DIGITÁLIS KULTÚRA

14.
TURBO PASCAL PROGRAMOZÁS ALAPJAI

oktatási segédanyag és feladatgyűjtemény

Összeállította: Kolman Krisztián

Tartalomjegyzék:

1 A pascal program szerkezete

- a pascal program alapszerkezete
- kiíratás képernyőre - **write, writeln**

1.1 A pascal program alapszerkezete

A Pascal programozási nyelvben minden programnak a következő szerkezete van:

```
program program_neve;
var változók deklarálása;
begin

   parancsok, amit a program végrehajtson;

end.
```

Az első sorban a **program** kulcsszó után megadjuk a programunk nevét. Ide bármilyen nevet választhatunk, de a program neve nem tartalmazhat szóközt és néhány speciális karaktert. A sor (parancs) végét pontosvesszővel (**;**) fejezzük be. Ezt az első sort nem kötelező megadnunk.

A második sorban a **var** kulcsszó *(variables = változók)* után felsoroljuk, hogy a programunkban milyen változókat fogunk használni. Az egyszerű statikus változót úgy képzelhetjük el, mint valamilyen tárolóhelyet a számítógép memóriájában, amelyben egyszerre csak egy számot vagy szöveget tárolhatunk. Ha például van benne már valamilyen szám és rakunk bele egy másikat, akkor az előző szám elveszik, mindig csak az utoljára belerakott érték marad benne.

A **var** parancsnál meg kell adnunk azt is, hogy az egyes változókban (tárolóhelyekben) milyet típusú adatot fogunk tárolni - egész számot (integer), szöveget (string), vagy más típust. A programunkban több változót is használhatunk, pl. kettő változót számok tárolására és egy változót szöveg tárolására. Itt is a sor végét pontosvesszővel fejezzük be. Amennyiben a programban nem használunk változót (bár ez nagyon ritkán fordul elő), ezt a sort is kihagyhatjuk.

A következő sortól kezdődik maga a program – azok az egymás után következő utasítások, melyeket a számítógép végrehajt a program futtatásakor. Ezeket az utasításokat a **begin** és az **end** kulcsszavak *(begin = kezdete, end = vége)* közé kell írnunk. Fontos, hogy minden parancs után a sort pontosvesszővel (**;**) fejezzük be. A programunk végét ponttal (**.**) zárjuk.

Célszerű egy úgynevezett **crt** unit használata. Például képernyőtörlésre. **(clrscr)** Ennek használata ajánlott az összes programban! (Aztán például a **delay**(2000) – a program 2000 milliszekundumig (2 másodpercig) várakozik.)

A programozás során egy fekete, 80 karakter széles és 35 karakter magas felületen dolgozunk!

Lássunk most egy egyszerű példát:

```
program ElsoProgram;
uses crt;
begin
clrscr;
 write('Hello!');
readln;
end.
```

Ez a program, csupán annyit csinál, hogy kiírja a képernyőre a Hello! mondatot. Ha programunkat begépeltük az editorba (FreePascal-ba), mentsük el, majd a **CTRL** + **F9** -el (vagy a főmenüből **run - run** paranccsal) fordítsuk le és futtassuk. Ekkor a program kiírja a *Hello!* mondatot a képernyőre, majd rögtön visszatér az editorba (ezért nem látjuk a kiírt mondatot). A FreePascal egy másik szöveges képernyővel dolgozik a program futtatása alatt, mint amiben mi írjuk magát a programot (ezért most nem látjuk a kiírt *Hello!* üzenetet). Erre a képernyőre az **ALT** + **F5** billentyűzetkombinációval válthatunk át. Vissza az editorba az **ALT** + **F5** újabb megnyomásával juthatunk.

*Megjegyzés: Ha a FreePascal DOS alatt futó verzióját használjuk, akkor a billentyűzetet a **CTRL** + **ALT** + **F1** billentyűzetkombinációval állíthatjuk át angol nyelvűre.*

1.2 Kiírás a képernyőre - write, writeln parancsok

Ha valamit ki szeretnénk írni a képernyőre, azt amint az előző példában is láthattuk, a **write** és **writeln** *(write = ír, write line = sort ír)* parancsokkal tehetjük meg.

A **write** parancs kiírja a megadott szöveget a képernyőre. A kurzor közvetlenül a kiírt szöveg után marad, így a következő kiírásnál rögtön ez után íródik ki a további szöveg. Például:

```
program Pelda01a;
uses crt;
begin
clrscr;
 write('Hat fele:');
 write(3);
readln;
end.
```

Ez a program a következő szöveget fogja kiírni a képernyőre:

```
Hat fele:3_
```

Az első **write** parancs kiírja a *Hat fele:* szöveget. A kurzor közvetlenül a szöveg mögött marad. A második **write** kiírja a *3* számot. A kurzor a 3-as mögé kerül. Megfigyelhettük, hogy ha szöveget akarunk kiíratni, akkor azt mindig aposztrófok (') közé kell raknunk. Az aposztrófok közé írt szöveg egy az egyben kiíródik a képernyőre. Ha számot vagy valamilyen változó értékét írjuk ki, akkor elhagyjuk az aposztrófokat. Egy **write** paranccsal kiírhatunk szöveget és számot ill. változót is, ekkor az egyes elemeket (szöveg, szám vagy változó) vesszővel választjuk el egymástól. Az előző programunk így is kinézhetett volna, a végeredmény ugyanaz:

```
program Pelda01b;
uses crt;
begin
clrscr;
 write('Hat fele:',3);
readln;
end.
```

Mivel itt mindkét példánkba a 3-ast csak kiírtuk, nem számoltunk vele (és nem egy változó értékét írattuk ki), ezért természetesen ebben az esetben a 3-as számot kiírhatjuk úgy is, mint karaktert, tehát a következő képen:

```
program Pelda01c;
uses crt;
begin
clrscr;
 write('Hat fele:3');
readln;
end.
```

Az alábbi két példában jól láthatjuk a különbséget a szöveg, ill. szám kiírása között:

```
program Pelda02a;
uses crt;
begin
clrscr;
 write('2+8');
readln;
end.


program Pelda02b;
uses crt;
begin
clrscr;
 write(2+8);
readln;
end.
```

Az első példa pontosan azt írja ki a képernyőre, ami az aposztrófok között van:

```
2+8_
```

A második példában elhagytuk az aposztrófokat, így a program kiszámolja a 2 és 8 számok összegét és ezt írja ki a képernyőre:

```
10_
```

A **writeln** parancs hasonlóan működik, mint a **write** parancs. A különbség csupán annyi, hogy a **writeln** parancs egy egész sort ír ki, tehát miután kiírta a parancsban megadott szöveget vagy számot, a kurzor a következő sor elejére ugrik. Ezért az utána következő kiírásnál már a kiírandó szöveg az új sorba fog kerülni. Lássunk most erre is egy példát:

```
program Pelda03a;
uses crt;
begin
clrscr;
 writeln('8 es 2 osszege:',8+2);
 writeln('8 es 2 kulonbsege:',8-2);
readln;
end.
```

Ez a program a következőt fogja kiírni a képernyőre:

```
8 es 2 osszege:10
8 es 2 kulonbsege:6

_
```

Az első **writeln** parancs kiírja a *8 és 2 összege:* szöveget, majd mivel a következő rész már nincs idézőjelek között, kiszámolja a 8 + 2 eredményét és kiírja a képernyőre a *10*-et. Ezután, mivel mindez a **writeln** paranccsal lett kiíratva, a kurzor a következő sor elejére ugrik.

A második **writeln** parancs hasonlóan kiírja a *8 és 2 különbsége:* szöveget, kiszámolja mennyi 8 – 2 és az eredményt, tehát a *6*-os számot írja ki. Ez után, mivel ezt is **writeln** paranccsal írattuk ki, a kurzor ismét a következő sor elejére ugrik.

Feladat: Írjuk át az előző programot úgy, hogy az eredményt ne írja közvetlenül a szöveg után, tehát a kettőspont után legyen egy üres hely.

Megoldás: Az aposztrófok közé a kettőspont után teszünk egy szóközt. Így ezt a szóközt is egy az egyben kiírja a program a képernyőre és csak a szóköz után fogja kiírni a két szám összegét ill. különbségét.

```
program Pelda03b;
uses crt;
begin
clrscr;
 writeln('8 es 2 osszege: ',8+2);
 writeln('8 es 2 kulonbsege: ',8-2);
readln;
end.
```

A **writeln** parancsot használhatjuk üres sor kihagyására is, ekkor nem kell a parancs után megadnunk semmit. Valójában a parancs annyit fog tenni, hogy nem ír ki semmit a képernyőre és mivel **writeln** parancsról van szó, ezért a kurzort a következő sor elejére viszi. Módosítsuk az előző példánkat úgy, hogy két sor között hagyjunk ki egy üres sort. Ehhez a két **writeln** parancs közé írjunk be egy újabb **writeln** parancsot, melynél azonban ne adjunk meg semmilyen szöveget amit kiírjon – ilyenkor a zárójeleket sem kell kiraknunk. Programunk így néz ki:

```
program Pelda03c;
uses crt;
begin
clrscr;
 writeln('8 es 2 osszege: ',8+2);
 writeln;
 writeln('8 es 2 kulonbsege: ',8-2);
readln;
end.
```

Ha a programunkat most lefuttatjuk, a következőket fogjuk látni a képernyőn:

```
8 es 2 osszege: 10

8 es 2 kulonbsege: 6
_
```

Láthatjuk, hogy az első sor után az üres **writeln** parancsnak köszönhetően a program kihagyott egy üres sort.

Fontos még megjegyezni, hogy a pascal programozási nyelvben a szorzást *-al, az osztást pedig /-al jelöljük. Ha tehát hasonlóan ki szeretnénk íratni 8 és 2 szorzatát, ill. hányadosát, ezeket a 8*2 ill. 8/2 segítségével tehetjük meg. Ezen kívül az egészrészes osztásra használni fogjuk még a **div**, maradék meghatározására pedig a **mod** műveletet.

A **div** segítségével kiszámolhatjuk két szám hányadosának egész részét. Pl. a 11 div 6 értéke 1, mivel 11 osztva 6-tal egyenlő 1,83333...-mal és ennek az egész része (a tizedeseket levágva) 1.

A **mod** segítségével kiszámolhatjuk két szám egész osztásának maradékát. Pl. a **11 mod 6** értéke 5, mivel 11 osztva 6-tal egészekre kiszámolva 1 és a maradék 5 (11 : 6 = 1, maradék 5).

2 Változók használata

- **integer**, **string** típusú változók
- változók beolvasása billentyűzetről - **readln**
- **byte, real, char**, és **boolean** típusú változók

2.1 Integer, string típusú változók

Ha a programunk futtatása alatt tárolni szeretnénk valamilyen adatokat egy bizonyos ideig, akkor változókra lesz szükségünk. Mielőtt azonban bármilyen változót használni szeretnénk, azt a program elején fel kell sorolnunk (deklarálnunk kell) és meg kell határoznunk hogy minek a tárolására fogjuk használni az adott változót – egész számnak vagy szövegnek. Ez szerint két adattípussal ismerkedünk most meg.

Az **integer** típusú változókban egész számok tárolhatunk, tehát az ilyen típusú változóink értéke egész szám lehet. Az ilyen változó –32768-tól 32767-ig vehet fel értékeket.

A **string** típusú változó szöveg tárolására alkalmas, tehát az ilyen típusú változó értéke bármilyen szöveg lehet, melynek azonban a hossza nem lehet több mint 255 karakter.

Amint már említettük, ha változókat akarunk a programunkba használni, azokat a program elején a **var** kulcsszó után fel kell sorolnunk, ahol meg kell adnunk a változóink típusát is. Pl.:

```
program Pelda04;
uses crt;
var i:integer;
    s:string;
begin
clrscr;

readln;
end.
```

Ebben a programban két változót deklaráltunk. Az egyik változó neve: i, melynek értéke egész szám lehet (mivel **integer** típusú), a másik változó neve: s, amelynek az értéke szöveg lehet (mivel **string** típusú). A változók neve, ami az esetünkben az i és az s, bármilyen betű vagy szó lehet, de minden változónév csak betűre kezdődhet és csak az angol ábécé betűit és számokat tartalmazhat. Ha több ugyanolyan típusú változót szeretnénk használni, akkor azokat felsorolással, vesszővel elválasztva adhatjuk meg:

```
program Pelda05a;
uses crt;
var a1,a2,i:integer;
begin
clrscr;

readln;
end.
```

Ebben a programban 3 változót használtunk: a1, a2 és i. Mindhárom változónk **integer** típusú. Természetesen ha nekünk úgy jobban tetszik, a változók típusát akár külön-külön is megadhatjuk, vagy némely hasonló nevűt együtt csoportosíthatunk, az eredmény ugyanaz lesz:

```
program Pelda05b;
uses crt;
var a1,a2:integer;
    i:integer;
begin
clrscr;

readln;
end.
```

A fenti program az a1 és a2 típusát egyszerre, az i változó típusát külön adtuk meg, de ez a programunkon semmit sem változtat, hisz itt is ugyanazok a változóink vannak és mindhármuk típusa **integer**.

Miután a programunk elején a **var** szócska után deklaráltuk a változókat, elkezdhetjük ezeket használni a programunkban. A változónak valamilyen értéket a := (olvassuk így: **legyen egyenlő**) segítségével adhatunk. Például az i változónak így adhatunk értéket (így rakhatjuk bele a 25-ös számot):

```
i:=25;
```

Szavakkal: i legyen egyenlő 25-tel. Ha szöveges változónak (**string**) szeretnénk valamilyen értéket adni, akkor hasonlóan mint a **write** és **writeln** parancsoknál aposztrófokat kell használnunk:

```
s:='alma';
```

Mivel ez egy parancs, amivel a változónak értéket adunk, ezért az értékadás után természetesen pontosvesszőt kell tennünk.

Feladat: Mi a különbség a következő két utasítás között?

```
a:='z';                    a:=z;
```

Megoldás: Látjuk, hogy a különbség csak az aposztrófokban van. Mit jelent ez valójában?

- Az első utasításnál az **a** változóba beleraktuk a z betűt. Itt csak egy váltózónk van, mégpedig az **a**, amely egy **string** típusú változó (mivel egy z betűt raktunk bele).
- A második utasításnál nincs aposztróf. Ez annyit jelent, hogy az **a** változóba beleraktuk a **z** változó értékét (tartalmát). Ha például a parancs kiadása előtt **z** változóban a 25-ös szám volt, akkor az a:=z; paranccsal az **a** változóba is beleraktunk a 25-ös számot. Ebben az esetben tehát bármi is volt eredetileg az **a** változóban, a parancs végrehajtása után az **a**-ban is a 25-ös szám lesz. Amint láthatjuk, itt két ugyanolyan típusú változónk van, mégpedig az **a** és a **z** változó.

Lássunk most egy rövid kis programot, melyben két változót fogunk használni: **a** és **b**. Mindkét változónk integer típusú lesz. A programban az **a** változóba berakjuk a 8-as számot, a **b** változóba pedig a 6-ost. Végül kiíratjuk a képernyőre a két változó összegét:

```
program Pelda06;
uses crt;
var a,b:integer;
begin
clrscr;
 a:=8;
 b:=6;
 writeln('A ket valtozo osszege: ',a+b);
readln;
end.
```

Programunk a következőt fogja kiírni a képernyőre:

```
A ket valtozo osszege: 14
_
```

Láthattuk, hogy a kiíratásnál az a+b nincs aposztrófok között. Ez így helyes, hiszen nem azt a három karaktert akartuk kiíratni a képernyőre, hogy: **a+b**, hanem a két változó értékeinek – vagyis a 8-nak és a 6-nak – az összegét, tehát a 14-et. Jegyezzük meg, hogy a változók értékeinek kiíratásánál soha ne rakjunk aposztrófokat, hiszen a változók értékeit akarjuk kiíratni (tehát azt amit tartalmaznak).

Feladat: Van három **integer** típusú változónk. Állapítsuk meg, mit csinál a következő programrész az a és b változók értékeivel:

```
x:=a;
a:=b;
b:=x;
```

Megoldás: A programrész felcseréli **a** és **b** változók értékeit. Vegyünk egy példát. Legyen a programrész lefutása előtt az **a=5** és a **b=8**. Ekkor fenti a programrész a következőt fogja tenni:

1. Az első sor az **a** változóban levő értéket berakja az **x** változóba. Tehát az első sor lefutása után az **a**, **b** értéke változatlan, az **x** értéke pedig ugyanaz, mint az **a** változó értéke (tehát **a=5**, **b=8**, **x=5**).
2. A második sorban a **b** változó értékét raktuk bele az **a** változóba (tehát **a=8**, **b=8**, **x=5**). Így az **a** változóban levő eredeti értékünk elveszett, de mivel az első sorban ezt beraktuk az **x** változóba, ezért ott még megvan.
3. A harmadik sorban az **x** változóból raktuk át (vissza) az értékét a **b** változóba (tehát **a=8**, **b=5**, **x=5**).

Így valójában az **a** és **b** változó értékét felcseréltük. Az **x** segédváltozóra azért volt szükségünk, hogy ebben megőrizzük az **a** értékét mielőtt az **a-**ba beleraktuk volna a **b** értékét. Végül ebből az **x** változóból raktuk vissza a megőrzött értéket a **b** változóba.

Végül lássunk egy példát, melynél ugyanabba a változóba teszünk előbb 25-öt, majd 17-et. Ellenőrzésképpen mindig amikor a k változónak új értéket adunk, kiíratjuk a változó értékét a képernyőre. Megfigyelhetjük, hogy a **k** változó értéke mindig csak az utoljára megadott szám (ez előtte belerakott szám elveszik).

```
program Pelda07;
uses crt;
var k:integer;
begin
clrscr;
 k:=25;
 writeln('A k valtozo erteke: ',k);
 k:=17;
 writeln('A k valtozo erteke: ',k);
readln;
end.
```

*Megjegyzés: egy változónak amíg nem adunk a programban semmilyen értéket, addig az értéke **integer** típusú változó esetén: 0, **string** típusú változó esetén pedig üres szó. Ez azonban csak a pascal nyelvben igaz, más programozási nyelvekben a változó kezdeti értéke nincs meghatározva, ami annyit jelent hogy értéke kezdetben bármi lehet.*

2.2 Változók beolvasása billentyűzetről - readln

Eddig a változó értékét mindig előre megadtuk a programban. Gyakran előfordulhat azonban, hogy a változó értékét a program felhasználójától szeretnénk megtudni. A **readln** parancs segítségével a program futása alatt a felhasználó billentyűzeten keresztül adhatja meg az a változó értékét.

Például szeretnénk egy olyan programot készíteni, amelyet ha elindítunk, akkor a számítógép előbb megkérdezi a nevünket, majd miután megadtuk (pl. beírtuk hogy Micimackó) üdvözöl bennünket (kiírja, hogy Szia Micimackó!). Mivel a nevünk valamilyen szó, ezért a név tárolására egy **string** típusú változót fogunk használni. Programunk a következőképpen néz ki:

```
program Pelda08;
uses crt;
var nev:string;
begin
clrscr;
 write('Kerlek add meg a neved: ');
 readln(nev);
 writeln('Szia ',nev,'!');
readln;
end.
```

Miután a programot elindítottuk, lefut a **write** parancs, ami kiír egy mondatot a képernyőre és a kurzor a kiírt szöveg mögött marad. Majd lefut a **readln** parancs és ahol állt a kurzor, ott kéri a nev változó értékének a megadását. Tehát a következő jelenik meg a képernyőn:

```
Kerlek add meg a neved: _
```

A **readln** parancsnál a program addig vár, amíg nem írunk be valamilyen nevet, majd nem nyomjuk meg az ENTER billentyűt. Ekkor a kurzor a következő sor elejére ugrik, majd folytatódik a program futása.

A következő parancs a programban a **writeln**, amely kiírja az üdvözlő mondatot, majd a nev változó értékét (amit előtte beírtunk a **readln** parancsnál) és végül kirak a képernyőre egy felkiáltójelet, majd a kurzort a következő sor elejére teszi. Most ez olvasható a képernyőnkön:

```
Kerlek add meg a neved: Micimacko
Szia Micimacko!
_
```

A program használhatóságának szempontjából fontos, hogy mindig mielőtt beolvasunk valamilyen változót, írjuk ki a képernyőre, hogy minek a megadását várjuk a felhasználótól. Ezt legcélszerűbb a **write** paranccsal kiírni, mivel ekkor a felhasználó rögtön a kiírt szöveg mellett adhatja meg a változó értéket. Ha itt **write** helyett **writeln**-t használnánk, akkor a szöveg kiírása után a kurzor új sorba ugorna, így a beolvasásnál a felhasználó a következő sor elejére írná be a nevét. Logikailag természetesen ez is ugyanolyan jó, de az előzőnek szebb formája van.

Hasonlóan, mint a **write**, **writeln** parancsoknál, a **readln** parancsnak is van egy **read** verziója. Ennek azonban elsősorban a külső állományokból való beolvasásnál van jelentősége. A különbséget a kettő között egy rövid példán szemléltetjük:

```
program Pelda09;
uses crt;
var a,b,c:integer;
begin
clrscr;
 write('Irj be 3 szamot helykozzel elvalasztva: ');
 read(a,b);
 read(c);
 writeln('A beolvasott szamok: ',a,' ',b,' ',c);
 write('Irj be masik 3 szamot helykozzel elvalasztva: ');
 readln(a,b);
 read(c);      { mivel az elozo readln parancs uj sorra ugrott, ezert itt ez
                 a read parancs keri billentyuzetrol a c ertek megadasat }
  writeln('A beolvasott szamok: ',a,' ',b,' ',c);
readln;
end.
```

Ha lefuttatjuk a programot, a következő történik:

```
Irj be 3 szamot helykozzel elvalasztva: 3 5 7
A beolvasott szamok: 3 5 7
Irj be masik 3 szamot helykozzel elvalasztva: 2 4 6
8
A beolvasott szamok: 2 4 8

_
```

Amint láttuk a példánkban az első **read(a,b)** beolvassa a **3**, **5** számokat, de mivel ezeket **read**-el olvastattuk be a legközelebbi beolvasás innen folytatódhat. Ezért a következő **read(c)** parancsnál nem kell megadnunk billentyűzeten keresztül semmit, itt automatikusan beolvassa a **7**-es számot.

Miután ismét megadtunk három számot, a **readln(a,b)** parancs beolvassa ezek közül a **2**, **4**-es számokat, majd mivel ezt a **readln** (read line) utasítással végeztettük el, a legközelebbi beolvasás a következő sorban fog folytatódni. Ezért a **read(c)** már a következő (új) sor elején várja a **c** változó értékének billentyűzeten keresztüli megadását. Itt miután beírtuk a 8-as számot, az bekerül a **c** változóba.

Mivel a mi programjainkban általában egyszerre csak egy változó értékét fogjuk bekérni, ezért szinte mindig a **readln** parancsot fogjuk használni a billentyűzetről való beolvasáshoz.

Fenti példánkban észrevehettük még a kapcsos zárójelek (**{** ... **}**) használatát. Ezek segítségével programunkba bármilyen megjegyzéseket tehetünk, így később (akár hónapok, évek múlva) is könnyen olvasható, érthető lesz a program. Természetesen a program fordításakor és futtatásakor a FreePascal a kapcsos zárójelek közötti szöveget figyelmen kívül hagyja.

Feladat: Készítsünk programot, amely billentyűzetről beolvassa a négyzet oldalának a hosszát (aminek egész számot adunk meg), majd a megadott érték alapján kiszámolja a négyzet kerületét és területét.

