BUDAPESTI MŰSZAKI EGYETEM
MÉRNÖKTOVÁBBKÉPZŐ INTÉZET

Benkő Tiborné – Hegedűs András – Benkő László

IBM PROGRAMOZÁSA TURBO PASCAL NYELVEN PÉLDATÁR I.

BUDAPEST
1991

5319

200,– Ft

BUDAPESTI MŰSZAKI EGYETEM
MÉRNÖKTOVÁBBKÉPZŐ INTÉZET

Benkő Tiborné - Hegedűs András - Benkő László

IBM PROGRAMOZÁSA TURBO PASCAL NYELVEN PÉLDATÁR I.

KÉZIRAT

BUDAPEST
1991

A kéziratot ellenőrizte:
Poppe András

ISSN 0865 - 3313
ISBN 963 431 744-8

A kiadásért felelős:
a Budapesti Műszaki Egyetem Mérnöktovábbképző Intézetének igazgatója
Megrendelve: 1991. február. Megjelent: 1991. március. Példányszám: 1018
Készült: kisofszet eljárással, az MSZ 5601-59
és az MSZ 5602-55 szabvány szerint

Nyomtatta és kötötte a Dabasi Nyomda
Felelős vezető: Bálint Csaba igazgató
Munkaszám: 91-0256

TARTALOMJEGYZÉK

BEVEZETÉS

A példatár I. kötete a korábban megjelent Benkő Tiborné - Hegedüs András : IBM PC programozása Turbo Pascal nyelven és Benkő Tiborné - Benkő László : Objektum-orientált programozás Turbo Pascal-ban 5.5 változat c. könyvek ismeretére támaszkodva készült.

A jegyzet első fejezete a Pascal nyelv legfontosabb utasításainak jobb megértésére nyújt példaprogramokat, melyekhez nem szükséges magasszintű matematika. Ennek a fejezetnek tanulmányozását azoknak javasoljuk, akik még bizonytalanok az utasítások teljes megértésében, illetve még önállóan nem készítettek programot.

A második fejezet részletesen foglalkozik a unitok használatával, amely az ún. saját könyvtár készítésére biztatja az olvasót. Egy játékprogram megszerkesztésén keresztül látható, hogy az önálló, de logikailag összetartozó feladatrészekből állnak elő a unitok és ezek felhasználásával készül el a főprogram.

Rendező és kereső algoritmusok gyűjteménye a harmadik fejezet, amely példákon keresztül bemutatja a működésüket. Nem szorítkozik csak a memóriában történő rendezésre, hanem véletlen elérésű file-ok tartalmának rendezésére is szolgáltat egy rendező módszert.

A negyedik fejezet mérési adatok kiértékelésére menüvezérelt programot ismertet, amely nemcsak statisztikát számol, hanem többfajta grafikus megjelenítő módszert is ajánl az adatok ábrázolására. Adott év havi statisztikai adatainak ábrázolását síkbeli és térbeli oszlopdiagramok teszik szemléletessé.

Az ötödik fejezet numerikus módszerekkel foglalkozik oly módon, hogy komplex mátrix determinánsa, sajátértéke és komplex egyenletrendszer megoldására (Gauss elimináció és LU dekompozíció) menüvezérelt programcsomagot ismertet.

A függelékben az ismertetett feladatok teljes listája megtalálható.

1. VEGYES GYAKORLÓ FELADATOK

Ebben a fejezetben viszonylag rövid feladatokat gyűjtöttünk össze a Turbo Pascal nyelv alapismereteinek begyakorlására. A feladatok témája változatos, megértésükhöz nem szükséges a középiskolainál magasabb szintű matematika. A tájékozódás megkönnyítésére az egyes pontok címe után zárójelben feltüntettük az érintett Turbo Pascal ismereteket.

A programok rövidsége lehetővé teszi, hogy - a későbbi hosszabb programokkal ellentétben - rögtön a feladat kitűzése után megadjuk egy lehetséges megoldás teljes listáját. Ezután általában néhány megjegyzés, magyarázat következik.

Az első feladatok kezdő programozóknak készültek, haladók ezeket nyugodtan ugorják át. A fejezet második részében találhatók azok a példák, amelyek vagy a probléma felvetése vagy a megoldás számítástechnikai megvalósítása miatt haladók érdeklődésre is számot tarthatnak.

A tizedik feladatig kerültük az alprogramok (függvények és eljárások) használatát, utána viszont nagy súlyt fektettünk ennek a gyakran gondot okozó technikának a begyakorlására.

1.1. Téglatest felszíne és térfogata

(beolvasás, kiíratás, alapműveletek, jelkészlet)

Irjunk programot, amely az élek hosszának lekérdezése után kiszámítja és kiírja a téglatest felszínét és térfogatát.

```
program tegla;

var
  a, b, c : real;
  felszin, terfogat : real;

begin
  writeln('Adja meg a téglatest oldalait!');
  write('a: ');
  readln(a);
  write('b: ');
  readln(b);
  write('c: ');
  readln(c);
  felszin := (a*b + b*c + a*c) * 2;
  terfogat := a*b*c;
  writeln;
  writeln('Felszine: ', felszin:8:2, '  térfogata: ', terfogat:8:2);
end.
```

Programunk hiányossága, hogy nem ellenőrzi a bemenő adatokat, így elfogad negatív értéket is, és a mértékegységek következetes használata is a program felhasználójára van bízva. A későbbi feladatokban több példa is található arra, hogyan kell a bemenő adatokat ellenőrizni, erre az első feladatra a lehető legegyszerűbb megoldást akartuk adni.

Másik probléma, hogy a kiíratási formátummal megszabtuk, hogy az eredményben kettőnél több tizedesjegy nem jelenhet meg. Ha szükséges, emeljük fel a tizedesjegyek számát:

```
writeln('Felszine: ', felszin:12:6, '  térfogata: ', terfogat:12:6);
```

vagy használjuk az általánosabb (de nehezebben olvasható) exponenciális alakú kiiratást:

```
writeln('Felszine: ', felszin, '  térfogata: ', terfogat);
```

Az ékezetes magyar betük nem tartoznak a Pascal nyelv jelkészletébe, így azonosítókban való szerepeltetésük nem megengedett. Ugyanakkor karakterláncokban, például kiíratandó szövegekben nyugodtan használhatjuk őket, mint a fenti példában is. Bevitelük azonban elég nehézkes (vagy egy segédprogramot igényelnek, vagy a funkcionális billentyűzeten kell begépelni az ASCII kódjukat az ALT billentyű lenyomása közben), ezért a további programokban többnyire kerüljük a használatukat.

1.2. Az első 100 szám négyzetének összege

(ciklus, alapműveletek, kiíratás, egész típusok)

Írjunk programot, amely kiszámítja és a képernyőre írja az első 100 természetes szám négyzetének összegét!

```
program negyzet_osszeg;
const
  n = 100;
var
  i, negyzet : integer;
  osszeg : longint;

begin
  osszeg := 0;
  for i := 1 to n do
    begin
      negyzet := i * i;
      osszeg := osszeg + negyzet;
    end;
  writeln('Az elsö ', n, ' szám négyzetének összege: ', osszeg);
end.
```

A programot lefuttatva a következő üzenetet kapjuk a képernyőn:

Az elsö 100 szám négyzetének összege 338350

Programunk rövidíthető, ha a ciklusmagban lévő két utasítást összevonjuk:

```
osszeg := osszeg + i * i;
```

így a **begin - end** utasításpár is elhagyható. Mi azért választottuk a hosszadalmasabb megoldást, hogy programunk áttekinthetőbb legyen, és jobban elmagyarázhassuk a tudnivalókat a különböző egész típusokról.

Az *osszeg* nevű változót azért deklaráltuk **longint**-nek, mert az **integer** típusú változókba elhelyezhető legnagyobb szám 32767. Ezt az értéket *osszeg* már n=100 esetén is túllépi. Ilyenkor - sajnálatos módon - hibajelzés nélkül túlcsordul, tehát hamis értéket fog tartalmazni, amit nem is mindig könnyű észrevenni.

Ennél is szomorúbb dolgokat fogunk tapasztalni, ha *n* értékét 200-ra növeljük. Mivel 200 négyzete 40.000, *negyzet* típusát **longint**-re módosítjuk, majd lefuttatjuk a programot. Az eredmény:

Az elsö 200 szám négyzetének összege 1441516

Ha utána számolunk, kiderül, hogy ez az első pillantásra hihető eredmény hamis. Ezt bizonyítja és a hiba okára is rávilágít, ha az utolsó ciklusiépés után kiíratjuk vagy a **WATCH** hibakeresővel megvizsgáljuk *negyzet* értékét: -25536! A Turbo Pascal ugyanis i*i értékét az *i* számára fenntartott helyen számítja ki, ahol túlcsordulás következik be, amin már nem segít, hogy ez a hamis eredmény egy **longint** típusú változóba kerül. Ezt a sok problémát okozó furcsa Turbo Pascal tulajdonságot érdemes kipróbálni és megjegyezni. Helyes eredményt csak akkor kapunk, ha magát *i*-t is **longint**-nek deklaráljuk:

Az elsö 200 szám négyzetének összege 2686700

Ha *n* értékét nem akarjuk tovább növelni, akkor *i*-t és *negyzet*-et a 32 bites **longint** helyett elegendő 16 bites előjel nélküli **word**-nek deklarálni, ilyen típusú változó maximális értéke 65535 lehet.

1.3. Oszthatóság

(konstansok, ciklus, feltételes utasítás)

Írjuk ki egy táblázatba azokat az ezernél kisebb természetes számokat, amelyek 7-tel oszthatóak, de 3-mal nem!

```
program oszthatosag;

const
  limit = 1000;
  amivel_oszthato = 7;
  amivel_nem_oszthato = 3;
  sorhossz = 78;
  max_egy_sorban = 15;
  egy_sorban : integer = 0;
var
  i, szam : integer;

begin
  for i := 1 to limit div amivel_oszthato do
    begin
      szam := i * amivel_oszthato;
      if szam mod amivel_nem_oszthato <> 0 then
        begin
          write(szam : sorhossz div max_egy_sorban);
          egy_sorban := egy_sorban + 1;
          if egy_sorban = max_egy_sorban then
            begin
              writeln;
              egy_sorban := 0;
            end;
        end;
    end;
    writeln;
end.
```

Ennek a példának fontos tanulsága a konstansok használata. A program végrehajtható részében egyetlen szám szerepel (az 1-es), minden más számértéket konstansként deklaráltunk. Ebben a rövid és egyszerű példában ez túlzásnak tűnhet, de a későbbi, komolyabb feladatainkban látni fogjuk, hogy ez a helyes programozási stílus. Ha például úgy döntünk, hogy megkeressük azon 777 alatti számokat, melyek 13-mal oszthatóak, de 5-tel nem és a táblázat egy sorába csak 10 számot kérünk, a program törzséhez nem kell nyúljunk, csak a deklarációs részben kell átírni a megfelelő konstansokat.

Újdonság még a konstans-deklarációs rész utolsó sora, az *egy_sorban* nevű tipizált konstans deklarációja. Ez csak elnevezése és a deklarálás helye szerint konstans, funkciója szerint valójában inicializált változó. Deklarációja típust és egyúttal kezdőértéket rendel hozzá. Igy a programban már nem kell értékadó utasítással kezdőértéket kapjon, ami azért is jó, mert erről gyakran megfeledkezünk.

A programban érdemes megfigyelni, hogy nem azt a kézenfekvő megoldást választottuk, hogy 1-től *limit*-ig egyesével megvizsgáljuk a számokat, oszthatóak-e 7-tel és 3-mal, hanem 7-esével lépegettünk és csak 3-mal való oszthatóságot vizsgáltunk. Ezzel sok időt takarítottunk meg, ami persze egy ekkora feladatban mérhetetlenül rövid.

Tanulságos még annak a megvalósítása is, hogy egy sorba annyi érték irodjon ki, amennyit elhatároztunk.

1.4. Kata szorozni tanul

(véletlenszám generálás, "végtelen" ciklus)

Kata most tanulja az egy és kétjegyű számok szorzását fejben. Írjunk programot, amely ilyen feladatokat ad neki és értékeli a válaszát!

```
program szorzo;

var
  x, y, tipp, szorzat : integer;

begin
  randomize;
  repeat
    x := random(98) + 2;
    y := random(8) + 2;
    writeln('Megadok ket szamot, szorozd ossze oket: ',x, ' es ', y);
    szorzat := x * y;
    write('A szorzat: ');
    readln(tipp);
    if tipp=0 then halt;
    if szorzat=tipp then writeln('Helyes!')
    else writeln ('Buta vagy, a szorzat helyesen: ', szorzat);
  until FALSE;
end.
```

Programunk első ránézésre egy végtelen ciklusból áll, hiszen a **repeat - until FALSE** utasítások közötti ciklusmag mindaddig ismétlődik, amig az **until** után álló logikai kifejezés igazzá nem válik. Ez persze sohasem következik be. A program mégis értelmesen megállítható, mert a ciklus belsejében van egy feltételes utasítás, aminek hatására a program leáll, ha 0-t kap válaszként a felhasználótól.

A szorzótényezőket véletlenszámként generáljuk. A **random(n)** függvény egy 0 és n-1 közötti egész számot ad vissza, tehát **random(98)** egy 0 és 97 közöttit. Ehhez azért adunk 2-t, hogy a szorzandó ne lehessen 0 vagy 1, azon nincs mit gyakorolni. A program által kiszámított helyes szorzatot a *szorzat* nevű változóban tároljuk, a felhasználó (Kata) válaszát a *tipp* nevű változóba olvassuk be. A kettő összevetésének eredményétől függően helyeslő vagy helytelenítő üzenetet írunk a képernyőre.

A **randomize** eljárás indítja be a véletlenszám generátort. Próbáljuk ki, mi történik, ha elhagyjuk ezt az utasítást! Semmi különös? Próbáljuk meg még egyszer lefuttatni a programot!

1.5. Római számok

(feltételes utasítás, ciklus, a programszöveg formázása)

Készítsünk táblázatot az arab számok római számra való átírásáról. A számokat 1-től 20-ig egyesével, onnan 100-ig tízesével, onnan 2000-ig százasával növeljük.

```
program romai_szamok;

var
  x, y : integer;

begin
  y:=1;
  repeat
    x:=y;
    write(x:4,': ');
    while x>=1000 do  begin write('M');  x:=x-1000; end;
       if x>=900 then begin write('CM'); x:=x-900;  end;
       if x>=500 then begin write('D');  x:=x-500;  end;
       if x>=400 then begin write('CD'); x:=x-400;  end;
    while x>=100 do  begin write('C');  x:=x-100; end;
       if x>=90 then begin write('XC'); x:=x-90;  end;
       if x>=50 then begin write('L');  x:=x-50;  end;
       if x>=40 then begin write('XL'); x:=x-40;  end;
    while x>=10 do  begin write('X');  x:=x-10; end;
       if x =9 then begin write('IX'); x:=x-9;  end;
       if x>=5 then begin write('V');  x:=x-5;  end;
       if x =4 then begin write('IV'); x:=x-4;  end;
    while x>=1 do begin write('I'); x:=x-1; end;
    writeln;
    if y<20  then y:=y+1  else
    if y<100 then y:=y+10 else
                  y:=y+100;
  until y>2000;
end.
```

A program működése nem igényel különösebb magyarázatot. Az átváltandó "arab" számot az *y* nevű változóban tartjuk, és 1-től növeljük a feladatban megadott módon. Az *x* munkaváltozó kezdeti értéke *y*, majd rendre levonjuk belőle azokat az értékeket, amelyeknek a római megfelelőjét már kiírattuk, míg el nem fogy.

Eddigi programjainkat igyekeztünk egységes, jól áttekinthető külsővel megírni, a **begin - end** párokat azonos bekezdéssel, belsejüket mélyebb bekezdéssel indítani. Ez általában jó olvashatóságot biztosít, de ebben a példában rontotta volna az áttekinthetőséget és nagyon megnövelte volna a sorok számát. A változtatással arra bíztatjuk olvasóinkat, hogy ne egy merev szabályhoz ragaszkodjanak, hanem mindig

a legkedvezőbb megoldást keressék. A programok formai jegyeiről egy nagyobb feladat kapcsán a következő fejezetben szólunk részletesen.

1.6. Utazás a négyes számhoz

(ciklus, párosság-vizsgálat, maximum keresés)

Keressük meg 1 és 250 között azt a számot, amelyiktől a leghosszabb úton jutunk el a négyes számhoz! Az "utazás" algoritmusa a következő: ha egy szám páratlan, szorozzuk meg hárommal és adjunk hozzá egyet, ha páros, osszuk el kettővel. Ezt ismételjük, míg eredményül négyet nem kapunk, s közben számoljuk a lépéseket. Például a hatos számtól hat lépéssel jutunk célba: 6-3-10-5-16-8-4.

```
program utazas4;

const
  limit = 250;
var
  i, max_lepes, lepes_szam : integer;
  eredmeny, keresett : integer;

begin
  max_lepes := 0;
  for i := 1 to limit do
    begin
      eredmeny := i;
      lepes_szam := 0;
      while eredmeny <> 4 do
        begin
          inc(lepes_szam);
          if odd(eredmeny) then eredmeny := eredmeny * 3 + 1
                           else eredmeny := eredmeny div 2;
        end;
      if lepes_szam > max_lepes then
        begin
          max_lepes := lepes_szam;
          keresett := i;
        end;
    end;
  writeln(keresett, '-tol ', max_lepes, ' lepesben jutottam el 4-ig.');
end.
```

A programban egy **for** ciklussal végignézzük az összes számot 1-től 250-ig. Az *eredmeny* nevű változó kezdőértéke a kiinduló szám (*i*), majd további értéke a feladatban leírt algoritmus szerint változik, míg 4 nem lesz. Az **odd**(*n*) függvény akkor ad vissza igaz logikai értéket, ha *n* páratlan. A lépéseket folyamatosan számoljuk, és ha célbaérve ez a szám

nagyobb, mint az addigi maximum, akkor bejegyezzük *max_lepes*-be, a kiinduló *i* értéket pedig *keresett*-be. Az inc(*n*) eljárás 1-gyel megnöveli a (sorszámozott típusú) *n* értékét, tehát az inc(*lepes_szam*) utasítás hatása azonos a *lepes_szam* := *lepes_szam* + *1* értékadásával. A két megoldás hatékonysága között olyan kicsi a különbség, hogy mindenki nyugodtan választhat ízlése szerint.

Ha több olyan szám van, amelyiktől a maximális lépésszám vezet a négyes számhoz, akkor ezek közül az elsőt jelzi ki a program.

Programunkban sok utat - feleslegesen - többször is bejárunk. Ezek kiszűrését az olvasóra bízzuk. A 250-es határ felemelése esetén ennek jelentős hatása lehet a futási időre.

1.7. Kockázás gyanús feltételekkel

(véletlenszám generálás, kezdeti érték adás, tömbök)

Kata és Marci kockázni akarnak. Marci azt javasolja, hogy ha hatost dobnak, Kata kapjon egy 12 forintos csokoládét, ha bármi mást dobnak, Marci kapjon egy 3 forintost. Katának gyanús a dolog és tudni szeretné, igazságosak-e a feltételek. Írjunk programot, amely játékukat modellezi, és különböző számú dobás esetén kiírja az eredményt!

```
program kockazas;

var
  hany_dobas, i, dobas : integer;
const
  hany : array [1..6] of integer = (0, 0, 0, 0, 0, 0);
  marci_ossz : integer = 0;
  kata_ossz : integer = 0;
  marci_nyer = 3;
  kata_nyer = 12;

begin
  randomize;
  write('Hany dobas legyen: ');
  readln(hany_dobas);
  for i:=1 to hany_dobas do
    begin
      dobas := random(6) + 1;
      inc(hany[dobas]);
      if dobas = 6 then kata_ossz := kata_ossz + kata_nyer
                   else marci_ossz := marci_ossz + marci_nyer;
    end;
  writeln(hany[1], ' egyes, ', hany[2], ' kettes, ', hany[3], ' harmas, ',
          hany[4], ' negyes, ', hany[5], ' otos, ', hany[6], ' hatos.');
  writeln('Marci nyert ', marci_ossz, ' forintot, Kata nyert ',
                          kata_ossz, ' forintot.');
  writeln;
```

```
  writeln('Usse le az ENTER billentyut...':48);
  readln;
end.
```

A program konstans deklarációs részében több példát találunk a korábban említett tipizált konstansokra. Köztük újdonság, hogy tömbnek is lehet így kezdőértéket adni, az egyes elemeket vesszővel kell elválasztani és az egészet zárójelbe tenni. Ez az első olyan példánk, melyben egydimenziós tömbbel találkozunk, figyeljük meg deklarálását és használatát.

Ha a legigazságosabb feltételeket akarjuk kikisérletezni, csak a deklarációs részben kell megváltoztatni a nyeremények értékét. Persze ebben az egyszerű feladatban ezt fejben hamarabb kiszámolhatjuk, de hamarosan látunk majd egy másik játékot (sakkozás), amellyel számítógép nélkül nehezen boldogulnánk.

Néhány olvasó már bizonyára bosszankodott azon, hogy ha a Turbo Pascal integrált környezetében futtatja a programját, és a képernyő nincs megfelelően megosztva a szövegszerkesztő (edit) és az eredmény (output) ablakok között, akkor az utolsó kiíratás után az üzenetek eltünnek a szeme elől. Példánk végén ennek a megelőzésére mutatunk egy módszert. A **readln** utasítás hatására mindaddig a képernyőn marad a program által kiírt végeredmény, míg nem ütjük le az ENTER billentyűt. Erről persze illik tájékoztatni a program felhasználóját, ezt teszi a **readln** előtti kiíratás.

1.8. Köbre emelés Marci módszerével

(ciklus, feltételes utasítás, logikai változó)

Marci azt állítja, hogy egy *n* természetes szám köbe kiszámítható *n* darab egymásutáni páratlan szám összegeként, ha a páratlan számok sorozatának kezdőtagja *n*(n-1)+1* . Például:

$$5^3 = 21 + 23 + 25 + 27 + 29$$

Írjunk programot, amely megvizsgálja az első 1000 számra, igaz-e Marci állítása!

```
program kob_marci;

uses crt;

const
  max = 1000;
  az_allitas_igaz : boolean = TRUE;
var
  hagyomanyos, marci : longint;
  i, kezdo : longint;
  j : integer;
  kar : char;

begin
  for i:=1 to max do
    begin
      hagyomanyos := i * i * i;
      kezdo := i * (i-1) + 1;
      marci := 0;
      for j:=1 to i do
        begin
          marci := marci + kezdo;
          kezdo := kezdo + 2;
        end;
      if hagyomanyos <> marci then az_allitas_igaz := FALSE;
      if i mod 50 = 0 then
      writeln(i:4, hagyomanyos:11, marci:11, az_allitas_igaz:7);
    end;
  writeln;
  write('Marci allitasa ');
  if az_allitas_igaz then writeln('igaz az elso ', max, ' szamra!')
                     else writeln('nem igaz!');
  writeln;
  writeln('Usson le egy billentyut...':48);
  kar := readkey;
end.
```

A deklarációs részben TRUE értéket adunk az *az_allitas_igaz* nevű logikai változónak, amely mindaddig megőrzi ezt az értéket, míg a ciklusban növelt *i* valamely értékéről ki nem derül, hogy hagyományos és Marci-féle módszerrel számított köbe eltér egymástól. A Marci-féle értéket úgy kapjuk meg, hogy a *kezdo* nevű változóban kiszámítjuk az első páratlan számot, majd egy belső ciklussal összegezzük a soronkövetkező páratlan számokat.

Újdonság a korábbi feladatokhoz képest az eredmény 50-enkénti kiíratása. Ez azért hasznos egy ilyen, viszonylag hosszabb futási idejű programnál, mert nem kell az üres képernyő előtt várni, amig a végeredmény feltünik. Kiírathatnánk persze minden ciklus lépésben is, de az egyrészt lelassitaná a program futását, másrészt az állandóan változó képernyőről semmit nem lehetne leolvasni.

Az előző feladatban megmutattuk, hogyan lehet elérni, hogy az integrált környezetből lefuttatott program eredménye ne tünjön el a képernyőről, amig az ENTER billentyűt le nem ütjük. Sokakban felmerülhetett, hogy miért pont az ENTER-t, miért nem akármit. Ebben a programban erre mutatunk megoldást - látható, hogy ez egy kicsit bonyolultabb. A **readkey** függvény ugyan pont arra való, hogy egy karaktert beolvasson a billentyűzetről, de deklarálni kell egy változót (*kar*), amelyik felveszi ezt az értéket (még ha nem is fogjuk semmire használni), a program elején pedig **uses** kulcsszóval használatba kell venni a **crt** unit-ot, amelyik a **readkey** függvényt tartalmazza.

További példáinkban nem fogunk hasonló képernyő-megállitást alkalmazni, akinek tetszik, beirhatja a programok végére.

Legyünk óvatosak, ha *max* értékét növelve meg akarjuk állapitani, meddig érvényes Marci állitása. A **longint** típusú változókba beirható legnagyobb érték 2147483647 és ezt már 1300 köbe is túllépi. Inkább igazoljuk algebrailag az állitást, nem túl nehéz. Akinek ehhez segitségre van szüksége, az páratlan számokat összegező ciklust alakitsa át egyetlen értékadó utasitássá. Ez a futási idő szempontjából is hasznos, és megmutatja az utat az algebrai levezetéshez is.

Az 1.2. példában magyaráztuk el, miért kell *i*-t is **longint**-nek deklarálni.

1.9. Öröknaptár

(címkék használata, többirányú elágazás)

Készitsünk programot, amely tetszőleges huszadik századi dátumról megmondja, hogy milyen napra esik! A kézikönyvek szerint a számitás nem is olyan egyszerű (kézikönyv nélkül lehet, hogy egyszerűbb lenne): az évszázad nélkül értett évek számához hozzáadjuk az évek számának negyedét (egészre csonkolva), a napok számát és egy jelzőszámot, amely a hónaptól függ. Az igy kapott értékből levonunk egyet, ha szökőév első két hónapjáról van szó. Az eredményt héttel elosztva a maradék megmutatja, hogy a hét hányadik napjára esik az adott dátum (az első nap vasárnap). A jelzőszám értéke januárban és októberben 0, májusban 1, augusztusban 2, februárban, márciusban és novemberben 3, júniusban 4, szeptemberben és decemberben 5, áprilisban és júliusban 6.

```
program oroknaptar;

var
  ev, ho, nap : integer;
  x, jelzoszam : integer;
  nap_rendben, szokoev : boolean;
  szokonap : integer;
const
  napok : array [0..6] of string[10] = ('vasarnap', 'hetfo',
          'kedd', 'szerda', 'csutortok', 'pentek', 'szombat');

begin
  writeln('Irjon be egy datumot, megmondom milyen napra esik!');
  ev := -1;
  while (ev < 1) or (ev > 99) do
    begin
      write('ev [1901-1999] : 19');
      readln(ev);
    end;
  szokoev:=(ev mod 4 = 0) and (ev mod 100 <> 0) or (ev mod 400 = 0);
  szokonap := ord(szokoev);
  repeat
    write('honap: ');
    readln(ho);
  until (ho >= 1) and (ho <= 12);
  repeat
    nap_rendben := TRUE;
    write('nap: ');
    readln(nap);
    if (nap < 1) or (nap > 31) then nap_rendben := FALSE;
    if (((ho=4) or (ho=6) or (ho=9) or (ho=11)) and (nap=31))
                      or ((ho=2) and (nap>28+szokonap)) then
      begin
        writeln('Ebben a honapban nincs ilyen nap.');
        nap_rendben := FALSE;
      end;
  until nap_rendben;
  case ho of
    1,10:    jelzoszam := 0;
    5:       jelzoszam := 1;
    8:       jelzoszam := 2;
    2,3,11:  jelzoszam := 3;
    6:       jelzoszam := 4;
    9,12:    jelzoszam := 5;
    4,7:     jelzoszam := 6;
  end; {case}
  x := ev + ev div 4 + jelzoszam + nap;
  if ((ho=1) or (ho=2)) and szokoev then x := x - 1;
  x := x mod 7;
  writeln('Az adott nap: ', napok[x]);
end.
```

Programunk két részre oszlik. A második rövidebb rész a feladatban kijelölt számítás végrehajtása. Itt érdemes megfigyelni az eddigi példákban nem használt **case** szerkezetet, amely többirányú elágazás megvalósítására szolgál, és az eredmény kiíratásánál a *napok* tömb használatát. Ennek értékeit a deklarációs részben tipizált konstansként adtuk meg. Jó megoldás lett volna az eredmény közlésére egy hétirányú **case** elágazás is a hét napjai szerint.

Az *x* változóban a program során különböző dolgokat tárolunk, előbb a teljes összeget, később annak hetes osztással kapott maradékát. (Nem véletlen, hogy az eddig látott hosszú és kifejező nevek helyett ilyen semleges nevet kapott.) Az eddigi programokban a jobb áttekinthetőség és érthetőség kedvéért igyekeztünk kerülni ezt a megoldást. Tudnunk kell azonban, hogy ez megengedett, sőt nagy programoknál, amikor takarékoskodni kell a memóriával, egyenesen elkerülhetetlen.

Programunk első, hosszabb része az adatok beolvasásával és ellenőrzésével foglalkozik. Mindhárom adatról (év, hó, nap) egyenként megállapítjuk elfogadható-e, és ha nem, újra kérdezzük. Egy valódi programban természetesen célszerű a három ellenőrzést azonos módon végrehajtani, de itt a gyakorlás kedvéért három különböző módszert használtunk.

Említést érdemel még a *szokonap* nevű változó, amelybe szökőév esetén 1-et, egyébként 0-t akarunk írni. Mivel a *szokoev* nevű logikai változót korábban kiszámítottuk, célunkat legegyszerűbben a

```
szokonap := ord(szokoev);
```

utasítással érhetjük el, a FALSE logikai érték sorszáma ugyanis 0, a TRUE-é pedig 1.

1.10. Hatványozás

(alprogramok, paraméter átadás, függvény használat)

A Pascal nyelvvel ismerkedők gyakran hiányolják a több más nyelvben létező hatványozási operátort. Nem vigasztalja őket, hogy a hatványt az

$$a^k = e^{k \cdot \ln(a)}$$

összefüggésből ki lehet számítani. Írjunk olyan függvényt, amelynek bemenő paraméterei az alap és a kitevő, visszaadott értéke pedig maga a hatvány! Használjuk a függvényt egy olyan táblázat elkészítésére, amely a 2 és 6 közötti egész számok 1.0 és 2.0 között 0.2-ként növekvő kitevőjű hatványait tartalmazza!

```
program hatvanyozas;

var
  i, j : integer;
  alap, kitevo, x : real;
  ki : text;

function hatvany(alap, kit : real) : real;
begin
  if alap <= 0.0 then
    begin
      writeln('Az alap nem pozitiv!');
      halt;
    end;
  hatvany := exp(kit * ln(alap));
end; {hatvany}

begin
  assign(ki, 'HATVANY.DAT');
  rewrite(ki);
  writeln(ki, 'Hatvanyozas':36);
  writeln(ki);
  write(ki, 'kitevo:');
  for j := 0 to 5 do
    write(ki, 1.0 + j * 0.2:9:1);
  writeln(ki);
  writeln(ki, 'alap'#25);
  for i := 2 to 6 do
    begin
      alap := i;
      write(ki, alap:6:1, '':2);
      for j := 0 to 5 do
        begin
          kitevo := 1.0 + j * 0.2;
          write(ki, hatvany(alap, kitevo):9:3);
        end;
      writeln(ki);
    end;
  close(ki);
  x := hatvany(-2.0, -2.5);
end.
```

Eddigi programjainkban kerültük az alprogramok (függvények és eljárások) használatát. Ezt csak a kezdő programozók kedvéért tettük, mert egyébként bizonyos, hogy minél jobban részekre van bontva egy program, annál áttekinthetőbb, annál könnyebb karbantartani.

