3. feladatsor
Speciális szoftverek – LaTeX

Vadon Viktória

2025/26/I. félév

Szükséges csomagok

- subcaption
- array, multirow
- xcolor vagy colortbl
- float
- listings
- wrapfig
- algpseudocode

Korábban már használt, de most is szükséges csomagok

- graphicx
- hulipsum (vagy lipsum, vagy blindtext)
- hyperref

1. Vetítés, demonstráció

1. Feladat (Képek, ábrák)**.**

a) `\includegraphics` bekezdésen belül („szöveggel egy sorba”)
b) kép méretezése
c) úsztatás: csomagolás `figure` környezetbe, float elhelyezési opciók
d) `\caption`, `\label`
e) `\listoffigures`

2. Feladat (Táblázat)**.**

a) `tabular` szintaxisa, rácsvonalak
b) cellák egyesítése: `\multicolumn`, `\multirow`, és kombinációjuk
c) oszlopra vonatkozó kód beszúrása `array`-jel, pl. szöveg színe

1. táblázat. Demonstráció

(a) Igazítás, rácsvonalak

	egy	kettő	három
Helló világ!	négy	öt	hat
	hét	nyolc	kilenc
lórum ipse	tíz		tizenkettő

(b) Cellák egyesítése

egy	kettő	
négy	öt	hat
tíz	nyolc	

3. Feladat (Programkód)**.**

a) verbatim: inline és környezet
b) `lstlisting`: inline és környezet, programnyelv beállítása
c) float, caption, title

2. Kötelező feladatok

4. Feladat (Részábrák)**.**

a) folytassuk a munkát a korábbi fájlban
b) a figure környezetbe (pl. a már ott lévő kép és `\caption` alá) helyezzünk el egy második képet, torzítás nélkül, max. 5cm-es szélességgel *és* max. 5cm-es magassággal!
c) transzformáljuk valamilyen módon: tükrözzük, forgassuk, stb.
d) adjunk `\caption`-t ennek a transzformált képnek is. mi történik helyileg, ill. a képek listájában?
e) töltsük be a `subcaption` csomagot, és a `figure`-ön belül készítsünk külön-külön `subfigure` környezetet a két képnek!
f) méretezzük úgy a `subfigure`-öket (és bennük a képeket), hogy elférjenek egymás mellett!
g) helyezzünk el `\caption`-öket „mindenhová”: a `subfigure`-ökben külön-külön, illetve a `subfigure`-ökön kívül a `figure`-ben! mi a különbség?
h) szépítsük az elrendezést:
   - `\hspace` segítségével tegyünk némi térközt a két subfigure közé
   - igazítsunk középre mindent (képeket a subfigure-ön belül, és az egész elrendezést a figure-ön belül)
   - ha szükséges, kísérletezzünk a subfigure-ök beállításaival, hogy függőlegesen is középre igazodjanak
i) helyezzünk el `\label`-öket mindhárom különböző `\caption` után! melyik label-(ök)re tudunk `\ref`, vagy `\subref` paranccsal hivatkozni? mi a különbség a kettő között?

5. Feladat (Táblázatok: `array` csomag)**.**

a) készítsünk egy másolatot a minta táblázatról.
b) növeljük meg a sorok magasságát 2pt-vel.
c) a fix szélességű és sortördelt, „bekezdés típusú” p oszloptípust cseréljük ki egy szintén fix szélességű, de függőlegesen középre zárt típusra!
d) szúrjunk be kódot ezen oszlop elé, hogy vízszintesen is középre igazítsuk!
e) definiáljunk egy új oszloptípust a fenti tulajdonságokkal (vízszintesen és függőlegesen is középre zárt, bekezdés típusú)!