Megoldás: A **readln** parancs segítségével beolvastatjuk a négyzet oldalának hosszát egy a változóba. Beolvasás előtt természetesen kiírjuk a felhasználónak hogy milyen adatot kérünk. Miután a felhasználó beírta a kért adatot, kiíratjuk a négyzet kerületét a matematikából jól ismert **k = 4.a** képlet segítségével, majd hasonlóan a négyzet területét a **T = a.a** képlet segítségével.

```
program Pelda10a;
uses crt;
var a:integer;
begin
clrscr;
 write('Kerem a negyzet oldalanak hosszat: ');
 readln(a);
 writeln('A negyzet kerulete: ',4*a);
 writeln('A negyzet terulete: ',a*a);
readln;
end.
```

Ha szeretnénk a kerületet és a területet is megőrizni (változókban tárolni), akkor használhatunk ezekre például egy k és egy t változót, melyekbe először kiszámoljuk a kerületet ill. területet, majd utána kiírjuk ezeknek a változóknak az értékét. Programunk ebben az esetben így néz ki:

```
program Pelda10b;
uses crt;
var a,k,t:integer;
begin
clrscr;
 write('Kerem a negyzet oldalanak hosszat: ');
 readln(a);
 k:=4*a;
 t:=a*a;
 writeln('A negyzet kerulete: ',k);
 writeln('A negyzet terulete: ',t);
readln;
end.
```

A fenti példában is láthattuk, hogy egy feladatnak több helyes megoldása is lehet. Próbáljuk meg ezek közül mindig az egyszerűbb, rövidebb, áttekinthetőbb megoldást megkeresni.

Feladat: Készítsünk hasonló programot a téglalap kerületének és területének kiszámítására. Feltételezzük, hogy itt is a téglalap oldalainak hossza egész szám.

Megoldás: A program a négyzet kerületének és területének kiszámításához hasonló lesz, de itt két értéket fogunk beolvasni: a téglalap a és b oldalának a hosszát, majd ezekből számoljuk ki a kerületet és a területet a matematikából jól ismert képletek segítségével. Egy helyes megoldás a feladatra:

```
program Pelda11a;
uses crt;
var a,b:integer;
begin
clrscr;
 writeln('Kerem a teglalap mereteit.');
 write('a = ');
 readln(a);
 write('b = ');
 readln(b);
 writeln('A teglalap kerulete, k = ',2*a+2*b);
 writeln('A teglalap terulete, T = ',a*b);
readln;
end.
```

2.3 byte, real, char, és boolean típusú változók

A **byte** típusú változónál 0-255-ig tudunk számot tárolni

A **real** (valós) típus ábrázolása 6 bájton, lebegőpontosan történik. Használata: x:real; x:8:2
Az értéktartomány:

$S_{min} = 2{,}9*10^{-39}$
$S_{max} = 1{,}7*10^{38}$

A **char** típus egy karakter használatánál hasznos. Egy bájtos típus, tehát $2^8 = 256$ különböző érték , az ASCII kódrendszer 256 elemének a tárolására képes. A karakter típusú változó egy ASCII kódot tartalmaz. Pl. ha a változóhoz tartozó memóriarekesz értéke 65, akkor mivel változónk típusa Char, ezt a rendszer 'A' betűként értelmezi.

A **boolean** egy logikai típusú változó két értéket vehet fel: *igaz* vagy *hamis*. Ábrázolására egy bájton történik (akár egy bit is elég lenne). Ha a bájt értéke 0, akkor a logikai típusúként értelmezett érték *hamis*, nullától eltérő érték esetén pedig *igaz. (true; false)*

Feladat: Kérj be egy szögértéket fokban, írasd ki a szinuszát! (használd a sin fügvényt)

```
program Pelda11b;
uses crt;
var x,r:real;
begin
  clrscr;
   Writeln('Kerem a szoget fokban: ');
   readln(x);
   r:=x*pi/180;
   writeln('A szog szinusza: ', sin(r):8:2);
  readln;
end.
```

Feladatok:

1. Írasd ki a képernyőre a nevedet a második sortól! (képernyőtörléssel) **(a01)**

```
Vezeteknev Keresztnev
```

2. Írasd ki a képernyőre a következő számokat egy karakter eltolással a minta alapján! **(a02)**

```
1
 2
  3
   4
    5
```

3. Rajzoltasd ki a következő alakzatot a képernyőre a minta alapján! **(a03)**

```
          *
         ***
        *****
       *******
      *********
     ***********
         ***
         ***
```

4. Írasd ki a következőket a képernyőre a következő matematikai képleteket, úgy hogy az eredményt számolás után írja ki! **(a04)**

```
5+4=9
8-6=2
2*3=6
```

5. Kérjél be egy karaktert, majd írasd ki egymás mellé ötször, egy szóköz távolsággal! **(a05)**

```
Add meg a karaktert: X
X X X X X
```

6. Kérjél be két karaktert, majd készíts díszítősort a minta alapján! **(a06)**

```
Add meg az elso karaktert: X
Add meg a masodik karaktert: O
XOXOXOXOXOXOXOXOXOXOXOXOXOXOXO
```

7. Készítsd el a következő adatbekérő kérdőívet, mely után kiíratod a következő mondatot! **(a07)**

```
Add meg a nevedet: Vezetek Keresztnev
Hol szulettel: Szombathely
Mikor szulettel: 1992.01.18
_____________________________________
Vezeteknev Keresztnevnek hivnak,
Szombathelyen szulettem, 1992.01.02-
an -en!
```

8. Kérjél be két számot, majd add össze, vond ki egymásból, aztán szorozd össze őket a minta a lapján! Az eredményt számolással kapd meg! **(a08)**

```
Add meg az elso szamot: 4
Add meg a masodik szamot: 2
_____________________________________
4+2=6
4-2=2
4*2=8
```

9. Bővítsd ki az előző példát úgy, hogy három számot kérjen be, és három számmal végezze el a műveleteket! A választóvonalat töröld! **(a09)**

```
Add meg az elso szamot: 6
Add meg a masodik szamot: 2
Add meg a harmadik szamot: 3
6+2+3=11
6-2-3=1
6*2*3=36
```

10. Készítsd el a következő két képletet, előre deklarált számokkal! Figyelve a zárójelek fontosságára) a=4 b=7 c=5 d=3
(a10)

```
(a+b)*c-d=(4+7)*5-3=52
a+b*(c-d)=4+7*(5-3)=18
```

11. Készíts programot, ami bekéri egy négyzet oldalának egységnyi méretét, majd kiszámolja a négyzet kerületét és területét! **(a11)**

```
Add meg a negyzet oldalat: 3
_____________________________
A negyzet kerulete: 12
A negyzet terulete: 9
```

12. Változtasd meg az előző programot úgy, hogy egy kocka felszínét és térfogatát számolja ki. Ebben a programban használj mértékegységet is! **(a12)**

```
Add meg a kocka oldalat: 5 cm
_____________________________
A kocka felulete: 150 cm2
A kocka terfogata: 125 cm3
```

13. A program számítógépbe írása nélkül határozzuk meg, hogy mit ír ki a képernyőre a következő program, majd futtassuk le számítógépen is, hogy meggyőződjünk válaszunk helyességéről.

```
program valtozo;
uses crt;
var x:integer;
begin
clrscr;
 x:=12;
 write(x,'-');
 x:=3;
 write(x,'=');
 writeln(12-x);
 write(x+6,'+',x,'=');
 x:=4*x;
 writeln(x);
readln;
end.
```

Írjunk egy rövidebb (egyszerűbb) programot, amely változó használata nélkül kiírja ugyanezt az üzenetet a képernyőre. A teljes programunk maximum 5 soros legyen! **(a13)**

```
??????????????
```

14. Almát szeretnénk vásárolni. Írjunk egy programot, amely billentyűzetről kérje be először azt, hogy mennyibe kerül egy kilogramm alma, majd azt, hogy hány kilogramm almát szeretnénk venni. A program számolja ki, hogy ennyi almáért hány koronát fogunk fizetni. **(a14)**

```
Egy kg alma ara: 12
Hany kg almat veszel: 3
Ennyi alma ara 36 korona.
```

15. Kérjünk be két, egy napon belüli, időpontot (először az órát, aztán a percet, végül a másodpercet). Számítsuk ki a két időpont közti különbséget másodpercekben és írassuk ki! **(a15)**
(Abs() függvény használható)

```
Elso idopont - ora: 7
Elso idopont - perc: 15
Elso idopont - masodperc: 26
Masodik idopont - ora: 8
Masodik idopont - perc: 16
Masodik idopont - masodperc: 30
A ket idopont kozott 3664 masodperc
telt el.
```

16. Írjunk programot, amely beolvas két természetes számot, majd kiírja a két szám hányadosát és maradékát az alábbi formában. A program az adatok beolvasása után hagyjon ki egy üres sort. **(a16)**

```
Elso szam: 17
Masodik szam: 3

17:3=5, maradek 2
```

17. A program kérjen be egy számot, majd írja ki a kis szorzótáblát erre a számra (1-től 5-ig). A program a beolvasás után hagyjon ki egy üres sort. **(a17)**

```
Melyik szorzotablat irjam ki: 12
1 . 12 = 12
2 . 12 = 24
3 . 12 = 36
4 . 12 = 48
5 . 12 = 60
```

18. Az alábbi program begépelése nélkül próbáljuk meg meghatározni, mit fog kiírni a képernyőre. Ellenőrzésképpen a programot írjuk be a számítógépbe.

```
??????????????
```

```
program szamok;
uses crt;
var k:integer;
begin
clrscr;
 k:=1;
 writeln(k);
 k:=k+1;
 write(k);
 k:=k+1;
 writeln(k);
 k:=k+1;
 write(k);
 k:=k+1;
 write(k);
 k:=k+1;
 writeln(k);
readln;
end.
```

Majd begépelve ellenőrizzük a helyességet! **(a18)**

19. Kérjünk be három természetes számot, ezek rendre 5, 2 és 1 koronásaink számát jelentik. Határozzuk meg, és írassuk ki a teljes összeget. **(a19)**

```
5 koronasok szama: 2
2 koronasok szama: 3
1 koronasok szama: 1
Ez osszesen 17 korona.
```

20. A program kérjen be egy pénzösszeget, majd határozza meg, és írja ki, hogy hogyan fizethetjük ki a lehető legkevesebb 10, 5, 2 és 1 koronás érmével (használjuk az első fejezet tananyagában megismert **mod** és **div** műveleteket)! **(a20)**

```
Kifizetendo penzosszeg: 26

2 darab 10 koronas erme,
1 darab 5 koronas erme,
0 darab 2 koronas erme,
1 darab 1 koronas erme.
```

3 A FOR ciklus

- a **FOR** ciklus
- a szám ill. szöveg kiírása előre megadott hosszúságú helyre
- a **BEGIN** ... **END;** kulcsszavak használata

3.1 A FOR ciklus

Gyakran előfordul, hogy a programunkban valamit többször meg szeretnénk ismételni. Ilyenkor ciklusokat használunk. Ezeknek több fajtájuk van, most a **for** ciklussal fogunk foglalkozni, melynek a következő szerkezete van:

```
for ciklusváltozó := kifejezés1 to kifejezés2 do utasítás ;
```

A *ciklusváltozó* felveszi először a *kifejezés1* értékét. Végrehajtja az *utasítás*-t, majd a *ciklusváltozó* növekszik eggyel és ismét végrehajtja az *utasítás*-t. Ezután ismét növekszik eggyel és végrehajtja az *utasítás*-t. Mindezt addig ismétli, amíg a *ciklusváltozó* nem lesz egyenlő a *kifejezés2* értékével. Ekkor még utoljára végrehajtja az utasítást.

Ha a *kifejezés2* értéke kisebb, mint a *kifejezés1* értéke, akkor az utasítást egyszer sem hajtja végre.

Ha a *kifejezés1* értéke egyenlő a *kifejezés2* értékével, akkor az utasítást csak egyszer hajtja végre.

Példaként készítsünk egy programot, amely 1-től 10-ig kiírja az összes egész számot és mellé a szám négyzetét is. Ehhez szükségünk lesz egy ciklusváltozóra - jelöljük ezt most i-vel. Mivel ez a változó egész számértékeket fog felvenni, ezért ezt **integer** típusú változónak deklaráljuk. Ennek a változónak az értéke először 1 lesz, majd 2, 3,... egészen 10-ig. Ezt a folytonos növekedést 1-től 10-ig ciklus segítségével fogjuk megoldani. A ciklusmagban mindig ki fogjuk íratni az i változó értékét és hozzá a szám négyzetét is, tehát az i*i értékét. Programunk így nézni ki:

```
program Pelda12a;
uses crt;
var i:integer;
begin
 clrscr;
 for i:=1 to 10 do writeln(i,' negyzete = ',i*i);
 readln;
end.
```

Ez a program a következőt fogja kiírni a képernyőre:

```
1 negyzete = 1
2 negyzete = 4
3 negyzete = 9
4 negyzete = 16
5 negyzete = 25
6 negyzete = 36
7 negyzete = 49
8 negyzete = 64
9 negyzete = 81
10 negyzete = 100
_
```

Itt fontos megjegyeznünk, hogy egy szám négyzetének kiszámítására létezik a pascalban matematikai függvény is, mégpedig az **sqr()**. A programunkban tehát az i*i helyett írhattunk volna sqr(i) kifejezést. A későbbiekben már ezt a függvényt fogjuk használni egy szám négyzetének a kiszámítására.

Hasonlóan a négyzetgyök kiszámítására is létezik egy függvény, ez pedig az **sqrt()**.

3.2 A szám ill. szöveg kiírása előre megadott hosszúságú helyre

A kiírásnál megfigyelhettük, hogy a program az 1, 2, ... 9 számokat egymás alá írta, de a 10 számnál (mivel ez kétjegyű), a szöveg további része egy hellyel arrébb tolódott. Szöveg kiírásánál van arra is lehetőségünk, hogy egy bizonyos szöveget, számot előre megadott hosszúságú helyre írjunk ki. Tehát például az 1, 2, ... 9 számokat is két helyre írjuk ki, még ha ezek csak egyet foglalnak is el. Ezt a **writeln** parancsban adhatjuk meg a kiírandó szám mögé írva a **:2**-t, ahol a 2 azt jelenti, hogy az előtte levő számot 2 helyre szeretnénk kiírni. Programunk ez után a módosítás után így néz ki:

```
program Pelda12b;
uses crt;
var i:integer;
begin
 clrscr;
 for i:=1 to 10 do writeln(i:2,' negyzete = ',sqr(i));
 readln;
end.
```

És ez fog megjelenni a képernyőn, miután lefuttattuk:

```
 1 negyzete = 1
 2 negyzete = 4
 3 negyzete = 9
 4 negyzete = 16
 5 negyzete = 25
 6 negyzete = 36
 7 negyzete = 49
 8 negyzete = 64
 9 negyzete = 81
10 negyzete = 100
_
```

Láthattuk, hogy az egyjegyű számokat is most már két helyre írja ki a program, mégpedig úgy, hogy a szám elé rak egy szóközt. Hasonlóan megoldhatjuk, hogy az 1, 4, 9, 16, ... számoknál is az egyesek az egyesek alatt, a tízesek a tízesek alatt, stb. legyenek. Mivel itt a legnagyobb szám háromjegyű, ezért itt minden számot három helyre íratunk ki:

```
program Pelda12c;
uses crt;
var i:integer;
begin
 clrscr;
 for i:=1 to 10 do writeln(i:2,' negyzete = ',sqr(i):3);
 readln;
end.
```

Ez a program már így fogja kiírni a számokat a képernyőre:

```
 1 negyzete =   1
 2 negyzete =   4
 3 negyzete =   9
 4 negyzete =  16
 5 negyzete =  25
 6 negyzete =  36
 7 negyzete =  49
 8 negyzete =  64
 9 negyzete =  81
10 negyzete = 100
_
```

A számhoz hasonlóan egy szöveges változónál vagy egy szövegnél is megadhatjuk, hogy azt milyen hosszú helyre akarjuk kiíratni, például így:

```
writeln('Hova kerul ez?':20);
```

Ilyenkor a szám kiírásához hasonlóan a szöveg elé megfelelő számú szóközt tesz ki a program. Néha ügyelnünk kell arra, hogy a szöveges képernyő felbontása 80 x 25, tehát egy sorban 80 karakter fér el. Ha ennél többet írunk ki, akkor a maradékot (ami nem fér ki) már a következő sor elejére fog kerülni.

3.3 A BEGIN ... END; kulcsszavak használata

Gyakran szükségünk lehet arra, hogy egy cikluson belül több utasítást is elvégezzen a programunk. Például az előző programunk ne csak kiírja a ciklusban azt, hogy melyik számnak mennyi a négyzete, hanem mindegyik sor után egy új sorba rajzoljon (írjon ki) egy vonalat is egyenlőség jelekből. Ekkor már a ciklusunkon belül két **writeln** parancsot kéne használnunk (egyet a ciklusváltozó négyzetének a kiírására, egyet pedig a vonal kiírására). Ez csak úgy oldható meg, ha a ciklus után a parancsokat **begin** ... **end;** kulcsszavak közé rakjuk. Ezt használva természetesen nem csak kettő, de akár több parancsot is összekapcsolhatunk a cikluson belül. Ilyen esetben a ciklusunk így néz ki:

```
for ciklusváltozó := kifejezés1 to kifejezés2 do  begin
                                                     első parancs ;
                                                     második parancs ;
                                                     ...
                                                     utolsó parancs ;
                                                  end ;
```

Így valójában a ciklusváltozó minden egyes értékére végrehajtódik a cikluson belüli első, második, ... utolsó parancsot. Ilyen **begin** ... **end;** parancsot máshol is fogunk még használni, ahol a cikluson (feltételen) belül egy parancs helyett többet szeretnénk elvégezni (összekapcsolni).

Próbáljuk meg most átírni a négyzetszámok nevű programunkat úgy, hogy minden kiírás után írjon ki új sorba egy egyenlőség jelekből álló vonalat is. Ezt a vonalat egy újabb **writeln** paranccsal fogjuk kiíratni, melyet az eddigi **writeln** parancsal együtt a **begin** ... **end;** kulcsszavak közé rakunk. Programunk tehát így néz ki:

```
program Pelda12d;
uses crt;
var i:integer;
begin
 clrscr;
 for i:=1 to 10 do begin
                   writeln(i:2,' negyzete = ',sqr(i):3);
                   writeln('==================');
                   end;
 readln;
end.
```

És ha lefuttatjuk, ezt írja ki a képernyőre:

```
 1 negyzete =   1
 2 negyzete =   4
 3 negyzete =   9
 4 negyzete =  16
 5 negyzete =  25
 6 negyzete =  36
 7 negyzete =  49
 8 negyzete =  64
 9 negyzete =  81
10 negyzete = 100
_
```

Feladat: Készítsünk programot, amely az előző feladathoz hasonlóan kiírja 1-től 10-ig mindegyik egész szám négyzetét egymás alá (az egyenlőségekből álló vonalakat most ne írja ki). Ez után az egész kiírás után rakjon ki egy mínusz jelekből álló vonalat és ez alá a vonal alá írja ki a program hogy mennyi a kiírt négyzetszámok összege, tehát hogy mennyi az 1+4+9+16+25+36+...+100 összeg.

Megoldás: A feladat megoldásához bevezetünk egy s változót, melynek típusa **integer**. Ennek a változónak a program elején 0 értéket adunk, majd a ciklusban mindig miután kiírtuk a négyzetszámot, hozzáadjuk ehhez a változóhoz is az éppen kiírt négyzetszámot. Így a ciklus lefutása után az s változóban a keresett összeg lesz. A ciklus lefutása után már csak kiírunk egy mínusz jelekből álló vonalat, majd kiírjuk hogy mennyi ennek az s változónak az értéke. Programunk így néz ki:

```
program Pelda13;
uses crt;
var i,s:integer;
begin
 clrscr;
 s:=0;
 for i:=1 to 10 do begin
                   writeln(i:2,' negyzete = ',sqr(i):3);
                   s:=s+sqr(i);
                   end;
 writeln('-----------------');
 writeln('Ezek osszege: ',s);
 readln;
end.
```

Miután lefuttattuk a programot, a következő jelent meg a képernyőn:

```
 1 negyzete =   1
 2 negyzete =   4
 3 negyzete =   9
 4 negyzete =  16
 5 negyzete =  25
 6 negyzete =  36
 7 negyzete =  49
 8 negyzete =  64
 9 negyzete =  81
10 negyzete = 100
-----------------
Ezek osszege: 385
_
```

A fenti programban érdekes lehet még az s:=s+sqr(i); sor. Ez azt csinálja, hogy az **s** változó értékéhez (melyet a program legelején beállítottunk 0-ra) hozzáadja az **i** változó értékének a négyzetét, majd az eredményt beteszi az **s** változóba. Tehát valójában az **s** változó értékéhez hozzáadja az **i** négyzetét. Mivel a ciklusban az **i** 1-től 10-ig megy, ezért az **s**-hez sorban hozzáadja az 1 négyzetét, majd a 2 négyzetét, 3 négyzetét, stb. egészen a 10 négyzetéig.

Feladatok:

1. Készítsünk programot, amely kiszámolja az első 100 drb. természetes szám összegét, majd kiírja az eredményt. (Az összeg kiszámolásához vezessünk be egy változót, amelyet a program elején kinullázunk, a ciklusmagban pedig mindig hozzáadjuk a ciklusváltozó értékét, tehát sorban az 1, 2, 3, 4, ..., 100 számokat.) **(for01)**

```
Az elso szaz szam osszege: XXXXX
```

2. Készítsünk programot, amely kiszámolja az első 7 db. természetes szám szorzatát egy ciklus segítségével. (A szorzat kiszámolásához vezessünk be egy változót, amelyet a program elején beállítunk 1-re, a ciklusmagban pedig mindig hozzászorozzuk a ciklusváltozó értékét, tehát sorban az 1, 2, 3, ..., 7 számokat.) **(for02)**

```
Az első 7 szam szorzata: XXXX
```

3. Készítsünk programot, amely kiszámolja 100-ig a páros számok összegét (A ciklus vegyük egytől ötvenig, majd a ciklusmagban vegyük a ciklusváltozó kétszeresét - így megkapjuk a páros számokat. Ezeket hasonlóan adjuk össze, mint az első feladatban). **(for03)**

```
Paros szamok osszege 100-ig: XXXX
```

4. Készítsünk programot, amely kiszámolja 100-ig a páratlan számok összegét (A ciklus vegyük egytől ötvenig, majd a ciklusmagban vegyük a ciklusváltozó kétszeresét eggyel csökkentve - így megkapjuk a páratlan számokat. Ezeket hasonlóan adjuk össze, mint az első feladatban). **(for04)**

```
Paratlan szamok osszege 100-ig:
XXXX
```

5. Készítsünk programot, amely kiírja az első tíz szám négyzetét! **(for05)**

```
1; 4; 9; 16; 25; 36; 49; 64; 81;
100
```

4 Egymásba ágyazott ciklusok

- egymásba ágyazott ciklusok
- csökkenő ciklusváltozó (**TO** helyett **DOWNTO**)

4.1 Egymásba ágyazott ciklusok

Programjaink készítésekor sokszor elő fog fordulni, hogy nem lesz elég egy ciklust használnunk, hanem vagy egymás után, vagy akár egymáson belül fogunk több ciklust használni. Lássunk most az utóbbi esetre egy példát. Mielőtt azonban belekezdenénk, készítsük egy egyszerűbb programot, melyben még csak egy ciklusunk lesz.

Feladat: Készítsünk programot, amely beolvas egy egész számot, majd kiír a képernyőre egymás mellé ennyi darab * (csillag) karaktert.