Első alprogramunk egy valós függvény, neve (*hatvany*) kifejezi funkcióját is. Belseje igen egyszerű, a feladatban megadott képletet írtuk át értékadó utasítássá a Turbo Pascal könyvtári függvényeinek felhasználásával. Előtte megvizsgáltuk, pozitív-e az alap. A függvény formális paramétereit szándékosan neveztük el úgy, hogy egyik

megegyezik, másik eltér a hívó programban használt aktuális paraméterektől. A két elnevezésnek semmi köze egymáshoz, nem célszerű, de nem okoz hibát, ha azonos neveket használunk.

A főprogram lényege, hogy egymásba ágyaztuk az alapot és a kitevőt növelgető **for** ciklusokat, és a közös ciklusmagban íratjuk ki a táblázat elemeit. Érdemes megfigyelni, milyen kiíratási paraméterekkel alakítottuk ki a táblázat formáját. #25 a 25-ös ASCII kódú karaktert, a lefele mutató nyilat jelenti. Így lehet kiíratni olyan karaktereket, amelyeknek nincs billentyű megfelelőjük.

Ez az első program, amelyben nem képernyőre, hanem egy diszk file-ba írunk. A file programbeli (logikai) és DOS alatti (fizikai) nevét az **assign** eljárással rendeltük össze, a file-t írásra a **rewrite** eljárással nyitottuk meg, és **close** eljárással zártuk le.

A program legvégén azért hívtuk meg negatív alappal a *hatvany* függvényt, hogy kipróbáljuk, működik-e az ellenőrzés. Ha ezt a hibás hívást a file lezárása elé tettük volna, károsodhatott volna a file tartalma. Ezért nem tanácsos hiba esetén egy alprogramból meghívni a **halt** eljárást. Helyesebb, ha visszatérünk a hívó programba és valamilyen módon (pl. egy visszaadott értékkel) jelezzük a hibát.

1.11. Gömb és gömbcikk felszíne és térfogata

(típus deklaráció, függvény és eljárás, többirányú elágazás)

Írjunk programot, amelyik a felhasználó kívánsága szerint egy gömb vagy egy gömbcikk adatait kérdezi, és felszínét, térfogatát vagy mindkettőt kiírja a képernyőre.

```
program testek;

type
  test_tipus = (gomb, gombcikk);
var
  sugar, magassag : real;
  valasz : char;
  test_fajta : test_tipus;

function felszin(milyen:test_tipus) : real;
begin
  if milyen=gomb then felszin := 4 * pi * sugar * sugar
  else felszin := sugar * pi * (2 * magassag +
                  sqrt(magassag * (2 * sugar - magassag)));
end;

function terfogat(milyen:test_tipus) : real;
begin
  if milyen=gomb then terfogat := 4 * pi * sugar * sugar * sugar / 3
  else terfogat := 2 * pi * sugar * sugar * magassag / 3;
end;
```

```
begin
  repeat
    write('Milyen test [g=gomb, c=gombcikk]: ');
    readln(valasz);
    case valasz of
      'g','G': begin
                 test_fajta := gomb;
                 write('Sugara: ');
                 readln(sugar);
               end;
      'c','C': begin
                 test_fajta := gombcikk;
                 write('Sugara: ');
                 readln(sugar);
                 repeat
                   write('Magassaga: ');
                   readln(magassag);
                   if magassag > 2*sugar then writeln('Tul magas!');
                 until magassag <= 2*sugar;
               end
      else writeln('Rossz valasz!');
    end; {case}
  until valasz in ['g','G','c','C'];

  repeat
    write('Mit szamoljak ki [f=felszin, v=terfogat, m=mindketto]: ');
    readln(valasz);
    case valasz of
      'f','F': writeln('A test felszine: ', felszin(test_fajta):8:3);
      'v','V': writeln('A test terfogata: ', terfogat(test_fajta):8:3);
      'm','M': begin
                 writeln('A test felszine: ', felszin(test_fajta):8:3);
                 writeln('A test terfogata: ', terfogat(test_fajta):8:3);
               end
      else writeln('Rossz valasz!');
    end; {case}
  until valasz in ['f','F','v','V','m','M'];
end.
```

Ebben a programban deklarálunk először egy felsorolt típust (*test_tipus*). Használ akkor láthatnánk igazán, ha kettőnél több féle testet kezelne a program, de ez a program méretét is nagyon megnövelné.

Programunkban két függvényt használunk, ezek a felszínt illetve a térfogatot számítják ki és adják vissza. Bemenő paraméterük a test típusa. A testek méreteit (sugár, stb.) azért nem kell paraméterlistán átadni, mert a főprogramban globális változókként deklaráltuk őket, így az alprogramokból is "látszanak". A függvények törzsében a megfelelő matematikai képletek átírását találjuk Pascal értékadó utasítássá.

A főprogram lényegében a felhasználóval való párbeszédet bonyolítja le. Láthatjuk hogyan kell a kis és nagybetűs válaszokat közösen kezelni, és hogy a case többirányú elágazás kiegészíthető egy **else** ággal, ahova akkor kerül a vezérlés, ha a **case** kulcsszó után álló kifejezés (*valasz*) aktuális értéke egyik case címkével sem egyenlő.

1.12. Tuvudsz ívígy beveszévélnivi?

(string-kezelés, halmaz műveletek)

Tuvud ava szávámívítóvógéváp ívígy beveszévélnivi? Csak ha megtanítjuk! Írjunk programot, amelyik a billentűzetről beolvasott mondatot így átalakítva írja vissza a képernyőre!

```
program vavevivovu;

const
  maganhangzok : set of char = ['A','E','I','O','U'];
var
  eredeti, uj : string;
  i, j : integer;
  KAR : char;

begin
  writeln('Kerek egy mondatot ekezetek nelkul!');
  readln(eredeti);
  j := 0;
  for i := 1 to length(eredeti) do
    begin
      inc(j);
      KAR := upcase(eredeti[i]);
      uj[j] := KAR;
      if KAR in maganhangzok then
        begin
          uj[j+1] := 'V';
          uj[j+2] := KAR;
          j := j+2;
        end;
    end;
  uj[0] := chr(j);
  writeln(uj);
end.
```

A szöveg átalakításának algoritmusa igen egyszerű: ha magánhangzót találunk tegyünk utána egy **v** betűt, majd ismételjük meg a magánhangzót. Két dolog okoz csak problémát: a kis és nagybetük valamint az ékezetes magyar magánhangzók. Az előbbi problémát cselesen megkerüljük: a kimenő szöveget csupa nagybetűvel fogjuk írni,

így nem kell foglalkozni a kisbetűkkel. Átalakításukat az **upcase** könyvtári függvény végzi, átmeneti tárolásukra a csupa nagybetűvel írt *KAR* változót használjuk. Ne feledjük azonban, hogy ez a megkülönböztetés csak a program olvasóját segíti, a Pascal nyelv a kulcsszavakban és azonosítókban nem tesz különbséget a kis és nagybetűk között.

Az ékezetes betűk kezeléséről szerényen lemondunk, persze a felhasználót figyelmeztetjük, hogy mondatába ne írjon ilyen betüket. Egyébként azon ékezetes betűk kezelése, amelyek kis és nagybetűs változatban is léteznek az IBM bővítésű ASCII kódtáblában, nem lenne nehéz (csak hosszadalmas), de néhány betű csak kisbetűs változatban létezik.

A program deklarációs részében egy halmazt generáltunk a magánhangzók tárolására. Ez azért előnyös megoldás, mert a későbbi vizsgálatok az **in** operátorral gyorsan és egyszerűen elvégezhetők.

Az eredeti szöveg hosszát a **length** függvénnyel állapítjuk meg, betűit egy ciklusban másoljuk át nagybetűvé alakítva a kimenő szövegbe, kiegészítve a *VA, VE,* stb. szótagokkal.

Még nem vagyunk készen! A Turbo Pascal string-jei ugyanis olyan karakter tömbök, melyeknek 0-ik eleme a string hosszát tartalmazza. Persze nem egész számként (akkor a tömb 0-ik eleme más típusú lenne, mint a többi!), hanem egy karakterként, amely az IBM bővítésű ASCII kódtáblában azt a sorszámot viseli, ahány karakterből áll a string. Ezt állítjuk elő a **chr** függvénnyel, és jegyezzük be az *uj* string 0-ik elemébe a program végén.

Évérthevetővő?

1.13. Palindrom szavak keresése

(string-kezelés, halmaz műveletek, eljárások)

Palindromnak nevezik azt a szót, amelyik oda és visszafele olvasva ugyanaz (pl. *kajak*). Írjunk programot, amely egy megadott karakteres (olvasható) file-ban megkeresi és a képernyőre listázza az összes ilyen szót!

```
program palindrom;

{Ez a program tetszoleges szovegben (pl. ebben a programban) megkeresi
azokat a szavakat, amelyek oda es visszafele olvasva azonosak. Pl. ebben
a mondatban "Did you refer to this kayak, Bob? = Erre a kajakra
celoztal Bob?" 6 palindrom szo talalhato.}

var
  be : text;
  file_nev, sor, s_oda, s_vissza : string;
  i, j : integer;
const
  szo_szam : integer = 0;
```

```
  pali_szam : integer = 0;

procedure vizsgalat;
var
  i : integer;
begin
  for i := 1 to j do
    begin
      s_vissza[j + 1 - i] := s_oda[i];
    end;
  s_vissza[0] := s_oda[0];
  if s_oda = s_vissza then
    begin
      writeln('palindrom: ', s_oda);
      inc(pali_szam);
    end;
end;

procedure szo_veg;
begin
  s_oda[0] := chr(j);
  if j > 1 then
    begin
      inc(szo_szam);
      vizsgalat;
    end;
  j := 0;
end;


begin
  write('Kérem a file-nevet: ');
  readln(file_nev);
  assign(be, file_nev);
  {$I-}
  reset(be);
  {$I+}
  if ioresult <> 0 then
    begin
      writeln('A file nem létezik!');
      halt;
    end;
  j := 0;
  while not eof(be) do
    begin
      readln(be, sor);
      for i:=1 to length(sor) do
      begin
        if sor[i] <> ' ' then
          begin
            if (sor[i] in ['a'..'z']) or (sor[i] in ['A'..'Z']) or
               (sor[i] in ['ü'..'ú']) then
```

```
                begin
                  inc(j);
                  s_oda[j] := upcase(sor[i]);
                  if sor[i]='é' then s_oda[j] := 'É ';
                  if sor[i]='ö' then s_oda[j] := 'Ö';
                  if sor[i]='ü' then s_oda[j] := 'Ü';
                        {Hogy Ürü se meneküljön!}
                end;
          end
        else  szo_veg;
      end;
    szo_veg;
    end; {while}
  writeln;
  writeln(szo_szam, ' szóból ', pali_szam, ' palindrom.');
  close(be);
end.
```

Programunk egy hosszabb, furcsa kommentárral indul. Ezt azért írtuk, hogy ha a programot lefuttatjuk bemenő file-ként magát a forrás szöveget használva, találjunk néhány palindrom szót. Valódi szövegben elég ritkák az ilyen szavak, előfordul, hogy többezer szó között egy sincs.

Főprogramunk a bemenő file nevének lekérdezésével indul. A file megnyitásának idejére a **{$I-}** direktívával átmenetileg kikapcsoljuk az input/output ellenőrzést, így biztosítjuk, hogy a file hiánya esetén ne tőlünk független rendszerüzenettel álljon le a program, hanem mi kezelhessük le a hibát.

Ezután egy **while** ciklusban soronként olvassuk be a szöveget, míg a file végére nem érünk. A sorokat egy belső **for** ciklussal dolgozzuk fel, a sor hosszát a **length** függvénnyel határozzuk meg. A szavak végét a SPACE karakterről vagy a sor végéről ismerjük fel, a szóveggel kapcsolatos teendőket egy eljárásba gyűjtöttük, hogy ne kelljen ugyanazokat az utasításokat kétszer leírnunk. De még mielőtt ezt az eljárást meghívnánk két dolgot kell elintéznünk. Egyrészt ki kell szűrnünk az írásjeleket, hogy ne zavarják a vizsgálatot. Ezt a három összekapcsolt **in** operátoros halmaz tartalmazás vizsgálattal érjük el, az írásjelek ugyanis mindhárom feltüntetett halmazon kívül esnek (lásd az ASCII kódtáblát). Másrészt minden betűt nagybetűvé kell alakítsunk, hogy pl. az "Apa" szót felismerje a program. Az **upcase** függvény persze csak az angol abc betűit alakítja át nagybetűkké, így a magyar ékezetes betűkkel megint külön megdolgozunk - de ezután az "Ürü" sem csúszhat át a vizsgálaton.

A *szo_veg* eljárás az előző feladatban részletezett módon bejegyzi a string hosszát a 0-ik karakterbe, és ha a szó egy betűnél hosszabb, számlálja és meghívja a *vizsgalat* eljárást. Az egybetűs szavakat (pl. az *a* névelőt) tehát nem tekintjük palindrom szónak.

A *vizsgalat* eljárás megfordítja az eredeti szót, és ha ugyanazt kapja, akkor képernyőre írja a talált palindromot.

Programunk végén kiírjuk a vizsgált és a talált szavak számát, majd lezárjuk az olvasott file-t. A magyar kettősbetüket nem

1.14. Sakkozás még gyanúsabb feltételekkel

(kétdimenziós tömbök)

Kata és Marci több sakk-készlet bábuit összegyűjtve a következőt játszák: kisorsolnak egy mezőt a táblán, s ha az üres, Marci rátesz egy világos bábut. Ha már van rajta egy világos bábu, akkor Kata leüti és helyére tesz egy sötétet. Ha sötét bábu áll rajta, semmi sem történik. Az nyer, akinek a végén több bábuja van a táblán. Marci szerint 64-szer kell sorsolni, mert ennyi mező van a táblán, de Kata szerint ez igazságtalan. Írjunk programot, amely eldönti a vitát!

```
program sakk;

const
  max_meret = 8;
type
  meret_tipus = 1..max_meret;
var
  sor, oszlop : meret_tipus;
  tabla : array [meret_tipus, meret_tipus] of integer;
  vilagos, sotet : integer;
  lepes_szam, i : integer;

begin
  randomize;
  for lepes_szam := 60 to 120 do
    begin
      for sor := 1 to max_meret do
        for oszlop := 1 to max_meret do
          tabla[sor, oszlop] := 0;
      vilagos := 0;
      sotet := 0;
      for i := 1 to lepes_szam do
        begin
          sor := random(max_meret) + 1;
          oszlop := random(max_meret) + 1;
          inc(tabla[sor, oszlop]);
        end;
      for sor := 1 to max_meret do
        for oszlop := 1 to max_meret do
          begin
            if tabla[sor, oszlop] > 0 then
              if tabla[sor, oszlop] = 1 then
                inc(vilagos)
              else inc(sotet);
          end;
      writeln(lepes_szam:4, ' lepes utan ', vilagos:3, ' vilagos es ',
              sotet:3, ' sotet gyozelem.');
    end;
end.
```

A tábla méretét (hány mező van egy oldalon) a *max_meret* nevű konstansba írjuk. Ennek hasznáról akkor győződhetünk meg, ha később (próba-számításokhoz, vagy egy másik játékhoz) meg akarjuk változtatni a tábla méretét. Így csak egy helyen kell módosítani a programot, egyébként pedig 9 helyen kéne.

A külső fő ciklusban a sorsolások számát (*lepes_szam*) változtatjuk 60 és 120 között. A kiírt eredményekből kell majd megállapítsuk, melyik lépésszám a legigazságosabb. Biztosat persze csak a program sokszori futtatása után mondhatunk. Lehetne szélesebb tartományt is vizsgálni, de a játék szabályaiból érezni lehet, hogy 60-nál kevesebb sorsolás világosnak, 120-nál több pedig sötétnek kedvez.

A kezdeti nullázások után véletlenszám generálással sorsoljuk a mezőt (sor és oszlop), a játék maga pedig egyszerűen a *tabla* nevű kétdimenziós tömb megfelelő elemének eggyel való növelését jelenti, hiszen 0 jelzi az üres mezőt, 1 a világos által elfoglalt mezőt, a sötét által elfoglalt mezők értéke pedig 2 vagy annál több.

Ebben a példában sok ciklust használunk, valamennyit **for** ciklussal valósítottuk meg. Gyakorlás kedvéért bármelyiket átírhatnánk **repeat** vagy **while** ciklusra, de ezekben az esetekben, amikor a ciklusváltozó egyesével növekszik az előre ismert kezdeti értéktől a végértékig, a **for** ciklus a legegyszerűbb és legáttekinthetőbb megoldás.

A programban csak ott írtuk ki a **begin - end** utasításpárokat, ahol feltétlenül szükséges volt. Reméljük, hogy a sorok lépcsőzetes bekezdéséből így is világos, hogy melyik ciklus hol kezdődik és hol végződik.

A program eredményét nem árulom el, maradjon titok. Érdemes elgondolkozni egy olyan programon, amelyik az igazságos lépésszámot közvetlenül kiszámolja, és nem a kiírt sok eredmény elemzéséből kell azt kikövetkeztetni.

1.15. Krimi a könyvkiadónál

("közgazdasági alkalmazás" rekordok és mutatók nélkül)

A következő 3 példa témamegjelölésénél azért tettük idézőjelbe a közgazdasági alkalmazást, mert bár a feladatok témája (tágabb értelemben) valóban közgazdaságtan, igazi alkalmazásnak egyszerűségük, kis méreteik miatt még aligha nevezhetők.

Egy könyvkiadó a KONYV.DAT nevű file-ban tárolja az általa kiadott művek adatait. A file karakteres (olvasható), minden egyes sora egy-egy könyvre vonatkozik: max. 24 karakter a mű címe, egy egész szám mutatja a fajtaját (pl. krimi=2), majd két egész szám a megjelenési és az eladott példányszámot.

Írjunk programot, amelyik végigolvassa a file-t, kiírja a képernyőre az elfogyott könyvek címeit és kiszámolja a krimik átlagos megjelenési példányszámát! Tételezzük fel, hogy a file létezik és ellenőrzött módon (pl. egy lekérdező programmal) készült, tehát adatait nem kell ellenőrizni. A korábbi példákban (palindrom szavak,

öröknaptár) már láthattuk, hogyan kell egy file létezését illetve a bemenő adatok helyességét ellenőrizni.

```
program konyvek;

const
  KRIMI = 2;
  krimi_db : integer = 0;
  ossz_krimi : longint = 0;
var
  be : text;
  cim : string[24];
  kiadott_db, eladott_db : longint;
  kod, krimi_atlag : integer;

begin
  assign(be, 'KONYV.DAT');
  reset(be);
  writeln('Az elfogyott konyvek listaja':30);
  while not eof(be) do
    begin
      readln(be, cim, kod, kiadott_db, eladott_db);
      if kiadott_db = eladott_db then writeln(cim);
      if kod = KRIMI then
        begin
          inc(krimi_db);
          ossz_krimi := ossz_krimi + kiadott_db;
        end;
    end;
  close(be);
  krimi_atlag := round(ossz_krimi / krimi_db);
  writeln;
  writeln('A krimik atlagos kiadasi peldanyszama: ', krimi_atlag);
end.
```

A **round** függvény valós értéket kerekítéssel egésszé konvertál. Azért használtuk az egyszerűbb és gyorsabb **div** egész osztás helyett, mert az az eredményt nem kerekíti, hanem csonkítja.

A program azért lehetett ilyen egyszerű, mert a kért adatok mind meghatározhatók a file egyszeri végigolvasásával. Ha a feladat olyan, hogy újra szükség lehet egy korábban olvasott sor adataira, akkor a file-t a program belső "adatbázisába" kell beolvasni, hogy minden adat mindig rendelkezésre álljon. (Az idézőjel arra utal, hogy adatbázisnak a nagy, összetett programok adatszerkezetét szokták nevezni.)

Ebben az esetben programunk deklarációs részéből kimaradnának a *cim, kod, kiadott_db, eladott_db* változók, s helyükre lépne a

konyv_tipus típusú rekordok tömbje, *konyv* :

```
type
  konyv_tipus = record
                  cim : string[24];
                  kod : integer;
                  kiadott_db : longint;
                  eladott_db : longint;
                end;
const
  max_konyv_szam = 500;
var
  konyv : array [1..max_konyv_szam] of konyv_tipus;
```

Ilyen adatszerkezetben pl. a 13. könyv kiadási példányszámára így hivatkozhatunk:

```
konyv[13].kiadott_db
```

Ha a könyvek maximális számát nem tudjuk, vagy nem akarjuk előre meghatározni, mert félünk, hogy - esetleg feleslegesen - túl sok helyet foglalunk le az adatszegmensben, akkor tömb helyett egy láncolt listában kell tárolni a könyveket. Erre látunk példát a következő feladatban.

1.16. Összeláncolt könyvek

("közgazdasági alkalmazás" rekordokkal és mutatókkal)

Használjuk az előző feladat adatait (KONYV.DAT) olyan számításokra, melyekhez célszerű az említett láncolt listás belső tárolás. Írjunk programot, amelyik a képernyőre írja annak a könyvnek a címét, amelyiknek kiadási példányszáma legkevésbé tér el az átlagostól!

```
program lancolt_konyvek;

type
  konyvre_mutato_tipus = ^konyv_tipus;
  konyv_tipus = record
                  cim : string[24];
                  kod : integer;
                  kiadott_db : longint;
                  eladott_db : longint;
                  kovetkezo : konyvre_mutato_tipus;
                end;
const
  ossz_kiadott : longint = 0;
  konyv_db : longint = 0;
  min_elteres : longint = 1000000;
```

```
var
  elso, utolso, uj : konyvre_mutato_tipus;
  be : text;
  keresett_cim : string[24];
  kiadott_atlag, elteres : longint;

begin
  assign(be, 'KONYV.DAT');
  {$I-}
  reset(be);
  {$I+}
  if ioresult <> 0 then
    begin
      writeln('Nincs bemeno file!');
      halt(1);
    end;

  elso := NIL;
  while not eof(be) do
    begin
      new(uj);
      inc(konyv_db);
      readln(be, uj^.cim, uj^.kod, uj^.kiadott_db, uj^.eladott_db);
      ossz_kiadott := ossz_kiadott + uj^.kiadott_db;
      if elso = NIL then elso := uj
                    else utolso^.kovetkezo := uj;
      utolso := uj;
    end;
  utolso^.kovetkezo := NIL;
  close(be);

  kiadott_atlag := round(ossz_kiadott / konyv_db);

  uj := elso;
  repeat
    elteres := abs(kiadott_atlag - uj^.kiadott_db);
    if elteres < min_elteres then
      begin
        keresett_cim := uj^.cim;
        min_elteres := elteres;
      end;
    uj := uj^.kovetkezo;
  until uj = NIL;

  writeln('Az atlagos kiadasi peldanyszam: ', kiadott_atlag,
          ' . Az ettol legkevesbe eltero');
  writeln('peldanyszamu konyv cime: ', keresett_cim);
end.
```

A program típus-deklarációs részében láthatjuk, hogyan kell mutató típust illetve azt tartalmazó rekordot deklarálni. A továbbiakban három ilyen rekordra mutató változót fogunk használni: *elso* a láncolt lista

első, *utolso* az utolsó elemére mutat, *uj* pedig a listára újonnan felkerülő elemre.

A program első részében a már ismert módon megnyitjuk és ellenőrizzük a bemenő file-t, majd következik a láncolt lista felépítése. A **new** eljárással foglalunk helyet az éppen következő könyvnek, melyre az *uj* nevü mutató mutat. Beolvassuk az adatait (cím, kód, stb.), majd hozzáillesztjük az eddigi lista végéhez. Ez úgy történik, hogy az *utolso* által mutatott eddigi utolsó könyv *kovetkezo* nevű mezőjébe bejegyezzük *uj* -at, majd az *utolso* -ban levő értéket felülírjuk *uj* -jal, hiszen mostantól ez az utolsó elem.

Mindezt addig ismételjük, míg van adat a bemenő file-ban. Különleges tennivalónk csak a lista megkezdésekor és lezárásakor van. A lista kezdését onnan ismerjük fel, hogy az *elso* -ben NIL-t azaz sehova se mutató mutatót találunk. Ilyenkor nem kell az előző elem *kovetkezo* nevű mezőjét kitölteni (hiszen nincs előző elem), csak *elso* -be beírni *uj* értékét. A legutolsó elem *kovetkezo* mezőjébe pedig NIL-t írunk, hiszen azt nem követi több elem, innen fogjuk felismerni a lista végét.

A lista felépítése közben összegeztük a példányszámokat és számoltuk a könyveket, így most kiszámíthatjuk az átlagot. Ezután újra végigmegyünk a listán, és megkeressük az átlaghoz legközelebb álló kiadási példányszámot.

Ha valaki sokallja, hogy minden mezőazonosító elé ki kell írni ugyanazt a rekord nevet, megtakaríthatja ezt egy **with** utasítással. Például

```
with uj^ do
```

utasítás után *uj^.cim* helyett elegendő *cim* -et írni, stb.

1.17. Bizony hitvány bizonyítvány

("közgazdasági alkalmazás" egyben és eljárásokra bontva)

Egy iskolai osztály bizonyítványát a BIZONY.DAT nevű karakteres file tartalmazza. A file első sorában két egész szám áll, az első a tantárgyak, a másik az osztály tanulóinak száma. Ezután soronként következnek a tantárgyak nevei, majd a további sorokban a tanulók nevei maximum 20 karakteren, minden név után az osztályzatok olyan sorrendben, ahogy a tantárgyakat felsoroltuk. Például:

```
3 2
fizika
kémia
matematika
Kiss Károly          3 4 2
Nagy Lajos           1 2 1
```

Készítsünk programot, amely az ATLAG.DAT nevű file-ba írja az osztály tanulóinak nevét és tanulmányi átlagát, sornyomtatóra nyomtatja a 4.0-nál jobb tanulmányi átlagot elérő tanulók neveit, és kiírja a képernyőre, melyik tantárgyból érte el az osztály a legjobb átlagot!

```
program bizonyitvany;

uses printer;

const
  max_tanulo_szam = 100;
  max_tantargy_szam = 25;
type
  tantargy_tipus = record
                     nev : string[20];
                     atlag : real;
                   end;
var
  be, ki : text;
  tanulo_szam, tantargy_szam : byte;
  jegy : array [1..max_tanulo_szam, 1..max_tantargy_szam] of 1..5;
  tantargy : array [1..max_tantargy_szam] of tantargy_tipus;
  nev : string[20];
  jegy_osszeg : word;
  atlag, legjobb_atlag : real;
  i, j : integer;

begin
BEGIN {nyitas}

{megnyitja a file-okat (lst-t nem kell, azt a printer unit elintezi),
ellenorzi, letezik-e a bemeno file}

  assign(be, 'BIZONY.DAT');
  {$I-}
  reset(be);
  {$I+}
  if ioresult <> 0 then
    begin
      writeln('Nincs bemeno file!');
      halt(1);
    end;
  assign(ki, 'ATLAG.DAT');
  rewrite(ki);
END;

BEGIN {beolvasas}

{beolvassa es ellenorzi a tantargyak es tanulok szamat,
majd a tantargyak neveit}
```

```
  readln(be, tantargy_szam, tanulo_szam);
  if (tanulo_szam > max_tanulo_szam) or
     (tantargy_szam > max_tantargy_szam) then
       begin
         writeln('Hibas bemeno adatok');
         halt(2);
       end;
  for j := 1 to tantargy_szam do
    readln(be, tantargy[j].nev);
END;

BEGIN {tanulo_atlag}

{kiirja a lista cimet, majd beolvassa a tanulok neveit es jegyeit,
kiszamitja es kiiratja a tanulok atlagait}

  writeln(lst, '4.0 - nal jobb atlagu tanulok':35);
  for i := 1 to tanulo_szam do
    begin
      read(be, nev);
      jegy_osszeg := 0;
      for j := 1 to tantargy_szam do
        begin
          read(be, jegy[i,j]);
          jegy_osszeg := jegy_osszeg + jegy[i,j];
        end;
      readln(be);
      atlag := jegy_osszeg / tantargy_szam;
      writeln(ki, nev, atlag:24-length(nev):2);
      if atlag > 4.0 then writeln(lst, nev);
    end;
  close(be);
END;

BEGIN {tantargy_atlag}

{a jegy tomb alapjan kiszamitja a tantargyi atlagokat
es megkeresi koztuk a legjobbat}

  legjobb_atlag := 0.0;
  for j := 1 to tantargy_szam do
    begin
      jegy_osszeg := 0;
      for i := 1 to tanulo_szam do
        jegy_osszeg := jegy_osszeg + jegy[i,j];
      atlag := jegy_osszeg / tanulo_szam;
      tantargy[j].atlag := atlag;
      if atlag > legjobb_atlag then legjobb_atlag := atlag;
    end;
END;
```

```
BEGIN {legjobb_atlagok}

{ismerjuk a legjobb tantargyi atlagot, de mivel tobb ilyen is
lehet, most ujraolvassuk a tombot es kiirjuk mindazokat a
tantargyakat, melyek atlaga a legjobbal azonos}

  writeln(legjobb_atlag:4:2, ' atlagu tantargy(ak)');
  for j := 1 to tantargy_szam do
    if tantargy[j].atlag = legjobb_atlag then
      writeln (tantargy[j].nev);
END;
  close(ki);
end.
```

Ebben a feladatban az előző példa mindkét változatában alkalmazott adatkezelési módot megtaláljuk. A tanulók neveit csak sorosan olvassuk a bemenő file-ból, de nem tároljuk el, erre az adott számításokhoz nincs szükség. Osztályzataikat egy kétdimenziós tömbben tároljuk, mert erre később még szükség lesz. Végül a tantárgyakat és tantárgyi átlagokat rekordokba kapcsoljuk össze, és a rekordokból képzett tömbben tároljuk. A deklarációs részben ennek az adatszerkezetnek a leírása található.

Már itt felhívjuk a figyelmet arra, hogy ebben a példában kétféle átlag (és jegyösszeg) létezik: tantárgyankénti és tanulónkénti. Ezeket egymás után ugyanazokban a változókban tároljuk, és fejben tartjuk, melyikre vonatkoznak (veszélyes, de gyakran alkalmazott módszer).

A feladat érdekessége, hogy eredményeit három különböző helyre írjuk. A képernyőre és egy diszk file-ba (ATLAG.DAT) a korábban megismert módon, míg a sornyomtatóra közvetlenül a **printer** unit segítségével. Ennek használatát a program elején a **uses** utasítással jeleztük, ezekután az **lst** perifériára küldött kiíratások közvetlenül a sornyomtatóra mennek.

A program működését itt nem magyarázzuk, mert a forrás szövegbe minden rész elejére kommentárt írtunk.

Ezt az eddigieknél hosszabb programot csak azért írtuk meg alprogramok nélkül, hogy megmutathassuk mennyivel áttekinthetőbb, kezelhetőbb lesz, ha eljárásokra bontjuk. Ezt készítettük elő a programba írt (egyébként felesleges) nagybetűs **BEGIN - END** utasításpárokkal. Ezek az így kijelölt részek szinte kínálják, hogy eljárásként deklaráljuk őket. Ez az adott példában csak annyit jelent, hogy minden nagybetűs **BEGIN** utasítás elé egy **procedure** kulcsszót írunk és az eljárás nevét, ami célszerűen a **BEGIN** után kapcsos zárójelben álló szó lehet. Ezután csak egy főprogramot kell írni a program végére:

```
begin
  nyitas;
  beolvasas;
  tanulo_atlag;
  tantargy_atlag;
  legjobb_atlagok;
  close(ki);
end.
```

Az egyszerűség kedvéért a deklarációs részhez nem is nyúltunk (hiszen így minden eljárás "látja"), de a gyakorlatban az a célszerű, ha azokat a deklarációkat, amelyeket csak egy eljárás használ, az adott eljárás belsejében helyezzük el.

