LaTeX: Makrók és programozás

Vadon Viktória

2023/24/I. félév

Makrók I

- a LaTeX (és TeX) legtöbb parancsa *makró*
  - ez mit jelent?
  - definíció `\parancs` = helyettesítendő szöveg formában
  - kifejtés: ahol `\parancs`-csal találkozik egy fájlban, a definíció alapján helyettesíti
  - több szintű kifejtés: ha a helyettesítésben talál újabb makrót, azt is helyettesíti, és így tovább, amíg csak alap parancsok (TeX primitívek) és megjeleníthető objektumok (betűk, képek, stb.) maradnak
  - makró lehet végrehajtható parancs és változónév egyaránt – mindkettő `\parancs` formában néz ki
  - a legtöbb beépített parancs, amit használunk (+bármi, ami csomagokból jön) már egy magasabb szintű makró – kényelmi funkció a primitívekhez képest
  - TeX programozása: létrehozhatók felhasználói makrók/parancsok

Makrók definiálása – argumentum nélkül I

- legegyszerűbb formában, (argumentum nélküli) új makró: `\newcommand{\parancs}{kifejtés}`
  - `\parancs`: \-szel kezdődik, angol abécé betűi, érzékeny kis- és nagybetűkre
  - ilyesmi mindig a preambulumba kerüljön!
  - ha már létezik `\parancs` néven makró, hibát dob
- példák:
- `\newcommand{\prob}{\mathbb{P}}`
  - `$\prob$` $\mathbb{P}$
- `\newcommand{\hublindtext}{Ennek a mondatnak semmi mondanivalója nincs, csak helykitöltő szövegnek fogjuk használni. }`
  - `\hublindtext\hublindtext` Ennek a mondatnak semmi mondanivalója nincs, csak helykitöltő szövegnek fogjuk használni. Ennek a mondatnak semmi mondanivalója nincs, csak helykitöltő szövegnek fogjuk használni.

Makrók definiálása – kötelező argumentummal I

- `\newcommand{\parancs}[argszám]{kifejtés}`
- argszám = argumentumok száma, max 9
- a kifejtésben #1, #2 stb., #9 néven lehet beilleszteni az argumentumokat
- híváskor `\parancs{}...{}`
  - argszám-nak megfelelő darab `{ }`
- `\newcommand{\probof}[1]{\prob(#1)}`

  `$\probof{A}$`

  $\probof{A}$
- `\newcommand{\pcond}[2]{\prob(#1\mid#2)}`

  `$\pcond{A}{B}$`

  $\pcond{A}{B}$

Makrók definiálása – opcionális argumentummal I

- `\newcommand{\parancs}[argszám][alapért]{kifejtés}`
- ahogy eddig: max 9 argumentum, a kifejtésben #1, #2 stb. néven
- *csak #1 tehető opcionálissá*(!)
- alapért = #1 alapértelmezett értéke
  - ha híváskor kimarad az opcionális argumentum, az alapértelmezést használja
  - hívható üres opcionális argumentummal, akkor üres sztringet helyettesít
  - alapértelmezést kötelező megadni (de lehet üres) – innen tudja, hogy #1 opcionális!
- híváskor `\parancs{#2}...`, vagy `\parancs[#1]{#2}...`

Makrók definiálása – opcionális argumentummal II

- példa: 2×2-es identitás-mátrix x-szerese – alapértelmezésben 1-szeres:

```
\newcommand{\idmx}[1][1]{\begin{pmatrix}%
#1&0\\0&#1\end{pmatrix}}
```

  - `$\idmx$` $\begin{pmatrix}1&0\\0&1\end{pmatrix}$
  - `$\idmx[x]$` $\begin{pmatrix}x&0\\0&x\end{pmatrix}$
- logaritmus, alapértelmezésben 2-es alappal

```
\newcommand{\logof}[2][2]{\log_{#1}(#2)}
```

