Vizsga és ZH feladatok gyűjteménye Programozás alapjai I. tárgyból informatikusoknak

Szerkesztő: Pohl László

BME, Elektronikus Eszközök Tanszéke
Budapest 2008

Tartalomjegyzék

Előszó

A következő oldalakon az elmúlt évek összes fellelt nagy ZH és vizsgafeladata megtalálható, a beugró feladatok mindegyikéhez készült mintamegoldás is, a maximumrész feladatoknak csak egy részéhez adunk megoldást. Javasoljuk, hogy oldjanak meg minél több feladatot önállóan, legalább a felét számítógépbe beírva, a mintamegoldásokhoz csak végső esetben forduljanak. Eredményes felkészülést kívánunk!

A feladatok és megoldások kidolgozásában közreműködött többek között Kohári Zsolt, Nagy Gergely, Pohl László, Poppe András, Szente-Varga Domonkos, Vitéz András és mások.

Nagy ZH beugró témakörei:

1. Adattípusok, operátorok, ciklusszervezés. Jellegzetes feladatok a számelméleti példák: osztó, prím stb., továbbá a bitszintű műveletek ismeretét igénylő feladatok: maszkolás, biteltolás, pl.: bitszámlálás, paritásvizsgálat, egyes bitek kimaszkolása.
2. Állapotgép, egy- és többdimenziós tömbök, dinamikus tömbök, pointerek, függvényírás. Bármilyen egyszerűbb szövegszűréses állapotgépes feladat előfordulhat, állapotgráfot vagy állapottáblát rajzolni tudni kell. Dinamikus tömb felszabadítását tudni kell. Tudni kell függvényt írni, függvényértéket visszaadni paramétersoron (pointerrel) vagy függvényértékként (returnnel). Előfordulhat olyan feladat, hogy függvény tömböt kap, és a tömb elemei közül ki kell válogatni bizonyos tulajdonságúakat (páros számokat, átlag alattiakat, egy paraméterként adott értéknél nagyobbakat stb.) Pointert visszaadó függvények.
3. Stringek, rendezés, keresés: könyvtári sztringfüggvények (strlen, strcat, strcmp, strcpy stb.) saját megírása; sztringek összefésülése betűnként, szavanként, mondatonként stb.; bizonyos tulajdonságú sztringrészek törlése, pl. szóközök, kisbetűk, nagybetűk, számok. Keresés sztringen belül, részsztring előállítása stb. Tudni kell rendezni, ismerni kell legalább egy rendező algoritmust. (Fenn lehet a kézzel írott puskán, de tudni kell használni).

Vizsga beugró témakörei:

1. Adattípusok, operátorok, ciklusszervezés, állapotgépek, egy- és többdimenziós tömbök. Jellegzetes feladatok a számelméleti példák: osztó, prím stb., továbbá a bitszintű műveletek ismeretét igénylő feladatok: maszkolás, biteltolás, pl.: bitszámlálás, paritásvizsgálat, egyes bitek kimaszkolása. Bármilyen egyszerűbb szövegszűréses állapotgépes feladat előfordulhat, állapotgráfot vagy állapottáblát rajzolni tudni kell. A tömb elemei közül ki kell válogatni bizonyos tulajdonságúakat (páros számokat, átlag alattiakat, egy paraméterként adott értéknél nagyobbakat stb.)
2. Dinamikus tömbök, pointerek, függvényírás, stringek, rendezés, keresés, hashing. Dinamikus tömb felszabadítását tudni kell. Tudni kell függvényt írni, függvényértéket visszaadni paramétersoron (pointerrel) vagy függvényértékként (returnnel). Függvény tömböt kap, pointert visszaadó függvények. Könyvtári sztringfüggvények (strlen, strcat, strcmp, strcpy stb.) saját megírása; sztringek összefésülése betűnként, szavanként, mondatonként stb.; bizonyos tulajdonságú sztringrészek törlése, pl. szóközök, kisbetűk, nagybetűk, számok. Keresés sztringen belül, részsztring előállítása stb. Tudni kell rendezni, ismerni kell legalább a közvetlen kiválasztásos, beszúrásos, buborék és

gyorsrendező algoritmust. (Fenn lehet a kézzel írott puskán, de tudni kell használni).

3. Függvénypointerek, rekurzió, láncolt és fésűs listák, bináris és többágú fák, bináris és szöveges fájlok. Tudni kell függvénypointert függvényparaméterként vagy típusként definiálni, átadott függvényt meghívni, függvénypointer tömböt kezelni. Láncolt egy- és kétirányú listát építeni és lebontani, elemt beszúrni és törölni, strázsával ellátott listát kezelni, listát bejárni, fát bontani és építeni, fát bejárni, függetlenül attól, hogy hány ága van. Bináris fájlokat megnyitni/bezárni/írni/olvasni, szöveges fájlokat megnyitni/bezárni/írni/ olvasni.

Halálfejes hibák

Az alábbi hibák nagyon súlyosak, a javítótól függ, hogy az egész feladatra 0 pontot ad-e, vagy megkegyelmez, és „csak” a pontok felét vonja le:

1. Inicializálatlan változó értékének használata

Pl.:
Rossz:

```
int a;
while(a<=0){
    printf("Kérek egy pozitív egész számot!");
    scanf("%d",&a);
}
```

Jó:

```
int a;
do{
    printf("Kérek egy pozitív egész számot!");
    scanf("%d",&a);
}while(a<=0);
```

2. Inicializálatlan pointer használata

Pl.:

```
char * a;
scanf("%s",a);
```

A pointernek olyan adatterületre kell mutatnia, ahová a művelet eredménye elhelyezhető. Ez lehet pl. tömb vagy dinamikus tömb.

3. A feladat által nem igényelt beolvasás vagy kiírás függvényben

Pl.: Készítsen C függvényt, amely kiszámítja és visszaadja két egész szám összegét:

Rossz:

```
void osszeg(int a,int b){
    printf("Kérek két számot:");
    scanf("%d%d",&a,&b);
    printf("a+b=",a+b);
}
```

Jó:

```
int osszeg(int a,int b){
    return a+b;
}
```

4. Ismeretlen méretű tömb létrehozása és használata

Pl.:

```
int t[],i=0;
while(scanf("%d",&t[i])==1)i++;
```

A tömb méretét konstanssal adjuk meg, a fenti megadás nem definiál egy tetszőleges méretű tömböt, C-ben nincs tetszőleges méretű tömb. Ha nem tudjuk, hány adatot kell eltárolnunk, akkor egyéb adatszerkezetet használjunk, például listát vagy fát. Megjegyzendő, hogy a beugróban szereplő feladatoknál nagyon ritka az, amikor ismeretlen mennyiségű adat érkezik, és azokat el is kell tárolni. Ilyen általában csak a maximumrészben fordul elő. Beugrónál tehát nagyon alaposan gondoljuk meg, hogy megoldható-e a feladat az adatok eltárolása nélkül!

5. Tömbméret megadása paraméterben, változóként

Pl.:

```
void fuggveny(int t[n]){
    int i;
    for(i=0;i<n;i++){....}
}
```

Helyesen:

```
void fuggveny(int t[],int n){
    int i;
    for(i=0;i<n;i++){....}
}
```

6. Függvénydefiníció függvényben

Pl.:

```
int main(){
    int a, b, c;
     printf("Jó reggelt!");
     int szr(int a, int b){
         ....
     }
     ....
}
```

7. Ha nem tudja, hogy az értékadás mindig jobbról balra történik

Pl.:

```
  int a,b;
  scanf("%d",&a);
  a=b;
```

Helyesen:

```
  int a,b;
  scanf("%d",&a);
  b=a;
```

Beugró feladatok

1. Adattípusok, operátorok, ciklusszervezés

1.1 Írjon olyan programot C nyelven, mely kiszámolja és kiírja az e^{3x}-4sin(2x/29)+4x^3-7x-8.0=0 egyenlet [0,1] intervallumba eső gyökét.

1.2 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy egész számról eldönti, hogy tökéletes szám-e! Az a tökéletes szám, amely egyenlő a nála kisebb osztói összegével. Pl. `6 = 1+2+3` `28 = 1+2+4+7+14` Bemenő paraméter: a vizsgálandó egész. Visszatérési érték: a vizsgálat eredménye, logikai érték. Tökéletes szám esetében ez IGAZ legyen.

1.3 Írjon C programot, amely bekér három pozitív számot, és eldönti, hogy lehetnek-e egy háromszög oldalai!

1.4 Írjon C programot, amely eldönti egy bekért pozitív egész számról, hogy prím-e!

1.5 Írjon C programot, amely kiírja a felhasználótól bekért pozitív egész szám prímtényezős felbontását.

1.6 Írjon olyan C ***függvényt***, amely egy paraméterként átadott `n` pozitív egész számról eldönti, hogy prímszám-e. Ha igen, akkor a függvény visszatérési értéke legyen logikai IGAZ, ha nem prímszám, akkor pedig a visszatérési érték legyen logikai HAMIS! (A függvény nem kommunikálhat a felhasználóval, a külvilággal való adatforgalmat a paraméterlistáján és a visszatérési értékén keresztül bonyolítja le.)

1.7 Írjon egy olyan C nyelvű teljes ***programot***, amely beolvas egy n pozitív egész számot (`n <= 25`) és kiírja a hozzá tartozó ún. latin négyzetet a képernyőre. Ez az `1` és `n` közötti egész számok permutációinak egy adott sorozata. Például `n==5` esetén:

```
1 2 3 4 5
2 3 4 5 1
3 4 5 1 2
4 5 1 2 3
5 1 2 3 4
```

1.8 Készítsen egy olyan szabványos ANSI C függvényt, amely két, paraméterként átadott, 1-nél nagyobb természetes számról eldönti, hogy azok relatív prímek-e! Két szám relatív prím, ha egy közös (valódi) osztójuk sincs. A döntés eredményét a függvény visszatérési értéke jelezze: ha relatív prímek, a visszatérési érték "igaz" legyen!

1.9 Írjon szabványos ANSI C függvényt, mely paraméterként kap két egész számot, és visszaadja a számok legkisebb közös többszörösét.

1.10 Készítsen egy olyan szabványos ANSI C függvényt, amely visszatérési értékül egy `double` szám `n`-edik byte-ja `m`-edik bitjét szolgáltatja. Bemenő paraméterek: maga a `double` típusú valós szám, valamint `n` és `m` értéke, amelyek 1 és 8 közötti egészek.

1.11 Készítsen szabványos ANSI C ***függvényt***, mely paraméterül kap egy nemnegatív egész értéket és egy 2..10 közé eső számot, és a szabványos kimenetre kiírja a kapott egész értéket a második paraméterként kapott számrendszerben!

1.12 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely a bementként kapott előjeles egész szám (`int`) bitmintáját kiírja a szabványos kimenetre. Pontosan azt a teljes `int`-ben tárolt bitsorozatot kell bináris szemjegyekkel megjeleníteni, amely a bemeneti adat konkrét értékét reprezentálja. Például ha az `int`-et 16 biten ábrázolják, akkor minden esetben 16 bináris számjegyet kell megfelelő sorrendben megjeleníteni. Az `int` típus konkrét mérete tekintetében semmiféle feltételezéssel nem élhet.

1.13 Írjon egy olyan C nyelvű teljes **programot**, amely meghatározza a szabványos bemenetről folyamatosan érkező térbeli pontok (valós számhármasok) súlypontját, és kiválasztja az origóhoz legközelebbi pontot is. A pontok derékszögű koordinátarendszerben vett *x*, *y* és *z* adataikkal vannak megadva. A pontsorozat végét az jelzi, hogy a beolvasást végző `scanf` függvény visszatérési értéke nem 3. A szabványos bemenetről a program csak számhármasokat kap, de előre nem ismert, hogy hány darabot. A két kívánt pont koordinátáit írja a program a szabványos kimenetre. (A súlypont valamely koordinátája az egyes pontok ugyanazon koordinátáinak az átlaga.)

1.14 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy pozitív egész számot, és logikailag IGAZ-at vagy HAMIS-at jelentő egész értékkel tér vissza. A visszatérési érték legyen IGAZ, ha a paraméterként kapott számnak egyen és önmagán kívül még pontosan két pozitív egész osztója van, és legyen HAMIS minden más esetben! (Pl. 8-nak osztói 1, 2, 4, 8, azaz erre IGAZ-at ad a függvény, de 10-re, melynek 1, 5, 10 az osztói, HAMIS-at.)

1.15 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy **előjel nélküli egész** számot (n), és **visszaadja** az n. prímet! (Feltételezheti, hogy az n. prím ábrázolható előjel nélküli egésszel.)
Pl.: n=5 esetén visszaadott érték 11, mert az első öt prím: 2, 3, 5, 7, 11

1.16 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy **előjel nélküli egész** számot (n), és **visszaadja** azt a legkisebb olyan egész számot, amely kettő egész hatványa, és n nem nagyobb nála.
Például ha n=7 => visszaad 8-at, n=128 => visszaad 128-at, n=513 => visszaad 1024-et.

1.17 Írjon olyan szabványos ANSI C ***programot***, amely kiírja az összes olyan háromjegyű számot, amelynek a három számjegye eltér egymástól! (pl. 142 jó, 141 nem jó, 012 nem jó)

1.18 Készítsen olyan szabványos ANSI C ***programot***, amely bekér a felhasználótól két pozitív egész számot, ellenőrzi, hogy valóban pozitív számokat kapott-e, és ha nem, hibaüzenetet ad és kilép. Ha pozitív számokat kapott, írja ki a két szám között található összes prímet olyan formában, hogy a kiírt számok mindegyike 10 helyet foglaljon a képernyőn! Ha a szám ennél rövidebb, szóközökkel töltse fel a szám előtti üres helyeket!

1.19 Készítsen olyan szabványos ANSI C ***programot***, amely bekér a felhasználótól egy olyan nemnegatív egész számot, mely kettes számrendszerben ábrázolva legfeljebb 31 bites! Ellenőrizze, hogy valóban ilyen számot adott-e meg a felhasználó, és ha nem, lépjen ki a programból hibaüzenettel! Ezután számolja meg, hogy hány 1-es bit van a számban! Szorozza meg a beolvasott számot kettővel, és ha az 1-es bitek száma páros volt, adjon hozzá 1-et a kapott értékhez! Írja ki a végeredményt!

1.20 Készítsen egy olyan szabványos ANSI ***C függvényt***, amely paraméterként két pozitív egészet kap és visszatérési értékével megadja azt, hogy az ezen két pozitív egész által meghatározott zárt intervallumban hány prímszám van.
A függvény csak a paraméterein és a visszatérési értékén keresztül kommunikálhat a külvilággal, a szabványos bemenetet és kimenetet nem kezelheti. Törekedjen arra, hogy kódja átlátható legyen. Ha kell, definiáljon további programszegmenseket.

1.21 Készítsen olyan szabványos ANSI C ***programot***, amely bekér a felhasználótól egy pozitív egész értéket, megfordítja a szám kettes számrendszerbeli alakjának bitmintáját, és kiírja az így kapott egész számot! Csak az eredeti szám értékes bitjeit vegye figyelembe a fordításnál! Pl.: ha a felhasználó 6-ot ad meg, akkor ennek 110 a kettes számrendszerbeli alakja, megfordítva 011, azaz 3. Pl. 333 esetén a bitminta 101001101, ez megfordítva 101100101, azaz 357. A megoldáshoz nem használhat tömböt!

1.22 Írjon egy teljes C ***programot***, amely megkeresi és a szabványos kimenetre írja azt a legnagyobb háromjegyű számot, amelynek számjegyösszege megegyezik a prímtényezőinek az összegével. (pl. 378=2·3·3·3·7, ezek összege 18, ami egyenlő a jegyek összegével is.)

1.23 Készítsen egy olyan szabványos ANSI C ***programot***, amely a szabványos bementről beolvasott hétjegyű budapesti telefonszámot binárisan kódolt decimális (BCD) alakúra konvertálja, melyet egy előjel nélküli egészben tárol! Feltételezzük, hogy az előjel nélküli egész legalább 32 bites. Pl. a bemenő szám: 4631111, ennek BCD alakja minden tízes számrendszerbeli számjegyet 4 biten tárol, azaz 4,6,3,1,1,1,1 számjegyek binárisan: 0100,0110,0011,0001,0001,0001,0001, azaz a BCD szám bináris alakja 01000110001100010001000010001, ami decimális alakban kiírva 73601297. Ezt a számot kell előállítani és kiírni!

2. Állapotgépek

2.1 Írjon C programot, amely a standard inputról érkező C forrásszöveget úgy másolja át a standard outputra, hogy a string konstansokból a ++ (pluszplusz) karaktersorozatokat elhagyja. Ügyeljen arra, hogy a sztringben idézőjelet \" (backslash idézőjel) sorozattal lehet elhelyezni.

2.2 Készítsen olyan szabványos ANSI C ***programot***, amely a szabványos bemenetről (billentyűzet) olvas be karaktereket sorvége jelig ('\n')! A beolvasott karakterek kizárólag 'p', 's' vagy 'z' betűk lehetnek. Ha nem ilyen érkezik, adjon hibaüzenetet és lépjen ki a programból! Ha több egyforma betű érkezik egymás után, csak egyet írjon ki a szabványos kimenetre (képernyő)! Pl.: be: pzppzsssssspsz, ki: pzpzspsz. A megoldáshoz használjon állapotgépet, rajzoljon állapotgráfot vagy állapottáblát!

3. Tömbös program

3.1 Írjon C programot, amely statisztikát készít a standard bemeneten olvasott számjegykarakterekről! A program számjegyenként (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) írja ki, hogy melyik számjegy hányszor fordult elő. A karaktereket a soremelés karakterig olvassa! Nem csak számjegyek fordulhatnak elő, a többi karaktert hagyja figyelmen kívül!

4. Függvényírás

4.1 Írjon egy olyan C függvényt, amely paraméterként kapja egy henger átmérőjének (d) és magasságának (h) az értékét és ebből kiszámítja a henger felszínét és ezt az értéket a fel paraméter által megcímzett memóriaterületen szolgáltatja. Az elkészítendő függvény prototípusa:

```
void felszin(double d, double h, double *fel);
```

Írjon egy olyan C nyelvű teljes programot, amely a felhasználótól bekért adattal aktiválja fenti függvényt és kiírja az eredményt a képernyőre. Az aktiváláshoz használja az alábbi deklarációkat:

```
void main()
{
  double d, h, felsz;
 …
}
```

4.2 Írjon olyan szabványos ANSI C függvényt, amely a szabványos bemenetről a file végéig érkező valós számok közül az utolsó 8 szám mértani középértékét szolgáltatja visszatérési értékként! A szabványos bemenetről a program csak számokat kap, de előre nem ismert, hogy hány darabot. Ha 8 számnál kevesebbet olvasna be a függvény, a hiányzó számokat tekintse 1-nek. Ügyeljen arra, hogy a függvény a lehető legegyszerűbb legyen, és ne végezzen felesleges adatmozgatásokat! Segítség: *n* db szám mértani közepét a következő képlettel lehet kiszámolni:

$$mean = \sqrt[n]{a_1 \cdot a_2 \cdot \ldots \cdot a_n}$$

4.3 Írjon C függvényt, amelyik megállapítja, hogy a paraméterként kapott n elemű double elemű tömb növekvően rendezett-e. A függvény paraméterként vegye át az elemek számát (n) is.

4.4 Írjon C függvényt, amely a szabványos bemenetről olvas be egy szöveget (fájlvége jelig), és megszámolja a szövegben lévő * és # karaktereket. A számolás végeztével az eredményt a függvény paraméterlistáján adja vissza!

4.5 Írjon olyan szabványos ANSI C függvényt, amely paraméterként kap egy double típusú számot (d), és egy egész értéket (n). Adja vissza a paraméterlistáján egész szám formájában a szám egészrészét, valamint a törtrészének n számjegyét egész számként. Például: bemenő adat: d=3.14, n=4; visszaadott: 3 és 1400.

4.6 Készítsen függvényt, amely paraméterként kapja egy ellipszis két átmérőjét (a,b), és visszaadja azt a terület, ami az ellipszis köré rajzolt téglalapból marad az után, hogy kivonjuk belőle az ellipszis területét. $T_{ellipszi}=ab\pi/4$, $T_{téglalap}=ab$.

```
double terulet(double a, double b);
```

4.7 Egy függvény prototípusa a következő:

```
int fordit(int ertek);
```

A függvény visszatérési értékként azt az egész számot szolgáltatja, amely a paraméterként kapott egész érték számjegyeit fordított sorrendben tartalmazza, és előjelét ellenkezőjére fordítja. Például a `printf("%d, %d",fordit(-1234),fordit(567));` kimenete: `4321, -765`. Valósítsa meg a függvényt szabványos C nyelven! Az int típus értéktartományára vonatkozóan semmiféle feltételezéssel nem élhet, csak abban lehet biztos, hogy a megfordított szám is belefér az int típusba!

4.8 Készítsen egy olyan szabványos ANSI C függvényt, amely úgy írja ki egy paraméterként átadott `unsigned int` típusú egész szám decimális értékét a szabványos kimenetre, hogy 3 számjegyenként tagolja azt egy-egy szóközzel. Pl. `1234567` kiírása: `1 234 567`. A feladat megoldása során az ábrázolható legnagyobb egész szám konkrét értékére vonatkozólag semmiféle feltételezéssel nem élhet, azt semmilyen előre definiált konstans értékből nem állapíthatja meg!

4.9 Készítsen egy ANSI C **függvényt**, mely paraméterként kap egy egész számokból álló tömböt, és egy egész számot, ami a tömb elemszáma. A függvény visszatérési értéke legyen 1, ha a tömb elemei relatív prímek, 0, ha nem azok. Akkor relatív prímek, ha a számok egyikének sincs bármely másikkal közös osztója. Törekedjen gyors megoldásra!

4.10 Írjon függvényt, amely paraméterként kap egy egészekből álló, kétdimenziós, négyzetes tömböt (egy n×n-es mátrixot) valamint annak méretét, és a visszatér egy logikai értékkel, amely azt mondja meg, hogy a mátrix főátlójában minden elem 0-e.

4.11 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy 1241×1956 elemű, és egy 1956×1241 elemű, **double** típusú elemeket tartalmazó kétdimenziós tömböt, és az első tömb transzponáltját a másodikba helyezi. A transzponálás a sorok és oszlopok felcserélését jelenti, azaz ami az i. sor j. eleme volt, az a j. sor i. eleme lesz. A hiányosan megadott programra maximum a pontszám harmada adható!

Pl.: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

4.12 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy **előjel nélküli egész** számot, és 1-et ad vissza, ha a megadott szám eleme a Fibonacci sorozatnak, 0-t, ha nem. A Fibonacci sorozat első két eleme 1, az n. eleme pedig az n-1. és n-2. elemének összege. A sorozat első néhány eleme a következő: 1, 1, 2, 3, 5, 8, 13, 21, 34,… Tehát a sorozatnak eleme pl. a 13, de nem eleme a 15.

4.13 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap két, egész értékeket tároló **egydimenziós** tömböt (`t1,t2`)! A `t1` tömbben egy kétdimenziós mátrix található sorfolytonosan. A függvény paraméterként átveszi a kétdimenziós mátrix `x,y` méretét (a kettő szorzata adja a tömb méretét). Ugyancsak paraméterként kap egy `x0`, `y0` és egy `dx`, `dy` egész típusú értékpárt, és `t2`-be belemásolja a paraméterként kapott mátrix `x0`, `y0` koordinátán kezdődő `dx`, `dy` méretű almátrixát (az indexelés 0,0-val kezdődik).

Pl. be: x=5, y=4, x0=2, y0=1, dx=3, dy=2, t= $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{bmatrix}$

azaz t[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

ki: t2[]= $\begin{bmatrix} 8 & 9 & 10 \\ 13 & 14 & 15 \end{bmatrix}$ azaz {8,9,10,13,14,15}

4.14 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kapja egy gömb valós értékű sugarát (r), kiszámítja a gömb felületét és térfogatát, a felületet visszatérési értékként adja, a térfogatot pedig **paraméterként adja vissza**! A gömb felülete $4\times pi\times r^2$, térfogata $\frac{4}{3}\times pi\times r^3$. Ne használja a pow függvényt!