1.18. Bizalmatlan hajótöröttek

(Egy legendás feladat Pascal megoldása)

Hat hajótörött (öt férfi és egy majom) egy kis lakatlan szigetre vetődik. Gyorsan összegyűjtik a szigeten található összes kókuszdiót, majd kimerülten elalszanak. Éjszaka felébred az egyik férfi, és attól tartva, hogy társai megrövidítik, ötfelé osztja a zsákmányt. Egy dió marad - azt a majomnak adja, majd saját részét elrejtve visszatér aludni. Hamarosan felébred egy társa és pontosan ugyanezt cselekszi (ötödölés, egy a majomnak, eldugás, alvás), majd sorban mindegyik. Reggel nevetve rekonstruálják az éjszaka eseményeit, majd a maradékot igazságosan elosztják. Ezuttal a majomnak nem jut, de ő ezt nem is bánja, eléggé jóllakott az éjszakai öt dióval. Hány dió volt eredetileg a szigeten?

A feladat azért nevezetes, mert a század elején, amikor az amerikai Saturday Evening Post-ban először jelent meg B.A.Williams novellájában, nagy sajtóbotrányt okozott. A rejtvényt ugyanis az egyik szereplő adta fel a másiknak, de a megoldás nem derült ki a történetből. Hamarosan olvasók tízezrei rohanták meg a szerkesztőséget a megoldást követelve...

A feladat papírral és ceruzával néhány órás számolással megoldható. De aki a Turbo Pascal-t ismeri egy ügyes programmal sokkal hamarabb eljuthat a célhoz.

```
program kokusz;

const
  megvan : boolean = FALSE;
var
  maradek, reggel, felebredo : integer;

function lehetseges(var maradek:integer):boolean;
begin
  lehetseges := false;
  if maradek mod 4 = 0 then
    begin
      lehetseges := true;
      maradek := maradek div 4 * 5 + 1;
    end;
end;
```

```
begin
  reggel := 0;
  repeat
    reggel := reggel + 20;
    if reggel mod 100 = 0 then writeln(reggel);
    maradek := reggel;
    felebredo := 5;
    while lehetseges(maradek) do
      begin
        felebredo := felebredo - 1;
        if felebredo=0 then megvan := TRUE;
      end;
  until megvan;
  writeln(maradek, ' kokuszdio volt a szigeten (reggelre ',
          reggel, ' maradt).');
end.
```

A feladatot próbálgatással oldjuk meg. Egyre növeljük a diók számát, míg meg nem találjuk a lehetséges megoldást.

A módszer kulcsa a *lehetseges* nevű logikai függvény. Bemenő paraméterként megkapja azt a számot, ahány kókuszdiót talál az éppen felébredő hajótörött. Ha ez nem lehet egy előző osztozás maradéka, mert nem osztható néggyel, akkor hamis értéket kapunk vissza; ha lehet, akkor igazat, és a paraméter értéke is módosul az előző felébredő által talált dió-számra (nem felejtve a majmot sem).

A főprogram megírása ezekután már nem is nehéz. A reggel talált diók számát a *reggel* nevű változóban fogjuk tárolni, és ezt fogjuk növelgetni, amíg megoldásig nem jutunk. A növelés mértéke 20, hiszen a reggeli dió-szám biztosan osztható 20-szal, mert osztható 5-tel (maradék nélkül el tudták osztani egymás között), és osztható 4-gyel (az utolsó felébredő hagyta úgy, hogy az öt kupacból egyet eldugott).

Ezzel a reggeli maradék értékkel hívjuk a *lehetseges* függvényt mindaddig, míg igaz értéket ad vissza. Ha ez ötször sikerül, megtaláltuk a megoldást, ha nem akkor növelni kell *reggel* értékét és tovább próbálkozni.

A feladatnak természetesen több megoldása van, a fenti program ezek közül a legkisebbet találja meg. Az olvasóra bizzuk a program olyan átalakítását, hogy a további megoldások is kiderüljenek. (Vigyázzunk az egészek típusának megválasztására, ha diók száma 32000 fölé is nőhet!)

Itt említjük meg, de több korábbi feladatra is érvényes, hogy egy logikai változónak egy feltételtől függően igaz vagy hamis értéket kétféleképpen lehet adni. A hosszabb, de kezdők számára kézenfekvőbb módszer:

```
if feltetel then logikai_valtozo := TRUE
            else logikai_valtozo := FALSE;
```

A tömörebb, de első látásra szokatlan módszer:

```
logikai_valtozo := feltetel;
```

Az utóbbi megoldással a *lehetseges* függvény illetve a főprogram egy-egy részlete így egyszerűsíthető:

```
lehetseges := maradek mod 4 = 0;

megvan := felebredo = 0;
```

Ha már a rövidítéseknél tartunk, elrettentésül megmutatjuk, hogyan lehet ezt a feladatot 6 sorban megoldani. Reméljük, mindannyian elborzadnak:

```
program rovid_kokusz; var m,reg,fel:integer;                { IGY    }
function leh(var m:integer):boolean; begin                  { NEM    }
leh:=m mod 4=0; if m mod 4=0 then m:=m div 4*5+1 end;       { SZABAD }
begin reg:=0; repeat reg:=reg+20; m:=reg; fel:=5;           { PROG-  }
while leh(m) do dec(fel) until fel=0;                       { RAMOZ- }
writeln(m,' dio volt, ', reg,' maradt.') end.               { NI!!!  }
```

2. UNIT-OK HASZNÁLATA

Ebben a fejezetben készítünk egy játékos programot a ferde hajítás tulajdonságainak tanulmányozására. A játékban a dobás szögének és kezdősebességének változtatásával kell egy adott távolságra lévő célt eltalálni. A dobás kezdőpontjába egy tankot, célpontjába egy házikót rajzolunk, de megnyugtathatunk mindenkit, hogy a program ennek ellenére nem nevel agresszivitásra, viszont kiváló eszköz a ferde hajítás fizikájával való megismerkedésre.

A programot igyekeztünk vonzó grafikával és hanghatásokkal, sőt zenével is gazdagítani, így mérete jelentősen meghaladja az eddigi kis gyakorló programokét. Még így is elférne egyetlen program szegmensben, de gondolva a későbbi esetleges bővítésre illetve az áttekinthetőségre és az egyes részek újrafelhasználhatóságára a programot több unit-ra bontottuk.

A *kozos* nevű unit-ba helyeztük el a globális deklarációkat és egy több más unit által is használt függvényt. Ezt a unit-ot a főprogram és valamennyi többi unit is használni fogja. A *grafika* nevű unit-ba a képi megjelenítéssel kapcsolatos dolgok kerültek, a dobás (lövés) paramétereit kijelző és a kezdőképet (tank és ház) felrajzoló eljárások. A *tuzeles* nevű unit tartalmazza a röppályát kiszámító, megjelenítő és hanggal kisérő valamint a találatot képpel és zenével jelző eljárásokat. Utóbbi két unit-ot a főprogram használja, míg a *zene* nevű zenélő unit-ot a *tuzeles* és a *grafika* nevű unitok.

Mielőtt az egyes unit-okat részletesebben ismertetnénk, ideje, hogy szóljunk a programírás stílusáról, formájáról is, hiszen az eddigi rövid példáknál ez nem lett volna igazán indokolt. Programunk külső megjelenése, formája, tagolása, olvashatósága nagyon fontos nem csak idegeneknek, hanem saját magunknak is, különösen ha hosszabb idő után vesszük elő egy régebbi programunkat. A hosszú, kifejező azonosítók használata okozta gépelési munka bőségesen megtérül a program könnyebb olvasásában és a kevesebb magyarázat-igényben. Az egyes sorok elrendezése is sokat javíthat az áttekinthetőségen. Ennek módjára nincs szabály, csak szokások. Mindenesetre érdemes az összetartozó utasításokat egymás alatt azonos bekezdéssel kezdeni, a beágyazott belső részeket a bekezdés változtatásával elkülöníteni. A Pascal nyelvben a kis és nagybetűk egyenértékűek: ez jól használható a kulcsszavak és azonosítók megkülönböztetésére vagy a lényeges részek kiemelésére. A közölt mintaprogramokban figyelemmel kellett legyek a könyv terjedelmi korlátaira is; valódi programjaim még szellősebbek, gyakran teszek üres sort az egyes részek elkülönítésére, sohasem írok több utasítást egy sorba. A programok, unit-ok elejére megjegyzésként mindig felírom a fontosabb adatokat és egy rövid leírást.

2.1. A főprogram

A játék főprogramja viszonylag rövid, és néhány számításon kívül csak eljáráshívásokat tartalmaz. A véletlenszám-generátor beindítása, a grafikát inicializáló *elokeszites* eljárás hívása és a címkép felrajzolása után következik a fő ciklus, melyben a játékot addig ismételjük, míg a felhasználó le nem állítja. Ez a leállítás a *robban* eljárás paramétereként visszaadott *valasz* karakterrel történik.

A főciklus magjában véletlenszámként generáljuk a cél távolságát, majd ennek függvényében beállítunk két értéket. Az egyik az a sebesség, amelyik a játék elején kijelzésre kerül, mint induló érték. Minél messzebb van a cél, ezt annál nagyobbra állítjuk, hogy a játékosnak ne kelljen túl sokat változtatni. A másik érték egy elemi távolság, amellyel a cél-házat leírjuk, ez annál kisebb, minél messzebb van a cél, így a távoli házat kisebbnek fogjuk ábrázolni a képernyőn.

A kezdőkép kirajzolása után kiszámítjuk a tényleges (méter) és a képernyő (pixel) távolságok arányát, ezt az értéket a lövedék pályájának felrajzolásához fogjuk használni. Kijelezzük a cél távolságát, majd egy belső ciklusban addig ismételjük a lövési paraméterek beállítását és a lövést, míg a lövedék el nem találja a célt. Ezt a *robban* eljárásban képpel és hanggal jelezzük.

A főprogram mellé került még két olyan eljárás, amely logikailag leginkább ide illeszkedik, a grafikát inicializáló *elokeszites* nevű eljárás, és a lövési paramétereket (cső szöge és kezdősebesség) beállító *beallit* eljárás. Ezek működését nem részletezzük, mert - reméljük - a hosszú, kifejező azonosítók és a sorvégi kommentárok jól olvashatóvá teszik a programot.

Az *elokeszites* eljárással kapcsolatban megjegyezzük, hogy a programot úgy írtuk meg, hogy különböző grafikus kártyákkal (EGA, VGA, stb.) egyaránt működjön. Ennek érdekében a rajzméreteket soha nem abszolút értékekkel, hanem a maximális méretekből számítva adjuk meg. Ugyanakkor a CGA grafika kezeléséről kis felbontása és kevés színe miatt le kellett mondjunk, aki mindenáron CGA-n akarja használni, annak néhány helyen módosítani kell a programot.

Programunk elején a $L direktívával beszerkesztünk programunkba két object formátumú file-t. Egyik a "triplex" karakter-fontokat leíró TRIP.CHR file, másik az EGA és VGA grafikát meghajtó EGAVGA.BGI file object formátumra átalakított változata. Az átalakítást a BINOBJ segédprogrammal végeztük. A beszerkesztett file-okat az EXTERNAL utasítással külső eljárásoknak deklaráljuk. Így elérjük, hogy a HAJIT.EXE program önmagában működőképes lesz, nem kell hozzá mellékelni az EGAVGA.BGI és a TRIP.CHR file-okat. Még egy teendőnk van ezzel kapcsolatban: az *elokeszites* eljárás elején, még az *initgraph* eljárás hívása előtt meghívni a *RegisterBGIdriver* illetve *RegisterBGIfont* függvényeket.

```
program hajit;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                                     *)
(*      program: HAJIT föprogram          file: HAJIT.PAS              *)
(*      változat: 1.3                     dátum: 1991.01.04.           *)
(*      programozó: Hegedüs A.            hardver: IBM PC AT           *)
(*      op.rendsz: DOS 3.2                nyelv: TurboPascal 5.0       *)
(*                                                                     *)
(*      A ferde hajitást demonstráló játék föprogramja. Tartalmaz-     *)
(*      za a grafikát inicializáló ELOKESZITES és a ferde hajítás      *)
(*      (tüzelés) paramétereit beállító BEALLIT eljárásokat is.        *)
(*                                                                     *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

uses kozos, tuzeles, grafika, crt, graph;

{$L trip.obj }                                  {triplex fontokat  }
procedure TriplexFontProc; EXTERNAL;            {es EGAVGA drivert }
                                                {tartalmazó file-ok}
{$L egavga.obj }                                {object formátumu  }
procedure EGAVGADriverProc; EXTERNAL;           {változatának      }
                                                {beszerkesztése    }
procedure elokeszites;
begin
if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
  begin
    writeln('Nem sikerült beépiteni a BGI file-t!');
    halt(1);
  end;
if RegisterBGIfont(@TriplexFontProc) < 0 then
  begin
    writeln('Nem sikerült beépiteni a font file-t!');
    halt(2);
  end;
driver := detect;                               {grafika       }
initgraph(driver, mode, '');                    {inicializálása}
if GraphResult <> grOK then
  begin
    writeln('Nem sikerült elindítani a grafikát!');
    halt(3);
  end;

settextjustify(centertext, centertext);
maxx := getmaxx;                                {maximális }
maxy := getmaxy;                                {képméretek}
alapvonal := 3 * maxy div 4;
d_min := maxy div 100;                          {elemi lépések a ház }
r := maxy div 40;                               {és a tank leirásához}
cso := 4 * r;
x_tav_ki := maxx div 6;                         {kijelzö mezök  }
```

```
x_szog_ki := maxx div 2;                    {helye és mérete}
x_seb_ki := x_tav_ki * 5;
y_ki := maxy div 6;
w_ki := maxx div 15;
h_ki := maxy div 25;
end; {elokeszites}


procedure beallit;
var
  kar : char;
const
  FEL = #72;  LE = #80;                     {érvényes }
  BAL = #75;  JOBB = #77;  ZERO = #0;       {billentyük}
begin
repeat
  kar := readkey;
  if kar = ZERO then                        {nyilak}
    begin
      kar := readkey;
      case kar of
      FEL: begin
             if szog_fok < max_szog then    {ha még nem érte       }
             begin                          {el a maximumot, akkor }
               inc(szog_fok);               {megnövelem a szöget,  }
               szog_rad := szog_fok*pi/180; {átszámítom radiánba,  }
               tank_cso(x_tank, alapvonal); {kirajzolom a csövet,  }
               szog_ki;                     {kijelzem a szögértéket}
             end;
           end;
      LE:  begin
             if szog_fok > min_szog then
             begin
               dec(szog_fok);
               szog_rad := szog_fok*pi/180;
               tank_cso(x_tank, alapvonal);
               szog_ki;
             end;
           end;
      BAL: begin
             if sebesseg > min_seb then     {ha még nem érte el    }
             begin                          {a minimumot, akkor    }
               dec(sebesseg);               {csökkentem a sebességet}
               sebesseg_ki;                 {és kijelzem az értékét }
             end;
           end;
```

```
      JOBB: begin
              if sebesseg < max_seb then
              begin
                inc(sebesseg);
                sebesseg_ki;
              end;
            end;
       end; {case}
    end;
until kar = SPACE;                          {addig olvasom a karaktereket,}
end; {beallit}                              {míg SPACE-t (tüz) találok    }


begin {main}
randomize;                          {véletlenszám generáláshoz}
elokeszites;
cim_kep;
repeat
  talal := FALSE;
  celtav_meter := random(max_tav - min_tav) + min_tav;
  skala := (max_tav-1) div 100 - celtav_meter div 100;
                            {SKALA egy kis egész szám, amely távoli cél}
                            {esetén 0, közelebbi cél  esetén  egyre nö }
  sebesseg := 10 * (6 - skala); {ettöl függ az induló sebesség        }
  d := d_min + skala;           {és a ház mérete is                   }
  kezdo_abra;
  ratio := celtav_meter/celtav_pixel;
  tav_ki;
  repeat
    beallit;
    loves;
  until talal;                  {addig ismétlem, míg el nem találja}
  robban(valasz);
until valasz <> SPACE;          {ha SPACE-t üt, ismétlem a játékot,}
closegraph;                     {ha mást, akkor befejezem          }
end.
```

2.2. A globális deklarációk unit-ja

A *kozos* nevű unit-ba helyeztük el mindazokat a deklarációkat, melyeket több másik unit használ. Az INTERFACE részbe kerültek a legkisebb és legnagyobb céltávolságot, cső-szöget, kezdősebességet meghatározó konstansok, a színek közül csak azok, amelyeket több unit-ban is használunk, és az összes globális változó. Ebbe a unit-ba kell beírni azokat a függvényeket és eljárásokat is, amelyeket több más unit-ból használunk, de esetünkben csak egy ilyen van, az egészet stringgé alakító *inttostr* függvény. Ennek feje szintén az INTERFACE részbe, a teljes függvény pedig az IMPLEMENTATION részbe kerül.

```
unit kozos;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                                     *)
(*      program: KOZOS unit              file: KOZOS.PAS               *)
(*      változat: 1.3                    dátum: 1990.12.30.            *)
(*      programozó: Hegedüs A.           hardver: IBM PC AT            *)
(*      op.rendsz: DOS 3.2               nyelv: TurboPascal 5.0        *)
(*                                                                     *)
(*      A ferde hajítást demonstráló játék unit-ja. Tartalmazza        *)
(*      a többi unit által közösen használt globális deklarációkat     *)
(*      és az egészet stringgé konvertáló INTTOSTR függvényt.          *)
(*                                                                     *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

INTERFACE

uses graph;

const
  g = 9.81;
  talal : boolean = FALSE;
  max_tav = 500;
  min_tav = 100;
  max_szog = 75;
  min_szog = 10;
  max_seb = 75;
  min_seb = 25;
  hatter_szin = cyan;
  tank_szin = lightblue;
  SPACE = #32;

var
  driver, mode : integer;
  maxx, maxy : integer;
  x_tav_ki, x_szog_ki, x_seb_ki : integer;
  y_ki, w_ki, h_ki : integer;
  x_haz, x_tank, fel_haz, cso : integer;
  alapvonal, celtav_meter, celtav_pixel : integer;
  sebesseg, szog_fok : integer;
  szog_rad, ratio : real;
  valasz : char;
  skala, d, d_min, r : integer;

function inttostr(number: longint): string;
```

```
IMPLEMENTATION

function inttostr(number: longint): string;
var
  number_string: string;
begin
  str(number, number_string);
  inttostr := number_string;
end;

end.
```

2.3. A tüzeléssel kapcsolatos eljárások unit-ja

A *tuzeles* nevű unit-ba két eljárás került, a lövedék pályáját kiszámító, megjelenítő és változó magasságú hanggal kisérő *loves* , és a találatot képpel, hanggal és felirattal jelző *robban* . Az eljárások feje itt is az INTERFACE részbe, a teljes eljárások pedig az IMPLEMENTATION részbe kerülnek.

A **$L** direktívával a szerkesztő (linker) programot arra utasítjuk, hogy a *preludio.obj* nevű object file-t szerkessze hozzá a programunkhoz, ezt külső (external) eljárásként fogjuk kezelni és egy zene (Bach: E preludium) adatait tartalmazza object formában.

A *loves* eljárás lényegében a ferde hajítás ismert képletének

$$y = x \operatorname{tg}\alpha - x^2 \frac{g}{2 v^2 \cos^2 \alpha}$$

átírása Pascal értékadó utasítássá, illetve a méterben kiszámított értékek átváltása képernyő koordinátákra. Minden képpont megjelenítéséhez egy hang tartozik, melynek magassága (frekvenciája) a röppálya magasságával arányos. A hangot a **sound** eljárással lehet megszólaltatni és a **nosound** eljárással kikapcsolni. A találatot jelző *talal* logikai változó akkor kap igaz értéket, ha a lövedék a ház alapjára esik, tehát a becsapódás és a cél távolságának eltérése kisebb, mint a ház félszélessége. A tank csövének töve és a ház alapja egy magasságban van.

A *robban* nevű eljárás egy háttér színű kitöltött téglalappal felülrajzolja a házat (tehát eltünteti), egy félkörívet ("gödröt") rajzol a helyére, majd a képernyő aljára kiírja a használati utasítást. Ezután egy ciklusban addig ismétli a zene lejátszását, míg a játékos le nem üt egy billentyűt. Ezt a karaktert kimenő paraméterként visszaadja a hívó eljárásnak, mert az ebből fogja megállapítani, hogy a felhasználó tovább akar-e játszani.

```
unit tuzeles;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                                   *)
(*      program: TUZELES unit          file: TUZELES.PAS             *)
(*      változat: 1.3                  dátum: 1991.01.02.            *)
(*      programozó: Hegedüs A.         hardver: IBM PC AT            *)
(*      op.rendsz: DOS 3.2             nyelv: TurboPascal 5.0        *)
(*                                                                   *)
(*      A ferde hajítást demonstráló játék unit-ja. Tartalmazza      *)
(*      a ferde hajítás (tüzelés) adatait számító és megjelenitö     *)
(*      LOVES és a találatot jelzö ROBBAN eljárásokat.               *)
(*                                                                   *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

INTERFACE

uses kozos, graph, crt, zene;

procedure loves;
procedure robban(var valasz:char);

IMPLEMENTATION

{$L preludio.obj}                          {a zenét tartalmazó object}
procedure preludio; EXTERNAL;              {file beszerkesztése      }

procedure loves;
var
  tav_meter, x_meter, y_meter : real;
  i, y_pixel, tav_pixel : integer;
begin
  tav_meter := sqr(sebesseg)*sin(2*szog_rad)/g;
  tav_pixel := round(tav_meter/ratio);
  for i:=0 to tav_pixel do          {x irányban egyenként lépegetve     }
    begin                           {kiszámitom a hozzátartozó y értéket}
      x_meter := i * ratio;         {és megjelenitem az adott pontot    }
      y_meter := x_meter*sin(szog_rad)/cos(szog_rad)-
                 sqr(x_meter)*g/(2*sqr(sebesseg)*sqr(cos(szog_rad)));
      y_pixel := round(y_meter/ratio);
      putpixel(i+x_tank, alapvonal-y_pixel, 1);
      if i<(tav_pixel div 2) then sound(i*30)        {félutig emelkedö,}
                  else sound((tav_pixel-i)*30);      {utána ereszkedö  }
    end;                                             {hangot generálok }
  nosound;
if abs(tav_pixel - celtav_pixel) < fel_haz then
  talal := TRUE;
end; {loves}
```

```
procedure robban(var valasz:char);
begin
  setfillstyle(solidfill, hatter_szin);
  bar(x_haz-fel_haz, alapvonal-3*fel_haz, x_haz+fel_haz, alapvonal);
                                                  {eltünteti a házat}
  setcolor(brown);
  arc(x_haz, alapvonal, 180, 360, fel_haz);  {gödröt rajzol a helyére}
  bar(0, maxy-4*h_ki, maxx, maxy);
  setcolor(magenta);
  settextstyle(defaultfont, horizdir, 1);
  outtextxy(x_szog_ki, maxy-3*h_ki,
        'Nyomja meg a SPACE billentyüt, ha folytatni akarja,');
  outtextxy(x_szog_ki, maxy-2*h_ki,                {kiírja a    }
        'bármi mást, ha nem.');                    {tudnivalókat}
  settextstyle(triplexfont, horizdir, 5);
  valasz := #0;
  repeat
    dallam(@preludio, TRUE, valasz);               {zenét játszik     }
  until valasz <> #0;                              {billentyü leütésig}
end; {robban}

end.
```

2.4. A grafikus eljárások unit-ja

A *grafika* nevű unit-ba 6 olyan eljárást gyüjtöttünk össze, amelyik a képernyőre rajzol illetve megjelenit valamit. Pontosabban az implementációs részben ennél több eljárás van, de ezek egy része csak a unit-on belül kerül felhasználásra, ezért az INTERFACE részben nem jelenik meg.

Az IMPLEMENTATION rész elején itt is beszerkesztünk egy object file-lá alakitott zeneszámot (Bach: C-moll preludium), majd következik a három kijelző eljárás. Itt a maguktól értetődő grafikus eljárás hivásokhoz csak annyit kell hozzáfűznünk, hogy a szöveg kiiratását azért ismételjük meg x irányban 1 pixellel eltolva, hogy vastagabb, jobban olvasható karaktereket kapjunk.

A *tank_cso* eljárás a tank csövét rajzolja ki, bemenő paraméterei a cső tövének x és y koordinátái, a cső másik végének helyét a cső hosszából és szögéből számitjuk ki.

A *kezdo_abra* nevű eljárásban először két belső eljárást deklarálunk: a ház felrajzolására szolgáló *haz* -at és a tankot rajzoló *tank* -ot. Mindkettőnek bemenő paraméterei a helyét meghatározó x - y koordináták. A ház méreteit a főprogramban kiszámitott *d* elemi méret határozza meg, igy annál nagyobb lesz, minél közelebb van a cél. A tank kerekeit azért rajzoljuk *ellipse* és nem *circle* eljárással, mert a kör rajzolását a Turbo Pascal *graph* unit-ja automatikusan korrigálja az eltérő x és y irányú felbontásnak megfelelően, igy a csatlakozó vonalak pontos helyét nehéz lenne meghatározni.

A *cim_kep* eljárás kék háttérre különböző szinű és stílusú (font és betűméret) feliratokat ír ki, majd a C-moll preludiumot játsza, míg le nem ütnek egy billentyűt.

```
unit grafika;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                                       *)
(*      program: GRAFIKA unit          file: GRAFIKA.PAS                 *)
(*      változat: 1.3                  dátum: 1991.01.03.                *)
(*      programozó: Hegedüs A.         hardver: IBM PC AT                *)
(*      op.rendsz: DOS 3.2             nyelv: TurboPascal 5.0            *)
(*                                                                       *)
(*      A ferde hajítást demonstráló játék unit-ja. Tartalmazza          *)
(*      az adatokat kijelzö, a kezdöképet (tank és ház) és a             *)
(*      címképet megjelenitö eljárásokat.                                *)
(*                                                                       *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

INTERFACE

uses kozos, graph, zene;

procedure tav_ki;
procedure szog_ki;
procedure sebesseg_ki;
procedure tank_cso(x, y:integer);
procedure kezdo_abra;
procedure cim_kep;

IMPLEMENTATION

{$L cminor.obj}                 {a zenét tartalmazó object}
procedure cminor; EXTERNAL;     {file beszerkesztése      }

procedure tav_ki;               {távolság érték kijelzése}
begin
  setfillstyle(solidfill, lightgray);
  setcolor(yellow);
  bar(x_tav_ki-w_ki, y_ki-h_ki, x_tav_ki+w_ki, y_ki+2*h_ki);
  outtextxy(x_tav_ki, y_ki+3, inttostr(celtav_meter));
  outtextxy(x_tav_ki+1, y_ki+3, inttostr(celtav_meter));
end;

procedure szog_ki;              {szög érték kijelzése}
begin
  setfillstyle(solidfill, lightcyan);
  setcolor(black);
  bar(x_szog_ki-w_ki, y_ki-h_ki, x_szog_ki+w_ki, y_ki+2*h_ki);
  outtextxy(x_szog_ki, y_ki+3, inttostr(szog_fok));
  outtextxy(x_szog_ki+1, y_ki+3, inttostr(szog_fok));
end;
```

```
procedure sebesseg_ki;              {sebesség érték kijelzése}
begin
  setfillstyle(solidfill, lightblue);
  setcolor(lightcyan);
  bar(x_seb_ki-w_ki, y_ki-h_ki, x_seb_ki+w_ki, y_ki+2*h_ki);
  outtextxy(x_seb_ki, y_ki+3, inttostr(sebesseg));
  outtextxy(x_seb_ki+1, y_ki+3, inttostr(sebesseg));
end;


procedure tank_cso(x, y: integer);       {tank csövének megjelenítése}
begin
  setcolor(tank_szin);
  setfillstyle(solidfill, hatter_szin);
  bar(x+1, y-1, x+cso+1, y-cso-1);
  line(x-1, y, x+round(cso*cos(szog_rad))-1, y-round(cso*sin(szog_rad)));
  line(x+1, y, x+round(cso*cos(szog_rad))+1, y-round(cso*sin(szog_rad)));
end; {tank_cso}


procedure kezdo_abra;
var
  dx, dy : integer;

procedure haz(x, y: integer);
begin
  fel_haz := 3*d;
  setfillstyle(solidfill, lightblue);
  bar(x-3*d, y-4*d, x+3*d, y);
  setfillstyle(solidfill, brown);
  setcolor(brown);
  line(x, y-7*d, x-3*d, y-4*d);                    {tetö}
  line(x, y-7*d, x+3*d, y-4*d);
  line(x-3*d, y-4*d, x+3*d, y-4*d);
  floodfill(x, y-5*d, brown);
  bar(x+d, y-7*d, x+2*d, y-5*d);                   {kémény}
  line(x-9*d, y, x+9*d, y);                        {föld}
  setfillstyle(solidfill, white);
  bar(x-2*d+1, y-3*d, x-d+1, y-2*d);               {ablakok}
  bar(x+d-1, y-3*d, x+2*d-1, y-2*d);
end; {haz}

procedure tank(x, y: integer);
begin
  setcolor(tank_szin);
  arc(x-2*r, y-r div 2, 25, 150, r);               {felsö rész}
  rectangle(x-4*r, y-r+2, x, y);

  ellipse(x-5*r, y+r, 0, 360, r-2, r-3);           {kerekek}
  ellipse(x-3*r, y+r, 0, 360, r-2, r-3);
  ellipse(x-r, y+r, 0, 360, r-2, r-3);
```

```
  ellipse(x+r, y+r, 0, 360, r-2, r-3);
  ellipse(x-5*r, y+r, 90, 270, r, r-1);          {lánctalp}
  ellipse(x+r, y+r, 270, 90, r, r-1);
  line(x-5*r, y+1, x+r, y+1);
  line(x-5*r, y+2*r-1, x+r, y+2*r-1);

  tank_cso(x, y);                                {csö}
end; {tank}


begin
  setcolor(blue);
  setfillstyle(solidfill, hatter_szin);          {háttér}
  bar(0, 0, maxx, maxy);
  x_tank := x_tav_ki;
  x_haz := maxx - x_tank;
  celtav_pixel := x_haz - x_tank;
  haz(x_haz, alapvonal);                         {ház}
  szog_fok := 30;
  szog_rad := szog_fok*pi/180;
  tank(x_tank, alapvonal);                       {tank}
  setcolor(magenta);
  settextstyle(defaultfont, horizdir, 1);
  outtextxy(x_tav_ki, y_ki-2*h_ki, 'távolság [m]');   {feliratok}
  outtextxy(x_szog_ki, y_ki-2*h_ki, 'szög [fok]');
  outtextxy(x_seb_ki, y_ki-2*h_ki, 'sebesség [m/s]');
  outtextxy(x_tav_ki, y_ki+3*h_ki, 'ADOTT');
  outtextxy(x_szog_ki, y_ki+3*h_ki, #24#25);
  outtextxy(x_seb_ki, y_ki+3*h_ki, #26#27);
  outtextxy(x_szog_ki, maxy-3*h_ki,
          'A nyilakkal állitsa be a csö szögét és a lövedék
           kezdösebességét, majd');
  outtextxy(x_szog_ki, maxy-2*h_ki,
          'a SPACE billentyüvel tüzeljen. A lövedéknek
           a ház alapjára kell esnie.');
  settextstyle(triplexfont, horizdir, 4);
  szog_ki;                                                {kijelzés}
  sebesseg_ki;
end; {kezdo_abra}


procedure cim_kep;
var
  valasz : char;
begin
  setfillstyle(solidfill, blue);
  bar(0, 0, maxx, maxy);
  setcolor(yellow);
  settextstyle(triplexfont, horizdir, 4);
  outtextxy(maxx div 2, maxy div 2, 'H A J I T S !');
  outtextxy(maxx div 2 + 1, maxy div 2, 'H A J I T S !');
  setcolor(cyan);
```

```
  settextstyle(defaultfont, horizdir, 2);
  outtextxy(maxx div 2, 2 * maxy div 3,
                'Játék a ferde hajítás tulajdonságainak');
  outtextxy(maxx div 2, 2 * maxy div 3 + 25, 'bemutatására');
  setcolor(white);
  settextstyle(defaultfont, horizdir, 1);
  outtextxy(maxx div 5, maxy div 7, 'Program: Hegedüs A.');
  outtextxy(4 * maxx div 5, maxy div 7, 'Zene: J. S. Bach');
  outtextxy(maxx div 5, maxy div 7 + 15, '(1991)');
  outtextxy(4 * maxx div 5, maxy div 7 + 15, '(1722)');
  outtextxy(maxx div 2, 8 * maxy div 9, ' üssön le egy billentyüt...');
  valasz := #0;
  repeat
    dallam(@cminor, TRUE, valasz);              {zenét játszik     }
  until valasz <> #0;                           {billentyü leütésig}
end; {cim_kep}

end.
```