  - `$\logof{x}$` $\log_2(x)$
  - `$\logof[16]{x}$` $\log_{16}(x)$

Makrók definiálása – kiterjesztések I

- az összesre ugyanaz a szintaxis, mint a `\newcommand`-ra
  - argumentumokkal is
- `\renewcommand{\parancs}{új kifejtés}`
  - *létező* makró *felül*definiálása
  - akkor dob hibát, ha `\parancs` még nem létezik
- `\providecommand{\parancs}{kifejtés}`
  - *csak akkor* definiálja `\parancs`-ot, ha még nem létezik ezen a néven makró
  - ha `\parancs` már létező makró, nem csinál semmit
  - egyik esetben sem dob hibát(!)
- `\newcommand*{\parancs}{új kifejtés}`
  - a létrehozott `\parancs` argumentumában/argumentumaiban nem enged új bekezdést használni

Makrók definiálása – kiterjesztések II

- saját makrók *-os verzióval
- nem triviális, mert a * nem betű
- legegyszerűbben `suffix` csomaggal
- `\newcommand{\parancs}{kifejtés}`
  `\WithSuffix\newcommand\parancs*{kifejtés}`
  - a `\WithSuffix`-es sorban ne legyen kapcsos zárójelben a `\parancs*`!
  - argumentumok számát (+alapértelmezést) `\parancs*` után szokásos módon szögletes zárójelben `[ ]` adhatunk meg
  - a parancsnak és *-os verziójának nem feltétlen azonosak az argumentumaik!

Makrók definiálása – kiterjesztések III

- például

```
\newcommand{\hw}{Hello world}
\WithSuffix\newcommand\hw*[1][world]%
{\MakeUppercase{hello~#1}}
```

  - \hw Hello world
  - \hw* HELLO WORLD
  - \hw*[me] HELLO ME

Törékeny parancsok I

- mi az a törékeny parancs?
  - parancsok egy kategóriája – parancshoz kötött tulajdonság
  - mitől törékeny? hibát produkálhat, ha más parancs argumentumába kerül
  - példák:
  - pl. verbatim, dobozok, egyes matematikai parancsok
  - parancsok opcionális argumentummal
  - parancsok ún. *mozgó* argumentummal
    - *mozgó* argumentum: ami külső fájlba kerül feldolgozáskor
    - pl. section és hasonló címsorok (tartalomjegyzék generálásakor)
    - pl. `\caption` tartalma (float-ok listázásakor)
- törékeny parancsok „védelme”: `\protect` kulcsszó eléjük
- törékeny parancsok használata definícióban: `\DeclareRobustCommand{\parancs}{kifejtés}`

Környezetek I

- saját parancsok `\begin{valami}`, `\end{valami}` formában
- argumentum nélkül

  `\newenvironment{név}{nyitó kód}{záró kód}`

  `név`: tetszőleges (akár ékezetes betűk és számok is), nem kell bele `\` (!)
- ezek után használható `\begin{név}` és `\end{név}` parancspár
- `\begin{név}` helyére a nyitó kódot, `\end{név}` helyére a záró kódot helyettesíti

Környezetek II

- például

```
\newenvironment{vonalzott}%
{\vspace{1ex}\hrule\vspace{1ex}}%
{\vspace{1ex}\hrule\vspace{1ex}}
```

- használat:

```
előtte
\begin{vonalzott}
Hello world
\end{vonalzott}
utána
```

előtte

Hello world

---

utána

Környezetek III

- variánsok
  - `\renewenvironment`: létező környezet felüldefiniálása
  - `\newenvironment*`: a környezet argumentumaiban (ld. következő dia) nem enged új bekezdést
    - a törzsre, azaz a `\begin{név}` és `\end{név}` közti tartalomra nem vonatkozik!
  - `\renewenvironment*`: újradefiniálás, és argumentumokban nem enged új bekezdést
- környezet *-os verziója:
  - `név` és `név*` néven egymástól teljesen *függetlenül* definiálható(k) környezet(ek)

