = raise Domain
  | // {num, den} = (real num) / (real den);      vö. (46), (48)

```

## SML-szintaxis: minták (folyt.)

### • Minta (*pat*: pattern)

(49) <i>pat</i> ::= <i>atpat</i>	atomi minta	atomic pattern
(50) <op> <i>longvid</i> <i>atpat</i>	értékkonstrukció	value construction
(51) <i>pat</i> <sub>1</sub> <i>vid</i> <i>pat</i> <sub>2</sub>	infix értékkonstrukció	infix value constr.
(52) <i>pat</i> : <i>ty</i>	minta típusmegkötéssel	typed pattern
(53) <op> <i>var</i> <: <i>ty</i> > as <i>pat</i>	réteges minta	layered pattern

### • Példa:

```

fun sum [] = 0                                     vö. (50)
  | sum [a : real] = a                             vö. (52)
  | sum (x::z::(yxs as y::xs)) = x + z + sum yxs   vö. (51), (53)
  | sum (x::y::xs) = x + y + sum xs               vö. (51)
  | sum (op::(x, xs)) = x + sum xs                vö. (50)

```

## SML-szintaxis: szintaktikai korlátozások

---

- Nem illeszthető minta kétszer ugyanarra a névre (*vid*). Nem illeszthető kifejezősor, mintasor vagy típuskifejezés-sor kétszer ugyanarra a mezőnévre (*lab*).
- Ugyanaz a név nem köthető le kétféleképpen egy *valbind*, *typbind*, *datbind* vagy *exbind* deklarációban. A *datbind* deklarációban ugyanez érvényes az adatkonstruktorokra is.
- Ugyanaz a típusváltozó (*tyvar*) nem szerepelhet kétszer egy *tyvarseq* sorozatban valamely *typbind* vagy *datbind* deklaráció bal oldali *tyvarseq tycon* részében. Minden olyan típusváltozónak (*tyvar*), amelyik előfordul a jobb oldalon, szerepelnie kell *tyvarseq*-ben.
- A *rec*-et követő minden *pat = exp* értékkötésben az *exp*-nek, szükség esetén zárójelben, *fn match* alakúnak kell lennie, ahol egy vagy több névhez típusmegkötés is társítható.
- *true*, *false*, *nil*, *::* és *ref* nem kaphat értéket *valbind*, *datbind* vagy *exbind*, *it* pedig *datbind* vagy *exbind* deklarációban.

## Modulok

---

### Mi a modul?

- fordítási egység
- a csoportosítás és az *elrejtés*, azaz az *absztrakció* eszköze

### Modulok az SML-ben

- Szignatúra (*signature*): specifikációs modul; a struktúra *specifikációja*, „*típusa*”.
- Struktúra (*structure*): implementációs modul; a szignatúra *megvalósítása*.
- Funktor (*functor*): generikus, azaz *struktúrával* paraméterezhető modul; eredménye is egy *struktúra*.

Egy struktúra akkor és csak akkor valósít meg egy szignatúrát, ha a struktúra kielégíti a szignatúra által támasztott összes követelményt. (Később pontosítjuk.)

## Szignatúra és struktúra

---

### A szignatúra alapváltozata

`sig specs end` alakú *szignatúrakifejezés*,

ahol a *specs* specifikációsorozat az alábbi elemeket tartalmazhatja:

- típusspecifikáció `type (tyvar1, ..., tyvarn) tycon [ = typ ]` alakban, ahol *typ* opcionális;
- adattípus-specifikáció (a *datatype*-deklarációval azonos alakban);
- kivételspecifikáció `exception excon of typ` alakban;
- értékspecifikáció `val id : typ` alakban.

### A struktúra alapváltozata

`struct decs end` alakú *struktúrakifejezés*,

ahol a *decs* deklarációsorozat az alábbi elemeket tartalmazhatja:

- típuskonstruktort létrehozó típusdeklaráció;
- új (felhasználói) adattípust létrehozó adattípus-deklaráció (*datatype*-deklaráció);
- kivételkonstruktort (állandót vagy függvényt) létrehozó kivételdeklaráció;
- megadott típusú új nevet (névkonstruktort) létrehozó értékdeklaráció.

## Szignatúra és struktúra (folyt.)

---

### A szignatúra-deklaráció

- `signature sigid = sigexp` alakú, ahol
  - `sigid` egy szignatúranév,
  - `sigexp` pedig egy szignatúrakifejezés.
- A szignatúranév a szignatúrakifejezés rövidítése, szinonimája.

### A struktúra-deklaráció egyszerű változata

- `structure strid = strexp` alakú, ahol
  - `strid` egy struktúranév,
  - `strex` pedig egy struktúrakifejezés.
- A struktúranév a struktúrakifejezés rövidítése, szinonimája.

### A struktúra-deklaráció bonyolultabb változata

- struktúra szignatúrához kötése; amely kétféle lehet:
  - áttetsző (opál, opaque): `structure strid :> sigid = strexp`
  - átlátszó (transzparens, transparent): `structure strid : sigid = strexp`

## Példa: a KCsiga és a Csiga szignatúra

---

- A KCsiga struktúra (a keretprogram) szignatúrája