2.5. A zenét szolgáltató unit

A *zene* nevű unit-ban két belső, és a kívülről is használható *dallam* névű eljárás található. Ez a unit N. J. Rubenking *Pianoman* nevű programjának felhasználásával készült, annak belső adatformátumát használja illetve dolgozza fel. A *Pianoman* programmal kottából vagy szabadon zenét rögzíthetünk, majd visszajátszhatjuk és módosíthatjuk. A program shareware terjesztésű, azaz mindenki ingyen hozzájuthat és kipróbálhatja, de ha használni akarja, ki kell fizetni a szerzőnek a (nem túl magas) regisztrálási dijat.

```
unit zene;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                                     *)
(*      program: ZENE unit              file: ZENE.PAS                 *)
(*      változat: 1.3                   dátum: 1990.12.30.             *)
(*      programozó: Hegedüs A.          hardvér: IBM PC AT             *)
(*      op.rendsz: DOS 3.2              nyelv: TurboPascal 5.0         *)
(*                                                                     *)
(*      A ferde hajítást demonstráló játék unit-ja. Tartalmazza        *)
(*      az adott dallamot lejátszó DALLAM eljárást.                    *)
(*      N.J.Rubenking PIANOMAN programja felhasználásával készült.     *)
(*                                                                     *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
```

```
INTERFACE

uses crt;

procedure dallam(p:pointer; stop:boolean; var kar:char);

IMPLEMENTATION

type
  hangjegy = record
               O, NS : Byte;
               D : Word;
             end;
  hangjegy_mut = ^hangjegy;
var
  oktav : array[0..8] of Real;
  frekv : array[1..12] of Real;

procedure frekvencia_beallitas;
var
  N : Byte;
begin
  frekv[1] := 1;
  frekv[2] := 1.0594630944;
  for N := 3 to 12 do
    frekv[N] := frekv[N - 1] * frekv[2];
  oktav[0] := 32.70319566;
  for N := 1 to 8 do
    oktav[N] := oktav[N - 1] * 2;
end;


procedure szolal(okt, staccato : Byte; terjedelem : Integer);
const
  faktor : array[0..10] of Real =
         (0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0);
var
  frekvencia : Real;
  hang, stac : Byte;
begin
  hang := staccato SHR 4;
  stac := staccato and $F;
  if stac > 10 then stac := 10;
  if stac < 0 then stac := 0;
  if okt > 8 then okt := 8;
  if okt < 1 then okt := 1;
  case hang of
    1..12 : begin
              frekvencia := oktav[okt] * frekv[hang];
              Sound(Round(frekvencia));
              Delay(Round(terjedelem * faktor[10 - stac]));
```

```
            if terjedelem > 0 then NoSound;
            Delay(Round(terjedelem * faktor[stac]));
          end;
    13 : begin NoSound; Delay(terjedelem); end;
  end;                          {case}
end;


procedure dallam(p:pointer; stop:boolean; var kar:char);
var
  T : hangjegy_mut;
  N, Num : Word;
begin
  T := hangjegy_mut(P);
  Inc(LongInt(T), SizeOf(hangjegy) * 5);
  Num := LongInt(T^) and $FFFF;
  Inc(LongInt(T), SizeOf(hangjegy) * 4);
  for N := 1 to Num do
    begin
      with T^ do
        szolal(0, NS, D);
      Inc(LongInt(T), SizeOf(hangjegy));
      if stop and KeyPressed then
        begin
          kar := ReadKey;
          Exit;
        end;
    end;
end;


begin {inicializalas}
  frekvencia_beallitas;
end.
```

3. RENDEZÉS ÉS KERESÉS

3.1. Rendezés különféle algoritmusokkal

A rendezésnek nevezzük azt a folyamatot, amikor egy halmaz elemeit valamilyen sorrend szerint (növekvőre vagy csökkenőre) átrendezzük. A rendezés hozzájárul ahhoz, hogy meggyorsítsa az elemek későbbi keresését, mivel egy rendezett halmazon a keresés jóval gyorsabban megy végbe. A rendezés kiemelkedően fontos tevékenység az adatfeldolgozásban is. Sokfajta rendező algoritmus van, mindegyik ugyanazt a feladatot végzi, mégis van közöttük optimálisan működő és van valamilyen előnye a többivel szemben. A rendezőmódszereket két csoportba sorolhatjuk: tömbök rendezése és (soros) file-ok rendezése. Gyakran a két csoportot nevezik még belső ill. külső rendezésnek, mivel a tömbök rendezése a számítógép gyors, *belső* tárában zajlik, míg a file-ok a lassúbb, de tágasabb *külső* tárra helyeződik át.

3.1.1. Tömbök rendezése

A legfontosabb követelmény a tömbök rendezőmódszereitől, hogy a rendelkezésre álló tárat gazdaságosan használják ki. Az ismertetett módszerek az adatok átrendezését a saját helyükön hajtják végre. Ezeknél a tártakarékos módszereknél végezhetünk hatékonyság, azaz időtakarékosság szerinti osztályozást. A hatékonyság jól mérhető a szükséges összehasonlítások C számának és az adatok mozgatásának (cserék) M számának meghatározásával. Természetesen ezek a mennyiségek a rendezésre váró adatok számának, az n-nek a függvényei. A jó rendező algoritmusokban $n \log(n)$ nagyságrendű összehasonlításra van szükség, azonban mégis ismertetünk több olyan ún. *közvetlen módszert*, melyben n^2 összehasonlítás szükséges. A közvetlen módszerek igen alkalmasak arra, hogy megértsük a legfontosabb rendezőelveket, mivel a programjaik könnyen érthetőek, viszonylag rövidebbek, tehát kevesebb helyet foglalnak le a memóriában. A bonyolultabb algoritmusok kevesebb művelettel működnek, de ezek a műveletek részleteikben jóval bonyolultabbak. Megjegyezzük, hogy a közvetlen módszerek elég kis n-re gyorsabbak, míg nagyobb n-ekre azonban már nem érdemes őket használni.

Az ismertetésre kerülő módszerek az eredeti helyükön rendezik át az adatokat növekvő sorrendre. Az alkalmazott módszereket az alábbi csoportokba sorolhatjuk:

1./ rendezés beszúrással
2./ rendezés kiválasztással
3./ rendezés cserével.

A függelékben találjuk meg a rendező algoritmusok bemutatására szolgáló programokat. A *rendez.pas* tartalmazza az adatok típusdefinícióját, az algoritmusokat, az adatokat beolvasó és az eredményeket kiíró eljárásokat.

Fordítás után *rendez.tpu* jön létre, amely

```
uses rendez;
```

utasítással a programba beépíthető. A rendező algoritmusokat bemutató programok használják a *rendez.tpu* file-t. A programokban a rendezendő adatok darabszámát tartalmazza a *db* változó, mely a *const* definícióban kap értéket. Az algoritmusok egész, valós és karakter típusú adatok rendezésére is alkalmasak, az eljárások azonban csak egy adott típus rendezését tudják elvégezni, ezért minden rendező algoritmusra 3 eljárást készült.
Az adatok típusa lehet:

itype	egész típusú,
rtype	valós típusú,
stype	sztring típusú.

A rendezendő adatok száma legyen 50, szükség esetén csak a konstans definíciónál kell javítani. Definiáljunk három fajta tömb típust segédváltozójukkal együtt:

```
const db=50;
type
      itype = array[1..db] of integer;
      ibuff = integer;

      rtype = array[1..db] of real;
      rbuff = real;

      stype = array[1..db] of string;
      sbuff = string;
```

Az adatok beolvasására és a rendezett adatok visszaírására is három eljárás szükséges.

Egész típusú adatokra:

```
procedure olvas_int(var x:itype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (integer tipusu) adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;
```

```
procedure kiir_int(x:itype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;
```

Valós típusú adatokra:

```
procedure olvas_real(var x:rtype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (real tipusu) adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_real(x:rtype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;
```

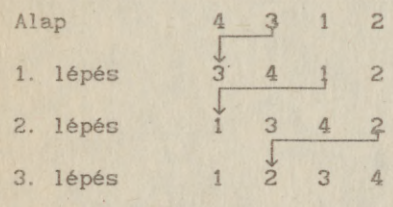
Sztring típusú adatokra:

```
procedure olvas_str(var x:stype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (string tipusu) adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
```

```
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_str(x:stype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;
```

3.1.1.1. Rendezés közvetlen beszúrással

Minden lépésben, i=2-től egyesével *m*-ig kiemeljük a csökkenő sorozat szerinti első elemet és beszúrjuk a megfelelő helyre.
Legyen a rendezendő sorozat:




```
procedure beszur_int(var x:itype; m:integer);
var i,j:integer;
    buf:ibuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       j  := i-1;
       while (j>0) and (buf<x[j]) do
       begin
            x[j+1] := x[j];
                 j := j-1;
        end;
        x[j+1] := buf;
    end;
end;
```

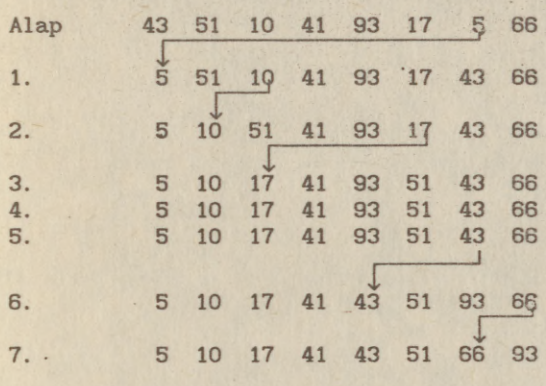
3.1.1.2. Rendezés bináris beszúrással

Sajnálatos módon, a bináris keresőmódszer alkalmazásával elért javítás csak az összehasonlítások számára vonatkozik, a szükséges mozgatásokra azonban nem. Arról nem is beszélve, hogy egy rendezett tömbnek az újrarendezése így több időt venne igénybe, mint amennyit a közvetlen beszúrás használna soros kereséssel!

```
procedure binaris_int(var x:itype; m:integer);
var i,j,l,r,bm:integer;
    buf:ibuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       l  := 1;
       r  := i-1;
       while l<=r do
       begin
            bm:=(l+r) div 2;
            if buf < x[bm] then r:=bm-1 else l:=bm+1
      end;
      for j:=i-1 downto l do x[j+1]:=x[j];
        x[l] := buf;
    end;
end;
```

3.1.1.3. Rendezés közvetlen kiválasztással

A közvetlen kiválasztásnak nevezett módszer bizonyos értelemben a közvetlen beszúrásnak a fordítottja. A közvetlen kiválasztás a *leadó sorozat* összes tételét tekintve találja meg a legkisebb elemet, amit a *fogadó sorozat* egy soronkövetkező helyére teszi.
Legyen a rendezendő sorozat:



```
procedure kozv_int(var x:itype; m:integer);
var i,j,k:integer;
    buf  :ibuff;
begin
    for i:=1 to m-1 do
    begin
       k:=i; buf:=x[i];
       for j:=i+1 to m do
          if x[j]<buf then
          begin
              k:=j; buf:=x[j]
          end;
        x[k]:= x[i]; x[i]:= buf;
    end;
end;
```

3.1.1.4. Rendezés közvetlen cserével

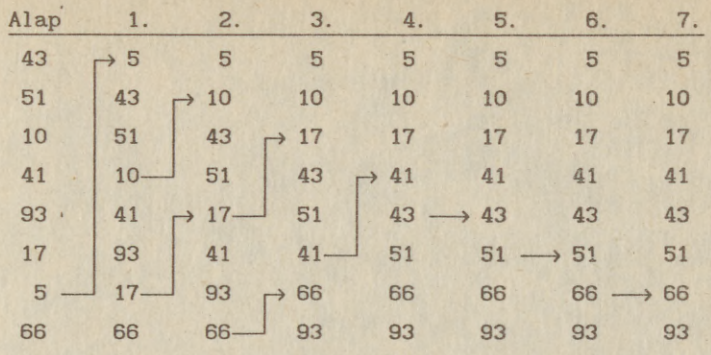
A közvetlen cserélő algoritmusok működési elve az, hogy az alkalmas tételpárok összehasonlítása és felcserélése során jutnak el az összes elem rendezéséhez. A legismertebb módszer a buborék rendezés.

Buborék rendezés

Ha a tömböt egy víztartályhoz hasonlítjuk és a rendezendő tételeknek a buborékok súlyát feleltetjük meg, akkor minden lépés egy buborék felemelkedését eredményezi a súlyának megfelelő szintre.

Ennél az algoritmusnál előfordul, hogy már nincs szükség cserére, mert az adatok már rendezve voltak, mégsem áll le a rendezési művelet. Ezen úgy javíthatunk, hogyha minden menetben feljegyezzük, hogy történt-e csere a menet alatt. Az első olyan menetnél, ahol nem történt csere az algoritmus befejezheti a munkáját. Sőt tovább javítható az algoritmus úgy, hogy megjegyezzük az utolsó csere indexét. Ha ez az index k, akkor ez azt jelenti, hogy a k alatti adatok már jó sorrendben állnak.

Legyen a rendezendő sorozat:

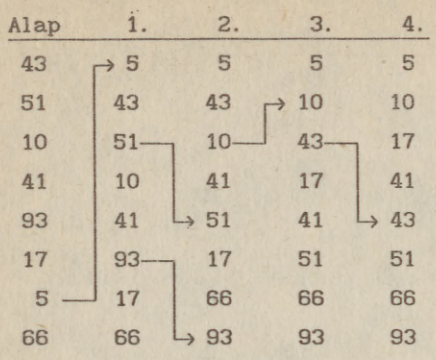



```
procedure buborek_int(var x:itype; m:integer);
var i,j:integer;
    buf:ibuff;
begin
    for i:=2 to m do
    begin
       for j:=m downto i do
         if x[j-1]>x[j] then
         begin
            buf   :=x[j-1];
            x[j-1]:=x[j];
            x[j]  :=buf;
         end;
    end;
end;
```

3.1.1.5. Keverő rendezés

Nem kerülheti figyelmünket el a módszer sajátos szimmetriája. Egyetlen rossz helyen tartózkodó buborék az egyébként rendezett tömb végéről egyetlen menetben a helyére tehető, míg a tömb elejéről nézve a rossz helyen álló adat csak lépésenként egy pozícióval tudjuk a helye felé közelíteni. Ezen természetellens asszimetria miatt javítsunk újra az algoritmuson, úgy hogy változtassuk meg az egymásutáni menetek irányát. Az így adódó algoritmust találóan nevezték *keverő rendezésnek*.

Legyen a rendezendő sorozat:

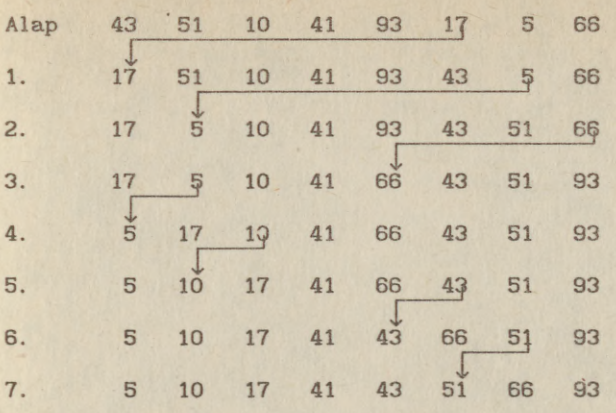



```
procedure kevero_int(var x:itype; m:integer);
var       j,k,l,r :integer;
              buf :ibuff;
begin
    l:=2; r:=m; k:=m;
    repeat
      for j:=r downto l do
        if x[j-1]>x[j] then
        begin
          . buf:=x[j-1];
            x[j-1]:= x[j];
            x[j]:=buf;
            k:=j;
        end;
        l:=k+1;
      for j:=l to r do
        if x[j-1]>x[j] then
        begin
            buf:=x[j-1];
            x[j-1]:= x[j];
            x[j]:=buf;
            k:=j;
        end;
        r:=k-1;
    until l>r;
end; { kevero_int }
```

3.1.1.6. Shell-rendezés

D.L. Shell 1959-ben továbbfejlesztette a közvetlen beszúró rendezést. Ez a többmenetes rendezés, ahol minden menet az összes tétellel foglalkozik, jogos kérdés, hogy valóban megtakarítást és nem Többletmunkát jelent-e.

Legyen a rendezendő sorozat:




```
procedure shell_int(var x:itype; m:integer);
const
    t=5;
var i,j,k,s,ms:integer;
    buf       :ibuff;
    h         :array[1..t] of integer;
begin
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=3; h[5]:=1;
    for ms:=1 to t do
    begin
        k:=h[ms];
        s:=-k;
    for i:=k+1 to m do
    begin
       buf:= x[i];
       j  := i-k;
       if s=0 then
       begin
          s:=-k;
          s:=s+1;
          x[s]:=buf;
       end;
       while (j>0) and (buf<x[j]) and (j<=m) do
       begin
            x[j+k] := x[j];
                 j := j-k;
        end;
        x[j+k] := buf;
    end;
  end;
end;
```

3.1.1.7. Kupacrendezés

Ez a módszer kevés tételre nem ajánlott, azonban nagy *n*-ekre már igen hatékony, minél nagyobb az *n*, annál jobban működik, még a Shell-rendezést is felülmúlja. Legrosszabb esetben *n .log(n)* nagyságrendű lépést végez, és ez a kupacrendezés egyik legnagyobb erénye.

Alap	4	1	5	2	3
1.	4	3	5	2	1
2.	5	3	4	2	1
3.	4	3	1	2	5
4.	3	2	1	4	5
5.	2	1	3	4	5
6.	1	2	3	4	5

```
procedure kupac_int(var x:itype; m:integer);
var l,r:integer;
    buf:ibuff;
    procedure sully;
    label ki;
    var i,j:integer;
    begin
      i:=l; j:=2*i; buf:=x[i];
      while j<=r do
      begin
       if j<r then
         if x[j]<x[j+1] then j:=j+1;
       if buf >=x[j] then goto ki;
       x[i]:=x[j]; i:=j; j:=2*i;
     end;
ki: x[i]:=buf;
    end;
begin
    l:=(m div 2)+1; r:=m;
    while l>1 do
       begin  l:=l-1; sully;
       end;
    while r>1 do
        begin buf:=x[1]; x[1]:=x[r]; x[r]:=buf;
              r:=r-1; sully;
        end;
end;
```

3.1.1.8. Quick (gyors) rendezés

A *qs* eljárás rekurzív módon saját magát hívja. A rekurzió ilyen felhasználása igen hatékony eszköz az algoritmusokban. Az iteratív megoldás lényege az, hogy a felosztások végehajtási igényét feljegyezzük egy várakozó listára. Minden lépésben két újabb felosztási feladat keletkezik. Csak az egyiket lehet azonnal elkezdeni a soronkövetkező iterációban, a másikat erre a várakozó listára helyezzük.

```
procedure quick_int(var x:itype; m:integer);
   procedure qs(l,r: integer; var ix: itype);
   var i,j      :integer;
       buf1,buf2 :ibuff;
   begin
      i:=l; j:=r;
      buf1:=ix[(l+r) div 2];
      repeat
        while ix[i] < buf1 do i:=i+1;
        while buf1 < ix[j] do j:=j-1;
        if i<=j then
        begin
            buf2:=ix[i];
            ix[i]:= ix[j];
            ix[j]:=buf2;
            i:=i+1; j:=j-1;
        end;
      until i>j;
      if l<j then qs(l,j,ix);
      if l<r then qs(i,r,ix);
   end;
begin
      qs(1,m,x);
end; {quick_int}
```

3.1.1.9. Rekord típusú adatok rendezése quick módszerrel

Definiáljunk egy olyan *cim* rekordot, amely tartalmazza egy lakos nevét, és teljes lakcímét (ország jele, irányítószám, város, utca és házszám.

```
type
      cim = record
          nev    : string[25];
          utca   : string[20];
          varos  : string[20];
          orszag: string[2];
          kod    : string[5];
       end;
      cim_t = array[1..db] of cim;
```

A *db* változó a lakosok számát tartalmazza, a *cim_t* pedig az összes adatot tárolja rekordban. Szükség van egy adat beolvasó *olvas_rec* és egy adatkiíró *kiir_rec* eljárásra.

```
procedure olvas_rec(var x:cim_t; var m:integer);
var i:integer;
begin
   writeln('Rendezendo adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=1 to m do
   begin
      write(' Nev    (max. 25 kar.)  = '); readln(x[i].nev);
      write(' Utca   (max. 20 kar.)  = '); readln(x[i].utca);
      write(' Varos  (max. 20 kar.)  = '); readln(x[i].varos);
      write(' Orszag (max. 2 kar. )  = '); readln(x[i].orszag);
      write(' Kod    (max. 5.kar. )  = '); readln(x[i].kod);
      writeln;
   end;
end;

procedure kiir_rec(x:cim_t; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' Nev    : ',x[i].nev);
      writeln(' Utca   : ',x[i].utca);
      writeln(' Varos  : ',x[i].varos);
      writeln(' Orszag : ',x[i].orszag);
      writeln(' Kod    : ',x[i].kod);
      writeln;
   end;
   writeln;
end;
```

Irjunk egy olyan rendező eljárást, amely a lakos neve szerint rendez quick módszerrel:

```
procedure qs_rekord(var x:cim_t; m:integer);
   procedure qs(l,r:integer; var xx:cim_t);
   var  i,j      :integer;
        buf1,buf2:cim;
   begin
      i:=l; j:=r;
      buf1:=xx[(l+r) div 2];
```

```
      repeat
        while xx[i].nev < buf1.nev  do i:=i+1;
        while buf1.nev  < xx[j].nev do j:=j-1;
        if i<=j then
        begin
           buf2 :=xx[i];
           xx[i]:=xx[j];
           xx[j]:=buf2;
           i:=i+1; j:=j-1;
       end;
     until i>j;
     if l<j then qs(l,j,xx);
     if l<r then qs(i,r,xx)
   end;
begin
     qs(1,m,x);
end;
```

3.1.1.10. Adatok rendezése természetes összefésüléssel

Az összefésülés lényege az, hogy az eredeti számsorozatot olyan láncokra osztja, amelyben a számok sorrendje már helyes. Ezekből a láncokból két láncsorozatot állít elő.

Legyen a következő sorozat

6,3,5,9,1,2,7,4,8

első lánc: 6,'',1,2,7
második lánc: 3,5,9,'',4,8

Ezután a megfelelő láncokat össze kell fésülni. Ez úgy történik, hogy az első láncsorozat első láncának első elemét összehasonlítja a második láncsorozat első láncának elemeivel, miközben a két láncból egy lesz, amelyben az elemek sorrendje már helyes lesz. Igy végig kell menni a két láncsorozaton. Ha eredetileg a láncok száma páratlan, akkor a végén egynek nem jut pár, ezt változtatás nélkül írja a keletkező új számsorozat után, ami általában feleannyi láncból áll, mint az eredeti. Ezt addig csinálja, míg az egész számsorozat egy lánc lesz, azaz az elemek sorrendje helyes lesz.

Kétfajta valós típusú tömböt definiáltunk, az adatra 101 elemre, a segédtömbökre pedig elég az eredeti tömb méretének felére (50).

```
const
      l=50;
      l1=101;
type  t1=array[1..l] of real;
      t2=array[1..l1] of real;
```

A természetes összefésülési módszert valós adatokra végzi el a *term_osszef* eljárás. A *lancolás* eljárás *a* és *b* tömbbe osztja szét az adatokat. Két láncsorozatból egy láncsorozatot készít az *ossze* eljárás.

```
procedure term_osszef(var c: t1; n:integer);
var vege: boolean;
    a,b :t2;
    i,j,k : integer;
    min,max : real;
procedure minmax(w:t1; k:integer);
var i:integer;
begin
   min:=w[1]; max:=w[1];
   for i:=2 to k do
   begin
    if w[i]>max then max:=w[i];
    if w[i]<min then min:=w[i];
   end;
end;

procedure lancolas;
begin
    i:=1; j:=1; k:=1;
    repeat
      a[j]:=c[i]; i:=i+1; j:=j+1;
      while (i<=n) and (c[i-1]<=c[i]) do
      begin
         a[j]:=c[i]; i:=i+1; j:=j+1;
      end;
      a[j]:=min; j:=j+1;
      if i<=n then
        begin
          b[k]:=c[i]; k:=k+1; i:=i+1;
        end
      else  if k>1 then k:=k-1;
      while (i<=n) and (c[i-1]<=c[i]) do
      begin
        b[k]:=c[i]; i:=i+1; k:=k+1;
      end;
      b[k]:=min; k:=k+1;
     until i>n;
     b[k]:=max; a[j]:=max;
end;

procedure ossze;
begin
     i:=1; j:=1; k:=1;
     if b[1]=min then k:=2;
     repeat
       if a[j]<b[k] then
       begin
         c[i]:=a[j]; i:=i+1; j:=j+1;
```

```
        end
      else
        begin
          c[i]:=b[k]; i:=i+1; k:=k+1;
        end;
      if a[j]= min then
        while (b[k]<>min) and (b[k]<>max) do
        begin
          c[i]:=b[k]; i:=i+1; k:=k+1;
        end
      else
      begin
      if b[k]=min then
        while (a[j]<>min) and (a[j]<>max) do
        begin
          c[i]:=a[j]; i:=i+1; j:=j+1;
        end;
    end;
    if b[k]=min then k:=k+1;
    if a[j]=min then j:=j+1;
 until i>n;
end;
begin
   vege:=false;
   minmax(c,n);
   min:=min-1; max:=max+1;
  repeat
   lancolas;
   if k=2 then vege:=true
     else  ossze;
  until vege;
end;
```

3.1.1.11. Random diszk file-ok rendezése

A rekord adatok quick módszerrel memóriban történő rendezését írjuk át diszk file-ra. Az adatokat a megadott néven diszk file-ra viszi fel a *kiir_rec* eljárás, a rendezett adatokat írja vissza *olvas_rec_rend* eljárás, a vizsgálandó adatok kiiratására a *olvas_rec_vizsg* eljárás szolgál. A véletlen elérésű diszk file adatait quick módszerrel végzi a *qs_rekord* eljárás, amely használja a *keres* nevű függvényt.

```
function keres(var f:recf; i:integer): str80;
var  p: cim;
begin
    i:=i-1;
    seek(f,i);
    read(f,p);
    keres:=p.nev;
end;
```

```
procedure qs_rekord(var f:recf; m:integer);
   procedure qs(l,r:integer);
   var  i,j,s          :integer;
        buf1,buf2,buf3:cim;
   begin
      i:=l; j:=r;
      s:=(l+r) div 2;
      seek(f,s-1);
      read(f,buf1);
      repeat
        while keres(f,i) < buf1.nev  do i:=i+1;
        while buf1.nev  < keres(f,j) do j:=j-1;
        if i<=j then
        begin
           seek(f,i-1); read(f,buf2);
           seek(f,j-1); read(f,buf3);
           seek(f,j-1); write(f,buf2);
           seek(f,i-1); write(f,buf3);
           i:=i+1; j:=j-1;
       end;
     until i>j;
     if l<j then qs(l,j);
     if l<r then qs(i,r)
   end;
begin
     qs(1,m);
end;
```

3.2. Keresési algoritmusok

3.2.1. Szekvenciális keresés

Egy rendezetlen tömbben a szekvenciális keresés a tömb első elemétől kezdődik és addig tart míg vagy megtalálta a keresendő elemet vagy a tömbnek végére ért. Ez az ejárás használható rendezetlen és rendezett halmazokon is.

```
function seq_keres(x: itype; m:integer; adat:ibuff):integer;
var i:integer;
begin
    i:=1;
    while (adat <> x[i]) and (i<=m) do i:=i+1;
    if i>m then seq_keres:=0
           else seq_keres:=i;
end; { seq_keres }
```

3.2.2. Bináris keresés

Ha az adathalmaz rendezett, akkor a bináris keresés bármely más keresésnél gyorsabb.

```
function bin_keres(x: itype; m:integer; adat:ibuff):integer;
var     a,f,k :integer;
        talal :boolean;
begin
      a:=1; f:=m;
      talal :=false;
      while (a<=f) and (not talal) do
      begin
          k:=(a+f) div 2;
          if adat<x[k] then f:=k-1
                       else
                  if adat>x[k] then a:=k+1
                       else talal :=true; { megtalalta }
      end;
      if talal then bin_keres:=k
               else bin_keres:=0;  { nem talalt }
end; { bin_keres }
```

4. MÉRÉSI ADATOK KIÉRTÉKELÉSE

4.1. Statisztikai adatok számítása

Mérési adatok kiértékeléséhez határozzuk meg az adatok átlagértékét, középértékét, szórását és keressük meg a legtöbbet előforduló adatot. Számítsuk ki a korrelációs együtthatót, próbáljunk az adatokra ráfektetni egy egyenest. Az egyenes adatait határozzuk meg a kétparaméteres regresszióval. Szemléletes módszer, ha rajzot készítünk az adatokról, de ábrázolhatjuk még vonalas diagrammal, illetve oszlopdiagrammal is.

A függelékben található *stat.pas* program menüvezérelve működik. Tartalmaz még olyan menüpontot is, amelynek segítségével az adatokat beolvashatjuk a klaviaturáról, tárolhatjuk diszk file-on, illetve olvashatjuk diszk file-ról is. Az adatok darabszáma maximálisan 100 lehet, amelyet természetesen szükség szerint lehet módosítani. A mérési adatok valós számok.

4.1.1. Átlagérték számítása

Ha az adatok *yd* tömbben vannak és darabszámukat az *n* változó tartalmazza, akkor az átlagérték

$$\text{mean} = \frac{1}{n} \sum_{i=1}^{n} yd_i$$

```
function Mean(yd: rtype; n: integer):real;
var  i: integer;
     s: real;
begin
   s:=0;
   for i:=1 to n do
     s:=s+yd[i];
   Mean:=s/n;
end; { Mean }
```

4.1.2. Középérték számítása

A középértéket a következő módon határozzuk meg, először rendezzük a mérési adatokat növekvő sorrendbe a quick módszerrel és válasszuk ki a sorozatból az *n/2* -dik elemet.

```
function Median(yd: rtype; n:integer):real;
var
   rend: rtype;
     i: integer;
```

```
begin
   for i:=1 to n do
      rend[i]:= yd[i];
   quick_int(rend,n);
   Median:=rend[n div 2];
end;
```

4.1.3. Gyakoriság meghatározása

Keressük meg a mérési sorozatban a leggyakrabban előforduló elemet. Legyen a sorozat

11 13 14 14 15 16 16 16 17 18 19

a 16 háromszor fordult elő, tehát a gyakoriság 16.