Környezetek argumentumai I

- saját parancshoz hasonló szintaxis:

  `\newenvironment{név}[argszám][alapért]%`
  `{nyitókód}{zárókód}`

  - [alapért] csak akkor szükséges, ha #1 opcionális
- az argumentumok csak a nyitókódban használhatók!
  - trükközni kell és makróként elmenteni, ha a zárókódban szeretnénk használni

Környezetek argumentumai II

- például korábbi vonalzott környezet „címsorral”:

```
\newenvironment{vonalzott*}[1]%
{\vspace{1ex}\hrule\vspace{1ex}%
\begin{center}#1\end{center}}%
{\vspace{1ex}\hrule\vspace{1ex}}
```

-
```
\begin{vonalzott*}{cím}
Hello world
\end{vonalzott*}
```

---

cím

Hello world

Környezetek argumentumai III

- példa argumentum mentésére: idézet szerzővel

```
\newenvironment{quotewithauthor}[1]%
{\newcommand{\quoteauthor}{#1}\begin{quotation}}
{\par\hfill--\quoteauthor\end{quotation}}
```

- ```
  \begin{quotewithauthor}{Konfuciusz}
  Tudni, hogy amit tudunk, azt tudjuk, ám amit nem
  tudunk, azt nem tudjuk; ez az igazi tudás.
  \end{quotewithauthor}
  ```

  Tudni, hogy amit tudunk, azt tudjuk, ám amit nem tudunk, azt nem tudjuk; ez az igazi tudás.

  —Konfuciusz
  ```

Környezetek – jó tudni I

- néha felesleges térköz vagy whitespace jelenik meg a környezet előtt vagy után
  - kiiktatás `\ignorespaces`, illetve a végén `\ignorespacesafterend` paranccsal
- bizonyos parancsokat nem lehet szétszakítani a nyitó-és zárókód között!
  - pl. ha `\fbox`-ba szeretnénk zárni a környezetet, akkor `\fbox{` a nyitó kódba kéne kerüljön, zárása `}` pedig a zárókódba – ami hibát okoz
  - sokszor egyszerű megoldás: parancs használata környezet helyett
  - vagy: környezet tartalmának mentése *doboz regiszterbe* (ld. 1. szakasz)
  - ha egy csoport nyitása és zárása kétfelé szakad, érdemes lehet `\begingroup`, `\endgroup` paranccsal helyettesíteni a zárójeleket

Helyi parancsok I

- mint más nyelvekben a helyi változók, LaTeX-ben csoporton, parancson vagy környezeten belül használhatók helyi makrók
  - létező makró felüldefiniálása is lehetséges helyileg
- például \emph felüldefiniálása csoportban:

```
{\renewcommand{\emph}[1]{\textbf{#1}}
\emph{blah}} \emph{blah}
```

blah *blah*

Helyi parancsok II

- parancson belül
  - a kifejtést csomagoljuk extra pár kapcsos zárójelbe/csoportba(!)
    - ugyanis kifejtéskor nem zár automatikusan csoportba, így a „helyi” parancsunk nem lenne többé helyi
  - a (felül)definíció legyen az első parancs
  - belső parancs argumentumai: ##1, stb.
    - külső parancs argumentumai #1, stb.
  - például