Megoldás: A programban deklarálni fogunk egy **n** (integer típusú) változót, melybe beolvassuk a program elején a kiírandó csillagok számát. Ezen kívül szükségünk lesz még egy ciklusváltozóra is - ez legyen i. Magában a programban egy egyszerű write('*') parancsot fogunk megismételtetni **n**-szer egy **for** ciklus segítségével (a ciklus most 1-től n-ig fog menni). Programunk így néz ki:

```
program Pelda14a;
uses crt;
var n,i:integer;
begin
 clrscr;
 write('Kerem a kiirando csillagok szamat: ');
 readln(n);
 for i:=1 to n do write('*');
 readln;
end.
```

Miután lefuttattuk a programot, a következő jelent meg a képernyőn:

```
Kerem a kiirando csillagok szamat: 8
********_
```

Módosítsunk most a fenti programon úgy, hogy ne csak egy sornyi csillagot írjon ki, hanem a csillagok segítségével rajzoljon ki egy négyzetet. Például n=8 esetre a következő jelenjen meg a képernyőn:

```
********
********
********
********
********
********
********
********
```

Tehát ne csak egy sorba írjon ki n drb. csillagot, hanem írjon ki n darab sort, melyek mindegyikében n darab csillag legyen. Ezt úgy érhetjük el, hogy a fenti programban levő ciklus után (amely kiír egy sornyi csillagot) teszünk egy writeln utasítást (ezzel a kiírt sor után egy új sorba kerülünk), majd a sor kiírására szolgáló ciklust az utána következő **writeln** paranccsal együtt (begin..end kulcsszavakkal összekapcsolva) megismételjük **n**-szer (egy külső ciklusban).

Programunk így néz ki:

```
program Pelda14b;
uses crt;
var n,i,j:integer;
begin
 clrscr;
 write('Kerem a kiirando csillagok szamat: ');
 readln(n);
 for j:=1 to n do
  begin
  for i:=1 to n do write('*');
  writeln;
  end;
 readln;
end.
```

A program a következőt fogja kiírni a képernyőre:

```
Kerem a kiirando csillagok szamat: 8
********
********
********
********
********
********
********
********
_
```

Fontos, hogy ha ilyen és ehhez hasonló egymásba ágyazott ciklusokat használunk, akkor a külső ciklusnak egy másik ciklusváltozót kell választanunk, mint a belső ciklusnak. A mi példánkban a külső ciklus ciklusváltozója j, a belső ciklusé pedig i. A két ciklus valójában a következő képen fut le:

- a külső ciklusnál a j kezdeti értéke 1 lesz (j=1),

  - a belső ciklusnál az i kezdeti értéke 1 lesz (i=1), majd végrehajtódik a write('*') parancs,
  - a belső ciklusnál az i értéke növekszik (i=2), majd végrehajtódik a write('*') parancs,
  - ...
  - a belső ciklusnál az i értéke növekszik (i=n), majd végrehajtódik a write('*') parancs,

  - a writeln parancs lefutásával a kurzor egy új sorba kerül,

- a külső ciklusnál a j értéke növekszik (j=2),

  - a belső ciklusnál az i kezdeti értéke 1 lesz (i=1), majd végrehajtódik a write('*') parancs,
  - a belső ciklusnál az i értéke növekszik (i=2), majd végrehajtódik a write('*') parancs,
  - ...
  - a belső ciklusnál az i értéke növekszik (i=n), majd végrehajtódik a write('*') parancs,

  - a writeln parancs lefutásával a kurzor egy új sorba kerül,

- a külső ciklusnál a j értéke növekszik (j=3),

  - a belső ciklusnál az i kezdeti értéke 1 lesz (i=1), majd végrehajtódik a write('*') parancs,
  - a belső ciklusnál az i értéke növekszik (i=2), majd végrehajtódik a write('*') parancs,
  - ...
  - a belső ciklusnál az i értéke növekszik (i=n), majd végrehajtódik a write('*') parancs,

  - a writeln parancs lefutásával a kurzor egy új sorba kerül,
- ...

Ez így folytatódik mindaddig, amíg a külső ciklus ciklusváltozója (j) nem éri el az n értékét, majd ezen belül a belső ciklus ciklusváltozója (i) is nem éri el az n értékét.

Ebből is látszódik, hogy ha a két ciklusban ugyanaz a ciklusváltozó lenne (pl. mindkettőben i), akkor a belső ciklus változtatná a külső ciklus által beállított értéket, ami hibához vezetne.

4.2 Csökkenő ciklusváltozó (TO helyett DOWNTO)

Néha előfordulhat, hogy olyan ciklusra van szükségünk, melyben a ciklusváltozó nem növekszik (pl. 1-től 10-ig), hanem csökken (pl. 10-től 1-ig). Ilyenkor egyszerűen a **to** helyett a **downto**-t fogjuk használni:

```
for ciklusváltozó := kifejezés1 downto kifejezés2 do utasítás ;
```

Ebben az esetben *ciklusváltozó* felveszi először a *kifejezés1* értékét. Végrehajtja az *utasítás*-t, majd a *ciklusváltozó* csökken eggyel és ismét végrehajtja az *utasítás*-t. Ezután ismét csökken eggyel és végrehajtja az *utasítás*-t. Mindezt addig fogja csinálni, amíg a *ciklusváltozó* nem lesz egyenlő a *kifejezés2* értékével. Ekkor még utoljára végrehajtja az utasítást.

Ha a *kifejezés2* értéke nagyobb, mint a *kifejezés1* értéke, akkor az utasítást egyszer sem hajtja végre.

Ha a *kifejezés1* értéke egyenlő a *kifejezés2* értékével, akkor az utasítást csak egyszer hajtja végre.

Feladat: Készítsünk programot, amely bekér egy **N** természetes számot, majd kihagy egy üres sort, és kiírja egymás mellé N-től 0-ig az összes egész számokat (mindegyik szám után egy szóközt rak).

Megoldás: Mivel a számokat csökkenő sorrendben kell kiírnunk, ezért a ciklusban a fent említett downto-t fogjuk használni:

```
program Pelda15;
uses crt;
var i,n:integer;
begin
 clrscr;
 write('Kerem az N szamot: ');
 readln(n);
 writeln;
 for i:=n downto 0 do write(i,' ');
 readln;
end.
```

Lássuk mit írt ki a programunk, ha a program futásakor az N számnak 15-öt írunk be:

```
Kerem az N szamot: 15

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 _
```

Próbáljuk programunkat lefuttatni többször is, mindig az N-nek más számot megadva. Hogy a következő futtatásnál ne a 0 után kezdődjön a kiírás a képernyőre, kiegészíthetjük ezt a programot még úgy, hogy a végére a ciklus után beírunk egy **writeln;** parancsot (esetleg kettőt, ha szeretnénk hogy programunk után egy üres sort is kihagyjon).

Feladatok:

1. Készítsünk programot, amely bekér egy **N** természetes számot, majd kirajzol a képernyőre egymás mellé **N**-szer az "**XO**" betűket és a kiírás után a kurzort a következő sor elejére teszi. **(for06)**

Például **N=3**-ra kiírja ezt a program:

XOXOXO

```
Add meg az n-t: 5
XOXOXOXOXO
```

2. Egészítsük ki az előző programunkat úgy, hogy az előző kiírás alá írja ki **N**-szer az "**OX**" betűket is egymás mellé, majd a kurzort ismét a következő sor elejére tegye. (Az előző ciklus után - NE bele a ciklusba! - tegyünk egy hasonló ciklust, ami most XO helyett OX betűket ír ki.) **(for07)**

Például **N=3**-ra kiírja ezt a program:

XOXOXO
OXOXOX

```
Add meg az n-t: 5
XOXOXOXOXO
OXOXOXOXOX
```

3. Egészítsük ki a programunkat úgy, hogy az előző két sort **N**-szer ismételje meg a program. (Az előző két egymás utáni ciklust tegyük bele egy külső ciklusba.) **(for08)**

Például **N=3**-ra kiírja ezt a program:

XOXOXO
OXOXOX
XOXOXO
OXOXOX
XOXOXO
OXOXOX

```
Add meg az n-t: 5
XOXOXOXOXO
OXOXOXOXOX
XOXOXOXOXO
OXOXOXOXOX
XOXOXOXOXO
OXOXOXOXOX
XOXOXOXOXO
OXOXOXOXOX
XOXOXOXOXO
OXOXOXOXOX
```

4. Készítsünk programot, amely beolvas egy **N** természetes számot, majd billentyűzetről bekér **N** db. természetes számot és ezeket a számokat összeadja, majd kiírja az eredményt. (Vegyünk egy változót, amit a program elején kinullázunk. Ehhez a cikluson belül mindig adjuk hozzá az éppen beolvasott számot. A szám beolvasása a ciklusban lehet N-szer ugyanabba a változóba, hiszen miután hozzáadtuk az összeghez, már nincs rá szükségünk, tehát használhatjuk a következő szám beolvasására.) **(for09)**

```
Add meg, hogy hany szamot kerjen be:
5
Add meg az 1. szamot: 2
Add meg az 2. szamot: 5
Add meg az 3. szamot: 3
Add meg az 4. szamot: 7
Add meg az 5. szamot: 1

A szamok osszege: 18
```

5. Készítsünk programot, amely kiszámolja az első **N** db. természetes szám szorzatát, majd kiírja az eredményt. (Ehhez vegyünk egy változót, amelyet a program elején beállítunk 1-re, majd a ciklusban ezt sorban megszorozzuk az 1, 2, 3, ..., N számokkal.) **(for10)**
Például **N=4**-re az eredmény **24**, mivel 1*2*3*4=24.

```
Add meg, hogy hany n-ig szorozzon: 6
1*2*3*4*5*6*=720
```

6. Készítsünk programot, amely kiszámolja az első **N** db. természetes szám összegét. (Ehhez vegyünk egy változót, amelyet a program elején beállítunk 0-ra, majd a ciklusban ehhez sorban hozzáadjuk az 1, 2, 3, ..., N számokat.) **(for11)**

Például **N=5**-re az eredmény **15**, mivel 1+2+3+4+5=15.

```
Add meg az n-t: 5
1+2+3+4+5+=15
```

7. Készítsünk programot, amely kiszámolja az első **N** db. páros szám összegét. (A ciklus 1-től **N div 2**-ig menjen, majd a ciklusmagban vegyük a számok kétszeresét.) **(for12)**

```
Add meg az n-t: 7
2+4+6+=12
```

8. Készítsünk programot, amely kiszámolja az első **N** db. páratlan szám összegét. (A ciklus 1-től **N div 2**-ig menjen, majd a ciklusmagban vegyük a számok kétszeresét eggyel csökkentve.) **(for13)**

```
Add meg az n-t: 7
1+3+5+7+=16
```

9. Készítsünk programot, amely bekéri a **K** pozitív egész számot, majd kiszámolja a következő összeget: **1·2 + 2·3 + 3·4 + 4·5 + ... + K·(K+1)** **(for14)**

```
Add meg az k-t: 8
1*2+2*3+3*4+4*5+5*6+6*7+7*8+8*9=240
```

10. Kérjünk be egy **N** természetes számot, majd írassuk ki a három összes olyan többszörösét, amely kisebb vagy egyenlő mint **N**. **(for15)**

```
Add meg az n-t: 7
3; 6;
```

11. Kérjünk be két természetes számot (**M**,**N**), majd rajzoljunk ki a képernyőre egy **M**x**N** méretű téglalapot csillag (*) jelekből. **(for16)**

Például **M=8** és **N=3**-ra:

```
********
********
********
```

```
Add meg az n-t: 4
Add meg az m-et: 5
*****
*****
*****
*****
```

12. Állítsuk elő és írassuk ki, a minta alapján az első **N** faktoriális számot! n! **(for17)**

```
Add meg az n-t: 5
n!=1*2*3*4*5=120
```

13. Állítsuk elő és írassuk ki az első **N** darab Fibonacci-szám összegét (a Fibonacci sorozatnak az a jellemzője, hogy bármelyik eleme egyenlő az előző kettő összegével). **(for18)** Az összeg, melyet számoljon ki a program az első **N** elemből: **1** + **1** + **2** + **3** + **5** + **8** + **13** + ...

```
Add meg az n-t: 7
1+1+2+3+5+8+13=33
```

5 Karakterláncok (string)

- karakterláncok (**string** típusú változók)

5.1 Karakterláncok (string típusú változók)

A **string** típusú változókba szöveget olvashatunk be. Az ilyen változó maximális hossza 255 karakter.

Feladat: Készítsünk programot, amely beolvas egy mondatot, majd kiírja hány karakterből áll a mondat.

Megoldás: A feladat megoldásához bevezetünk egy s változót, melynek típusa **string**. Beolvasás után ennek a változónak a hosszát a **length()** függvénnyel kapjuk meg. Programunk így néz ki:

```
program Pelda16;
uses crt;
var s:string;
begin
 clrscr;
 write('Irj be egy mondatot: ');
 readln(s);
 writeln('A mondat hossza: ',length(s),' karakret.');
 readln;
end.
```

Feladat: Készítsünk programot, amely beolvas egy mondatot, majd kiírja a mondatot nagy betűkkel.

Megoldás: A feladat megoldásához bevezetünk egy s változót, melynek típusa **string**. Beolvasás után a mondat minden egyes karakterét egy cikluson belül megváltoztatjuk nagy betűre. Az **s** változóban levő mondat első betűjét az **s[1]** adja meg, második betűjét az **s[2]** adja meg, stb. Egy karaktert nagy betűre az **upcase()** függvény segítségével változtathatunk meg. Programunk így néz ki:

```
program Pelda17;
uses crt;
var s:string;
    i:integer;
begin
 clrscr;
 write('Irj be egy mondatot: ');
 readln(s);
 for i:=1 to length(s) do
  s[i]:=upcase(s[i]);
 writeln(s);
 readln;
end.
```

Feladat: Készítsünk programot, amely kiírja a karakterek ASCII kódját a 32-es kódtól (helyköz) a 255-ös kódig.

Megoldás: Ha tudjuk egy karakter ASCII kódját, a karaktert a **chr()** függvénnyel tudjuk kiíratni. Programunk így néz ki:

```
program Pelda18;
uses crt;
var i:integer;
begin
 clrscr;
 for i:=32 to 255 do
   begin
   write(i:6);
   write(chr(i):2);
   end;
 readln;
end.
```

Ha egy karakternek szeretnénk megtudni, hogy mi az ASCII kódja, akkor azt az **ord()** függvénnyel határozhatjuk meg. Pl. **ord('A')** értéke 65, mivel az A betű ASCII kódja 65. Az **ord()** függvény pont az ellentettje a **chr()** függvénynek. A karakterek ASCII kódját és az említett két függvényt fogjuk kihasználni olyan programok készítésénél, ahol a programunkat a billentyűzet segítségével fogjuk irányítani (pl. nyilak segítségével játékprogramnál).

ASCII kódtábla (32-255):

32=	33=!	34="	35=#	36=$	37=%	38=&	39='	40=(	41=)
42=*	43=+	44=,	45=-	46=.	47=/	48=0	49=1	50=2	51=3
52=4	53=5	54=6	55=7	56=8	57=9	58=:	59=;	60=<	61==
62=>	63=?	64=@	65=A	66=B	67=C	68=D	69=E	70=F	71=G
72=H	73=I	74=J	75=K	76=L	77=M	78=N	79=O	80=P	81=Q
82=R	83=S	84=T	85=U	86=V	87=W	88=X	89=Y	90=Z	91=[
92=\	93=]	94=^	95=_	96=`	97=a	98=b	99=c	100=d	101=e
102=f	103=g	104=h	105=i	106=j	107=k	108=l	109=m	110=n	111=o
112=p	113=q	114=r	115=s	116=t	117=u	118=v	119=w	120=x	121=y
122=z	123={	124=|	125=}	126=~	127=•	128=€	129=	130=‚	131=ƒ
132=„	133=…	134=†	135=‡	136=ˆ	137=‰	138=Š	139=‹	140=Œ	141=
142=Ž	143=	144=	145=‘	146=’	147=“	148=”	149=•	150=–	151=—
152=˜	153=™	154=š	155=›	156=œ	157=	158=ž	159=Ÿ	160=	161=¡
162=¢	163=£	164=¤	165=¥	166=¦	167=§	168=¨	169=©	170=ª	171=«
172=¬	173=	174=®	175=¯	176=°	177=±	178=²	179=³	180=´	181=µ
182=¶	183=·	184=¸	185=¹	186=º	187=»	188=¼	189=½	190=¾	191=¿
192=À	193=Á	194=Â	195=Ã	196=Ä	197=Å	198=Æ	199=Ç	200=È	201=É
202=Ê	203=Ë	204=Ì	205=Í	206=Î	207=Ï	208=Ð	209=Ñ	210=Ò	211=Ó
212=Ô	213=Õ	214=Ö	215=×	216=Ø	217=Ù	218=Ú	219=Û	220=Ü	221=Ý
222=Þ	223=ß	224=à	225=á	226=â	227=ã	228=ä	229=å	230=æ	231=ç
232=è	233=é	234=ê	235=ë	236=ì	237=í	238=î	239=ï	240=ð	241=ñ
242=ò	243=ó	244=ô	245=õ	246=ö	247=÷	248=ø	249=ù	250=ú	251=û
252=ü	253=ý	254=þ	255=ÿ						

Feladatok:

1. Készítsünk programot, amely bekér egy mondatot, majd kiírja ugyanezt a mondatot úgy, hogy mindegyik betű (karakter) után kirak egy szóközt. **(string01)**

```
Ird be a mondatot: Ez egy szep
nap.
E z   e g y   s z e p   n a p .
```

2. Kérjünk be egy keresztnevet, majd írassuk ki ezt a nevet betűnként függőlegesen lefelé a képernyőre. **(string02)**
Például ha megadjuk névnek a "**Peti**"-t, a program írja ki ezt:

P
e
t
i

```
Ird be a nevet: Zsofia
Z
s
o
f
i
a
```

3. Készítsünk programot, amely bekér egy mondatot, majd kiírja ugyanezt a mondatot fordítva. **(string03)**
Például ha beírjuk "**Szep napunk van ma.**", kiírja ezt: "**.am nav knupan pezS**"

```
Ird be a mondatot: Vagyok aki
vagyok.
.koygav ika koygaV
```

4. Olvassunk be egy **A** természetes számot és egy **CH** karaktert (**char** típusú változót - hasonlóan minta **string** típust, csak a deklarálásnál **string** helyett **char**-t írunk). Rajzoljunk ki a beolvasott karakterből egy **A** oldalú négyzetet a képernyőre (minden sorban **A** drb. karakter legyen és összesen **A** drb. sorunk legyen) egymásba ágyazott cilusok segítségével. **(string04)**
Például **A=4**-re és **CH='M'**-re rajzolja ki a program ezt:

MMMM
MMMM
MMMM
MMMM

```
Ird be a betut: A
Ird be a szamot: 6
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
```

5. Kérjünk be egy keresztnevet, és egy db számot, majd írassuk ki ezt a nevet betűnként függőlegesen lefelé a képernyőre, soronként annyiszor a betűket amennyi db számot megadtunk **(string05)**
Például ha megadjuk névnek a "**Peti**"-t, és a **db-r 4** et a program írja ki ezt:

PPPP
eeee
tttt
iiii

```
Ird be a nevet: Peti
Hanyszor: 6
PPPPPP
eeeeee
tttttt
iiiiii
```

6. Kérjél be egy 6 karakterből álló keresztnevet! Az név első felét, és a második felét külön sorba írasd ki! (pl: Zsófia) **(string06)**

```
Ird be a 6 karakter hosszu nevet:
Zsofia

________________
Zso
fia
```

7. Írasd ki a képernyőre az Angol ABC nagybetűit az ASCII kódtáblából! (az 'A' a 65.; a 'Z' a 90.)

A betűk között legyen egy szóköz távolság! (Az első két sorba írasd ki az 'A' és a 'Z' ASCII kódját a minta alapján!) **(string07)**

```
A B C D E F G H I J K L M N O P Q
R S T U V X Y Z
```

8. Alakítsd át az előző programot úgy, hogy kérjen be két számot 32-255 között, és azokat az ASCII kódokat írja ki, három karakter távolságban!

(Arra vigyázzunk. hogy az első szám kisebb legyen, mint a második szám.) **(string08)**

```
Adjal meg egy 32 es 255 kozotti
szamot: 90
Adjal meg egy elozonel nagyobb, de
255 nel kisebb szamot: 103
 Z [ \ ] ^ _ ' a b c d e f g
```

9. Készíts olyan programot, amely bekér egy keresztnevet, majd a következő sorban kiírja a névben szereplő betűk ASCII kódját, 5 karakter távolságban! **(string09)**

```
Ird be a nevet: Anaszatázia
   65  110  97  115  122  116  160
122  105   97
```

10. Készíts programot, amely kiírja egymás mellé (A=65) (B=66) (C=67) formátumban az angol ABC nagybetűit,
és alá az angol ABC kisbetűit az ASCII kódrendszerből!
(a=97) (b=98) (c=99)
(string10)

```
(A=65)(B=66)(C=67)…(X=88)(Y=89)
(Z=90)

(a=97)(b=98)(c=99)…(x=120)(y=121)
(z=122)
```

6 Az IF feltételvizsgálat

- az **IF** ... **THEN** parancs
- a feltétel **ELSE** ága

6.1 Az IF ... THEN parancs

Gyakran előfordul, hogy programunkban valamilyen feltételtől függően szeretnénk parancsot végrehajtani. Például akkor szeretnénk végrehajtani valamilyen utasítást, ha az **i** változó értéke nagyobb mint **5**. Erre általánosan a következő utasítás szolgál:

```
if  fetétel  then  parancs;
```

A mi példánkban (ha i nagyobb mint 5) ez így nézne ki:

```
if  i>5  then  parancs;
```

A feltételben bármilyen két kifejezést összehasonlíthatunk (pl. az **i+1**-et a **j+5**-tel). A kifejezések összehasonlítására használhatjuk a **<**, **>**, **=**, **<=**, **>=**, **<>** (nem egyenlő) jeleket, melyek igaz/hamis (true/false) logikai értékeket adnak vissza. Ha ez a logikai érték igaz (true), akkor a parancs végrehajtódik.

A logikai értékeket összekapcsolhatjuk az **AND** (és), **OR** (vagy), **NOT** (nem), **XOR** (kizáró vagy) műveletekkel. Így egyszerre több feltételt is vizsgálhatunk. Ilyenkor a feltételeket, amelyeket összekapcsoljuk, zárójelbe tesszük. Pl. ha azt szeretnénk, hogy a parancsunk csak akkor hajtódjon végre, ha az **i** változó értéke nagyobb mint **5** és egyidejűleg a **k** változó értéke egyenlő **7**-tel:

```
if  (i>5)  and  (k=7)  then  parancs;
```

Természetesen itt is, hasonlóan mint a **for** ciklusnál egyszerre több parancsot is végrehajthatunk a feltételen belül. Ekkor a parancsokat a **begin ... end;** közé rakjuk:

```
if  feltétel  then begin
                        parancs1;
                        parancs2;
                        ...
                        parancsN;
                      end;
```

Feladat: Készítsünk programot, amely bekér három pozitív egész számot, és kiírja őket nagyság szerint növekvő sorrendben!

Megoldás: A programban a három számnak 3 változót fogunk használni (**a**, **b**, **c**). Ezeket cserélgetéssel sorba fogjuk rakni úgy, hogy az **a** változóban legyen a legkisebb szám, a **b**-ben a középső és a **c**-ben a legnagyobb szám. A cserékhez egy **x** segédváltozót használunk.

```
program Pelda19;
uses crt;
var a,b,c,x:integer;
begin
 clrscr;
 write('Kerem az elso szamot: ');
 readln(a);
 write('Kerem a masodik szamot: ');
 readln(b);
 write('Kerem a harmadik szamot: ');
 readln(c);
 if a>b then begin
             x:=a;
             a:=b;
             b:=x;
             end;
 if a>c then begin
             x:=a;
             a:=c;
             c:=x;
             end;
 if b>c then begin
             x:=b;
             b:=c;
             c:=x;
             end;
 write('A harom szam novekvo sorrendben: ');
 writeln(a,' < ',b,' < ',c);
 readln;
end.
```

Feladat: Készítsünk programot, amely bekér egy mondatot, majd kiírja, hogy hány nagybetű van benne.