```
signature KCsiga =
sig
  val csigaBe : string -> TCsiga.feladvanyleiro
  val csigaKi : string * TCsiga.csigatabla list -> unit
  val megold : string * string -> string
end
```

- A Csiga struktúra (a főmodul) szignatúrája

```
signature Csiga =
sig
  val buvosCsiga :
    TCsiga.feladvanyleiro -> TCsiga.csigatabla list
end
```

Mindkét szignatúra csupán `val id : typ` alakú *értékspecifikációt* tartalmaz.

## Példa: a TCsiga struktúra és törzsszignatúrája

- A TCsiga struktúra (a típusleíró modul) és *törzsszignatúrája*

```

structure TCsiga =
struct
  type meret          = int
  type ciklus         = int
  type sorszam        = int
  type oszlopszam     = int
  type ertek          = int
  type adottElem      =
    sorszam * oszlopszam * ertek
end

(* TCsiga.sml
   törzsszignatúrája *)
type meret          = int
type ciklus         = int
type sorszam        = int
type oszlopszam     = int
type ertek          = int
type adottElem      =
  int * int * int

type feladvanyleiro =
  meret * ciklus * adottElem list

type ertekVagyUres  = int
type sor            =
  ertekVagyUres list

type csigatabla     =
  sor list

type feladvanyleiro =
  int * int * (int * int * int) list

type ertekVagyUres  = int
type sor            =
  int list

type csigatabla     =
  int list list

```

## Példa: a TCsiga struktúra és törzsszignatúrája (folyt.)

- *Törzsszignatúra* (principal signature): egy struktúra összetevőinek legspecifikusabb leírása.
- TCsiga csupa típusspecifikációt tartalmaz
 

```

type (tyvar1, ..., tyvarn) tycon [ = typ ],

```

 pontosabban
 

```

type tycon = typ

```

 alakban.
- Egy struktúra szignatúra-megkötés nélkül nem rejti el a részleteket, azaz *gyenge absztrakciót* valósít meg.

## Példa: változatok a TCsiga struktúrára és szignatúrára

---

### • Struktúra *átlátszó* szignatúrával

```
structure TCsiga : TCsiga =
struct
  type meret          = int
  ...
  type adottElem     =
    sorszam * oszlopszam * ertekek
  type feladvanyleiro =
    meret * ciklus * adottElem list
  type ertekekVagyUres = int
  type sor           =
    ertekekVagyUres list
  type csigatabla     = sor list
end
```

### • Struktúra *áttetsző* szignatúrával

```
structure TCsiga :> TCsiga =
struct
  type meret          = int
  ...
  type adottElem     =
    sorszam * oszlopszam * ertekek
  type feladvanyleiro =
    meret * ciklus * adottElem list
  type ertekekVagyUres = int
  type sor           =
    ertekekVagyUres list
  type csigatabla     = sor list
end
```

### • Szignatúra (a részleteket *elrejtő*) absztrakt adattípus megvalósításához

```
signature TCsiga =
sig
  type feladvanyleiro
  type csigatabla
end
```