Bevezetés a MATLAB programozásba
Feladatok II

Beadási határidő: 2013. november 18. (23:00)

A feladatok utáni zárójelben a feladatra kapható maximális pontszám szerepel. A megszerezhető összpontszám: 600. A gyakorló feladatokra összesen 300 pont, az alkalmazásra szintén 300 pont szerezhető.

1. Gyakorló feladatok

1. Feladat (20)**.** Hozzunk létre egy az 1. táblázatnak megfelelő `Subject` nevű struktúrát, határozzuk meg (a válaszokat cellákban adjuk meg):

a) a három legidősebb alany nevét,

b) a harminc év alattiak neveit,

c) a legalább 160 cm magas és legfeljebb 60 kg alanyok neveit,

d) az átlag életkort,

e) a súlyok átlagát és szórását,

f) a magasságok átlagát és szórását,

g) az átlagon felülien magasakat,

h) a legfiatalabb alany magasságát.

Subject			
Name	Age	Height	Weight
Kovács Gabriella	34	173	78
Papp Tamás	40	169	90
Szabó András	27	195	103
Tóth Virág	17	162	45
Varga Emese	18	156	40
Zombori Anna	65	170	52

1. táblázat. Adatok a *Subject* struktúrához

2. Feladat (50)**.** Írjunk egy `getBMI` nevű függvényt, amelynek bemenete az előző feladatban létrehozott `Subject` struktúra tömb egy eleme. Ha a függvényt kimenet nélkül hívjuk meg, akkor írja ki a megadott struktúrában lévő alany BMI indexét és osztályozását szövegesen. Ha egy kimenettek hívjuk meg a függvényt, akkor a függvény a BMI indexszel térjen vissza. Ha a függvényt két kimenettel hívjuk meg, akkor a BMI indexen túl az osztályozást is adja meg egy egész szám formájában.

Toovábbá a `getBMI` függvény használatával:

	Osztályozás	
BMI [kg/m^2]	Szöveges	Szám
–16	*súlyos sovány*	–3
16–16.99	*mérsékelt sovány*	–2
17–18.49	*enyhe sovány*	–1
18.5–24.99	*normál testsúly*	0
25–29.99	*túlsúlyos*	1
30–34.99	*I. fokú elhízás*	2
35–39.99	*II. fokú elhízás*	3
40–	*III. fokú elhízás*	4

2. táblázat. A testtömeg-index (BMI) szerinti osztályozás

a) írassuk ki a BMI információt minden alanyról,

b) írassuk ki a BMI információt a sovány alanyokról,

c) írassuk ki a BMI információt a túlsúlyos alanyokról,

d) határozzuk meg a sovány alanyok átlag magasságát,

e) határozzuk meg a normális testsúllyal rendelkezők átlag korát.

f) határozzuk meg az átlag életkor alatti alanyok neveit.

```
>> getBMI(Subject(3))

Nev: Szabo Andras
BMI: 21.8
Osztalyozas: normalis testsuly
>> bmi = getBMI(Subject(3))
bmi =
   21.8277
>> [bmi bmiClass] = getBMI(Subject(3))
bmi =
   21.8277
bmiClass =
     0
```

1. Programlista. Példa a `getBMI` függvény használatára

3. Feladat (50)**.** Készítsünk egy menüvel rendelkező programot, aminek segítségével interaktívan kezelhetjük a `Subject` struktúrát. A program kezdetekor ellenőrizzük, hogy létezik-e már a `Subject` nevű változó. Ha a `Subject` nevű változó nem létezik, akkor próbálkozzunk a `Subject.mat` állomány betöltésével, ha pedig ez sem megy, akkor hozzunk létre egy `Subject` stuktúrát a megfelelő mezőkkel. Értesítsük a felhasználót, hogy mi történik. A program elindítása után a következő menüpontok legyenek elérhetők:

Új adat bevitele Ezt a menüpontot választva a program kérdezze végig a lehetséges adatokat, majd térjen vissza a menühöz.

Adat törlése Ezt a menüpontot választva a program lehetőséget ad törölni név vagy sorszám alapján. A törlés előtt a program kiírja a törölni kívánt alany adatait és megerősítést kér a törléshez. A törlés után a program visszatér a menühöz.

Adat lekérdezése Ezt a menüpontot választva a program lehetősget ad keresésre név vagy sorszám szerint. A keresés után a program kiírja a találatokat, majd felajánlja az új keresés lehetőségét. Ha a felhasználó nem keres tovább a program térjen vissza a menühöz.

Kilépés Ezt a menüpontot választva a program kérdezze meg a felhasználót, hogy szeretné-e menteni a változásokat vagy sem, és a válasznak megfelelően cselekedjen, majd lépjen ki a programból.