Megoldás: Egy karakterről úgy tudjuk eldönteni, hogy nagybetű-e, hogy megnézzük nagyobb vagy egyenlő-e mint 'A' és egyben kisebb vagy egyenlő-e mint 'Z'. Ha ez igaz, akkor a karakter A-tól Z-ig van, tehát egy nagybetű. Fogunk használni még egy **nagy** nevű változót, amely a nagybetűk számát fogja számolni.

```
program Pelda20;
uses crt;
var s:string;
    i,nagy:integer;
begin
 clrscr;
 write('Irj be egy mondatot: ');
 readln(s);
 for i:=1 to length(s) do
   if (s[i]>='A') and (s[i]<='Z') then inc(nagy);
 writeln('A mondatban ',nagy,' darab nagy betu van.');
 readln;
end.
```

A programunkban használtunk egy új függvényt, az **inc()** függvényt (increase). Ez a függvény növeli a változó értékét 1-gyel, tehát az **inc(nagy)** helyett írhattuk volna ezt is: **nagy:=nagy+1**.

Hasonlóan létezik egy függvény, amely csökkenti a változó értékét eggyel, ez a **dec()** függvény (decrease).

6.2 A feltétel ELSE ága

A programozás során sokszor előfordul, hogy akkor is végre akarunk hajtani valamilyen utasítást, ha a feltétel nem igaz. Ebben az esetben az **if ... then** parancsunk bővülni fog egy **else** résszel:

```
if  feltétel  then  parancs1
                 else  parancs2;
```

Ha a feltétel igaz, akkor a *parancs1* hajtódik végre, egyébként a *parancs2* hajtódik végre. Fontos megjegyeznünk, hogy ha az **if** feltételvizsgálatnak van **else** ága, akkor a *parancs1* után nem rakunk pontosvesszőt. Természetesen itt is mindkét parancs helyett tehetünk több egymást követő utasítást is. Ilyenkor minden esetben a **begin ... end** kulcsszavakkal kapcsoljuk össze az egymást követő parancsokat:

```
if  feltétel  then begin
                          parancs1i;
                          parancs2i;
                          ...
                          parancsNi;
                         end
                else begin
                          parancs1h;
                          parancs2h;
                          ...
                          parancsNh;
                         end;
```

Feladat: Készítsünk programot, amely bekér két egész számot, majd kiírja szavakkal, hogy az első szám kisebb, nagyobb, vagy egyenlő mint a második szám.

Megoldás: A két számot összehasonlítjuk, ha az első szám kisebb mint a második, akkor ezt kiírjuk. Ha ez nem igaz (else), akkor a két számot újra összehasonlítjuk, ha az első szám a nagyobb, akkor kiírjuk, hogy az első szám a nagyobb. Ha ez sem igaz (belső feltétel else ága) akkor pedig kiírjuk, hogy a két szám egyenlő.

```
program Pelda21;
uses crt;
var a,b:integer;
begin
 clrscr;
 write('Kerem az elso szamot: ');
 readln(a);
 write('Kerem a masodik szamot: ');
 readln(b);
 if a<b then writeln('Az 1. szam a kisebb.')
        else if a>b then writeln('Az 1. szam a nagyobb.')
                    else writeln('A ket szam egyenlo.');
 readln;
end.
```

Feladatok:

1. Kérjünk be egy mondatot. Számoljuk meg és írassuk ki, hogy hány szóköz van benne. **(if01)**

```
Ird be a mondatot: A mondatan van
szokoz.
A mondatban 3 db szokoz van.
```

2. Kérjünk be egy mondatot, majd írassuk ki ugyanezt a mondatot szóközök nélkül. **(if02)**

```
Ird be a mondatot: Vagyok aki vagyok.
Vagyokakivagyok.
```

3. Kérjünk be **N** darab természetes számot (először **N**-t kérjük be). Az adatok beírása után a program írja ki a páros és páratlan számok darabszámát, és a páratlan számok összegét! **(if03)**

```
Add meg az n-t: 6
Add meg a(z) 1. szamot: 12
Add meg a(z) 2. szamot: 67
Add meg a(z) 3. szamot: 89
Add meg a(z) 4. szamot: 34
Add meg a(z) 5. szamot: 21
Add meg a(z) 6. szamot: 55
A szamok kozott 2 db paros szam van.
A szamok kozott 4 db paratlan szam
van.
A paratlan szamok osszege: 232
```

4. A húsvét vasárnap dátumát a níceai zsinat a következőképpen határozta meg: a tavaszi napéjegyenlőséget követő első holdtölte utáni első vasárnap. A dátum március 22-e és április 25-e között változhat. A dátum meghatározására alkalmas a következő algoritmus! Jelölje **T** az évszámot (**1800**<=**T**<=**2099**). Kiszámítjuk a következő osztási maradékokat:

A = T / 19 maradéka
B = T / 4 maradéka
C = T / 7 maradéka
D = (19 . A + 24) / 30 maradéka
E = (2 . B + 4 . C + 6 . D + 5) / 7 maradéka

Ezekből a húsvét vasárnap dátuma:

H = 22 + D + E, ami márciusi dátum, ha **H <= 31**, különben áprilisban **H – 31** -e.

Két kivétel van:
- ha **E = 6** és **D = 29**, akkor **H = 50**,
- ha **E = 6** és **D = 28** és **A > 10**, akkor **H = 49**.

Készítsünk programot, ami bekér egy évszámot, és meghatározza, majd kiírja a húsvét vasárnap dátumát! **(if04)**

```
Adjal meg egy 1900 es 2099 kozotti
evszamot: 2010

A husvet a 2010 -es evben aprilis 4-ra
(-re) esik.
```

5. Határozzuk meg és írassuk ki az összes hárommal és öttel egyaránt osztható, 1000-nél kisebb természetes számot. **(if05)**

```
  15  30  45  60  75  90
…
960 975 990
```

6. Készítsünk programot, amely beolvas egy **N** természetes számot, majd billentyűzetről bekér **N** drb. természetes számot és a beolvasás után kiírja melyik ezek közül a számok közül a legkisebb. (Ehhez vezessünk be egy **min** nevű változót, melyet mindegyik szám beolvasása után összehasonlítunk a számmal, és ha a szám kisebb, akkor megjegyezzük ebben a változóban. A **min** változót a program elején állítsuk be a lehető legnagyobb számra, aminél biztos hogy mindegyik szám kisebb - pl. 32768, vagy ez helyett a beállítás helyett az első számot olvassuk be (állítsuk be) a **min** változóba.)
(if06)

```
Add meg az n-t: 6
Add meg a(z) 1. szamot: 12
Add meg a(z) 2. szamot: 67
Add meg a(z) 3. szamot: 89
Add meg a(z) 4. szamot: 900
Add meg a(z) 5. szamot: 7
Add meg a(z) 6. szamot: 55

A legkisebb szam: 7
```

7. Egészítsük ki az előző programunkat úgy, hogy a beolvasás után a számok közül ne csak a legkisebbet, de a legnagyobbat is írja ki. (Ehhez vezessünk be egy **max** nevű változót, melyet mindegyik szám beolvasása után összehasonlítunk a számmal, és ha a szám nagyobb, akkor megjegyezzük ebben a változóban. A **max** változót a program elején állítsuk be a lehető legkisebb számra, aminél biztos hogy mindegyik szám nagyobb - pl. – 32767, vagy ez helyett a beállítás helyett az első számot olvassuk be (állítsuk be) a **max** változóba.)
(if07)

```
Add meg az n-t: 6
Add meg a(z) 1. szamot: 12
Add meg a(z) 2. szamot: 67
Add meg a(z) 3. szamot: 89
Add meg a(z) 4. szamot: 900
Add meg a(z) 5. szamot: 7
Add meg a(z) 6. szamot: 55

A legkisebb szam: 7
A legnagyobb szam: 900
```

8. A program döntse el, hogy a bekért **a**, **b**, **c** természetes számok lehetnek-e egy derékszögű háromszög oldalhosszúságai. Az **a** és **b** legyen a két befogó (használjuk Pitagorasz-tételét).
(if08)

```
Add meg az a befogot: 6
Add meg az a befogot: 8
Add meg az a atfogot: 10

A haromszog derekszogu!
```

```
Add meg az a befogot: 12
Add meg az a befogot: 5
Add meg az a atfogot: 6

A haromszog nem derekszogu!
```

9. 3 bekért szakaszhosszból döntsük el, hogy szerkeszthető-e háromszög belőlük! **(if09)**

```
Kerem az „a” oldalt: 21
Kerem az „b” oldalt: 33
Kerem az „c” oldalt: 45

Lehet haromszog!
```

10. Egy akváriumot készítünk. A halaknak 50000 cm3 vízre van szükség. A kérdés az, hogy megfelelnek-e az adatok? **(if10)**

```
Kerem az „a” oldalt: 50
Kerem az „b” oldalt: 30
Kerem az „c” oldalt: 45
Térfogat: 67500 cm3
Megfelel!
```

7 A CASE elágazás

- a **CASE** elágazás

7.1 A CASE elágazás

Az **if** mellett a pascal nyelv másik fajta elágazásra szolgáló utasítása a **case**. Ez egy változó értékétől függően hajtja végre valamelyik parancsot. Formája:

```
case változó of
  érték1 : paracs1 ;
  érték2 : paracs2 ;
  ...
  értékN : paracsN ;
end;
```

Ebben az utasításban csak olyan típusú változót használhatunk, melynek a következő és előző értéke egyértelműen meghatározható. Pl. használhatunk **integer**, **byte**, **char** típusokat (mivel az egész számoknál - byte, integer - egyértelműen meghatározható, hogy pl. a 8 után a 9 következik, a karaktereknél - char - pedig szintén meghatározható, hogy a C karakter után a D következik). Nem használhatunk azonban **real**, **string** típusokat (mert a real típusnál az 1.2 után mi következik? 1.3? 1.21? 1.201? és hasonlóan a string típusnál az 'iskola' szó után milyen szó következik?)

A **case** utasítás így működik: Ha a változó értéke az *érték1*-gyel egyenlő, akkor a *parancs1*-et hajtja végre. Ha az *érték2*-vel egyenlő, akkor a *parancs2*-t hajtja végre. Ha az *érték3*-mal egyenlő, akkor a *parancs3*-at hajtja végre, stb. A **case** parancsot is az **if**-hez hasonlóan kiegészíthetjük egy olyan **else** résszel, amelyet akkor hajt végre a program, ha a *változó* értéke a felsorolt *értékek* egyikével sem egyenlő. Ekkor a **case** utasításunk formája így bővül ki:

```
case változó of
  érték1 : paracs1 ;
  érték2 : paracs2 ;
  ...
  értékN : paracsN ;
else
  parancs ;
end;
```

Itt is ha bármelyik parancs helyett több egymást követő parancsot szeretnénk a géppel végrehajtatni, használnunk kell a **begin..end;** kulcsszavakat. A **case** parancsot a gyakorlatban a következő példa szemlélteti:

```
program Pelda22;
uses crt;
var c:char;
begin
clrscr;
 writeln('Mit visz a kis hajo?');
 write('Irj be egy kis betut: ');
 readln(c);
 case c of
  'a': writeln('Almat!');
  'b': writeln('Banant!);
  'c': begin
       writeln('Cernat,');
       writeln('cicat!');
       end;
  'd': writeln('Datolyat!');
 else
  writeln('Erre nem tudok mit lepni.');
 end;
 readln;
end.
```

Feladatok:

1. Készítsünk programot, amely beolvas egy egész számot (**N**), majd kiírja szavakkal, hogy a hét N-dik napja milyen nap (hétfő, kedd, szerda, ...)
Ha olyan számot adunk meg, ami kisebb mint 1 vagy nagyobb mint 7, akkor Írja ki, hogy „Nincs ilyen nap! Egy hét csak 7 napból áll!)” **(case01)**

```
A het hanyadik napjat irjam ki?
Adj meg egy szamot: 2
_____________________________
A het 2. napja kedd!
```

2. Készíts egy programot, ami bekér az angol ABC első 5 betűje közül egyet, Majd azzal a kezdőbetűvel írjon ki egy gyümölcsnevet. (pl.: a-> alma; b-> banán)
Ha nem ebből az 5 betű közül add meg egyet a felhasználó, akkor írja ki, hogy „Nem jó betű!” **(case02)**

```
Add meg a kezdobetut: c
_____________________________
citrom
```

3. Készíts programot, melyben bekér két egész számot, és egy műveleti jelet. A műveleti jeltől függően adjon össze, vonjon ki, szorozzon, vagy osszon! Vigyázz a típusokra és formázásra!
Pl.: 1. szám: 6
2. szám: 2
A műveleti jel: - + * /
6-2=4 6+2=8 6*2=12 6/2=3
Ha nem megfelelő a karakter, akkor írja ki, hogy
„Nem megfelelő formátum!” **(case03)**

```
Add meg az elso szamot: 8
Add meg a masodik szamot: 4
A muveleti jel: -
_____________________
8-4=4
```

4. Készítsünk programot, amely beolvas egy egész számot (**N**), majd kiír egy sorba annyi „*” karaktert a képernyőre, amennyi a szám! (alkalmazz case-t; ne for-t)
A szám 1és 10 között legyen! Ha kisebb, vagy nagyobb, mint az előzőekben megadott, akkor írja ki, hogy „Csak 1 és 10 közötti szám lehet!’ **(case04)**

```
Adjal meg egy szamot: 8
_____________________________
********
```

5. Készítsünk programot, amely bekér egy egész számot (1-től 100-ig), majd kiírja az adott számot szavakkal.

A szó kiírásához előbb nézzük meg hogy a szám tízzel osztható-e, ha igen, akkor írjuk ki **case** segítségével: *tíz, húsz, harminc, stb.*

Ha a szám nem osztható tízzel, nézzük meg mi áll a tízesek helyén a számban (**div** fügvénnyel) és ez szerint előbb írjuk ki egy **case** segítségével hogy: *tizen, huszon, harminc, stb.* (ha nulla van a tízesek helyén akkor semmit ne írjunk ki), majd nézzük meg hogy mi áll az egyesek helyén (**mod** fügvénnyel) és ez alapján írjuk ki mellé egy másik **case** segítségével hogy: *egy, kettő, három, stb.* **(case05)**

```
Add meg a szamot: 89
_________________________
Szoveggel: nyolcvankilenc
```

8 A WHILE..DO ciklus

- a **WHILE** ... **DO** ciklus

8.1 A WHILE ... DO (elöltesztelős ciklus)

Ez a ciklus a **for** ciklushoz hasonlóan megismétel néhány parancsot többször egymás után. A különbség abban van, hogy míg a **for** ciklusnál a ciklusváltozó kezdő- és végértéke határozta meg az ismétlések számát (pl. for i:=1 to 8 ciklusnál az adott parancsot nyolcszor ismételte meg), a **while..do** ciklusnál az ismétlések számát nem egy ciklusváltozó, hanem egy feltétel határozza meg (pl. a<b). Amíg a feltétel igaz, addig az adott parancsokat a **while..do** ciklus ismételni fogja. A ciklus formája:

```
while fetétel  do parancs;
```

vagy ha több parancsot szeretnénk elvégezni a cikluson belül, akkor hasonlóan mint a **for** ciklusnál és az **if** feltételvizsgálatnál, használjuk a **begin..end** utasításokat:

```
while feltétel do  begin
                     parancs1;
                     parancs2;
                     ...
                     parancsN;
                   end;
```

Magyarul: amíg a feltétel igaz, ismételd a ciklusban levő parancsokat.

A feltétel itt is hasonlóan mint az **if** feltételvizsgálatnál lehet egy vagy több kifejezésből is, melyeknél használhatjuk az **AND** (és), **OR** (vagy), **NOT** (nem), **XOR** (kizáró vagy) műveleteket.

A ciklus a következő képen működik: a számítógép megnézi, hogy a feltétel igaz-e.

- Ha nem igaz, akkor a ciklusban levő parancsokat nem hajtja végre egyszer sem, hanem a program folytatódik a ciklus utáni utasításokkal.
- Ha a feltétel igaz, végrehajtja a ciklusban levő parancsokat. Ismét megvizsgálja a feltételt, amely ha még mindig igaz, akkor ismét végrehajtja a ciklusban levő parancsokat. Majd ismét megvizsgálja a feltételt, amely ha meg mindig igaz, akkor ismét végrehajtja a parancsokat... Ha a feltétel már nem igaz, akkor a parancsokat nem hajtja végre, hanem folytatódik a program futása a ciklus utáni utasításokkal.

Feladat: Készítsünk programot az **n!** kiszámítására a **for** ciklus használata nélkül (a **while** ciklus segítségével).

Megoldás: A programunk a következő képen néz ki:

```
program Pelda23;
uses crt;
var n,szorzat:integer;
begin
clrscr;
  write('Add meg az N erteket: ');
  readln(n);
  szorzat:=1;
  while n>1 do begin
                szorzat:=szorzat*n;
                dec(n); { csokkenti n erteket 1-gyel,
                irhattuk volna helyette n:=n-1-et is }
               end;
  writeln('N! = ',szorzat);
readln;
end.
```

Ha programon belül megjegyzést szeretnénk hozzáfűzni valamihez, akkor { } jelek között tehetjük meg.

Feladatok:

1. Készítsünk programot, amely beolvas egy egész számot, majd elosztja 2-vel annyiszor, ahányszor lehet és közben felírja a számot a kettes számok szorzataként megszorozva egy olyan számmal, amely már nem osztható 2-vel.

Ha a szám egyszer sem osztható kettővel, akkor:

Ahhoz, hogy a szám osztható-e kettővel használjuk a **mod** függvényt. Pl. X akkor osztható 2-vel, ha **X mod 2 = 0**. A szám elosztását a **div** függvénnyel végezzük el.
(while01)

```
Kerek egy egesz szamot: 120
120 = 2*2*2*15
```

```
Kerek egy egesz szamot: 17
17 = 17
```

2. Készítsünk programot, amely bekér egy egész számot, majd mindaddig kér be további egész számokat, amíg nem adjuk meg a 0-t. A program határozza meg és írja ki a beadott egész számok közül a legnagyobbat.
(while02)

```
Kerek egy szamot: 23
Kerek egy szamot: 56
Kerek egy szamot: 16
Kerek egy szamot: 32
Kerek egy szmot: 0
--------------------------------
A legnagyobb szam: 56
```

3. Készítsünk programot, amely bekér egy n egész számot, majd kiíratjuk az első „n” szám összegét. (while ciklussal)
(while03)

```
Adj meg egy szamot: 5
--------------------------------
Összeg: 15
```

4. Készítsünk programot, amely bekér két számot 0 és 20 között, majd írasson ki annyi „X”-et, amennyi a két szám különbsége. (while)
(while04)

```
Kerem az elso szamot: 17
Kerem a masodik szamot: 13
--------------------------------
XXXX
```

```
Kerem az elso szamot: 23
A szam nem 0 és 20 kozotti!
Kerem az elso szamot:
```

5. Készítsünk programot, amellyel bekérünk egy „n” számot. Majd kiíratjuk az „n” szám szorzatait 100 –ig. (Az „n” egy 0 és 10 közötti szám)
(while05)

```
Kerem az „n” szamot: 8
--------------------------------
8; 16;24;32;40;48;56;64;72;80;88;96;
```

```
Kerem az „n” szamot: 23
A szam nem 0 és 10 kozotti!
Kerem ujra az „n” szamot!
```

9 A REPEAT..UNTIL ciklus

- a **REPEAT** ... **UNTIL** ciklus
- véletlen számok generálása

9.1 A REPEAT ... UNTIL (hátultesztelős ciklus)

A **while..do** ciklusnál a számítógép először megvizsgálta a feltételt és csak utána hajtotta végre a ciklusban levő parancsokat (amennyiben a feltétel igaz volt). Ha a feltétel mindjárt az első vizsgálatnál hamis volt, akkor a ciklusban levő utasításokat egyszer sem hajtotta végre.

A **repeat..until** ciklusnál a számítógép először végrehajtja a ciklusban levő parancsokat (repeat..until közötti részt), majd utána vizsgálja meg a feltételt. Ha a feltétel igaz, kilép a ciklusból. Ha a feltétel hamis, megismétli ismét a ciklusban levő parancsokat majd ismét megvizsgálja a feltételt. Ennél a ciklusnál tehát egyszer mindenképpen lefutnak a ciklusban levő parancsok.

A **repeat..until** ciklusból a számítógép a **while..do** ciklussal ellentétben akkor lép ki, ha a feltétel igaz (a **while..do** ciklusnál akkor lépett ki, ha a feltétel hamis volt).

A ciklus szerkezete:

```
repeat
    parancs1;
    parancs2;
    ...
    parancsN;
until feltétel;
```

Magyarul: ismételd a ciklusban levő parancsokat, amíg a feltétel nem lesz igaz (tehát amíg a feltétel hamis).

9.2 Véletlen számok generálása

A **random(n)** funkció egy véletlen számot ad vissza a <0,n-1> intervallumból.

Például:.

a := random(7); kigenerál egy egész számot, melynek értéke 0-tól 6-ig lehet
a := 3 + random(10); kigenerál egy egész számot, melynek értéke 3-tól 12-ig lehet

A véletlenszám generátor a program indítása után mindig ugyanonnan kezdi a számok generálását. A programunk elején ezért mindig inicializáljuk a **randomize;** paranccsal (ezt a parancsot úgy foghatjuk fel, mintha megkeverné a számokat). A **randomize;** parancsot elég csak egyszer, a program legelején megadnunk, mielőtt használjuk a **random()** funkciót.

Feladat: A számítógép véletlenszerűen válasszon (gondoljon) egy számot 1-től 5-ig. Kérdezze meg a felhasználótól melyik ez a szám. A felhasználó addig találgathat, amíg nem találja el ezt a számot. A számítógép csak annyit írjon neki ki: "eltaláltad, ez az a szám", vagy "sajnos nem ez az a szám".