11 13 14 14 15 16 16 16 17 18 18 18 19

ebben a sorozatban 2 olyan adat van (16 és a 18), amely háromszor fordult elő. Az eljárás az ugyanazon gyakorisággal előforduló elemek közül utoljára megtaláltat szolgáltatja.

```
function Mode_last(yd: rtype; n:integer): real;
var
  i,j,c,c_r : integer;
  ml, ml_r  : real;
begin
  ml_r :=0; c_r:=0;
  for i:=1 to n do
  begin
    ml:=yd[i]; c:=1;
    for j:=i+1 to n do
      if ml=yd[j] then c:=c+1;
    if c>c_r then
            begin
               ml_r:=ml;
               c_r :=c;
            end;
  end;
  Mode_last:=ml_r;
end; { Mode_last}
```

4.1.4. Szórás számítása

Ha az adatok *yd* tömbben vannak és darabszámukat az *n* változó tartalmazza, akkor a szórás:

$$std = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (yd_i - mean)^2}$$

```
function StdDev(yd: rtype; n:integer):real;
var
   i  : integer;
   s,g: real;
begin
   g:=Mean(yd,n);
   s:=0;
   for i:=1 to n do
     s:=s+((yd[i]-g)*(yd[i]-g));
  s:=s/n;
  StdDev:=sqrt(s);
end; { StdDev }
```

4.2. Kétparaméteres regresszió

Keressünk egy egyenest, amely legjobban illeszkedik a mért adatokra. Az egyenes egyenlete:

$$Y = a + b X$$

Az *a* és *b* értékét meghatározhatjuk a legkisebb négyzetek módszerével, amely megpróbálja minimalizálni a távolságot az adatpontok és az egyenes között. Először meghatározzuk a *b* értékét:

$$b = \frac{\sum_{i=1}^{n} (X_i - M_x)(Y_i - M_y)}{\sum_{i=1}^{n} (X_i - M_x)^2}$$

ahol M_x az X koordináták, az M_y pedig az Y koordináták átlagértéke.
Az *a* értékének meghatározása

$$a = M_y - b \cdot M_x$$

Határozzuk meg még a korrelációs együtthatót (correlation coefficient), amelynek 0 és 1 között kell lenni. Ha a korrelációs együttható 1, akkor az adat teljesen rásimul a regresszió által számított egyenesre. Ha

értéke 0, akkor egyetlen egy pontja sincs az egyenesen, ebben az esetben ez a közelítés nem megfelelő.

A korrelációs együttható számítása

$$cor = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - M_x)(Y_i - M_y)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - M_x)^2}\ \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - M_y)^2}}$$

```
procedure Regress(yd:rtype; n:integer);
var
    a,b,x_atl,y_atl,t1,t2,cor: real;
    yt         : rtype;
    i,min,max  : integer;
    ch         : char;
    gm,gd      : integer;
begin
   y_atl:=0 ; x_atl:=0;
   for i:=1 to n do
   begin
     y_atl:=y_atl+yd[i];
     x_atl:=x_atl+i;
   end;
   x_atl:=x_atl/n;
   y_atl:=y_atl/n;
   t1:=0; t2:=0;
   for i:=1 to n do
   begin
      t1:=t1+(yd[i]-y_atl)*(i-x_atl);
      t2:=t2+(i-x_atl)*(i-x_atl);
   end;
   b:=t1/t2;
   a:=y_atl-(b*x_atl);
   { kiszamitja a korrelacio egyutthatojat }
   for i:=1 to n do
     yt[i]:=i;
     cor:=t1/n;
     cor:=cor/(StdDev(yd,n)*StdDev(yt,n));
     writeln;
     writeln('KETPARAMETERES REGRESSZIO');
     writeln;
     writeln('Regresszio egyenlete   : y = ',a:15:5,'+',b:15:5,'*x');
     writeln('Korrelacios egyutthato : ',cor:15:5);
     writeln;
```

```
      write('Adatok es regresszios egyenes rajzolasa (i/n) ');
      readln(ch);
      writeln;
      if ch<>'n' then begin
      gd:=CGA;
      gm:=CGAC1;
      initgraph(gd,gm,'');
      setcolor(0);
      for i:=1 to n*2 do
         yt[i]:=a+(b*i);
      min:=GetMin(yd,n)*2;
      max:=GetMax(yd,n)*2;
      s_plot(yd,n,min,max,n*2,1);
      s_plot(yt,n*2,min,max,n*2,2);
      readln;
      closegraph;
    end;
end;
```

4.3. Mérési adatok ábrázolása

4.3.1. Ábrázolás regressziós egyenessel

Ábrázoljuk a mért adatokat a regressziós egyenessel.

```
procedure s_plot(yd: rtype; n,ymin,ymax,xmax:integer;color:integer);
var
  xk,yk,dx,i : integer;
  a,norm,s :real;
  ch: char;
  s1   :string;
begin
   if ymin>0 then ymin:=0;
     s:=ymax-ymin;
     norm:=190/s;
     str(ymin:2,s1);
     outtextxy(1,180,s1);
     str(ymax:4,s1);
     outtextxy(1,8,s1);
     str(xmax:4,s1);
     outtextxy(288,192,s1);
     setcolor(1);
     for i:=1 to 19 do putpixel(0,i*10,3);
     line(0,190,320,190);
     setcolor(2);
     for i:=1 to n do
     begin
       a:=yd[i]-ymin;
       a:=a*norm;
       yk:=trunc(a);
```

```
      dx:=300 div xmax;
      xk:=((i-1)*dx)+20;
      putpixel(xk,190-yk,color);
    end;
end; { s_plot }
```

4.3.2. Ábrázolás vonalas diagrammal

A mérési adatokat ábrázolhatjuk vonalas diagrammal is.

```
procedure  Bar_plot(yd: rtype; n:integer);
var
  xk,yk,max,min,dx,i : integer;
  a,norm,s :real;
  ch: char;
  gd,gm:integer;
  s1   :string;
begin
   gd:=CGA;
   gm:=CGAC1;
   initgraph(gd,gm,'');
   max:=GetMax(yd,n);
   min:=GetMin(yd,n);
   if min>0 then min:=0;
     s:=max-min;
     norm:=190/s;
     str(min:2,s1);
     outtextxy(1,180,s1);
     str(max:4,s1);
     outtextxy(1,8,s1);
     str(n:4,s1);
     outtextxy(288,192,s1);
     setcolor(1);
     for i:=1 to 19 do putpixel(0,i*10,1);
     line(0,190,320,190);
     setcolor(2);
     for i:=1 to n do
     begin
       a:=yd[i]-min;
       a:=a*norm;
       yk:=trunc(a);
       dx:=300 div n;
       xk:=((i-1)*dx)+20;
       line(xk,190,xk,190-yk);
    end;
    readln;
    closegraph;
end; { Bar_plot }
```

4.3.3. Ábrázolás oszlopdiagrammal

Mérési adatokat szemléletesen ábrázolja az oszlopdiagram. A rajzon megtaláljuk az adatok minimális és maximális értékét kiírva.

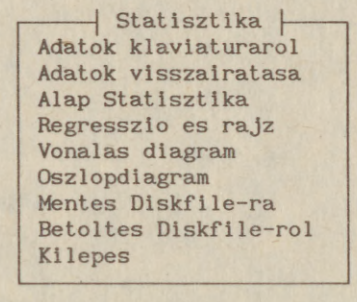
```
procedure oszlop(t:rtype; k:integer; sz: string);
var i:integer;
    tmax,tmin: real;
    gd,gm    : integer;
    color    : array[1..5] of integer;
    xm,ym:integer;
    m    : integer;
    snorm    :real;
    tsz      : integer;
    xx1,yy1,xx2,yy2 :integer;
    yact :integer;
    szam,szam1 : string[6];
    dx: integer;
    x1,x2,y1,y2: integer;
begin
   max_kereses(t,k,tmax,tmin);
   DetectGraph(gd,gm);
   case gd of
   1: begin
       gm:=1;
       xm:=319;
       ym:=199;
       color[1]:=1;
       color[2]:=2;
       color[3]:=3;
      end;
   3: begin
       gm:=0;
       xm:=639;
       ym:=199;
       color[1]:=cyan;
       color[2]:=lightred;
       color[3]:=blue;
      end;
   9: begin
       gm:=0;
       xm:=639;
       ym:=199;
       color[1]:=cyan;
       color[2]:=lightred;
       color[3]:=blue;
      end;
   end;
   initgraph(gd,gm,'');
   setcolor(color[1]);
```

```
if gd=1 then setbkcolor(black) else
        setbkcolor(white);
settextjustify(Centertext,Centertext);
setcolor(1);
outtextxy(xm div 2,10,sz);
x1:=40;
y1:=30;
x2:=xm-20;
y2:=ym-15;
if gd=1 then begin
x1:=10; x2:=xm-10; end;
setcolor(color[3]);
settextjustify(LeftText,BottomText);
   str(tmax:6:0,szam);
   str(tmin:6:0,szam1);
outtextxy(x1,28,'minimum: '+szam1+'     maximum: '+szam);
m:=y2-y1-2;
snorm:=m/tmax;
yact:=y2-y1;
if gd=1 then tsz:=(x2-x1-1) div k else tsz :=(x2-x1-1) div k;
setcolor(2);
if gd=1 then begin xx2:=x1+2; tsz:=tsz-3; dx:=3; end else
             begin xx2:=x1+2; tsz:=tsz-4; dx:=4; end;
    x2:=x1+ k*(tsz+dx)+dx+2;
setcolor(color[1]);
rectangle(x1,y1,x2,y2+2);
setfillStyle(1,color[2]);
for i:=1 to k do
begin
   xx1:=xx2+dx;
   yy1:=y1+yact-round(y[i]*snorm);
   xx2:=xx1+tsz;
   yy2:=y1+round(y[i]*snorm);
   setcolor(1);
   bar(xx1,yy1,xx2,y2);
   rectangle(xx1,yy1,xx2,y2);
   str(i:2,szam);
   setcolor(color[1]);
 if gd=1 then begin
               if k<25 then begin
                SetTextStyle(SmallFont,HorizDir,1);
                SetUserCharSize(13,12,14,17);
                outtextxy(xx1+1,y2+10,szam);
                end
              end
              else
if k<31 then  outtextxy(xx1+5,y2+12,szam);
end;
if ((gd=1) and (k>24)) or ((gd<>1) and (k>30)) then
                  begin
                     outtextxy(x1+3,y2+13,'1');
                     str(k:3,szam);
```

```
                    outtextxy(xx2-22,y2+13,szam);
                end;
  readln;
  closegraph;
end;
```

4.4. Menürendszer felépítése

A menürendszer a *util.tpu* felhasználásával készült.




```
begin

  textcolor(lightgray);
  textbackground(blue);
  clrscr;

  with Col do begin
    FgNormal := black;      BgNormal := lightgray;
    FgBorder := lightgray; BgBorder := red;
    FgSelect := yellow;     BgSelect := green;
  end;

  Point[1] := 'Adatok klaviaturarol';
  Point[2] := 'Adatok visszairatasa';
  Point[3] := 'Alap Statisztika';
  Point[4] := 'Regresszio es rajz';
  Point[5] := 'Vonalas diagram';
  Point[6] := 'Oszlopdiagram';
  Point[7] := 'Mentes Diskfile-ra';
  Point[8] := 'Betoltes Diskfile-rol';
  Point[9] := 'Kilepes';

  Choose := 1;
  num:=0;
  repeat
    MenuWin(30, 10, ThinLine, Col, Point, 9, 'Statisztika', Choose);
    textcolor(lightgray);
    textbackground(black);
```

```
    clrscr;
    gotoxy(20, 5);
    case Choose of
      0 : exit;
      1 :   begin hatter;
                Enter(y,num);
            end;
      2 :   begin hatter;
                Display(y,num);
                end;
      3 :  begin
           hatter;
           if vizsg(num) then begin
              a:=Mean(y,num);
              m:=Median(y,num);
              std:=StdDev(y,num);
              md:=Mode_last(y,num);
              writeln;
              writeln('STATISZTIKAI ADATOK');
              writeln;
              writeln('Mean                : ',a:15:5);
              writeln('Median              : ',m:15:5);
              writeln('Standard deviation: ',std:15:5);
              writeln('Mode                : ',md:15:5);
              writeln;
              tovabb;
            end;
           end;
      4:   begin
             hatter;
             if vizsg(num) then begin
                          Regress(y,num);
                          end;
           end;

      5:  begin
            hatter;
            if vizsg(num) then begin
                          Bar_plot(y,num);
                          end;
          end;
      6:  oszlop(y,num,'S T A T I S Z T I K A');
      7: Save(y,num);
      8: Load(y,num);
      9: exit;
      end;
    clrscr;
  until false;

end.
```

A melléklet tartalmazza a teljes programot.

Futtassuk a programot, válasszuk ki az *Adatok beolvasása* klaviaturáról

```
Az adatok szama: 24
  1. adat :  10
  2. adat :  10
  3. adat :  11
  4. adat :  9
  5. adat :  8
  6. adat :  8
  7. adat :  9
  8. adat :  10
  9. adat :  10
 10. adat :  13
 11. adat :  11
 12. adat :  11
 13. adat :  11
 14. adat :  11
 15. adat :  12
 16. adat :  13
 17. adat :  14
 18. adat :  16
 19. adat :  17
 20. adat :  15
 21. adat :  15
 22. adat :  15
 23. adat :  16
 24. adat :  14
```

Őrizzük meg az adatokat diszkfile-on. Ezt megtehetjük a *Mentes Diskfile-ra* menüben

```
ADAT KIMENTESE DISKFILE-RA

Az adat file neve : st1.dat

Az adat kimentve

        Nyomj Enter-t!
```

Az adatok ellenőrizhetjük az *Adatok visszairatasa* menüben

```
MERESI ADATOK

  1 :         10.00000
  2 :         10.00000
  3 :         11.00000
  4 :          9.00000
  5 :          8.00000
  6 :          8.00000
  7 :          9.00000
```

```
 8 :        10.00000
 9 :        10.00000
10 :        13.00000
11 :        11.00000
12 :        11.00000
13 :        11.00000
14 :        11.00000
15 :        12.00000
16 :        13.00000
17 :        14.00000
18 :        16.00000

      Nyomj Enter-t!


19 :        17.00000
20 :        15.00000
21 :        15.00000
22 :        16.00000
23 :        14.00000
24 :        16.00000
Az adat vege !

      Nyomj Enter-t!
```

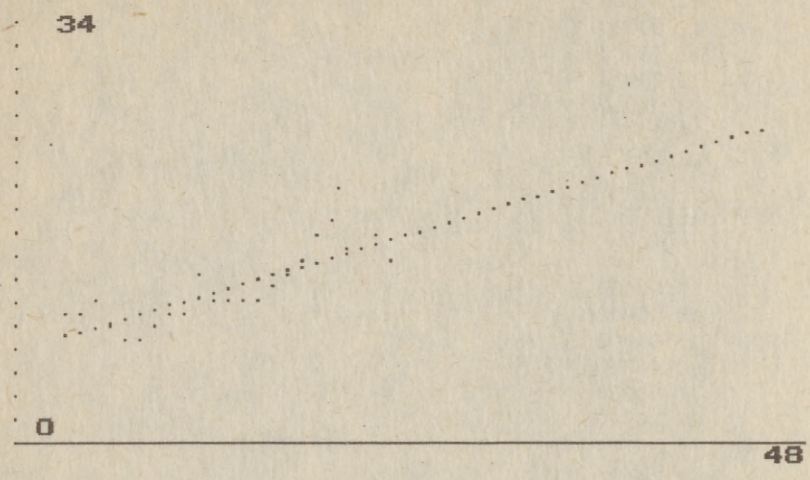
Válasszuk az *Alap statisztika* menüpontot, az alábbi eredményt kapjuk

```
STATISZTIKAI ADATOK

Atlagérték   :      12.08333
Kozepertek   :      11.00000
Szoras       :       2.67577
Gyakorisag   :      11.00000

      Nyomj Enter-t
```

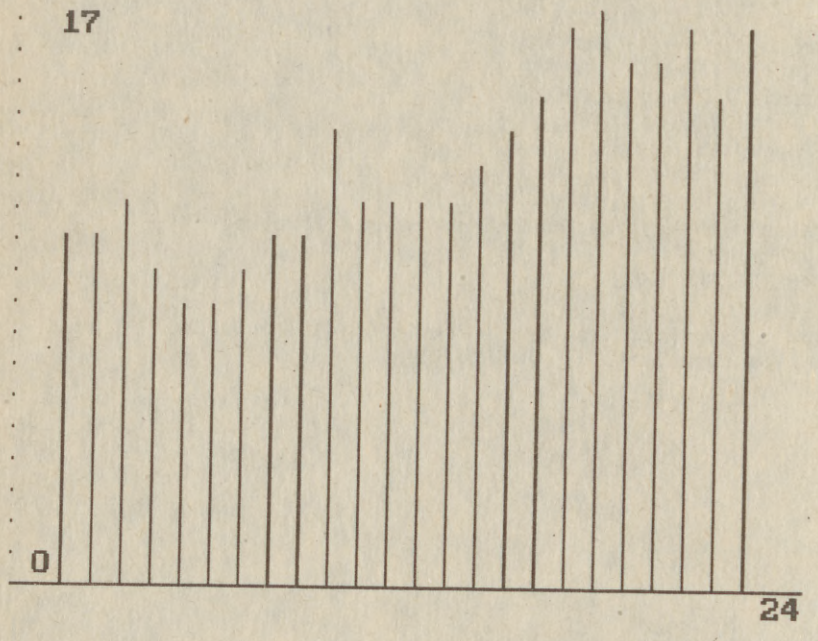
A *Regresszio es rajz* menüben az eredmény a következő

```
KETPARAMETERES REGRESSZIO

Regresszio egyenlete   : y = ( 7.899E+00) + ( 3.348E-01) * x
Korrelacios egyutthato :   8.661E-01
```

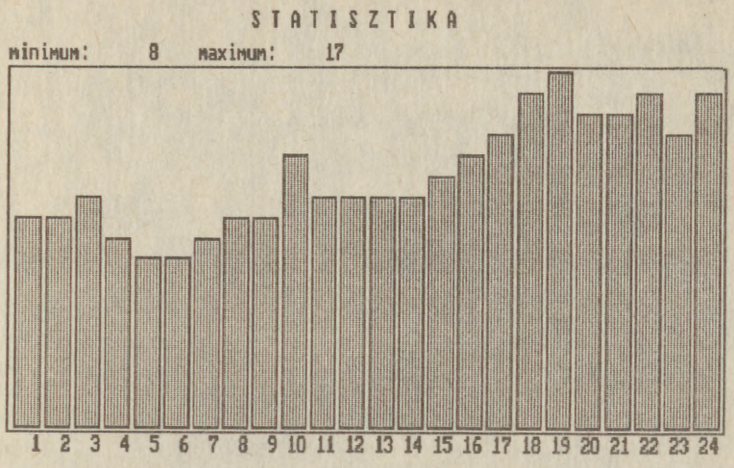
Adatok es regresszios egyenes rajzolasa (i/n) i



A *Vonalas diagram* menü eredménye



Az *Oszlopdiagram* menü eredménye



4.5. Oszlopdiagram rajzolása

Statisztikai adatokat szemléletesen ábrázolja az oszlopdiagram. Az *oszlop* eljárás *tomb* típusú tömbnek *k* adatát ábrázolja és az *sz* sztrigben megadott fejlécet írja a diagram fölé.

4.6. Éves statisztikai adatok ábrázolása oszlopdiagrammal

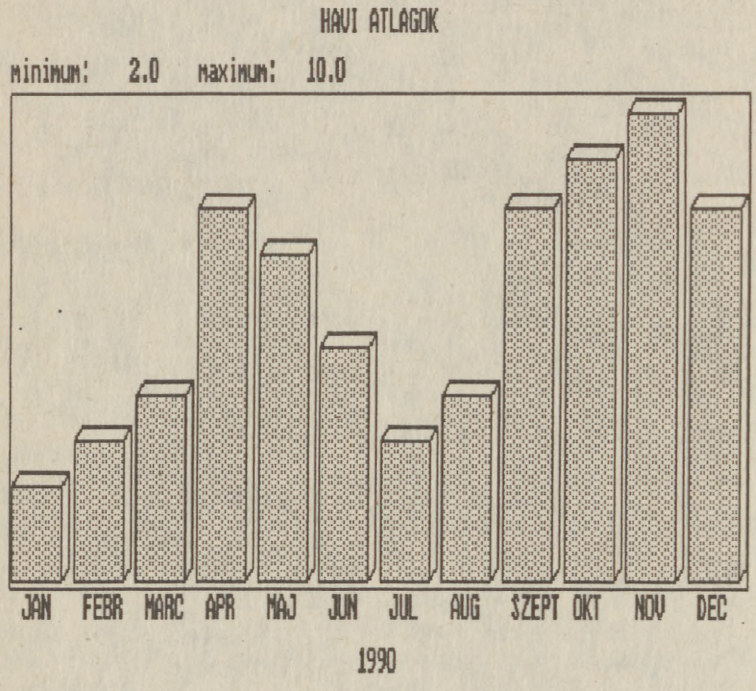
Éves statisztikai adatok ábrázolására alkalmas a *rect_ev* eljárás, amely *tomb12* típusú adatokat havi bontásban rajzolja meg, az *sz* sztringben megadott szöveget írja fejlécként és az *ev* változóban lévő számot írja a hónapok alá. A *típus* paraméter értéke határozza meg a diagram típusát, amely lehet:

1 térbeli oszlopdiagram,
2 síkbeli oszlopdiagram.

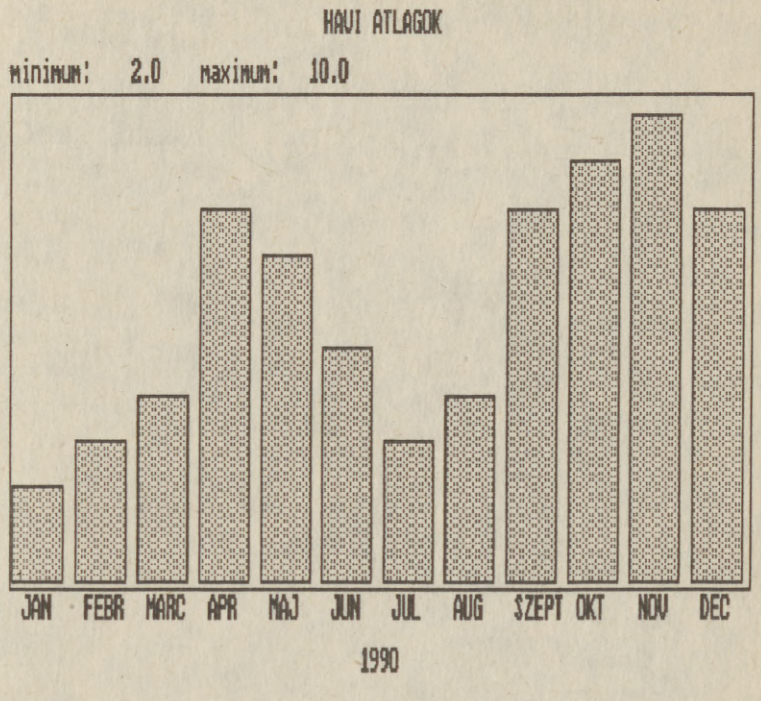
Indítsuk el az *ev_diag* programot. Az adatokat az alábbiakban kell megadni:

```
Adatok beolvasasa havi bontasban
Az adatok szama: 12
  1. adat : 2
  2. adat : 3
  3. adat : 4
  4. adat : 8
```

```
 5. adat : 7
 6. adat : 5
 7. adat : 3
 8. adat : 4
 9. adat : 8
10. adat : 9
11. adat : 10
12. adat : 8
A diagram fejlece: HAVI ATLAGOK
Evszam           : 1990
Abrazolas: oszlop (1) teglalap (2) : 1
```



Futtassuk le a programot ugyanezen adatokkal, azzal a különbséggel, hogy az ábrázolás most téglalappal történjen.



5. KOMPLEX ARITMETIKA

5.1. Algebrai műveletek komplex számokkal

Ebben a fejezetben azokkal a komplex műveletekkel foglalkozunk, amelyek szükségesek a komplex mátrixművelet végzéséhez.

5.1.1. Komplex szám abszolútértéke

A komplex szám algebrai alakja két valós számból áll, amely a komplex szám valós és képzetes részét képviseli. Legegyszerűbben úgy számolhatunk a komplex számokkal a Pascal programokban, hogy létrehozunk pl. egy *compl* nevű rekord típust:

```
type  compl = record
                re, im : real
              end;
```

Komplex szám abszolútértékét megkapjuk, ha a komplex szám valós és képzetes részének négyzetösszegéből gyököt vonunk.

$$|x| = \sqrt{(x.re)^2 + (x.im)^2}$$

Írjunk c_abs néven egy függvényt, ahol x komplex szám a bemenő paraméter és az eredmény a c_abs eljárás fejében keletkezik:

```
function c_abs(x:compl):real;
begin
    c_abs:=sqrt(sqr(x.re)+sqr(x.im));
end;
```

5.1.2. Komplex szám összeadása

Legyen x és y két komplex szám, összegük

$$z.re = x.re + y.re$$
$$z.im = y.im + y.im$$

Írjunk eljárást komplex számok összeadására, az eredmény z-ben keletkezzen:

```
procedure c_osszead(x,y:compl;var z:compl);
var w:compl;
begin
    w.re:=x.re+y.re;
    w.im:=x.im+y.im;
```

```
    z   :=w;
end;
```

5.1.3. Komplex szám kivonása

Legyen x és y két komplex szám, különbségük

$$z.re = x.re - y.re$$
$$z.im = x.im - y.im$$

Írjunk eljárást komplex számok kivonására, az eredmény z-ben keletkezzen:

```
procedure c_kivon(x,y:compl;var z:compl);
var w:compl;
begin
    w.re:=x.re-y.re;
    w.im:=x.im-y.im;
    z   :=w;
end;
```

5.1.4. Komplex szám szorzása

Legyen x és y két komplex szám, szorzatuk:

$$z.re = x.re * y.re - x.im * y.im$$
$$z.im = x.re * y.im + x.im * y.re$$

Írjunk eljárást két komplex szám szorzatára, az eredmény z-ben keletkezzen:

```
procedure c_szoroz(x,y:compl;var z:compl);
var w:compl;
begin
    w.re:=x.re*y.re-x.im*y.im;
    w.im:=x.re*y.im+x.im*y.re;
    z   :=w;
end;
```

5.1.5. Komplex szám osztása

Legyen x és y két komplex szám, hányadosuk

$$z.re = (x.re * y.re + x.im * y.im)/(y.re * y.re + y.im * y.im)$$
$$z.im = (y.re * x.im - x.re * y.im)/(y.re * y.re + y.im * y.im)$$

Írjunk eljárást két komplex szám osztására, az eredmény z-ben keletkezzen:

```
procedure c_oszt(x,y:compl;var z:compl);
var w:compl;
begin
     w.re:=(x.re*y.re+x.im*y.im)/sqr(c_abs(y));
     w.im:=(x.im*y.re-x.re*y.im)/sqr(c_abs(y));
     z   :=w;
end;
```

5.2. Műveletek komplex mátrixokkal

Ebben a fejezetben nem foglalkozunk elmélettel, az idevonatkozó anyagot megtaláljuk *IBM PC programozása Turbo Pascal nyelven* jegyzet 7. fejezetében, amely a numerikus módszerekkel foglalkozik.

A *KOMPLEX.pas* program menüvezérelve végez műveleteket mátrixokkal, amely nem kötelezően komplex. A menü a következő

```
┌─┤ Komplex matrix ├──┐
│Matrix olvasasa      │
│Vektor olvasasa      │
│Matrix kiiratasa     │
│Vektor kiiratasa     │
│Gauss elimináció     │
│Matrix determinansa  │
│LU dekompozicio      │
│Matrix invertalasa   │
│Sajat-ertek (Spur)   │
│Kilepes              │
└─────────────────────┘
```

Oldjuk meg az alábbi feladatot:

$$\begin{bmatrix} 3-5i & 9+i & -9-2i \\ 2+2i & 11-i & 14+3i \\ 4-i & 14+9i & 2-3i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1+i \\ 2+2i \\ 3+3i \end{bmatrix}$$

Először válasszuk ki a *Matrix olvasasa* menut. Ezen menüpont alatt nemcsak az adatok beolvasása történik meg, hanem itt kell megadni az eredmény file nevét, melyet utólag is kiértékelhetünk.