```
\renewcommand{\textit}[1]%
{{\renewcommand{\emph}[1]{\textbf{##1}}%
{\itshape #1}}}
```

  - `\textit{italic \emph{emph}} utána`
  - *italic* ***emph*** utána

Helyi parancsok III

- környezeten belül
  - (újra)definíció a nyitókódba
  - nincs szükség extra csoportra, a környezetet kifejtéskor csoportba zárja
  - ```
    \newenvironment{italic}{%
    \renewcommand{\emph}[1]{\textbf{##1}}%
    \itshape}{}
    ```
  - ```
    \begin{italic}
    italic \emph{emph}
    \end{italic}
    utána
    ```
  - *italic* ***emph*** utána

2 Változók

- Típusok
- Számlálók
- Dobozok
  - Doboz változó
- Hosszok

Változók típusai LaTeX-ben I

- már ismert: makrók, parancsok
- számláló (counter) – egész (általában természetes) szám
- doboz (box)
  - tartalma tetszőleges
  - különbség egy doboz és egy `\newcommand`-os makró között, hogy a parancsot mindig újra fel kell dolgoznia, míg a dobozt „kiszedve”, elrendezésével együtt menti – úgy illeszti be, mint egy pecsétnyomó
  - ha valamit sokszor, változatlan formában használunk, pl. cégnév és logó a fejlécben, érdemes dobozt használni parancs helyett
- hossz (length) – (valós) szám, mértékegységgel
  - akár plusz-mínusz tűréssel együtt

Számlálók I

- számláló: egész szám
- beépítve ezt használja számozásra – oldalszámozás, szakaszolási egységek, számozott listák, float-ok, tételkörnyezetek, stb.
- számláló definiálása: `\newcounter{név}`
  - figyeljük meg: `név` nem tartalmaz \-t(!)
  - tetszőleges karaktersorozat
  - hibát dob, ha az adott néven már létezik számláló
  - 0-val inicializál

Számlálók II

- számláló kiíratása
  - alapértelmezett számtípussal: `\thenév`
  - adott formázással
    - arab számok `\arabic{név}`
    - kis- és nagybetűs római számok `\roman{név}`, `\Roman{név}`
    - kis-és nagybetűk `\alph{név}`, `\Alph{név}`
    - szimbólumok (csak 1-9-ig): `\fnsymbol{név}`
- érték lekérdezése belső változóként való használatra `\value{név}`
- számláló manipulálása
  - léptetés `\stepcounter{név}`, `\refstepcounter{név}`
    - egyet hozzáad és visszaállítja az összes számlálót, aminek `név` *számlálóőse*
    - `\refstepcounter` beállítja a `\label` hivatkozási sorszámát is
  - hozzáadás (vagy kivonás): `\addtocounter{név}{-3}`
  - adott értékre beállítás: `\setcounter{név}{0}`

Számlálók III

- *számlálóős*
  - pl. section számlálója *számlálóőse* subsection számlálójának: valahányszor section számlálója *lép egyet*, subsection számlálója *újraindul*
  - számlálóős megadása: `\newcounter{név}[számlálóős]`
  - számlálóős felvétele utólag: `\counterwithin{név}{számlálóős}`
  - számlálóős eltávolítása utólag: `\counterwithout{név}{számlálóős}`

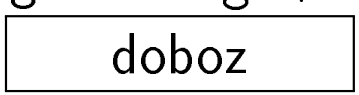
Számlálók IV

- jó tudni: `\setcounter`, `\addtocounter` parancsokban csak egy szám használható, vagy `\value{másikszámláló}`
  - kiszámítandó kifejezés használatához `\numexpr`, például