5. Dinamikus tömb

5.1 Írjon olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy double értékeket tartalmazó tömböt és a tömb elemszámát, meghatározza a tömbben található értékek átlagát, megszámolja, hogy hány átlag alatti érték van a tömbben, létrehoz egy megfelelő méretű dinamikus tömböt és ebbe átmásolja az átlag alatti értékeket. Adja vissza paramétersoron a dinamikus tömb méretét, valamint függvényértékként a dinamikus tömb címét!

6. Sztring

6.1 Írjon függvényt, mely két sztringet kap a felhasználótól, és a második sztringet átmásolja az elsőbe. A függvény prototípusa legyen ez: `void Copy(char cel[],char forras[]);`

6.2 Készítsen osszefuz nevű függvényt, mely paraméterként kap két stringet, és az első végéhez fűzi a másodikat! A függvény más függvényt nem hívhat!

6.3 Írjon C függvényt, amelyik megfordítja a paraméterként kapott, nullával (‘\0’) lezárt karaktersorozatot (stringet) úgy, hogy az visszafelé legyen olvasható! A függvény az eredményt a bemeneti adat helyén adja vissza!

6.4 Készítsen C függvényt, amely paraméterként egy stringre mutató pointert és egy karaktert kap, megkeresi a sztringben a karakter első előfordulási helyét, és visszatérési értékül egy erre mutató pointert ad vissza. Ha nem találja, NULL pointert ad vissza (return NULL;)

6.5 Készítsen C függvényt, amely a paraméterként kapott sztringben minden kisbetűt nagybetűre cserél!

6.6 Készítsen függvényt, mely paraméterként kap három karaktertömböt, és az első kettőben lévő stringet összefésülve, azaz betűnként felváltva átmásolja a harmadikba. Pl. s1=”abcde”, s2=”1234567”, az eredmény: s3=”a1b2c3d4e567”. Ne felejtse el lezárni a stringet!

6.7 Írjon olyan szabványos ANSI C ***függvényt***, amely átmásol egy sztringet egy másikba olymódon, hogy a másolatban az egymást követő azonos karakterekből csak egyet tart meg. A függvény paraméterként kapja a sztringek címét, és feltételezzük, hogy az eredmény-sztring alkalmas az adatok tárolására. A függvény prototípusa a következő:

```
void spec_strcpy(char *source, char *destination);
```

6.8 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy paraméterén átvett sztringből eltávolítja a szóköz karaktereket! Pl. ha az átadott karaktertömb a ”Jobb ma egy veréb, mint holnap egy túzok!” sztringet tartalmazza, akkor a függvény a sztringet ”Jobbmaegyveréb,mintholnapegytúzok!”-ra módosítsa!

6.9 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap **egy stringet és egy karaktert**, és a stringet **eredeti helyén** úgy változtatja meg, hogy törli belőle a megadott karakterrel megegyező karaktereket! Nem használhat segédtömböt!
Például be: ”abrakadabra sikeres C vizsga” és ’a’, ki: ”brkdbr sikeres C vizsg”

6.10 Írjon olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy sztringet, melyet úgy alakít át, hogy törli belőle a nagybetűket! Pl.: be: `”Az EET a kedvenc-M!”`; ki: `”z  a kedvenc-”`. Nem használhat segédtömböt!

6.11 Írjon olyan szabványos ANSI C ***függvényt***, amely paraméterül kap három karaktertömböt! Az első két karaktertömbben lévő sztring tartalmát úgy másolja át a harmadikba, hogy felváltva másol át egy-egy szót az elsőből ill. a másodikból! Minden szó, ami nem whitespace karakter. Pl.: be1: „Nincs mint EET-n”, be2: „jobb, az tanulni!”, ki: „Nincs jobb, mint az EET-n tanulni!”

6.12 Írjon egy olyan ANSI C **függvényt**, amely egy szabályos C sztring címét, valamint egy karaktert kap paraméterként, és megkeresi a karakter legutolsó (legjobboldalibb) előfordulását a sztringben. A függvény visszatérési értéke a megtalált karakter indexe, ha a karakter nem szerepel a sztringben, akkor –1 legyen. Ne használjon könyvtári függvényeket!

6.13 Adott a következő ***függvény*** prototípus:

```
void dekodolo(char*cel,char*szamok,char*betuk);
```

A függvény a `szamok` tömbben egy kódolt üzenetet kap paraméterül, melyet a `betuk` tömb segítségével dekódol, és az eredmény stringet a `cel` tömbben helyezi el. A `szamok` tömb tartalma egy kétjegyű decimális számok sorozatából álló string. A kétjegyű számok a `betuk` tömbben lévő karakterek indexét jelentik. A számpárokat sem szóköz, sem egyéb karakter nem választja el egymástól! Készítse el a függvény definícióját szabványos ANSI C nyelven! Ne felejtse el a `cel` stringet megfelelően lezárni!
Pl.: `szamok=”00000602050302060400020100030607”, betuk=”EB SHIT!” => cel=”EET IS THE BEST!”`

6.14 Írjon olyan szabványos ANSI C függvényt, amely paraméterként kap két stringet, és megvizsgálja, hogy az első string tartalmazza-e a másodikat! Ha igen, adja vissza a megtalált string címét, ha nem, NULL pointert adjon vissza! Más függvényt nem hívhat!

6.15 Írjon olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy sztringet, benne egy csak számjegyeket tartalmazó telefonszámmal, továbbá paraméterként kap egy másik sztringet, melyben az eredményt kell eltárolni. A függvény visszatérési értéke egy előjel nélküli egész érték, melyben a körzetszámot kell visszaadni. A bemenő telefonszám vagy tartalmaz körzetszámot, vagy nem. Ha tartalmaz, akkor a szám első két jegye 06, ezt követi egy vagy két számjegy, a körzetszám. A körzetszám lehet 1, vagy 20-99 közötti érték. Ha a telefonszám nem tartalmaz körzetszámot, akkor a függvény 0-t adjon vissza! A paraméterként kapott második (eredmény) sztringben adja vissza a körzetszámtól megfosztott telefonszámot! (Pl. be: 0614631111, vissza paraméter: 4631111, vissza return: 1)

7. Rendezés

7.1 Készítsen rendez nevű függvényt, amely paraméterként kap egy stringet, melynek karaktereit növekvő sorrendbe rendezi! A függvény más függvényt nem hívhat!

7.2 Írjon C függvényt, amely egy 100-szor 100-as méretű, double elemeket tartalmazó mátrixot sorfolytonosan rendez.

7.3 Írjon C függvényt, amely a paraméterként kapott n elemű valós tömböt egy másik, szintén n elemű integer tömb alapján helyben rendez. A második tömb 0-tól n-1-ig tartalmazza az egész számokat. Ez határozza meg, hogy milyen sorrendben kell a valós tömböt rendezni. Tehát ahol index[x] = 0, ott tömb[x] = max. Az írandó függvény prototípusa:

```
void rendez(double tomb[], int index[], int n);
```

7.4 Írjon C függvényt, amely sztringekre mutató pointerek tömbjét kapja paraméterként, és a sztringeket csökkenő sorrendbe rendezi.

7.5 Írjon C függvényt, mely sztringeket tartalmazó tömböt rendez. A tömbben tárolt stringek maximális hossza 80 karakter.

a. beszúrásos rendezéssel növekvő sorrendbe
b. beszúrásos rendezéssel csökkenő sorrendbe

7.6 Írjon szabványos ANSI C függvényt, mely paraméterként kap egy

```
typedef struct{int kulcs; double ertek;} elem;
```

típusú elemekből álló tömböt, valamint a tömb méretét, és sorba rendezi a tömb elemeit kulcs szerinti csökkenő sorrendbe. A rendezéshez tetszőleges algoritmus használható, de nem használhatja a `stdlib.h`-ban deklarált `qsort()` könyvtári függvényt!

7.7 Adott a következő típusdefiníció:

```
typedef struct{double re,im;}komplex;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy `komplex` elemekből álló tömböt, valamint a tömb elemszámát veszi át paraméterként, és rendezi a tömb elemeit re^2+im^2 norma szerinti csökkenő sorrendben. A tömb rendezéséhez felhasználhatja az `stdlib.h`-ban deklarált szabványos `qsort` függvényt, vagy írhat maga is saját rendező kódot.

7.8 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap **egy stringet**, melynek páros indexű karaktereit ASCII kód alapján, nagyság szerinti növekvő sorrendbe **rendezi** az **eredeti stringen belül**. Nem használhat segédtömböt!
Pl. be: ”szeretem_a_c_nyelvet”
ki: ”_z_r_temeaecenlesvyt”

7.9 Készítsen olyan szabványos ANSI C ***függvényt***, amely paraméterként kap egy int elemeket tartalmazó tömböt és egy előjel nélküli egész értéket (n)! A tömbben számpárok találhatók, n érték a számpárok számát jelenti, vagyis a tömb mérete 2n. Rendezze sorba a számpárokat a számpárok első tagjának értéke szerinti növekvő sorrendbe! Pl.: be: 4, -6, 2, 10, 3, 0, 1, 9; ki: 1, 9, 2, 10, 3, 0, 4, -6. Itt n=4.

7.10 Készítsen egy olyan C ***függvényt***, amely egy x, y, z valós értékeket tartalmazó struktúrákból álló, paraméterként átadott tömb elemeit az origótól való távolság szerint növekvő sorba rendezi! Az x, y, z értékek egy pont koordinátái derékszögű koordinátarendszerben. Definiálja a szükséges struktúratípust is!

8. Függvénypointer

8.1 Írjon C függvényt, amely a paraméterként kapott double f1(double), és double f2(double) függvények különbségét táblázatos formában kiírja a standard outputra a szintén paraméterként kapott [a,b] zárt intervallumban e lépésközzel, és visszaadja a legkisebb különbség értéket.

8.2 Írjon ANSI C ***függvényt***, amely paraméterként kap egy valós, egyváltozós matematikai függvényre mutató pointert, egy intervallum két határát és egy harmadik valós számot, ami a felbontást (lépésköz) adja meg a függvény numerikus integrálásához. A függvény visszatérési értéke a paraméterként kapott függvény integrálja legyen az adott határok között, az adott felbontással. Tetszőleges numerikus integrálási algoritmust használhat.

8.3 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kapja egy double paraméterű, double visszatérési értékű függvény címét (`f`), valamint három double típusú értéket (`a`, `b`, `dx`), és visszaadja a függvény [`a`,`b`] zárt intervallumon felvett maximumát. A maximumot úgy határozza meg, hogy végigjárja az [`a`,`b`] intervallumot `dx` osztásközzel, és minden lépésben megvizsgálja a függvény értékét. Ahol a függvény értéke a legnagyobb, azt tekintjük maximumnak. Ha a paraméterátadás rossz, de a függvény törzse hibátlan, 4 pont jár. (Az `f` függvénypointer használata: `y=f(x);`, mintha egyszerű függvényhívás lenne. (Az is.))

9. Lista, fa

9.1 Adottak a következő definíciók:

```
typedef struct bf{
    int adat;
    struct bf *bmut, *jmut;
} bifa, *pbifa;
```

Készítsen egy olyan szabványos ANSI C függvényt, amely meghatározza, hogy egy ilyen `bifa` típusú elemekből álló bináris fában hány olyan csomópont van, amelynek csak egy gyermeke van. Az adatszerkezet "végét" a megfelelő mutató `NULL` értéke jelzi. A függvény bemenő paramétere a fa gyökerére mutató pointer, visszatérési értéke a fenti módon meghatározott érték legyen!

9.2 Egy n irányba elágazó fa ábrázolásához az alábbi adatstruktúrát definiáljuk:

```
typedef struct nf {
    double adat;
    unsigned n;       /* elágazások száma: 0 vagy pozitív egész */
    struct nf **agak;
    double value;
} nfa, *pnfa;
```

Ha `0` az elágazások száma, `n` értéke `0` és `agak` ertéke `NULL`. Ha `n>0`, akkor `agak[i]!=NULL`, ahol `0<=i` és `i<n`. Készítsen egy olyan szabványos ANSI C függvényt, amely visszatérési értékül szolgáltatja egy ilyen fa `value` adatainak összegét!

9.3 Egy fésűs listához az alábbi két struktúrát használjuk:

```
typedef struct br{int n; struct br *a;} ag,*pag;
typedef struct tzs{struct tzs *t; pag a;} torzs,*ptorzs;
```

A `torzs` típusú elemekből felépített láncolt lista elemeinek `a` tagjai `ag` típusú elemekből felépített láncolt listákra mutatnak. A listák végét NULL pointer jelzi. Készítsen egy olyan szabványos ANSI C függvényt, amely visszatérési értékül szolgáltatja egy ilyen fésűs lista `n` adatainak összegét!

9.4

```
typedef struct li {
    double adat;
    unsigned n;
    struct li *next;
} list, *plist;
```

Írjon szabványos ANSI C függvényt, amely paraméterként kap egy `plist` típusú pointert, mely egy láncolt lista első elemére mutat, valamint egy pozitív egész számot (x), és kitörli a lista x-edik elemét. A függvény visszatérési értéke a módosított lista első elemére mutató pointer. Figyeljen arra, hogy a lista kezdőelemét is kitörölhesse a felhasználó. Ha a megadott sorszám nagyobb, mint az elemek száma, akkor NULL pointert adjon vissza.

9.5 Egy bináris fa a következő típusú elemekből áll:

```
typedef struct bfg{
    char szo[20], int hossz, int db;
    struct bfg *bal,*jobb;
} elem,*pelem;
```

Készítsen szabványos C függvényt, amely paraméterként kapja a fa gyökéreleménének címét, valamint egy karaktertömböt, és megkeresi a fában a megadott stringet, majd visszatér a megtalált elem típusú struktúra címével! Ha a keresett szó nem szerepel a fában, NULL pointerrel térjen vissza!

9.6 Adottak a következő definíciók:

```
typedef struct bf {
    double ertek;
    struct bf *bmut, *jmut;
} bifa, *pbifa;
```

Készítsen egy olyan szabványos ANSI C függvényt, amely meghatározza az ilyen elemekből álló bináris fában tárolt értékek összegét. Az adatszerkezet "végét" a megfelelő mutató NULL értéke jelzi. A függvény bemenő paramétere a fa gyökerére mutató pointer, visszatérési értéke a fenti módon meghatározott érték legyen! Válasszon megfelelő paramétereket! Törekedjen gyors megoldásra!

9.7 Adottak a következő definíciók:

```
typedef struct qf {
    int adat;
    struct qf *mut1, *mut2, *mut3, *mut4;
} quadfa, *pquadfa;
```

Készítsen egy olyan szabványos ANSI ***C függvényt***, amely meghatározza, hogy egy ilyen `quadfa` típusú elemekből álló kavdrális (négyágú) fában hány olyan csomópont van, amelynek *pontosan négy gyermeke van*. Az adatszerkezet "végét" a megfelelő mutató `NULL` értéke jelzi. A függvény bemenő paramétere a fa gyökerére mutató pointer, visszatérési értéke a fenti módon meghatározott érték legyen! Nem használhat globális változót!

9.8 Adottak a következő definíciók:

```
typedef struct bf {
    int adat;
    struct bf *bmut, *kmut, *jmut;
} trifa, *ptrifa;
```

Készítsen egy olyan szabványos ANSI C függvényt, amely meghatározza, hogy egy ilyen `trifa` típusú elemekből álló fa végpontjainak a számát. Végpontnak azokat a csomópontokat nevezzük, amelyeknek egy gyermeke sincs (azaz nem indul belőle egy ág sem). A függvény bemenő paramétere a fa gyökerére mutató pointer, visszatérési értéke a fenti módon meghatározott érték legyen! Nem használhat globális változót!

9.9 Adottak a következő definíciók:

```
typedef struct bf {
    double ertek;
    struct bf *bmut, *jmut;
} bifa, *pbifa;
```

Készítsen egy olyan szabványos ANSI C ***függvényt,*** amely a szabványos kimenetre kiírja az ilyen elemekből álló bináris fa adott szintjén lévő elemek értékét! A gyökérelem az 1. szinten van, gyerekei a 2. szinten, stb. A függvény egyik bemenő paraméterében adjuk át a kiírandó szint számát, egy másik paramétere pedig a fa gyökerére mutat. Az adatszerkezet "végét" a megfelelő mutató NULL értéke jelzi. Nem használhat globális változót! Törekedjen gyors megoldásra!

9.10 Adottak a következő definíciók:

```
typedef struct bf {
    double ertek;
    struct bf *bmut, *jmut;
} bifa, *pbifa;
```

Írjon egy olyan szabványos ANSI C ***függvényt***, mely bemenő paraméterként egy ilyen `bifa` típusú elemekből álló fa gyökerére mutató pointert, valamint egy zárt intervallumot kap, és visszatérési értékül adja a fa azon csomópontjainak számát, melyek `ertek` adattagja a megadott zárt intervallumba esik! A függvény nem használhat globális vagy statikus változókat! Az adatszerkezet "végét" a megfelelő mutató NULL értéke jelzi. Törekedjen gyors megoldásra!

9.11 Adottak a következő definíciók:

```
typedef struct nf {
    int adat;
    int n;
    struct nf **mut;
} nfa, *pnfa;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely meghatározza egy ilyen `nfa` típusú elemekből álló n-ágú fa maximális mélységét. Egy csomópontról kiinduló ágak számát (azaz a mut tömb méretét) az `n` mező értéke adja meg, `mut[i]` a csomópontból kiinduló `i`-edik ágra mutat. Ha `mut[i]` értéke `NULL`, akkor abban az irányban "vége" az adatszerkezetnek. A függvény bemenő paramétere a fa gyökerére mutató pointer, visszatérési értéke a fenti módon meghatározott érték legyen! Nem használhat globális változót!

9.12 Írjon C függvényt mely paraméterként kapja egy bináris fa gyökéreleménének címét, valamint két valós számot, a-t és b-t! A fa elemei a következő típusúak:

```
typedef struct nf {
    int adat;
    int n;
    struct nf **mut;
} nfa, *pnfa;
```

A fa név szerint rendezett. A függvény írja ki ABC sorrendben azoknak a neveit a fából, akiknek az átlaga az [a,b] zárt intervallumba esik.

9.13 Egy városnak olyan felépítésű a csatornarendszere, hogy minden csomópontban kétfelé ágazik el, így egy bináris fával modellezhető. A fában minden elemnek van egy *unsigned* típusú adatmezője, amely megadja az előző csomóponttól az aktuálisig tartó csatornaszakasz hosszát méterben és van egy másik *double* típusú adatmezője, amely megadja ezen csatornaszakasz keresztmetszetét négyzetméterben. A csatornát egy áradás következtében elöntötte a víz. Definiáljon alkalmas C struktúrát a csatornarendszert leíró bináris fa ábrázolására! Írjon egy olyan C függvényt, amely paraméterként kapja a bináris fa gyökér elemére mutató pointert, és megmondja, hogy hány köbméter vizet kell kiszivattyúzni a rendszerből!

9.14 Adott a következő definíció:

```
typedef struct mf {
    double adat;
    unsigned m;
    struct mf **p;
} magufa, *pmagufa;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely meghatározza, egy ilyen `magufa` típusú elemekből álló fa n-edik szintjén található `adat`-ok összegét! Egy csomópont p mezője egy m elemű dinamikus tömbre mutat, amelyben a csomópont gyerekeire mutató pointereket tároljuk. Ha egy elemnek nincs gyereke, p NULL pointer. A függvény bemenő paramétere a fa gyökerére mutató pointer, valamint n, azaz a keresett szint. Visszatérési értéke az n-edik szinten tárolt adatok összege legyen! Nem használhat globális változót!

9.15 Adott a következő definíció:

```
typedef struct trifa {
    struct trifa *mutb, *mutk, *mutj;
    long adat;
} TriFa, *PTriFa;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely meghatározza egy ilyen `TriFa` típusú elemekből álló fa mélységét. A függvény egyetlen bemenő paramétere a fa gyökerére mutató pointer legyen, a visszatérési értéke pedig a fa mélységét megadó előjel nélküli egész szám. Globális változót nem használhat.

9.16 Adott a következő típusdefiníció:

```
typedef struct melon{long dinnye; struct melon *left,*right;}fa,*pfa;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy `fa` elemekből álló bináris fát vesz át paraméterként, és visszatérési értékként megadja, hogy a fa levélelemeiben hány `dinnye` található. Levélelem az az elem, amelynek nincs gyermekeleme, azaz a `left` és `right` pointer egyaránt `NULL` értékű. A függvény visszatérési értéke tehát a levélelemekben található `dinnye` tagváltozók összege.

9.17 Adott a következő típusdefiníció:

```
typedef struct fa{
    unsigned kulcs;
    char szo[50];
    struct fa *bal,*kozep,*jobb;
}trifa,*ptrifa;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy `trifa` elemekből álló háromágú fát vesz át paraméterként. A háromágú fa **kulcs szerint rendezett**: a bal ágakon a gyökérelemnél kisebb kulcsú, a középső ágakon a gyökérelemmel megegyező kulcsú, a jobb ágakon a gyökérelemnél nagyobb kulcsú elemek találhatók A függvény írja a szabványos kimenetre kulcs szerint növekvő sorrendben azon elemek `szo` adattagját, amelyek kulcsa páros szám! Azonos kulcsú elemek esetében a fában a gyökérhez legközelebbi elem kiírásával kezdje, és a legtávolabbi felé haladjon! Ha nem növekvő sorrendben írja ki, de a megoldás amúgy jó, max. 4 pontot kaphat. Ha valamelyik ág nem létezik, a megfelelő pointer értéke NULL.

9.18 Adott a következő típusdefiníció:

```
typedef struct list{
    unsigned kulcs;
    char szo[50];
    struct list *bal,*jobb;
}elem,*pelem;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy `elem` típusú elemekből álló duplán **láncolt lista** első elemére mutató pointert (p) és egy előjel nélküli egész értéket (k) vesz át paraméterként, és kiírja azon listaelemek `szo` adattagját, amelyek után olyan elem következik, melyek kulcs adattagja k-val egyenlő.

9.19 Írjon függvényt, amely az alábbi struktúrából felépülő rendezetlen láncolt listát megrendezi tetszőleges módszerrel, csökkenő sorrendben, az `a` mező szerint.