Megoldás: A programunk a következő képen néz ki:

```
program Pelda24;
uses crt;
var a,b:integer;
begin
 clrscr;
  randomize; { megkeveri a szamokat }
  a:=random(5)+1; { kigeneral egy egesz szamot 1-tol 5-ig }
  repeat
    write('Melyik szamra gondoltam? ');
    readln(b);
    if a=b then writeln('Eltalaltad, ez az a szam!')
           else writeln('Sajnos nem ez az a szam.');
  until a=b;
  writeln('Nyomd meg az ENTER billentyut.');
  readln; { a szamitogep var az ENTER megnyomasara }
 readln;
end.
```

Feladatok:

1. Írasd ki *Repeat* ciklussal 100-13-ig a páros számokat! (5 karakter távolsággal) **(repeat01)**

```
  100   98    96    94    92
  90    88    86    84    82
  80    78    76    74    72
  70    68    66    64    62
  60    58    56    54    52
  50    48    46    44    42
  40    38    36    34    32
  30    28    26    24    22
  20    18    16    14
```

2. Készítsünk programot, amely bekér egész számokat mindaddig, amíg nem adjuk meg a 0-t. A program határozza meg és írja ki a beadott egész számok közül a legkisebbet és a legnagyobbat. (A 0-t ne számítsa bele a beadott számokba, ez csak a bevitel végét jelzi.) A számok beolvasását a 0 végjelig repeat .. until ciklus segítségével valósítsuk meg! **(repeat02)**

```
Adja meg a szamot: 12
Adja meg a szamot: 33
Adja meg a szamot: 7
Adja meg a szamot: 48
Adja meg a szamot: 0
---------------------------------
A legkisebb szam: 7
A legnagyobb szam: 48
```

3. Készítsünk játékprogramot, amely gondol egy számot 1 és 50 között. A felhasználó addig találgathat, amíg nem találja el a keresett számot. A számítógép minden rossz tipp után írja ki, hogy a gondolt szám nagyobb vagy kisebb. **(repeat03)**

```
Gondoltam egy szamra 1 es 50
kozott! Tipp: 25
Kisebb
Tipp: 12
Nagyobb
Tipp: 18
Eltalaltad!!! Gratulalok!
```

4. Egészítsük ki az előző programunkat úgy, hogy a játékos csak maximum 7-szer tippelhessen. Ha a hetedik tippre sem találja el a gondolt számot, a program írja ki a gondolt számot majd fejeződjön be. **(repeat04)**

```
...
7. tipp: 33
Nem talaltad el! A gondolt szam:
32.
```

5. Olvassunk be pozitív egész számokat 0 végjelig. Írjuk ki a számok átlagát. (A 0-t ne számítsa bele a beadott számokba, ez csak a bevitel végét jelzi.) **(repeat05)**

```
Adja meg a szamot: 12
Adja meg a szamot: 33
…
Adja meg a szamot: 0
A szamok atlaga: 7,321
```

6. Készítsünk programot, amely ki fogja kérdezni a matematikát (két szám összeadását, az <1,10> intervallumból). A két számot a számítógép véletlenszerűen válassza ki. A program akkor fejeződjön be, ha a felhasználó 10 példát kiszámolt helyesen. Rossz válasz esetén kérdezze újra ugyanazt a példát. A program végén írjuk ki az eredményességet százalékokban. **(repeat06)**

```
Szamolj!
1. 2+5=7
2. 8+5=13
3. 4+5=10
3. 4+5=9
...
10. 3+9=12
Eredmeny: 10-bol 9 jo megoldas
```

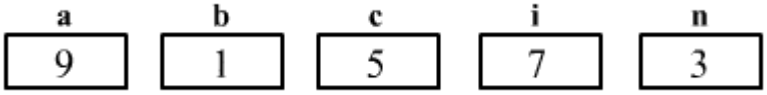
10 Tömbök (array of ...)

- tömbök (**array of ...**)
- konstansok használata
- tömb elemeinek generálása

10.1 Tömbök (array of ...)

Eddigi programjainkban egyedi változókkal dolgoztunk, mindegyik változónak külön neve volt. Pl

```
var a,b,c,i,n:integer;
```



Sok feladatot lehetetlen egyedi változókkal megoldanunk. Gondoljunk például egy névsor tárolására, melyben akár 100-200 név is lehet.

Ehhez szükségünk van olyan adattípusra, melyben több adatot tárolhatunk. Az eddig megismert típusok **elemi típusok** voltak. Ezekből az elemi típusokból adatszerkezeteket, **összetett típusok**at építhetünk. Ilyen összetett típus a tömb is.

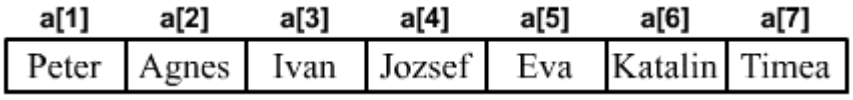
A **tömb** típusát a következő képpen kell megadnunk:

```
var tömb_neve : array [ indexhatárok ] of alaptípus ;
```

Az *indexhatárok* megadják a tömb méretét (hogy hány elemet tárolhatunk benne), az *alaptípus* pedig az elemek típusát (ez lehet pl. **integer**, **string**, **char**, **boolean**, **byte**, stb).

Például:

```
var a:array [1..7] of string;
```



Ez létrehoz egy 7-elemű, **string**ekből álló tömböt. Ebben tárolhatunk például egy 7 elemből álló névsort. A tömb egyes elemeit az elemek indexén keresztül érhetjük el úgy, hogy a tömb neve után szögletes zárójelben megadjuk az elem **index**ét. A tömbünk tehát a következő elemekből áll: **a[1]**, **a[2]**, ..., **a[7]**.

A tömb beolvasása történhet például egy egyszerű ciklussal:

```
for i:=1 to 7 do begin
                 write('Add meg a(z) ',i,'. nevet: ');
                 readln(a[i]);
                 end;
```

Hasonlóan történhet a tömb kiírása, feldolgozása is ciklusok segítségével.

Feladat: Készítsünk programot, amely beolvas 10 egész számot egy tömbbe, majd kiírja őket fordított sorrendben.

Megoldás: A programunk a következő képen néz ki:

```
program Pelda25a;
uses crt;
var a:array[1..10] of integer;
    i:integer;
begin
 clrscr;
  for i:=1 to 10 do begin
                    write(i,'. szam: ');
                    readln(a[i]);
                    end;
  writeln('A szamok forditott sorrendben:');
  for i:=10 downto 1 do write(a[i]:6);
  writeln;
readln;
end.
```

10.2 Konstansok használata

Programunkban használhatunk konstansokat is. Ezeket a **var** előtt a **const** utasítással adhatjuk meg:

```
const konstansneve = értéke ;
```

Például ha az előző programban 10 elem helyett 20-at szeretnénk beolvasni majd kiírni, több helyen meg kéne változtatnunk a programunkat - a 10-et átírni 20-ra. Ha program elején bevezetünk egy konstanst (állandót) a tömb elemeinek számára és a programban mindenhol ezt használjuk, akkor később ha meg akarjuk változtatni a programban a beolvasandó elemek számát elég ennek a konstansnak az értékét megváltoztatni a programunk elején. Az előző programunk például így is nézhetett volna ki:

```
program Pelda25b;
uses crt;
const tombmerete=10;
var a:array[1..tombmerete] of integer;
    i:integer;
begin
 clrscr;
  for i:=1 to tombmerete do begin
                            write(i,'. szam: ');
                            readln(a[i]);
                            end;
  writeln('A szamok forditott sorrendben:');
  for i:=tombmerete downto 1 do write(a[i]:6);
 readln;
end.
```

10.3 Tömb elemeinek generálása

Készítsünk programot, melyben egy tömb elemeit véletlenszerűen kigenerálja majd kiírja a képernyőre. A tömb elemeinek értéke 1 és 100 közötti szám legyen.

Ehhez a már régebben megismert **random** függvényt fogjuk használni. Mivel egy tömbről van szó, a generálást ciklus segítségével fogjuk elvégezni, hasonlóan ahogy az elemek kiírását is a képernyőre. Programunk így néz ki:

```
program Pelda26;
uses crt;
const n=20; {a tömb elemeinek a száma}
var a: array[1..n] of integer;
    i: integer;
begin
 clrscr;
 randomize;
 for i:=1 to n do a[i]:=random(100)+1;
 for i:=1 to n do write(a[i],', ');
 readln;
end.
```

Feladatok:

1. Készíts programot, melyben létrehozol egy 5 elemből álló tömböt, amelybe keresztneveket veszel fel, majd egymás mellé kiíratod pontosvesszővel elválasztva! **(array01)**

```
Add meg az 1. nevet: Eva
Add meg az 2. nevet: Cecil
Add meg az 3. nevet: Janos
Add meg az 4. nevet: Sandor
Add meg az 5. nevet: Endre
--------------------------------
Eva; Cecil; Janos; Sandor; Endre
```

2. Készíts programot, melyen karakterekből álló tömböt hozol létre! Az elején konstansban add meg, hogy 6 karaktert keljen megadni. Majd tükörszerűen írasd ki egymás mellé kétszer a következő sorban a minta alapján! **(array02)**

```
Add meg az 1. karaktert: A
Add meg az 2. karaktert: B
Add meg az 3. karaktert: C
Add meg az 4. karaktert: D
Add meg az 5. karaktert: E
Add meg az 6. karaktert: F
--------------------------------
ABCDEF|FEDCBA
```

3. Készíts programot a minta alapján, mellyel ötöslottó sorsolást generálsz! (tömb használatával) **(array03)**

```
A(z) 1. szam: 21
A(z) 2. szam: 35
A(z) 3. szam: 90
A(z) 4. szam: 13
A(z) 5. szam: 54
```

4. Olvassunk be egész számokat egy hét elemű tömbe! Utána irassuk ki a tömb elemeit sorban, majd visszafelé, aztán írassuk ki a legkisebb elemet és a legnagyobbat. Alá pedig a tömb elemeinek az átlagát!
(array04)

```
Add meg a szamot: 22
Add meg a szamot: 43
Add meg a szamot: 67
Add meg a szamot: 44
Add meg a szamot: 69
Add meg a szamot: 13
Add meg a szamot: 35
----------------------------
22;43;67;44;69;13;35;35;13;69;44;67;43;22
A legkisebb: 13
A legagyobb:69
Atlag: 41,85
```

5. Állítsunk elő egy 30 elemű tömböt véletlen egész számokból (0-tól 99-ig). Írjuk ki a tömböt a képernyőre.
(array05)

```
22;34;68;97;41;12;2;78;33;;52;37;97;
23;45;67;89;12;32;4;21;54;72;48;36;
61;28;31;54;67;82
```

11 Műveletek tömbökkel

- legkisebb, legnagyobb elem megkeresése
- tömb elemeinek összeadása
- tömb tükrözése
- tömbök rendezése
  - egyszerű cserés rendezés
  - minimumkiválasztásos rendezés
- két rendezett tömb összefésülése

11.1 Legkisebb, legnagyobb elem megkeresése

Készítsünk programot, amely egy 20 elemű tömbbe kigenerál 1 és 150 közötti véletlen számokat, majd az elemek kiírása után megkeresi a tömbben található legkisebb és legnagyobb elemet.

Ehhez bevezetünk két változót: **min** és **max**. Kezdetben beállítjuk mindkét változó értékét a tömb első elemének értékére - feltételezve hogy ez a legkisebb és legnagyobb elem is a tömbben, majd a második elemtől a tömb végéig végignézzük az elemeket (van-e a beállított első elemnél nagyobb vagy kisebb). Ha bármelyik elem kisebb mint a **min** változó értéke, akkor az új elem értékét megjegyezzük a **min** változóban. Hasonlóan, ha olyan elemet találunk, amely nagyobb mint a **max** értéke, akkor azt megjegyezzük a **max** változóban. Így a ciklus lefutása (tömb elemeinek átnézése) után a **min** és a **max** változók a tömb legkisebb ill. a legnagyobb elemét fogja tartalmazni. Programunk így néz ki:

```
program Pelda27a;
uses crt;
const n=20; {a tömb elemeinek a száma}
var a: array[1..n] of integer;
    i,min,max: integer;
begin
 clrscr;
 randomize;
 for i:=1 to n do a[i]:=random(150)+1;
 for i:=1 to n do write(a[i],', ');
 writeln;
 {a tömb legkisebb és legnagyobb elemének keresése...}
 min:=a[1];
 max:=a[1];
 for i:=2 to n do begin
                  if a[i]<min then min:=a[i];
                  if a[i]>max then max:=a[i];
                  end;
 writeln('Legkisebb: ',min);
 writeln('Legnagyobb: ',max);
readln;
end.
```

11.2 Tömb elemeinek összeadása

Bővítsük tovább az előző programunkat. Egészítsük ki egy olyan programrésszel, amely összeadja a tömb elemeit, majd kiírja az összeget.

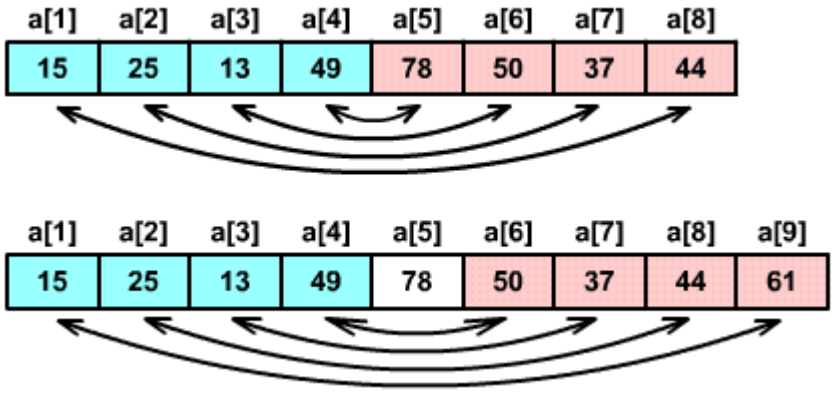
Ehhez bevezetünk egy újabb változót - **osszeg**, melynek kezdeti értékét beállítjuk 0-ra, majd a ciklus segítségével (amit módosítottunk, hogy 1-től menjen) végigmegyünk a tömb elemein és ehhez a változóhoz sorban hozzáadjuk a tömb első, második, ... utolsó elemét. Így a ciklus lefutása után az **osszeg** változó a tömb eleminek összegét fogja tartalmazni. Módosított programunk így néz ki:

```
program Pelda27b;
uses crt;
const n=20; {a tömb elemeinek a száma}
var a: array[1..n] of integer;
    i,min,max,osszeg: integer;
begin
clrscr;
 randomize;
 for i:=1 to n do a[i]:=random(100)+1;
 for i:=1 to n do write(a[i],', ');
 writeln;
 {a tömb legkisebb és legnagyobb elemének keresése }
 {az összeadással kibővítve... }
 min:=a[1];
 max:=a[1];
 osszeg:=0;
 for i:=1 to n do begin
                  if a[i]<min then min:=a[i];
                  if a[i]>max then max:=a[i];
                  osszeg:=osszeg+a[i];
                  end;
 writeln('Legkisebb: ',min);
 writeln('Legnagyobb: ',max);
 writeln('Az elemek osszege: ',osszeg);
 readln;
end.
```

11.3 Tömb tükrözése

Most készítsünk egy újabb programot. A program elején egy N=20 elemű tömbbe (N egy konstans legyen) generáljunk 10 és 99 közötti véletlen számokat, majd írjuk ki a tömböt. Ezek után tükrözzük a tömböt, tehát az első elemet cseréljük ki az utolsóval, a másodikat az utolsó előttivel, stb. Az így módosított tömböt írjuk ki újból a képernyőre.

A tömb elemeinek cseréjét (a tükrözést) egy 8 és egy 9 elemű tömbön az alábbi ábra szemlélteni:



Ebben is láthatjuk, hogy a tükrözéshez valójában egy olyan ciklusra van szükségünk, amely páros számú elemek esetében egytől a tömb feléig, páratlan számú elemek esetében a felénél eggyel kevesebbig megy (az ábrán a kék elemek) és elvégzik a megfelelő cserét. Ezt egyszerűen egy olyan for ciklussal tudjuk elvégezni, melynek ciklusváltozója **1**-től **(n div 2)**-ig megy.

A cikluson belül valójában melyik elemeket kell cserélni melyikkel? Az **1.** elemet az **n**-dik elemmel, a **2.** elemet az **(n-1)**-dik elemmel, a **3.** elemet az **(n-2)**-dik elemmel, stb. Tehát ha vesszünk az említett for ciklus ciklusváltozóját (**i**), akkor az **i**-dik elemet mindig az **(n-i+1)**-dik elemmel kell kicserélnünk (a cserét egy **x** segédváltozó segítségével végezzük el). Programunk tehát így néz ki:

```
program Pelda28;
uses crt;
const n=20;
var a: array[1..n] of integer;
    i,x: integer;
begin
clrscr;
 randomize;
 for i:=1 to n do a[i]:=random(90)+10;
 {a tömb elemeinek kiírása...}
 for i:=1 to n do write(a[i]:3);
 writeln;
 {a tömb tükrözése...}
 for i:=1 to n div 2 do begin
                        x:=a[i];
                        a[i]:=a[n-i+1];
                        a[n-i+1]:=x;
                        end;
 {a tömb elemeinek kiírása...}
 for i:=1 to n do write(a[i]:3);
 readln;
end.
```

11.4 Tömbök rendezése

Egy N elemű sorozatot nagyság szerint sorba kell rendezni. Nagyon sok rendezési algoritmus létezik. Az alábbiakban ezek közül kettőt fogunk megvizsgálni. Közös vonásuk az lesz, hogy az eredmény, a rendezett sorozat, helyben keletkezik, így az eredeti sorrend elvész. Az a rendezési algoritmus jó, amelynek kicsi a tárigénye és nagyon gyorsan végrehajtja a rendezést, és persze egyszerű, könnyen megérthető a működése.

Tökéletes, minden igényt kielégítő rendezési eljárás nem létezik. Az, hogy mikor melyik rendező algoritmust használjuk, sok összetevő függvénye. Tisztázni kell egy-egy konkrét eljárás kiválasztásakor, hogy az algoritmus elsősorban gyors legyen vagy inkább helytakarékos. Az alábbiakban tárgyalt algoritmusokbanaz N=5 elemű A() vektort fogjuk növekvő sorrendbe rendezni.

	1.	2.	3.	4.	5.
A()	**5**	**3**	**9**	**1**	**7**

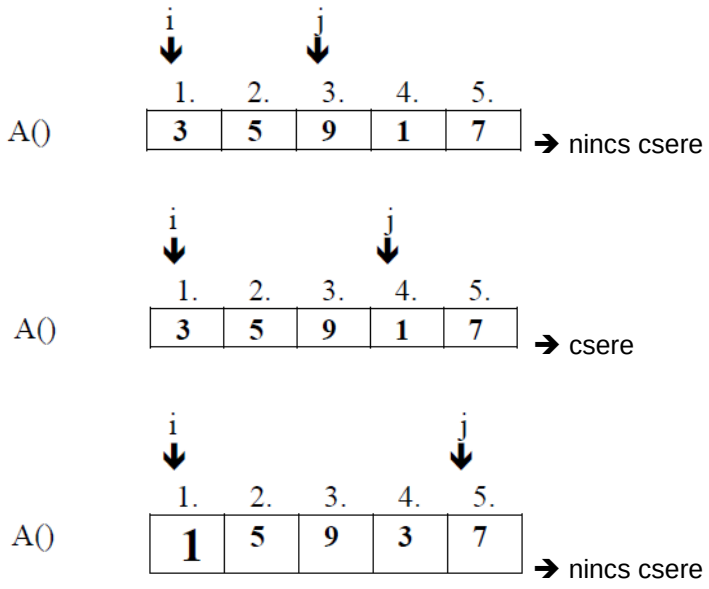
rendezve

	1.	2.	3.	4.	5.
A()	**1**	**3**	**5**	**7**	**9**

11.4.1 Egyszerű cserés rendezés

Hasonlítsuk össze a() tömb első elemét a sorozat összes többi mögötte lévő elemével, s ha valamelyik kisebb nála, akkor cseréljük meg azzal! Ezzel elérhetjük, hogy a sorozat első helyére a legkisebb elem kerül. Folytassuk ugyanezen elven a sorozat második elemével, utoljára pedig az utolsó előttivel.

	i ↓	j ↓				
	1.	2.	3.	4.	5.	
A()	**5**	**3**	**9**	**1**	**7**	➔ csere



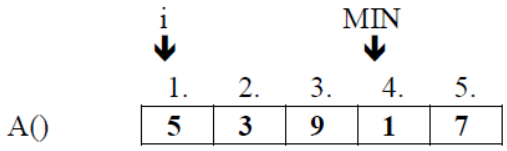
A **j** -vel végigértünk a sorozaton, és az 1. helyre megtaláltuk a sorozat legkisebb elemét. Most **a(1)** biztosan a legkisebb elem. Az **i** -t növeljük eggyel és a 2. helyre keressük meg, a **j** segítségével, **a(2)**, **a(3)**, **a(4)**, **a(5)** közül a legkisebbet. Ez persze biztos nem lesz kisebb **a(1)** - nél, különben ő került volna az 1. helyre. A fentiekben leírt gondolatmenetet követjük, amíg **i** -vel végig nem érünk a sorozaton, azaz az **n-1.** helyre meg nem találtuk **a(n-1)** és **a(n)** közül a kisebbet. Az **n.** helyre pedig nyilvánvalóan a sorozat legnagyobb eleme marad.

```
program cserrend;
uses crt;
const n=5;
var a:array[1..n] of integer;
    i,j,x:integer;
begin
 clrscr;
 randomize;
 i:=0;
 j:=0;
 x:=0;
   write('5 db 1 es 9 kozotti szam generalasahoz nyomj ENTER-t!');
   readln;
   for i:=1 to n do
    begin
     a[i]:=random(10)+1;
     write(a[i],', ')
    end;
   for i:=1 to n-1 do
    for j:=i+1 to n do
     if a[i]>a[j] then
      begin
       x:=a[i];
       a[i]:=a[j];
       a[j]:=x;
      end;
   writeln;
   for i:=1 to n do write(a[i],', ');
 readln;
end.
```

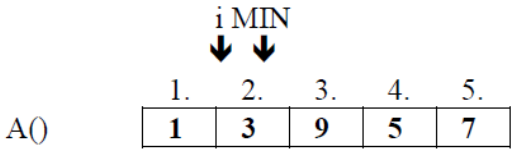
11.4.2 Minimumkiválasztásos rendezés

Az előző módszer hátránya a sok felesleges csere. Célszerűbb lenne az aktuális, **i.** , elemet a mögötte lévők közül egyedül a legkisebbel felcserélni. Ez a felismerés vezet a módszer javításához, a minimumkiválasztásos rendezéshez.

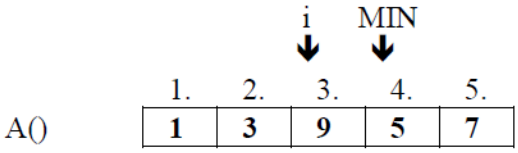
Az **a()** tömb 1. helyére keressük ki az **a(1), a(2), a(3), a(4), a(5)** elemek közül a legkisebbet (minimumkiválasztás)! Ha megtaláltuk a legkisebb elemet, akkor cseréljük ki az **a(1)** - gyel!



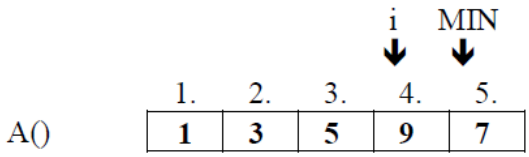
Ekkor az **a(1)** biztosan a vektor legkisebb eleme. Növeljük az **i** -t 1 -gyel, így az **a()** tömb 2. helyére keressük ki **a(2), a(3), a(4), a(5)** közül a legkisebbet!



Most **a(i)** -t cseréljük fel a(min) -nel! (Ez a csere tulajdonképpen felesleges, hiszen most **a(2)** - t cseréljük fel **a(2)** -vel! növeljük **i** -t 1 -gyel!



Most **a(i)** -t cseréljük fel **a(min)** -nel! majd növeljük **i** -t 1 -gyel!



Most **a(i)** -t cseréljük fel **a(min)** -nel! az **i** -vel elértük **n-1** -t, ezután az **a()** biztosan rendezett!