```
\newcounter{countegy}
\newcounter{countketto}
\setcounter{countegy}{5}
\setcounter{countketto}%
{\numexpr 2*\value{countegy}}
countegy: \thecountegy, countketto: \thecountketto
```

    countegy: 5, countketto: 10
  - hasonlóképp `\dimexpr` és `glueexpr` számolandó hosszúságra és nyújtható ragasztóra
  - ha nincs argumentumba zárva, vagy mértékegység utána, `\relax` paranccsal zárható a kifejezés

Dobozok I

- környezetek definíciójában hasznosak lehetnek
- float-hoz hasonlóan a dobozokat nem tördeli oldalak között a LaTeX (tulajdonképpen a float-ok is dobozokon alapulnak)
- két nagyobb típus aszerint, hogy a *dobozon belül* sorokra tördeljük-e a szöveget
- tördeletlen dobozok: `\mbox{szöveg}`, `\makebox{szöveg}`
  - *mire jó?* pl. elválasztás megakadályozása helyileg `\mbox{A-21}`
- csak a `\makebox` verziónak vannak opcionális argumentumai:
  - `\makebox[szélesség][igazítás]{szöveg}`
  - tartalom elrendezése *adott szélességen*, de *egy sorban*
  - gyakran: 0 szélességű dobozzal tartalom belógatása a margóra
- argumentumok használata:
  - szélesség: `szöveg \fbox{\makebox[2cm]{doboz}} szöveg`
  - szöveg  szöveg

Dobozok II

- szélesség + igazítás:
  `\makebox[szélesség][igazítás]{tartalom}`
  - igazítás második opcionális argumentum – csak szélességgel együtt használható!
  - igazítás lehet:
  - c: középre – alapértelmezett
  - l: balra
  - r: jobbra
  - s: nyújtás – csak ha nyújtható hosszt tartalmaz a doboz (mint `\hfill`)
- például címke a bal *margóba*: sor elejére 0 széles doboz, tartalom jobb széle igazodik hozzá

```
\makebox[0pt][r]{(címke)} A címke a bal margóba lóg bele, ide pedig jön a tartalom...
```

(címke) A címke a bal margóba lóg bele, ide pedig jön a tartalom...

Dobozok III

- dobozok sortördelt tartalommal
- egy bekezdés engedélyezett:
  `\parbox{szélesség}{tartalom}`
- több bekezdés engedélyezett:
  `\begin{minipage}{szélesség}`, `\end{minipage}` környezet
- !! mindkettőhöz kötelező argumentum a szélesség
  - tetszőleges LaTeX hossz, pl. `5cm`, vagy `0.8\linewidth`
- opcionális argumentumok:

```
\parbox[igazítás][magasság][belső igazítás]%
{szélesség}{tartalom}
\begin{minipage}[igazítás][magasság]%
[belső igazítás]{szélesség}
tartalom
\end{minipage}
```

Dobozok IV

- *igazítás*: a környező szöveg alapvonalához a parbox/minipage melyik része igazodik:
- `c`: parbox/minipage közepe (alapért.), `t`: legfelső alapvonal, `b`: legalsó alapvonal
- akkor van jelentősége, ha nem tölti ki a teljes sorszélességet, és szöveggel egy sorba kerül
- *magasság*: tetszőleges LaTeX hosszban – alapért: automatikusan méretez a tartalomhoz
- *belső igazítás*: tartalom igazítása a parbox-on/minipage-en belül
- `c` középre (alapért.), `t` felülre, `b` alulra, `s`: nyújtás* (ha tartalmaz nyújtható térközt, mint `\vfill`)

- korábban látott `\fbox`, `\colorbox`, stb. a `\makebox`-on alapulnak, ezért nem tördeli a tartalmukat
  - megoldás: tegyünk a hasába `\parbox`-ot vagy `minipage` környezetet

Doboz változó I

- doboz változó: elrendezéssel együtt mentett tartalom, amit „pecsétnyomóként” illeszt be
- deklarálás: `\newsavebox{\nev}`
  - neve `\nev`: formailag makró(!)
  - `\`-szel kezdődik, csak angol ábécé betűi
  - nem egyezhet meg más, létező makróval
- mentés dobozba
  - mentés előtt a `\nev` nevű dobozt deklarálni kell!
  - `\sbox{\nev}{tartalom}`, `\savebox{\nev}{tartalom}`
  - szintaxis, opcionális argumentumok megegyeznek `\makebox`-szal:
    - `\savebox{\nev}[szélesség]{tartalom}`
    - `\savebox{\nev}[szélesség][igazítás]{tartalom}`
- mentett doboz beillesztése: `\usebox{\nev}`

Doboz változó II

- `\savebox` környezet verziója: `lrbox`
- ```
  \begin{lrbox}{\nev}
  tartalom
  \end{lrbox}
  ```
- ez a környezet verzió használható saját környezet definíciójában is!
- például keretezett környezet definiálása