A futtatás a következő:

```
Az eredmeny file neve: ered.dat
A matrix merete: 3
```

```
1. sor:
   1.elem: valos: 3
        kepzetes: -5
   2.elem: valos: 9
        kepzetes: 1
   3.elem: valos: -9
        kepzetes: -2

2.sor:
  1.elem: valos: 2
       kepzetes: 2
  2.elem: valos: 11
       kepzetes: -1
  3.elem: valos: 14
       kepzetes: 3

2.sor:
  1.elem: valos: 4
       kepzetes: -1
  2.elem: valos: 14
       kepzetes: 9
  3.elem: valos: 2
       kepzetes: -3
```

A *matrix kiiratasa* menü alatt ellenőrihetjük újra a mátrix adatait:

```
MATRIX ELEMEI(3x3)
  3.000e+00-i 5.000e+00   9.000e+00+i 1.000e+00  -9.000E+00-i 2.000e+00
  2.000e+00+i 2.000e+00   1.100e+01-i 1.000e+00   1.400E+01+i 3.000e+00
  4.000e+00-i 1.000e+00   1.400e+01+i 9.000e+00   2.000E+00-i 3.000e+00
```

Válasszuk ki a *Vektor olvasasa* menüpontot:

Az egyenletrendszer jobb oldala:

```
1.sor: valos: 1
    kepzetes: 1
2.sor: valos: 2
    kepzetes: 2
3.sor: valos: 3
    kepzetes: 3
```

Ellenőrihetjük a vektor adatait, ha kiválasztjuk a *Vektor olvasasa* menut:

A VEKTOR ELEMEI:

```
   1.000E+00i 1.000E+00
   2.000E+00i 2.000e+00
   3.000E+00i 3.000E+00
```

5.2.1. Gauss elimináció teljes főelemkiválasztással

A *Gauss eliminacio* menu az alábbi eredményt szolgáltatja:

```
GAUSS ELIMINACIO
MATRIX ELEMEI (3x3)

 1.000E+00+i 0.000e+00  3.610E-03-i 2.166E-01  1.697E-01-i 1.805E-01
-1.455E-11+i 1.819E-12  1.000E+00+i 0.000E+00  1.167E-01+i 2.488E-01
-1.455E-11-i 1.819E-12  0.000E+00+i 0.000E+00  1.000E+00+i 0.000E+00

Az egyenletrendszer gyokei

 -6.689E-01+i 2.331E-01
  2.638E-01-i 7.983E-02
  1.258E-01+i 2.597E-01
```

5.2.2. Komplex mátrix determinánsa

A *Matrix determinansa* menu kiválasztására az alábbi eredményt láthatjuk

```
AZ LU DEKOMPONALT MATRIX
MATRIX ELEMEI (3 x 3)

 3.000E+00-i 5.000E+00   9.000E+00+i 1.000E+00   -9.000E+00-i 2.000E+00
-1.176E-01+i 4.706E-01   1.253E-01-i 5.118E+00    1.200E+01+i 7.000E+00
 5.000E-01+i 5.000E-01   5.723E-01+i 5.530E-01    2.504E+00-i 8.142E+00

A MATRIX DETERMINANSA

 -6.050E+02-i 2.930E+02
```

5.2.3. Komplex egyenletrendszer megoldása LU dekompozicióval

Az LU dekompozíció eredménye

```
AZ EGYENLETRENDSZER MEGOLDASA LU DEKOMPOZÍCIÓVAL

 -6.689E-01+i 2.331E-01
  2.638E-01-i 7.983E-02
  1.258E-01+i 2.597E-01
```

5.2.4. Komplex mátrix invertálása

Az invertálás eredménye

```
A MATRIX INVERZE
MATRIX ELEMEI ( 3 x 3)

 3.325E-01+i 1.745E-01   2.272E-01+i 2.882E-02  -3.349E-01+i 7.295E-02
-6.560E-02+i 3.177E-02  -2.586E-02+i 4.558E-02   6.977E-02-i 9.825E-02
 4.630E-03-i 1.031E-01   4.230E-02-i 8.330E-02   3.451E-02+i 1.122E-01
```

5.2.5. Komplex mátrix sajátértékei Spur-módszerrel

Spur módszerrel a sajátértékek

```
A MATRIX SAJATERTEKE

    1.000E+00+i 0.000E+00
   -1.600E+01+i 9.000E+00
   -1.090E+02-i 3.010E+02
    6.050E+02+i 2.930E+02
```

Az eredmény *ered.dat* néven is létrejött, párhuzamosan a képernyőre kiírt eredményekkel, tartalma az alábbi

```
Az eredmeny file neve : ered.dat

A matrix merete :3
A matrix elemei


 1.sor:
   1.elem:  valos:  3.000E+00
         kepzetes: -5.000E+00
   2.elem:  valos:  9.000E+00
         kepzetes:  1.000E+00
   3.elem:  valos: -9.000E+00
         kepzetes: -2.000E+00

 2.sor:
   1.elem:  valos:  2.000E+00
         kepzetes:  2.000E+00
   2.elem:  valos:  1.100E+01
         kepzetes: -1.000E+00
   3.elem:  valos:  1.400E+01
         kepzetes:  3.000E+00

 3.sor:
   1.elem:  valos:  4.000E+00
         kepzetes: -1.000E+00
   2.elem:  valos:  1.400E+01
         kepzetes:  9.000E+00
```

```
   3.elem:  valos:  2.000E+00
         kepzetes: -3.000E+00

MATRIX ELEMEI ( 3x 3)

   3.000E+00-i 5.000E+00   9.000E+00+i 1.000E+00  -9.000E+00-i 2.000E+00
   2.000E+00+i 2.000E+00   1.100E+01-i 1.000E+00   1.400E+01+i 3.000E+00
   4.000E+00-i 1.000E+00   1.400E+01+i 9.000E+00   2.000E+00-i 3.000E+00
Az egyenletrendszer jobb oldala:

 1.sor: valos:  1.000E+00
     kepzetes:  1.000E+00
 2.sor: valos:  2.000E+00
     kepzetes:  2.000E+00
 3.sor: valos:  3.000E+00
     kepzetes:  3.000E+00


A VEKTOR ELEMEI

   1.000E+00+i 1.000E+00

   2.000E+00+i 2.000E+00

   3.000E+00+i 3.000E+00

GAUSS ELIMINACIO
MATRIX ELEMEI ( 3x 3)

   1.000E+00+i 0.000E+00   3.610E-03-i 2.166E-01   1.697E-01-i 1.805E-01
  -1.455E-11+i 1.819E-12   1.000E+00+i 0.000E+00   1.167E-01+i 2.488E-01
  -1.455E-11-i 1.819E-12   0.000E+00+i 0.000E+00   1.000E+00+i 0.000E+00

Az egyenletrendszer gyokei

  -6.689E-01+i 2.331E-01
   2.638E-01-i 7.983E-02
   1.258E-01+i 2.597E-01

AZ LU DEKOMPONALT MATRIX
MATRIX ELEMEI ( 3x 3)

   3.000E+00-i 5.000E+00   9.000E+00+i 1.000E+00  -9.000E+00-i 2.000E+00
  -1.176E-01+i 4.706E-01   1.253E+01-i 5.118E+00   1.200E+01+i 7.000E+00
   5.000E-01+i 5.000E-01   5.723E-01+i 5.530E-01   2.504E+00-i 8.142E+00

A MATRIX DETERMINANSA

  -6.050E+02-i 2.930E+02
```

AZ EGYENLETRENDSZER MEGOLDASA LU DEKOMPOZICIOVAL

-6.689E-01+i 2.331E-01

2.638E-01-i 7.983E-02

1.258E-01+i 2.597E-01

A MATRIX INVERZE
MATRIX ELEMEI (3x 3)

3.325E-01+i 1.745E-01	2.272E-01+i 2.882E-02	-3.349E-01+i 7.295E-02
-6.560E-02+i 3.177E-02	-2.586E-02+i 4.558E-02	6.977E-02-i 9.825E-02
4.630E-03-i 1.031E-01	4.230E-02-i 8.330E-02	3.451E-02+i 1.122E-01

A MATRIX SAJATERTEKE

1.000E+00+i 0.000E+00

-1.600E+01+i 9.000E+00

-1.090E+02-i 3.010E+02

6.050E+02+i 2.930E+02

Oldjunk meg egy 4 ismeretlenes egyenletrendszert, melynek együttható-mátrixa és a jobb oldala nem tartalmaznak komplex számokat. Ha az adat nem komplex szám, akkor adat megadásánál a képzetes rész 0 lesz.

$$\begin{bmatrix} 3.06 & -2.18 & 1.63 & 0.85 \\ -0.72 & 0.88 & 1.13 & -2.25 \\ 1.41 & -3.18 & 2.11 & 0.63 \\ 1.21 & 0.82 & -3.12 & 0.61 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1.31 \\ 2.15 \\ 1.01 \\ 3.15 \end{bmatrix}$$

A *Matrix olvasasa* memü kiválasztásával beolvastatjuk az együttható mátrixot, majd a *Vektor olvasasa* menüben a jobb oldalt. Az eredményt az *ered1.dat* file-ba kérjük.

Az eredmény file tartalma:

```
Az eredmeny file neve : ered1.dat

A matrix merete :4
A matrix elemei

 1.sor:
   1.elem:  valos:  3.060E+00
         kepzetes:  0.000E+00
   2.elem:  valos: -2.180E+00
         kepzetes:  0.000E+00
   3.elem:  valos:  1.630E+00
         kepzetes:  0.000E+00
   4.elem:  valos:  8.500E-01
         kepzetes:  0.000E+00

 2.sor:
   1.elem:  valos: -7.200E-01
         kepzetes:  0.000E+00
   2.elem:  valos:  8.800E-01
         kepzetes:  0.000E+00
   3.elem:  valos:  1.130E+00
         kepzetes:  0.000E+00
   4.elem:  valos: -2.250E+00
         kepzetes:  0.000E+00
```

```
3.sor:
  1.elem:  valos:  1.410E+00
           kepzetes:  0.000E+00
  2.elem:  valos: -3.180E+00
           kepzetes:  0.000E+00
  3.elem:  valos:  2.110E+00
           kepzetes:  0.000E+00
  4.elem:  valos:  6.300E-01
           kepzetes:  0.000E+00

4.sor:
  1.elem:  valos:  1.210E+00
           kepzetes:  0.000E+00
  2.elem:  valos:  8.200E-01
           kepzetes:  0.000E+00
  3.elem:  valos: -3.120E+00
           kepzetes:  0.000E+00
  4.elem:  valos:  6.100E-01
           kepzetes:  0.000E+00

Az egyenletrendszer jobb oldala:

 1.sor: valos:  1.310E+00
     kepzetes:  0.000E+00
 2.sor: valos:  2.150E+00
     kepzetes:  0.000E+00
 3.sor: valos:  1.010E+00
     kepzetes:  0.000E+00
 4.sor: valos:  3.150E+00
     kepzetes:  0.000E+00


MATRIX ELEMEI ( 4x 4)

[ 1, 1]:  3.060E+00+i 0.000E+00
[ 1, 2]: -2.180E+00+i 0.000E+00
[ 1, 3]:  1.630E+00+i 0.000E+00
[ 1, 4]:  8.500E-01+i 0.000E+00

[ 2, 1]: -7.200E-01+i 0.000E+00
[ 2, 2]:  8.800E-01+i 0.000E+00
[ 2, 3]:  1.130E+00+i 0.000E+00
[ 2, 4]: -2.250E+00+i 0.000E+00

[ 3, 1]:  1.410E+00+i 0.000E+00
[ 3, 2]: -3.180E+00+i 0.000E+00
[ 3, 3]:  2.110E+00+i 0.000E+00
[ 3, 4]:  6.300E-01+i 0.000E+00

[ 4, 1]:  1.210E+00+i 0.000E+00
[ 4, 2]:  8.200E-01+i 0.000E+00
```

```
[ 4, 3]: -3.120E+00+i 0.000E+00
[ 4, 4]:  6.100E-01+i 0.000E+00


A VEKTOR ELEMEI

   1.310E+00+i 0.000E+00

   2.150E+00+i 0.000E+00

   1.010E+00+i 0.000E+00

   3.150E+00+i 0.000E+00

GAUSS ELIMINACIO
MATRIX ELEMEI ( 4x 4)

[ 1, 1]:  1.000E+00+i 0.000E+00
[ 1, 2]: -6.635E-01+i 0.000E+00
[ 1, 3]: -4.434E-01+i 0.000E+00
[ 1, 4]: -1.981E-01+i 0.000E+00

[ 2, 1]: -9.095E-13+i 0.000E+00
[ 2, 2]:  1.000E+00+i 0.000E+00
[ 2, 3]: -6.109E-01+i 0.000E+00
[ 2, 4]: -2.999E-01+i 0.000E+00

[ 3, 1]:  3.638E-12+i 0.000E+00
[ 3, 2]: -2.274E-13+i 0.000E+00
[ 3, 3]:  1.000E+00+i 0.000E+00
[ 3, 4]:  2.145E-01+i 0.000E+00

[ 4, 1]: -9.095E-13+i 0.000E+00
[ 4, 2]: -1.819E-12+i 0.000E+00
[ 4, 3]: -9.095E-13+i 0.000E+00
[ 4, 4]:  1.000E+00+i 0.000E+00


Az egyenletrendszer gyokei

   9.428E-01+i 0.000E+00
  -1.419E+00+i 0.000E+00
  -1.520E+00+i 0.000E+00
  -2.576E+00+i 0.000E+00

AZ LU DEKOMPONALT MATRIX
MATRIX ELEMEI ( 4x 4)

[ 1, 1]:  3.060E+00+i 0.000E+00
[ 1, 2]: -2.180E+00+i 0.000E+00
[ 1, 3]:  1.630E+00+i 0.000E+00
[ 1, 4]:  8.500E-01+i 0.000E+00
```

```
[ 2, 1]: -2.353E-01+i 0.000E+00
[ 2, 2]: -2.175E+00+i 0.000E+00
[ 2, 3]:  1.359E+00+i 0.000E+00
[ 2, 4]:  2.383E-01+i 0.000E+00

[ 3, 1]:  4.608E-01+i 0.000E+00
[ 3, 2]: -1.687E-01+i 0.000E+00
[ 3, 3]: -2.714E+00+i 0.000E+00
[ 3, 4]:  4.582E-01+i 0.000E+00

[ 4, 1]:  3.954E-01+i 0.000E+00
[ 4, 2]: -7.732E-01+i 0.000E+00
[ 4, 3]: -6.422E-01+i 0.000E+00
[ 4, 4]: -1.716E+00+i 0.000E+00


A MATRIX DETERMINANSA

  -3.099E+01+i 0.000E+00

AZ EGYENLETRENDSZER MEGOLDASA LU DEKOMPOZICIOVAL

   9.428E-01+i 0.000E+00

  -1.419E+00+i 0.000E+00

  -1.520E+00+i 0.000E+00

  -2.576E+00+i 0.000E+00


A MATRIX INVERZE
MATRIX ELEMEI ( 4x 4)

[ 1, 1]:  4.127E-01+i 0.000E+00
[ 1, 2]:  1.250E-01+i 0.000E+00
[ 1, 3]: -2.193E-01+i 0.000E+00
[ 1, 4]:  1.126E-01+i 0.000E+00

[ 2, 1]:  4.256E-01+i 0.000E+00
[ 2, 2]: -1.253E-01+i 0.000E+00
[ 2, 3]: -7.210E-01+i 0.000E+00
[ 2, 4]: -3.107E-01+i 0.000E+00

[ 3, 1]:  3.090E-01+i 0.000E+00
[ 3, 2]: -9.841E-02+i 0.000E+00
[ 3, 3]: -3.504E-01+i 0.000E+00
[ 3, 4]: -4.317E-01+i 0.000E+00

[ 4, 1]:  1.895E-01+i 0.000E+00
[ 4, 2]: -5.829E-01+i 0.000E+00
```

```
[ 4, 3]: -3.878E-01+i 0.000E+00
[ 4, 4]: -3.743E-01+i 0.000E+00


A MATRIX SAJATERTEKE

  1.000E+00+i 0.000E+00

 -6.660E+00+i 0.000E+00

  1.720E+01+i 0.000E+00

 -1.355E+01+i 0.000E+00

 -3.099E+01+i 0.000E+00
```

A. FÜGGELÉK

MINTAPROGRAMOK LISTÁJA

1. Rendező algoritmusok: RENDEZ.pas

```
UNIT rendez;

INTERFACE
const db=50;
type
      itype = array[1..db] of integer;
      ibuff = integer;

      rtype = array[1..db] of real;
      rbuff = real;

      stype = array[1..db] of string;
      sbuff = string;

procedure olvas_int(var x:itype; var m:integer);
procedure kiir_int(x:itype; m:integer);
procedure olvas_real(var x:rtype; var m:integer);
procedure kiir_real(x:rtype; m:integer);
procedure olvas_str(var x:stype; var m:integer);
procedure kiir_str(x:stype; m:integer);

{ beszuro rendezes }
procedure beszur_int(var x:itype; m:integer);
procedure beszur_real(var x:rtype; m:integer);
procedure beszur_str(var x:stype; m:integer);

{ beszuro binaris rendezes }
procedure binaris_int(var x:itype; m:integer);
procedure binaris_real(var x:rtype; m:integer);
procedure binaris_str(var x:stype; m:integer);

{ kozvetlen kivalasztas  }
procedure kozv_int(var x:itype; m:integer);
procedure kozv_real(var x:rtype; m:integer);
procedure kozv_str(var x:stype; m:integer);

{ rendezes buborek modszerrel }
procedure buborek_int(var x:itype; m:integer);
procedure buborek_real(var x:rtype; m:integer);
procedure buborek_str(var x:stype; m:integer);

{ rendezes kevero modszerrel }
procedure kevero_int(var x:itype; m:integer);
procedure kevero_real(var x:rtype; m:integer);
procedure kevero_str(var x:stype; m:integer);
```

```
{ rendezes shell modszerrel }
procedure shell_int(var x:itype; m:integer);
procedure shell_real(var x:rtype; m:integer);
procedure shell_str(var x:stype; m:integer);

{ rendezes kupac modszerrel }
procedure kupac_int(var x:itype; m:integer);
procedure kupac_real(var x:rtype; m:integer);
procedure kupac_str(var x:stype; m:integer);

{ rendezes quick (gyors) modszerrel }
procedure quick_int(var x:itype; m:integer);
procedure quick_real(var x:rtype; m:integer);
procedure quick_str(var x:stype; m:integer);


IMPLEMENTATION

procedure olvas_int(var x:itype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (integer tipusu) adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_int(x:itype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;

procedure olvas_real(var x:rtype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (real tipusu) adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
```

```
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_real(x:rtype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;

procedure olvas_str(var x:stype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (string tipusu) adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_str(x:stype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;

procedure beszur_int(var x:itype; m:integer);
var i,j:integer;
    buf:ibuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       j  := i-1;
```

```
        while (j>0) and (buf<x[j]) do
        begin
             x[j+1] := x[j];
                  j := j-1;
         end;
        x[j+1] := buf;
    end;
end;

procedure beszur_real(var x:rtype; m:integer);
var i,j:integer;
    buf:rbuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       j  := i-1;
       while (j>0) and (buf<x[j]) do
       begin
            x[j+1] := x[j];
                 j := j-1;
        end;
        x[j+1] := buf;
    end;
end;

procedure beszur_str(var x:stype; m:integer);
var i,j:integer;
    buf:sbuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       j  := i-1;
       while (j>0) and (buf<x[j]) do
       begin
            x[j+1] := x[j];
                 j := j-1;
        end;
        x[j+1] := buf;
    end;
end;

procedure binaris_int(var x:itype; m:integer);
var i,j,l,r,bm:integer;
    buf       :ibuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       l  := 1;
       r  := i-1;
```

```
        while l<=r do
        begin
             bm:=(l+r) div 2;
             if buf < x[bm] then r:=bm-1 else l:=bm+1
      end;
      for j:=i-1 downto l do x[j+1]:=x[j];
        x[l] := buf;
    end;
end;

procedure binaris_real(var x:rtype; m:integer);
var i,j,l,r,bm:integer;
    buf       :rbuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       l  :=1;
       r  := i-1;
       while l<=r do
       begin
            bm:=(l+r) div 2;
            if buf < x[bm] then r:=bm-1 else l:=bm+1
      end;
      for j:=i-1 downto l do x[j+1]:=x[j];
        x[l] := buf;
    end;
end;

procedure binaris_str(var x:stype; m:integer);
var i,j,l,r,bm:integer;
    buf       :sbuff;
begin
    for i:=2 to m do
    begin
       buf:= x[i];
       l  :=1;
       r  := i-1;
       while l<=r do
       begin
            bm:=(l+r) div 2;
            if buf < x[bm] then r:=bm-1 else l:=bm+1
      end;
      for j:=i-1 downto l do x[j+1]:=x[j];
        x[l] := buf;
    end;
end;

procedure kozv_int(var x:itype; m:integer);
var i,j,k:integer;
    buf  :ibuff;
begin
```

```
    for i:=1 to m-1 do
    begin
       k:=i; buf:=x[i];
       for j:=i+1 to m do
          if x[j]<buf then
          begin
              k:=j; buf:=x[j]
          end;
        x[k]:= x[i]; x[i]:= buf;
    end;
end;

procedure kozv_real(var x:rtype; m:integer);
var i,j,k:integer;
    buf  :rbuff;
begin
    for i:=1 to m-1 do
    begin
       k:=i; buf:=x[i];
       for j:=i+1 to m do
          if x[j]<buf then
          begin
              k:=j; buf:=x[j]
          end;
        x[k]:= x[i]; x[i]:= buf;
    end;
end;

procedure kozv_str(var x:stype; m:integer);
var i,j,k:integer;
    buf  :sbuff;
begin
    for i:=1 to m-1 do
    begin
       k:=i; buf:=x[i];
       for j:=i+1 to m do
          if x[j]<buf then
          begin
              k:=j; buf:=x[j]
          end;
        x[k]:= x[i]; x[i]:= buf;
    end;
end;

procedure buborek_int(var x:itype; m:integer);
var i,j:integer;
    buf:ibuff;
begin
    for i:=2 to m do
    begin
       for j:=m downto i do
         if x[j-1]>x[j] then
```

```
          begin
             buf   :=x[j-1];
             x[j-1]:=x[j];
             x[j]  :=buf;
          end;
    end;
end;

procedure buborek_real(var x:rtype; m:integer);
var i,j:integer;
    buf:rbuff;
begin
    for i:=2 to m do
    begin
       for j:=m downto i do
         if x[j-1]>x[j] then
         begin
             buf   :=x[j-1];
             x[j-1]:=x[j];
             x[j]  :=buf;
        end;
    end;
end;

procedure buborek_str(var x:stype; m:integer);
var i,j:integer;
    buf:sbuff;
begin
    for i:=2 to m do
    begin
       for j:=m downto i do
         if x[j-1]>x[j] then
         begin
             buf   :=x[j-1];
             x[j-1]:=x[j];
             x[j]  :=buf;
        end;
    end;
end;

procedure kevero_int(var x:itype; m:integer);
var      j,k,l,r :integer;
             buf :ibuff;
begin
    l:=2; r:=m; k:=m;
    repeat
      for j:=r downto l do
        if x[j-1]>x[j] then
        begin
            buf:=x[j-1];
            x[j-1]:= x[j];
            x[j]:=buf;
```

```
            k:=j;
          end;
          l:=k+1;
        for j:=l to r do
          if x[j-1]>x[j] then
          begin
              buf:=x[j-1];
              x[j-1]:= x[j];
              x[j]:=buf;
              k:=j;
          end;
          r:=k-1;
      until l>r;
end; { kevero_int }

procedure kevero_real(var x:rtype; m:integer);
var       j,k,l,r :integer;
              buf :rbuff;
begin
      l:=2; r:=m; k:=m;
      repeat
        for j:=r downto l do
          if x[j-1]>x[j] then
          begin
              buf:=x[j-1];
              x[j-1]:= x[j];
              x[j]:=buf;
              k:=j;
          end;
          l:=k+1;
        for j:=l to r do
          if x[j-1]>x[j] then
          begin
              buf:=x[j-1];
              x[j-1]:= x[j];
              x[j]:=buf;
              k:=j;
          end;
          r:=k-1;
      until l>r;
end; { kevero_real }

procedure kevero_str(var x:stype; m:integer);
var       j,k,l,r :integer;
              buf :sbuff;
begin
      l:=2; r:=m; k:=m;
      repeat
        for j:=r downto l do
          if x[j-1]>x[j] then
          begin
              buf:=x[j-1];
```

```
            x[j-1]:= x[j];
            x[j]:=buf;
            k:=j;
        end;
        l:=k+1;
      for j:=l to r do
        if x[j-1]>x[j] then
        begin
            buf:=x[j-1];
            x[j-1]:= x[j];
            x[j]:=buf;
            k:=j;
        end;
        r:=k-1;
    until l>r;
end; { kevero_str }

procedure shell_int(var x:itype; m:integer);
const
    t=5;
var i,j,k,s,ms:integer;
    buf       :ibuff;
    h         :array[1..t] of integer;
begin
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=3; h[5]:=1;
    for ms:=1 to t do
    begin
        k:=h[ms];
        s:=-k;
    for i:=k+1 to m do
    begin
       buf:= x[i];
       j  := i-k;
       if s=0 then
       begin
          s:=-k;
          s:=s+1;
          x[s]:=buf;
       end;
       while (j>0) and (buf<x[j]) and (j<=m) do
       begin
            x[j+k] := x[j];
                j := j-k;
        end;
        x[j+k] := buf;
    end;
  end;
end;
```

```
procedure shell_real(var x:rtype; m:integer);
const
    t=5;
var i,j,k,s,ms:integer;
    buf       :rbuff;
    h         :array[1..t] of integer;
begin
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=3; h[5]:=1;
    for ms:=1 to t do
    begin
        k:=h[ms];
        s:=-k;
    for i:=k+1 to m do
    begin
       buf:= x[i];
       j  := i-k;
       if s=0 then
       begin
          s:=-k;
          s:=s+1;
          x[s]:=buf;
       end;
       while (j>0) and (buf<x[j]) and (j<=m) do
       begin
            x[j+k] := x[j];
                 j := j-k;
        end;
        x[j+k] := buf;
    end;
  end;
end;

procedure shell_str(var x:stype; m:integer);
const
    t=5;
var i,j,k,s,ms:integer;
    buf       :sbuff;
    h         :array[1..t] of integer;
begin
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=3; h[5]:=1;
    for ms:=1 to t do
    begin
        k:=h[ms];
        s:=-k;
    for i:=k+1 to m do
    begin
       buf:= x[i];
       j  := i-k;
       if s=0 then
       begin
          s:=-k;
          s:=s+1;
```

```
          x[s]:=buf;
        end;
        while (j>0) and (buf<x[j]) and (j<=m) do
        begin
             x[j+k] := x[j];
                  j := j-k;
         end;
         x[j+k] := buf;
    end;
  end;
end;

procedure kupac_int(var x:itype; m:integer);
var l,r:integer;
    buf:ibuff;
    procedure sully;
    label ki;
    var i,j:integer;
    begin
      i:=l; j:=2*i; buf:=x[i];
      while j<=r do
      begin
       if j<r then
         if x[j]<x[j+1] then j:=j+1;
       if buf >=x[j] then goto ki;
       x[i]:=x[j]; i:=j; j:=2*i;
     end;
ki: x[i]:=buf;
    end;
begin
    l:=(m div 2)+1; r:=m;
    while l>1 do
       begin  l:=l-1; sully;
       end;
    while r>1 do
        begin buf:=x[1]; x[1]:=x[r]; x[r]:=buf;
              r:=r-1; sully;
        end;
end;

procedure kupac_real(var x:rtype; m:integer);
var l,r:integer;
    buf:rbuff;
    procedure sully;
    label ki;
    var i,j:integer;
    begin
      i:=l; j:=2*i; buf:=x[i];
      while j<=r do
      begin
       if j<r then
         if x[j]<x[j+1] then j:=j+1;
```

```
        if buf >=x[j] then goto ki;
        x[i]:=x[j]; i:=j; j:=2*i;
      end;
ki: x[i]:=buf;
    end;
begin
    l:=(m div 2)+1; r:=m;
    while l>1 do
       begin  l:=l-1; sully;
       end;
    while r>1 do
        begin buf:=x[1]; x[1]:=x[r]; x[r]:=buf;
              r:=r-1; sully;
        end;
end;

procedure kupac_str(var x:stype; m:integer);
var l,r:integer;
    buf:sbuff;
    procedure sully;
    label ki;
    var i,j:integer;
    begin
      i:=l; j:=2*i; buf:=x[i];
      while j<=r do
      begin
       if j<r then
         if x[j]<x[j+1] then j:=j+1;
       if buf >=x[j] then goto ki;
       x[i]:=x[j]; i:=j; j:=2*i;
      end;
ki: x[i]:=buf;
    end;
begin
    l:=(m div 2)+1; r:=m;
    while l>1 do
       begin  l:=l-1; sully;
       end;
    while r>1 do
        begin buf:=x[1]; x[1]:=x[r]; x[r]:=buf;
              r:=r-1; sully;
        end;
end;

procedure quick_int(var x:itype; m:integer);
   procedure qs(l,r: integer; var ix: itype);
   var i,j       :integer;
       buf1,buf2 :ibuff;
   begin
      i:=l; j:=r;
      buf1:=ix[(l+r) div 2];
```

```
      repeat
        while ix[i] < buf1 do i:=i+1;
        while buf1 < ix[j] do j:=j-1;
        if i<=j then
        begin
            buf2:=ix[i];
            ix[i]:= ix[j];
            ix[j]:=buf2;
            i:=i+1; j:=j-1;
        end;
      until i>j;
      if l<j then qs(l,j,ix);
      if l<r then qs(i,r,ix);
   end;
begin
      qs(1,m,x);
end; {quick_int}

procedure quick_str(var x:stype; m:integer);
   procedure qs(l,r: integer; var ix: stype);
   var i,j       :integer;
       buf1,buf2 :sbuff;
   begin
      i:=l; j:=r;
      buf1:=ix[(l+r) div 2];
      repeat
        while ix[i] < buf1 do i:=i+1;
        while buf1 < ix[j] do j:=j-1;
        if i<=j then
        begin
            buf2:=ix[i];
            ix[i]:= ix[j];
            ix[j]:=buf2;
            i:=i+1; j:=j-1;
        end;
      until i>j;
      if l<j then qs(l,j,ix);
      if l<r then qs(i,r,ix)
   end;
begin
      qs(1,m,x);
end; {quick_int}

procedure quick_real(var x:rtype; m:integer);
   procedure qs(l,r: integer; var ix: rtype);
   var i,j       :integer;
       buf1,buf2 :rbuff;
   begin
      i:=l; j:=r;
      buf1:=ix[(l+r) div 2];
```

```
      repeat
        while ix[i] < buf1 do i:=i+1;
        while buf1 < ix[j] do j:=j-1;
        if i<=j then
        begin
            buf2:=ix[i];
            ix[i]:= ix[j];
            ix[j]:=buf2;
            i:=i+1; j:=j-1;
        end;
      until i>j;
      if l<j then qs(l,j,ix);
      if l<r then qs(i,r,ix)
   end;
begin
     qs(1,m,x);
end; {quick_int}
BEGIN
END.
```

2. Beszúró rendezés: REND1.pas

```
program beszuro_rendezes;
{ rend1.pas }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
writeln('BESZURO MODSZER');
 (* egesz tipusu adatok rendezese  *)
   olvas_int(y,n);
   beszur_int(y,n);
   kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
   olvas_real(r,n);
   beszur_real(r,n);
   kiir_real(r,n);

 (* sztring tipusu adatok rendezese *)
   olvas_str(s,n);
   beszur_str(s,n);
   kiir_str(s,n);
end.
```

3. Beszúró bináris rendezés: REND2.pas

```
program beszuro_bin_rendezes;
{ rend2.pas }
{ rendezes binaris beszurassal }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
  writeln('RENDEZES BINARIS BESZURASSAL ');
 (* egesz tipusu adatok rendezese  *)
   olvas_int(y,n);
   binaris_int(y,n);
   kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
   olvas_real(r,n);
   binaris_real(r,n);
   kiir_real(r,n);

 (* sztring tipusu adatok rendezese *)
   olvas_str(s,n);
   binaris_str(s,n);
   kiir_str(s,n);
end.
```

4. Közvetlen kiválasztás: REND3.pas

```
program kozvetlen_rendezes;
{ rendezes kozvetlen kivalasztassal }
{ rend3.pas }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
  writeln('RENDEZES KOZVETLEN KIVALASZTASSAL');
 (* egesz tipusu adatok rendezese  *)
   olvas_int(y,n);
   kozv_int(y,n);
   kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
   olvas_real(r,n);
   kozv_real(r,n);
   kiir_real(r,n);
```

```
 (* sztring tipusu adatok rendezese *)
   olvas_str(s,n);
   kozv_str(s,n);
   kiir_str(s,n);
end.
```

5. Rendezés buborék módszerrel: REND4.pas

```
program buborek_rendezes;
{ rend4.pas }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
 writeln('BUBOREK MODSZER');
 (* egesz tipusu adatok rendezese  *)
   olvas_int(y,n);
   buborek_int(y,n);
   kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
   olvas_real(r,n);
   buborek_real(r,n);
   kiir_real(r,n);

 (* sztring tipusu adatok rendezese *)
   olvas_str(s,n);
   buborek_str(s,n);
   kiir_str(s,n);
end.
```

6. Keverő rendezés: REND5.pas

```
program kevero_rendezes;
{ rend5.pas }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
 writeln('KEVERO MODSZER');
 (* egesz tipusu adatok rendezese  *)
   olvas_int(y,n);
   kevero_int(y,n);
   kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
   olvas_real(r,n);
```

```
   kevero_real(r,n);
   kiir_real(r,n);

 (* sztring tipusu adatok rendezese *)
   olvas_str(s,n);
   kevero_str(s,n);
   kiir_str(s,n);
end.
```

7. Shell-rendezés: REND6.pas

```
program shell_rendezes;
{ rend6.pas }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
 writeln('SHELL MODSZER');
 (* egesz tipusu adatok rendezese  *)
   olvas_int(y,n);
   shell_int(y,n);
   kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
   olvas_real(r,n);
   shell_real(r,n);
   kiir_real(r,n);

 (* sztring tipusu adatok rendezese *)
   olvas_str(s,n);
   shell_str(s,n);
   kiir_str(s,n);
end.
```

