1. Számítsuk ki a következő függvény értékét egy adott x (valós szám) pontban!

$$f(x)=\begin{cases} 2x & \text{, ha } -2<x<0 \\ x\cdot x & \text{, ha } 0\le x<2 \\ 10 & \text{, ha } x>2 \end{cases}$$

Útmutatás ■ Vigyázat, vannak értékek, amelyekre nem definiált a függvény. Ezekben az esetekben írjunk ki megfelelő üzenetet!

2. Adott három szigorúan pozitív valós szám: *a*, *b*, *c*. Képezhetik-e ezek a számok egy háromszög oldalait? Ha igen, határozzuk meg és írjuk ki a háromszögbe írt, illetve a háromszög köré írt kör sugarát! Ha nem, írjunk ki megfelelő üzenetet.

3. Adva van egy nullától különböző természetes szám (*n*). Tervezzünk algoritmust, amely eldönti, hogy az adott szám prímszám-e vagy sem!

4. Számítsuk ki két természetes szám legnagyobb közös osztóját!

5. Legyen egy természetes szám. Döntsük el, hogy a szám palindrom szám-e vagy sem. Egy számot *palindromszám*nak (vagy tükörszámnak) hívunk, ha egyenlő a „fordított"-jával, vagyis azzal a számmal, amelyet a szám számjegyei fordított sorrendben alkotnak.

6. A méhek szaporodásáról ismert, hogy csak a méhkirálynő rak petéket és a megtermékenyített petékből kelnek ki a dolgozók(nők), a meg nem termékenyített petékből pedig a hímek. Írjon programot, amely adott *n*-re kiírja egy here családfájának *n*-generációs felmenőinek számát minden generáció esetén.

7.

a) Írj függvényt, mely egy gyümölcsneveket tartalmazó listát és egy betűt kap paraméterként. A függvény adja vissza az adott betűvel kezdődő gyümölcsök listáját. (Feltételezheted, hogy csak kisbetűk léteznek, és nincs üressztring a listában.)
b) Hívd meg a függvényed az [’citrom’, ’vérnarancs’, ’dinnye’, ’narancs’] listával és az ’n’ betűvel! Jelenítsd meg a visszakapott listát!
c) Írj egy *kiir* nevű void függvényt (eljárást), mely egy (gyümölcsnév, mennyiség) rendezett kettesekből (tuple) álló listát kap paraméterként. A függvény jelenítse meg azon gyümölcsök nevét, amelyből több mint 5 van (szigorúan több).
d) Teszteld le a függvényed az alábbi listára: gyumolcs = [(’citrom’, 10), (’vérnarancs’, 5), (’dinnye’, 4), (’narancs’, 6)]

3.

```
Algoritmus Prím_2(n,válasz):  { bemeneti adat: n, kimeneti adat: válasz }
  válasz ← igaz
  Minden osztó=2,[√n] végezd el:
    Ha maradék[n/osztó] = 0 akkor
      válasz ← hamis
    vége(ha)
  vége(minden)
Vége(algoritmus)
```

Megoldások::

4. **Megoldás** ■ Alkalmazzuk *Eukleidész algoritmusá*t. Kiszámítjuk a két szám osztási maradékát. Ha ez nem 0, ismételten kiszámítjuk az aktuális osztó és az aktuális maradék egész osztási maradékát. Az algoritmus véget ér, amikor az aktuális maradék értéke 0.

A 360 és a 225 legnagyobb közös osztójának meghatározása az euklideszi algoritmussal:

a b r

$$360 = 225 \cdot 1 + 135$$
$$225 = 135 \cdot 1 + 90$$
$$135 = 90 \cdot 1 + 45$$
$$90 = 45 \cdot 2 + 0$$

Tehát a legnagyobb közös osztó a 45.

```
Algoritmus Eukleidész(a,b,lnko):
  Ismételd                { bemeneti adatok: a, b, kimeneti adat: lnko }
    r ← maradék[a/b]      { kiszámítjuk az aktuális maradékot }
    a ← b                 { az osztandót felülírjuk az osztóval }
    b ← r                 { az osztót felülírjuk a maradékkal }
  ameddig r != 0          { amikor a maradék 0, véget ér az algoritmus }
  lnko ← a                { lnko egyenlő az utolsó osztó értékével }
Vége(algoritmus)
```

5. **Megoldás** ■ A számot számjegyekre bontjuk, és a bontással párhuzamosan felépítjük az új számot.

Az új szám generálását a *Horner-séma* néven ismert módszer segítségével végezzük (a ciklusban *újszám* ← *újszám* · *10* + *számjegy*). Mivel az algoritmus a számjegyeket úgy határozza meg, hogy ismételten osztja az eredeti számot 10-zel, az algoritmus végén az eredeti szám értéke 0. De nekünk szükségünk van az eredeti értékre, hogy összehasonlíthassuk a kapott új számmal. Ezért a beolvasott számról a feldolgozás előtt készítünk egy másolatot.

```
Algoritmus Palindrom(szám,válasz):
  másolat ← szám   { bemeneti adat: szám, kimeneti adat: válasz }
  újszám ← 0
  Amíg szám > 0 végezd el:
    számjegy ← maradék[szám/10]
    újszám ← újszám*10 + számjegy
    szám ← [szám/10]
  vége(amíg)
  válasz ← újszám = másolat { ha újszám = másolat, akkor válasz értéke igaz }
                            { ha újszám ≠ másolat, akkor válasz értéke hamis }
Vége(algoritmus)
```

6. A Fibonacci-számokat az alábbi rekurzív összefüggéssel definiáljuk:

$F_0 = 0$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$, ahol $i \geq 2$ (*)

Minden Fibonacci-szám tehát a megelőző kettőnek az összege (kivétel az első kettő, amelyek adottak).

A Fibonacci-számok sorozata: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

A megoldás lépéseinek leírása ■ A megoldásban a (*) összefüggést alkalmazzuk, amelyből kiderül, hogy az első *n* Fibonacci-szám generálásához elegendő három változó: *a*, *b* és *c*.

A számsor egy új tagját (*c*) úgy állíthatjuk elő, hogy a már kiszámolt elemeket egyszerűen balra „toljuk” egy pozícióval. Ily módon rendre megkapjuk *a* és *b* új értékét, majd ezek ismeretében kiszámíthatjuk *c* új értékét. A fenti lépéseket addig ismételjük, amíg a feladat által kért számú elemet elő nem állítottuk!

```
Algoritmus Fibonacci(n):
  a ← 0
  b ← 1
  Ha n = 1 akkor
    Ki: a
  különben
    Ha n = 2 akkor
      Ki: a, b
    különben
      c ← a + b
      Ki: a, b, c
      k ← 3
      Amíg k < n végezd el:
        a ← b
        b ← c
        c ← a + b
        Ki: c
        k ← k + 1
      vége(amíg)
    vége(ha)
  vége(ha)
Vége(algoritmus)
```

A fenti algoritmus generálja és kiírja egyenként a Fibonacci-sorozat első *n* elemét, amelyeket három változó segítségével állapít meg. Lássunk egy olyan algoritmust, amely mindössze két segédváltozót használ ahhoz, hogy kiszámítsa és kiírja a Fibonacci-sorozat *n* elemét.

```
Algoritmus Fibonacci(n):
  a ← 1
  b ← 0
  Minden k=1,n végezd el:
    Ki: b
```

```
    b ← a + b
    a ← b - a
  vége(minden)
Vége(algoritmus)
```