```
\newsavebox{\dobozom}
\newenvironment{keretes}%
{\begin{lrbox}{\dobozom}%
\begin{minipage}{\linewidth}}%
{\end{minipage}\end{lrbox}%
\fbox{\usebox{\dobozom}}}
```

  - a `minipage` csak a tördeléshez lett beszúrva
```

Hosszok I

- hossz: mértékegységgel ellátott (valós) szám, pl. 20pt
- ragasztó: hossz, plusz és/vagy mínusz tűréssel, pl. 20pt plus 3pt minus 2pt – de keverni is lehet a mértékegységeket
- hossz/ragasztó típusú változó deklarálása: `\newlength{\hossz}`
  - neve `\hossz` formailag makró: `\`-szel kezdődik, csak angol ábécé betűit használhatja
  - szabad makrónak kell lennie
  - bármilyen értékbeállítás előtt deklarálni kell a változót!
- beállítás adott értékre: `\setlength{\hossz}{érték}`, ahol az érték lehet fix, de tartalmazhat plusz-mínusz tűrést is
- hossz növelése (vagy csökkentése): `\addtolength{\hossz}{hozzáadandó}`
  - akár mindkettő tartalmazhat tűrést(!)
  - nem feltétlen kell azonos mértékegységben lenniük

Hosszok II

- tartalom méretének lekérdezése/mentése hossz változóba
  - `\settowidth{\hossz}{tartalom}`: `\hossz` új értéke `tartalom` szélessége
  - hasonlóképp: `\settoheight{\hossz}{tartalom}` magasság – *alapvonal fölötti* magasság(!)
  - `\settodepth{\hossz}{tartalom}` mélység – mennyivel nyúlik az alapvonal alá

3. Feltételvizsgálat, ciklus
   - Feltételvizsgálat, elágazás
   - Ciklus

Feltételvizsgálat, elágazás I

- `ifthen` csomaggal – dokumentáció (link)
- feltétel és elágazás:

  `\ifthenelse{feltétel}{ha igaz}{ha hamis}`

feltételben használható:

- szám $<$ szám, szám $=$ szám, szám $>$ szám
  - szám lehet pl. 10, vagy `\value{számláló}`
- `\isodd{szám}`: igaz, ha páratlan
  - hamis akkor is, ha az argumentum nem szám(!)
- `\isundefined{\parancs}`: igaz, ha a parancs nem definiált
- `\lengthtest{hossz < hossz}`, vagy $>$, vagy $=$
  - LaTeX hosszak összehasonlítása
  - itt is lehet pl. 3cm, 2in, stb., vagy hossz változó
  - nem kell megegyeznie a mértékegységeknek

Feltételvizsgálat, elágazás II

- `\equal{token}{token}`: „egyenlők”; ha a tokenek parancsok, akkor kifejtés után egyenlők-e
  - „token” jelenthet parancsot vagy egyszerűen sztringet
- `\boolean{név}`: igaz-hamis változó
  - beépített boolean változók
    - `mmode`: matematikai mód
    - `inner`: belső mód
    - `true`, `false`: konstans igaz és hamis
  - saját boolean változók
    - deklarálás: `\newboolean{név}`
    - deklarálás csak akkor, ha még nem létezik: `\provideboolean{név}`
    - beállítás: `\setboolean{név}{érték}`, ahol érték `true` igaz vagy `false` hamis

Feltételvizsgálat, elágazás III

- feltételek összekapcsolása
  - `\NOT`, `\not` – tagadás (egyváltozós)
  - `\AND`, `\and` – és
  - `\OR`, `\or` – vagy
  - zárójelezés: `\(`, `\)`
  - ÉS és VAGY közt nincs precendencia, balról jobbra dolgozza fel – ha kell, zárójelezzünk!
- például
  `\ifthenelse{\isodd{\value{page}}}{páratlan}{páros}` oldal
- páratlan oldal
- oldalszám: `\thepage` 37