```
typedef struct _e {
    int a;
    struct _e *n, *p;
} e;
```

9.20 Adott a következő típusdefiníció:

```
typedef struct lala{
    long pont;
    struct lala * next;
}lista;
```

Írjon olyan szabványos ANSI C ***függvényt***, amely paraméterül kap egy lista elemekből felépített egyszeresen láncolt listát! A listában oldalhosszakat tárolunk a, b, c, a, b, c,… sorrendben. A függvény számolja meg, hogy hány olyan hossz-trió van, ahol az értékek lehetnek egy háromszög oldalai (bármely két oldal hossza nagyobb a harmadiknál), és ezt az értéket adja vissza a függvény! A lista végét a next pointer NULL értéke jelzi. Ha a lista elemeinek száma nem osztható hárommal, azaz az utolsó hossz-trió nem teljes, a függvény adjon vissza -1-et!

9.21 Készítsen egy olyan C ***függvényt***, amely paraméterként kapja egy egyszeresen láncolt lista kezdőelemének címét! A listaelemek egy-egy valós értéket tárolnak. Definiálja a lista létrehozásához szükséges adattípust is! A függvény adja visszatérési értékként a lista szigorúan monoton csökkenő szakaszainak számát!

9.22 Egy telefonkönyvi adatokat tartalmazó, név szerint rendezett bináris fa ábrázolásához az alábbi adatstruktúrát definiáljuk:

```
typedef struct bf {
    char *nev,*szam;
    struct bf *bal,*jobb;
} bfa, *pbfa;
```

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként veszi át a fa gyökerének címét, és kiírja az összes „Nagy” vezetéknevű személyt telefonszámostul ABC sorrendben! A „Nagyító” vezetéknevű személy nem „Nagy” vezetéknevű személy!

10. Fájl

10.1 Írjon olyan szabványos ANSI C programot, amely a `szoveg.txt` nevű szövegfájl utolsó 8 sorát írja ki fordított sorrendben. Egy sor hossza maximum 1000 karakter lehet.

10.2 A koordi.dat nevű bináris fájl struct{double x,y;}; felépítésű struktúrákat tartalmaz. Írjon programot, amely kiírja a leghosszabb vektor koordinátáit! A fájlt csak egyszer olvashatja!

10.3 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap **két stringet**, mint fájlnevet; az első fájt **bináris** olvasásra, a másodikat **bináris** írásra nyitja meg, és átmásolja az első fájlt a másodikba, majd bezárja a fájlokat.

Maximumrész feladatok

1. ZH és viszgafeladatok

(mindkettőben lehetnek ilyenek)

1.1 Írjon olyan egy olyan C nyelvű teljes programot, amely a szabványos bemenetről érkező valós számok közül az utolsó 8 szám mértani középértékét szolgáltatja visszatérési értékként! A szabványos bemenetről a program csak számokat kap, de előre nem ismert, hogy hány darabot. Ha 8 számnál kevesebbet olvasna be a függvény, a hiányzó számokat tekintse 1-nek. A számok végét a 0.0 érték beolvasása jelzi. Ügyeljen arra, hogy a függvény a lehető legegyszerűbb legyen, és ne végezzen felesleges adatmozgatásokat!

1.2 Írjon olyan maximum kereső függvényt (8 pont), amely paraméterként kapja egy tömb (y) elemeit és az aktuális darabszámát (n), megkeresi a tömb legkisebb elemét. Cím szerint átadott paramétereiben (min_elem) szolgáltassa magát a megtalált legnagyobb tömbelemet és visszatérési értékként szolgáltassa annak indexét! A megoldás során használja az alábbi deklarációkat:

```
typdef int *tomb;
int min_keres(tomb y, int n, int *min_elem);
```

1.3 Írjon egy olyan teljes C programot (7 pont), amely a felhasználótól bekéri a tömb maximális méretét, létrehozza azt és feltölti a felhasználótól bekért adatokkal, majd a fenti függvény aktivizálásával megállapítja a legkisebb tömbelemet és azt az indexével együtt kiírja a képernyőre. A tömbelemek beolvasása addig történjék, amíg a tömb be nem telik vagy 0-t nem ad a felhasználó tömbelemként. A program visszatéréskor szabadítsa fel az általa lefoglalt memóriát!

1.4 Adott a `typedef int tomb[100];` típus és egy ilyen típusú `t` tömb, véletlen értékekkel teljesen feltöltve. Írjon egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kapja a fenti `t` tömböt és azt a következő módon rendezi:

`t[0]`	1. legkisebb	`t[99]`	2. legkisebb
`t[1]`	3. legkisebb	`t[98]`	4. legkisebb

...
stb.

Az eljárás csak helyben rendezheti a tömböt; másik, segédtömböt vagy dinamikusan foglalt tárterületet nem használhat.

1.5 A standard inputon előre nem ismert számú (`x,y`) koordinátapárt olvasunk. Az egyes valós számértékeket *whitespace* karakterek határolják. Írjon olyan C programot, amely kiírja az egyes koordinátapontok távolságát az összes pont, mint pontrendszer súlypontjától mérve. Feltehető, hogy az adatok hibátlanok (helyes formátumú, páros számú valós szám). A program file-okat nem használhat. Az esetlegesen dinamkusan lefoglalt memóriát szabadítsa fel!

A súlypont x_{sp} koordintája így számolható (azonos "tömegű" pontok esetén): $x_{sp} = \sum_{i=1}^{n} x_i / n$, ahol n a pontok száma.

1.6 Írjon egy olyan *mondatvizsgalo* nevű C ***függvényt***, amelyben a szabványos bemenetről olvassuk a karaktereket, ehhez használjuk a *getchar()* függvényt. Figyelje az adatfolyam végét jelentő EOF-ot! A mondatokat típusonként számlálja meg, azaz külön a kijelentő mondatokat, a felkiáltó mondatokat és a kérdő mondatokat, amelyeket a rendre a . (pont), a ! (felkáltó jel) és ? (kérdőjel) zár le. (A mondatok többi karakterét a vizsgálat során figyelmen kívül hagyjuk.) A kijelentő és a kérdő mondatokat számláló változókat a paraméterlistán, a felkiáltó mondatokat számláló változót értékét pedig a **return-**nel, visszatérsi értékként adja vissza!

```
int mondatvizsgalo(int *kijelento, int *kerdo);
```

Egészítse ki a fenti függvényt egy ***teljes C programmá*** egy olyan *main* függvénnyel, amelyben aktiválja a *mondatvizsgalo* függvényt az alábbiakban deklarált változókkal és a vizsgálat eredményét a szabványos kimenetre nyomtatja!

```
void main() { int kij,ker,fel;
```

1.7 Készítsen egy olyan teljes ANSI C programot, amely meghatározza, hogy a szabványos bemeneten beolvasott szövegben hány karakteres a leghosszabb sor és kiírja ennek a sornak a hosszát, valamint kiírja azt is, hogy ez hányadik sor volt. A sorokat 1-től sorszámozzuk.

1.8 Készítsen egy olyan teljes ANSI C programot, amely beolvassa és kiírja a szabványos bemeneti állomány (`stdin`) tartalmát, de úgy, hogy két szó között pontosan egy szóközt hagy csak. A szavakat az eredeti szövegben tetszőleges számú szóköz választhatja el. Szónak tekintünk minden olyan karaktersort, amelyet tetszőleges számú *whitespace* karakter határol. A TAB karaktert (`'\t'`) 8 darab space karakternek kell tekinteni. Az újsor jel (`'\n'`) a kiírás szempontjából nem tekintendő space karakternek; a szöveg sorokra tagolása maradjon az eredeti!

1.9 Írjon szabványos ANSI C programot, mely egy áruház beléptető rendszerét kezeli a következő feltételeknek megfelelően:

- Az őrök bármikor beléphetnek az áruházba, de az utolsó őr csak akkor léphet ki, ha rajta kívül senki sincs már bent.
- A főnökök bármikor bejöhetnek, amikor van bent őr, és bármikor elmehetnek.
- Az alkalmazottak csak akkor léphetnek be, ha van benn főnök és őr, de csak akkor léphetnek ki, ha kilépésük után legalább tizedannyi alkalmazott marad benn, mint ahány ügyfél.
- Az ügyfelek csak akkor léphetnek be, ha belépésük után az ügyfelek összes száma nem haladja meg a benn lévő alkalmazottak tízszeresét, és bármikor kiléphetnek.

A be- illetve kilépési szándékot a standard inputról érkező karakterpár jelzi, az első karakter: O: őr, F: főnök, A: alkalmazott, U: ügyfél; a második karakter + vagy -, ha az illető be- ill. kilépni szeretne. Feltételezheti, hogy nem érkezik más a bemeneten, csak a megadott párosok. A program a következő négy szöveg valamelyikét írja ki: BELÉPHET, NEM LÉPHET BE, KILÉPHET, NEM LÉPHET KI.

1.10 Írjon szabványos ANSI C függvényt, amely meghatározza a paraméterként kapott double visszatérési értékű, egyváltozós, double paraméterű függvény minimumát a szintén paraméterként kapott [A,B] intervallumban! A minimum keresését egy paraméterként adott lépésközzel végezze! A függvény függvényértékként adja vissza a megtalált minimum helyét.

1.11 Ön egy optikai karakterfelismerő programot fejlesztő csapat tagjaként feladatként kapja egy függvény elkészítését. A függvény deklarációja alább látható. Paraméterek: a `kep` egydimenziós, `sor*osz` méretű tömbben található a beolvasott kép, a tömb elemei a képpontok. A képpontok elrendezése a következő: először jön a kép első sorának `osz` darab képpontja, majd a második sor `osz` darab képpontja, és így tovább. Minden képpont 0 és 255 közötti értéket vehet fel: a 0 teljesen fekete, a 255 teljesen fehér, a kettő között különféle szürkeárnyalatok találhatók. Úgy tekintjük, hogy egy adott képpontsorban akkor van szöveg, ha a képpontok legalább 2%-a sötét, azaz értéke 0 és 127 között van. A függvény a) paraméterként adja vissza a `szoveg` pointer által mutatott változóban, hogy a szöveg egy sora átlagosan hány képpont magas, b) a `sorkoz` segítségével, hogy a szöveg sorai közötti távolság átlagosan hány képpont, c) visszatérési értékként pedig, hogy a képen hány sornyi szöveg látható. Az első szövegsor előtti, és az utolsó szövegsor utáni szövegmentes területet ne vegye figyelembe a sorok közti távolság kiszámításánál!

```
unsigned sorszam(unsigned kep[], unsigned sor, unsigned osz, unsigned * szoveg, unsigned * sorkoz);
```

1.12 Készítsen programot, mely a standard bemenetről (billentyűzetről) érkező szöveget jeleníti meg a standard kimeneten (képernyőn) olyan formában, hogy ha egy szó nem fér el az adott sorban, akkor az egész szót a következő sorba írja ki, tehát a szavakat nem vághatja ketté. A képernyőn egy sor 80 karakter hosszú. A szövegben minden szó rövidebb 80 karakternél. Szónak számít minden, ami nem whitespace karakter (szóköz, tabulátor, soremelés).

1.13 Egy szöveg típusú állományban az irodalmi hivatkozásokat szögletes zárójelbe tett egész számok jelzik pl: **[131]**. A szövegben más célra is alkalmaznak szögletes zárójeleket, de ekkor a zárójelet mindig megelőzi egy *backslash* karakter pl: **\[** vagy **\]**.

Tervezzen szabványos ANSI **C függvényt**, amely beolvassa egy megnyitott szöveges állományból a soron következő irodalmi hivatkozás azonosítószámát. Definiálja a függvényt úgy, hogy az adja vissza a beolvasott azonosítót, mint egy egész számot és jelezze, ha már nincs több irodalmi hivatkozás az állományban! A megnyitott állomány pointerét paraméterként vegye át.

- Definiálja a függvény paramétereit, és magyarázza meg döntését!
- A feladat megoldásához tervezzen állapotgépet, amit állapotátmeneti gráffal, vagy állapottáblával adjon meg!
- Valósítsa meg a függvényt!

Példa a bemenő formátumra:

Remek olvasmány a Winetou c. mű [12]. Ebben \[számos\] remek történetet olvashatunk. Ajánlhatók még a[81][82] művek is.

1.14 Egy autószerviz számítógépe minden megbízásról egy-egy sort ír a SZERVIZ.TXT fájlba. Egy sor a következő adatokat tartalmazza:

```
RENDSZÁM            : 6 karakter
TULAJDONOS NEVE     : 40 karakter
FORG. ENG. SZÁMA    : 8 karakter
DÁTUM               : 11 karakter, (2006.06.19.)
ÖSSZEG              : 6 karakter (egész szám)
```

Írjon C programot, amely a SZERVIZ.TXT fájl alapján névsorban kiírja azoknak az autótulajdonosoknak a nevét, akik az átlagosnál többször javíttatták 2006 februárjában autójukat. A fájlt csak egyszer olvashatja be! Használjon dinamikus adatszerkezetet!

1.15 Készítsen egy olyan szabványos ANSI C programot, amely a szabványos bemenetén egy hibátlan, szabványos Pascal program szövegét fogadja, és külön-külön meghatározza, hogy hány olyan kommentárt tartalmaz ez a forrásállomány, amelyet a `(* *)`, illetve a `{ }` karakterek határolnak. Feltételezheti, hogy a szöveg nem tartalmaz "vegyes", azaz `(* }`, illetve `{ *)` karakterek által határolt kommentárokat. Feltételezheti, hogy nincsenek egymásba ágyazott megjegyzések. A Pascal nyelvben a sztringkonstansokat az aposztrof karakter határolja. Ezekben tetszőleges karaktersorozat lehet. Összetett kifejezések felírásmódja, függvény- ill. eljáráshívás a Pascal-ban hasonló a C-ben megszokottakhoz. A * operátor a Pascal-ban a szorzás jele, nincs egyoperandusú alakja. A Pascal nyelv egyéb szintaktikai szabályaira vonatkozó ismeret nem szükséges a feladat megoldásához.

1.16 Írjon programot, mely a standard bemenetről érkező szövegben minden kisbetűt nagybetűre cserél, az eredményt a standard kimenetre (képernyő) írja. A []-ek között álló szöveget változatlan formában írja ki (itt tehát nem alakítja a kisbetűket naggyá), a szöveg végét EOF jelzi. A zárójelek egymásba ágyazhatók, tehát pl.: be: ”abc de14 x[un1[z3]k2]rs5a” => ki: ”ABC DE14 X[un1[z3]k2]RS5A”.

1.17 Adott a `typedef int tomb[100];` típus és egy ilyen típusú `t` tömb, véletlen értékekkel teljesen feltöltve. Írjon egy olyan szabványos ANSI ***C függvényt***, amely paraméterként kapja a fenti `t` tömböt és azt a következő módon rendezi:

`t[1]`	1. legnagyobb	`t[99]`	2. legnagyobb
`t[2]`	3. legnagyobb	`t[98]`	4. legnagyobb

...
stb.

Az eljárás csak helyben rendezheti a tömböt; másik, segédtömböt vagy dinamikusan foglalt tárterületet nem használhat.

1.18 Egy házaspár pontosan ugyanannyi fiúgyermeket szeretne, mint lányt. Elhatározzák, hogy addig vállalnak újabb gyereket, amíg ez nem teljesül, de legfeljebb `M` gyerekük lesz. Készítsen olyan szabványos ANSI C ***programot***, amely `N` ilyen házaspár esete alapján meghatározza, hogy átlagosan hány gyerekük lesz és hány százalékuknak sikerül elérni az azonos fiú-lány számot. `N` és `M` a program parancs-sor paramétereként az operációs rendszerben megadott bemeneti adat, és feltesszük, hogy a fiúk és a lányok születési valószínűsége azonos. Használhatja az `<stdlib.h>` fejlécfile-ban deklarált `int random(int num)` véletlen szám generátor függvényt, amely minden meghívása alkalmával egy-egy új, `0` és `num-1` közé eső, egyenletes eloszlású véletlen értéket ad..

1.19 Készétsen egy olyan teljes ANSI C programot, amely beolvassa és kiírja a szabványos bemeneti állomány (`stdin`) tartalmát, de úgy, hogy két szó közé pontosan csak egy space karatert ír. A szavakat az eredeti szövegben tetszőleges számú *szóköz* választhatja el. *Szóköznek* tekintjük tetszőleges számú *whitespace* karakter (space karakter, TAB kerakter, újsor jel) tetszőleges hosszúságú sorozatát. Tehát szónak tekintünk minden olyan karaktersort, amelyet tetszőleges számú *whitespace* karakter határol. A TAB karaktert (`'\t'`) 8 darab space karakternek kell tekinteni. Az újsor jel (`'\n'`) a kiírás szempontjából nem tekintendő space karakternek; a szöveg sorokra tagolása maradjon az eredeti!

1.20 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely úgy írja ki egy paraméterként átadott `unsigned int` típusú egész szám decimális értékét a szabványos kimenetre, hogy 3 számjegyenkét tagolja azt egy-egy szóközzel. Pl. `1234567` kiírása: `1 234 567`. A feladat megoldása során az ábrázolható legnagyobb egész szám konkrét értékére vonatkozólag semmiféle feltételezéssel nem élhet, azt semmilyen előre definiált konstans értékből nem állapíthatja meg!

1.21 Készítsen egy olyan szabványos ANSI C programot, amely a szabványos bementről beolvasott angol nyelvű szövegben található nagybetűk előfordulási gyakoriságáról statisztikát készít. A statisztikát a szabványos kimenetre listázza ki az előfordulások számának növekvő sorrendjében a következőképpen: betű, előfordulások száma, relatív gyakoriság a szöveg összes karakterére vetítve, relatív gyakoriság a nagybetűk összes előfordulására vetítve. Ha egy betű egyszer sem fordult elő a szövegben, akkor az a listán ne szerepeljen! Például:

E	46	0.032	0.12
A	32	0.022	0.08

...

Definiáljon alkalmas adatstruktúrákat a feladat hatékony megoldásához és magyarázza meg azt! Az optimális adatstruktúra-választást és a hozzá tartozó magyarázatot 2 ponttal honoráljuk.

1.22 Készítsen egy olyan teljes ANSI C ***programot***, amely beolvas a szabványos bemenetről legfeljebb 2000 nevet, amelyekről tudjuk, hogy egyikük sem hosszabb 30 karakternél, és mindegyik külön sorban van, majd kiírja azokat névsorban. A program írja ki azt is, hogy a rendezés során konkrétan hány összehasonlítást végzett. A neveket csak helyben rendezheti! Adja meg rendezési módszere nevét és a rendezéshez szükséges összehasonlítások maximális számát is. (A sztringműveletekhez használja a `<string.h>` fejléc file-ban deklarált szabványos ANSI C sztringkezelő függvényeket.

1.23 Írjon egy olyan szabványos ANSI C ***programot***, amely a szabványos bemenetről beolvas egy teljes sort és eldönti, hogy a beolvasott szöveg palindrom-e. Palindromnak hívjuk azt a szöveget, amely visszafelé olvasva is ugyanaz, pl. "Géza, kék az ég." vagy "Indul a görög aludni.". A beolvasott szöveg kiértékelésekor nem számítanak az írásjelek, illetve a *white-space* karakterek. A szövegben esetleg előforduló kéttagú magyar mássalhangzókat egymást követő független karakterekként kell tekinteni. A magyar ékezetes betűket (Á,É,á,é stb.) is betűnek kell tekinteni, ennek elmulasztása esetén maximum 6 pont adható a feladatra. Figyeljen a kis- és nagybetűkre is!

1.24 Egy egyébként hibátlan, szabványos ANSI C ***programot*** egy olyan billentyűzeten gépeltek be, amelyen egyes speciális karakterek, amelyek létfontosságúak egy C programban, nem működtek. Ezek a karakterek: `[ ] { }`. Hogy mégis be lehessen írni a C programot, a fenti karaktereket helyettesítő karaktert, illetve karakterszekvenciákat használtak. Írjon egy olyan szabványos ANSI C programot, amely ezeket a helyettesítő karaktereket lecseréli a C-ben definiált szabványos speciális karakterekre az alábbiak szerint:

Helyettesítő karakterek: `(.  .)  (*  *)`
Szabványos karakterek: `[  ]  {  }`

Tehát pl. a `(.` sorozatot `[`-re kell cserélni. A feldolgozandó szöveg a szabványos bementen érkezik, a feldolgozott szöveget (tehát a szabványos szintaktikájú C programkódot) a szabványos kimenetre írja ki. A feldolgozandó szöveg semmiféleképpen nem fér el a számítógép memóriájában (sem a RAM-ban, sem a diszken). A feldolgozandó szöveg végét az jelzi, ha az `EOF` szimbólummal jelölt előjeles egész szám értéket olvassuk be.

A szabványos bemenetről érkező karakterek az `stdio.h` fejléc állományban deklarált `int getchar()` függvénnyel olvashatók. A szabványos kimenetre a `putchar(int)` alakú függvénnyel lehet karaktereket írni. Definiáljon a program működéséhez szükséges adattípusokat! A feldolgozandó programok kommentárokat és sztringkonstansokat nem tartalmaznak.

1.25 Írjon egy olyan szabványos ANSI C ***függvényt***, amely meghatározza sin(x) értékét az alábbi képlet segítségével, 7 tizedesjegy pontossággal!

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \ldots$$

A kívánt pontosságot akkor érjük el, ha egy új tag hozzávételével a következő összefüggés teljesül: $\left| \frac{s_n - s_{n-1}}{s_n} \right| < 10^{-7}$,

ahol s_n a $\sin x$ n tagból álló közelítése. A függvény semmilyen standard C függvényt nem használhat!

1.26 Írjon ANSI C ***programot***, amely a felhasználótól bekér a felhasználó által megadott darabszámú valós számot, amelyek egy mérés követő értékei. A program jelenítse meg a mért értékeket grafikusan egy 600 × 400-as vásznon úgy, hogy az első pont x-koordinátája 0, az utolsóé 600 legyen. Ezen túl y-irányban is feszítse rá a vászonra a "függvényt" úgy, hogy a legkisebb értékű pont y-koordinátája 0, a legnagyobbé 600 legyen. A vászont a következő függvénnyel éri el a program: `void put_pixel(int x, int y)`. A mérési értékek diszkrét pontjait nem kell összekötni egyenesekkel.

1.27 Készítsen olyan szabványos ANSI C ***programot***, amely a szabványos bemenetről érkező szöveget csupa nagybetűssé alakítva küldi a szabványos kimenetre, kivéve azokat a részeket, amelyek idézőjelek között szerepelnek. Ezeket változatlanul hagyja. Mind az idézőjelek közötti szöveg, mind az azon kívüli szöveg tartalmazhat \" karakterpárokat Az ezekben szereplő idézőjelet nem tekintjük idézet kezdetét vagy végét jelző idézőjelnek! Pl.:
Be: A "rugacs" olyan ragacs, ami \"szep "ra\"gacs".
Ki: A "rugacs" OLYAN RAGACS, AMI \"SZEP "ra\"gacs".
A szöveg végét fájlvége jelzés jelzi.

1.28 Írjon egy olyan szabványos ANSI C **programot**, amely a felhasználótól előbb egy elemszámot kér, majd ennyi valós számpárt olvas be, amelyek vektorokat írnak le! Definiáljon alkalmas struktúra típust a vektorok tárolására, és a beolvasott számpárokat ilyen elemekből álló dinamikus tömbben helyezze el! A tömböt rendezze a vektorok hossza szerinti növekvő sorrendbe a maga által írt, quicksort algoritmusú rendezőfüggvény segítségével! Az `stdlib.h`-ban definiált `qsort` függvényt természetesen nem használhatja!

1.29 Írjon szabványos ANSI C **programot**, amely bekér a felhasználótól egy maximum 500 karakter hosszú stringet! A string egy matematikai kifejezést tartalmaz, mely nemnegatív, nem kitevős alakban megadott valós számokból, összeadás (+) és szorzás (*) jelekből, valamint nyitó és záró kerek zárójelekből (()) áll. A program számítsa ki a megadott kifejezés helyes eredményét, figyelembe véve a zárójelezést, valamint azt, hogy a szorzás precedenciája magasabb az összeadásénál! A számítás eredményét írja a standard kimenetre! (Feltételezheti, hogy a felhasználó mindig helyes képletet ad meg.)

Pl.

be: (9+4)*2	ki: 26
be: 2+3*4	ki: 14
be: 8.1+(0.25+15+333333.445)*78.2*15+777	ki: 391018804.335

1.30 Írjon egy olyan szabványos ANSI C ***programot***, amely a szabványos bemeneten kapott szöveget úgy írja a szabványos kimenetre, hogy minden „`float`” szót „`double`” szóra cserél, ha az nem idézőjelek között található! A szavak betűkből, számjegyekből és aláhúzás jelekből állhatnak. Figyeljen arra, hogy pl. az floatnum azonosító nem cserélendő doublenum azonosítóra! A feladatot állapotgéppel oldja meg, ellenkező esetben maximum 5 pontot kaphat!

1.31 Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kap két stringet, melyek ASCII kódú racionális számokat tartalmaznak. Ezek első karaktere kötelezően az előjel ('+' vagy '-'), és lehet bennük tizedespont vagy tizedesvessző (mindkét forma megengedett), de nem kötelező tizedesjelnek lennie, ha egész számról van szó. A függvény foglaljon dinamikusan karaktertömböt, melybe biztosan belefér a két szám kivonásának eredménye, ezután végezze el a kivonást, az eredményt az új tömbben tárolja! Az eredménybe tizedespontot tegyen, ha az nem egész szám! A kivonás eredménye teljesen pontos legyen! (7p) Egészítse ki teljes ***programmá*** a függvényt! Hozzon létre két 200 elemű karaktertömböt, ebbe kérjen a felhasználótól számokat, számolja ki a különbségüket a függvény segítségével, az eredményt írja ki a standard kimenetre, a függvény által lefoglalt dinamikus memóriát szabadítsa fel! Pl. be: "+9.111111" és "-81,355", ki: "+90.466111"

1.32 Írjon egy olyan szabványos ANSI C ***programot***, amely a forgalomszámlálást segíti! A program a szabványos bemenetről olvas be nemnegatív egész számokat, míg 0-t nem kap. A számok különböző típusú gépjárműveket jelentenek; összesen 30 féle jármű van, 1-től 30-ig jelölve a típusukat. A program számolja meg, hogy hány konvoj haladt el az úton a számlálás időtartama alatt, és a számlálás után írja ki azokat a járműtípusokat, amelyből legalább egy konvoj érkezett! Konvojnak azt nevezzük, amikor legalább 3 azonos típusú jármű haladt el egymás után. A feladatot állapotgép segítségével oldja meg! Nem állapotgépes megoldás esetén maximum 5 pontot kaphat. Nem jár pont, ha minden gépjárműtípushoz külön állapotot akar rendelni.

1.33 Készítsen olyan szabványos ANSI C ***függvényt***, amely négy előjel nélküli egész értéket kap paraméterként: a, b, c, d; kiszámolja a/b+c/d művelet eredményét (két tört összegét), majd az eredményt törzsalakú törté (e/f) alakítja (amely már nem egyszerűsíthető tovább)! Ügyeljen arra, hogy b*d szorzatot nem számíthatja ki, mert az eredmény nem biztos, hogy elfér bármilyen egész szám típusban, viszont egyszerűsítés után e/f alakban f már meghatározható szorzással.

1.34 Adott az alábbi típusdefiníció:

```
typedef struct {
    char nev[101],neptun[7];
    int kzh[4],nzh;
} hallgato;
```

A fenti definíciót felhasználva készítsen egy olyan teljes, szabványos ANSI C ***programot***, amely megkérdezi a felhasználót, hogy maximum hány hallgató adataival kíván dolgozni, majd az alábbi függvényeket felhasználva bekéri az adatokat, és sorba rendezve kiírja az eredményt!

Készítsen **függvényt**, amely a szabványos bemenetről (alapesetben: billentyűzet) beolvassa a folyamatosan érkező, max. 100 karakteres neveket, max. 6 karakteres NEPTUN kódokat és a hozzájuk tartozó kis ZH jegyeket és nagy ZH pontszámokat! Ha egy név első karaktere az `'_'` karakter (aláhúzás karakter), akkor vége a neveknek, és már az aláhúzással kezdődő nevet sem kell feldolgozni.

Készítsen **függvényt**, amely NEPTUN kód szerinti növekvő sorrendben kilistázza a szabványos kimeneten (alapesetben: a képernyőn) azokat a hallgatókat, akik aláírást szereztek, azaz két legjobb kis ZH-juk átlaga legalább 2.0, és nagy ZH-juk legalább 15 pont.

A NEPTUN kódok összehasonlítására használhatja a könyvtári az `int strcmp(char *s1, char *s2)` függvényt, a sorba rendezéshez pedig a `void qsort(void*t, int size, int n, int (*fp)(void*,void*))` függvényt, de saját maga is megírhatja a szükséges kódot. Az `strcmp` visszatérési értéke negatív, 0 vagy pozitív a paraméterként átadott sztrigek közötti <, == vagy > reláció szerint.

1.35 Készítsen egy olyan teljes, szabványos ANSI C ***programot***, amely egy, a felhasználó által megadott, 1 és 99999 közötti természetes számot képes kiírni betűvel! 2000-ig minden számot egybeírunk, 2000 fölött az ezres és ezer alatti rész közé kötőjelet kell tenni. Példák:

```
625: hatszazhuszonot
44: negyvennegy
1975: ezerkilencszazhetvenot
8000: nyolcezer
23470: huszonharomezer-nyolcszazhetven
```

1.36 Egy kozmetikai üzletben vendégek használnak kabinokat. A kabinokban különböző hosszúságú kezelések folytathatók, egy nap összesen legfeljebb 100 darab. A kezeléseknek van kezdő időpontjuk és egy hosszuk. A kezelések csak minden negyedórában kezdhetők meg.
A nyitva tartást reprezentáló nyitó és záró időpontokat a programban megadott konstansok tárolják.
Készítsen egy olyan teljes, szabványos ANSI C ***programot***, amely az érkező vendég számára kilistázza az aznap még szabadon maradt kezdőidőpontokat annak függvényében, hogy milyen hosszú kezelésre kívánjuk a kabint lefoglalni.
A program az időpontokat percben (0 és 1440 között) tárolja, az időpontok kiírását a megszokott ÓÓ:PP formában végezze. A kiválasztott időpont foglalását tárolja el, következő foglalásokat ne engedjen úgy felvenni, hogy az előzőekkel átfedés jöjjön létre!

1.37 Készítsen ***függvényt*** C nyelven, amely paraméterként kap egy struktúratömböt (t), ezen tömb elemeinek a számát (n), egy sztringet (s) és egy egész számot (a)! A struktúrák egy 50 elemű karaktertömböt (uzi), valamint egy függvénypointert (fv) tartalmaznak. A függvénypointer void visszatérési típusú, egy darab egész értéket váró függvényre mutat. A struktúratömb elemei az uzi sztring szerint ABC sorba rendezettek. A függvény **bináris kereséssel** keresse meg az s sztringet a t tömb uzi sztringjei között, és hívja meg a hozzá tartozó függvényt a paraméterként kapott a egész értékkel! Ha az s sztring nincs a tömbben, ne tegyen semmit!

1.38 Készítsen SMS dekódoló **függvényt** szabványos ANSI C nyelven! A függvény egy forrás- és egy célsztringet kap paraméterként. A forrássztringben az SMS szövege billentyűlenyomás szerint van kódolva. Az egyes billentyűkhöz a következő karakterek tartoznak:

1	2	3	4	5	6	7	8	9	*	0	#
./,/-/?/!/1	A/B/C/2	D/E/F/3	G/H/I/4	J/K/L/5	M/N/O/6	P/Q/R/S/7	T/U/V/8	W/X/Y/Z/9	kisbetű/nagybetű/*	+/0	szóköz/#

A forrásszövegben a billentyűérték annyiszor ismétlődik, ahányadik a hozzá tartozó listában a küldeni kívánt karakter. Pl. az 1 jelentése . (pont), az 1111 jelentése ?. Az egyes dekódolandó karakterek szorosan egymást követik, kivéve, ha azonos csoportból származnak, ebben az esetben szóköz választja el őket egymástól. A szöveg karakterei alapértelmezés szerint nagybetűk.
Pl. be: „2*9999#**33 338#*8266777799993355#6334#6444663444 41 1 1”
ki: „Az EET tanszek meg mindig…”

2. Viszgafeladatok

(nagy ZH-ban nem szerepelnek)

2.1 Írjon C programot, amely

a) Beolvassa az adatok.dat bináris fájl tartalmát. A fájl data típusú struktúrákból áll. A struktúrák a következő felépítésűek:

```
typedef struct{
    int x,y;
    double value;
}data;
```

Az adatok tárolásához láncolt listát, vagy más dinamikus adatszerkezetet használjon.

b) Kiírja a beolvasott adatokat az adatok.txt nevű szöveges fájlba, mégpedig a beolvasás sorrendjével ELLENTÉTES sorrendben. A fájl első sora a következő szöveget tartalmazza (ezt is ki kell írni!):

"*Ez az adatok.dat fordított sorrendben*"

Tehát a fájl a következő felépítésű legyen:

Ez az adatok.dat fordított sorrendben

x	y	value
x	y	value
x	y	value
x	y	value

Ahol a második sor az adatok.dat utolsó elemének adatai, a harmadik sor az adatok.dat utolsó előtti elemének adatai, stb.

2.2 Olyan ékezetes magyar nyelvű szöveget tartalmazó 7 bites ASCI karaktereket tartalmazó text file-okat szeretnénk "olvashatóvá" tenni, amelyekben az ékezetes betűk kódolása a következő:

a1 – á, hasonlóan az é-re és az í-re is,
o1 – ó o2 – ö o3 – ő, hasonlóan az u-ra is,
valamint az összes nagybetűre is.

Készítsen egy olyan szabványos ANSI C programot, amely parancssor paraméterként kapja egy kódolt szöveget tartalmazó állomány nevét és annak a file-nak a nevét, amelybe a program kiírja ugyanezt a szöveget, de 8 bites extended ASCII kódokkal írt ékezetes magyar szöveg formájában. Az egyes ékezetes betűket a következő karakterkonstansok formájában adhatja meg: `'á'`, `'é'`, `'ö'`, stb.

2.3 Adott az alábbi fa struktúra, amely a Morse-ABC szimbólumait tárolja. Írjon olyan *szabványos* ANSI C ***függvényt***, amely ennek segítségével a szabványos bementeről érkező pont, kötőjel és szóközök alkotta karakterfolyamot, mint Morse-üzenetet ezen dekódoló fa segítségével "megfejti". Az egyes Morse-szimbólumokat egy space karakter választja el egymástól. A szavak végét legalább két szóköz karakter jelzi. A fa "gyökere" üres. Feltételezheti, hogy a dekódoló fa helyese van kitöltve. Ha egy nódusban az egyik rész-fa hiányzik, a megfelelő pointer mező értéke NULL. Az adatok végét EOF jelzi. Pl. a `... --- ...` karaktersorozatraz `SOS` karaktersorozatot kell szolgáltatni

```
typedef struct m {
    char ertek;
    struct m *ti, *ta;
} morse;

morse dekod;
```

Az függvény feje pedig:

```
void dekodolo(morse *gyoker);
```

2.4 Írjon olyan C függvényt, amely statisztikát készít arról, hogy a szabványos bemenetről érkező szöveges (ASCII) állományban az egyes szavak hossza milyen gyakorisággal fordul elő, azaz az 1, 2, 3, stb. karakteres szavak hányszor fordulnak elő a szövegben. A különböző hosszúságu szavak előfordulásainak a számát növekvő szóhosszak szerinti sorrendben írja ki táblázatosan, a szabványos kimenetre. A beolvasandó szavak hosszát nem ismerjük, azok bármilyen hosszúságúak lehetnek. Szónak tekinthető minden olyan karaktersor, amelyet tetszőleges számú un. *whitespace* (szóköz, újsor jel vagy tabulátor) határol. A megoldáshoz definiáljon alkalmas adatstruktúrát és magyarázza meg azt! A függvény nem használhat globális adatokat, a hívó programmal csak paraméterlistáján keresztül kommunikálhat. A függvény a lehető legegyszerűbb legyen! Ügyeljen arra, hogy a függvényből való kilépéskor minden dinamikusan lefoglalt memóriaterületet szabadítson fel!

2.5 Használja az alábbi típusdefiníciót:

```
typedef struct k{
    double x,y;
    double atlag;
    struct k *kov;
} sz, *psz;
```

Írjon egy olyan *SzamokOlvasasa* azonosítójú C függvényt, amelyben beolvassa a struktúra két adatát (*x,y*) a szabványos bemenetről. Ezek az adatok csak pozitív valós számok lehetnek, az *x* változó zérus értéke állítja le az adatok további beolvasását. Hozza létre a struktúra dinamikus elemét, majd töltse fel a beolvasott adatokkal, számítsa ki a struktúra *atlag* mezejét, ezután a dinamikusan létrejött struktúra elemeit láncoljuk a memóriába. Ha már nincs több adat, a függvény térjen vissza és visszatérési értékként a felépített lista első elemére mutató pointert szolgáltassa. A függvény deklarációja:

```
psz SzamokOlvasasa(void);
```

Írjon egy olyan *MaxAtlag* azonosítójú C függvényt, amely a paraméterlistáján megkapott első elemre mutató pointer (*e*) felhasználásával megkeresi a fenti láncolt lista a legnagyobb átlagot tartalmazó elemét. Visszatérési értékként ezt az átlagértéket szolgáltassa a függvény. Az ehhez az átlaghoz tartozó két adatot pedig a paraméterlistán keresztül adja vissz. A függvény deklarációja:

```
double MaxAtlag(psz e, double *xx, double *yy);
```

A fenti függvényeket aktivizálja az alábbiakban deklarált változókkal, nyomtassa ki a szabványos kimenetre a *MaxAtlag* által szolgáltatott értékeket és végül a lánc által lefoglalt memóriaterületeket szabadítsa fel.

```
void main(void) {
    double maxatlag, adat1, adat2;
                psz elso, p, p1;
```

2.6 A szabálytalankodó autósok hibapontjait a következő típusú elemekből álló szöveges adatállományban tartják nyilván:

```
typedef struct {
    char nev[50];
    char jogsi[15]; /* jogositvany sorszáma */
    int  pont;
    /* ... további adatok */
} elem;
```

A szükséges változásokról is ilyen elemekből álló file készül, melyben újabb szabálytalanság esetén a pontszám pozitív, illetve ha elévül egy szabálytalanság, akkor negatív. A file-ok a jogosítványok száma szerint növekvő sorrendben rendezettek. Minden file-ban az egyes struktúra mezők egy-egy külön sorban szerepelnek. Készítsen olyan teljes ANSI C ***program***ot, mely az előző állapot és a változások alapján elkészíti az aktualizált adatállományt! A program első parancssor paramétere az előző állapotot tartalmazó file neve, a második parancssor paramétere pedig a változásokat tartalmazó allomány neve. Az aktualizált állapottal írja felül az előző állapotot. Lehet, hogy valakinek nem volt még nyilvántartott szabálytalansága. Ha valakinek a pontszáma nullára csökken, akkor törölni kell a nyilvántartásból.

2.7 Készítsen egy olyan teljes ANSI C programot, amely parancssori paraméterként kap egy fájlnevet! A fájlban egy BMP formátumú, 256 színű kép van. Számolja meg, hogy melyik színből hány szerepel a képen! A bináris fájl formátuma a következő: 1078 bájt (unsigned char) fejléc, majd a fájl végéig minden bájt egy-egy képpont színe. A fájl hosszát nem ismerjük, csak azt tudjuk biztosan, hogy egyik színből sincs több, mint amennyi egy int típusú változóba belefér. Írja ki azon színek sorszámát, és azt, hogy mennyi van a képen az adott színből, amelyekből van legalább egy a képen; a legnagyobb sorszámtól a legkisebb felé haladjon a kiírásban!

2.8 Írjon egy olyan szabványos ANSI C programot, amely a szabványos bemenetről érkező szöveget a következők szerint dolgozza fel:

- A beolvasott szöveg hossza ismeretlen, de egy szó maximum 20 karakter hosszú. A szöveg szavakból, és azokat elválasztó karakterekből áll, minden szó betűkből és/vagy számjegyekből áll. Minden egyéb karakter elválasztó karakter. A szöveg végét EOF jelzi. A feladat elvégzésére alkalmas dinamikus adatszerkezeteket használjon!
- Válassza szét a szövegben a legalább 10 karakter hosszú, és az ennél rövidebb szavakat!
- Írja ki a legalább 10 karakter hosszú szavakat ABC sorba rendezve!
- Írja ki a 10 karakternél rövidebb szavakat a beérkezés sorrendjével ellentétes sorrendben!

2.9 Egy autószerviz számítógépe minden megbízásról egy-egy sort ír a SZERVIZ.TXT fájlba. Egy sor a következő adatokat tartalmazza:

```
RENDSZÁM            : 6 karakter
TULAJDONOS NEVE     : 40 karakter
FORG. ENG. SZÁMA    : 8 karakter
DÁTUM               : 11 karakter, (2006.06.19.)
ÖSSZEG       : 6 karakter (egész szám)
```

Írjon C programot, amely a SZERVIZ.TXT fájl alapján névsorban kiírja azoknak az autótulajdonosoknak a nevét, akik az átlagosnál többször javíttatták 2006 februárjában autójukat. A fájlt csak egyszer olvashatja be! Használjon dinamikus adatszerkezetet!

2.10 Egy láncolt lista a következő elemeket tartalmazza:

```
struct lanc_st {
    int     kulcs;
    float   terulet;
    struct  lanc_st *A;
};
```

A lista nagyság szerint növekvően rendezett. Írjon C függvényt, amely a paraméterként kapott láncolt listán megduplázza (ismételten felveszi) az összes, a szintén paraméterként kapott kulcsnak (egész szám) megfelelő elemet! Definiálja a függvényt! Ügyeljen arra, hogy az eljárás akkor is jól működjön, ha a lánc üres! A lánc maradjon rendezett.

2.11 Készítsen egy olyan teljes ANSI C programot, amely a szabványos bemenetről érkező szöveget dolgozza fel a következőképpen: a szöveget szavakra bontja, és a szavakat bináris fában tárolja. A bináris fa csomópontjai a szükséges pointerek mellett tehát eltárolják a szót, valamint a szó hosszát, és előfordulásának számát (minden szót egyszer tárolunk). A fa maximum 19 karakteres szavakat tárolhat. Ha egy szó több, mint 19 karakter, akkor a 20. karaktertől kezdve hagyja figyelmen kívül a többi karaktert! A szó betűkből áll, minden más karakter elválasztó karakternek tekintendő.
Miután beolvasta a szöveget, írja ki a szavakat ABC sorrendben, feltűntetve a szavak hosszát és előfordulásuk számát is!

2.12 Készítsen szabványos ANSI C programot, amely első parancssor paramétereként kap egy 1 és 9 közötti egész értéket, majd a második parancssor paramétere által megadott nevű szöveges fájl tartalmát úgy írja át a szabványos kimenetre, hogy a specifikált egész értéknél rövidebb szavakat a szó hosszának megfelelő pont (.) sorozattal helyettesíti. Szavaknak tekintjük a betűkből álló sorozatokat. Szót határol minden nem betű. A fájlt csak egyszer olvashatja be, és nem használhat munka fájlt.

2.13 Egy szöveges file kézilabda játékosok évekre bontott eredményességi adatait tartalmazza soronként az alábbi tabulált formában:

vezetéknév(10 kar.) keresztnév(10 kar.) klub(6 kar.) születési év(4 kar.) idény(4 kar) gólok száma(4 kar)

…

Az egyes mezők között egy-egy szóköz is található. Egy játékos minden idényéhez található egy ilyen sor a file-ban. Semmilyen rendezettséget nem tételezhetünk fel a file-ról.
Írjon egy teljes, szabványos ANSI C programot, amely a fenti file feldolgozása után kiírja a 2004-es idényben szerepelt 20 legfiatalabb játékos alábbi adatait: neve, klubja, 2004-ben szerzett gólok száma, összes góljainak száma. A kiírás legyen a 2004-es idény eredményessége szerint csökkenően rendezve. A program működjön helyesen akkor is, ha 20-nál kevesebb játékos volt az adatbázisban. A program a file nevét parancssor paraméterként kapja meg. (További segédfüggvény(eke)t is készíthet és használhat a program megvalósításához.) A feldolgozandó file nevét a program első parancssor paramétere tartalmazza.

2.14 Egy szöveges fájl kézilabda játékosok évekre bontott eredményességi adatait tartalmazza soronként az alábbi tabulált formában:

vezetéknév(10 kar.) keresztnév(10 kar.) klub(6 kar.) születési év(4 kar.) idény(4 kar) gólok száma(4 kar)
…

Az egyes mezők között egy-egy szóköz is található. Egy játékos minden idényéhez található egy ilyen sor a fájlban, de nem minden játékos játszott minden idényben. Semmilyen rendezettséget nem tételezhetünk fel a fájlról.
Írjon egy teljes, szabványos ANSI C ***programot***, amely a fenti fájl feldolgozása után kiírja a 2006-os idényben szerepelt 20 legfiatalabb játékos alábbi adatait: neve, klubja, 2006-ban szerzett gólok száma, összes góljainak száma. A kiírás életkor szerint növekvő sorrendben, az azonos születési évű játékosok esetében pedig névsor szerint növekvően rendezve történjen (legfeljebb 20 játékos adatait írja ki). A program működjön helyesen akkor is, ha 20-nál kevesebb játékos volt az adatbázisban. (További segédfüggvény(eke)t is készíthet és használhat a program megvalósításához.)
A fájlban szereplő játékosok nem mindegyike játszott 2006-ban. Ügyeljen arra, hogy az összes gólok száma akkor is helyes legyen, ha a fájlban a játékos valamely nem 2006-os szezonhoz tartozó adatsora előbb van, mint a 2006-os szezonhoz tartozó. A fájl méretét nem tudjuk. A fájlt csak egyszer olvashatja be. Nem használhatja a `realloc` függvényt. Definiáljon alkalmas adatszerkezeteket! A feldogozandó fájl nevét a *program első parancssor paramétere* tartalmazza. Programja kerülje a felesleges adatmozgatásokat!

2.15 Tudjuk, hogy egy egészekből álló, hosszú file-ban kevés különböző érték fordul elő. Az összes, a file-ban előforduló adat egyesével nem, a különböző értékek azonban együtt is elférnek a memóriában. Készítsen egy szabványos ANSI C ***programot***, amely kiírja a szabványos outputra, hogy mely érték hányszor fordult elő! **Először** írja ki **érték szerint csökkenő** sorrendben azokat az elemeket, melyek páratlan számszor fordultak elő, **utána** írja ki **az érték a fájlban való első előfordulása szerinti** sorrendben azokat az elemeket, amelyek páros számszor fordultak elő! A file-t csak egyszer olvashatja végig. A file egy szöveges állomány, amelyben az egyes `int` értékeket egymástól egy vagy több *whitespace* karakter választja el egymástól. Nem használhatja a `realloc` függvényt. A beolvasott értékeket csak egy példányban tárolhatja! A file nevét a program első parancssor paraméterként kapja meg. A program a kiírás végeztével szabadítsa fel az általa dinamikusan lefoglalt memóriaterületeket! (Javasolt megoldás: rendezett bináris fa.)

2.16 Írjon egy olyan szabványos ANSI C ***programot***, amely a szabványos bemenetről érkező angol nyelvű szöveget a következők szerint írja ki soronként a szabványos kimenetre:

- Ha egy sor első karaktere magánhangzó, akkor a kiírási sorrend a beolvasási sorrenddel egyezzen meg.
- Ha egy sor első karaktere mássalhangzó, akkor az illető sort megfordítva írja ki.

Az `stdin`-ről érkező sorok hosszát nem ismerjük, de azt tudjuk, hogy egy sor tartalma elfér akár a veremben (a stacken), akár normál adatmemóriában. A sorok első karaktere biztosan betű.

2.17 Adottak a következő definíciók:

```
typedef struct jj {
    char nev[50];
    char sorszam [15]; /* jogositvany sorszáma */
    int  pont;
    /* ... további adatok */
} jogsi;
```

Írjon egy olyan szabványos ANSI C ***függvényt***, amely egy tetszőleges méretű `jogsi` típusú tömböt *tetszőleges adatmezője* szerint csökkenő vagy növekvő sorrendbe tudja rendezni a *gyorsrendező algoritmus* segítségével. *Saját implementációt kell készítenie*, nem használhatja a könyvtári `qsort()` függvényt. Definiáljon további típusokat és alkalmas formális paraméterekt pl. ahhoz, hogy a rendezés szempontját a függvény aktuális bemenő paraméterei szabhassák meg.

2.18 Adott egy bináris fa, amely az alábbi elemekből épül fel:

```
typedef struct _kodfejto {
    char kar;
    struct _kodfejto *bal;
    struct _kodfejto *jobb;
} Kodfejto;
```

A fában rejtettek el egy kódszót, amit csak az találhat meg, aki rendelkezik a megfelelő térképpel. Szerencsére nekünk pont megvan ez a térkép, ami egy láncolt lista alakjában található meg a memóriában. A térkép ábrázolására a következő adatszerkezetet használtuk:

```
typedef enum {bal, jobb} Irany;
typedef struct _terkep {
    Irany irany;
    struct _terkep *kov;
} Terkep;
```

A térkép egyes elemei azt mondják meg, hogy – a gyökértől kiindulva – az aktuális csomópontból merre haladjunk tovább. A kódszó egyes karakterei sorban a meglátogatott elemekben szereplő *kar* mezőkből állnak össze. Írjon egy olyan ANSI C ***függvényt***, amely paraméterként kapja a kódfa gyökérelemére, valamint a térkép első elemére mutató pointert és visszaadja a megfejtett kódszót tartalmazó stringre mutató pointert. A kódszóról tudjuk, hogy nem hosszabb 40 karakternél.

2.19 Készítsen egy olyan teljes, szabványos ANSI C programot, amely a szabványos bemenetről (alapesetben: billentyűzet) folyamatosan érkező, max. 100 karakteres neveket beolvassa és megállapítja az összes beolvasott név átlagos hosszúságát és azt, hogy a leghosszabb név hány karakterrel hosszabb, mint az átlag. Nem tudjuk, hogy hány nevet kell beolvasnunk. Az egyes neveket, mint teljes sztringeket a `scanf("%s",nevtomb)` függvényhívással olvashatja be, ahol `nevtomb` egy, a nevek tárolására alkalmas méretű karaktertömb. Ha egy név első karaktere az `'_'` karakter (aláhúzás karakter), akkor vége a neveknek és már ezt az aláhúzással kezdődő nevet sem kell feldolgozni.

2.20 Adottak az alábbi típusdefiníciók:

```
typedef char nevtomb[100+1];
typedef struct{
    nevtomb nev;
    int jegy;
} nevlista, *nevlistamut;
```

A fentieket felhasználva készítsen egy olyan teljes, szabványos ANSI C programot, amely előbb megkérdezi a felhasználót, hogy maximum hány listaelemmel kíván dolgozni. Ezt követően a szabványos bemenetről (alapesetben: billentyűzet) beolvassa a folyamatosan érkező, max. 100 karakteres neveket és a hozzájuk tartozó jegy értékeket, és **ABC szerint növekvő sorrendben kilistázza** a szabványos kimeneten (alapesetben: a képernyőn) a neveket és a hozzájuk tartozó jegy értékeket. **Nem tudjuk, hogy hány nevet kell beolvasnunk**, de azt tudjuk, hogy maximum annyit, amennyit a felhasználó kért, és elegendő memória áll rendelkezésre ahhoz, hogy az összes nevet és nevenként még `sizeof(int)` darab byte-ot eltároljunk. Ha egy név első karaktere az `'_'` karakter (aláhúzás karakter), akkor vége a neveknek és már az aláhúzással kezdődő nevet sem kell feldolgozni. A nevek összehasonlítására használhatja a könyvtári az `int strcmp(char *s1, char *s2)` függvényt, a sorba rendezéshez pedig a `void qsort(void*t, int size, int n, int (*fp)(void*,void*))` függvényt, de saját maga is megírhatja a szükséges kódot. Az `strcmp` visszatérési értéke negatív, 0 vagy pozitív a paraméterként átadott sztrigek közötti <, == vagy > reláció szerint. Ha egy név többször fordul elő, akkor is csak egyszer szerepeljen a listában.

2.21 Adott két, egész adatokból álló szöveges file (be1.txt és be2.txt), amelyek mindegyike külön-külön növekvő sorrendbe rendezett adatokat tartalmaz. Készítsen egy olyan szabványos ANSI C **programot**, amely ezen egész értékeket további két file-ba válogatja. Azon számok, amelyek mindkét bemeneti file-ban szerepelnek, kerüljenek az első kimeneti állományba (kozos.txt). Azok a számok pedig, amelyek csak az egyik bemeneti file-ban szerepelnek, kerüljenek a második kimeneti állományba (kulon.txt). A fájlokat csak egyszer olvashatja be, és nem használhat munka fájlt. Az összes adat biztosan nem fér el a memóriában.

2.22 Adottak az alábbi típusdefiníciók:

```
typedef char nevtomb[100+1];
typedef struct nl{
nevtomb nev;
    int jegy;
    struct nl *mut;
} nevlista, *nevlistamut;
```

A fentieket felhasználva készítsen egy olyan teljes, szabványos ANSI C programot, mely a szabványos bemenetről (alapesetben: billentyűzet) beolvassa a folyamatosan érkező, max. 100 karakteres neveket és a hozzájuk tartozó jegy értékeket, és **ABC szerint növekvő sorrendben kilistázza** a szabványos kimeneten (alapesetben: a képernyőn) a neveket és a hozzájuk tartozó jegy értékeket. **Nem tudjuk, hogy hány nevet kell beolvasnunk**, de azt tudjuk, hogy elegendő memória áll rendelkezésre ahhoz, hogy az összes beolvasott adatot eltároljunk. Ha egy név első karaktere az `'_'` karakter (aláhúzás karakter), akkor vége a neveknek és már az aláhúzással kezdődő nevet sem kell feldolgozni. A nevek összehasonlítására használhatja a könyvtári `int strcmp(char *s1, char *s2)` függvényt, de saját maga is megírhatja a szükséges kódot. Az `strcmp` visszatérési értéke negatív, 0 vagy pozitív a paraméterként átadott sztrigek közötti <, == vagy > reláció szerint. Ha egy név többször fordul elő, akkor is csak egyszer szerepeljen a listában.

2.23 Írjon olyan teljes ANSI C programot, ami egy parancssori paraméterként kapott szöveges fájlban található, angol nyelvű vers szavait kiírja ABC rendben és mindegyik szó mellett feltűnteti, hogy hányszor szerepelt a szövegben. A vers összes szava biztosan elfér egyszerre a memóriában. A fájlt csak egyszer olvashatja végig.

2.24 A filter.txt nevű szöveges fájl első eleme a fájlban tárolt szavak száma, ezt követik szóközzel elválasztva maguk a szavak, amelyek mindegyike maximum 30 karakterből áll. Pl.:

6 alma körte szilva narancs kanász bóbita

Készítsen egy olyan ANSI C ***programot***, amely kiszűri a szabványos bemenetről érkező szövegből azokat a szavakat, melyek megtalálhatók a filter.txt-ben; helyettük a <censored> szöveg jelenjen meg! A cenzúrázott szöveget a szabványos kimeneten jelenítse meg! A szabványos bemenetről érkező szövegben tetszőleges hosszúságú szavak előfordulhatnak!

Pl. A körte sokkal jobb, mint a kanász almája. => A <censored> sokkal jobb, mint a <censored> almája.

2.25 Készítsen olyan szabványos ANSI C ***programot***, mely parancssori paraméterként kapja egy szöveges fájl nevét. A fájl sorai a következő adatokat tartalmazzák pontosvesszővel elválasztva: név;anyja neve;születés helye;születés ideje;TAJ szám. Egy-egy adat hossza nem haladja meg az 50 karaktert. A fájl az ország lakosságának egy évre vonatkozó összes orvoslátogatását tartalmazza, tehát egy személy akárhányszor szerepelhet benne. Olvassa be a fájlt egy TAJ szám szerint rendezett bináris fába, és számolja meg, hogy egy-egy beteg összesen hányszor volt orvosnál!

Írja ki a szabványos kimenetre azon betegek adatait TAJ szám szerint növekvő sorrendben, valamint a vizitdíj visszatérítés összegét azoknál, akiknek az jár! Azoknak jár vizitdíj visszatérítés, akik húsznál több alkalommal jártak orvosnál. Egy vizit díja 300 Ft. A beteg annyiszor 300 Ft-ot kap vissza, ahánnyal meghaladta a húszat az orvosnál tett látogatásainak száma.

2.26 Egy cég az alkalmazottai adatait az alábbi adatszerkezet segítségével tárolja el:

```
typedef srtuct alkalmazott {
    char nev[20];
    int id;
    Beosztas beosztas;
    int fonok_id;
    double fizetes;
    struct alkalmazott *bal, *jobb;
} Alkalmazott, *PAlkalmazott;
```

Írjon ANSI C ***függvényt***, amely egyetlen paramétéként megkapja a fenti bináris fa gyökerére mutató pointert, és a céges átlagfizetés 66 %-ánál kevesebbet kereső alkalmazottak fizetését megemeli 12 %-kal. A függvény visszatérő értéke az a többletköltség legyen, amit a fizetésemelés jelent a cégnek. A fa a beosztottak azonosítója (`id`) szerint van rendezve.

2.27 Írjon olyan teljes ANSI C ***programot***, ami egy parancssori paraméterként kapott szöveges fájlban található C-s függvényhívásokhoz hasonló szintaktikájú utasításokat dolgoz fel. Az utasítások szerkezete a következő:

```
utasításnév(paraméter1, paraméter2, …);
```

Az utasítások tetszőleges számú paramétert tartalmaznak, de tudjuk, hogy sem az utasításnevek, sem a paraméterek nem hosszabbak 20 karakternél. Az utasításokat, illetve a paramétereket tetszőleges számú *whitespace* karakter (szóköz, tabulátor, újsor jel) választhatja el. A program készítsen egy fésűs listát, amelynek fő listája tartalmazza az utasításneveket és ezekhez egy listában tárolja el a paramétereket. Definiálja a szükséges adatszerkezeteket is!

2.28 Adott az alábbi típusdefiníció:

```
typedef struct utelagazas{
    struct utelagazas *irany[4];
    double x,y;
    int flag;
}elag, *pelag;
```

Az `elag` típusú struktúrák labirintust építenek fel. Az `irany` tömb négy eleme a négy égtájat jelenti. Amennyiben az adott irányban nem lehet továbbhaladni, azt `NULL` pointer jelzi. A labirintus csomópontjai fa-szerkezetet alkotnak, azaz a labirintusban nem lehet körbe menni. Az `x` és az `y` a csomópont koordinátáit jelentik méterben. A kijárat koordinátái x=0, y=0. A `flag` értéke garantáltan 0.

Készítsen egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kapja a labirintus bejáratának, mint `elag` típusú csomópontnak a címét, és kiírja a bejárat és a kijárat közötti útvonalat, azaz az útba eső csomópontok `x,y` koordinátáit, méghozzá a bejárattól a kijárat felé haladva! (Ellenkező irányú kiírás esetén 3 pont levonás!) A kiírt útvonal egy csomópontot csak egyszer tartalmazhat, azaz a bejárat és kijárat közötti legrövidebb útvonalat kell kiírnia! Ha a függvény nem a legrövidebb útvonalat írja ki, a feladatra maximum 3 pont adható! A `flag` értékét a függvény szabadon megváltoztathatja, azonban kilépés előtt mindet vissza kell állítani 0-ra! Ennek elmulasztása 3 pont levonással jár! (Segédfüggvény(eke)t is írhat!)

2.29 Készítsen olyan szabványos ANSI C ***programot***, amely első parancssori paraméterként kapja egy tilosban parkolások adatait tartalmazó szöveges fájl nevét. A fájl sorai a következő adatokat tartalmazzák pontosvesszővel elválasztva: rendszám;név;cím;év;hó;nap. A rendszám max. 8 karakter, a név és a cím hossza pedig nem haladhatja meg a 100 karaktert.

a) A program írja a második parancssori paraméterként megadott fájlba azokat a sorokat, melyek egy évnél régebbi tilosban parkolást mutatnak. (5p)

b) A program írja a harmadik parancssori paraméterként megadott fájlba azokat a rendszámokat, valamint neveket és címeket pontosvesszővel elválasztva (azaz a sort dátum nélkül), amelyek esetében az adott rendszámú autó az elmúlt egy évben legalább három alkalommal tilosban parkolt. (10p)

A jelenlegi dátumot a program negyedik, ötödik és hatodik parancssori paraméterként kapja év, hó és nap sorrendben (szöveges formátumban, azaz stringként). A bemenő adatokat tartalmazó fájlt csak egyszer olvashatja!

2.30 Egy ügyvédi irodában egy adott napra beírt tárgyalásokat az alábbi struktúrán alapuló listával tartják számon:

```
typedef struct targyalas {
    char megnevezes[20];
    int kezdete;
    int idotartama;
    struct targyalas *kov;
} Targyalas;
```

A tárgyalás kezdete az éjfél óta eltelt percek számában adott, a munkaidő reggel 8 órától (480. perc) délután 5 óráig (1020. perc) tart. Írjon egy olyan szabványos ANSI C ***függvényt***, amely paraméterként kapja a fenti elemekből álló rendezetlen lista első elemére mutató pointert, egy kezdeti időt és egy időtartamot. A függvény ellenőrizze le, hogy az adott napra beszúrható-e az adott percben kezdődő és adott időtartamú tárgyalás. Ha nem, akkor adja vissza a legkorábbi olyan tárgyalásra mutató pointert, amivel ütközik, ha igen, akkor adjon vissza `NULL` pointert!

2.31 Az új elsős évfolyam adatai egy szöveges fájlban vannak, amelynek szerkezete soronként a következő:

családnév – 20 karakter
keresztnév – 20 karakter
Neptun-kód – 6 karakter
tankör szám – decimális egész, 2 számjegy
felvétel éve – decimális egész, 4 számjegy
születés éve – decimális egész, 4 számjegy
születés hónapja – decimális egész, 2 számjegy
születés napja – decimális egész, 2 számjegy
öt darab tárgy osztályzatai – 5 karakter

Készítsen statisztikai **programot** szabványos ANSI C nyelven! A program a bemenő bináris fájl nevét parancssori paraméterként vegye át! Írja ki a képernyőre, hogy az egyes érdemjegyekből hány született, azt, hogy hány hallgató átlaga tért el felfelé illetve lefelé az évfolyamátlagtól legalább 20%-kal, valamint a legjobb három átlaggal rendelkező hallgató nevét és Neptun-kódját. A fájlt csak egyszer olvashatja végig! A fájl tartalma teljes egészében elfér a memóriában.

2.32 Adottak a következő típusdefiníciók:

```
struct alagut;
typedef struct {
    struct alagut * kov;
    unsigned max,akt;
    double hossz;
}adat,*padat;
typedef struct alagut{
    padat tomb;
    unsigned n;
}alag,*palag;
```

Egy városi kábelcsatorna-hálózatot leíró program két függvényét készítjük el. Az `alagut` típusú struktúrák egy irányított gráf csomópontjai (a csatorna elágazásai). A csomópontok `tomb` adattagja a szomszédos csomópontokba vezető élek (csövek) adatait tartalmazzák, az `n` pedig a tömb elemszáma (ha `n==0`, akkor `tomb==NULL`). Az élek adatai: `kov`: a szomszédos csomópontra mutató pointer. `NULL`, ha nincs szomszéd; `max`: egy csőben hány adatkábel futhat; `akt`: jelenleg hány adatkábel fut; `hossz`: a cső hossza. Egy adott csomópontba több úton is el lehet jutni, de visszafelé nem, sem közvetlenül, sem közvetve.

a) Írjon egy olyan szabványos ANSI C **függvényt**, amely paraméterként kapja két csomópont címét, visszatérési értékke pedig a csomópontok közötti legrövidebb olyan út hossza, amelyen még lefektethető kábel a két csomópont között (`akt<max`)! Ha nincs ilyen út, -1-et adjon vissza! (6p)

b) Írjon egy olyan szabványos ANSI C **függvényt**, amely paraméterként kapja két csomópont címét, és behúz egy kábelt a két csomópont közötti útra, azaz az útvonal minden csomópontjának `akt` értékét eggyel megnöveli! Ha sikerült behúzni a kábelt, a függvény 1-et adjon vissza, ha nem, akkor 0-t! (4p)

2.33 Írjon egy olyan szabványos ANSI C **programot**, amely megnyitja az első parancssori paraméterként kapott **szövegfájlt**, és úgy írja a második parancssori paraméterként kapott szövegfájlba, minden „`int`” szót „`long`” szóra cserél, ha az nem idézőjelek között található! Figyeljen arra, hogy pl. az intel szó nem cserélendő longel szóra! A szavak betűkből, számjegyekből és aláhúzás jelekből állhatnak. A feladatot állapotgéppel oldja meg, ellenkező esetben maximum 5 pontot kaphat!

2.34 Írjon egy olyan szabványos ANSI C **programot**, amely parancssori paraméterként kapja egy szövegfájl nevét! A szövegfájl sorai az alábbi adatokat tartalmazzák, az adatok pontosvesszővel vannak elválasztva, egy adat hossza maximum 50 karakter, az irányítószám maximum 6 karakter:

Név;Irányítószám;Település;Cím;E-mail cím
Pl.: Műszaki Egyetem;1111;Budapest;Műegyetem rakpart 3;bme@bme.hu

A program hozzon létre menüt az öt tárolt adatnak megfelelően, és legyen menüelem a kilépés is! Az adott menüelem választását követően a program kérjen a felhasználótól egy megfelelő adatot (pl. név), és listázza a képernyőre **rendezve** azokat a sorokat, amelyekben a választott adat megegyezik a megadott adattal! Név esetén a rendezési feltétel az e-mail cím, a többi esetben a név. E-mail cím esetén a felhasználó a @ utáni részt adja meg, és a program ennek megfelelően szűrjön! A fájlt csak egyszer olvashatja! A megoldást lista vagy fa adatszerkezettel készítse el!

2.35 Írjon egy olyan szabványos ANSI C **programot**, amely parancssori paraméterként kapja **két szövegfájl** nevét! Az első fájl egy weboldalt leíró html szöveg.

a) A program **gyűjtse ki a fájlban található e-mail címeket**, melyeket aztán fa vagy lista adatszerkezetben tároljon (egy címet elég egyszer eltárolni)! Az e-mail címek olyan karaktersorozatok, melyek az angol ABC kis- és nagybetűiből, számokból, pontokból (.), kötőjelekből(-) és aláhúzás (_) jelekből állnak, és kötelezően tartalmaznak pontosan egy @ karaktert. Az e-mail címben nem állhat egymás mellett két pont, valamint nem állhat egymás mellett egy pont és egy @. Minden egyéb, itt fel nem sorolt karakter elválasztó karakternek minősül. Egy e-mail cím hossza maximum 100 karakter. Ha ennék hosszabb, amúgy érvényes karaktersorozatra bukkanunk, azt ne tekintsük e-mail címnek! A nem e-mail cím karaktersorozatok hossza bármekkora lehet, ezek fix hosszúságú tömbben való eltárolása súlyos hiba, minimum 5 pont levonással jár. (10p)

b) A második parancssori paraméterként kapott szövegfájl minden sora egy-egy, korábban kigyűjtött e-mail címet tartalmaz. A program olvassa be az e-mail címeket, és fésülje össze a most talált címekkel, majd mentse el ugyanebbe a fájlba az összes címet abc sorrendben, minden címet csak egyszer tároljon! (5p)

2.36 Adott a következő típusdefiníció:

```
typedef struct fa{
    unsigned kulcs;
    char szo[50];
    struct fa *bal,*jobb;
}bifa,*pbifa;
```

a) Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy `bifa` típusú elemekből álló, `kulcs` szerint **rendezett bináris fa** első elemére mutató pointert (`gy`) és egy beszúrandó, `bifa` típusú elemre vagy részfára mutató pointert (`uj`) vesz át paraméterként, és az `uj` elemet **beszúrja** a rendezett fa megfelelő helyére! A függvény adja vissza a módosított fa gyökérelemének címét! (5p)

b) Készítsen egy olyan szabványos ANSI C ***függvényt***, amely egy `bifa` típusú elemekből álló, `kulcs` szerint **rendezett bináris fa** első elemére mutató pointert (`gy`) és egy előjel nélküli egészet (`torol`) vesz át paraméterként; megkeresi a fában a `torol` paraméterrel megegyező kulcsú elemet, és ha benne van, **törli** ezt az elemet! **Csak ezt az elemet törölje**, a belőle induló két részfát a fa más elemeihez kell csatolnia, méghozzá oly módon, hogy a fa továbbra is kulcs szerint rendezett maradjon! (5p)

2.37 Egy mészárszék felvásárlói számára készítsen egy olyan szabványos ANSI C ***programot***, amely parancssori paraméterként kapja egy szövegfájl nevét, melyben versenylovak adatai találhatók! A szövegfájl sorai az alábbi adatokat tartalmazzák, az adatok pontosvesszővel vannak elválasztva, egy adat hossza maximum 50 karakter, az életkor maximum 4 karakter:

Ló neve;Életkora(hónap);Anyja neve;Apja neve;Tulajdonos neve;eredményesség
Pl.: Őrült Táncos;88;Veronika;Üstökös;Beviz Elek;-3.825

A program írja ki azon lovak nevét, amelyek elég olcsóak ahhoz, hogy a mészárszék felvásárlói megvegyék cégüknek lókolbász gyártása céljából. A felvásárlók azokat a lovakat tekintik elég olcsónak, amelyek legalább öt évesek (60 hó), és eredményességük alacsony. Azok a lovak számítanak alacsony eredményességűnek, melyekre igaz, hogy `eredményesség<min+(átlag-min)/4`, ahol `min` a fájlban szereplő, minimum öt éves, legkisebb eredményességű ló, `átlag` pedig az összes, legalább öt éves ló eredményességének átlaga. Az eredményesség értéke pozitív és negatív is lehet. A fájlt csak egyszer olvashatja!

2.38 Adott a következő típusdefiníció:

```
typedef struct {unsigned a,b; double s,v;}gps;
```

Egy fejlett GPS program számára két ***függvény***t (`epit`, `ido`) kell elkészítenie szabványos ANSI C nyelven! A `gps` struktúra az `a` és `b` azonosítójú útkereszteződés távolságát (`s`), valamint a köztük való haladás sebességét (`v`) tárolja. (Segédfüggvényeket is írhat.)

a) Definiáljon olyan struktúratípust (`gpsfa`), amely tartalmazza egy útkereszteződés azonosítóját, pointereket a maximum 6 szomszédos kereszteződésre, illetve a szomszédos kereszteződések távolságait és a haladási sebességeket! (3p)

b) Az `epit` függvény paraméterként kap egy `gps` elemekből álló tömböt (t), a tömb elemszámát (n), valamint egy útkereszteződés azonosítóját (x), és létrehozza azt a `gpsfa` elemekből álló fa adatszerkezetet, melynek gyökéreleme az x útkereszteződés, a többi eleme pedig a többi útkereszteződés. A függvény adja vissza a gyökérelem címét! A bemenő tömbben lévő útvonalak irányítottak, mindig a-ból b-be értendők, és garantált, hogy x-ből bármely csomópontba maximum egy útvonalon lehet eljutni. (8p)

c) Az `ido` függvény paraméterként kapja a `gpsfa` típusú elemekből álló fa címét (p), valamint egy útkereszteződés azonosítóját (y). és visszaadja, hogy a gyökérből mennyi idő alatt lehet eljutni y csomópontba. Ha nincs a fának y eleme, -1-et adjon!(4p)

2.39 Készítsen egy olyan C programot, amely parancssori paraméterként vesz át két fájlnevet. Mindkét fájl bináris fájl! Mindkét fájl csupa egész számot tartalmaz. A program az első fájlt olvasásra, a másodikat írásra nyissa meg, ezután tömörítse az első fájl tartalmát a másodikba a következő módszerrel: ha legalább négy egyforma érték érkezik, akkor az ismétlődést három egész érték-kel tárolja: -1 jelzi, hogy tömörített érték következik, második érték maga az egész, harmadik érték pedig a darabszám. Ha az eredeti számban -1 van, akkor azt -1,-1,darab sorozattal helyettesítse! (A példában szereplő vesszők nem részei a fájlnak!) Pl.

be: -8,9,2,4,4,1,-1,5,5,5,5,5,5,0,0,0,0,0,0,0,0,9,
ki: -8,9,2,4,4,1,-1,-1,1,-1,5,6,-1,0,8,9

2.40 Adott a következő típusdefiníció és függvénydeklarációk:

```
typedef struct{unsigned transId,buy,sum,pc; char shareId[5];}stock;
stock * stin();
void stout(stock *);
```

Készítsen tőzsdeprogramot C nyelven! A fenti függvényeket készen kapja, csak a deklarációkat kell a programban megfelelően elhelyezni. A felhasználóktól érkező tranzakciós igényeket az `stin` függvény adja folyamatosan, a tőzsdenap végét `NULL` pointer jelzi. A készítendő program feladata, hogy a beérkező igényeket egy megfelelően választott önhivatkozó dinamikus adatszerkezetben (pl. lista, fa, továbbiakban „tároló") szükség szerint tárolja és feldolgozza. A struktúra adattagjai a következők: `transId`: a tranzakció egyedi azonosítója; `buy`: 0/1 lehet, 0=eladás, 1=vétel; `sum`: mekkora összeget szán egy részvényre; `pc`: hány részvényt akar eladni/venni; `shareId`: a részvény (cég) 4 karakteres azonosítója sztringként. A program olvassa folyamatosan a tranzakciókat, és minden beolvasás után ellenőrizze, hogy van-e már olyan tranzakciós igény (esetleg több is), amely részben vagy egészben összepárosítható az új igénnyel, azaz a tranzakcióknak azonos a `shareId`-je és a `sum`-ja, továbbá ellentétes a `buy` értéke. Ebben az esetben (ha az új igénnyel több korábbi igény is párosítható, a korábbi igények beérkezési sorrendjében) végezze el a lehetséges tranzakciókat! A tranzakció elvégzése azt jelenti, hogy mindkét igény `pc` értékéből vonja le a kisebbik `pc` értéket. Legalább az egyik 0 lesz így. Amelyik 0, azt adja vissza a `stout` függvény segítségével! Ha a korábban beérkezett igény nullázódott, törölje a tárolóból! Ha az összes korábban beérkezett igény kielégítése után az új igény `pc` értéke >0, akkor vegye fel a tárolóba! A tőzsdenap végén a tárolóban maradt összes tranzakciót adja vissza a `stout` függvény segítségével, és egyúttal szabadítsa fel a memóriát!

2.41 Készítsen egy olyan szabványos ANSI C ***programot***, amely megnyitja az első parancssori paraméterként érkező nevű szövegfájlt, melyben telefon-előfizetők adatai találhatók, szűrje ki az adóssággal rendelkező ügyfelek adatait, és mentse el őket a második parancssori paraméterként érkező nevű szöveges fájlba az adósság nagysága szerinti csökkenő sorrendben! A rendezéshez használjon **maga által írt gyorsrendező algoritmust**! Ha több előfizetőnek is azonos az adóssága, őket név szerint ABC sorrendben tárolja! Ha a név is azonos, a sorrend közöttük mindegy. A bemeneti fájl formátuma a következő: első sor 1 db egész szám, az adóssággal rendelkező ügyfelek száma; a további sorok egyforma felépítésűek: *'Név;Cím;Telefonszám;Adósság összege'*. Ha nincs adósság, az összeg 0. Pl.:
Budapesti Műszaki és Gazdaságtudományi Egyetem;1111 Budapest, Műegyetem rakpart 3;4631111;0
Adós Aladár;7083 Tolnanémedi, Esernyő fasor 999;78679824065432;64213

2.42 Adottak a következő típusdefiníciók:

```
struct scella;
typedef struct{double hossz; struct scella * sz;}szomszed;
typedef struct scella{unsigned azonosito,n; szomszed * sz;}cella;
```

A cella struktúrák egy-egy mobiltelefon bázisállomás szomszédsági viszonyait írják le. Az adott cellának `n` szomszédja van, `sz` pedig `n` elemű tömb. A tömb elemei a szomszédos cellákat leíró struktúrák címét (`sz`), valamint a szomszédos cella távolságát (`hossz`) tartalmazzák. A távolságok pozitív és negatív értékek egyaránt lehetnek. Írjon függvényt, amely meghatározza két `cella` között a legrövidebb távolságot! **A függvény egy `cella` címét, valamint két `azonosito-t` kap.** Először a megadott cellából indulva keresse meg a két keresett azonosítójú cellát, majd számolja ki a köztük lévő távolságot mindkét irányban! A **függvény visszatérési értéke** a legrövidebb távolság (pozitív érték), **paramétersoron pedig** annak a cellának az azonosítóját adja a kettő közül, amelyből rövidebb úton lehet eljutni a másikba. (A-ból B-be nem feltétlenül ugyanannyi az út, mint B-ből A-ba, forgalomtól függ.) A hálózatban a távolságértékek úgy szerepelnek, hogy hurok ne alakulhasson ki, ha valaki útvonalat keres egy cellából a másikba. Ezt úgy oldották meg, hogy egy kisebb azonosítójú cellából nagyobb azonosítójú szomszédba csak pozitív hossz esetén lehet menni, egy nagyobb azonosítójúból kisebbe pedig csak negatív hossz esetén. Ez azt jelenti, hogy ha A és B szomszédok, akkor A cella B-hez tartozó hossz értéke és B cella A-hoz tartozó hosszértéke ellentétes előjelű, de nem feltétlenül azonos abszolút értékű. Az útvonalhosszak összehasonlításakor természetesen a kapott hossz abszolút értékét kell figyelembe venni.

Megoldások

Az alábbi mintamegoldások nem jelentik az egyedüli üdvözítő utat, sok esetben található jobb megoldás, még csak az sem biztos hogy hibátlanok, viszont el lehet lesni egy-két trükköt. Ha valaki nem gyakorolja önállóan a feladatmegoldást, az itt közölt megoldások fabatkát sem érnek.

Beugró

1.1

```
#include <stdio.h>
#include <math.h>

double fv(double x){return exp(3*x)-4*sin(2*x/29.0)+4*x*x*x-7.0*x-8.0;}

int main(){
    double a=0.0,b=1.0,y,x,fa,fb;
    fa=fv(a);
    fb=fv(b);
    if(fa>fb){ a=1.0; b=0.0; }
    if(fa>0||fb<0){printf("Azonos elojel!\n");return -1;}
    while(b-a>1e-8){
        x=(a+b)/2.0;
        y=fv(x);
        if(y<0)a=x;
        else b=x;
    }
    printf("%g\n",a);
    return 0;
}
```

1.2

```
unsigned tokeletes(unsigned u){
    unsigned i,s=1,n=u/2;
    for(i=2;i<=n;i++)if(u%i==0)s+=i;
    return s==u;
}
```

1.3

```
#include <stdio.h>

int main(){
    double a,b,c;
    printf("Kerek harom pozitiv szamot: ");
    scanf("%lg%lg%lg",&a,&b,&c);
    if(a<b+c&&b<a+c&&c<a+b)puts("Lehetnek haromszog oldalai");
    else puts("Nem lehetnek haromszog oldalai");
    return 0;
}
```

1.4

```
#include <stdio.h>
#include <math.h>

int main(){
    unsigned i,n,g;
    printf("Kerek egy pozitiv szamot: ");
    scanf("%u",&n);
    g=(unsigned)sqrt((double)n);
    for(i=2;i<g;i++)if(n%i==0){
        printf("Nem prim.\n");
        return 0;
    }
    printf("Prim.\n");
    return 0;
}
```

1.5

```
#include <stdio.h>

int main(){
    unsigned i=2,n;
    printf("Kerek egy pozitiv szamot: ");
    scanf("%u",&n);
    while(n>1){
        if(n%i)i++;
        else{
            printf("%u ",i);
            n/=i;
        }
    }
    return 0;
}
```

1.6

```
int prim(int p) {
    int i;
    for (i = 2; i <= sqrt(p); i++) if (!(p%i)) return 0;
    return 1;
}
```

1.7

```
void latin(void) {
     int n, i, j;
     int t[25];

     printf("Adjon meg egy szamot 1 es 25 kozott: ");
     while (1 != scanf("%d", &n) || n <= 0 || n > 25) {
           printf("Hibas adat, adja meg ujra: ");
           fflush(stdin);
     }

     for (i = 0; i < n; i++) t[i] = i + 1;

     for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
             printf("%2d ", t[(i + j) % n]);
         }
         printf("\n");
     }
}
```

1.8

```
int relativ_prim(int a, int b){
   int i, minab = (a < b) ? a : b;

   for (i = 2; i < minab/2+1; i++) if ((!a%i) && (!b%i)) return 0;
   return 1;
}
```

1.9

```
unsigned lkkt(unsigned a,unsigned b){
    unsigned i;
    for(i=a>b?a:b ; i%a!=0 || i%b!=0 ; i++);
    return i;
}
```

1.10

```
int bitteszt(double d,int n, int m){
      unsigned char * t=(unsigned char*)(&d);
      return (t[7-n]&(1<<m))?1:0;
}
```

1.11 Két megoldást is adunk, a második rekurzív függvény.

```
// 1. verzió

void szamrendszer(unsigned x,unsigned r){
      unsigned jegy=x,n=0,mul=1,temp=x/r;
      for(;temp;temp/=r){jegy=temp;n++;mul*=r;}
      for(;n>0;n--){
            printf("%d",jegy);
            x-=mul*jegy;
            mul/=r;
            jegy=x/mul;
      }
      printf("%d",jegy);
}

// 2. verzió

void dec_tagolas(unsigned n)
{
    unsigned m ;
    m=n%1000; n/=1000;
    if(n)   /*meg van mit kiirni*/
    {
       dec_tagolas(n);
       printf(" %03u",m); /*ott a space!!/
    }
    else    /*az elso csoportot vezeto 0-k nelkul */
    {
       printf("%u",m);
    }
```

1.12

```
void bitminta(int a) {
    unsigned jegyek = 1;
    int temp = 1;
    while (temp <<= 1) jegyek++;
    for (temp = jegyek; temp; temp--)
        printf("%d", a & (1 << (temp - 1)) ? 1 : 0);
}
```

1.13

```
#include <stdio.h>

int main(){
    double x,y,z,sx=0,sy=0,sz=0,cx,cy,cz;
    unsigned n;
    puts("Harom koordinatat kerek:");
    for(n=0;scanf("%lg%lg%lg",&x,&y,&z)==3;n++){
        sx+=x; sy+=y; sz+=z;
        if(n>0){
            if(cx*cx+cy*cy+cz*cz > x*x+y*y+z*z){cx=x,cy=y,cz=z;}
        }
        else {cx=x,cy=y,cz=z;}
        puts("Harom koordinatat kerek:");
    }
    if(n>0){
        printf("Sulypont: (%g,%g,%g)\nLegkozelebbi: (%g,%g,%g)\n", sx/n,
sy/n, sz/n, cx, cy, cz);
    }
    return 0;
}
```

1.14.

```
unsigned ketoszto(unsigned n){
    unsigned i,x=0;
    for(i=2;i<n;i++)if(n%i==0)x++;
    return x==2;
}
```

1.15

```
unsigned nprim(unsigned n){
    unsigned i,x,p=1,pr;
    for(x=0;x<n;x++){
        do{
            for(i=2, pr=1, p++; pr && i<p; i++)if(p%i==0)pr=0;
        }while(!pr);
    }
    return p;
}
```

1.16

```
unsigned belefer(unsigned ertek){
    unsigned h=1;
    while(ertek>h)h*=2;
    return h;
}
```

1.17

```
#include <stdio.h>

int main(){
    int i,a,b,c;
    for(i=102;i<987;i++){
        a=i%10; b=i/10%10; c=i/100;
        if(a!=b && a!=c && b!=c)printf("%d ",i);
    }
}
```

1.18

```
#include <stdio.h>

int main(){
    unsigned a,b,i;
    printf("Kérek két pozitív egészet: ");
    scanf("%u%u",&a,&b);
    if(a<1||b<1){puts("Mondom, pozitív!\n");return -1;}
    if(a>b){unsigned c=a;a=b;b=c;}
    for(i=a;i<=b;i++){
        unsigned j,p;
        for(j=2,p=1; p && j<i; j++)if(i%j==0)p=0;
        if(p)printf("%10u",i);
    }
    return 0;
}
```

1.19

```
#include <stdio.h>

int main(){
    unsigned a,b,n;
    printf("Kérek egy pozitív egészet: ");
    scanf("%u",&a);
    if(a>>31!=0){puts("Max 31 bit!\n");return -1;}
    for(n=0,b=a;b;b>>=1)if(b&1)n++;
    a=2*a+1-(n&1);
    printf("%u\n",a);
    return 0;
}
```

1.20

```
int hanyprim(int a,int b){
    int db=0,i;
    for(i=a;i<=b;i++){
        int j,p;
        for(j=2,p=1; p && j<i; j++)if(i%j==0)p=0;
        if(p)db++;
    }
    return db;
}
```

1.21

```
#include <stdio.h>

int main(){
    unsigned n,a;
    printf("Kerek egy pozitiv egesz szamot: ");
    scanf("%u",&n);
    for(a=0;n;n>>=1)a=(a<<1)+(n&1);
    printf("%u\n",a);
}
```

1.22

```
#include <stdio.h>
#include <stdlib.h>

int primtosszeg(int n){
    int i=2,s=0;
    while(n>1){
        if(n%i==0){s+=i;n/=i;}
        else i++;
    }
    return s;
}

int jegyosszeg(int n){
    int s=0;
    while(n>0){ s+=n%10; n/=10; }
    return s;
}

int main(){
    int i;
    for(i=999;primtosszeg(i)!=jegyosszeg(i);i--);
    printf("%d\n",i);
}
```

1.23

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    unsigned be,eredmeny=0,i,h16;
    scanf("%u",&be);
    for(i=0, h16=1; i<7; i++, h16*=16, be/=10) eredmeny += be % 10 * h16;
    printf("%u",eredmeny);
    return 0;
}
```

2.1

```
#include <stdio.h>

int main(){
    enum allapot{alap,str,bs,plusz} a=alap;
    int ch;
    while((ch=getchar())!=EOF){
        switch(a){
            case alap: putchar(ch); if(ch=='\"')a=str; break;
            case str: if(ch=='+')a=plusz; else{
                            putchar(ch);
                            switch(ch){
                                case '\"': a=alap; break;
                                case '\\': a=bs; break;
                            }
                        }
                       break;
            case bs: putchar(ch); a=str; break;
            case plusz: if(ch!='+')printf("+%c",ch);
                switch(ch){
                    case '\\': a=bs; break;
                    case '\"': a=alap; break;
                    default: a=str;
                }
                break;
        }
    }
}
```

2.2

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int ch;
    enum all{alap,piros,sarga,zold} a=alap;
    while((ch=getchar())!=EOF&&ch!='\n'){
        if(ch!='p'&&ch!='s'&&ch!='z'){
            printf("Hiba! Nem [p,s,z] jött, hanem %c.\n",ch);
            exit(-1);
        }
        switch(a){
            case alap:
                putchar(ch);
                if(ch=='p')a=piros;
                else if(ch=='s')a=sarga;
                else a=zold;
                break;
            case piros:
                if(ch!='p'){
                    putchar(ch);
                    if(ch=='s')a=sarga;
                    else a=zold;
                }
                break;
            case sarga:
                if(ch!='s'){
                    putchar(ch);
```

```
                    if(ch=='p')a=piros;
                    else a=zold;
                }
                break;
            case zold:
                if(ch!='z'){
                    putchar(ch);
                    if(ch=='s')a=sarga;
                    else a=piros;
                }
                break;
        }
    }
}
```

3.1

```
#include <stdio.h>

int main(){
    unsigned t[10],i;
    char ch;
    for(i=0;i<10;i++)t[i]=0;
    do{
        scanf("%c",&ch);
        if(ch>='0'&&ch<='9')t[ch-'0']++;
    }while(ch!='\n');
    for(i=0;i<10;i++)printf("%u: %u db\n",i,t[i]);
    return 0;
}
```

4.1

```
#include <stdio.h>

void felszin(double d, double h, double *fel){
    *fel=3.14*d*(h+d/2);
}

void main(){
    double d, h, felsz;
    printf("Atmero: ");
    scanf("%lg",&d);
    printf("Magassag: ");
    scanf("%lg",&h);
    felszin(d,h,&felsz);
    printf("Felszin= %g\n",felsz);
}
```

4.2

```
double mertani(){
    double t[8];
    int i;
    for(i=0;i<8;i++)t[i]=1;
    for(i=0; scanf("%lg",t+i)==1; i=(i+1)%8);
    for(i=1;i<8;i++)t[0]*=t[i];
    return pow(t[0],1.0/8.0);
}
```

4.3

```
int novekvoe(double t[],int n){
    int i;
    for(i=1;i<n;i++)if(t[i]<=t[i-1])return 0;
    return 1; // Ez már a cikluson kívül van!
}
```

4.4

```
void szamol(int * cs,int * hm){
    int ch;
    *cs=*hm=0;
    while((ch=getchar())!=EOF)if(ch=='*')(*cs)++;else if(ch=='#')(*hm)++;
}
```

4.5 (csak a függvény a feladat megoldása, a többi ajándék)

```
#include <stdio.h>

void szetvag(double d, int n, int *eg, int *to){
    int i;
    *eg=(int)d;
    d-=*eg;
    if(d<0)d=-d;
    for(i=0;i<n;i++)d*=10;
    *to=(int)d;
}

void main(){
    int a,b;
    szetvag(-3.14,4,&a,&b);
    printf("%d %d\n",a,b);
}
```

4.6 (Ilyenre azért ne számítsanak a nagy zh-n vagy a vizsgán!)

```
double terulet(double a, double b){
    return a*b*(1-4.14/4);
}
```

4.7

```
int fordit(int ertek){
      int ret=0,sig=ertek<0?-1:1;
      ertek*=sig;
      while(ertek){ret=ret*10+ertek%10; ertek/=10;}
      return ret*sig;
}
```

4.8

```
void szakaszos(unsigned szam){
      unsigned jegy=szam,n=0,mul=1,temp=szam/10;
      for(;temp;temp/=10){jegy=temp;n++;mul*=10;}
      for(;n>0;n--){
            printf("%d",jegy);
            if(n%3==0)printf(" ");
            szam-=mul*jegy;
            mul/=10;
            jegy=szam/mul;
      }
      printf("%d",jegy);
}
```

4.9

```
int relprim(int t[],int n){
    int i,j,max=t[0],oszt;
    for(i=1;i<n;i++)if(t[i]>max)max=t[i];
    max=(int)sqrt(max+0.5); // hogy ne legyen gond a kerekítésből
    i=1;
    do{
        i++;
        oszt=0;
        for(j=0;j<n;j++)if(t[j]%i==0)oszt++;
    }while(oszt<2&&i<=max);
    return oszt<2;
}
```

4.10

A fealadat specifikációja nem egyértelmű, hiszen kétdimenziós nemdinamikus tömb csak úgy adható át függvénynek, ha megmondjuk a tömb második dimenziójának méretét, négyzetes mátrixnál pedig ez nyilván megegyezik az első dimenzió méretével is, következésképp felesleges volna a feladatban specifikált n méret. Innentől fogva rajtunk áll, hogy milyen megoldást választunk, a javítóknak minden olyan helyes megoldást el kell fogadnia, ami megfeleletethető a feladatnak. A következő megoldás egy olyan pointertömböt vár, melynek elemei a mátrix soraira mutatnak.

```
int nullae(int **t,int n){
    int i;
    for(i=0;i<n;i++)if(t[i][i]!=0)return 0;
    return 1;
}
```

4.11

```
void transzponalo(double be[1241][1956],double ki[1956][1241]){
    int i,j;
    for(i=0;i<1241;i++)
        for(j=0;j<1956;j++)
            ki[j][i]=be[i][j];
}
```

4.12

```
unsigned fibi(unsigned n){
      unsigned i=1,i1=1,i2=1;
      while(i<n)i2=i1,i1=i,i=i1+i2;
      return i==n;
}
```

4.13

```
void almatrix(char *t1,char *t2,unsigned x,unsigned y,
              unsigned x0,unsigned y0,unsigned dx,unsigned dy){
      unsigned i,j;
      for(i=0;i<dy;i++)
            for(j=0;j<dx;j++)
                  t2[i*dx+j]=t1[(i+y0)*x+x0+j];

}
```

4.14

```
double gomb(double r,double * v){
    *v=4*3.14/3*r*r*r;
    return 4*3.14*r*r;
}
```

5.1

```
double * atlagalatt(double t[],int n,int *meret){
    int i;
    double atlag=0,*p;
    for(i=0;i<n;i++)atlag+=t[i];
    atlag/=n;
    for(*meret=i=0;i<n;i++)
        if(t[i]<atlag)(*meret)++;
    p=(double*)calloc(*meret,sizeof(double));
    if(!p){puts("Sikertelen memoriafoglalas");exit(-1);}
    for(*meret=i=0;i<n;i++)
        if(t[i]<atlag)p[(*meret)++]=t[i];
    return p;
}
```

6.1

```
void Copy(char cel[],char forras[]){
    int i;
    for(i=0;cel[i]=forras[i];i++);
}
```

6.2

```
void osszefuz(char cel[],char forras[]){
    int i,j;
    for(i=0;cel[i];i++);
    for(j=0;cel[i]=forras[j];i++,j++);
}
```

6.3

```
void fordit(char s[]){
    int i,n;
    for(n=0;s[n];n++);
    for(i=0,n--;i<n;i++,n--){char c=s[i];s[i]=s[n];s[n]=c;}
}
```

6.4

```
char * keres(char s[],char c){
    int i;
    for(i=0;s[i]&&s[i]!=c;i++);
    return s[i]?s+i:NULL;
}
```

6.5

```
void keres(char s[],char c){
    int i;
    for(i=0;s[i];i++)s[i]=toupper(s[i]);
}
```

6.6

```
void fesul(char s1[],char s2[],char s3[]){
    int i=0,j=0;
    while(s1[i] || s2[j]){
        if(s1[i]) s3[i+j]=s1[i++];
        if(s2[j]) s3[i+j]=s2[j++];
    }
    s3[i+j]=0;
}
```

6.7

```
void spec_strcpy(char *source, char *destination){
      unsigned i;
      for(i=0;source[i];i++)
        if(source[i]!=source[i+1])
            *destination++=source[i];
      *destination=0;
}
```

6.8

```
void torol(char s[]){
    int i=0,j=0;
    while(s[i]&&s[i]!=' ')i++;
    j=i;
    while(s[j]){
        if(s[j]==' ')j++;
        else s[i++]=s[j++];
    }
    s[i]=0;
}
```

6,9

```
void torol(char*t,char c){
      char *s=t;
      while(*s)if(*s!=c)*t++=*s++;else s++;
      *t=0;
}
```

6.10

```
void torol(char * s){
    int i=0,j;
    while(s[i]){
        if(isupper(s[i]))for(j=i;s[j]=s[j+1];j++);
        else i++;
    }
}
```

6.11

```
void fesul(char s1[],char s2[],char s3[]){
    int i=0,j=0;
    while(s1[i] || s2[j]){
        if(s1[i]) while((s3[i+j]=s1[i])&&!isspace(s1[i++]));
        if(s2[j]) while((s3[i+j]=s2[j])&&!isspace(s2[j++]));
    }
    s3[i+j]=0;
}
```

6.12

```
int jobbkeres(char s[],char c){
    int n=-1,i;
    for(i=0;s[i];i++)if(s[i]==c)n=i;
    return n;
}
```

6.13

```
void dekodolo(char*cel,char*szamok,char*betuk){
    int i;
    for(i=0;szamok[i];i+=2)
        cel[i]=betuk[(szamok[i]-'0')*10+szamok[i+1]-'0'];
}
```

6.14

```
char * tartalmaz(char * s1, char * s2) {
      int i = 0, j, talalat = 1;
      char *cime = NULL;

      for (; s1[i]; i++) {
            talalat = 1;
            for (j = 0 ; s2[j] && s1[i + j] && talalat; j++) {
                  if (s1[i + j] != s2[j]) talalat = 0;
                  else {
                        if (j == 0) cime = (s1 + i);
                  }
```

```
            }
            if (talalat) return cime;
        }
        return NULL;
}
```

6.15

```
unsigned korzetszam(char * be, char * ki){
    if(be[0]=='0' && be[1]=='6'){
        if(be[2]=='1'){
            strcpy(ki,be+3);
            return 1;
        }
        strcpy(ki,be+4);
        return (be[2]-'0')*10+be[3]-'0';
    }
    strcpy(ki,be);
    return 0;
}
```

7.1

```
void karrendez(char *t){
    int i,j,m,n;
    for(n=0;t[n];n++);
    for(i=0;i<n-1;i++){
        m=i;
        for(j=i+1;j<n;j++)if(t[j]>t[m])m=j;
        if(m!=i){char c=t[i];t[i]=t[m];t[m]=c;}
    }
}
```

7.2

```
void matrix(double t[100][100]){
    int i,j,m;
    for(i=0;i<9999;i++){
        m=i;
        for(j=i+1;j<10000;j++)
            if(t[j/100][j%100]>t[m/100][m%100])m=j;
        if(m!=i){
            double c=t[i/100][i%100];
            t[i/100][i%100]=t[m/100][m%100];
            t[m/100][m%100]=c;
        }
    }
}
```

7.3 A feladat nem mondja meg, hogy az index tömb átrendezhető-e vag sem, én ezt feltételeztem.

```
void rendez(double tomb[], int index[], int n){
    int i,j,m;
    for(i=0;i<n-1;i++){
        m=i;
        for(j=i+1;j<n;j++)if(index[j]>index[m])m=j;
        if(m!=i){
```

```
            int c=index[i];
            double d=tomb[i];
            index[i]=index[m]; index[m]=c;
            tomb[i]=tomb[m]; tomb[m]=d;
        }
    }
}
```

7.4 Ha pointertömböt kell rendezni, akkor a pointereket kell cserélgetni. Tilos a szöveget másolni, mert ebben az esetben nem garantált, hogy egyforma méretű karaktertömbökre mutatnak a pointerek, így szövegmásolás esetén kiléphetnénk a tömbből. Továbbá a pointerek mitathatnak konstans sztringekre is, amelyek nem felülírhatók.