A módszer hátránya, hogy bizonyos esetekben (ha i=min) egy elemet önmagával cserélünk fel. Ez nyilvánvalóan felesleges csere.

```
program minrend;
uses crt;
const n=5;
var a:array[1..n] of integer;
    i,j,x,min:integer;
begin
 clrscr;
 randomize;
 i:=0;
 j:=0;
 x:=0;
   write('5 db 1 es 9 kozotti szam generalasahoz nyomj ENTER-t!');
   readln;
   for i:=1 to n do
    begin
     a[i]:=random(10)+1;
     write(a[i],', ')
    end;
    for i:=1 to n do
     begin
      min:=i;
      for j:=i+1 to n do
       if a[min]>a[j] then min:=j;
      x:=a[i];
      a[i]:=a[min];
      a[min]:=x;
     end;
     writeln;
   for i:=1 to n do write(a[i],', ');
  readln;
end.
```

11.5 Két rendezett tömb összefésülése

A következő példában legyen két tömbünk - **a** és **b**, melyek közül az elsőnek van **na**, a másodikak **nb** darab eleme. Mindkét tömbünk rendezett, növekvő sorrendben (ehhez a program elején a tömbök deklarálásánál megadjuk az tömbök elemeinek értékét). Készítsünk egy olyan programot, amely az **a** és **b** tömböt összefésüli egy új - **c** tömbbe, tehát veszi az **a** és **b** tömb elemeit és azokat sorban egymás után átrakja a **c** tömbbe úgy, hogy a c tömb is rendezett legyen.

Nézzük a feladat tömbjeit ábrán szemléltetve (az **a** és **b** tömbök a program elején adottak, melyek elemeiből a program hozza létre a **c** tömböt):

a[1]	a[2]	a[3]	a[4]	a[5]
8	10	12	17	21

b[1]	b[2]	b[3]	b[4]	b[5]
9	15	19	28	57

c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]	c[10]
8	9	10	12	15	17	19	21	28	57

A programban fogunk használni további két változót - **ai** és **bi**, melyek fogják jelölni, hogy az **a** ill. **b** tömb éppen melyik eleménél járunk.

Az **ai**, **bi** változókat a program elején beállítjuk 1-re, majd a programban (a **c** tömb létrehozására szolgáló ciklusban) mindig az **a** tömb **ai**-dik eleme vagy a **b** tömb **bi**-dik eleme közöl a kisebbet tesszük át a **c** tömbbe (és növeljük az **ai** vagy **bi** értékét egyel attól föggően melyik tömbből raktuk át az elemet a **c** tömbbe). A c tömbhöz is bevezetünk egy **ci** változót, mely azt fogja jelölni, hogy éppen hol járunk a c tömbben (kezdetben ennek az értéke is 1 lesz, majd minen egyes átrakásnál növeljük egyel).

A fent leírt, elemek átrakására szolgáló algoritmust addig fogjuk ismételni, amíg nem érünk az **a** vagy a **b** tömb végére (**ai** vagy **bi** nem éri el a tömb végét, tehát amíg nem lesz több, mint az adott tömb elemeinek száma - **an**, **bn**). Ehhez egy repeat..until ciklust használunk. Ez után már csak a másik tömbből (amelyiknél még nem értünk a tömb végére) átrakjuk az összes megmaradt elemet a **c** tömbbe (**ci**-től kezdve a végéig).

Végül ellenőrzésképpen kiírjuk a képernyőre az eredeti **a**, **b** és az általunk létrehozott **c** tömböt. Programunk így néz ki:

```
program Pelda29;
uses crt;
const an=5;
      bn=5;
      cn=an+bn;
var a: array[1..an] of integer = (8,10,12,17,21);
    b: array[1..bn] of integer = (9,15,19,28,57);
    c: array[1..cn] of integer;
    i,ai,bi,ci: integer;
begin
clrscr;
 ai:=1;
 bi:=1;
 ci:=1;
 repeat
  if a[ai]<b[bi] then begin
                      c[ci]:=a[ai];
                      inc(ai);
                      end
                 else begin
                      c[ci]:=b[bi];
                      inc(bi);
                      end;
  inc(ci);
 until (ai>an) or (bi>bn);
 if (ai>an) then begin
                 for i:=bi to bn do c[ci+(i-bi)]:=b[i];
                 end
            else begin
                 for i:=ai to an do c[ci+(i-ai)]:=a[i];
                 end;
 write('A tomb: ');
 for i:=1 to an do write(a[i],', ');
 writeln;
 write('B tomb: ');
 for i:=1 to bn do write(b[i],', ');
 writeln;
 write('C tomb: ');
 for i:=1 to cn do write(c[i],', ');
 readln;
end.
```

A programban gondolkozzunk el többek között a `for i:=bi to bn do c[ci+(i-bi)]:=b[i];` soron.

Itt valójában egy ciklus segítségével, amelyben az **i** ciklusváltozó értéke **bi**-től **bn**-ig megy átrakjuk a maradék elemeket a **b** tömbből a **c** tömbbe. Ha jobban megfigyeljük, akkor a `c[ci+(i-bi)]` kifejezésben az `(i-bi)` értéke először 0 (hiszen az **i** kezdeti értéke **bi**), majd 1, 2, ... Tehát a **b** tömb maradék elemeit (`b[bi]`, `b[bi+1]`, `b[bi+2]`, ...) a `c[ci]`, `c[ci+1]`, `c[ci+2]`, ... elemekbe rakjuk át, ami a valódi célunk volt.

A `for i:=ai to an do c[ci+(i-ai)]:=a[i];` sor hasonlóan működik, csak itt az **a** tömb maradék elemét rakjuk át a **c** tömbbe.

Feladatok:

1. Olvassunk be egész számokat 0 végjelig egy maximum 100 elemű tömbbe (a tömböt 100 eleműre deklaráljuk, de csak az elejéből használjunk annyi elemet, amennyit a felhasználó a nulla végjelig beír).
- Írjuk ki a számokat a beolvasás sorrendjében.
- Írjuk ki az elemek közül a legkisebbet és a legnagyobbat, tömbindexükkel együtt.
- Írjuk ki az elemeket fordított sorrendben.

(array06)

```
Add meg a(z) 1. szamot: 43
Add meg a(z) 2. szamot: 31
…
Add meg a(z) 8. szamot: 77
Add meg a(z) 9. szamot: 0

_____________________________
43; 32; 89; 48; 21; 56; 54; 77;
Legnagyobb szam: 89 [3]
Legkisebb szam: 21 [5]
77; 54; 56; 21; 48; 89; 32; 43;
```

2. Olvassunk be egész számokat egy 20 elemű tömbbe, majd kérjünk be egy egész számot. Keressük meg a tömbben az első ilyen egész számot, majd írjuk ki a tömbindexét. Ha a tömbben nincs ilyen szám, írjuk ki, hogy a beolvasott szám nincs a tömbben. **(array07)**

```
Adjal meg 20 db egesz szamot!
3, 56, 78, 21, 43, 67, 82, 27, 95, 33,
83, 22, 44, 56, 2, 16, 17, 29, 78, 45
A keresett szam: 78
Az elso elofordulas: 3. elem
```

3. Állítsunk elő egy 50 elemű tömböt véletlen egész számokból (0-tól 9-ig terjedő számok legyenek).
- Írjuk ki a kigenerált tömböt a képernyőre.
- Számítsuk ki az elemek összegét és számtani középértékét.
- Olvassunk be egy 0 és 9 közötti egész számot, majd határozzuk meg, hogy a tömbben ez a szám hányszor fordul elő. **(array08)**

```
50 db 0 és 9 kozotti szam generalasahoz
nyomj ENTERT!
3 6 8 1 3 4 5 9 6  6 2 ……. 3 4 5 6 2 8
7

_____________________________
A szamok osszege: 298
A szamtani kozep: 4.56

_____________________________
Melyik szamot keressem: 4
7 db 4-es szam van a tombben
```

4. Állítsunk elő egy 30 elemű tömböt véletlen egész számokból (0-tól 99-ig).
- Írjuk ki a kigenerált tömböt a képernyőre.
- Olvassunk be egy egész számot. Határozzuk meg, hogy a tömbben melyik számok vannak a legközelebb ehhez a beolvasott számhoz, majd írjuk ki az összes ilyen számot a tömbből a tömbindexükkel együtt. (A két szám közti különbség meghatározásához használjuk az abszolút érték funkciót, pl. **abs(x–y)**) **(array09)**

```
30 db 0 és 99 kozotti szam
generalasahoz nyomj ENTERT!
43 66 78 11 43 84 55 94 63 56 52 ……. 23
54 58 86 82 38 71

_____________________________
Melyik szamhoz legkozelebbit keressuk:
40
Eredmeny: 43 [5]
```

5. Állítsunk elő egy 150 elemű tömböt véletlen egész számokból –999-től 999-ig. Rendezzük ezt a tömböt nagyság szerint növekvő sorrendben, majd írjuk ki a képernyőre.
A rendezésre a következő rendezési algoritmust használjuk (rendezés a legkisebb elem kiválasztásával):
- kiválasztjuk a tömb 1.-150. elemei közül a legkisebbet, ezt kicseréljük a tömb 1. elemével (így a legkisebb szám a tömbben az első helyre került),
- kiválasztjuk a tömb 2.-150. elemei közül a legkisebbet, ezt kicseréljük a tömb 2. elemével,
- végül kiválasztjuk a tömb 149.-150. elemei közül a legkisebbet, ezt kicseréljük a tömb 149. elemével.

(array10)

```
150 db -999 és 999 kozotti szam
generalasahoz nyomj ENTERT!
-543 636 718 -911 -643 384 355 294 -663
756 532 ……. -523 -654 588 -386 812
138 471


_____________________

A tomb rendezve:

-989, -976, -970, …

…

… 957, 978, 998, 999
```

12 Kétdimenziós tömbök

- kétdimenziós tömbök

12.1 Kétdimenziós tömbök

Az eddig használt egydimenziós tömbökön túl használhatunk többdimenziós tömböket is. Példaként nézzük meg, hogyan is képzelhetünk el egy 4x5 méretű kétdimenziós tömböt és hogyan hivatkozhatunk a tömb elemeire:

a[1,1] 0	a[1,2] 0	a[1,3] 0	a[1,4] 0	a[1,5] 0
a[2,1] 0	a[2,2] 0	a[2,3] 0	a[2,4] 0	a[2,5] 0
a[3,1] 0	a[3,2] 0	a[3,3] 0	a[3,4] 0	a[3,5] 0
a[4,1] 0	a[4,2] 0	a[4,3] 0	a[4,4] 0	a[4,5] 0

Láthotjuk, hogy az egyes elemekre az elem sorának és oszlopának megadásával hivatkozhatunk, például a harmadik sor harmadik eleme: a[3,3].

Az ábrán látható tömbben minden elem értéke 0. Mivel a többdimenziós tömbökkel egymásba ágyazott ciklusok segítségével dolgozunk (a külső ciklus adja meg a sorindexeket, a belső az oszlopindexeket), a tömb összes elemének 0-ra állítása is két ilyen egymásba ágyazott ciklus segítségével oldható meg:

```
program Pelda30a;
uses crt;
var a:array[1..4,1..5] of integer;
    i,j:integer;
begin
clrscr;
 for i:=1 to 4 do
  for j:=1 to 5 do a[i,j]:=0;
readln;
end.
```

A tömb kiírása a képernyőre is két egymásba ágyazott ciklus segítségével lesz megoldva. A belső ciklus fog kiírni egy sort, a külső ciklus pedig megadja a sorok számát. Az előző példa kiírással kiegészítve:

```
program Pelda30b;
uses crt;
var a:array[1..4,1..5] of integer;
    i,j:integer;
begin
clrscr;
 for i:=1 to 4 do
  for j:=1 to 5 do a[i,j]:=0;
 for i:=1 to 4 do
  begin
  for j:=1 to 5 do write(a[i,j]:2);
  writeln;
  end;
writeln;
end.
```

Készítsünk most egy programot, amely kigenerálja egy 10x10-es kétdimenziós tömbbe a 10-es szorzótáblát, majd kiírja a képernyőre. A tömbünkbe tehát a következő értékeket szeretnénk kigenerálni:

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Ehhez a tömb mindegyik elemébe berakjuk az oszlopindexének és a sorindexének a szorzatát. Programunk így néz ki:

```
program Pelda31a;
uses crt;
var t:array[1..10,1..10] of integer;
    i,j:integer;
begin
 clrscr;
 { kigenerálás... }
 for i:=1 to 10 do
  for j:=1 to 10 do t[i,j]:=i*j;
 { kiírás... }
 for i:=1 to 10 do
  begin
  for j:=1 to 10 do write(t[i,j]:4);
  writeln;
  end;
 writeln;
end.
```

A fenti programban a tömb kigenerálását és kiírását elvégezhetjük ugyanabban a ciklusban is. Programunk ekkor így módosul:

```
program Pelda31b;
uses crt;
var t:array[1..10,1..10] of integer;
    i,j:integer;
begin
 clrscr;
 { kigenerálás és kiírás ugyanabban a ciklusban }
 for i:=1 to 10 do
  begin
  for j:=1 to 10 do begin
                    t[i,j]:=i*j;
                    write(t[i,j]:4);
                    end;
  writeln;
  end;
 readln;
end.
```

13 Műveletek kétdimenziós tömbökkel

- legkisebb, legnagyobb elem megkeresése
- tömb tükrözése a függőleges tengelye szerint

13.1 Legkisebb, legnagyobb elem megkeresése

Készítsünk programot, amely egy 6x6-os kétdimenziós tömbbe kigenerál 10 és 70 közötti véletlen számokat, majd kiírja a kigenerált tömböt. A program második része keresse meg a tömb legkisebb és legnagyobb elemét, majd írja ki ezt a képernyőre.

A tömb kigenerálása - ahogy már megszoktuk - két egymásba ágyazott ciklus segítségével lesz megvalósítva.

A tömb legkisebb ill. legnagyobb elemét hasonlóan fogjuk megkeresni, mint az egydimenziós tömbnél. Veszünk két változót - **min**, **max**, melyekbe a keresés előtt beállítjuk az **a[1,1]** elem értékét. Ez után végigmegyünk a tömb összes elemén (két egymásba ágyazott ciklus segítségével), és ha találunk kisebb ill. nagyobb elemet, akkor ezt megjegyezzük a **min** ill. **max** változókban. Végül a ciklusok lefutása után kiírjuk a **min** és **max** változók értékét.

```
program Pelda32a;
uses crt;
var a: array[1..6,1..6] of integer;
    i,j,min,max:integer;
begin
clrscr;
 { tömb kigenerálása }
 randomize;
 for i:=1 to 6 do
  for j:=1 to 6 do a[i,j]:=random(61)+10;
 { tömb kiírása }
 for i:=1 to 6 do
  begin
  for j:=1 to 6 do write(a[i,j]:3);
  writeln;
  end;
 { min, max keresése }
 min:=a[1,1];
 max:=a[1,1];
 for i:=1 to 6 do
  for j:=1 to 6 do
   begin
   if a[i,j]<min then min:=a[i,j];
   if a[i,j]>max then max:=a[i,j];
   end;
 { min, max kiírása }
 writeln('Legkisebb elem: ',min);
 writeln('Legnagyobb elem: ',max);
 readln;
end.
```

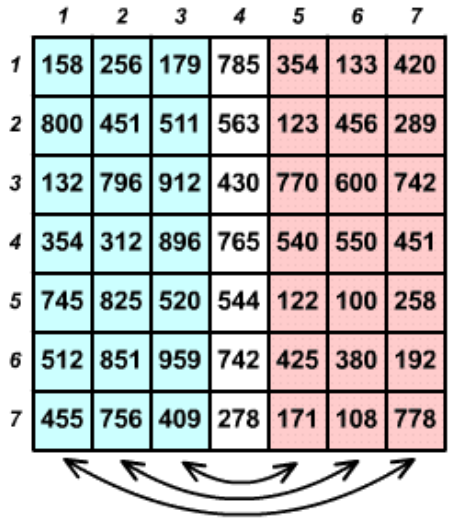
Próbáljuk meg módosítani az előző feladatot úgy, hogy ne csak a maximum és minimum értékét írja ki, hanem azt is, hogy ezeket az elemeket melyik sor melyik oszlopában találta meg (tehát a maximum és a minimum tömbindexeit is).

Ehhez bevezetük négy új változót: **min_i**, **min_j**, **max_i**, **max_j**, melyekben mindig amikor változtatunk a **min** ill. **max** értékén, megjegyezzük az éppen elmentett elem tömbindexeit. Programunk ezekkel a módosításokkal így néz ki:

```
program Pelda32b;
uses crt;
var a: array[1..6,1..6] of integer;
    i,j,min,max,min_i,min_j,max_i,max_j:integer;
begin
 clrscr;
 { tömb kigenerálása }
 randomize;
 for i:=1 to 6 do
  for j:=1 to 6 do a[i,j]:=random(61)+10;
 { tömb kiírása }
 for i:=1 to 6 do
  begin
  for j:=1 to 6 do write(a[i,j]:3);
  writeln;
  end;
 { min, max keresése }
 min:=a[1,1];
 min_i:=1;
 min_j:=1;
 max:=a[1,1];
 max_i:=1;
 max_j:=1;
 for i:=1 to 6 do
  for j:=1 to 6 do
   begin
   if a[i,j]<min then begin
                      min:=a[i,j];
                      min_i:=i;
                      min_j:=j;
                      end;
   if a[i,j]>max then begin
                      max:=a[i,j];
                      max_i:=i;
                      max_j:=j;
                      end;
   end;
 { min, max kiírása }
 writeln('Legkisebb elem: ',min,' (',min_i,'. sor ',min_j,'. oszlop)');
 writeln('Legnagyobb elem: ',max,' (',max_i,'. sor ',max_j,'. oszlop)');
 readln;
end.
```

13.2 Tömb tükrözése a függőleges tengelye szerint

Készítsünk programot, amely egy 7x7-es kétdimenziós tömbbe kigenerál 100 és 999 közötti véletlen számokat, majd tükrözi a tömböt a függőleges tengelye szerint. A program írja ki az eredeti, majd a tükrözés utáni tömböt is.



	1	2	3	4	5	6	7
1	158	256	179	785	354	133	420
2	800	451	511	563	123	456	289
3	132	796	912	430	770	600	742
4	354	312	896	765	540	550	451
5	745	825	520	544	122	100	258
6	512	851	959	742	425	380	192
7	455	756	409	278	171	108	778

Ehhez előbb a tömb első, majd a második, harmadik, stb. sorait fogjuk tükrözni egymás után. Egy külső ciklus fogja megadni hogy éppen melyik sor tükrözését végezzük. Ezen belül lesz egy belső ciklus, amely az adott sort tükrözését végzi el – ez egy hasonló ciklus, mint amilyent az egydimenziós tömb tükrözésénél használtunk. Programunk így néz ki:

```
program Pelda33;
uses crt;
const n=7;
var a: array[1..n,1..n] of integer;
    i,j,x:integer;
begin
 clrscr;
 { tömb kigenerálása }
 randomize;
 for i:=1 to n do
  for j:=1 to n do a[i,j]:=random(900)+100;
 { tömb kiírása }
 for i:=1 to n do
  begin
  for j:=1 to n do write(a[i,j]:4);
  writeln;
  end;
 { tömb tükrözése }
 for i:=1 to n do
  begin
  { egy sor tükrözése }
  for j:=1 to n div 2 do begin
                         x:=a[i,j];
                         a[i,j]:=a[i,n-j+1];
                         a[i,n-j+1]:=x;
                         end;
  end;
 { tömb kiírása }
 writeln;
 for i:=1 to n do
  begin
  for j:=1 to n do write(a[i,j]:4);
  writeln;
  end;
readln;
end.
```

Feladatok:

1. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe generáljunk véletlen egész számokat 10-tol 99-ig. (A tömböt 10x10-esre deklaráljuk, de ennek csak az NxN-es részét használjuk.)

```
 34 58 19 15
 85 50 41 54
 99 25 17 84
 45 78 10 95
```

- Írjuk ki a kigenerált tömböt, pl. N=4-re:
(array11)

2. Készítsünk programot, amely egy 10x10-es tömbbe beírja a számokat 1-től 100-ig a minta alapján:
A kigenerált tömböt írjuk ki a képernyőre.
(array12)

```
  1   2   3   4   5   6   7   8   9  10
 11  12  13  14  15  16  17  18  19  20
 21  22  23  24  25  26  27  28  29  30
 31  32  33  34  35  36  37  38  39  40
 41  42  43  44  45  46  47  48  49  50
 51  52  53  54  55  56  57  58  59  60
 61  62  63  64  65  66  67  68  69  70
 71  72  73  74  75  76  77  78  79  80
 81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99 100
```

3. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe olvassunk be természetes számokat. (1-99-ig)
- Írjuk ki a tömböt a képernyőre, pl. N=4-re és a következő számokra ilyen formában::

```
 60 19 17 53
 17 82 26 10
  8 11 29 90
 64 77 19 70

  0 19 17 53
 17  0 26 10
  8 11  0 90
 64 77 19  0
```

- Írjunk a tömb átlójába 0-kat, majd írjuk ki a tömböt újra a képernyőre:
(array13)

4. Generáljunk ki egy 8x8-as tömbbe véletlen számokat (8-88-ig), majd írjuk ki a tömböt.

```
 23  15  77  …
  8  27  …
 43  …
 …
A legkisebb: 8
A legnagyobb: 86
A számtani közép: 40,55
```

- Keressük meg a tömb legkisebb és legnagyobb elemét.
- Számítsuk ki a tömb elemeinek összegét és számtani középértékét.
(array14)

5. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe generáljunk véletlen egész számokat 10-tol 99-ig.
- Írjuk ki a kigenerált tömböt, pl. N=3-ra:
- Keressük meg a tömb legkisebb elemét és ezt az elemet cseréljük ki a tömb első sorának első elemével. Írjuk ki újból a tömböt
(array15)

```
 34 58 19
 85 50 11
 99 25 17

 11 58 19
 85 50 34
 99 25 17
```

6. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe olvassunk be természetes számokat. (1-99-ig)

```
  1 58 19 20
 10  5 17 16
  9 25 50 70
 33 91 80 17

 33 91 80 17
 10  5 17 16
  9 25 50 70
  1 58 19 20
```

- Írjuk ki a tömböt a képernyőre, pl. N=4-re és a következő számokra ilyen formában:

- Cseréljük ki a tömbben az első és az utolsó sorban levő elemeket, majd írjuk ki a tömböt újra a képernyőre **(array16)**

7. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe generáljunk véletlen egész számokat 10-tol 99-ig.

- Írjuk ki a kigenerált tömböt, pl. N=6-ra:

- Tükrözzük a tömböt a vízszintes tengelye mentén (első sort cseréljük ki az utolsóval, másodikat az utolsó előttivel, stb.):
(array17)

```
34 58 19 51 11 58
85 50 11 51 13 33
99 25 17 22 27 91
12 34 87 42 65 70
11 10 69 74 88 92
30 34 74 75 54 47

30 34 74 75 54 47
11 10 69 74 88 92
12 34 87 42 65 70
99 25 17 22 27 91
85 50 11 51 13 33
34 58 19 51 11 58
```

8. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe generáljunk véletlen egész számokat 10-tol 99-ig.