6. Feladat (Táblázatok színezése)**.**

a) töltsük be a `colortbl` csomagot, vagy az `xcolor` csomagot `[table]` opcióval + az opciók nagy részéhez szükség lesz `array` csomagra is
b) a 2. táblázat mintájára:
   - az egyik oszlop kapjon sötét hátteret és világos betűszínt!
   - az egyik sor is kapjon színes hátteret
   - állítsuk át az összes rácsvonal színét egységesen
   - állítsuk át csak az egyik függőleges rácsvonal színét egy másik színre kézzel!
   - egy cella kapjon külön háttérszínt
   - egy cella kapjon külön betűszínt

2. táblázat. Tarka

egy	kettő	három
négy	öt	hat
hét	nyolc	kilenc
tíz	tizenegy	tizenkettő

7. Feladat (Programkód 1 – input, formázás alapok)**.**

a) `lstinputlisting` segítségével olvassuk be a mellékelt .C fájlt.
b) állítsuk be a beolvasott sorokat úgy, hogy csak az egyik függvényt jelenítse meg!
c) állítsunk be 2 szóköznyi tabulátorszélességet, a szóközök és tabulátorok legyen láthatatlanok!
d) számozzuk a kódsorokat a bal oldalon, pl. 4-esével
e) válasszunk egy szimpatikus keretezési módot!
   - használhatunk előre definiált keret-stílust, vagy kézi kombinációt is
   - ha szükséges, növeljük meg a bal margót, hogy a sorszámozás a kereten belülre kerüljön!

tipp az 1. C kód minta/inspiráció a formázáshoz.

8. Feladat (Float csomag)**.**

a) dolgozzunk a korábbi fájlban, ahol a képek, ill. programkódok szerepelnek.
b) a float csomaggal keretezzük automatikusan az összes ábrát (figure-t)!
c) float csomag segítségével definiáljunk két új float környezetet, külön a Python és C kódoknak! ezeket *ne* keretezzük!

1. C program. Bináris keresés C-ben

```
1  binarySearch(arr, x, low, high)
     repeat till low = high
       mid = (low + high)/2
         if (x == arr[mid])
5          return mid

         else if (x > arr[mid])   // x is on the right side
           low = mid + 1
9
         else        // x is on the left side
           high = mid - 1
```

d) csomagoljuk a listing-jeinket a megfelelő float környezetekbe, és adjunk nekik caption-t!
   - definiáljuk a típusok megjelenítendő nevét is, hogy a caption-ök „magyarul olvashatók" legyenek!

e) készítsünk egy másolatot a C kódrészletről, és módosítsuk a beolvasott sorokat, hogy a másik függvényt tartalmazza! ennek megfelelően módosítsuk a caption-t is.

f) listáztassuk ki külön-külön a Python, illetve C kódokat!

3. Választható feladatok

9. Feladat (`wrapfigure`)**.**

a) szúrjunk be egy képet 5cm-es szélességgel

b) csomagoljuk `wrapfigure` környezetbe, amit a jobb, vagy külső margóra helyezzünk el

c) a wrapfigure előtt és után is tegyünk 1-2 bekezdésnyi helykitöltő szöveget. figyeljük meg, hogy „'körbefut"-e a szöveg a kép körül?
   - ha szükséges, változtassuk meg a wrapfigure elhelyezés argumentumát, vagy a kódban cseréljük fel a wrapfigure és a helykitöltő szöveg sorrendjét.

d) állítsuk be a wrapfigure szélességét és a ezen belül a kép igazítását, hogy szép legyen!

e) ha szükséges, adjuk meg kézzel a `wrapfigure` magasságát is, hogy szép legyen!

10. Feladat (Táblázatsorok váltakozó színezése)**.**

a) ez sajnos rendszertől, compiler-től függően vagy működik, vagy nem :(
b) a 3. táblázat mintájára:
   - színezzük egy táblázat páros ill. páratlan sorait különböző háttérszínnel
   - a színezés csak a második sortól legyen érvényben!
   - automatikusan illesszük be a vízszintes rácsvonalat is minden sor után.