Ciklus I

- `ifthen` csomaggal while ciklus:
  - `\whiledo{feltétel}{ciklusmag}`
  - feltétel azonos szintaxissal, mint az `\ifthenelse`
  - ciklusváltozót magunknak kell inicializálni a ciklus előtt és növelni a ciklusmagban!
  - például

    ```
    \newcounter{y}
    \whiledo{\value{y}<10}{\stepcounter{y}\roman{y}~}
    ```

  - i ii iii iv v vi vii viii ix x

Ciklus II

- pgffor csomaggal (tikz rajzoló csomag része) foreach ciklus
  - rendhagyó szintaxis!
  - `\foreach \változó in { vesszővel, elválasztott, % lista } { ciklusmag }`
  - változópárokkal:

    `\foreach \váltegy/\váltkettő in { érték1/érték2, % közös_érték, egy/kettő } { ciklusmag }`

    - akár változóhármassal, stb.
    - ha kevesebb az érték, mint a változó, az utolsó (szereplő) értéket ismétli
  - például

    ```
    \foreach \x/\y in {a/b, c/d, e} %
    { \x \dotfill \y \hfill }
    ```

  - a . . . . . b c . . . . . d e . . . . . e

Ciklus III

- a lista elmenthető makróba
  - fenti példával ekvivalens:

    ```
    \newcommand{\listam}{a/b, c/d, e}
    \foreach \x/\y in \listam {\x \dotfill \y \hfill}
    ```
- egyetlen változó esetén számtani sorozatot is tud:
  `\foreach \x in {1,2,...,10} {\x~}` 1 2 3 4 5 6 7 8 9 10
  - a ~ csak a törhetetlen szóköz
  - kell: kezdőérték, második érték (ebből számolja a növekményt), majd ..., és „végérték”
  - a végértéket nem veszi fel, ha nincs a sorozatban, de nem megy rajta túl
  - negatív számokat és negatív növekményeket is tud kezelni!
  - `\foreach \x in {9,7,...,-10} {\x~}`
    9 7 5 3 1 -1 -3 -5 -7 -9
- megjegyzés: `tikz` rajzoló csomag – `\foreach` ismétlődő rajzolásra is használható!

4 Makrók – haladó

- Kifejtés
- Plain TEX makró definíció
- Kifejtés – `\def` VS `\let`
- Kifejtés további szabályozása
- Rejtett, belső makrók

Kifejtés újra I

- eddigiek: LaTeX szintaxis
- Plain TeX szintaxissal mélyebbre lehet ásni, finomabban kontrollálni a definíciót és viselkedést
- többek között: kontrollálhatjuk a kifejtést (lásd még: 1. szakasz)
  - makró definíciója: `\parancs` = kifejtés
  - de a kifejtésben szerepelhetnek újabb makrók! – kifejtés második, harmadik, stb. szintje/lépése
  - mikor történik a kifejtés? általában csak a fájl fordításakor
  - Plain TeX szintaxissal a makró definíciójakor is kifejthető a jobb oldal, egy szintig, vagy teljesen

Plain TEX makró definíció I

- `\newcommand` kb. megfelelője: `\def` parancs
  - szintaxis, pl. két argumentummal:
    `\def\skalarszorz#1#2{\langle#1\mid#2\rangle}`
  - itt is max. 9 argumentum, #1 stb., de fel kell sorolni őket
  - az argumentumokat nem feltétlen csoport { } jelöli, alternatív minta alapú felismerés:
    `\def\pointtocomma #1.#2 {(#1,#2)}`,
    `\pointtocomma 3.14` (3,14)
    - figyeljünk az utolsó elválasztóra, hogy jól ismerje fel a minta végét!