4. Feladat (20)**.** Állítsuk elő az `{'a', 'b', 'c'}` és `{1, 2, 3, 4, 5}` cellákból az

```
S = {{'a', 1}, {'a', 2}, ..., {'a', 5}, {'b', 1}, {'b', 2},...,{'c', 5}}
```

cellát egymásba ágyazott ciklusok segítségével. A `randperm` függvény használatával sorsoljunk[1] 5 elemet az `S` cellából.

5. Feladat (20)**.** Egymásba ágyazott cilusok segítségével határozzuk meg, hogy 1729 miként bontható fel két különböző módon két természetes szám köbére: $1729 = a^3 + b^3$.

6. Feladat (20)**.** Egymásba ágyazott cilusok segítségével határozzuk meg, 100-nál nem nagyobb Pithagoraszi számhármasokat, vagyis azokat az $a, b, c \leq 100$ természetes számokat, amelyekre: $a^2 + b^2 = c^2$.

7. Feladat (120)**.** Készítsük el a következő kérdezz-felelek típusú játékot. A számítógép ellen játszunk. A számítógép „gondol” egy számra 1 és 100 között, a feladatunk kitalálni ezt a számot. Minden tippelésünk után a számítógép elárulja, hogy nagyobb vagy kisebb számra gondolt, mint amire tippeltünk. A játék végén, amikor is sikerült eltalálni a számot, írassuk ki, hogy hanyadik próbálkozásra találtuk ki a számot, és mennyi ideig tartott a játék.

8. Feladat (50)**.** Írjunk egy `iterate(f, x0)` függvényt, amelynek az első bemenete egy f függvény, második bemenete pedig az x_0 kezdeti feltétel. A függvény kimenetét pedig a következőképpen határozzuk meg: tekintsük az $x_{k+1} = f(x_k)$ iterációval keletkező $x_0, x_1, x_2, \ldots, x_n, \ldots$ sorozatot. Ha az iteráió során valamikor a szomszédos tagok relatíve közel kerülnek egymáshoz: $|x_k - x_{k-1}| \leq \varepsilon x_k$, akkor az iterációt megszakítjuk és a függvény kimenete az x_k szám lesz. Ha egyik szomszédos tag sem került egymáshoz elég közel az első 1000 lépés esetén, akkor a függvény kimenete legyen x_{1000}. Teszteljük a függvény működését az alábbi iterációkra:

a) $x_{k+1} = x_k + 1,\ x_0 = 0,$

b) $x_{k+1} = \cos(x_k),\ x_0 = 0,$

c) $x_{k+1} = \sqrt{1 + x_k},\ x_0 = 1,$

d) $x_{k+1} = -0.9x_k + 3.8,\ x_0 = 0,$

e) $x_{k+1} = (x_k + 2/x_k)/2,\ x_0 = 2,$

f) $x_{k+1} = x_k + 1/x_k,\ x_0 = 1.$

[1] Vagyis válasszunk véletlenszerűen 5 különböző elemet.

2. Egyszerűsített Black Jack játék

A játék és játékosok adait tartalmazó adatszerkezet a `BJGame` nevű struktúra. Ennek a struktúrának három mezője van. A `BJGame.Bet` egy szám, ami az aktuális tét. A `BJGame.CurrentPlayer` egy szöveg, ami az aktuális játékos nevét tartalmazza. A `BJGame.Players` pedig egy leképezés (Container.Map) típus, aminek a kulcsai a játékosok nevei, értékei pedig a játékosok pénze. Nem létező adat esetén hozzuk létre az alapértelmezett értékekkel: három játékos neve: *Arató Anna, Bodnár Bence, Cirmos Cecília;* mindenki 1000 egységnyi pénzt kap, az aktuális játékos legyen *Arató Anna,* és a tét legyen 10.

```
>> BJGame.Bet
ans =
    10
>> BJGame.CurrentPlayer
ans =
Arato Anna
>> BJGame.Players(BJGame.CurrentPlayer)
ans =
        1000
```

2. Programlista. Példa a `BJGame` struktúra használatára

setBet Írjunk egy `setBet` nevű függvényt, amely kiírja a lehetséges tételeket és felajánlja új tét választását. A függvénynek nincs bemenete, és egy kimenete van: az új tét értéke. Ellenőrizzük, hogy a megadott új tét megfelelő-e vagyis szám-e és megegyezik-e valamelyik lehetséges téttel. Ha bármi hiba történne a a függvény térjen vissza az alapértelmezett 10 értékkel. Használjuk a `try`, `catch` konstrukciót.

```
>> setBet
Uj tet: 1 | 5 | 10 | 20 | 50 | 100 | 500 | 1000
Valassz tetet: 500

ans =
   500
```

3. Programlista. Példa a `setBet` függvény használatára

setPlayer Írjunk egy `setPlayer` nevű függvényt, amelynek a bemenete és kimenete is `BJGame` struktúra. A függvény meghívásakor egy menü jelenjen meg, aminek első menüpontja: új játékos létrehozása, a többi menüpontja pedig a már meglévő játékosok nevei.

