4. feladatsor
Bevezetés a TEX szövegszerkesztésbe

Vadon Viktória

2022/23/I. félév

Szükséges csomagok

(A csomagok közül némelyik az alap LaTeX telepítés része, így nem mindet kell kézzel telepíteni)

- `amsthm`
- `algpseudocode` – `algorithmicx` *bundle* része (kézi letöltéskor a bundle nevét keressük MikTeX Console-ban)
- `algorithm`
- `listings`

Korábban már használt, de most is szükséges csomagok

- `babel`
- zagyva szöveg: `lipsum` vagy `hulipsum` vagy `blindtext`
- `float`

Egyéb szükséges források

- megírt kódrészlet(ek) tetszőleges (nem LaTeX) programnyelven
  - ha nincs, a feladatsor végén, van egy Python kódrészlet – forrása pedig a honlapomon az órai mappában `binsearch.py` néven

1. Feladat (Tételkörnyezetek)**.**

a) definiáljunk *Tétel* és *Definíció* (megjelenítendő) néven tételkörnyezeteket, és helyezzünk el belőlük a dokumentumban 2-2-t!

b) definiáljunk *Lemma* néven egy tételkörnyezetet, amit a Tétellel együtt számoz! helyezzünk el a dokumentumba (legalább) 1 Lemmát!

c) definiáljunk *Feladat* néven egy tételkörnyezetet, amit `section`-önként számoz! hozzunk létre két `section`-t, és legalább 2+1 Feladatot bennük! a korábbi tételkörnyezetekből is helyezzünk át párat a második `section`-be
d) milyen csomaggal tudjuk megváltoztatni a tételek stílusát? töltsük be!
e) a Tétel és Lemma legyenek `plain` stílusúak, a Definíció legyen `definition` stílusú, a Feladat pedig `remark` stílusú!
f) az egyik Tétel után helyezzünk el egy *bizonyítást*!
g) definiáljunk egy *számozatlan* Megjegyzés tételkörnyezetet, és helyezzünk el egy Megjegyzést a dokumentumba!

2. Feladat (Verbatim (és saját float))**.**

a) helyezzünk el szöveggel egy sorba két-három verbatim `\LaTeX` parancsot!
b) külön-külön verbatim környezetben szemléltessük egy tételkörnyezet és egy lista kódját!
c) töltsük be a `float` csomagot és az előző órai anyag alapján definiáljunk egy új float típust a `verbatim` csomagolására, pl. `forraskod` néven
d) csomagoljuk be a verbatim környezeteket egy-egy `forraskod` float-ba, és feliratozzuk őket!
    - gondoskodjunk róla, hogy a feliratban szereplő típus magyarul legyen!
e) helyezzünk el a dokumentumba (a float környezeteken *kívül*!) némi zagyva szöveget, hogy lássuk az úsztatás hatását!
f) a dokumentum elején listáztassuk ki a `forraskod` float-okat! válasszunk magyar címsort!

3. Feladat (Programkód)**.**

a) töltsünk be a `listings` csomagot, és segítségével helyezzünk el vele egy 10-20 soros programrészletet (lehetőleg egy procedúrát vagy függvényt) a LaTeX fájlba
    - lehet külső fájlból beolvasott, vagy copy-paste-elt is
    - ha túl hosszú a rendelkezésre álló kód, csípjünk ki belőle tartomány(oka)t!
b) állítsuk be a programnyelvet az automatikus formázáshoz
c) állítsunk be egy szimpatikus/a nyelvhez megfelelő tabulátorszélességet!
d) számozzuk a kódsorokat pl. 4-esével (vagy ahogy szimpatikus)
e) kísérletezzünk a keretezéssel és válasszunk egy szimpatikus megjelenést!
f) alakítsuk a `listing`-ünket float-tá!
g) adjunk neki feliratot is, és listáztassuk a fájl elején!
    - ahol szükséges, fordítsuk magyarra a generált szövegeket!

4. Feladat (Pszeudokód)**.**

a) töltsük be az `algpseudocode` csomagot!
b) készítsük el vele egy rendezési algoritmus pszeudokódját!
    - pl. adatstruktúrák és algoritmusokból tanult; a dokumentum végén gyorsrendezés (és felosztás)

c) számozzuk a sorokat pl. kettesével!

d) töltsük be az `algorithm` csomagot, és csomagoljuk az `algorithm` környezetbe a pszeudokódunk, hogy úszhasson! feliratozzuk és listáztassuk!
   - fordítsuk magyarra a lista és típus nevét!

e) definiáljuk saját blokként a `do-while` ciklust! teszteljük is!
   - vigyázzunk, a parancsok ne egyezzenek meg létezőkkel!

f) definiáljuk saját *folytatható* blokként a `cases` esetszétválasztást!

Programkód 1. Bináris keresés Python-ban

```
1  def binary_search(arr, val, start, end):
     if start == end:
       if arr[start] > val:
         return start
5      else:
         return start+1
     elif start > end:
       return start
9    else:
       mid = (start+end)/2
       if arr[mid] < val:
         return binary_search(arr, val, mid+1, end)
13     elif arr[mid] > val:
         return binary_search(arr, val, start, mid-1)
       else: # arr[mid] = val
         return mid
17
   def insertion_sort(arr):
       for i in xrange(1, len(arr)):
           val = arr[i]
21         j = binary_search(arr, val, 0, i-1)
           arr = arr[:j] + [val] + arr[j:i] + arr[i+1:]
       return arr
```

Algoritmus 1 Gyorsrendezés

```
    procedure QUICKSORT(@A,a,b)
Require: A írható tömb
Require: 1 ≤ a ≤ b ≤ Hossz[A] indexek
Ensure: a–b indextartományt rendezzük
 2:     if a=b then
            return A                          ▷ egyelemű tömb mindig rendezett
 4:     else
            FELOSZT(@A,a,b,A(a),@q)           ▷ k=A(a), a tartomány első eleme
 6:         QUICKSORT(@A,a,q)
            QUICKSORT(@A,q+1,b)
 8:         return A
        end if
10: end procedure
```

Algoritmus 2 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
```