LaTeX: kód beillesztése és tételkörnyezetek

Programkód és pszeudokód, tételek és definíciók

Vadon Viktória

2022/23/I. félév

Tételszerű környezetek

- céljuk: tételek, definíciók formázása tudományos cikkekben
- mire jó még? pl. feladatok, kódrészletek vizuális kiemelése
- formázással emel ki: „címsor”, térköz, opcionálisan betűstílusok
- tördelhető, nem float, az adott helyre illesztjük be
- számozott, így lehet `\label`-t helyezni bele

Csomagok nélkül I

- preambulumban definiálható a típus, pl.

  `\newtheorem{tet}{Tétel}`

  - `tet`: környezetnév
  - `Tétel`: megjelenítendő név
- opcionális argumentumok a számozásra:
  - számozás szakaszolási egységen, pl. `section`-ön belül

    `\newtheorem{tet}{Tétel}[section]`
  - együtt számozás (már definiált) másik tételszerű környezettel:

    `\newtheorem{defin}[tet]{Definíció}`
  - figyeljünk a különbségre a pozícióban! ill. egyszerre csak az egyik opcionális argumentumot használjuk!

Csomagok nélkül II

- használat:

```
\begin{defin}
A derékszögű háromszög...
\end{defin}
\begin{tet}[Pitagorasz]
Pitagorasz tétele...
\end{tet}
```

  [Pitagorasz]: opcionális argumentumban megadható a tétel (vagy definíció, stb.) neve
- jó tudni: ha listával kezdődik a tétel, `\leavevmode` paranccsal tudjuk új sorban kezdeni

amsthm csomag I

- amsthm csomaggal (kicsit) bővíthető a funkcionalitás
- tételstílusok \theoremstyle{}
  - plain: alapértelmezett, alap LaTeX tételstílus: címsor vastag betűs, szöveg dőlt betűs
  - definition: címsor vastag betűs, szöveg sima
  - remark: címsor dőlt betűs, szöveg sima
  - szintén preambulumban használandó, az *utána* definiált tételszerű környezetekre vonatkozik (a következő \theoremstyle parancsig)
- számozatlan típusok \newtheorem* paranccsal – értelemszerűen nincsenek opcionális argumentumai

amsthm csomag II

- bizonyítás: `proof` környezet

  ```
  \begin{proof}[Pitagorasz tétel bizonyítása]
  ...
  \end{proof}
  ```

  - `[Pitagorasz tétel bizonyítása]` opcionális argumentumban bizonyítás neve (pl. ha nem közvetlen követi a tételt a bizonyítás)
  - alapértelmezésben kiírja hogy proof/bizonyítás (`babel`!)
  - a bizonyítás végén automatikusan kiteszi a négyzetet: □
  - kézzel: `\qed`
  - áthelyezés `proof`-on belül: `\qedhere`

2 Verbatim

Verbatim I

- `verbatim`: feldolgozatlan szöveg, karakterről karakterre* megjelenít, akár LaTeX kódot is; `typewriter` betűkészlettel formázza
  - * szinte; pl. tabulátor kivétel
- `\verb||` parancs: „egysoros” verbatim
  - szöveggel egy sorba kerül; tartalmát nem tördeli a LaTeX, és nem lehet benne sortörés!
  - argumentumát *nem* kapcsos zárójelek határolják! a `\verb` parancsot követő *első karakter*, ami tetszőleges, de a következő ugyanolyan karaterig olvas. általában: |
- pl. kód:

```
sorközi \verb|\texttt{verbatim}| szöveg
```

hatás:

sorközi \texttt{verbatim} szöveg

Verbatim II

- `verbatim` környezet: többsoros verbatim
  - `\begin{verbatim}` nyitja, *új sorba tett* `\end{verbatim}` zárja
  - a sortöréseket is megtartja
  - előtte, utána új sor / saját függőleges blokkjába kerül
- pl. kód:

```
\begin{verbatim}
\begin{figure}
\includegraphics{kep}
\end{figure}
\end{verbatim}
```

hatás:

```
\begin{figure}
\includegraphics{kep}
\end{figure}
```

3 Pszeudokód

- `algpseudocode` csomag
- Alap parancsok
- Blokkok
- Procedúra, függvény
- Algoritmusok mentése, folytatása
- Saját parancsok

Algoritmus, pszeudokód

- pszeudokód: algoritmusok programnyelvtől független leírása
- LaTeX-ben: `\usepackage{algpseudocode}` csomag, kész környezet és parancsok pszeudokód írására és formázására
  - `algorithmicx` *bundle* (= „csomagok csomagja") része – kézi telepítés esetén ezt keressük
- `algorithmic` környezet – (`\begin{algorithmic}`, `\end{algorithmic}`)
  - csak ezen belül élnek a csomag parancsai
  - környezet opcionális argumentuma (pozitív egész szám): (kód)sorok számozása, pl. *minden 5. sor*:

    `\begin{algorithmic}[5]`
- általános tudnivaló: figyeljünk a nagybetűkre a parancsokban!

algorithmic alap parancsai I

- figyeljünk a nagybetűkre!
- utasítás sor: `\State`, értékadás, műveletek, stb.
  - szinte minden sor elejére – behúzások kezeléséhez
- pl. `\State $x \gets 0$`
- matematikai szimbólumok, formulák írása (gyorstalpaló)
  - minden matematikai szimbólumok vagy képlet: `$` jelek közé
  - pl. értékadás `$\gets$` $\leftarrow$
  - pl. index: `$x_i$` x_i, `$x^n$` x^n
  - pl. (halmaz) eleme: `$\in$` $\in$, tagadva `$\not\in$` $\notin$
  - későbbi órán bővebben
- üres sor: `\Statex` (üres sor a forrásban nem csinál semmit)
- megjegyzés beszúrása `\Comment{megjegyzés}`
  - pl. `\State $x \leftarrow 0$ \Comment{inicializálás}`

    $x \leftarrow 0$ ▷ inicializálás

Blokkok – ciklus, feltételvizsgálat, függvény I

- példa `while` ciklusra:

```
\begin{algorithmic}[1]
\State x $\leftarrow$ 0
\While{x < 10}
\State print(x)
\State x $\leftarrow$ x+1
\EndWhile
\end{algorithmic}
```

1: $x \leftarrow 0$
2: **while** $x < 10$ **do**
3: print(x)
4: $x \leftarrow x+1$
5: **end while**

Blokkok – ciklus, feltételvizsgálat, függvény II

- az `algorithmic` környezet blokkokként kezeli és jeleníti meg a ciklust, feltételvizsgálatot, függvényt
- behúzással jelzi – automatikusan
- van nyitó- és záró parancsa – ne felejtsük el zárni őket!
- alapértelmezésben ki is írja a záró parancsot
- záró parancsok „kikapcsolása”: `noend` csomag opció, `\usepackage[noend]{algpseudocode}` – de attól még nekünk a LaTeX forráskódban zárni kell a blokkokat!

Blokkok – ciklus, feltételvizsgálat, függvény III

- `while` ciklus: nyitás `\While{feltétel}`, zárás `\EndWhile`
- `for` ciklus
  - nyitás `\For{fejléc}`, pl. `\For{x $\leftarrow$ 1 \textbf{to} 10}`
  - *vagy* `\ForAll{fejléc}`, pl. `\ForAll{x $\in$ A}`
  - zárás (mindkettőhöz!) `\EndFor`
- feltételvizsgálat, `if`, `elif`, `else`:
  - nyitás `\If{feltétel}`
  - opcionális (közbülső) `\ElsIf{feltétel}` (betűzés!)
  - opcionális (közbülső) `\Else`
  - zárás `\EndIf`
- `repeat until`: `\Repeat`, `\Until{feltétel}`

Porcedúra, függvény I

- procedúra: nyitás `\Procedure{név}{paraméterek}`, zárás `\EndProcedure`
- függvény hasonlóképp `\Function{név}{param}`, `\EndFunction`
- ```
  \begin{algorithmic}[1]
  \Procedure{bubblesort}{@A}
  \State ...
  \EndProcedure
  \end{algorithmic}
  ```

procedure BUBBLESORT(@A)
2: ...
end procedure
  ```

Porcedúra, függvény II

- procedúra „hívás”: `\Call{név}{paraméterek}`
  - csak formáz – nem dolgoz fel semmit, nem kell, hogy a hivatkozott procedúra/függvény „definiálva” legyen pszeudokódban
  - utasítás sorban, `\State` paranccsal együtt használjuk!

    ```
    \State \Call{binsort}{A,@B,@C}
    ```

    BINSORT(A,@B,@C)
- egyéb: `\Require`, `\Ensure` – „feltételek”
  - `\State` nélkül használhatók!

    ```
    \Require A egy írható tömb
    \Ensure n egy pozitív egész szám
    ```

    Require: A egy írható tömb
    Ensure: n egy pozitív egész szám

Algoritmusok mentése, folytatása I

- cél: hosszabb algoritmus több részre bontása, mentés és folytatás
- mentés
  - mentés `\algstore{név}` – muszáj folytatni
  - vagy: `\algstore*{név}` – tilos folytatni(!)
  - mentés legyen az `algorithmic` utolsó sora!
  - mentéskor automatikusan zárja a blokkokat, menti a kódsor számát és behúzást
- betöltés
  - betöltés `\algrestore{név}` – létező mentés neve kell; automatikusan törli is a mentést(!)
  - betöltés, a mentés törlése nélkül: `\algrestore*{név}`
  - betöltés legyen az `algorithmic` első sora
  - betöltéskor automatikusan újranyitja a blokkokat, folytatja a sorszámozást, és megfelelő behúzással folytat

Saját parancsok I

- blokkok definiálása:

```
\algblockdefx[blokknév]{nyitás}{zárás}%
[nyitó paraméterszám]{nyitáskor kiírandó}%
[záró paraméterszám]{záráskor kiírandó}
```

  - nyitás: blokkot nyitó parancs, amit \-szel egészít ki – azaz \nyitás lesz a nyitó parancs(!)
  - zárás: hasonlóképp
  - blokknév opcionális – automatikusan nyitás a blokk neve
  - paraméterszám opcionális – ha nincs, hagyjuk ki!
  - paraméterek behelyettesítése a kiírandó szövegben: #1, #2, stb.
  - pl. az alap If blokk definíciója valahogy így nézhet ki:

```
\algblockdefx{If}{EndIf}%
[1]{\textbf{if}~#1~\textbf{then}}%
{\textbf{end if}}
```

Saját parancsok II

- folytatásos blokk definiálása – pl. `ElsIf`, `Else` ágak *hozzáadása* a *már létező* `If`-hez:
  - gyakorlatilag „megszakítja” a régi blokkot és újat kezd (aminek történetesen lehet ugyanaz a záró parancsa, de nem kötelező)
  - `\algcblockdefx[újblokknév]{régiblokk}%`
    `{folytatás}{zárás}%`
    `[folytatás paraméterszáma]{folytatáskor kiírandó}%`
    `[zárás paraméterszáma]{záráskor kiírandó}`
  - `{régiblokk}`: a blokk neve, amit megszakít
  - `[újblokknév]`: opcionális, alapértelmezésben `folytatás`
  - egyébként a szintaxis megegyezik a `kezdés`-sel
- az `Else` ág kódja valami ilyesmi:

```
\algcblockdefx{If}{Else}{EndIf}%
{\textbf{else}}%
{\textbf{end if}}
```

Saját parancsok III

- az `ElsIf` ág kódja valami ilyesmi:

```
\algcblockdefx[If]{If}{ElsIf}{EndIf}%
[1]{\textbf{else if}~#1~\textbf{then}}%
{\textbf{end if}}
```

  - vegyük észre: az `újblokknév` *is* `If`!
  - ez miért jó? ami meg tudja szakítani az `If` blokkot (`Else` és `ElsIf`), az meg tudja szakítani az `ElsIf` blokkot is(!)
  - azaz, `ElsIf` halmozható, és szintén követheti `Else`
  - az `Else`-nél direkt nem csináltunk ilyet, mert az `Else` után nem jöhet más, csak az `EndIf`!

4 Felirat, úsztatás

- Lehetőségek
- `algorithm` csomag

Felirat, úsztatás

- a `verbatim` és az `algorithmic` is csak a kódot formázzák
- ha szeretnénk feliratozni, sorszámozni, esetleg törhetetlenné tenni és úsztatni, *be kell csomagolni* egy másik, arra alkalmas környezetbe
- ha nem kell úsztatni, csak feliratozni és sorszámozni, ez a csomagoló lehet egy tételszerű környezet – ld. 1 szakasz
- ha szeretnénk, hogy törhetetlen legyen és ússzon:
  - `float` csomaggal saját float típus(ok) (ld. 3. óra anyaga)
  - `algorithm` csomag: definiálja az `algorithm` float típust (következő oldal)

algorithm csomag I

- \usepackage{algorithm}
  - csomag opció: algorithm float számozása valamilyen szakaszolási egységen belül
  - pl. \usepackage[section]{algorithm}: section-ön belül
  - lehet még: part, chapter, subsection, subsubsection
- algorithm környezet: új float típus, csomagolható bele úsztatásra szánt kód
- szokásos módon működik a float elhelyezése hbtp! betűkkel, a \caption és \label – ld. 3. óra anyaga
- listázás \listofalgorithms paranccsal

algorithm csomag II

- babel nem fordítja le a \caption-ben generált „algorithm” feliratot és a lista címsorát
  - átnevezés kézzel
  - algoritmus neve: \floatname{algorithm}{Algoritmus}
  - lista címsora