Új játékos bevitele esetén a felhasználótól kérjük be az új játékos nevét, majd ezt adjuk a `BJGame.Players` leképezéshez, az alapértelmezett 1000 egységnyi pénzzel. Az újonnan létrehozott játékost tegyük meg aktív játékossá a `BJGame.CurrentPlayer` értékének átírásával. Már meglévő játékost választva egyszerűen tegyük a kiválasztott játékost aktívvá. Ha bármi hiba történik, a függvény térjen vissza a bemenetében megadott (eredeti) struktúrával (használjuk a `try`, `catch` konstrukciót).

drawRanking Írjunk egy `drawRanking` függvényt, amelynek bemenete `BJGame` struktúra, kimenete nincs. A függvény meghívása után kiírja a bemenetében megadott struktúrában szereplő adatok alapján a rangsort.

Egy kártyalapot egy kételemű cellával reprezentálunk, ahol a cella elemei az alábbi cellákban található elemek egyike.

`suits = { 'Club', 'Diamond', 'Heart', 'Spade'},`
`ranks = {'2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A'}.`

Például: `{'Diamond', 'J'}` vagy `{'Heart', '3'}`. A paklit egy kártyalapokat tartalmazó cellával reprezentáljuk: `{{'Club', '2'}, {'Club', '3'},..., {'Spade', 'A'}}`.

getPack Írjunk egy `getPack` nevű függvényt, aminek nincs bemente és kimenete pedig egy kevert pakli.

Az osztó és a játékos lapjait a `dealer` és a `player` vektorokban tároljuk. A vektorok elemei egész számok, jelentésük pedig, hogy a pakliból hanyadik kártya a játékosé, illetve az osztóé. Például ha az osztó kapja az első lapot és a játékos a következő kettőt, akkor `dealer = [1]` és `player=[2, 3]`. Vagy például ha a játékos kap egy lapot, majd az osztó és aztán megint a játékos, akkor `dealer = [2]` és `player=[1, 3]`.

drawTable A `drawTable` függvény adott. Három bemenete van: a pakli, az osztó lapjai és a játékos lapjai, például: `drawTable(getPack, [1], [2, 3])`.

getValue A `getValue` függvény adott. Bemenete egy cella, amelynek elemei kártyalapok. Kimenete pedig a lapok összértéke (a Black Jack szabályai szerint). Amikor két lap értéke 21, annak kitüntetett szerepe van: Black Jack, ilyenkor a lapok értékét a -1 számmal jelöljük, egyébként pedig a lapok rendes értékével tér vissza. Például `getValue({{'Spade', 'A'},{'Heart', 'J'}})` értéke -1, `getValue({{'Spade', 'A'},{'Heart', '2'},{'Club', '10'}})` értéke 13.

evaluateRound Az `evaluateRound` függvény adott. Három bemenete és egy kimenete van. Az első két bemenete az osztó és játékos lapjainak értéke (tipikusan azok az értékek, amikkel a `getValue` függvény tér vissza); harmadik bemenete pedig a tét. A kimenete pedig az összeg, amit játékos kap (ez az érték lehet negatív, ekkor a játékos negatív értéket nyer, vagyis veszít). Például: `evaluateRound(-1, 23, 500)` értéke -500.

playRound Írjunk egy `playRound` nevű függvényt, amelynek nincs bemenete és két kimenete van. A függvény meghívásával az osztó és a játékos játszik egy kört, majd a kör végén a függvény visszatér az osztó és a játékos lapjainak értékeivel.

