

# ESETSZÉTVÁLASZTÁS, OPCIONÁLIS ÉRTÉK

## Esetszétválasztás (case)

```
case E of P1 => E1 | P2 => E2 | ... | Pn => En
```

Az SML-értelmező – balról jobbra és fölülről lefelé haladva – megpróbálja E-t P1-re illeszteni, ha nem sikerül, P2-re s.í.t. A case-kifejezés eredménye az E kifejezésre illeszkedő első P<sub>i</sub> mintához tartozó E<sub>i</sub> kifejezés lesz.

A case is csak szintaktikus édesítőszerszer, ui. helyettesíthető fn-jelöléssel:

```
(fn P1 => E1 | P2 => E2 | ... | Pn => En) E
```

Például a lady függvényt így is definiálhattuk volna:

```
datatype degree = Duke | Marquis | Earl | Viscount | Baron
(* lady : degree -> string
   lady p = p főnemes
           hitvesének rangja *)
fun lady p =
  case p of Duke      => "Duchess "
          | Marquis   => "Marchioness"
          | Earl      => "Countess"
          | Viscount  => "Viscountess"
          | Baron     => "Baroness"

(* lady : degree -> string
   lady p = p főnemes
           hitvesének rangja *)
fun lady p =
  (fn Duke      => "Duchess "
   | Marquis   => "Marchioness"
   | Earl      => "Countess"
   | Viscount  => "Viscountess"
   | Baron     => "Baroness") p
```

## Opcionális érték kezelése ('a option)

---

```
datatype 'a option = NONE | SOME of 'a
```

Függvények az Option könyvtárból:

```
val getOpt      : 'a option * 'a -> 'a
val isSome     : 'a option -> bool
val valOf      : 'a option -> 'a
val filter     : ('a -> bool) -> 'a -> 'a option
val map        : ('a -> 'b) -> 'a option -> 'b option
val mapPartial : ('a -> 'b option) -> ('a option -> 'b option)
```

```
getOpt (xopt, d) = x if xopt is SOME x, d otherwise.
```

```
isSome xopt = true if xopt is SOME x, false otherwise.
```

```
valOf xopt = x if xopt is SOME x, raises Option otherwise.
```

```
filter p x = SOME x if p x is true, NONE otherwise.
```

```
map f xopt = SOME(f x) if xopt is SOME x, NONE otherwise.
```

```
mapPartial f xopt = f x if xopt is SOME x, NONE otherwise.
```

## Példák opcionális értékek kezelésére

---

- Egészlista legnagyobb elemének kiválasztása

Üres listának nincs legnagyobb eleme; egyelemű lista egyetlen eleme a „legnagyobb”; legalább kételemű lista legnagyobb eleme az első elem és a maradéklista elemei közül a legnagyobb.

```
(* maxl : int list -> int option
   maxl ns = az ns egészlista legnagyobb eleme *)
fun maxl []      = NONE      (* üres *)
  | maxl [n]    = SOME n    (* egyelemű *)
  | maxl (n::ns) =          (* legalább kételemű *)
    SOME(Int.max(n, valOf(maxl ns)))
```

- Füzér elején álló karaktersorozat átalakítása egész számmá

```
val Int.fromString : string -> int option (* Overflow *)
```

```
Int.fromString s = SOME i if a decimal integer numeral can be scanned
  from a prefix of string s, ignoring any initial whitespace;
  NONE otherwise. A decimal integer numeral, after any initial
  whitespace, must have the form: [+~-]?[0-9]+
```

```
Int.fromString "1234"; Int.fromString "-1234"; Int.fromString "~1234";
Int.fromString "+1234"; Int.fromString "+007"; Int.fromString "alma"
```

# KIVÉTELKEZELÉS

## Kivételkezelés

- Kivételt az `exception` kulcsszóval deklarálunk, a `raise` kulcsszóval jelzünk, a `handle` kulcsszóval bevezetett kifejezésben kezelünk.
- A kivételeket leggyakrabban hibák jelzésére használjuk.
- A kivételkonstruktor lehet állandó (pl. `Domain`) vagy függvény (pl. `Fail`).
- A kivételkonstruktorállandónak, ill. a kivételkonstruktorfüggvény eredményének a típusa: `exn`.
- Az `exn` speciális típus:
  - ◇ a kivételkonstruktorok halmaza *bővíthető*,
  - ◇ az `exn` típust tartalmazó ún. *kivételcsomag* minden típussal kompatibilis:

```
- fun // {den = 0, ...} = raise Domain
  | // {num = n, den = d} = (real n) / (real d);
> val // = fn : {den : int, num : int} -> real
```

pedig

```
- Domain;
> val it = Domain : exn
```

## Kivételkezelés (folyt.)

- A `raise` kulcsszó olyan ún. *kivételcsomagot* hoz létre, amelyben `exn` típusú érték is van.
- A kivétel kezelése a `case`-szerkezetre emlékeztet: `E handle P1 => E1 | ... | Pn => En`
- Ha `E` „közönséges” értéket ad eredményül, a kivételkezelő egyszerűen továbbadja az eredményt.
- Ha `E` eredménye *kivételcsomag*, az SML megpróbálja illeszteni a `P1`, ..., `Pn` mintákra.
  - ◊ Ha `Pi` ( $1 \leq i \leq n$ ) az első illeszkedő minta, akkor `Ei` a kivételkezelő eredménye.
  - ◊ Ha egyetlen minta sem illeszkedik a kivételcsomagra, a kivételkezelő továbbpasszolja.
- Példa visszalépés programozására kivételkezeléssel

```
exception Valtas;
(* valtas : int list -> int -> int list
   valtas ermelista osszeg = a lehető legkevesebb érmét tartalmazó olyan
   érmelista, amely elemeinek összege osszeg
   PRE : osszeg >= 0 *)
fun valtas _ 0 = []
  | valtas [] _ = raise Valtas
  | valtas (erme::ermelista) osszeg =
    if erme > osszeg then valtas ermelista osszeg
    else erme :: valtas (erme::ermelista) (osszeg-erme)
      handle Valtas => valtas ermelista osszeg
```

