LaTeX float-ok kiegészítés

Pszeudokód

Vadon Viktória

2022/23/I. félév

1. Pszeudokód
   - `algpseudocode` csomag
   - Alap parancsok
   - Blokkok
   - Procedúra, függvény
   - Algoritmusok mentése, folytatása
   - Saját parancsok
   - Felirat, úsztatás

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`!

Felirat, úsztatás lehetőségei

- az `algorithmic` is csak a kódot formázza
- 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 – később tanuljuk
- ha szeretnénk, hogy törhetetlen legyen és ússzon:
  - `float` csomaggal saját float típus(ok)
  - vagy: `algorithm` csomag: definiálja az `algorithm` float típust (folyt. köv.)

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}
```