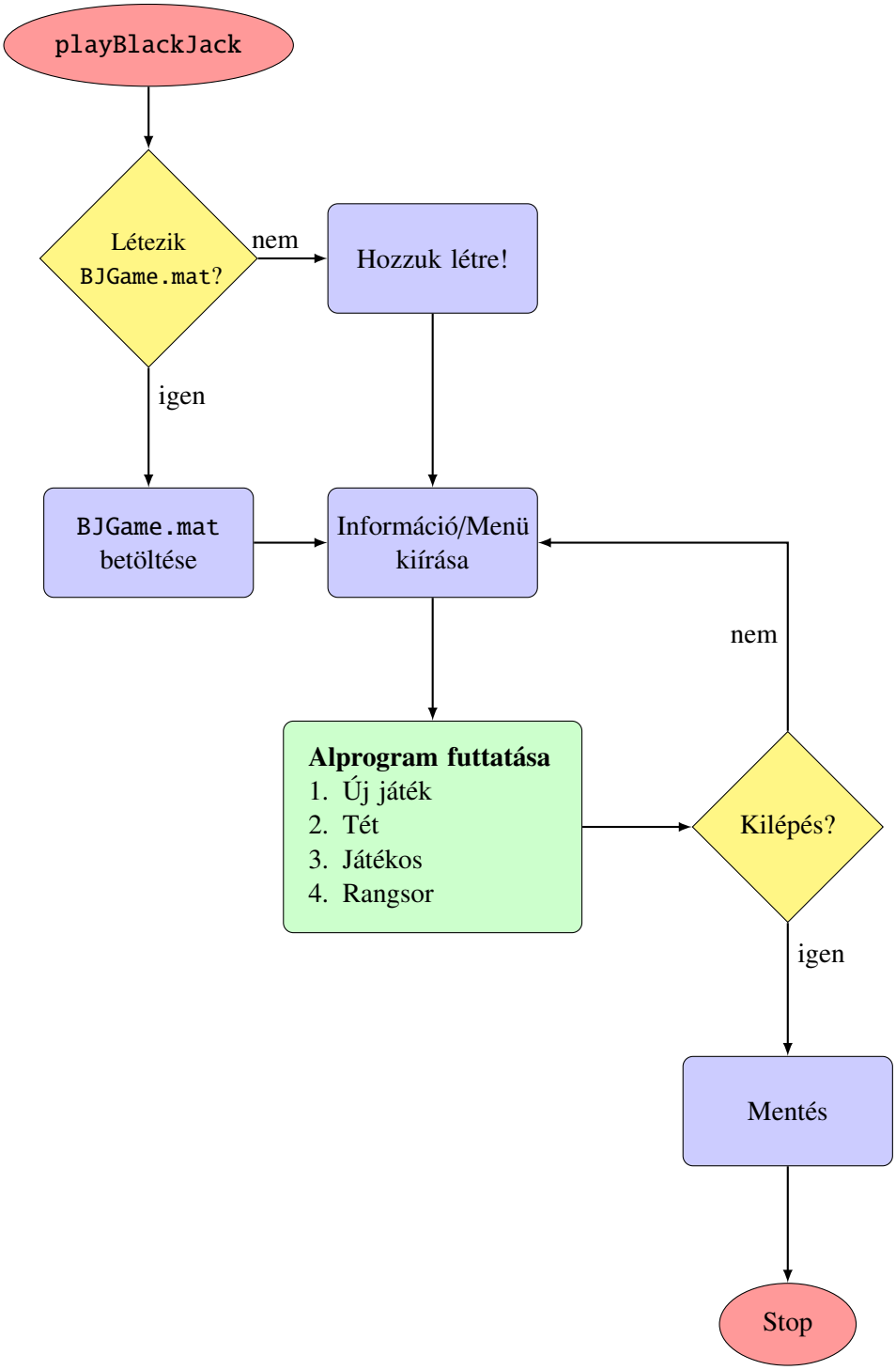
Kezdetben az osztó egyet, a játékos két lapot kap. Majd a játékos választ, hogy kér-e további lapot, vagy sem. Ha az új lap kérésével a lapok értéke meghaladja vagy eléri a 21-et a játék átkerül az osztóhoz, aki megpróbálja legyőzni a játékost a következő stratégiával: addig húz újabb lapokat, amíg nem nyer, vagy nem lesz túl sok a lapjainak értéke.

playBlackJack A játékot ezzel függvénnyel indíthatjuk el. A főprogram folyamatábráját láthatjuk az 1. ábrán. Az adatok betöltésére és mentésére használjuk a `load` és `save` parancsokat, illetve az állomány vagy változó létezésének eldöntésére az `exist` függvényt.

drawMenu Írjunk egy `drawMenu` függvényt, amelynek egy bemete van és nincs kimenete. A bemente a `BJGame` struktúra, és meghívásakor kiírja az aktuális játékos nevét, pénzét és a tétet; továbbá kirajzolja a játék főmenüjét.

```
*************************
*
* Arato Anna: 1000
* Tet: 10
*
*************************
*
* 1. Uj jatek
* 2. Tet
* 3. Jatekos
* 4. Ranglista
* 5. Kilepes
*
*************************
```

4. Programlista. A játék főmenüjének kirajzolása a `drawMenu` függvénnyel



1. ábra. A főprogram folyamatábrája