## Kivételkezelés (folyt.)

- A leggyakoribb belső kivételek

Megnevezés	Művelet, amely a kivételt kiválthatja
Bind	Értékdeklarációban a jobb oldali kifejezés nem illeszkedik a bal oldali mintára.
Chr	<code>chr pred succ</code>
Div	<code>/ div mod</code>
Domain	Az érték kilóg az értelmezési tartományból.
Empty	<code>hd tl last</code>
Fail	<code>compile load loadOne</code>
Interrupt	Megszakítás <code>ctrl/c</code> -vel.
Io	Ki/beviteli hiba. <code>Io of {function : string, name : string, cause : exn}</code>
Match	Mintaillesztési hiba <code>case</code> és <code>handle</code> kifejezésben, vagy függvényalkalmazásban.
Option	Hiba egy <code>Option</code> könyvtárbeli függvény alkalmazásakor.
Overflow	<code>~ + - * / div mod abs ceil floor round trunc</code>
Size	<code>^ array concat fromList implode tabulate translate vector</code>
Subscript	<code>copy drop extract nth sub substring take update</code>

# LISTÁK RENDEZÉSE

---

Listák rendezése 7-10

## Listák rendezése

---

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

## Beszűrő rendezés

---

- Az `ins` segédfüggvény az `x` elemet a megfelelő helyre rakja be az `ys` listában:

```
(* ins : real * real list -> real list
   ins (x, ys) = ys kibővítve x-szel a <= reláció szerint
   PRE: ys a <= reláció szerint rendezve van *)
fun ins (x, y::ys) = if x <= y then x::y::ys else y::ins(x, ys)
| ins (x : real, []) = [x]
```

- `inssort`-tal rekurzívan rendezzük a lista maradékát; végrehajtási ideje  $O(n^2)$ :

```
(* inssort : ('a * 'b list -> 'b list) -> 'a list -> 'b list
   inssort f xs = az xs elemeiből álló, az f felhasználásával
                 rendezett lista *)
fun inssort f (x::xs) = f(x, inssort f xs)
| inssort _ [] = []
```

- Példa `inssort` alkalmazására:

```
inssort ins [4.24, 4.1, 5.67, 1.12, 4.1, 0.33, 8.0]
```

## Beszűrő rendezés, generikus változat

---

- Az `ins` függvényt generikussá tesszük:

```
(* ins : ('a * 'a -> bool) -> 'a * 'a list -> 'a list
   ins cmp (x, ys) = ys kibővítve x-szel a cmp reláció szerint
   PRE: ys a cmp reláció szerint rendezve van *)
fun ins cmp (x, ys) =
  let fun ins0 (y::ys) =
        if cmp(x, y) then x::y::ys else y::ins0 ys
      | ins0 [] = [x]
  in ins0 ys
  end
```

- Ezzel `inssort` egy újabb változata:

```
(* inssort : ('a * 'a -> bool) -> 'a list -> 'a list
   inssort cmp xs = az xs elemeiből álló, a cmp reláció
                   szerint rendezett lista *)
fun inssort cmp (x::xs) = ins cmp (x, inssort cmp xs)
| inssort _ [] = []
```

## Beszűrő rendezés, generikus változat (folyt.)

- `insort` eddigi változatai előbb elemeire szedik szét a rendezendő listát, majd hátulról visszafelé haladva, rendezés közben építik fel az újat.
- A jobbrekurziót és akkumulátort használó változatnak (`insort2`) kisebb veremre van szüksége, mivel a listáról leválasztott elemeket balról jobbra haladva azonnal berakja a helyükre az eredménylistában. (A két megoldás futási idejét később összehasonlítjuk).

```
(* insort2 : ('a * 'a -> bool) -> 'a list -> 'a list
   insort2 cmp xs = az xs elemeiből álló, a cmp reláció
                   szerint rendezett lista *)

fun insort2 cmp xs =
  let (* sort : 'a list -> 'a list -> 'a list
       sort xs zs = zs kibővítve az xs-nek a cmp reláció
                   szerint rendezett elemeivel
       PRE: zs cmp szerint rendezve van *)
      fun sort (x::xs) zs = sort xs (ins cmp (x, zs))
        | sort [] zs = zs
  in
    sort xs []
  end
```

## Beszűrő rendezés `foldr`-rel és `foldl`-lel

- A második argumentumát akkumulátorként használó `foldl` kisebb vermet használ `foldr`-nél, ezért `insortL` hosszabb listákat tud rendezni:

```
fun insortR cmp = foldr (ins cmp) []
fun insortL cmp = foldl (ins cmp) []
```

- Példák `insort`-tal és `insort2`-vel:

```
insort op<= [4.24, 4.1, 5.67, 1.12, 4.1, 0.33, 8.0];
insort2 op>= [4, 4, 5, 1, 0, 8];
insort op< (explode "qwerty")
```

- Példák `foldr` és `foldl` felhasználásával:

```
fun insortRi cmp = foldr (ins cmp) [];
fun insortLr cmp = foldl (ins cmp) ([] : real list)

insortRi op>= [4, 4, 5, 1, 0, 8];
insortLr op>= [4.24, 4.1, 5.67, 1.12, 4.1, 0.33, 8.0]
```