- Írjuk ki a kigenerált tömböt, pl. N=6-ra:

- Tükrözzük a tömböt a bal felső sarokból a jobb alsó sarokba húzódó átlója szerint (tehát valójában az első oszlop felcserélődik az első sorral, a második oszlop a második sorral, stb.):

(array18)

```
34 58 19 51 11 58
85 50 11 51 13 33
99 25 17 22 27 91
12 34 87 42 65 70
11 10 69 74 88 92
30 34 74 75 54 47

34 85 99 12 11 30
58 50 25 34 10 34
19 11 17 87 69 74
51 51 22 42 74 75
11 13 27 65 88 54
58 33 91 70 92 47
```

9. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe olvassunk be természetes számokat. (1-99-ig)

- Írjuk ki a tömböt a képernyőre, pl. N=4-re és a következő számokra ilyen formában::

- Írjunk a tömb külső keretébe 0-kat, majd írjuk ki a tömböt újra a képernyőre:
(array19)

```
60 19 17 53
17 82 26 10
 8 11 29 90
64 77 19 70

 0  0  0  0
 0 82 26  0
 0 11 29  0
 0  0  0  0
```

10. Készíts egy kétdimenziós 5*5 –os tömböt, melyet feltöltesz páros számokkal!
(array20)

```
 2  4  6  8 10
12 14 16 18 20
22 24 26 28 30
32 34 36 38 40
42 44 46 48 50
```

14 Eljárások (alprogramok), függvények

- eljárások (**procedure**)
- változók hatásköre
- függvények (**function**)

14.1 Eljárások (procedure)

A programozás során gyakran előfordulhat, hogy valamilyen programrészt a programunkban többször végrehajtunk. Például a tömböknél kiírattuk a tömböt, változtattunk rajta valamit majd ismét kiírattuk a képernyőre. Ilyenkor egyszerűbb a tömb kiírására egy eljárást (alprogramot) írni és kiírásnál csak ezt az eljárást meghívni. A programunk ebben az esetben így nézhet ki:

```
program Pelda34;
uses crt;
const n=10;
var i:integer;
    a:array[1..n] of integer;

 procedure kiir;
 begin
  write('A tomb elemei: ');
  for i:=1 to n do write(a[i],' ');
  writeln;
 end;

begin
 clrscr;
 { tomb kigeneralasa: }
 for i:=1 to n do a[i]:=random(99)+1;
 { tomb kiirasa alprogrammal: }
 kiir;
 { tomb megvaltoztatasa, pl. mindegyik elemet megszorozzuk 2-vel: }
 for i:=1 to n do a[i]:=2*a[i];
 { tomb kiirasa ismet alprogrammal: }
 kiir;
 readln;
end.
```

Az eljárást (alprogramot) mindig a **procedure** szóval kezdjük írni még a főprogram kezdete előtt, a változók deklarálása után. A **procedure** szó után megadjuk az eljárás nevét, majd a következő sortól írhatjuk az eljárást **begin** és **end;** közé. Az eljárást a főprogramból egyszerűen az eljárás nevével hívhatjuk meg. Egy programba írhatunk több eljárást is, pl. egyet a tömb generalására, egyet a kiírására, stb.

Az eljárásunknak átadhatunk a főprogramból valamilyen értékeket is paraméterek segítségével. Például, ha szeretnénk készíteni egy olyan eljárást, amely kiír valamilyen szöveget úgy, hogy minden betűjét a szövegnek megváltoztatja nagy betűre, akkor a kiírandó szöveget átadhatjuk az eljárás paraméterében. Ekkor a programunk így nézhet ki:

```
program Pelda35a;
uses crt;
var s:string;

 procedure nagybetukkel(m:string);
 var i:integer;
 begin
  for i:=1 to length(m) do m[i]:=upcase(m[i]);
  writeln(m);
 end;

begin
 clrscr;
 nagybetukkel('Kerek egy szoveget: ');
 readln(s);
 nagybetukkel(s);
 writeln('S erteke a program vegen: ',s);
 readln;
end.
```

Ez program kiírja nagy betűkkel, hogy "**KEREK EGY SZOVEGET:** ", majd bekér egy mondatot és ezt kiírja nagy betűkkel. Végül még kiírja az **s** értékét (eredeti szöveget). Az eljárásban paraméterként szereplő **m** változót csak az eljáráson belül használhatjuk, máshol nem érhető el. Hasonlóan az eljárásban deklarált **i** változót is csak az eljáráson belül használhatjuk - ezek úgynevezett lokális (helyi) változók.

Az eljárás az **s** változó értékét nem változtatja meg, csak a helyi **m** változóét. Tehát az eljárás lefutása után az **s** változóban marad az eredeti, kisbetűs szöveg (ez kiíródik a főprogram végén). Az ilyen **m** változót nevezzük formális paraméternek.

Ha valami végett mégis olyan eljárást szeretnénk írni, amely az **s** változó értékét is megváltoztatja (tehát azt szeretnénk, hogy az eljárás lefutása után az **s** változóban is a nagybetűs szöveg legyen - ugyanaz mint az eljáráson belül az **m** változóban), akkor az eljárásunk paraméterének megadásánál az **m** változó előtt használhatjuk a **var** utasítást. Az ilyen paramétert nevezzük valódi paraméternek.

```
program Pelda35b;
uses crt;
var s:string;

 procedure nagybetukkel(var m:string);
 var i:integer;
 begin
  for i:=1 to length(m) do m[i]:=upcase(m[i]);
  writeln(m);
 end;

begin
 clrscr;
 s:='Kerek egy szoveget: ';
 nagybetukkel(s);
 readln(s);
 nagybetukkel(s);
 writeln('S erteke a program vegen: ',s);
 readln;
end.
```

Eljárásunkban használhatunk több változót is, melyek közül lehet némelyik formális paraméter, némelyik valódi paraméter. Például egy eljárást megadhatunk ilyen paraméterezéssel is:

```
procedure szamol(s:string; a,b:integer; var c:integer);
```

Ebben az esetben csak a **c** valódi paraméter, tehát a főprogramban az eljárás lefutása után csak a **c** helyén megadott változó értéke változik meg.

14.2 Változók hatásköre

A programozás során használhatunk **globális változókat** és **lokális (helyi) változókat**.

A **globális változók** azok a változók, melyet a programunkban bárhol elérhetünk, bárhol használhatjuk - a főprogramba is és az eljárásban is. A globális változókat a programunk elején a **var** szó után soroljuk fel, ahogy eddig is tettük.

A **lokális (helyi) változók** azok a változók, melyek a főprogramban nem érhetők el, csak az eljárásban. Tehát ezeket a változókat csak az adott eljárásban használhatjuk, a főprogramban nem. Ezeket szintén a **var** utasítás segítségével adjuk meg, de nem a programunk elején, hanem abban az eljárásban, melyben ezeket használni szeretnénk.

Ha ugyanolyan nevű változót deklarálunk globális és lokális változóként is, akkor az eljárásban csak a lokális (helyi) változóval tudunk dolgozni, az eljáráson kívül pedig a globális változóval. Az eljárás nem fogja megváltoztatni a globális változó értékét. Példaként vizsgáljuk meg és próbáljuk ki ezt a programot:

```
program Pelda36;
uses crt;
var v:integer;

 procedure alprg;
 var v:integer;
 begin
 v:=888;
 writeln('Lokalis V nevu valtozo erteke: ',v);
 end;

begin
 clrscr;
 v:=555;
 writeln('Globalis V nevu valtozo erteke: ',v);
 alprg;
 writeln('Globalis V nevu valtozo erteke: ',v);
 readln;
end.
```

A program kiírja először az 555-t, majd a 888-at végül ismét az 555-t.

14.3 Függvények (function)

A Pascalban vannak előre definiált függvények, ilyen például az **abs()**, **sin()**, **cos()**, **upper()**, **length()**, stb. Készíthetünk mi is saját függvényt. Ezt hasonlóan kell megírnunk mint a saját eljárásunkat, a különbség csak abban van, hogy a **procedure** szó helyett a **function**-t használjuk, és meg kell adnunk hogy milyen típust adjon vissza az általunk készített függvény. Továbbá a függvényünkön belül (általában a végén) egy ilyen típusú értéket kell adunk a függvénynek (függvény nevének).

Példaként készítsünk egy függvényt, amely megszámolja, hogy egy megadott szövegben mennyi szóköz van és ezt a számot adja vissza függvényértékként:

```
program Pelda37;
uses crt;
var s:string;

 function helyekszama(x:string):integer;
 var i,h:integer;
 begin
  h:=0;
  for i:=1 to length(x) do
    if x[i]=' ' then h:=h+1;
  helyekszama:=h;
 end;

begin
 clrscr;
 write('Irj be egy modnatot: ');
 readln(s);
 writeln('A mondatban ',helyekszama(s),' szokoz van.');
 readln;
end.
```

Feladatok:

1. Készíts programot, melyben ötöslottó-sorsolást generálsz! A program írásakor a generálást és a kiíratást alprogrammal old meg! Tíz heti lottószámot gemeráljál! (a számok az ötöslotóban 1-90-ig vannak)
(proc01)

```
1. het: 7; 35;62;44;81;
2. het: 56;34;87;23;55;
...
9. het: 68;90;24;51;11;
10. het: 4;55;28;56;14;
```

2. Készíts programot, melyben generálsz egy 5*5 –ös kétdimenziós mátrixot, véletlen számokkal feltöltve. Az eredeti számok 1 és 10 közöttiek legyenek! Majd mátrix elemeit, megszorozzuk mindig 2-vel, tehát az a[1,1]=3, majd 6, aztán 12, majd 24, végül 48. A számok 5 szóköz távolságra legyenek!
A mátrixokat vonalakkal válasszuk el, melyet alprogrammal oljál meg.
A generálás és a kiíratás legyen külön procedura-ban!
(proc02)

```
     3     5     2     9     4
    10     6     3     5     3
     1     2     2     8     9
     6     5     3     4     2
     8     7     4     1    10
_____________________________________
     6    10     4    18     8
    20    12     6    10     6
…
```

3. Írjál programot, benne egy olyan eljárással, amely kicserél két paraméter értékét! (csere(a,b))
(proc03)

```
Add meg az „x”-et: 9
Add meg az „y”-t: 2

Cserelve: x=2  y=9
```

4. Írjál programot, melyben létrehozol egy függvényt amiben a^b-t számolja!
(func01)

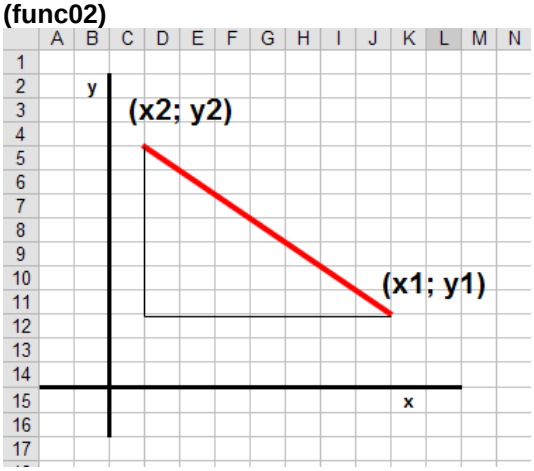
```
Add meg az alapszamot: 3
Add meg a hatvanyt: 2

Eredmeny: 9
```

5. Írj programot, melyben létrehozol egy függvényt, ami visszaadja egy vektor hosszát!

- haszáld az sqr (négyzetreemelés) és sqrt (gyökvonás) beépített függvényeket

$$a^2+b^2=c^2$$

(func02)




```
Add meg a koordinatakat:
X1: 1
Y1: 5
X2: 3
Y2: 9

_________________________

A vektor hossza: 4,47
```

15 Unitok

5.1 Unitok

Mik azok a unitok?

A unit önállóan lefordítható programegység, amely jól definiált kapcsolódási felületen (interface) keresztül kapcsolódik a program más részeihez. Alapvetõ elv a moduláris programozás során, hogy a modulok belsõ része (implementation) rejtett marad a külvilág számára. Valójában az egyes unitok újabb eljárásokat, függvényeket, konstansokat és típusokat tartalmaznak, így ezeket a unitokat használva további eljárásokat, függvényeket ill. típusokat használhatunk a programjainkban.

A pascal rendszer sok szabványos modullal rendelkezik, amelyek hasznos típust, konstanst, eljárást és függvényt tartalmaznak. Ezek közül a leggyakrabban használtak:

- **system unit** - ez a unit automatikosan hozzákapcsolódik minden pascal programhoz, ez tartalmazza tartalmazza a szabványos pascal nyelv eljárásait és függvényeit, melyek közül már néhányat megismerkedtünk (pl. write, writeln, read, readln, inc, dec, abs, ...),
- **crt unit** - elsõsorban a képernyõ szöveges üzemmódját támogató eljárásokat és függvényeket tartalmaz. Ezek segítségével lehetõség van a kurzor pozícionálására, a használt színek beállítására. Ezen kívül a modul tartalmaz néhány eljárást, amelyek segítségével a billentyűzet speciális módon érhetõ el, továbbá a unit segítségével lehetõség adódik hanggenerálásra is.
- **dos unit** - azok az eljárásokat és függvényeket tartalmazza, melyek segítségével hatékonyabban kihasználhatjuk az DOS operációs rendszer lehetõségeit - pl. idő. dátum lekérdezése, mappákkal, állományokkal való munka,
- **graph unit** - a képernyõ grafikus üzemmódját támogató típusok, konstansok, eljárások és függvények gazdag készletét tartalmazza,
- **wincrt unit** - window operációs rendszer alatt futó program grafikus ablakában használható crt unit,
- **winmouse unit** - windows operációs rendszer alatt futó program grafikus ablakában használható unit, amely az egér kezelésére szolgál,
- **string unit** - #0 karakterrel végzõdõ dinamikus stringek tamogatása.

Ahhoz hogy valemelyik unitot (unitokat) használhassunk a programunkban, meg kell adnunk a **uses** hivatkozás után a unit nevét. Ez alól egyedüli kivétel a system unit, mivel ez automatikusan hozzákapcsolódik minden pascal programhoz.

```
program unitok;
uses crt;
var .... ;
begin
  ....
end.
```

Használhatunk egyszerre több unitot is, ekkor vesszõvel választjuk el õket:

```
program unitok;
uses crt, dos, graph;
var .... ;
begin
  ....
end.
```

15.2 CRT unit

A **Crt** unit a billentyőzetről való bevitel és a képernyőre írás valamennyi elemét tartalmazza. Segítségével lekérdezhető a billentyőzet, beállíthatók a képernyő üzemmódjai és színei, ablakok definiálhatók, hangok állíthatók elő stb.

A **Crt** egység néhány fontosabb elemei:

ClrScr; képernyőtörlés
Delay (ms:Word) késleltetés (ms-ban)
GotoXY (X,Y:Byte) kurzor helyzete
KeyPressed : Boolean
ReadKey : Char
TextBackGround (szín:Byte) a háttér színe
TextColor (szín:Byte) a betű színe
Window (X1,Y1,X2,Y2:Byte) belső ablakot hoz létre

A **Crt** egységben vannak a színkódok is, ezekkel lehet a képernyőt, az ablakok és az írás színeit beállítani (a **TextBackGround** és a **TextColor** eljárások felhasználásával).

A **színkódok** a következők (háttérszín 0..7, betűszín 0..15):

0 Black (fekete)	**8** DarkGray (sötétszürke)
1 Blue (kék)	**9** LightBlue (világoskék)
2 Green (zöld)	**10** LightGreen (világoszöld)
3 Cyan (türkiz)	**11** LightCyan (világostürkiz)
4 Red (piros)	**12** LightRed (világospiros)
5 Magenta (lila)	**13** LightMagenta (világoslila)
6 Brown (barna)	**14** Yellow (sárga)
7 LightGray (világosszürke)	**15** White (fehér)

Így ha valamilyen színt szeretnénk használni, megadhatjuk a szín számát (pl. 4) vagy helyette a hozzá tartozó konstant, ami nem más mint a szín angol neve (pl. red). A szöveg színét a **textcolor** paranccsal állíthatjuk be. A textcolor parancs után addig lesz érvényes a megadott szín, amíg azt nem állítjuk át egy másik színre a **textcolor** paranccsal. Pl.:

```
program Pelda38;
uses crt;
begin
 clrscr;
 textcolor(red);
 writeln('Ez a ket sor');
 writeln('pirossal van irva.');
 textcolor(10);
 writeln('Ez pedig vilagoszolddel.');
 textcolor(7);
 writeln('Ez mar az eddig megszokott szurkevel.');
 readln;
end.
```

A **clrscr** parancs a képernyõ letörlésére szolgál. A képernyõtörlés után a kurzor átáll az elsõ sor elsõ oszlopába.

A szöveg (betűk) színéhez hasonló módon megadhatjuk a szöveg háttérszínét is. Ezt a **textbackground** paranccsal tehetjük meg.

```
program Pelda39;
uses crt;
begin
 textbackground(blue);
 clrscr; {fontos, hogy a képernyőtörlés később legyen, mint a háttérszín beáll.}
 textcolor(yellow);
 writeln('Sarga betu kek haterrel.');
 textbackground(0);
 textcolor(7);
 writeln('Eredeti - szurke betu feketen.');
readln;
end.
```

A **gotoxy(** *oszlop* **,** *sor* **)** parancs segítségével beállíthatjuk a kurzor pozícióját a képernyõn, így segítségével bárhová ki tudunk írni szöveget a képernyõre. Ehhez tudnunk kell, hogy a képernyõ felbontása szöveges módban 80x25, tehát 1-tõl 80-ig állíthatjuk be az oszlopot és 1-tõl 25-ig a sort. A következõ példa letörli a képernyõt, majd kiírja a képernyõ közepére a "Hello" szót.

```
program Pelda40;
uses crt;
begin
 clrscr;
 gotoxy(39,13);
 write('Hello');
 readln;
end.
```

Feladat: Készíts egyszerű programot, melyben sárga betűkkel, a 10. sorban a 35. karakterpozíciótól kiíratod a Vezeték és Keresztnevedet, kék háttérszínre!

```
program Pelda41;
uses crt;
begin
 textbackground(1);
 clrscr; {fontos, hogy a képernyőtörlés később legyen, mint a háttérszín beáll.}
 textcolor(14);
 gotoxy(35,10);
 writeln('Vezeteknev Keresztnev');
 readln;
end.
```

A crt unit tartalmaz a bilentyűzetről való beolvasáshoz is egy függvényt, ez a **readkey**. Ez a parancs vár egy billentyű megnyomására. A lenyomott billentyűt a **readkey** függvény visszadja, így azt megjegyezhetjük (egy char típusú változóban).

```
program Pelda42;
uses crt;
var c:char;
begin
 clrscr;
 writeln('Nyomj meg egy betut!');
 c:=readkey;
 writeln('Az ',c,' billentyut nyomtad meg!');
 writeln('Nyomj meg egy tetszoleges billentyut.');
 readkey;
end.
```

Másik hasznos billentyűzettel kapcsolatos függvény a **keypressed**. Ennek értéke igaz/hamis lehet. Értéke akkor igaz, ha a felhasználó lenyomott valamilyen billentyűt. Ilyenkor a lenyomott billentyűt a **readkey** segítségével olvashatjuk ki. A következő program addig fogja a "Hello!" szót kiírni a képernyőre véletlenszám generátorral kigenerált helyre véletlen színnel, amíg nem nyomunk meg egy billentyűt.

```
program Pelda43;
uses crt;
var c:char;
begin
 randomize;
 clrscr;
 repeat
  gotoxy(random(75)+1,random(24)+1);
  textcolor(random(15)+1);
  write('Hello!');
  delay(100);
 until keypressed;
 c:=readkey;
end.
```

A **delay(100)** parancs leállítja 100 millisekundumra a program futását, tehát minden kiírás után vár 100 msec-ot. Ennek a parancsnak a segítségével tudjuk a fenti programunkban megadni, hogy milyen gyorsan írja ki a program a képernyõre az üzeneteket.

A következõ program kiírja egy lenyomott billentyű ASCII kódját. Ez a program hasznos segédprogram lehet a későbbiekben, ha meg szeretnénk határozni melyik billentyűhöz milyen ASCII kód tartozik. Az ESC kódja a 27, ezért van a ciklus feltételénél a c=#27 kifejezés. A ciklus tehát akkor fejezõdik be, ha a c-ben levõ karakter ASCII kódja egyenlõ 27-tel, vagyis ha az ESC billentyűt nyomtuk meg.

```
program Pelda44;
uses crt;
var c:char;
begin
 clrscr;
 repeat
  c:=readkey;
  writeln(c,' kodja: ',ord(c));
 until c=#27;
end.
```

Ennek a programnak a segítségével figyeljük meg az egyes billentyűk ASCII kódjait. Észrevehetjük, hogy némelyik billentyűnél - pl. F1, F2, ... és a nyilak lenyomásánál két kód jelenik meg. Ez azért van, mivel ezeknél a gomboknál a billentyűzet két kódot küld a számítógépnek, először a 0-t, majd egy másik kódot.

Feladat: Készíts programot melyben induláskor kék hátteret látsz, majd ENTER lenyomására rajzol egy zöld téglalapot (10,10,20,20), melybe sárga betűkkel legyen beleírva a „Teglalap” szöveg!

```
program Pelda45;

uses ctr;
begin
 textbackground(blue);
 clrscr;
 readln;
 textcolor(14);
 window(10,10,20,20);
 textbackground(green);
 clrscr;
 writeln('Teglalap');
readlan;
emd.
```

Az alábbi program (következő oldal) bemutatja hogyan készíthetünk egy egyszerű menüt az eddig tanultak alapján, melyben a felfelé ill. lefelé nyilak segítségével mozoghatunk, Enter-rel aktiválhatjuk a kiválasztott menüpontot és Esc-el léphetünk ki a programból (a későbbiekben sokszor felhasználhatjuk ezt a programot):

```
program Pelda46;
uses crt;
var k:integer; { ebben fogjuk tarolni, hogy eppen melyik menuponton vagyunk }
    c:char;
begin
 clrscr;
 k:=1;
 { kiirja a menut }
 textbackground(red);
 textcolor(white);
 gotoxy(10,10);
 write(' Elso ');
 textbackground(blue);
 textcolor(yellow);
 gotoxy(10,11);
 write(' Masodik ');
 gotoxy(10,12);
 write(' Harmadik ');
 { a kurzort beallitjuk a jobb also sarokba }
 gotoxy(80,25);
 { beolvas egy billentyut es a menut ettol fuggoen atrajzolja }
 repeat
  c:=readkey;
  { ha valamelyik nyil lett megnyomva }
  if c=#0 then begin
               { atfestjuk a kivalasztottat kekre }
               textbackground(blue);
               textcolor(yellow);
               gotoxy(10,9+k);
               case k of
                1: write(' Elso ');
                2: write(' Masodik ');
                3: write(' Harmadik ');
               end;
               { megnezzuk melyik billentyut nyomta meg a felhasznalo
                es ettol fuggoen megvaltoztatjuk a k erteket }
               c:=readkey;
               case c of
                #72: if k>1 then dec(k); { #72 = felfele nyil }
                #80: if k<3 then inc(k); { #80 = lefele nyil }
               end;
               { atfestjuk az uj kivalasztottat pirosra }
               textbackground(red);
               textcolor(white);
               gotoxy(10,9+k);
               case k of
                1: write(' Elso ');
                2: write(' Masodik ');
                3: write(' Harmadik ');
               end;
               { a kurzort beallitjuk a jobb also sarokba }
               gotoxy(80,25);
               end;
  { ha #13 = Enter lett megnyomva }
  if c=#13 then begin
                gotoxy(10,15);
                textbackground(0);
                textcolor(7);
                writeln('Kivalaszottad a(z) ',k,'. menupontot!');
                gotoxy(80,25);
                end;
 until c=#27;
end.
```

A crt unit segítségével hangokat is adhatunk ki. Erre a **sound(** *frekvencia* **)** ill. a **nosound** parancsok szolgálnak. Pl.

```
program Pelda47;
uses crt;
begin
 clrscr;
 sound(800);
 delay(1000);
 sound(1200);
 delay(500);
 sound(1000);
 delay(500);
 nosound;
 readln;
end.
```