```
\renewcommand{\listalgorithname}%
{Algoritmusok listája}
```

5 Programkód

- `listings` csomag
- `listing` kódok beállításai
- Úsztatás

Programkód beillesztése – `listings`

- javasolt: `listings` csomag – dokumentáció
  - sokféle programnyelvet ismer és annak megfelelően formáz
- kód beszúrása:
  - egysoros (mint a `\verb||`): `\lstinline|kód|` – tetszőleges* határoló karakterrel (kivéve nyitó szögletes zárójel `[`)
  - külső fájlból `\lstinputlisting{relatív/elérés/fájl.kit}`
  - többsoros, LaTeX kódba bemásolt kód (mint a `verbatim` környezet): `lstlisting` környezetbe
- beállítása:
  - általánosan: csomag opciók `\usepackage[kapcsoló,kulcs=érték]{listings}`
  - szintén általánosan: `\lstset{kapcsoló,kulcs=érték}`
    - preambulumban vagy dokumentumtörzsben is!
    - az ez után definiált kód(részlet)ekre, felülírásig
  - helyileg, parancs vagy környezet opcionális argumentumában

listing kódok beállításai I

- programnyelv (ami szerint formáz): `language=...`, pl. `python`, `c`, stb.
  - `\lstinline[language=python]|kód|` – CHECK!
  - `\begin{lstlisting}[language=python]`
  - `\lstinputlisting[language=python]{fájlnév.kit}`
- megjelenített sorok (pl. ha egy hosszabb fájlból csak egy procedúrát szeretnénk beilleszteni)
  - tartomány(ok): `linerange={5-20,56-82}`
  - csak első sor `firstline=` és/vagy utolsó sor `lastline=` megadása (ha csak némi whitespace-t kel lecsípni, vagy egyetlen tartomány kell)

listing kódok beállításai II

- sorok számozása
  - ki-bekapcsolás és elhelyezés: `numbers=left` balra, vagy `=right` jobbra, `=none` kikapcsolás
  - hány soronként: pl. minden 3. sor számozása `stepnumber=3`
  - milyen számmal kezdi az első sor számozását: pl. `firstnumber=2`
- sorszámozás folytatása
  - az előző `listing`-ből `firstnumber=last`
  - szériák létrehozása: `name=szérianév` minden összetartozó `listing`-be (névnek hívja, de nem kell, hogy egyedi legyen)
  - `firstnumer=auto` opcióval folytonosan számozza az azonos nevűeket (a szériát)

listing kódok beállításai III

- szóközök és tabulátorok
  - `tabsize=4`: tabulátor hány szóköz
  - `showspaces`, vagy `showspaces=true`: szóközök mutatása
  - kikapcsolás: `showspaces=false`
  - hasonlóképp tabulátorra: `showtabs`
- keretezés `frame=`
  - beépített stílusok:
  - `frame=none` nincs keret (alapértelmezett)
  - `=leftline`, `=topline`, `=bottomline`: egyetlen vonal bal oldalt, felül, ill. alul; `=lines`: felül és alul vonal
  - `=single`: egyszeres keret körbe, `=shadowbox`: „árnyékolt” doboz
  - kézzel:
  - `lrtbLRTB` betűk tetszőleges kombinációja
  - l – left, bal, r – right, jobb, t – top, felül, b – bottom, alul
  - kisbetű: egyszeres, nagybetű: dupla vonal; hiányzó: nincs vonal

`listing` úsztatása I

- a `listing` kilóg a sorból, őt nem feltétlenül kell csomagolni, hanem beépítve float-tá tudja alakítani magát!
- `float` opció az adott `lstlisting` környezet (vagy `\lstinpulisting` parancs) opcionális argumentumába:
  - pl. `\begin{lstlisting}[float]`
  - float elhelyezése: `float=hbt!` formában; szokásos betűk használhatók
- szintén opcionális argumentumba(!) `caption={}`, `label=` – úsztatás *nélkül* is működnek(!)
- például

```
\begin{lstlisting}[float=hb!,%
caption={[Buborék]A buborék rendezési algoritmus},%
label=lst:buborek]
```

- zölddel: `\caption` rövid verziója

`listing` úsztatása II

- számozás nélküli felirat: `title=` (a `caption=` helyett)
- listázás: `\lstlistoflistings` (betűzés!)
  - adott kód elrejtése a listából: `nolol` az opcionális argumentumba
- `babel` nem fordítja le a lista címsorát és a `listing` típusát a `caption`ben – kézzel újradefiniálhatók:
  - típus: `\renewcommand{\lstlistingname}{Programkód}`
  - lista címe:

    ```
    \renewcommand{\lstlistlistingname}%
    {Programkódok listája}
    ```

    (betűzés!)
- ha több nyelv kódjait külön szeretnénk számozni és listázni, akkor szükség lesz a `float` csomagra és saját float-ok definiálására