```
void strpoirendez(char **t,int n){
    int i,j,m;
    for(i=0;i<n-1;i++){
        m=i;
        for(j=i+1;j<n;j++)if(strcmp(t[j],t[m])>0)m=j;
        if(m!=i){char * p=t[i];t[i]=t[m];t[m]=p;}
    }
}
```

7.5 Ha sztringtömböt kell rendezni, t[i] nem valódi pointer, tehát ezeket cserékgetni sem lehet, viszont nyugodtan lehet stringet másolni.

a)

```
void strtomb(char t[][80],int n){
    int i,j,m;
    for(i=0;i<n-1;i++){
        m=i;
        for(j=i+1;j<n;j++)if(strcmp(t[j],t[m])>0)m=j;
        if(m!=i){
            char p[80];
            strcpy(p,t[i]);
            strcpy(t[i],t[m]);
            strcpy(t[m],p);
        }
    }
}
```

b)

`strcmp(t[j],t[m])>0` helyett `strcmp(t[j],t[m])<0`

7.6

```
void rendez(elem * t,int n){
      int i,j,maxindex;
      for(i=0;i<n-1;i++){
            for(maxindex=i,j=i+1;j<n;j++)
                  if(t[maxindex].kulcs<t[j].kulcs)maxindex=j;
            if(maxindex!=i){
            elem temp=t[maxindex];
            t[maxindex]=t[i];
            t[i]=temp;
        }
      }
}
```

7.7 Hogy legyen qsortos megoldás is.

```
void hasonlit(const void * pa,const void * pb){
    komplex *a=(komplex*)pa, *b=(komplex*)pb;
    double ahossz=a->re*a->re+a->im*a->im,bhossz=b->re*b->re+b->im*b->im;
    if(ahossz>bhossz)return -1;
    if(ahossz<bhossz)return 1;
    return 0;
}

void rendez(komplex * t,int n){
    qsort(t,n,sizeof(komplex),hasonlit);
}
```

7.8

```
void rendez(char * s){
    int i,j,m,n;
    for(n=0;s[n];n++);
    for(i=0;i<n-1;i+=2){
        m=i;
        for(j=i+2;j<n;j+=2)if(s[j]<s[m])m=j;
        if(m!=i){char temp=s[m];s[m]=s[i];s[i]=temp;}
    }
}
```

7.9

```
void szampar(int t[],int n){
    int i,j,m;
    for(i=0;i<n-2;i++){
        m=i;
        for(j=i+2;j<n;j+=2)if(t[j]>t[m])m=j;
        if(m!=i){
            int c=t[i];   t[i  ]=t[m];   t[m  ]=c;
                c=t[i+1]; t[i+1]=t[m+1]; t[m+1]=c;
        }
    }
}
```

7.10

```
typedef struct{
    double x,y,z;
}koord;

void rendez(koord t[],int n){
    int i,j,mi;
    for(i=0;i<n-1;i++){
        mi=i;
        for(j=i+1;j<n;j++)
            if(t[i].x*t[i].x+t[i].y*t[i].y+t[i].z*t[i].z <
               t[mi].x*t[mi].x+t[mi].y*t[mi].y+t[mi].z*t[mi].z)mi=i;
        if(mi!=i){
            koord temp=t[mi]; t[mi]=t[i]; t[i]=temp;
        }
    }
}
```

8.1

```
double kulonbseg(double (*f1)(double),double(*f2)(double),
                 double a,double b,double e){
    double min=f1(a)-f2(a),d;
    for(;a<b;a+=e){
        d=f1(a)-f2(a);
        if(d<min)min=d;
        printf("%15g: %15g\n",a,d);
    }
    return min;
}
```

8.2

```
double integral(double (*fv)(double),double a,double b,double e){
    double s=fv(a)*e;
    for(a+=e;a<b;a+=e)s+=fv(a)*e;
    return s;
}
```

8.3

```
double maximum(double (*fv)(double),double a,double b,double dx){
      double max=fv(a),i,t;
      for(i=a+dx;i<=b;i+=dx)if((t=fv(i))>max)max=t;
      return max;
}
```

9.1

```
int egy_gyerekes_falevel(pbifa gyoker){
   int jgyerek, bgyerek

   if (pbifa == NULL) return 0;
   jgyerek = pbifa->jmut != NULL;
   bgyerek = pbifa->bmut != NULL;
   return (jgyerek && !bgyerek) || (bgyerek && !jgyerek) +
           egy_gyerekes_falevel(pbifa->jmut) +
           egy_gyerekes_falevel(pbifa->bmut);
}
```

9.2

```
double nagu_fa(pnfa gyoker){
   int i;
   double sum;
   if (gyoker==NULL) return 0;
   sum=gyoker->value;
   for (i = 0; i < gyoker->n; i++) sum += nagu_fa(gyoker->agak[i]);
   return sum;
}
```

9.3

```
int osszeg(ptorzs p){
      int sum=0;
      pag q;
      while(p!=NULL)for(q=p->a;q!=NULL;q=q->a)sum+=q->n;
      return sum;
}
```

9.4

```
plist torol(plist start,int x){
    int i;
    plist p1=start,p2;
    for(i=1; p1!=0&&i<=x; i++, p2=p1, p1=p1->next);
    if(!p1)return NULL;
    if(x==1){ p1=start->next; free(start); return p1; }
    p2->next=p1->next;
    free(p1);
    return start;
}
```

9.5

```
pertek keres(pertek * p,char * s){
    pertek q;
    if(p)return NULL;
    if(strcmp(p->szo),s)return p;
    q=keres(p->bal,s);
    if(q)return q;
    return keres(p->jobb,s);
}
```

9.6

```
double osszegez(pbifa gyoker){
    return gyoker ? gyoker->ertek+
                    osszegez(gyoker->bmut)+
                    osszegez(gyoker->jmut)
                  : 0.0;
}
```

9.7

```
int negygyerekes(pquadfa p){
    int s=0;
    if(p==NULL)return 0;
    s=negygyerekes(p->mut1);
    s+=negygyerekes(p->mut2);
    s+=negygyerekes(p->mut3);
    s+=negygyerekes(p->mut4);
    return s + (p->mut1 && p->mut2 && p->mut3 && p->mut4);
}
```

9.8

```
int szamol(ptrifa p){
    if(!p)return 0;
    if(p->bmut==NULL&&p->kmut==NULL&&p->jmut==NULL)return 1;
    return szamol(p->bmut)+szamol(p->kmut)+szamol(p->jmut);
}
```

9.9

```
void szintkiir(unsigned szint,pbifa gyoker){
// inorder bejárás
    if(gyoker==NULL)return;
    szintkiir(szint-1,gyoker->bmut);
    if(szint==1)printf("%g\n",gyoker->ertek);
    szintkiir(szint-1,gyoker->jmut);
}
```

9.10

```
int interval(pbifa p,double min,double max){
      if(!p)return 0;
      return interval(p->bmut,min,max)+
           interval(p->bmut,min,max)+
           (p->ertek>=min&&p->ertek<=max);
}
```

9.11

```
unsigned melyseg(pnfa gyoker) {
    int i, temp;
    int max = -1; // 0 is jó
    if (gyoker == NULL) return 0;
    for (i = 0; i < gyoker->n; i++) {
        if (max < (temp = melyseg(gyoker->mut[i]))) max = temp;
    }
    return max;
}
```

9.12

```
void intervallum(pbifa p,double a,double b){
    if(!p)return;
    intervallum(p->bal,a,b); // inorder bejárás
    if(p->atlag>=a && p->atlag<=b)printf("%s: %g\n",p->Nev,p->atlag);
    intervallum(p->jobb,a,b);
}
```

9.13

```
typedef struct csat{
    double km;
    unsigned hossz;
    struct csat *bal, *jobb;
}bifa,*pbifa;

double aradas(pbifa p){
    if(p==NULL)return 0.0;
    return aradas(p->bal)+aradas(p->jobb)+p->hossz*p->km;
}
```

9.14

```
double adatosszeg(pmagufa p,int n){
    double s=0.0;
    unsigned i;
    if(!p)return 0;
    if(n==0)return p->adat;
    for(i=0;i<p->m;i++)
        s+=adatosszeg(p->p[i],n-1);
    return s;
}
```

9.15

```
int melyseg(PTriFa p){
    int a,b,c;
    if(!p)return 0;
    a=melyseg(p->mutb);
    b=melyseg(p->mutk);
    c=melyseg(p->mutj);
    return 1+(a>b ? (a>c?a:c) : (b>c?b:c));
}
```

9.16

```
long dinnyelevel(pfa p){
    long s;
    if(p==NULL)return 0;
    s=dinnyelevel(p->left)+dinnyelevel(p->right);
    if(p->left==NULL&&p->right==NULL)s+=p->dinnye;
    return s;
}
```

9.17

```
void paros(ptrifa p){
    if(!p)return;
    paros(p->bal);
    if(p->kulcs%2==0)puts(p->szo); // inorder fabejárás
    paros(p->kozep); // ha ez az if előtt lenne, akkor alulról felfelé írná ki
    paros(p->jobb);
}
```

9.18

```
void kiir(pelem p,unsigned k){
      for(; p!=NULL && p->jobb; p=p->jobb)
        if(p->jobb->kulcs==k)
            printf("%s\n",p->szo);
}
```

9.19

```
e *bubble(e *f) {
      int changed;
      e *m, *t1, *t2;
      int first_changed;

      do {
            changed = 0;
            for (m = f; m->n != NULL; m = m->n) {
                  if (m->a > m->n->a) {
                        first_changed = 0;
                        changed = 1;
                        if (m == f) {
                              f = m->n;
                              first_changed = 1;
                        }
                        t1 = m;
                        t2 = m->n;
                        t1->n = t2->n;
                        t2->p = t1->p;
                        if (!first_changed) t1->p->n = t2;
                        t1->p = t2;
                        if (t2->n != NULL) t2->n->p = t1;
                        t2->n = t1;
                        m = t2;
                  }
            }
      } while (changed);
      return f;
}
```

9.20

```
int f4(lista * elso) {
    int adatok = 0, hszogek = 0, oldalak[3];
    lista * mozgo = elso;

    for (; mozgo != NULL; mozgo = mozgo->next) {
        oldalak[adatok % 3] = mozgo->pont;
        if (adatok % 3 == 2)
            hszogek += (oldalak[0] + oldalak[1] > oldalak[2]) &&
                       (oldalak[0] + oldalak[2] > oldalak[1]) &&
                       (oldalak[2] + oldalak[1] > oldalak[0]);
        adatok++;
    }
    if (adatok % 3 != 2) return -1; else return hszogek;
}
```

9.21

```
typedef struct stlist{
    double d;
    stlist * next;
}lista;

int monoton(lista *p){
    int db=0,nov=1;
    if(p==NULL||p->next==NULL)return 0;
    for( ; p->next; p=p->next){
        if(p->d > p->next->d){
            if(nov){ db++; nov=0; }
        }
        else nov=1;
    }
    return db;
}
```

9.22

```
void nagyito(pbfa gy){
    if(gy==NULL)return;
    nagyito(gy->bal);
    if( strncmp(gy->nev,"Nagy",4)==0 && (gy->nev[4]==0 || isspace(gy->nev[4])) )
        printf("%s: %s\n",gy->nev,gy->szam);
    nagyito(gy->jobb);
}
```

10.1

```
#include <stdio.h>

int main(){
    FILE *fp=fopen("szoveg.txt","rt");
    char sor[9][1001]; // az utolsó beolvasás sikertelen, az a 9.
    int i,j;
    if(fp==NULL){printf("Sikertelen fájlnyitás");return -1;}
    for(i=0; fgets(sor[i%9],1001,fp); i++);
    fclose(fp);
    for(j=0;j<8;j++)puts(sor[--i%9]);
    return 0;
}
```

10.2

```
#include <stdio.h>

typedef struct{double x,y;} koord;

int main(){
    FILE *fp=fopen("koordi.dat","rb");
    koord max,temp;
    double hmax,htemp;
    if(fp==NULL){printf("Sikertelen fájlnyitás");return -1;}
    if(fread(&max,sizeof(koord),1,fp)==1){
        hmax=max.x*max.x+max.y*max.y;
        while(fread(&temp,sizeof(koord),1,fp)==1){
            htemp=temp.x*temp.x+temp.y*temp.y;
            if(htemp>hmax){hmax=htemp;max=temp;}
        }
        printf("Leghosszabb: x=%g, y=%g\n",max.x,max.y);
    }
    fclose(fp);
    return 0;
}
```

10.3

```
void copy(char*n1,char*n2){
      FILE * f1,*f2;
      char c;
      f1=fopen(n1,"rb");
      f2=fopen(n2,"wb");
      while(fread(&c,1,1,f1))fwrite(&c,1,1,f2);
      fclose(f1);
      fclose(f2);
}
```

Maximumrész

1.6

```
int mondatvizsgalo(int *kijelento, int *kerdo) {
    int felkialto = 0;
    int c;
    *kijelento = *kerdo = 0;
    while (EOF != (c = getchar())) {
          switch (c) {
                case '.':
                     (*kijelento)++;
                     break;
                case '?':
                     (*kerdo)++;
                     break;
                case '!':
                     felkialto++;
                     break;
          }
    }
    return felkialto;
    /*EOF-ot kézi bevitelnél a ctrl-Z billentyu-kombinációval lehet kivál-
tani
      (utána még le kell ütni az ENTER-t)*/
}
```

1.7

```
#include <stdio.h>
void main(void){
    int aktmaxsor = 0;, aktmaxhossz = 0;
    int sor = 0, sorhossz = 0;
    int c;

    while (EOF != (c = getchar())){
        if (c != '\n') sorhossz++;
        else{
            if (sorhossz > aktmaxhossz)
                { aktmaxhossz = sorhossz; aktmaxsor = sor; }
            sorhossz = 0; sor++;
        }
    }
    printf("A %d sor %d karakterrel volt a leghosszabb."
           ,akmaxsor, aktmaxhossz);
}
```

1.10

```
double minkeres(double fv(double),double A,double B,double step){
    double minhely=A,min=fv(A),temp;
    for(;A<=B;A+=step){
        if((temp=fv(A))<min){
            minhely=A;
            min=temp;
        }
    }
    return minhely;
```

```
}
```

1.13

```
/*                  '\\'
         --------- ----> ---------
  egyéb |  ALAP |       |  bs   | '\\'
         --------- <---- ---------
         '[' |      egyéb
             |
    ']' ----˘---- egyéb
  <---- |  hiv  | ----> Hiba (-2)
         ---------
             számjegy
*/

int hivatkozas(FILE * fp){
    char c;
    int sum=0;
    enum allapot{alap,bs,hiv} all=alap;
    while(fscanf(fp,"%c",&c)==1){
        switch(all){
            case alap: if(c=='\\')all=bs;
                       else if(c=='[') all=hiv;
                       break;
            case bs:   if(c!='\\')all=alap;
                       break;
            case hiv:  if(isdigit(c))sum=sum*10+c-'0';
                       else if(c==']')return sum; // hivatkozás visszaadása
            else return -2; // nemszámjegy karakter a hivatkozásban
        }
    }
    return -1; // nincs (befejezett) hivatkozás a fájl végéig
}
```

1.14

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct l1{
    char rsz[7];
    char tul[41];
    char forg[9];
    char dat[12];
    char ossz[7];
    int jav;
    struct l1 *prev,*next;
}elem,*pelem;

void hozzaad(pelem start,pelem stop,pelem uj);

void main(){
    FILE * fp;
    elem start,stop,uj;
    pelem p;
    int atlag,n;
    char ttt[20];
```

```
    start.prev=stop.next=uj.prev=uj.next=NULL;
    start.next=&stop;
    stop.prev=&start;
    uj.jav=1; // legalább egyszer volt javíttatni, ha szerepel a fájlban
/*
    // 1. változat, csak akkor működik helyesen,
    // ha egy adatsorban nincs whitespace karakter

    if((fp=fopen("SZERVIZ.TXT","rt"))==NULL)
        {puts("Fajlnyitas sikertelen.");exit(-1);}
    while(5==fscanf(fp,"%6s%40s%8s%11s%6s",uj.rsz,
                   uj.tul,uj.forg,uj.dat,uj.ossz)){
        hozzaad(&start,&stop,&uj);
    }
*/
      // 2. változat, akkor is működik,ha van szóköz a szövegben

      if((fp=fopen("SZERVIZ.TXT","rt"))==NULL)
        {puts("Fajlnyitas sikertelen.");exit(-1);}
      while(
        fgets(uj.rsz,7,fp) && fgets(uj.tul,41,fp) &&
        fgets(uj.forg,9,fp) && fgets(uj.dat,12,fp) && fgets(uj.ossz,7,fp)){
        hozzaad(&start,&stop,&uj);
        fgets(ttt,20,fp);// a soremelést veszi ki
      }

    for(p=start.next,atlag=0,n=0;p!=&stop;p=p->next,n++)atlag+=p->jav;
    atlag/=n;

    for(p=start.next;p!=&stop;p=p->next)if(p->jav>atlag)puts(p->tul);

    for(p=start.next;p!=&stop;){ // a lista törlése
        p=p->next;
        free(p->prev);
    }
}

void hozzaad(pelem start,pelem stop,pelem uj){
    pelem p;
    uj->dat[7]=0; // így csak év és hó marad
    if(strcmp(uj->dat,"2006.02")!=0)return; // nem a keresett hónap
    uj->dat[7]='.'; // visszarakjuk a pontot
    for(start=start->next;start!=stop;start=start->next){
        int cmp=strcmp(start->tul,uj->tul);
        if(cmp==0){
            start->jav++;
            return;
        }
        if(cmp>0)break;
    }
    if((p=(pelem)malloc(sizeof(elem)))==NULL)
        {puts("Allokalas sikertelen");exit(-1);}
    *p=*uj;
    p->prev=start->prev;
    p->next=start;
    p->prev->next=p;
    p->next->prev=p;
}
```

1.18

```
void main1(int argc,char ** argv){
    int M,N,i,gyerekszam=0,azonos=0;

    if(argc<3)exit(-1);
    if(sscanf(argv[1],"%d",&N)!=1)exit(-1);
    if(sscanf(argv[2],"%d",&M)!=1)exit(-1);

    for(i=0;i<N;i++){
        int fiu=0,lany=0,j=0;
        do{
            if(random(2))lany++;else fiu++;
            gyerekszam++;
        }while(j<M&&fiu!=lany);
        if(fiu==lany)azonos++;
    }
    printf("Atlagos gyerekszam= %g\n",gyerekszam/(double)(N));
}
```

1.19

```
void main(){
    enum{alap,szokoz}all=alap;
    int c;
    while((c=getchar())!=EOF)
        if(c=='\n')putchar(c);
        else{
            switch(all){
                case alap:
                    if(isspace(c)){putchar(' ');all=szokoz;}
                    else putchar(c);
                    break;
                case szokoz:
                    if(!isspace(c)){putchar(c);all=alap;}
            }
        }
}
```

1.20

```
void szakaszos(unsigned szam){
    unsigned jegy=szam,n=0,mul=1,temp=szam/10;
    for(;temp;temp/=10){jegy=temp;n++;mul*=10;}
    for(;n>0;n--){
        printf("%d",jegy);
        if(n%3==0)printf(" ");
        szam-=mul*jegy;
        mul/=10;
        jegy=szam/mul;
    }
    printf("%d",jegy);
}
```

1.21

```
typedef struct{int betu,db;}blokk;

void main_4(){
    blokk t['Z'-'A'+1];
    int b=sizeof(t)/sizeof(blokk),n_betu=0,n_nagy=0,c,i,j;
    for(i=0;i<b;i++){t[i].db=0;t[i].betu='A'+i;}
    while((c=getchar())!=EOF){
        n_betu++;
        if(isupper(c)){t[c-'A'].db++;n_nagy++;}
    }
    for(i=0;i<b;i++)for(j=i+1;j<b;j++)
        if(t[i].db<t[j].db){blokk temp=t[i];t[i]=t[j];t[j]=temp;}
    for(i=0; i<b && t[i].db>0 ;i++)
        printf("%c\t%d\t%g\t%g\n",t[i].betu,
                t[i].db,t[i].db/(double)(n_betu),
                t[i].db/(double)(n_nagy));
}
```

1.22

```
void main_5(){
    char t[2000][31];
    int i,j,n,minindex,osszehas=0;
    for(n=0;n<2000&&fgets(t[n],31,stdin);n++);
    for(i=0;i<n;i++){
        minindex=i;
        for(j=i+1;j<n;j++){
            osszehas++;
            if(strcmp(t[minindex],t[j])>0)minindex=i;
        }
        if(minindex!=i){
            char temp[31];
            strcpy(temp,t[minindex]);
            strcpy(t[minindex],t[i]);
            strcpy(t[i],temp);
        }
    }
    for(i=0;i<n;i++)puts(t[i]);
    printf("Osszehas szam: %d\n",osszehas);
    // Kozvetlen kivalasztas, max. osszehas: n*(n-1)/2
}

// 2. verzió

#include <stdio.h>
#include <stdlib.h>

typedef enum {false, true} bool;

int main() {
    char nevek[2000][30], temp[30];
    unsigned i, darab = 0, csere = 0;
    bool rendezett;
    int c;

    do {
        printf("Adja meg a nevet: ");
        gets(nevek[darab++]); fflush(stdin);
```

```
        printf("Folytatja? [i/n] ");
        c = getchar(); fflush(stdin);
    } while ((c == 'i' || c == 'I') && darab <= 2000);

    do {
        rendezett = true;
        for (i = 1; i < darab; i++) {
            if (strcmp(nevek[i - 1], nevek[i]) == 1) {
                strcpy(temp, nevek[i]);
                strcpy(nevek[i], nevek[i - 1]);
                strcpy(nevek[i - 1], temp);
                rendezett = false;
                csere++;
            }
        }
    } while (!rendezett);

    printf("A buborek rendezes soran %d csere volt, a rendezett tomb:\n\n",
csere);

    for (i = 0; i < darab; i++) {
        printf("%s\n", nevek[i]);
    }
    return 0;
}
```

1.23

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef enum {false, true} bool;

bool _isalpha(int c) {
    char abc[] =
"aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyzAÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZ";
    unsigned meret = sizeof(abc);
    unsigned i;
    for (i = 0; i < meret; i++) if (c == abc[i]) return true;
    return false;
}

int _toupper(int c) {
    char kis_abc[] = "aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz";
    char nagy_abc[] = "AÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZ";
    unsigned meret = sizeof(kis_abc);
    unsigned i;
    for (i = 0; i < meret; i++) if (c == kis_abc[i]) return nagy_abc[i];
    return c;
}

bool palindrom(const char *str) {
    int i = 0, j = strlen(str) - 1;

    do {
        while (str[i]&&!_isalpha(str[i])) i++;
        if(!str[i])return false;
        while (j>=0&&!_isalpha(str[j])) j--;
        if(j<0)return false;
        if (_toupper(str[i]) != _toupper(str[j])) return false;
```

```
    } while (++i < --j);

    return true;
}

void main(void) {
    printf("%s", palindrom("Géza, kék az ég!")
                 ? "palindrom" : "nem palindrom");
}
```

1.24

```
#include <stdio.h>
typedef enum{alap,zarojel,pont,csillag}allapot;
int main(){
    int c;
    allapot a=alap;
    while((c=getchar())!=EOF){
        switch(a){
            case alap:
                switch(c){
                    case '(': a=zarojel; break;
                    case '.': a=pont; break;
                    case '*': a=csillag; break;
                    default:  putchar(c);
                }
                break;
            case zarojel:
                switch(c){
                    case '(': putchar('('); break;
                    case '.': putchar('['); a=alap; break;
                    case '*': putchar('{'); a=alap; break;
                    default:  putchar('('); putchar(c); a=alap;
                }
                  break;
            case pont:
                switch(c){
                    case '(': putchar('.'); a=zarojel; break;
                    case ')': putchar(']'); a=alap; break;
                    case '.': putchar('.'); break;
                    case '*': putchar('.'); a=csillag; break;
                    default:  putchar('.'); putchar(c); a=alap;
                }
                break;
            case csillag:
                switch(c){
                    case '(': putchar('*'); a=zarojel; break;
                    case ')': putchar('}'); a=alap; break;
                    case '.': putchar('*'); a=pont; break;
                    case '*': putchar('*'); break;
                    default:  putchar('*'); putchar(c); a=alap;
                }
                break;
        }
    }
    return 0;
}
```

1.29

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

const char * matek(const char * s,double * res){
      double tmp=0,mul=1;
      *res=0;
      while(*s!=0&&*s!=')'){
            if(*s=='(')s=matek(s+1,&tmp);
            else{
                  if(sscanf(s,"%lg",&tmp)!=1)
                     {printf("Hiba: nem szam (%s)\n",s);exit(-1);}
                  while(isdigit(*s)||*s=='.')s++;
            }
            mul*=tmp;
            if(*s!='*')*res+=mul;
            switch(*s){
                  case '+': mul=1;
                  case '*': s++;
                  case 0:
                  case ')': break;
                  default:  printf("Hiba: ismeretlen karakter (%c)",s[-1]);
            }
      }
      return *s==')'?s+1:s;
}

void main(){
      double res;
      matek("8.1+(0.25+15+333333.445)*78.2*15+777",&res);
      printf("%f\n",res);
}
```