8. Kupac rendezés: REND7.pas

```
program sullyest_rendezes;
{ Rendezes kupac eljarassal }
{ rend7.pas }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
writeln('KUPACRENDEZES');
 (* egesz tipusu adatok rendezese  *)
   olvas_int(y,n);
```

```
  kupac_int(y,n);
  kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
  olvas_real(r,n);
  kupac_real(r,n);
  kiir_real(r,n);

 (* sztring tipusu adatok rendezese *)
  olvas_str(s,n);
  kupac_str(s,n);
  kiir_str(s,n);
end.
```

9. Quick rendezés: REND8.pas

```
program quick_rendezes;
{ rend8.pas }
uses rendez;
var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
begin
 writeln('QUICK MODSZER');
 (* egesz tipusu adatok rendezese  *)
  olvas_int(y,n);
  quick_int(y,n);
  kiir_int(y,n);

 (* valos tipusu adatok rendezese   *)
  olvas_real(r,n);
  quick_real(r,n);
  kiir_real(r,n);

 (* sztring tipusu adatok rendezese *)
  olvas_str(s,n);
  quick_str(s,n);
  kiir_str(s,n);
end.
```

10. Rekord rendezése: REKORD.pas

```
program rekord_rendezes;
{ rekord rendezes quick modszerrel }
{ rekord.pas }
const db=50;
type
      cim = record
          nev   : string[25];
          utca  : string[20];
          varos : string[20];
          orszag: string[2];
          kod   : string[5];
       end;
     cim_t = array[1..db] of cim;

var   lakos : cim_t;
      r     : cim;
      n     : integer;

procedure olvas_rec(var x:cim_t; var m:integer);
var i:integer;
begin
   writeln('Rendezendo adatok');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=1 to m do
   begin
      write(' Nev    (max. 25 kar.)  = '); readln(x[i].nev);
      write(' Utca   (max. 20 kar.)  = '); readln(x[i].utca);
      write(' Varos  (max. 20 kar.)  = '); readln(x[i].varos);
      write(' Orszag (max. 2 kar. )  = '); readln(x[i].orszag);
      write(' Kod    (max. 5.kar. )  = '); readln(x[i].kod);
      writeln;
   end;
end;

procedure kiir_rec(x:cim_t; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' Nev    : ',x[i].nev);
      writeln(' Utca   : ',x[i].utca);
      writeln(' Varos  : ',x[i].varos);
      writeln(' Orszag : ',x[i].orszag);
      writeln(' Kod    : ',x[i].kod);
      writeln;
```

```
   end;
   writeln;
end;

procedure qs_rekord(var x:cim_t; m:integer);
   procedure qs(l,r:integer; var xx:cim_t);
   var  i,j      :integer;
        buf1,buf2:cim;
   begin
      i:=l; j:=r;
      buf1:=xx[(l+r) div 2];
      repeat
        while xx[i].nev < buf1.nev  do i:=i+1;
        while buf1.nev  < xx[j].nev do j:=j-1;
        if i<=j then
        begin
           buf2 :=xx[i];
           xx[i]:=xx[j];
           xx[j]:=buf2;
           i:=i+1; j:=j-1;
       end;
     until i>j;
     if l<j then qs(l,j,xx);
     if l<r then qs(i,r,xx)
   end;
begin
     qs(1,m,x);
end;

begin
 writeln('REKORD ADATAINAK RENDEZESE');
 (* rekord adatok rendezese  *)
   olvas_rec(lakos,n);
   qs_rekord(lakos,n);
   kiir_rec(lakos,n);
end.
```

11. Természetes összefésülés: TERM_F.pas

```
program rendez;
{ A program adatokat rendez }
{ termeszetes osszefesulessel }
{ term_f.pas }
const
      l=50;
      l1=101;
type  t1=array[1..l] of real;
      t2=array[1..l1] of real;

var
      y: t1;
```

```
      db : integer;
      i  : integer;

procedure olvas(var x:t1; var m:integer);
begin
  writeln('Adatok rendezese termeszetes osszefesulessel ');
  repeat
     write('Az adatok szama : '); readln(m);
     writeln;
  until m<=100;
  for i:=1 to db do
  begin
    write(i:2,'. adat : '); readln(x[i]);
  end;
end;

procedure kiir_int(x:t1; m:integer);
var i:integer;
begin
   writeln;
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]:10);
   end;
   writeln;
end;

procedure kiir_intb(x:t2; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]:10);
   end;
   writeln;
end;

procedure term_osszef(var c: t1; n:integer);
var vege: boolean;
    a,b :t2;
    i,j,k : integer;
    min,max : real;
procedure minmax(w:t1; k:integer);
var i:integer;
begin
   min:=w[1]; max:=w[1];
   for i:=2 to k do
   begin
```

```
      if w[i]>max then max:=w[i];
      if w[i]<min then min:=w[i];
    end;
end;
procedure lancolas;
begin
    i:=1; j:=1; k:=1;
    repeat
      a[j]:=c[i]; i:=i+1; j:=j+1;
      while (i<=n) and (c[i-1]<=c[i]) do
      begin
         a[j]:=c[i]; i:=i+1; j:=j+1;
      end;
      a[j]:=min; j:=j+1;
      if i<=n then
        begin
          b[k]:=c[i]; k:=k+1; i:=i+1;
        end
      else  if k>1 then k:=k-1;
      while (i<=n) and (c[i-1]<=c[i]) do
      begin
        b[k]:=c[i]; i:=i+1; k:=k+1;
      end;
      b[k]:=min; k:=k+1;
    until i>n;
    b[k]:=max; a[j]:=max;
end;
procedure ossze;
begin
     i:=1; j:=1; k:=1;
     if b[1]=min then k:=2;
     repeat
       if a[j]<b[k] then
       begin
         c[i]:=a[j]; i:=i+1; j:=j+1;
       end
      else
       begin
         c[i]:=b[k]; i:=i+1; k:=k+1;
       end;
     if a[j]= min then
       while (b[k]<>min) and (b[k]<>max) do
       begin
         c[i]:=b[k]; i:=i+1; k:=k+1;
       end
      else
      begin
     if b[k]=min then
       while (a[j]<>min) and (a[j]<>max) do
       begin
          c[i]:=a[j]; i:=i+1; j:=j+1;
      end;
```

```
    end;
    if b[k]=min then k:=k+1;
    if a[j]=min then j:=j+1;
 until i>n;
end;
begin
   vege:=false;
   minmax(c,n);
   min:=min-1; max:=max+1;
  repeat
   lancolas;
   if k=2 then vege:=true
     else  ossze;
  until vege;
end;

begin
  olvas(y,db);
  term_osszef(y,db);
  kiir_int(y,db);
end.
```

12. Random diszk file rendezés: SORT_RF.pas

```
program random_disk_file_rendezes;
uses Dos;
{ random disk file rendezes quick modszerrel }
{ sort_rf.pas }
const db=50;
type
      cim = record
          nev   : string[25];
          utca  : string[20];
          varos : string[20];
          orszag: string[2];
          kod   : string[5];
       end;
     cim_t = array[0..db] of cim;

     str80 = string[80];
     recf  = file of cim;

var   lakos : cim_t;
      adat  : cim;
      t,t2  : integer;
      r     : cim;
      n     : integer;
      tfile : recf;
    file_nev: string[12];
```

```
function FileExists: boolean;
var
     s:searchrec;
begin
    FindFirst(file_nev,anyfile,s);
    FileExists:=doserror=0;
end;

procedure kiir_rec(var x:cim_t; var m:integer);
Label Ujra;
var i:integer;
    c:char;
begin
      writeln('Rendezendo adatok');
Ujra: write(' File neve : '); readln(file_nev);
    · if FileExists then
                  begin
                     writeln('        ',file_nev,' letezik!');
                     write('        Felulirhato (i/n): ');
                     readln(c); c:=upcase(c);
                     writeln;
                    if (c<>'I') then goto Ujra;
                  end;
   assign(tfile,file_nev);
   rewrite(tfile);
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=0 to m-1 do
   begin
      write(' Nev    (max. 25 kar.) = '); readln(x[i].nev);
      write(' Utca   (max. 20 kar.) = '); readln(x[i].utca);
      write(' Varos  (max. 20 kar.) = '); readln(x[i].varos);
      write(' Orszag (max. 2  kar.) = '); readln(x[i].orszag);
      write(' Kod    (max. 5  kar.) = '); readln(x[i].kod);
      write(tfile,x[i]);
      writeln;
   end;
   close(tfile);
end;

procedure olvas_rec_rend(var x:cim_t;var m:integer);
Label Ujra;
var i:integer;
    c:char;
begin
   writeln('Rendezett adatok ');
   i:=-1;
```

```
Ujra: write(' File neve : '); readln(file_nev);
      if not(FileExists) then
                begin
                   writeln('        ',file_nev,' nem letezik!');
                   goto Ujra;
                end;

   assign(tfile,file_nev);
   reset(tfile);
   repeat
      i:=i+1;
      seek(tfile,i);
      read(tfile,x[i]);

      writeln(' Nev    : ',x[i].nev);
      writeln(' Utca   : ',x[i].utca);
      writeln(' Varos  : ',x[i].varos);
      writeln(' Orszag : ',x[i].orszag);
      writeln(' Kod    : ',x[i].kod);
      writeln;
   until eof(tfile);
   writeln;
   m:=i+1;
   writeln('Az adatok szama : ',m);
   close(tfile);
end;

procedure olvas_rec_vizsg(var x:cim_t;var m:integer);
var i:integer;
begin
   writeln('Vizsgalt adatok ');
   i:=-1;
   assign(tfile,file_nev);
   reset(tfile);
   repeat
      i:=i+1;
      read(tfile,x[i]);
      writeln(' Nev    : ',x[i].nev);
      writeln(' Utca   : ',x[i].utca);
      writeln(' Varos  : ',x[i].varos);
      writeln(' Orszag : ',x[i].orszag);
      writeln(' Kod    : ',x[i].kod);
      writeln;
   until eof(tfile);
   writeln;
   m:=i+1;
   writeln('Az adatok szama : ',m);
end;
```

```
function keres(var f:recf; i:integer): str80;
var  p: cim;
begin
    i:=i-1;
    seek(f,i);
    read(f,p);
    keres:=p.nev;
end;

procedure qs_rekord(var f:recf; m:integer);
   procedure qs(l,r:integer);
   var  i,j,s          :integer;
        buf1,buf2,buf3:cim;
   begin
      i:=l; j:=r;
      s:=(l+r) div 2;
      seek(f,s-1);
      read(f,buf1);
      repeat
        while keres(f,i) < buf1.nev  do i:=i+1;
        while buf1.nev  < keres(f,j) do j:=j-1;
        if i<=j then
        begin
           seek(f,i-1); read(f,buf2);
           seek(f,j-1); read(f,buf3);
           seek(f,j-1); write(f,buf2);
           seek(f,i-1); write(f,buf3);
          i:=i+1; j:=j-1;
       end;
     until i>j;
     if l<j then qs(l,j);
     if l<r then qs(i,r)
   end;
begin
     qs(1,m);
end;

begin
 writeln('REKORD ADATAINAK RENDEZESE QUICK MODSZERREL');
 (* rekord adatok rendezese  *)
   kiir_rec(lakos,n);
   olvas_rec_vizsg(lakos,n);
   qs_rekord(tfile,n);
   olvas_rec_rend(lakos,n);
end.
```

13. Keresési algoritmusok: KERESES.pas

```
program kereses;
{ szekvencialis es binaris kereses }
{ kereses.pas }
const db=50;
type
      itype = array[1..db] of integer;
      ibuff = integer;

      rtype = array[1..db] of real;
      rbuff = real;

      stype = array[1..db] of string;
      sbuff = string;

var   y: itype;
      r: rtype;
      s: stype;
      n: integer;
      j: integer;
      adat : ibuff;

procedure olvas_int(var x:itype; var m:integer);
var i:integer;
begin
   writeln('Integer tipusu adatok beolvasasa');
 repeat
   write('Az adatok szama : '); readln(m);
   if m>db then writeln('Az adatok max. szama : ',db);
 until m<=db;
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_int(x:itype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;
```

```
procedure olvas_str(var x:stype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (string tipusu) adatok');
   write('Az adatok szama : '); readln(m);
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_str(x:stype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;

procedure olvas_real(var x:rtype; var m:integer);
var i:integer;
begin
   writeln('Rendezendo (real tipusu) adatok');
   write('Az adatok szama : '); readln(m);
   for i:=1 to m do
   begin
      write(' [',i:2,']= '); readln(x[i]);
   end;
end;

procedure kiir_real(x:rtype; m:integer);
var i:integer;
begin
   writeln('Rendezett adatok ');
   writeln('Az adatok szama : ',m);
   for i:=1 to m do
   begin
      writeln(' [',i:2,']= ',x[i]);
   end;
   writeln;
end;

function seq_keres(x: itype; m:integer; adat:ibuff):integer;
var i:integer;
begin
    i:=1;
    while (adat <> x[i]) and (i<=m) do i:=i+1;
    if i>m then seq_keres:=0
```

```
          else seq_keres:=i;
end; { seq_keres }

function bin_keres(x: itype; m:integer; adat:ibuff):integer;
var     a,f,k :integer;
        talal :boolean;
begin
      a:=1; f:=m;
      talal :=false;
      while (a<=f) and (not talal) do
      begin
          k:=(a+f) div 2;
          if adat<x[k] then f:=k-1
                       else
                  if adat>x[k] then a:=k+1
                       else talal :=true; { megtalalta }
      end;
      if talal then bin_keres:=k
               else bin_keres:=0;  { nem talalt }
end; { bin_keres }

begin
 (* egesz tipusu adatok olvasasa  *)
   olvas_int(y,n);
   writeln('Szekvencialis kereses ');
   for j:=1 to 4 do
   begin
     write('adat : '); readln(adat);
     writeln(adat,' helye ',seq_keres(y,n,adat));
   end;
   Writeln('Binaris kereses ');
   for j:=1 to 4 do
   begin
     write('adat : '); readln(adat);
     writeln(adat,' helye ',seq_keres(y,n,adat));
   end;
end.
```

14. Statisztika program: STAT.pas

```
program statisztika;
{ stat.pas }
uses crt,dos,graph,util;
const  db=100;
type
    adat  = real;
    rtype = array[1..db] of adat;
    rbuff = adat;
    str80 = string[80];
    str12 = string[12];
var
    y          : rtype;
    a,m,md,std : real;
    num        : integer;
    ch         : char;
    adatfile   : file of adat;

    Col        : MenuColors;
    Point      : ItemType;
    Choose     : byte;
    Key        : char;
    f_nev      : str12;

procedure hatter;
begin
     textbackground(blue); clrscr;
     textcolor(yellow);
end;

procedure tovabb;
var c:char;
begin
   writeln;
   writeln('        Nyomj Enter-t!');
   readln;
end;

function vizsg(n:integer):boolean;
begin
    if n=0 then begin
                    writeln;
                    writeln('Nincs adat a memoriaban!');
                    writeln;
                    tovabb;
                    vizsg:=false;
                end
               else
               vizsg:=true;
end;
```

```
function FileExists:boolean;
var
   s:searchrec;
begin
 FindFirst(f_nev,anyfile,s);
 FileExists:=doserror=0;
end;

procedure save(yd: rtype; n:integer);
label Ujra;
var
    c    : char;
    i    : integer;
    t    : real;
begin
  hatter;
  if vizsg(n) then begin
   writeln;
   writeln('ADAT KIMENTESE DISKFILE-RA');
   writeln;
Ujra:   write('Az adat file neve : '); readln(f_nev);
   writeln;
   if FileExists then
                  begin
                    writeln('         ',f_nev,' file letezik!');
                      write('         Felulirhato (i/n): ');
                    readln(c);
                    c:=upcase(c);
                    writeln;
                    if (c <> 'I') then goto Ujra;
                    writeln;
                  end;

   assign(adatfile,f_nev);
   rewrite(adatfile);
   t:=n;
   write(adatfile,t);
   for i:= 1 to n do write(adatfile,yd[i]);
   close(adatfile);
   writeln;
   writeln('Az adat kimentve');
   tovabb;
  end;
end; { Save }

procedure Load(var yd: rtype;  var n:integer);
var
    i     : integer;
    t     : real;
begin
   hatter;
   writeln;
```

```
    writeln('ADAT BEOLVASASA DISKFILE-ROL');
    writeln;
    write('Az adat file neve : '); read(f_nev);
    writeln;
    if FileExists then
    begin
      assign(adatfile,f_nev);
      reset(adatfile);
      Read(adatfile,t);
      n:=trunc(t);
      for i:=1 to n do
        read(adatfile,yd[i]);
      close(adatfile);

    writeln('Az adat betoltodott!');
    readln;
    end
    else begin
          writeln('A  ',f_nev,' file nem letezik!');
          readln;
         end;
    tovabb;
end;

function Mean(yd: rtype; n: integer):real;
var  i: integer;
     s: real;
begin
   s:=0;
   for i:=1 to n do
     s:=s+yd[i];
   Mean:=s/n;
end; { Mean }

procedure quick_int(var yd:rtype; m:integer);
   procedure qs(l,r: integer; var ydd: rtype);
   var i,j       :integer;
       buf1,buf2 :rbuff;
   begin
      i:=l; j:=r;
      buf1:=ydd[(l+r) div 2];
      repeat
        while ydd[i] < buf1 do i:=i+1;
        while buf1 < ydd[j] do j:=j-1;
        if i<=j then
        begin
            buf2:=ydd[i];
            ydd[i]:= ydd[j];
            ydd[j]:=buf2;
            i:=i+1; j:=j-1;
        end;
      until i>j;
```

```
      if l<j then qs(l,j,ydd);
      if l<r then qs(i,r,ydd)
   end;
begin
      qs(1,m,yd);
end; {quick_int}

function Median(yd: rtype; n:integer):real;
var
   rend: rtype;
      i: integer;
begin
   for i:=1 to n do
      rend[i]:= yd[i];
   quick_int(rend,n);
   Median:=rend[n div 2];
end;

function Mode_last(yd: rtype; n:integer): real;
var
  i,j,c,c_r : integer;
  ml, ml_r  : real;
begin
  ml_r :=0; c_r:=0;
  for i:=1 to n do
  begin
    ml:=yd[i]; c:=1;
    for j:=i+1 to n do
      if ml=yd[j] then c:=c+1;
    if c>c_r then
              begin
                 ml_r:=ml;
                 c_r :=c;
              end;
  end;
  Mode_last:=ml_r;
end; { Mode_last}

function StdDev(yd: rtype; n:integer):real;
var
   i  : integer;
   s,g: real;
begin
   g:=Mean(yd,n);
   s:=0;
   for i:=1 to n do
     s:=s+((yd[i]-g)*(yd[i]-g));
  s:=s/n;
  StdDev:=sqrt(s);
end; { StdDev }
```

```
function GetMax(yd: rtype; n:integer):integer;
var
    i  : integer;
    max: real;
begin
    max:=yd[1];
    for i:=2 to n do
       if yd[i]>max then max:=yd[i];
    GetMax:=trunc(max);
end; { GetMax }

function GetMin(yd: rtype; n:integer): integer;
var
    i   : integer;
    min : real;
begin
    min:=yd[1];
    for i:=1 to n do
       if yd[i]<min then min:=yd[i];
    GetMin:=trunc(min);
end; { GetMin }

function Ok(ch:char; s:str80):boolean;
var i:integer;
begin
  Ok:=false;
  for i:=1 to length(s) do
     if s[i]=ch then Ok:=true;
end; { Ok }

procedure s_plot(yd: rtype; n,ymin,ymax,xmax:integer;color:integer);
var
  xk,yk,dx,i : integer;
  a,norm,s   : real;
  ch         : char;
  s1         : string;
begin
   if ymin>0 then ymin:=0;
     s:=ymax-ymin;
     norm:=190/s;
     str(ymin:2,s1);
     outtextxy(1,180,s1);
     str(ymax:4,s1);
     outtextxy(1,8,s1);
     str(xmax:4,s1);
     outtextxy(288,192,s1);
     setcolor(1);
     for i:=1 to 19 do
       putpixel(0,i*10,3);
     line(0,190,320,190);
     setcolor(2);
     for i:=1 to n do
```

```
      begin
        a:=yd[i]-ymin;
        a:=a*norm;
        yk:=trunc(a);
        dx:=300 div xmax;
        xk:=((i-1)*dx)+20;
        putpixel(xk,190-yk,color);
      end;
end; { s_plot }

procedure Regress(yd:rtype; n:integer);
var
    a,b         : real;
    x_atl,y_atl: real;
    t1,t2,cor   : real;
    yt          : rtype;
    i,min,max   : integer;
    ch          : char;
    gm,gd       : integer;
begin
   y_atl:=0 ; x_atl:=0;
   for i:=1 to n do
   begin
     y_atl:=y_atl+yd[i];
     x_atl:=x_atl+i;
   end;
   x_atl:=x_atl/n;
   y_atl:=y_atl/n;
   t1:=0; t2:=0;
   for i:=1 to n do
   begin
      t1:=t1+(yd[i]-y_atl)*(i-x_atl);
      t2:=t2+(i-x_atl)*(i-x_atl);
   end;
   b:=t1/t2;
   a:=y_atl-(b*x_atl);
   { kiszamitja a correlacio egyutthatojat }
   for i:=1 to n do
     yt[i]:=i;
     cor:=t1/n;
     cor:=cor/(StdDev(yd,n)*StdDev(yt,n));
     writeln;
     writeln('KETPARAMETERES REGRESSZIO');
     writeln;
     writeln('Regresszio egyenlete   : y = (',a:10,
     ') + (',b:10,') * x');
     writeln('Korrelacios egyutthato : ',cor:10);
     writeln;
     write('Adatok es regresszios egyenes rajzolasa (i/n) ');
     readln(ch);
     writeln;
     if (ch='i') or (ch='I') then begin
```

```
      gd:=CGA;
      gm:=CGAC1;
      initgraph(gd,gm,'');
      setcolor(0);
      for i:=1 to n*2 do
         yt[i]:=a+(b*i);
      min:=GetMin(yd,n)*2;
      max:=GetMax(yd,n)*2;
      s_plot(yd,n,min,max,n*2,1);
      s_plot(yt,n*2,min,max,n*2,2);
      readln;
      closegraph;
    end;
end;

procedure Display(yd: rtype; n:integer);
var
   i: integer;
begin
 if vizsg(n) then begin
   writeln;
   writeln('MERESI ADATOK ');
   writeln;
   for i:=1 to n do
   begin
       writeln(i:3,' : ',yd[i]:15:5);
       if (i mod 18) = 0 then begin
                                tovabb;
                                clrscr;
                                writeln;
                                writeln('MERESI ADATOK ');
                                writeln;
                               end;
   end;
   writeln('Az adat vege !');
   tovabb;
  end;
end; { Display }

procedure Enter(var yd:rtype; var n: integer);
var
   i : integer;
begin
  repeat
    write('Az adatok szama : '); readln(n);
    if n>db then
              writeln('Az adatok max. szama : ',db);
  until n<=100;
   for i:=1 to n do
   begin
     write(i:3,'. adat : '); readln(yd[i]);
   end;
```

```
  tovabb;
end; { Enter }

procedure  Bar_plot(yd: rtype; n:integer);
var
  xk,yk,max,min,dx,i : integer;
  a,norm,s :real;
  ch: char;
  gd,gm:integer;
  s1   :string;
begin
   gd:=CGA;
   gm:=CGAC1;
   initgraph(gd,gm,'');

   max:=GetMax(yd,n);
   min:=GetMin(yd,n);
   if min>0 then min:=0;
     s:=max-min;
     norm:=190/s;
     str(min:2,s1);
     outtextxy(1,180,s1);
     str(max:4,s1);
     outtextxy(1,8,s1);
     str(n:4,s1);
     outtextxy(288,192,s1);
     setcolor(1);
     for i:=1 to 19 do putpixel(0,i*10,1);
     line(0,190,320,190);
     setcolor(2);
     for i:=1 to n do
     begin
       a:=yd[i]-min;
       a:=a*norm;
       yk:=trunc(a);
       dx:=300 div n;
       xk:=((i-1)*dx)+20;
       line(xk,190,xk,190-yk);
    end;
    readln;
    closegraph;
end; { Bar_plot }

procedure max_kereses(t: rtype; k: integer;var tmax: real;var tmin:real);
var i: integer;
begin
    tmax:=t[1];
    tmin:=t[1];
    for i:=2 to k do
    begin
      if t[i]>tmax then tmax:=t[i];
      if t[i]<tmin then tmin:=t[i];
```

```
    end;
end;

procedure oszlop(t:rtype; k:integer; sz: string);
var i          : integer;
    tmax,tmin  : real;
    gd,gm      : integer;
    color      : array[1..3] of integer;
    xm,ym      : integer;
    m          : integer;
    snorm      : real;
    tsz        : integer;
    xx1,yy1    : integer;
    xx2,yy2    : integer;
    yact       : integer;
    szam,szam1 : string[6];
    dx         : integer;
    x1,x2,y1,y2: integer;
begin
   max_kereses(t,k,tmax,tmin);
   DetectGraph(gd,gm);
   case gd of
   1: begin
       gm:=1;
       xm:=319;
       ym:=199;
       color[1]:=1;
       color[2]:=2;
       color[3]:=3;
      end;
   3: begin
       gm:=0;
       xm:=639;
       ym:=199;
       color[1]:=cyan;
       color[2]:=lightred;
       color[3]:=blue;
      end;
   9: begin
       gm:=0;
       xm:=639;
       ym:=199;
       color[1]:=cyan;
       color[2]:=magenta;
       color[3]:=red;
      end;
   end;
   initgraph(gd,gm,'');
   setcolor(color[1]);
   if gd=1 then setbkcolor(black) else
           setbkcolor(white);
   settextjustify(Centertext,Centertext);
```

```
setcolor(1);
outtextxy(xm div 2,10,sz);
x1:=40;
y1:=30;
x2:=xm-20;
y2:=ym-15;
if gd=1 then begin
x1:=10; x2:=xm-10; end;
setcolor(color[3]);
settextjustify(LeftText,BottomText);
   str(tmax:6:0,szam);
   str(tmin:6:0,szam1);
outtextxy(x1,28,'minimum: '+szam1+'     maximum: '+szam);

m:=y2-y1-2;
snorm:=m/tmax;
yact:=y2-y1;
if gd=1 then tsz:=(x2-x1-1) div k else tsz :=(x2-x1-1) div k;
setcolor(2);
if gd=1 then begin xx2:=x1+2; tsz:=tsz-3; dx:=3; end else
             begin xx2:=x1+2; tsz:=tsz-4; dx:=4; end;
    x2:=x1+ k*(tsz+dx)+dx+2;
setcolor(color[1]);
rectangle(x1,y1,x2,y2+2);

setfillStyle(1,color[2]);
for i:=1 to k do
begin
   xx1:=xx2+dx;
   yy1:=y1+yact-round(y[i]*snorm);
   xx2:=xx1+tsz;
   yy2:=y1+round(y[i]*snorm);
   setcolor(1);

   bar(xx1,yy1,xx2,y2);
   rectangle(xx1,yy1,xx2,y2);
   str(i:2,szam);
   setcolor(color[1]);
 if gd=1 then begin
               if k<25 then begin
                SetTextStyle(SmallFont,HorizDir,1);
                SetUserCharSize(13,12,14,17);
                outtextxy(xx1+1,y2+10,szam);
                end
              end
              else
if k<31 then  outtextxy(xx1+5,y2+12,szam);
end;
if ((gd=1) and (k>24)) or ((gd<>1) and (k>30)) then
                   begin
                      outtextxy(x1+3,y2+13,'1');
                      str(k:3,szam);
```

```
                        outtextxy(xx2-22,y2+13,szam);
                      end;

   readln;
   closegraph;
end;


begin

  textcolor(lightgray);
  textbackground(blue);
  clrscr;

  with Col do begin
    FgNormal := black;      BgNormal := lightgray;
    FgBorder := lightgray;  BgBorder := red;
    FgSelect := yellow;     BgSelect := green;
  end;

  Point[1] := 'Adatok klaviaturarol';
  Point[2] := 'Adatok visszairatasa';
  Point[3] := 'Alap Statisztika';
  Point[4] := 'Regresszio es rajz';
  Point[5] := 'Vonalas diagram';
  Point[6] := 'Oszlopdiagram';
  Point[7] := 'Mentes Diskfile-ra';
  Point[8] := 'Betoltes Diskfile-rol';
  Point[9] := 'Kilepes';

  Choose := 1;
  num:=0;
  repeat
    MenuWin(30, 10, ThinLine, Col, Point, 9, 'Statisztika', Choose);
    textcolor(lightgray);
    textbackground(black);
    clrscr;

    gotoxy(20, 5);
    case Choose of
      0 : exit;
      1 :   begin hatter;
                Enter(y,num);
            end;
      2 :   begin hatter;
                Display(y,num);
                end;
      3 :  begin
           hatter;
           if vizsg(num) then begin
              a:=Mean(y,num);
              m:=Median(y,num);
```

```
            std:=StdDev(y,num);
            md:=Mode_last(y,num);
            writeln;
            writeln('STATISZTIKAI ADATOK');
            writeln;
            writeln('Atlagertek   : ',a:15:5);
            writeln('Kozepertek   : ',m:15:5);
            writeln('Szoras       : ',std:15:5);
            writeln('Gyakorisag   : ',md:15:5);
            writeln;
            tovabb;
           end;
         end;
     4:  begin
           hatter;
           if vizsg(num) then begin
                        Regress(y,num);
                        end;
         end;

     5:  begin
           hatter;
           if vizsg(num) then begin
                        Bar_plot(y,num);
                        end;
         end;
     6:  oszlop(y,num,'S T A T I S Z T I K A');
     7:  Save(y,num);
     8:  Load(y,num);
     9:  exit;
     end;
   clrscr;
 until false;

end.
```