3. táblázat. Zebra

egy	kettő	három
négy	öt	hat
hét	nyolc	kilenc
tíz	tizenegy	tizenkettő

11. Feladat (Programkód 2 – stílus, formázás öröklődése)**.**
a) dolgozzunk a korábbi Python kódrészlettel, vagy másoljuk le.
b) az `\lstdefinestyle` segítségével definiáljunk egy *stílust* a kód színezésére
   - pl. legyen a háttér halványszürke, a kulcsszavak/parancsok sötétpirosak, a függvénynevek stb. sötétkékek, a kommentek pedig szürkék
c) alkalmazzuk ezt a saját stílust, illetve a Python nyelvet a kódunkra!
d) hogyan tudjuk elérni, hogy a tabulátorok és számozás formázása, amit a korábbi C kódhoz állítottunk be, érvényben maradjon?
e) a keretet távolítsuk el a Python kódunkból!
   - a margó beállítására viszont a színes háttérhez is szükség lehet!

tipp 1. Python kód minta/inspiráció a formázáshoz

12. Feladat (Pszeudokód)**.**
a) ehhez a forrásanyag külön szorgalmi diasorban található!
b) töltsük be az `algpseudocode` csomagot!
c) készítsük el vele egy tetszőleges algoritmus pszeudokódját!
   - pl. egy adatstruktúrák és algoritmusok tárgyból tanult algoritmust;
   - mintaként lentebb mellékelt a felosztás (2. algoritmus) és gyorsrendezés (1. algoritmus), az `algpseudocode` csomaggal formázva.
d) számozzuk a sorokat, pl. kettesével!
e) `[haladó]` definiáljuk saját blokként a `do-while` ciklust! teszteljük is!
   - vigyázzunk, az általunk definiált parancsok ne egyezzenek meg létezőkkel!
f) `[haladó+]` definiáljuk saját *folytatható* blokként a `switch-cases` esetszétválasztást!

1. Python program. Bináris keresés és beszúró rendezés Python-ban

```
def binary_search(arr, val, start, end):
  if start == end:
    if arr[start] > val:
      return start
    else:
      return start+1
  elif start > end:
    return start
  else:
    mid = (start+end)/2
    if arr[mid] < val:
      return binary_search(arr, val, mid+1, end)
    elif arr[mid] > val:
      return binary_search(arr, val, start, mid-1)
    else: # arr[mid] = val
      return mid

def insertion_sort(arr):
    for i in xrange(1, len(arr)):
        val = arr[i]
        j = binary_search(arr, val, 0, i-1)
        arr = arr[:j] + [val] + arr[j:i] + arr[i+1:]
    return arr
```

1. Algoritmus Gyorsrendezés

procedure QUICKSORT(@A,a,b)
Require: A írható tömb
Require: $1 \leq a \leq b \leq$ Hossz[A] indexek
Ensure: a–b indextartományt rendezzük
  if a=b **then**
    return A ▷ egyelemű tömb mindig rendezett
  else
    FELOSZT(@A,a,b,A(a),@q) ▷ k=A(a), a tartomány első eleme
    QUICKSORT(@A,a,q)
    QUICKSORT(@A,q+1,b)
    return A
  end if
end procedure

2. Algoritmus Felosztás

```
    procedure FELOSZT(@A,a,b,k,@q)
Require: A írható tömb
Require: 1 ≤ a ≤ b ≤ Hossz[A] indexek
Require: k A-beli kulcs
Ensure: A átrendezése és q választása úgy, hogy: a – q indextartomány elemei
    ≤ k, (q+1) – b indextartomány elemei ≥ k
 2:     i ← a-1
        j ← b+1
 4:     while i<j do                          ▷ ekvivalens: while true do...
            repeat
 6:             INC(i)
            until A(i) ≥ k
 8:         repeat
                DEC(j)
10:         until A(j) ≤ k
            if i < j then
12:             A(i) ↔ A(j) csere
            else
14:             q ← j
                return (A,q)
16:         end if
        end while
18: end procedure
```