Kifejtés – \def VS \let I

- a \def definiáláskor nem fejt ki semmit
- \let parancs: egy kifejtési lépést végez a jobb oldalon, azt adja értékül a bal oldalnak
  - azaz, a jobb oldal egy makró kell legyen, és az ő definícióját másoljuk, az lesz a bal oldal definíciója is
- különbség a kettő között: másképp reagálnak a hivatkozott/másolt parancs újradefiniálására
- kezdetben legyen

  ```
  \newcommand{\vektor}[1]{\vect{#1}}
  ```

  ```
  $\vektor{x}$
  ```
  $\vektor{x}$ $\vec{x}$
- ```
  \def\vekdef#1{\vektor{#1}}
  \let\veklet\vektor
  \renewcommand{\vektor}[1]{\overline{#1}}
  ```
- $\vektor{x}$ $\overline{x}$
  ```

Kifejtés – \def VS \let II

- `$\vekdef{x}$` $\overline{x}$
  - `\vekdef` csak egy szinonima `\vektor`-ra, tulajdonképpen meghívja `\vektor`-t, ezért az újradefiniálással változik ő is
- `$\veklet{x}$` $\vec{x}$
  - `\veklet` az eredeti definíciót másolta, innentől kezdve független `\vektor`-tól, és ragaszkodik az eredeti formához

Kifejtés további szabályozása I

- `\edef` parancs: teljesen kifejti a jobb oldalt, úgy adja értékül a bal oldalnak
- `\def`, `\let` és `\edef` a három definíciós parancs
- eléjük illeszthető módosítók:
  - `\global`: globális, avagy csoporton belül definiálva sem lesz helyi parancs
  - `\long`: argumentumban engedélyezett új bekezdés – itt a nem engedélyezés az alapértelmezett (kb. `\newcommand*` ellentéte)
  - `\outer`: „külső” parancs, bizonyos parancsok argumentumaiban nem használható – kb. törékeny parancsnak felel meg
- definíción belül használható módosítók:
  - `\expandafter`: második argumentumot fejti ki először
  - `\noexpand`: a közvetlenül következő makrót nem fejti ki – `\let`-en vagy `\edef`-en belül hasznos

Kifejtés további szabályozása II

- egyéb hasznos, érdekes parancsok
  - `\empty`: üres makró, első kifejtéskor eltűnik
  - `\relax`: nem csinál semmit, de kifejtéskor ott marad
  - `\string`: kb. `\verb`-nek felel meg, értelmezés nélkül, verbatim kiírja a követő parancsot
  - lehetséges *dupla behelyettesítés*, makró/változó helyettesítése másik makró nevébe
    - `\csname parancs \endcsname` ekvivalens `\parancs`-csal (csname = command sequence name, parancsnév), de közte történhet behelyettesítés
    - például

```
\def\varempty{Üres!}
\def\varfull{Tele van!}
\def\status{full}
\csname var\status \endcsname
```

      Tele van!
    - azaz `\varfull`-t helyettesített

Rejtett, belső makrók I

- emlékeztető: `\ + betűsorozat` típusú parancs beolvasása véget ér, ha nem betű karakterrel találkozik
- egy LaTeX dokumentumban a @ jel nem betűnek számít, hanem „egyéb” karakternek
- ellenben a definíciós fájlokban (mint osztályok, csomagok) a @ jelet betűként kategorizálja(!)
- ez szándékos: belső makrók tartalmazzák a @ jelet – ezzel „rejti el” a belső makrókat a laikus/átlagfelhasználó elől
- de programozáshoz van lehetőség belső makrók meghívására, esetleges átdefiniálására
  - a következő két parancs között tudjuk használni őket
  - `\makeatletter`: betűvé teszi a @-t – hogy elérhetők legyenek a belső parancsok
  - `\makeatother`: ismét „egyéb” karakterré teszi a @-t – újra „elrejtjük” őket, hogy véletlenül ne piszkáljunk bele