15. Oszlopdiagram rajzolása: OSZLOP.pas

```
program oszlop_diagram;
{ oszlop.pas }
uses crt,graph;
const  l=50;
type
     tomb = array[1..l] of real;
var
     y          : tomb;
     n          : integer;
     i          : integer;
     fsz        : string;
```

```
procedure oszlop(t:tomb; k:integer; sz: string);
var i          :integer;
    tmax,tmin  : real;
    gd,gm      : integer;
    color      : array[1..3] of integer;
    xm,ym      : integer;
    m          : integer;
    snorm      : real;
    tsz        : integer;
    xx1,yy1    : integer;
    xx2,yy2    : integer;
    yact       : integer;
    szam,szam1 : string[6];
    dx         : integer;
    x1,y1,x2,y2: integer;

procedure max_kereses(t: tomb; k: integer;var tmax: real;var tmin:real);
var i: integer;
begin
    tmax:=t[1];
    tmin:=t[1];
    for i:=2 to k do
    begin
      if t[i]>tmax then tmax:=t[i];
      if t[i]<tmin then tmin:=t[i];
    end;
end;

begin
   max_kereses(t,k,tmax,tmin);
   DetectGraph(gd,gm);
   case gd of
   1: begin
       gm:=1;
       xm:=319;
       ym:=199;
       color[1]:=1;
       color[2]:=2;
       color[3]:=3;
      end;
   3: begin
       gm:=0;
       xm:=639;
       ym:=199;
       color[1]:=cyan;
       color[2]:=lightred;
       color[3]:=blue;
      end;
   9: begin
       gm:=0;
       xm:=639;
       ym:=199;
```

```
    color[1]:=cyan;
    color[2]:=lightred;
    color[3]:=blue;
   end;
end;
initgraph(gd,gm,'');
setcolor(color[1]);
if gd=1 then setbkcolor(black) else
        setbkcolor(white);
settextjustify(Centertext,Centertext);
setcolor(1);
outtextxy(xm div 2,10,sz);
x1:=40;
y1:=30;
x2:=xm-20;
y2:=ym-15;
if gd=1 then begin
x1:=10; x2:=xm-10; end;
setcolor(color[3]);
settextjustify(LeftText,BottomText);
   str(tmax:6:0,szam);
   str(tmin:6:0,szam1);
outtextxy(x1,28,'minimum: '+szam1+'     maximum: '+szam);

m:=y2-y1-2;
snorm:=m/tmax;
yact:=y2-y1;
if gd=1 then tsz:=(x2-x1-1) div k else tsz :=(x2-x1-1) div k;
setcolor(2);
if gd=1 then begin xx2:=x1+2; tsz:=tsz-3; dx:=3; end else
             begin xx2:=x1+2; tsz:=tsz-4; dx:=4; end;
    x2:=x1+ k*(tsz+dx)+dx+2;
setcolor(color[1]);
rectangle(x1,y1,x2,y2+2);

setfillStyle(1,color[2]);
for i:=1 to k do
begin
   xx1:=xx2+dx;
   yy1:=y1+yact-round(y[i]*snorm);
   xx2:=xx1+tsz;
   yy2:=y1+round(y[i]*snorm);
   setcolor(1);

   bar(xx1,yy1,xx2,y2);
   rectangle(xx1,yy1,xx2,y2);
   str(i:2,szam);
   setcolor(color[1]);
 if gd=1 then begin
              if k<25 then begin
               SetTextStyle(SmallFont,HorizDir,1);
               SetUserCharSize(13,12,14,17);
```

```
                    outtextxy(xx1+1,y2+10,szam);
                    end
                  end
                  else
   if k<31 then   outtextxy(xx1+5,y2+12,szam);
   end;
   if ((gd=1) and (k>24)) or ((gd<>1) and (k>30)) then
                       begin
                          outtextxy(x1+3,y2+13,'1');
                          str(k:3,szam);
                          outtextxy(xx2-22,y2+13,szam);
                       end;

   readln;
   closegraph;
end;

begin
   writeln('Adatok beolvasasa ');
  repeat
   write('Az adatok szama : '); readln(n);
   writeln;
  until n<=50;
   for i:=1 to n do
   begin
     write(i:2,'. adat : ');
     readln(y[i]);
   end;
   write('A diagram fejlece : '); readln(fsz);
   oszlop(y,n,fsz);
end.
```

16. Éves statisztika ábrázolása diagramokkal: EV_DIAG.pas

```
program eves_diagram;
{ ev_diag.pas }
uses crt,graph;
type
     tomb12 = array[1..12] of real;
var
     ye          : tomb12;
     n           : integer;
     i           : integer;
     fsz         : string;
     evszam      : integer;
     tip         : integer;

procedure ev_diagram(t: tomb12; k: integer; sz: string;
                     ev: integer; tipus: integer);
type  s5= string[5];
      s4= string[4];
```

```
var i           :integer;
    tmax,tmin   : real;
    gd,gm       : integer;
    color       : array[1..3] of integer;
    xm,ym       : integer;
    m           : integer;
    snorm       : real;
    tsz         : integer;
    xx1,yy1     : integer;
    xx2,yy2     : integer;
    yact        : integer;
    szam,szam1  : string[6];
    x1,x2,y1,y2: integer;
    ho          : array[1..12] of s5;
    ho1         : array[1..12] of s4;
    xk          : integer;

procedure max_kereses(t: tomb12; k: integer;
                      var tmax: real;var tmin:real);
var i: integer;
begin
    tmax:=t[1];
    tmin:=t[1];
    for i:=2 to k do
    begin
      if t[i]>tmax then tmax:=t[i];
      if t[i]<tmin then tmin:=t[i];
    end;
end;

begin
   max_kereses(t,k,tmax,tmin);
   DetectGraph(gd,gm);
   case gd of
   1: begin
       gm:=1;
       xm:=319;
       ym:=199;
       color[1]:=1;
       color[2]:=2;
       color[3]:=3;
      end;
 3,4: begin
       gm:=0;
       xm:=639;
       ym:=199;
       color[1]:=cyan;
       color[2]:=lightred;
       color[3]:=lightblue;
      end;
   9: begin
       gm:=0;
```

```
    xm:=639;
    ym:=199;
    color[1]:=cyan;
    color[2]:=lightred;
    color[3]:=lightblue;
   end;
else begin
       writeln('A display ismeretlen ');
       writeln;
       writeln('   Nyomj ENTER-t !');
       readln;
       exit;
     end;
end;
ho[1]:='JAN'; ho[2]:='FEBR'; ho[3]:='MARC';
ho[4]:='APR'; ho[5]:='MAJ';  ho[6]:='JUN';
ho[7]:='JUL';  ho[8]:='AUG';  ho[9]:='SZEPT';
ho[10]:='OKT'; ho[11]:='NOV'; ho[12]:='DEC';
ho1[1]:='JAN'; ho1[2]:='FEBR'; ho1[3]:='MARC';
ho1[4]:='APR'; ho1[5]:='MAJ';  ho1[6]:='JUN';
ho1[7]:='JUL';  ho1[8]:='AUG';  ho1[9]:='SZEP';
ho1[10]:='OKT'; ho1[11]:='NOV'; ho1[12]:='DEC';

initgraph(gd,gm,'');
if gd = 1 then setbkcolor(black) else   setbkcolor(white);
settextjustify(Centertext,Centertext);
setcolor(color[3]);
outtextxy(xm div 2,10,sz);
   str(ev:4,szam);
outtextxy(xm div 2, 192, szam);

x1:=20;
y1:=30;
x2:=xm-20;
y2:=ym-30;
if gd=1 then begin x1:=5; x2:=xm-5; end;

settextjustify(LeftText,BottomText);
   str(tmax:6:1,szam);
   str(tmin:6:1,szam1);
outtextxy(x1,28,'minimum: '+szam1+'     maximum: '+szam);
rectangle(x1,y1,x2,y2+2);

m:=y2-y1-6;
snorm:=m/tmax;
yact:=y2-y1;
if gd=1 then tsz:=(x2-x1-9) div k
        else
             tsz :=(x2-x1-10) div k;
tsz :=tsz-9;
setcolor(2);
xx2:=x1-8;
```

```
    if gd=1 then xx2:=-4;
    setfillStyle(9,color[2]);
    if gd=1 then  begin
                    SetTextStyle(SmallFont,HorizDir,1);
                    SetUserCharSize(13,12,14,17);
                  end;
    for i:=1 to k do
    begin
       xx1:=xx2+10;
       yy1:=y1+yact-round(ye[i]*snorm);
       xx2:=xx1+tsz;
       yy2:=y1+round(ye[i]*snorm);
       setcolor(1);
   if tipus = 1 then
                      bar3d(xx1,yy1,xx2,y2,6,true)
                 else
                      begin
                        bar(xx1,yy1,xx2,y2);
                        rectangle(xx1,yy1,xx2,y2);
                      end;
       if gd=1 then xk:=xx1 else xk:=xx1+8;
       if gd=1 then outtextxy(xk,y2+12,ho1[i])
               else
                      outtextxy(xk,y2+12,ho[i]);
    end;
    readln;
    closegraph;
end;

begin
    writeln('Adatok beolvasasa havi bontasban ');
    writeln('Az adatok szama : 12');
    for i:=1 to 12 do
    begin
      write(i:2,'. adat : '); readln(ye[i]);
    end;
    write('A diagram fejlece : '); readln(fsz);
    write('Evszam            : '); readln(evszam);
   repeat
    write('Abrazolas: oszlop(1) teglalap (2) : ');
    readln(tip);
    writeln;
   until (tip=1) or (tip=2);

    ev_diagram(ye,12,fsz,evszam,tip);

end.
```

17. Komplex mátrix műveletek: KOMPLEX.pas

```
program komplex(out);
(* A program komplex elemu matrixokkal vegez muveleteket *)
(* muveletek:
        teljes foelemkivalasztas Gauss eliminacioval,
        matrix determinansa
        LU dekompozicio
        inverzmatrix
        matrix sajaterteke SPUR modszerrel
*)
uses Crt,Util;
const l=11;

type  compl      =record re,im:real end;
      matrix     =array[1..l,1..l] of compl;
      vekt       =array[1..l] of integer;
      cvekt      =array[0..l] of compl;

var
      a,inv,sz,bel   :matrix;
      ip             :vekt;
      t,det          :compl;
      n              :integer;
      b,b1,sp        :cvekt;
      v_flag,m_flag  :boolean;

      Col    : MenuColors;
      Point  : ItemType;
      Choose : byte;
      Key    : char;
      mfile  : string[12];
      out    : text;

procedure tovabb;
begin
      gotoxy(26,24);
      textcolor(LIGHTRED+BLINK);
      writeln('==> [ENTER] ');
         repeat
          until keypressed;
     textcolor(yellow);
end;

function c_abs(x:compl):real;
begin
     c_abs:=sqrt(sqr(x.re)+sqr(x.im));
end;
```

```
procedure c_szoroz(x,y:compl;var z:compl);
var w:compl;
begin
     w.re:=x.re*y.re-x.im*y.im;
     w.im:=x.re*y.im+x.im*y.re;
     z   :=w;
end;

procedure c_oszt(x,y:compl;var z:compl);
var w:compl;
begin
     w.re:=(x.re*y.re+x.im*y.im)/sqr(c_abs(y));
     w.im:=(x.im*y.re-x.re*y.im)/sqr(c_abs(y));
     z   :=w;
end;

procedure c_kivon(x,y:compl;var z:compl);
var w:compl;
begin
     w.re:=x.re-y.re;
     w.im:=x.im-y.im;
     z   :=w;
end;

procedure c_osszead(x,y:compl;var z:compl);
var w:compl;
begin
     w.re:=x.re+y.re;
     w.im:=x.im+y.im;
     z   :=w;
end;

procedure matrix_olvasasa;
var i,j:integer;
begin

     write('A matrix merete: '); readln(n);
if n<=10 then
     begin
     writeln(out,'A matrix merete :',n);
     writeln(out,'A matrix elemei '); writeln(out);
     writeln('A matrix elemei '); writeln;
     for i:=1 to n do
      begin
        writeln;
        writeln(i:2,'.sor: ');
        writeln(out);
        writeln(out,i:2,'.sor: ');
       for j:=1 to n do
        begin
         write('  ',j:2,'.elem: ');
         write('valos: '); readln(a[i,j].re);
```

```
          write('        kepzetes: '); readln(a[i,j].im);
          writeln(out,'  ',j:2,'.elem:  valos: ',a[i,j].re:10);
          writeln(out,'         kepzetes: ',a[i,j].im:10);
        end;
      end;
      m_flag:=true;
      writeln(out);
  end
  else
  writeln('A matrix elemeinek szama max. 10 lehet! ');
end;

procedure vektor_olvasasa;
var i:integer;
begin
if m_flag then
begin
     writeln('Az egyenletrendszer jobb oldala: ');
     writeln;
     writeln(out,'Az egyenletrendszer jobb oldala: ');
     writeln(out);
     for i:=1 to n do
     begin
      write(i:2,'.sor: ');
      write('valos: '); readln(b[i].re);
      write('     kepzetes: '); readln(b[i].im);
      writeln(out,i:2,'.sor: valos: ',b[i].re:10);
      writeln(out,'     kepzetes: ',b[i].im:10);
     end;
     v_flag:=true;
     writeln(out);
     b1:=b;
end
else
  writeln('Matrix beolvasasa hianyzik!');

end;

procedure kiirm(c:matrix);
var i,j:integer;
    p  :char;
begin
if m_flag then
    begin
      writeln('MATRIX ELEMEI (',n:2,'x',n:2,')');
      writeln;
      writeln(out,'MATRIX ELEMEI (',n:2,'x',n:2,')');
      writeln(out);
      for i:=1 to n do
      begin
       for j:=1 to n do
       begin
```

```
        if c[i,j].im<0 then begin p:='-';c[i,j].im:=abs(c[i,j].im);
        end else p:='+';
       if n<=3 then
        begin
           write('  ',c[i,j].re:10,p,'i',c[i,j].im:10);
           write(out,'  ',c[i,j].re:10,p,'i',c[i,j].im:10);
        end
        else
        begin
           writeln('[',i:2,',',j:2,']: ',
           c[i,j].re:10,p,'i',c[i,j].im:10);
           writeln(out,'[',i:2,',',j:2,']: ',
           c[i,j].re:10,p,'i',c[i,j].im:10);
        end;
       end;
       if n<=3 then  begin
                          writeln;
                          writeln(out);
                     end
                     else begin
                           writeln(out);
                           writeln('==>ENTER');
                           readln;
                          end;
     end;
   end

   else
   writeln('Hianyzik a matrix adatainak beolvasasa ! ');
end;

procedure gauss_ir(c:matrix);
var i,j:integer;
   p  :char;
begin
      writeln;
      writeln('Az egyenletrendszer gyokei ');writeln;
      writeln(out);
      writeln(out,'Az egyenletrendszer gyokei ');writeln(out);
      j:=n+1;
      for i:=1 to n do
      begin
        if c[i,j].im<0 then begin p:='-';c[i,j].im:=abs(c[i,j].im);
        end else p:='+';
        writeln('  ',c[i,j].re:10,p,'i',c[i,j].im:10);
        writeln(out,'  ',c[i,j].re:10,p,'i',c[i,j].im:10);
       end;
       writeln;
       writeln(out);
end;
```

```
procedure kiirv(c:cvekt);
var i  :integer;
    p  :char;
begin
if v_flag then
 begin
      writeln;
      writeln(out);
      for i:=1 to n do
      begin
        if c[i].im<0 then begin p:='-';c[i].im:=abs(c[i].im)
        ;end else p:='+';
        writeln('  ',c[i].re:10,p,'i',c[i].im:10);
        writeln;
        writeln(out,'  ',c[i].re:10,p,'i',c[i].im:10);
        writeln(out);
      end;
  end
  else
  writeln('Hianyzik a jobb oldal beolvasasa!');
end;

procedure spur_ir(c:cvekt);
var i  :integer;
    p  :char;
begin
      writeln; writeln(out);
      for i:=0 to n do
      begin
        p:='+';
        if c[i].im<0 then begin p:='-';c[i].im:=abs(c[i].im) end;
        writeln('  ',c[i].re:10,p,'i',c[i].im:10);
        writeln;
        writeln(out,'  ',c[i].re:10,p,'i',c[i].im:10);
        writeln(out);
      end;
end;

procedure kiirc(c:compl);
var p  :char;
begin
      writeln; writeln(out);
        if c.im<0 then begin p:='-';c.im:=abs(c.im);end else p:='+';
        writeln('  ',c.re:10,p,'i',c.im:10);
        writeln(out,'  ',c.re:10,p,'i',c.im:10);
end;

procedure gauss(var a:matrix;var b:cvekt; n:integer);
var ip          :vekt;
    t,u,max     :compl;
    h,h1        :compl;
    i,j,k,m,l   :integer;
```

```
    x          :cvekt;
    hiba       :boolean;
begin
     for i:=1 to n do
     begin
      a[i,n+1].re:=b[i].re;
      a[i,n+1].im:=b[i].im;
    end;
     for i:=1 to n do ip[i]:=i;
     hiba:=false;
     for i:=1 to n do
      begin
        max.re:=0;max.im:=0;j:=i;l:=i;
        for m:=i to n do
         for k:=i to n do
          if c_abs(a[m,k])>c_abs(max) then
            begin
                 max:=a[m,k];j:=m;l:=k;
            end;
         if j<>i then
            for k:=i to n+1 do
             begin
                  t:=a[i,k];
                  a[i,k]:=a[j,k];
                  a[j,k]:=t;
             end;
         if l<>i then
            begin
             for m:=1 to n do
              begin
                   t:=a[m,i];
                   a[m,i]:=a[m,l];
                   a[m,l]:=t;
              end;
              m:=ip[i];
              ip[i]:=ip[l];
              ip[l]:=m;
            end;
        t:=a[i,i];
       if c_abs(t)>1.e-7 then
       begin
          h.re:=1.0; h.im:=0;
          c_oszt(h,t,t);

          for k:=i to n+1 do
          begin
             c_szoroz(t,a[i,k],a[i,k]);
          end;
        if i<>n then
        begin
          for j:=i+1 to  n do
          begin
```

```
            t:=a[j,i];
            for k:=i to n+1 do
            begin
              c_szoroz(t,a[i,k],h);
              c_kivon(a[j,k],h,a[j,k]);
            end;
          end;              end;
    end
    else hiba:=true;
    end;
    if not hiba then
    begin
       for i:=n downto 1 do
       begin
          t:=a[i,n+1];
          for k:=i+1 to n do
          begin
             c_szoroz(a[i,k],a[k,n+1],h);
             c_kivon(t,h,t);
          end;
             c_oszt(t,a[i,i],a[i,n+1]);
       end;
       for i:=1 to n do x[ip[i]]:=a[i,n+1];
       for i:=1 to n do a[i,n+1]:=x[i];
     end;
   end;

procedure dekomp(var a:matrix;var ip:vekt;var det:compl;n:integer);
var i,j,k,m:integer;
    t,u,v  :compl;
begin
     det.re:=1;det.im:=0;ip[n]:=1;
     for k:=1 to n do
      begin
        if k<>n then
         begin
            m:=k;
            for i:=k+1 to n do
              if c_abs(a[i,k])>c_abs(a[m,k]) then m:=i;
            ip[k]:=m;t:=a[m,k];
            c_szoroz(det,t,det);
            if m<>k then
             begin
                ip[n]:=-ip[n];
                a[m,k]:=a[k,k];
                a[k,k]:=t;
             end;
            if c_abs(t)>1.e-12 then     { <> 0.0 }
             begin
              for i:=k+1 to n do
               begin
                 c_oszt(a[i,k],t,a[i,k]);
```

```
                end;
               for j:=k+1 to n do
                begin
                 t:=a[m,j];
                 if m<>k then
                  begin
                   a[m,j]:=a[k,j];
                   a[k,j]:=t;
                  end;
                 if c_abs(t)>1.e-12 then    { <> 0.0 }
                  for i:=k+1 to n do
                   begin
                     c_szoroz(a[i,k],t,u);
                     c_kivon(a[i,j],u,a[i,j]);
                   end;
                end;
              end;
             end;
             if c_abs(a[k,k])<1e-12 then ip[n]:=0;
      end;
      c_szoroz(det,a[n,n],u);
      v.re:=ip[n]; v.im:=0;
      c_szoroz(u,v,det);
end;

procedure solve(var a:matrix;var b:cvekt;ip:vekt;n:integer);
var i,j,k,m:integer;
    t,u    :compl;
begin
     for k:=1 to n-1 do
      begin
       m:=ip[k];
       if m<0 then m:=-m;
       t:=b[m];
       if m<>k then
        begin
         b[m]:=b[k];
         b[k]:=t;
        end;
       for i:=k+1 to n do
        begin
         c_szoroz(a[i,k],t,u);
         c_kivon(b[i],u,b[i]);
        end;
      end;
      for k:=n downto 1 do
       begin
        c_oszt(b[k],a[k,k],b[k]);
        for j:=k+1 to n do
         begin
          c_szoroz(a[k,j],b[j],u);
          c_oszt(u,a[k,k],u);
```

```
          c_kivon(b[k],u,b[k]);
         end;
       end;
end;

procedure inverz(var inv:matrix);
var e  :cvekt;
    f,m:integer;
begin
     for f:=1 to n do
     begin
          for m:=1 to n do begin e[m].re:=0;e[m].im:=0;end;
          e[f].re:=1;e[f].im:=0;
          solve(bel,e,ip,n);
          for m:=1 to n do
          inv[m,f]:=e[m];
     end;
end;

procedure szor(var a,inv,sz:matrix;n:integer);
var m,i,l,gomb:integer;
    u ,h      :compl;
begin
     for i:=1 to n do
     for l:=1 to n do
     begin
      sz[i,l].re:=0;sz[i,l].im:=0;
     for m:=1 to n do
     begin
           c_szoroz(a[i,m],inv[m,l],u);
           c_osszead(sz[i,l],u,sz[i,l]);
     end;
     end;
end;

procedure spur(var a:matrix;var x:cvekt);
var i,j,k  :integer;
    b,c    :matrix;
    t,u,h  :compl;
    d      :cvekt;
begin
     for i:=1 to n do
      for j:=1 to n do
        begin
         c[i,j]:=a[i,j];
        end;
     for i:=1 to n do
     begin
          d[i].re:=0;
          d[i].im:=0;
          for j:=1 to n do
          begin
```

```
            c_osszead(d[i],c[j,j],d[i]);
          end;
          for j:=1 to n do
           for k:=1 to n do
            begin
              b[j,k]:=c[j,k];
            end;
         szor(a,b,c,n);
     end;
     x[0].im:=0; x[0].re:=1;
     for i:=1 to n do
     begin
       x[i].im:=0; x[i].re:=0;
       for j:=1 to i-1 do
       begin
         c_szoroz(x[i-j],d[j],u);
         c_osszead(x[i],u,x[i]);
       end;
       c_osszead(x[i],d[i],u);
       t.re:=i;
       t.im:=0;
       c_oszt(u,t,h);
       x[i].re:=-h.re;
       x[i].im:=-h.im;
    end;
 end;

procedure hatter;
begin
  window(1,1,80,25);
  textbackground(blue);
  textcolor(yellow);
  clrscr;
end;

procedure hiba1;
begin
   if not m_flag then  begin
                       gotoxy(20,10);
                       write('Matrix beolvasasa hianyzik! ');
                       end;
   tovabb;
end;

procedure hiba2;
begin
   if not m_flag then begin
                        gotoxy(20,10);
                        write('Matrix beolvasasa hianyzik! ');
                        end;
   if not v_flag then  begin
                       gotoxy(20,12);
```

```
                        write('A jobb oldal beolvasasa hianyzik! ');
                        end;
   tovabb;
end;

procedure kilep;
begin
  textcolor(lightgray);
  textbackground(black);
  clrscr;
 if m_flag then  close(out);

end;

begin
  textcolor(lightgray);
  textbackground(blue);
  clrscr;

  with Col do begin
    FgNormal := black;      BgNormal := cyan;
    FgBorder := white;      BgBorder := cyan;
    FgSelect := yellow;     BgSelect := magenta;
  end;

  Point[1] := 'Matrix olvasasa';
  Point[2] := 'Vektor olvasasa';
  Point[3] := 'Matrix kiiratasa';
  Point[4] := 'Vektor kiiratasa';
  Point[5] := 'Gauss eliminacio';
  Point[6] := 'Matrix determinansa';
  Point[7] := 'LU dekompozicio ';
  Point[8] := 'Matrix invertalasa';
  Point[9] := 'Sajat-ertek (Spur)';
  Point[10]:= 'Kilepes ';

  Choose := 1;
    m_flag:=false; v_flag:=false;
  repeat
    MenuWin(30, 10, DoubleLine, Col, Point, 10, 'Komplex matrix', Choose);

    hatter;
    case Choose of
      0 : kilep;
      1 : { matrix beolvasasa }
          begin
          if m_flag then close(out);
             write('Az eredmeny file neve : ' );readln(mfile);
             assign(out,mfile);
             rewrite(out);
             writeln(out,'Az eredmeny file neve : ',mfile);
             writeln(out);
```

```
     matrix_olvasasa;
     tovabb;
    end;

2 : { A jobb oldal beolvasasa }
    begin
     vektor_olvasasa;
     tovabb;
    end;
3 : { matrix kiiratasa }
    begin
     kiirm(a);
     tovabb;
    end;
4 : { A jobb oldal kiiratasa }
    begin
     writeln('A VEKTOR ELEMEI ');
     writeln(out);
     writeln(out,'A VEKTOR ELEMEI ');
     kiirv(b);
     tovabb;
    end;
5 : { Gauss eliminacio }
    begin
     if(v_flag and m_flag)
     then
      begin
      bel:= a;
      writeln('GAUSS ELIMINACIO');
      writeln(out,'GAUSS ELIMINACIO');
      gauss(bel,b,n);
      kiirm(bel);
      gauss_ir(bel);
      tovabb;
      end
     else
       begin
          hiba2;
       end;
    end;
6 : { LU dekompozicioval a matrix determinansa }
    begin
     if m_flag then
      begin
      bel:=a;
      dekomp(bel,ip,det,n);
      writeln('AZ LU DEKOMPONALT MATRIX');
      writeln(out,'AZ LU DEKOMPONALT MATRIX');
      kiirm(bel);
      writeln(out);
      writeln;
      writeln('A MATRIX DETERMINANSA');
```

```
      writeln(out,'A MATRIX DETERMINANSA');
      kiirc(det);
      writeln(out);
      tovabb;
     end
    else
      hiba1;
    end;
7 : { LU dekompozicio }
    begin
    if (m_flag and v_flag) then
    begin

      bel:=a;
      dekomp(bel,ip,det,n);
      solve(bel,b1,ip,n);
      writeln('AZ EGYENLETRENDSZER MEGOLDASA LU DEKOMPOZICIOVAL');
      writeln(out,'AZ EGYENLETRENDSZER MEGOLDASA LU DEKOMPOZICIOVAL');
      kiirv(b1);
      tovabb;
    end
   else
   begin
     hiba2;
   end;
   end;

8 :  { matrix invertalasa  }
    begin
    if m_flag then
    begin
      bel:=a;
      dekomp(bel,ip,det,n);
      inverz(inv);
      writeln('A MATRIX INVERZE');
      writeln(out,'A MATRIX INVERZE');
      kiirm(inv);
      tovabb;
    end else hiba1;
    end;

9 : { Matrix sajaterteke Spur modszerrel }
   begin
   if m_flag then
   begin
     bel:=a;
     spur(bel,sp);
     writeln('A MATRIX SAJATERTEKE');
     writeln(out);
     writeln(out,'A MATRIX SAJATERTEKE');
     spur_ir(sp);
     tovabb;
```

```
        end
        else hiba1;
         end;
      10 : begin
             kilep;
             exit;
           end;

    end;
    clrscr;
    Key := readkey;
  until false;
end.
```

18. Menükezelés : UTIL.pas

```
unit Util;

{ Általános segédeljárások }

interface

const
  NoLine     = 0;
  ThinLine   = 1;
  FatLine    = 2;
  DoubleLine = 3;
  DoubleTop  = 4;
  DoubleSide = 5;

  EnterKey = #13;
  ESC      = #27;
  UpKey    = #72;
  DownKey  = #80;
  HomeKey  = #71;
  EndKey   = #79;

type
  MenuColors = record
    FgNormal, BgNormal,
    FgBorder, BgBorder,
    FgSelect, BgSelect : byte;
  end;
  Str80      = string[80];
  ItemType   = array[1..25] of Str80;


procedure CursorOn;                          { A kurzor bekapcsolása }
procedure CursorOff;                         { A kurzor kikapcsolása }

procedure MenuWin(     X, Y,
                                  { Bal felső sarok koordinátái }
                       Border  : byte;       { Keret stílusa }
                       MenuCol : MenuColors; { A menü színei }
                       Items   : ItemType;   { A menü pontok }
                       ItemNum : byte;       { A menüpontok száma }
                       Title   : Str80;      { A menü címe }
                   var Choice  : byte);      { A választott menüpont }


implementation

uses Crt, Dos;
```

```
const
  NoCursor = $2000;

  Borders : array[0..5] of string[10] =
    ('          ',
     '┌┐ ╓─╖ │ │',        { 218 196 191 179 217 196 192 179 180 195 }
     '└┘ ╙─╜ │ │',        { 220 219 220 219 223 219 223 219 221 222 }
     '╔╗ ╓─╖ ╞ ╡',        { 201 205 187 186 188 205 200 186 181 198 }
     '╚╝ ╙─╜ ╞ ╡',        { 213 205 184 179 190 205 212 179 181 198 }
     '┌┐ ╓─╖ ├ ┤');       { 214 196 183 186 189 196 211 186 180 195 }

var
  OldCursor : word;

{-------------------------------------------------------------------------}

procedure SetCursor(NewCursor : word);
  var
    Reg : registers;
  begin
    with Reg do
    begin
      AH := 1;
      BH := 0;
      CX := NewCursor;
      intr($10, Reg);
    end;
  end; { procedure SetCursor }

function GetCursor : word;
  var
    Reg : registers;
  begin
    with Reg do
    begin
      AH := 3;
      BH := 0;
      intr($10, reg);
      GetCursor := CX;
    end;
  end; { function GetCursor }

procedure CursorOn;
  begin
    SetCursor(OldCursor);
  end; { procedure CursorOn }

procedure CursorOff;
  begin
    if GetCursor<>NoCursor then begin
      OldCursor := GetCursor;
      SetCursor(NoCursor);
```

```
    end;
  end; { procedure CursorOff }

{-------------------------------------------------------------------------}

procedure MenuWin;
  var
    Max, i, k : byte;
    Str       : Str80;
    Key       : char;
    Quit      : boolean;

  begin
    CursorOff;

    Max := length(Title);
    for i := 1 to ItemNum do
      if length(Items[i]) > Max then Max := length(Items[i]);
    Max := Max + 2;

    textcolor(MenuCol.FgBorder);
    textbackground(MenuCol.BgBorder);
    gotoxy(X, Y);
    Str := Borders[Border, 1];
    for i := 1 to Max do
      Str := Str + Borders[Border, 2];
    Str := Str + Borders[Border, 3];
    write(Str);

    gotoxy(X+(Max-length(Title)) div 2, Y);
    write(Borders[Border, 9]+Title+Borders[Border, 10]);

    for i := 1 to ItemNum do begin
      gotoxy(X, Y+i);
        write(Borders[Border, 8]);
      gotoxy(X+Max+1, Y+i);
        write(Borders[Border, 4]);
    end;

    gotoxy(X, Y+i+1);
    Str := Borders[Border, 7];
    for i := 1 to Max do
      Str := Str + Borders[Border, 6];
    Str := Str + Borders[Border, 5];
    write(Str);

    if Choice = 0 then Choice := 1;

    repeat
      for i := 1 to ItemNum do begin
          gotoxy(X+1, Y+i);
          if Choice = i then begin
```

```
            textcolor(MenuCol.FgSelect);
            textbackground(MenuCol.BgSelect);
            write(' '+Items[i]);
          end
          else begin
            textcolor(MenuCol.FgNormal);
            textbackground(MenuCol.BgNormal);
            write(' '+Items[i]);
          end;
          for k := 1 to Max-length(Items[i])-1 do
            write(' ');
        end;

        repeat
          Key := readkey;
          Quit := true;
          if Key = #0 then begin
            Key := readkey;
            case Key of
            UpKey   : begin
                        if Choice = 1 then Choice := ItemNum
                        else dec(Choice);
                      end;
            DownKey : begin
                        if Choice = ItemNum then Choice := 1
                        else inc(Choice);
                      end;
            HomeKey : Choice := 1;
            EndKey  : Choice := ItemNum
            else
              Quit := false;
            end;
          end
          else
            Quit := false;
            if Key = ESC then begin
              Choice := 0;
              Quit := true;
            end;
            if Key = EnterKey then Quit := true;
        until Quit;

      until Key in [EnterKey, ESC];
      CursorOn;

    end; { procedure MenuWin }

end.
```

IRODALOMJEGYZÉK

1. Benkő Tiborné - Hegedüs András : IBM PC programozása Turbo Pascal Nyelven
   BME Mérnöktovábbképző Intézet
   Budapest, 1989.

2. Benkő Tiborné - Benkő László : Objektum-orientált programozás Turbo Pascal-ban 5.5 változat
   BME Mérnöktovábbképző Intézet
   Budapest, 1990.

3. N.Wirth : Algoritmusok + adatstruktúrák = programok
   Műszaki Könyvkiadó
   Budapest, 1982.

4. Advanced Turbo Pascal
   Osborne McGraw-Hill, 1986.