Készítsen ***avg*** nevű függvényt, ami egy paraméterül kapott tömb elemeinek az átlagát veszi úgy, hogy a számítás során nem veszi figyelembe a tömb legnagyobb és a legkisebb elemét. Feltételezhető, hogy a tömbben egy érték csak egyszer szerepel, így nem kell azzal az esettel foglalkozni, ha több legkisebb vagy legnagyobb érték található, illetve azzal sem, ha a legkisebb és legnagyobb érték ugyanaz.

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
double avg(double lista[], int hossz) {}
```

Ellenőrzés

Question author's solution:

```
double avg(double lista[], int hossz) {
  int min = 0;
  int max = 0;
  double avg = 0.0;
  for(int i = 1; i < hossz; i++) {
    if(lista[i] < lista[min]) {
      min = i;
    }
    if(lista[i] > lista[max]) {
      max = i;
    }
  }
  for(int i = 0; i < hossz; i++) {
    if(i != min && i != max) {
      avg += lista[i];
    }
  }
  avg /= hossz - 2;
  return avg;
}
```

Készítsen ***diff*** nevű függvényt, ami egy tömbben tárolt, időben változó digitális jelsorozat deriváltják közelíti véges differenciával. A tömb i-edik indexű eleme megadja, hogy a digitális jel milyen értéket vett fel az i-edik időpillanatban.

A véges differenciát az alábbi módon határozzuk meg: A kimeneti oldalon az i-edik időpillanathoz tartozó közelítő derivált érték legyen a bemeneti jel i-edik időpillanatban vett értékének és az (i-1)-edik időpillanatban vett értékének a különbsége. Ahol az (i-1)-edik időpillanat nem értelmezhető, ott a kimeneti oldalon 0 érték szerepeljen.

Answer: (penalty regime: 0 %)

Reset answer

```
1 double* diff(double jel[], int hossz) {}
```

Ellenőrzés

Question author's solution:

```
double* diff(double jel[], int hossz) {
  double* kimenet = (double*)malloc(hossz * sizeof(double));
  kimenet[0] = 0;
  for(int i = 1; i < hossz; i++) {
    kimenet[i] = jel[i] - jel[i-1];
  }
  return kimenet;
}
```

Készítsen ***darabszam*** függvényt, ami egy szivarok adatait tartalmazó listán megszámolja a Churchill és Toro méretű szivarokat.

A szivarok az alábbi struktúrában kerülnek tárolásra (a struktúrát nem kell létrehoznia):

```
struct szivar {
  double hossz;
  int atmero;
  szivar* kov;
};
```

Minden szivar esetén adott a hossza hüvelykben és a gyűrű mérete (átmérője).

A ***darabszam*** függvény bemenetként vár egy szivarokból álló láncolt listát, amin a megfelelő méretű szivarokat kell megszámolni. A darabszámokat a bemenetként kapott memóriaterületekre kell írni. Egy szivart akkor tekintünk Churchill méretűnek, ha hossza minimum 6,75 és maximum 8, valamint gyűrű mérete minimum 46 és maximum 48. Toro méretről akkor beszélünk, ha a szivar hossza minimum 5,5 és maximum 6,5, illetve a gyűrűje minimum 48 és maximum 54 méretű.

Answer: (penalty regime: 0 %)

Reset answer

```
1 void darabszam(szivar* szivarok, int* churchill, int* toro) {}
```

Question author's solution:

```
void darabszam(szivar* szivarok, int* churchill, int* toro) {
  *churchill = 0;
  *toro = 0;
  while(szivarok != NULL) {
    if(szivarok->hossz >= 5.5 && szivarok->hossz <= 6.5 and szivarok->atmero >= 48 and szivarok->atmero <= 54) {
      (*toro)++;
    }
    if(szivarok->hossz >= 6.75 && szivarok->hossz <= 8.0 and szivarok->atmero >= 46 and szivarok->atmero <= 48) {
      (*churchill)++;
    }
    szivarok = szivarok->kov;
  }
}
```

Készítsen ***akkord*** nevű függvényt, ami megadja egy zenei hármashangzat hangjait az alaphang és a hangnem alapján.

A lehetséges zenei hangok sorrendben a következők (a programban nagybetűvel tárolva): C, C#, D, D#, E, F, F#, G, G#, A, B, H. A hangok folytonosan követik egymást, tehát a "H" hang után ismét "C" következik, majd "C#" és így tovább.

A függvény első paramétere az alaphang (nagybetűvel), majd a hangnem, ami "moll" vagy "dur" lehet (kisbetűvel, ékezet nélkül).

Ha a hangnem moll, akkor a hármashangzat sorrendben a következő hangokból áll: alaphang, alaphang után következő 3. hang, és az alaphang után következő 7. hang.

Ha a hangnem dúr, akkor a hármashangzat sorrendben a következő hangokból áll: alaphang, alaphang után következő 4. hang, és az alaphang után következő 7. hang.

A kimeneti string-ben egymástól egy-egy szóközzel elválasztva jelenjenek meg a hangok nevei.

Például:

Az alaphang C és a hangnem dúr, akkor a kimenet: "C E G".
Az alaphang A és a hangnem moll, akkor a kimenet: "A C E".

Feltételezheti, hogy a függvény helyesen kerül paraméterezésre, így a paraméterek helyességének vizsgálatával nem kell foglalkoznia.

Answer: (penalty regime: 0 %)

Reset answer

```
string akkord(string alap, string hangnem) {}
```

Question author's solution:

```
string akkord(string alap, string hangnem) {
  string hangok[] = {"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "H", "B"};
  int i = 0;
  while(hangok[i] != alap) {
    i++;
  }
  string hangsor = hangok[i] + " ";
  if(hangnem == "moll") {
    hangsor = hangsor + hangok[(i + 3) % 12] + " " + hangok[(i + 7) % 12];
  } else if(hangnem == "dur") {
    hangsor = hangsor + hangok[(i + 4) % 12] + " " + hangok[(i + 7) % 12];
  }
  return hangsor;
}
```

Készítsen ***v2m*** névvel függvényt, ami vektorból mátrixot készít. A függvény paraméterül várja a vektort, annak elemszámát és az előállítandó mátrix sorainak, oszlopainak számát. A függvényt úgy kell megvalósítani, hogy az a vektor elemeivel, a sorban az elsőtől az utolsóig haladva, feltölti a mátrixot. Először a mátrix első sorát töltse fel az első oszloptól az utolsóig haladva, majd a második sor következik, és így tovább a mátrix utolsó soráig. Feltételezheti, hogy a függvény mindig megfelelően kerül paraméterezésre, így nem kell azzal az esettel foglalkozni, ha a mátrix oszlop és sorszámának szorzata nem egyezik a vektor hosszával. A mátrixot tároló tömböt úgy alakítsa ki, hogy az első index a sort, a második az oszlopot határozza meg.

Answer: (penalty regime: 0 %)

Reset answer

```
1 double** v2m(double* v, int hossz, int sor, int oszlop) {}
```

Ellenőrzés

Question author's solution:

```
double** v2m(double* v, int hossz, int sor, int oszlop) {
  double** m = (double**)malloc(sor * sizeof(double*));
        for (int i = 0; i < sor; i++)
        {
                m[i] = (double*)malloc(oszlop * sizeof(double));
        }
  for(int i = 0; i < sor; i++) {
    for(int j = 0; j < oszlop; j++) {
      m[i][j] = v[i * oszlop + j];
    }
  }
  return m;
}
```