1.30

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

void main(){
    enum {alap,hosszu,stringben}allapot=alap;
    int ch,db=0;
    char szo[7]="";
    while((ch=getchar())!=EOF){
        switch(allapot){
            case alap:
                if(isalnum(ch)||ch=='_'){
                    szo[db++]=ch;
                    szo[db]=0;
                    if(db>5){
                        allapot=hosszu;
                        printf("%s",szo);
                        szo[0]=db=0;
                    }
                  }
                else{
                    if(ch=='\"')allapot=stringben;
                    if(strcmp(szo,"float")==0)printf("double");
                    else printf("%s",szo);
                    szo[0]=db=0;
                    putchar(ch);
                  }
                  break;
            case hosszu:
                putchar(ch);
                if(!isalnum(ch)&&ch!='_'){
                    if(ch=='\"')allapot=stringben;
                    else allapot=alap;
                }
                break;
            case stringben:
                putchar(ch);
                if(ch=='\"')allapot=alap;
                break;
        }
    }
}
```

1.38

```
void sms(char * be,char * ki){
    char *t[11]={"+0",".,-?!1","ABC2","DEF3","GHI4","JKL5","MNO6","PQRS7","TUV8"
                 ,"WXYZ9"," #"};
    unsigned i,darab,index=0,nagybetu=1;
    char elozo;
    if(be[0]==0){ki[0]=0;return;}
    elozo=be[0];
    darab=1;
    for(i=1;elozo;i++){//nem be[i]-t figyeljük, mert így jó lesz a lezáró 0-ra is
        if(elozo==be[i])darab++;
        else{
            if(isdigit(elozo))
                ki[index++]=nagybetu?t[elozo-'0'][darab-1]:tolower(t[elozo-'0'][darab-1]);
            else if(elozo=='#')ki[index++]=nagybetu?t[10][darab-1]:tolower(t[10][darab-1]);
            else if(elozo=='*'){
```

```
                switch(darab){
                    case 1: nagybetu=0; break;
                    case 2: nagybetu=1; break;
                    default: ki[index++]='*';
                }
            }
            // egyébként szóköz volt az elozo, tehát nem írjuk ki
            elozo=be[i];
            darab=1;
        }
    }
    ki[index]=0;
}
```

2.5

```
psz SzamokOlvasasa(void) {
    psz elso = NULL, aktualis = NULL, uj = NULL;
    double x, y;

    do {
        printf("Adjon meg egy poz. valos szamot, vagy 0-t lezaraskent: ");
        while (1 != scanf("%lf", &x) || x < 0) {
            printf("Hibas adat, adja meg ujra: ");
            fflush(stdin);
        }
        printf("Adjon meg egy pozitiv valos szamot: ");
        while (1 != scanf("%lf", &y) || y <= 0) {
            printf("Hibas adat, adja meg ujra: ");
            fflush(stdin);
        }
        if (!(uj = (psz)(malloc(sizeof(psz))))) {
            printf("Memoria foglalasi hiba!\n");
            return elso;
        }
        uj->x = x;
        uj->y = y;
        uj->atlag = (x + y) / 2.0;
        uj->kov = NULL;
        if (!elso) { elso = uj; aktualis = uj;}
        else {aktualis->kov = uj; aktualis = uj;}
    } while (x);

    return elso;
}

double MaxAtlag(psz e, double *xx, double *yy) {
    double max;
    psz max_ptr = e;

    if (!e) {
        *xx = 0;
        *yy = 0;
        return 0;
    }
    else max = e->atlag;
    for(e = e->kov; e; e = e->kov) {
        if (e->atlag > max) {
            max = e->atlag;
            max_ptr = e;
        }
    }

    *xx = max_ptr->x;
    *yy = max_ptr->y;
    return max;
}
```

2.12

```
typedef enum{rovid,hosszu,egyeb} allapot;
void main(int argc,char ** argv){
    char t[10];
    FILE * fp;
    int c,n,i;
    allapot all=egyeb;

    if(argc<3)exit(-1);
    if(sscanf(argv[1],"%d",&n)!=1)exit(-1);
    if(n<1||n>9)exit(-1);
    if((fp=fopen(argv[2],"rt"))==NULL)exit(-1);

    while((c=getc(fp))!=EOF){
        switch(all){
            case egyeb:
                if(isalpha(c)){ // szó kezdődik
                    t[0]=(char)(c);
                    all=rovid;
                    i=1;
                }
                else putchar(c);
                break;
            case rovid:
                if(isalpha(c)){
                    if(i>=n-1){
                        t[i]=0;
                        printf("%s",t);
                        putchar(c);
                        all=hosszu;
                    }
                    else t[i++]=(char)(c);
                }
                else{
                    while(i--)putchar('.');
                    putchar(c);
                    all=egyeb;
                }
                break;
            case hosszu:
                putchar(c);
                if(!isalpha(c))all=egyeb;
                break;
        }
    }
}
```

2.15

```
typedef struct e1{
    int szam,n;
    struct e1 *bal,*jobb,*next;
}elem,*pelem;

pelem foglal(int ujszam){
    pelem p=(pelem)malloc(sizeof(elem));
    if(!p)exit(-1);
    p->szam=ujszam;
    p->n=1;
```

```
    p->bal=p->jobb=p->next=NULL;
    return p;
}

pelem add(pelem gyoker,int ujszam,pelem * plast){
    if(!gyoker)return *plast=foglal(ujszam);
    if(!*plast)exit(-1); //biztos van, ha eljut ide
    if(ujszam==gyoker->szam){gyoker->n++;return gyoker;}
    if(ujszam<gyoker->szam){
        if(gyoker->bal)add(gyoker->bal,ujszam,plast);
        else *plast=(*plast)->next=gyoker->bal=foglal(ujszam);
    }
    else{
        if(gyoker->jobb)add(gyoker->jobb,ujszam,plast);
        else *plast=(*plast)->next=gyoker->jobb=foglal(ujszam);
    }
    return gyoker;
}

void kiir_inorder(pelem gyoker){
    if(!gyoker)return;
    kiir_inorder(gyoker->jobb);
    if(gyoker->n%2)
        printf("%d:\t%d\n",gyoker->szam,gyoker->n);
    kiir_inorder(gyoker->bal);
}

void kiir_gyoker(pelem gyoker){
    for(;gyoker;gyoker=gyoker->next)
        if(gyoker->n%2==0)
            printf("%d:\t%d\n",gyoker->szam,gyoker->n);
}

void delfa(pelem gyoker){
    if(!gyoker)return;
    delfa(gyoker->bal);
    delfa(gyoker->jobb);
    free(gyoker);
}

void main(int argc,char ** argv){
    FILE * fp;
    int szam;
    pelem gyoker=NULL,last=NULL;

    if(argc<2)return;
    if((fp=fopen(argv[1],"rt"))==NULL)return;
    while(fscanf(fp,"%d",&szam)==1){
        gyoker=add(gyoker,szam,&last);
    }
    kiir_inorder(gyoker);
    kiir_lista(gyoker);
    delfa(gyoker);
}
```

2.16

```
#include <stdio.h>
#include <stdlib.h>

typedef enum {false, true} bool;

struct lista {
    char c;
    struct lista *kov;
};

bool mgh(int c) {
    char mgh[] = "aeiouAEIOU";
    unsigned meret = sizeof(mgh), i;
    for (i = 0; i < meret; i++) if (c == mgh[i]) return true;
    return false;
}

struct lista *elore(struct lista *eleje, char c) {
    struct lista *uj;
    if (NULL == (uj = (struct lista *)(malloc(sizeof(struct lista)))))
exit(1);
    uj->c = c;
    uj->kov = eleje;
    return uj;
}

struct lista *vegere(struct lista *eleje, char c) {
    struct lista *mozgo, *uj;
    if (NULL == (uj = (struct lista *)(malloc(sizeof(struct lista)))))
exit(1);
    uj->c = c; uj->kov = NULL;
    if (eleje == NULL) return uj;
    for (mozgo = eleje; mozgo->kov != NULL; mozgo = mozgo->kov);
    mozgo->kov = uj;
    return eleje;
}

void torol(struct lista *eleje) {
    if (eleje == NULL) return;
    torol(eleje->kov);
    free(eleje);
}

void kiir(struct lista *eleje) {
    while(eleje != NULL) {
        putchar(eleje->c);
        eleje = eleje->kov;
    }
    putchar('\n');
}

int main5(){
    int c;
    struct lista *sor = NULL;
    bool sor_eleje = true, _mgh;

    while (EOF != (c = getchar())) {
        if (sor_eleje) {
            _mgh = mgh(c);
```

```
            sor_eleje = false;
        }
        if (c == '\n') {
            kiir(sor);
            torol(sor);
            sor = NULL;
            sor_eleje = true;
        } else {
            if (_mgh) sor = vegere(sor, c);
            else sor = elore(sor, c);
        }
    }
    return 0;
}

//2. verzió

#include <stdio.h>

int elore(int c){
    if(c==EOF)return 0;
    putchar(c);
    if(c!='\n')return elore(getchar());
    return 1;
}

int hatra(int c){
    int ret;
    if(c==EOF)return 0;
    if(c!='\n'){ret=hatra(getchar());putchar(c);}
    return c=='\n'?1:ret;
}

int mgh(int c) {
    char mgh[] = "aeiouAEIOU";
    unsigned meret = sizeof(mgh), i;
    for (i = 0; i < meret; i++) if (c == mgh[i]) return 1;
    return 0;
}

void main5(){
    int c,d;
    do{// a soremelés a fordított sornál is a sor végére kell
        if(mgh(c=getchar()))d=elore(c);
        else {d=hatra(c);putchar('\n');}
    }while(d);
}
```

2.17

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef enum {false, true} bool;

typedef struct jj {
    char nev[50];
    char sorszam [15];
    int  pont;
} jogsi;
```

```c
typedef int (*comparef)(jogsi *, jogsi *);

void quicksort(jogsi *, int, int, comparef);
int divide(jogsi *, int, int, comparef);
void lista(jogsi *, int);
void change(jogsi *x, jogsi *y);

int compare_pont(jogsi *, jogsi *);
int compare_sorszam(jogsi *, jogsi *);
int compare_nev(jogsi *, jogsi *);

int main(void) {
    jogsi jogsik[] = {{"Toth Bela", "23489134", 21},
                      {"Nagy Karolina", "24234123", 34},
                      {"Soos Olga", "68745634", 6},
                      {"Alap Oszkar", "98797863", 2}};
    int meret = sizeof(jogsik) / sizeof(jogsi);

    printf("A jogositvanyok nev szerint:\n");
    quicksort(jogsik, 0, meret - 1, compare_nev);
    lista(jogsik, meret);

    printf("A jogositvanyok pont szerint:\n");
    quicksort(jogsik, 0, meret - 1, compare_pont);
    lista(jogsik, meret);

    printf("A jogositvanyok sorszam szerint:\n");
    quicksort(jogsik, 0, meret - 1, compare_sorszam);
    lista(jogsik, meret);
}

void quicksort(jogsi *A, int p, int r, comparef cmp) {
    int q;
    if (p < r) {
        q = divide(A, p, r, cmp);
        quicksort(A, p, q, cmp);
        quicksort(A, q+1, r, cmp);
    }
}

int divide(jogsi *A, int p, int r, comparef cmp) {
    jogsi x = A[p];
    int i = p - 1;
    int j = r + 1;
    while (1) {
        do j--; while (cmp(&A[j], &x) == 1);
        do i++; while (cmp(&A[i], &x) == -1);
        if (i < j) change(A + i, A + j);
        else return j;
    }
}

void lista(jogsi *t, int size) {
    int i;

    for (i = 0; i < size; i++) {
        printf("%s %s %d\n", t[i].nev, t[i].sorszam, t[i].pont);
    }
    printf("\n");
}
```

```
void change(jogsi *x, jogsi *y) {
    jogsi tmp;
    if (x == y) return;
    tmp = *y;
    *y = *x;
    *x = tmp;
}

int compare_pont(jogsi *j1, jogsi *j2) {
    if (j1->pont < j2->pont) return -1;
    else if (j1->pont == j2->pont) return 0;
    else return 1;
}

int compare_sorszam(jogsi *j1, jogsi *j2) {
    return strcmp(j1->sorszam, j2->sorszam);
}

int compare_nev(jogsi *j1, jogsi *j2) {
    return strcmp(j1->nev, j2->nev);
}
```

2.20

```
#include <stdio.h>
#include <stdlib.h>

typedef char nevtomb[100+1];
typedef struct{
    nevtomb nev;
    int jegy;
} nevlista, *nevlistamut;

int hasolnlit(void*a,void*b){
    return strcmp(((nevlistamut)a)->nev,((nevlistamut)b)->nev);
}

int main(){
    unsigned m,n,i;
    int c;
    nevlistamut p;

    printf("max. hany elemmel akar dolgozni? ");
    scanf("%u",&m);
    p=(nevlistamut)calloc(m,sizeof(nevlista));
    if(p==NULL){puts("Memoriafogllasi hiba");return -1;}

    i=0;
    do{
        puts("Kerem a nevet es a jegyet!");
        scanf("%s %d",p[i].nev,&p[i].jegy);
        puts("Akar ujabb adatot felvenni?(I/N)");
        c=getchar();
    }while(i++<m&&(c=='i'||c=='I'));
    n=i;

    qsort(p,n,sizeof(nevlista),hasonlit);

    printf("A liata:\n");
    for(i=0;i<n;i++){printf("-30%s %d\n",p[i].nev,p[i].jegy);
```

```
    free(p);
}
```

2.28

```
int kijelol(pelag p){
    if(!p)return 0;
    if(p->x==0&&p->y==0){
        p->flag=2;
        return 1;
    }
    if(kijelol(p->irany[0])||kijelol(p->irany[1])
        ||kijelol(p->irany[2])||kijelol(p->irany[3])){
        p->flag=1;
        return 1;
    }
    return 0;
}

void kiir(pelag p){
    if(!p||!p->flag)return;
    printf("%g\t%g\n",p->x,p->y);
    if(p->flag==1){
        kiir(p->irany[0]);
        kiir(p->irany[1]);
        kiir(p->irany[2]);
        kiir(p->irany[3]);
    }
    p->flag=0;
}

void labiut(pelag p){
    if(!p)return;
    kijelol(p);
    kiir(p);
}
```

2.36

```
pbifa beszur(pbifa gy,pbifa uj){
    pbifa gyt=gy;
    if(!gy||!uj)return uj;
    while(gyt->bal!=uj||gyt->jobb!=uj){
        if(uj->kulcs<=gyt->kulcs&&gyt->bal)gyt=gyt->bal;
        else if(uj->kulcs>=gyt->kulcs&&gyt->jobb)gyt=gyt->jobb;
        else if(uj->kulcs<=gyt->kulcs)gyt->bal=uj;
        else gyt->jobb=uj;
    }
    return gy;
}

pbifa torol(pbifa gy,unsigned kulcs){
    pbifa gyt=gy,apa=NULL;
    if(!gy)return NULL;
    while(1){
        if(kulcs==gyt->kulcs){
            pbifa p;
            if(gyt->bal){p=gyt->bal; beszur(p,gyt->jobb);}
            else        {p=gyt->jobb;beszur(p,gyt->bal);}
            free(gyt);
```

```
            if(apa){//bár töröltük, a pointer értéke még jó
                  if(gyt==apa->bal)apa->bal=p;
                  else apa->jobb=p;
            }
            else gy=p; //gy==gyt, a legtetején vagyunk
            return gy;
        }
        else if(kulcs<gyt->kulcs&&gyt->bal){apa=gyt;gyt=gyt->bal;}
        else if(kulcs>gyt->kulcs&&gyt->jobb){apa=gyt;gyt=gyt->jobb;}
        else return gy;//nincs ilyen elem
    }
}
```

2.37

```
#include <stdio.h>
#include <stdlib.h>

void error(const char * s){
    printf("Hiba: %s\n",s);
    exit(-1);
}

typedef struct le{
    char nev[51];
    double ere;
    struct le * next;
}list,*plist;

const char * getblokk(const char * s,unsigned n){
// n=0-5
    unsigned index=0;
    while(n&&s[index])if(s[index++]==';')n--;
    if(s[index]==0)error("Hiányzik a sor vége");
    return s+index;
}

double getnum(const char * s,unsigned n){
    return atof(getblokk(s,n));
}

void copyblokk(const char * s,unsigned n,char * d){
    s=getblokk(s,n);
    while(*s!=';'&&*s)*d++=*s++;
    *d=0;
}

plist add(plist p,char *s){
    plist q=(plist)malloc(sizeof(list));
    if(!q)error("memória foglalás");
    copyblokk(s,0,q->nev);
    q->ere=getnum(s,5);
    q->next=p;
    return q;
}

void kiir_torol(plist p,double limit){
    while(p){
        plist q=p->next;
        if(p->ere<limit)printf("%s\n",p->nev);
        free(p);
        p=q;
```

```
    }
}

int main(int nn,char ** para){
    FILE * fp;
    char temp[255];
    plist gyoker=NULL;
    double min,atlag=0,dt;
    unsigned db=0;
    if(nn<2)error("Túl kevés paraméter");
    if((fp=fopen(para[1],"rt"))==NULL)error(para[1]);
    while(fgets(temp,255,fp))if(getnum(temp,1)>=60){
        atlag+=dt=getnum(temp,5);
        if(!gyoker||min>dt)min=dt;
        gyoker=add(gyoker,temp);
        db++;
    }
    atlag/=db;
    kiir_torol(gyoker,min+(atlag-min)/4);
}
```

2.38

```
typedef struct xfa{
    unsigned a;
    struct xfa *p[6];
    double s[6],v[6];
}gpsfa,*pgpsfa;

void seged(pgpsfa gyoker,gps t[],unsigned n){
    unsigned i,db=0;
    if(!gyoker)return;
    for(i=0;i<6;i++)gyoker->p[i]=NULL;
    for(i=0;i<n&&db<6;i++)if(t[i].a==gyoker->a){
        gyoker->p[db]=(pgpsfa)malloc(sizeof(gpsfa));
        if(gyoker->p[db]==0){printf("Alloc failed\n");exit(-1);}
        gyoker->p[db]->a=t[i].b;
        gyoker->s[db]=t[i].s;
        gyoker->v[db]=t[i].v;
        db++;
    }
    for(i=0;i<db;i++)seged(gyoker->p[i],t,n);
}

pgpsfa epit(gps t[],unsigned n,unsigned x){
    pgpsfa gyoker=(pgpsfa)malloc(sizeof(gpsfa));
    if(gyoker==0){printf("Alloc failed\n");exit(-1);}
    gyoker->a=x;
    seged(gyoker,t,n);
    return gyoker;
}

double ido(pgpsfa p,unsigned y){
    unsigned i;
    double t;
    if(!p)return -1;
    if(p->a==y)return 0; //itt vagyunk
    for(i=0;i<6;i++)if((t=ido(p->p[i],y))!=-1)return t+p->s[i]/p->v[i];
    return -1;
}
```

2.41

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    char sor[100];
    unsigned osszeg;
}ados;

unsigned adossag(char * s){
// kiolvassa az adósság értékét a sorból
    unsigned pv=0;
    while(pv<3)if(*s++==';')pv++; // a 3. ; után van az érték
    if(sscanf(s,"%u",&pv)!=1)exit(-1);
    return pv;
}

void qs(ados * t,int meret){
    int left=0,right=meret-1,med,csere;
    if(meret<2)return;
    med=(t[right].osszeg+t[left].osszeg)/2;
    do{
        while(t[left].osszeg>med)left++;//fordított sorrend => fordított kacsacsőr
        while(t[right].osszeg<med)right--;
        csere=1;
        if(t[left].osszeg==med && t[right].osszeg==med
           && strcmp(t[left].sor,t[left].sor)<0)csere=0;
        if(csere && right>=left){ // másodlados sorbarendezés ABC szerint!
            ados temp=t[left];t[left]=t[right];t[right]=temp;
            left++; right--;
        }
    }while(right>=left);
    qs(t,right+1);
    qs(t+left,meret-left);
}

int main(int argc,char ** argv){
    ados * tomb;
    unsigned darab,i;
    FILE * fp;
    char s[100];

    if(argc<3)return -1;
    fp=fopen(argv[1],"r");
    if(fp==NULL)return -1;
    if(fgets(s,100,fp)==NULL)return -1;
    if(sscanf(s,"%u",&darab)!=1)return -1;
    tomb=(ados*)calloc(darab,sizeof(ados));
    if(tomb==NULL)return -1;
    for(i=0;i<darab;){
        if(fgets(tomb[i].sor,100,fp)==NULL)return -1;
        tomb[i].osszeg=adossag(tomb[i].sor);
        if(tomb[i].osszeg>0)i++;
    }
    fclose(fp);
    qs(tomb,darab);
    fp=fopen(argv[2],"w");
    if(fp==NULL)return -1;
    for(i=0;i<darab;i++)fputs(tomb[i].sor,fp);
    fclose(fp);
}
```

2.42

```
cella * keres(cella * gy,unsigned azonosito){
// ha az aktuálisnál nagyobbat keresek, akkor pozitív hosszon szabad menni
// ha az aktuálisnál kisebbet keresek, akkor csak negatív hosszon szabad menni
    cella * talalt=NULL;
    unsigned i;
    if(gy==NULL)return NULL;
    if(gy->azonosito==azonosito)return gy;
    for(i=0; talalt!=NULL && i<gy->n; i++)
        if((gy->azonosito<azonosito && gy->sz[i].hossz>0) ||
           (gy->azonosito>azonosito && gy->sz[i].hossz<0))
               talalt=keres(gy->sz[i].sz,azonosito);
    return talalt;
}

double minhossz(cella * gy,unsigned azonosito){
// -1-et ad vissza, ha az adott útvonalon nem lehet eljutni
// ha van útvonal, akkor pozitív hosszat ad
    double min=-1,temp;
    unsigned i;
    if(gy==NULL)return -1;
    if(gy->azonosito==azonosito)return 0;
    for(i=0; i<gy->n; i++)
        if(gy->azonosito<azonosito && gy->sz[i].hossz>0){
            if(gy->sz[i].sz->azonosito==azonosito)return gy->sz[i].hossz;
            temp=minhossz(gy->sz->sz,azonosito);
            if(temp>0&&(min<0||min>temp+gy->sz[i].hossz))
                min=temp+gy->sz[i].hossz;
        }
        else if(gy->azonosito>azonosito && gy->sz[i].hossz<0){
            if(gy->sz[i].sz->azonosito==azonosito)return -gy->sz[i].hossz;
            temp=minhossz(gy->sz->sz,azonosito);
            if(temp>0&&(min<0||min>temp-gy->sz[i].hossz))
                min=temp-gy->sz[i].hossz;
        }
    return min;
}


double fenyevtavolsag(cella * gy, unsigned az1, unsigned az2, unsigned *
visszaz){
    cella *p1,*p2;
    unsigned i;
    double h1,h2;

    if(az1>az2){i=az1;az1=az2;az2=i;}
    p1=keres(gy,az1);
    p2=keres(gy,az2);
    h1=minhossz(p1,az2);
    h2=minhossz(p2,az1);
    if(h1<0||h2<0){
        printf("Nincs útvonal a cellák között.\n");
        exit(-1);
    }
    if(h1<h2){
        *visszaz=az1;
        return h1;
    }
    *visszaz=az2;
    return h2;
}
```