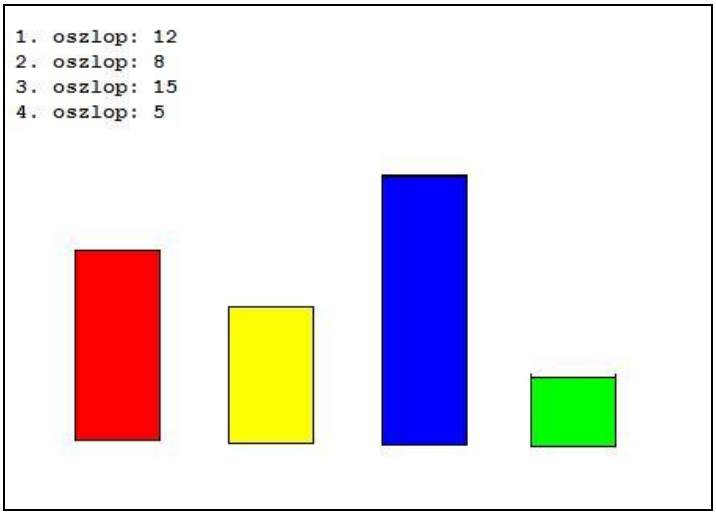
Feladat: Mozgassunk egy téglalapot (egy kis képernyőt) benne egy szöveggel a képenyőn a kurzormozgató billentyűk segítségével!

```
program Pelda48;
uses Crt;
var  x1, x2, y1, y2: byte;
     c: char;
begin
  x1:=35; y1:=12; x2:=45; y2:=16;
  repeat
    {A régi ablak törlése}
    TextBackground(black);
    ClrScr;
    {Új ablak és a szöveg megjelenítése}
    Window(x1, y1, x2, y2);
    TextBackground(blue);
    TextColor(red);
    ClrScr;
    GotoXY(3, 3);
    WriteLn('szoveg');
    {Várakozás egy billentyű leütésére}
    c := ReadKey;
    {Ha a billentyűzetnek két bájtos kódja van (az első bájt #0), a második bájt
beolvasása.
     Ilyenek a kurzormozgató billentyűk.}
    if c = #0 then
      begin
        c := ReadKey;
        case c of
          #72: begin Dec(y1); Dec(y2) end; {Felfele nyíl}
          #80: begin Inc(y1); Inc(y2) end; {Lefele nyíl}
          #77: begin Inc(x1); Inc(x2) end; {Jobbra nyíl}
          #75: begin Dec(x1); Dec(x2) end; {Balra nyíl}
        end;
      end
    {Kilépés ESC-re}
  until c = #27;
  NormVideo;           {Eredeti színek visszaállítása}
  ClrScr;
end.
```

Feladatok:

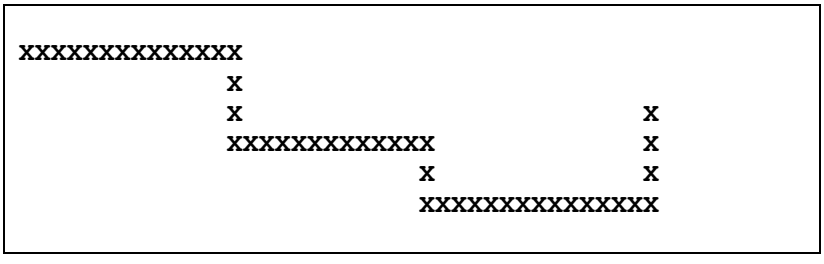
1. Készíts programot, melyben a teljes hátteret világosszürkére színezed! Aztán a minta alapján készítesz egy belső ablakot kék háttérszínnel! Majd készíts fehér betűkkel egy menürendszert: négyzet; téglalap; kör felirattal! Aztán, ha a kezdőbetűk valamelyikét lenyomjuk, akkor attól függően számoljon kerületet és területet!
Ha rossz karaktert ütünk be, akkor írja ki a „`Nem megfelelo karakter!`” szöveget, és enter lenyomására lépjen ki!
(crt01)

2. Készíts programot, melyben világosszürke háttérre rajzoljon ki 4 oszlopot, ahol a magasság változik a szélesség nem! (tehát négy változó érték kell)
Az oszlopok színe sorban: piros; sárga; kék; zöld legyen!
Az adatokat kérje be a program a minta alapján! A szöveg színe legyen fekete!
(crt02)



3. Bővítsd az előző programot úgy, hogy az oszlopok tetejére írja ki az értékeket!
(crt03)

4. Készíts programot, melyben a kurzormozgató billentyűk segítségével rajzolja ki „X” betű útvonalát a minta alapján! Világosszürke háttérrel! Az X-eknek minden lépésben véletlenszerűen váltja a színeket! Esc gombra lépjen ki!
(crt04)



5. Készítsünk képernyővédő animációt, mely kiír a képernyőre csillag karakterekből egy ábrát, majd egy kis idő eltelte után (500ms) ezt az ábrát letörli és kiírja egy másik véletlen pozícióra a képernyőn. **(crt05)**

16 Felsorolt típus, record, set típusok

- felsorolt típus
- **record** típus
- **set** (halmaz) típus

16.1 Felsorolt típus

A programozásban az eddig felsorolt típusokon kívül (integer, string, byte, boolean, char, array...) használhatunk saját adattípusokat is. Ezeket először a **type** utasítással meg kell adnunk. Az egyik ilyen típus a felsorolt típus, amely csak a felsorolt értékek valamelyikét veheti fel. Pl.

```
program Pelda49;
type nagybetu = 'A'..'Z';
     szamjegy = 0..9;
     nap = (hetfo, kedd, szerda, csutortok, pentek,
            szombat, vasarnap);
var a:nagybetu;
    b:szamjegy;
    n:nap;
begin
 { program }
end.
```

Ezeket a típusokat nem olvashatjuk be közvetlenül a **readln** paranccsal billentyűzetről és nem is írhatjuk ki az értéküket a **write** paranccsal a képernyőre. Beolvasásuk és kiírásuk a következőképpen történhet:

```
program Pelda50;
uses crt;
type nap = (hetfo, kedd, szerda, csutortok, pentek, szombat, vasarnap);
var n:nap;
    i:integer;
begin
clrscr; {beolvasas}
writeln('1 - hetfo');
writeln('2 - kedd');
writeln('3 - szerda');
writeln('4 - csutortok');
writeln('5 - pentek');
writeln('6 - szombat');
writeln('7 - vasarnap');
write('Addj meg egy szamot (1-7): ');
readln(i);
 case i of
  1: n:=hetfo;
  2: n:=kedd;
  3: n:=szerda;
  4: n:=csutortok;
  5: n:=pentek;
  6: n:=szombat;
  7: n:=vasarnap;
 end; {kiiras}
write('Az altalad megadott nap a ');
 case n of
      hetfo: writeln('hetfo.');
       kedd: writeln('kedd.');
     szerda: writeln('szerda.');
  csutortok: writeln('csutortok.');
     pentek: writeln('pentek.');
    szombat: writeln('szombat.');
   vasarnap: writeln('vasarnap.');
 end;
readln;
end.
```

16.2 Record típus

A **record** típus mezõkbõl áll, melyek egyedi változóként kezelhetők. Használatát példán szemléltetjük:

```
program Pelda51;
uses crt;
type szemely = record
                 nev: string;
                 szul_ev:integer;
               end;
var a:array [1..5] of szemely;
    i,jelen:integer;
function beolvas:szemely;
 var x:szemely;
 begin
  write('Nev: ');
  readln(x.nev);
  write('Szuletesi ev: ');
  readln(x.szul_ev);
  beolvas:=x;
 end;
begin
clrscr;
for i:=1 to 5 do
  begin
  writeln('Kerem az ',i,'. szemely adatait...');
  a[i]:=beolvas;
  end;
write('Milyen evet irunk most? (pl.2004) :');
readln(jelen);
for i:=1 to 5 do
  writeln(a[i].nev,' ',jelen-a[i].szul_ev,' eves.');
readln;
end.
```

16.3 Set (halmaz) típus

A halmaz típus (**set of ...**) használatát is egy példa segítségével mutatjuk be.

A feladatunk az volt, hogy egy beolvasott szövegben felasznált betűket írjuk ki ABC sorrendben, minden betűt csak egyszer és minden betűnek a nagybetűs formája legyen kiírva. A feladathoz definiáltunk egy halmazt, amelyet először beállítottunk üresre, majd a betűket sorban beleraktuk ebbe a halmazba. Végül egy **for** ciklus segítségével végignéztük, hogy az ABC melyik betűje van ebben a halmazban (**in**). Ha az adott betű benne volt a halmazban, kiírtuk a képernyőre.

```
program Pelda52;
uses crt;
const betu = ['A'..'Z','a'..'z'];
type nagybetu = 'A'..'Z';
var h:set of nagybetu;
    s:string;
    i:integer;
    ch:char;
begin
clrscr;
 write('Irj be egy modnatot: ');
 readln(s);
 h:=[];
 for i:=1 to length(s) do
   if s[i] in betu then h:=h+[upcase(s[i])];
 write('Felhasznalt betuk:');
 for ch:='A' to 'Z' do
   if ch in h then write(ch:2);
 writeln;
readln;
end.
```

Feladatok:

1. Készíts programot felsorolt adattípus felhasználásával, ahol megkérdezi a program, hogy melyik negyedév első hónapját írja ki; és az abban a negyedévben szereplő hónapok közül kiírja a képernyőre az elsőt! (Ha nem létezik olyan negyedév, akkor írja ki, hogy „Nem létezik ilyen negyedév!" **(tipus01)**

```
Melyik negyedev elso honapjat írjam
ki?
2

_____________________________________
A(z) 2. negyedev elso hónapja a(z)
aprilis
```

2. Készíts rekord típus felhasználásával programot, melyben bekér 4 időpontot (óra, perc, másodperc) , majd kiírja hogy éjfélig még mennyi idő van hátra. **(tipus02)**

```
Kerem a(z) 1. idopontot:
 Ora: 23
 Perc: 59
 Masodperc 59
Kerem a(z) 2. idopontot:
…

_________________________
A(z) 1. idopontol 0 ora 0 perc 1 masodperc van hatra
ejfelig.
A(z) 2. idopontol …
```

3. Készítsünk programot, mely egy 100 elemű rekord típusú tömbben tárolja a diákok neveit és adatait. Mindegyik diáknak jegyezzük meg a keresztnevét, vezetéknevét, születési évét és a jegyeinek átlagát (real típusban). A programban legyen lehetőség új diák adatainak bevitelére, vagy már egy meglévő diák törlésére a tömbből. Készíts egyszerű menüt a minta alapján! (Használj saját függvényeket a megoldáshoz!) **(tipus03)**

```
l-listazas | u-uj | t-torles | k-kilepes
valassz: u

Add meg az 1. diak adatait!
Vezeteknev: Kiss
Kereszetnev: Peter
Szuletes eve: 2002
Atlag: 3.54

_____________________________________
l-listazas | u-uj | t-torles | k-kilepes
valassz: u

Add meg az 2. diak adatait!
Vezeteknev: Nagy
Kereszetnev: Janos
Szuletes eve: 2003
Atlag: 4.11

_____________________________________
l-listazas | u-uj | t-torles | k-kilepes
valassz: t

Melyik rekordot toroljem (1-2): 2

_____________________________________
l-listazas | u-uj | t-torles | k-kilepes
valassz: l

A(z) 1. diak Kiss Peter, 2002-ben szuletett és az atlaga
3.54;
A(z) 2. diak   ,0-ben szuletett atlaga 0
```

4. Készítsünk programot lottószámok kigenerálására ügyelve arra, hogy véletlenül sem legyen kigenerálva ugyanaz a szám kétszer. Az éppen kigenerált lottószámot mindig adjuk hozzá egy halmazhoz. A következő lottószám generálásakor nézzük előbb meg, hogy ez a szám szerepel-e már a halmazban, ha nem, adjuk hozzá (ha igen, újat generáljunk ki helyette). Az öt szám kigenerálása (és a halmazba berakása) után írjuk ki a halmazból a lottószámokat növekvő sorrendben. **(tipus04)**

```
A generalashoz
nyomj ENTER-t.

_________________
A szamok novekvo
sorrendben:
12;34;55;61;68;
```

17 Állományok kezelése

- szöveges állomány
- típusos állomány
- típus nélküli állomány

17.1 Szöveges állomány

A programból elmenthetünk ill. beolvashatunk adatokat állományokból. A Pascal-ban három fajta állománytípust különböztethetünk meg: szöveges állomány, típusos állomány és típus nélküli állomány.

Az első programunkban egy szöveges állományt fogunk kiíratni a képernyőre. Ehhez előbb hozzunk létre egy ilyen állományt. Szöveges állományt megírhatjuk pl. a windows jegyzettömbjében. Írjunk tehát a jegyzettömbben pár sort, majd mentsük el a szöveget abba a könyvtárba, ahova a Pascal programjainkat is szoktuk menteni. A szöveget "szoveg.txt" néven mentsük el. Ezen előkészületek után indítsuk el a FreePascal-t. Az alábbi program segítségével kiírathatjuk az elmentett állományt a képernyőre:

```
program Pelda53;
uses crt;
var f:text;
    s:string;
begin
clrscr;
assign(f,'szoveg.txt');
{$I-}
reset(f);
{$I+}
if IOResult<>0 then begin
                    writeln('Hiba: nincs meg a file.');
                    readln;
                    halt;
                    end;
while not eof(f) do begin
                    readln(f,s);
                    writeln(s);
                    readln;
                    end;
close(f);
end.
```

Ha szöveges állományt használunk, azt a változók deklarálásánál **text** típusként kell deklarálnunk.

Az **assign(f,'szoveg.txt');** parancs hozzárendeli az **f** állományhoz a merevlemezen található **szoveg.txt** állományt.

Ezek után a **reset(f);** parancs megnyitja az állományt olvasásra. A **{$I-}** és **{$I+}** utasítások kikapcsolják/bekapcsolják a pascal hibaellenőrzését, tehát azt, hogy ha a file megnyitásánál hiba keletkezik, akkor azt ne a pascal értékelje ki, hanem mi. A megnyitás után az **IOResult** változó értéke **0**, ha sikerült megnyitni az állományt, különben a hiba kódját tartalmazza. Hiba esetén a **halt;** parancsnál befejeződik a programunk futása.

Ha sikerült megnyitni az állományt, akkor egy ciklus segítségével kiolvasunk belőle egy sort a **readln(f,s);** paranccsal, majd ezt kiírjuk a képernyőre. A ciklus ezt a két utasítást addig ismétli, amíg nem értünk az állomány végére - **not eof(f)**. (eof = end of file).

Végül a **close(f)** paranccsal bezárjuk az állományt.

Mindig, amikor a programunkban állományt használunk előbb az **assign** paranccsal hozzárendeljük a változót a merevlemezen található állományhoz. Utána a **reset** paranccsal megnyithatjuk az állományt olvasásra vagy a **rewrite** paranccsal felülírásra. Szöveges állományból olvashatunk a **read, readln**

parancsokkal, írhatunk bele a **write, writeln** parancsokkal. Típusos állományoknál a **readln** és **writeln** parancsokat nem használhatjuk. Miután befejeztük a munkánkat az állománnyal, ne felejtsük el bezárni a fájlt a **close** paranccsal.

A következő program azt mutatja be, hogyan írhatunk szöveges állományba. Miután ezzel a programmal beírtunk valamit az állományba, megnyithatjuk azt olvasásra az előző programmal vagy akár bármilyen szövegszerkesztővel (pl. windows jegyzettömb).

```
program Pelda54;
uses crt;
var f:text;
    s:string;
begin
clrcsr;
assign(f,'szoveg.txt');
rewrite(f);
s:='elso sor';
writeln(f,s);
writeln(f,'masodik sor');
write(f,'harmadik ');
writeln(f,'sor');
close(f);
readln;
end.
```

A **rewrite** parancs törli az állomány tartalmát ha már létezik, majd megnyitja írásra. Ha még nem létezik ez az állomány a merevlemezen, akkor létrehozza azt.

17.2 Típusos állomány

A típusos állományba bármilyen típusú változókat (akár általunk létrehozott rekordokat) menthetünk el. Ezt a fajta állományt ha megnyitnánk szövegszerkesztővel, nem egy egyszerű szöveget látnánk, hanem mindenféle jeleket is (akárcsak pl. egy futtatható állomány vagy kép megnyitásakor).

A következő program megnyit egy állományt (ha létezik), beolvassa belőle az adatokat egy tömbbe majd a program végén elmenti a tömbből az adatokat az állományba.

```
program Pelda55;
var f:file of string;
    a:array [1..1000] of string;
    i,n:integer; { n - a tombben levo elemek szamat jeloli }
begin
n:=0;
{beolvasas allomanybol}
assign(f,'nevsor.dat');
{$I-}
reset(f);
{$I+}
if ioresult=0 then begin
                   while not eof(f) do begin
                                       n:=n+1;
                                       read(f,a[n]);
                                       end;
                   close(f);
                   end;
{ ... itt dolgozunk a tömbbel, megvaltoztatjuk, stb ... }
{kiiras allomanyba}
rewrite(f);
for i:=1 to n do write(f,a[i]);
close(f);
end.
```

17.3 Típus nélküli állomány

Míg egy típusos állományba csak ugyanolyan típusú változókat írhattunk, addig a típus nélküli állományba tetszés szerint egymás után beírhatunk bármilyen típusú változókat, vagy akár a memória egy tetszőleges részét. Amire azonban ügyelnünk kell, hogy az adatokat ugyanolyan sorrendben olvassuk ki, ahogy beírtuk azokat.

A következő program létrehoz egy ilyen típus nélküli állományt, majd beleír egy string típust, ezután három integer típust, végül még egy real típust is. A program második része megnyitja az elmentett állományt olvasásra, kiolvassa az adatokat és kiírja a képernyőre.

```
program Pelda56;
var f:file;
    nev:string;
    ev,honap,nap:integer;
    r:real;
begin
 write('Kerlek add meg a neved: ');
 readln(nev);
 write('Szuletesi datumod - ev (pl. 1985): ');
 readln(ev);
 write('Szutetesi datumod - honap (1..12): ');
 readln(honap);
 write('Szuletesi datumod - nap (1..31): ');
 readln(nap);
 write('Kerlek adj meg egy tetszoleges tizedes szamot: ');
 readln(r);
 {mentes tipus nelkuli fajlba}
 assign(f,'adatok.dat');
 {$I-}
 rewrite(f,1);
 {$I+}
 if ioresult<>0 then begin
                     writeln('Hiba a fajl megnyitasanal!');
                     halt;
                     end;
 blockwrite(f,nev,sizeof(nev));
 blockwrite(f,ev,sizeof(ev));
 blockwrite(f,honap,sizeof(honap));
 blockwrite(f,nap,sizeof(nap));
 blockwrite(f,r,sizeof(r));
 close(f);
 {beolvasas a fajlbol}
 {$I-}
 reset(f,1);
 {$I+}
 if ioresult<>0 then begin
                     writeln('Hiba a fajl megnyitasanal!');
                     halt;
                     end;
 blockread(f,nev,sizeof(nev));
 blockread(f,ev,sizeof(ev));
 blockread(f,honap,sizeof(honap));
 blockread(f,nap,sizeof(nap));
 blockread(f,r,sizeof(r));
 close(f);
 {adatok kiirasa a kepernyore}
 writeln('Fajlbol visszaolvasott adatok:');
 writeln('Nev: ',nev);
 writeln('Szul.dat.: ',ev,'.',honap,'.',nap,'.');
 writeln('Tizedes szam: ',r);
end.
```

A típus nélküli állomány deklarálásánál a **file** kulcsszót kell használnunk.

Az állományt a **reset** ill. **rewrite** paranccsal nyithatjuk meg olvasásra ill. írásra. Itt a **reset** ill. **rewrite** parancsok második paramétereként (ezt az előző fájltípusoknál nem adtuk meg) meg kell adnunk egy blokk méretét. Ezt célszerű 1 bájtra adni, ahogy a mintapéldánkban is tettük.

Ezek után a típus nélküli állományból a **blockread** paranccsal olvashatunk, írni pedig a **blockwrite** paranccsal írhatunk. Ezeknek a parancsoknak az első paramétere maga a fájl, a második paraméter a fájlba beírandó vagy kiolvasandó változó neve (ill. mutató egy memóriacímre), a harmadik paraméter pedig egy szám, amely megadja, hogy mennyi blokkot akarunk olvasni ill. írni a fájlba (a második paraméterben megadott memóriacímtől kezdődően). Mi a példaprogramban a harmadik paraméter megadásánál a **sizeof** függvény segítségével megállapítottuk, hogy mennyi bájton van tárolva az adott változó a memóriában és ennyi blokkot olvastunk ill. írtunk a fájlba (mivel a mi esetünkben 1 block = 1 bájt). **seek**(file,pozíció) direkt file-ban a megadott sorszámú rekordra áll (file elején 0). **filepos**(file) függvény, megadja, hogy a direkt file hányadik rekordjánál tartunk.

Végül a fájlt a **close** paranccsal be kell zárnunk, hasonlóan mint a többi fájltípusnál.

Feladat:
A következő példaprogram egy típusos állomány kezelésére készült. A program egy bolt árucikkeinek adatait (név, kód, ár) tárolja és kezeli egy állományban. Hozzunk létre egy menüt. A peldaprogramban alkalmazzunk függvényeket, melyeket csak meghívunk a főprogramban.

```
1. Adatbevitel
2. Modositas
3. Torles
4. Listazas
5. Vege
Valassz!
```

Megoldás:

```
program Pelda57;
uses crt;
type TAru = record        {A fajl alaptipusa.}
         kod: string;
         nev: string[15];
         ar: real;
         t: boolean;
      {Ez a mezo jelzi, hogy e rekord torolt-e (logikai törlés).}
       end;

var bolt: file of TAru;
    aru: TAru;
    mkod: string;
    mvalasz: char;

{Megkeres egy adott kodu rekordot az allomanyban.}
function Van(kodja: string): boolean;
var talalt: boolean;
begin
  seek(bolt,0);
  talalt := false;
  while not Eof(bolt) and not talalt do
    begin
      read(bolt, aru);
      if (aru.kod = mkod) and not aru.t then
         talalt := true;
     end;
  van := talalt;
end;
```

```
{Egy rekord felvitele az allomanyba.}
procedure Bevitel;
begin
  ClrScr;
  WriteLn('Kerem a kodot!');
  ReadLn(mkod);
  if not Van(mkod) then
    begin
      Seek(bolt, filesize(bolt));
          {Pozicionalas a fajl vegere.}
      WriteLn('Kerem az aru nevet!');
      ReadLn(aru.nev);
      WriteLn('Kerem az aru arat!');
      ReadLn(aru.ar);
      aru.t := false;
      aru.kod := mkod;
      Write(bolt, aru);
    end
  else
    begin
      WriteLn('Mar van ilyen kod!');
      ReadKey
    end
end;

{Egy rekord modositasa a fajlban.}
procedure Modosit;
begin
  ClrScr;
  WriteLn('Kerem az aru kodjat!');
  ReadLn(mkod);
  if Van(mkod) then
    begin
      Seek(bolt, FilePos(bolt) - 1);
      WriteLn('Kerem az aru nevet!');
      ReadLn(aru.nev);
      WriteLn('Kerem az aru arat!');
      ReadLn(aru.ar);
      aru.t := false;
      aru.kod := mkod;
      Write(bolt, aru);
    end
  else
    begin
      Writeln('Nincs ilyen aru!');
      ReadKey
    end;
end;
```

```
{Egy rekord logikai torlese: a t mezot True ertekure allitja,
 az ilyen rekordokat a program nem letezonek tekinti.
 Fizikai torles kilepeskor.}

procedure Torles;
begin
  ClrScr;
  WriteLn('Kerem az aru kodjat!');
  ReadLn(mkod);
  if Van(mkod) then
    begin
      Seek(bolt, FilePos(bolt) - 1);
      aru.t := true;
      Write(bolt, aru);
    end
  else
    begin
      WriteLn('Nincs ilyen aru!');
      ReadKey
    end
end;

{A fajl tartalmanak kiirasa a kepernyore.}
procedure Lista;
begin
  ClrScr;
  Seek(bolt, 0);
  while not Eof(bolt) do
    begin
      Read(bolt, aru);
      if aru.t = false then
        begin
          Write(aru.kod);
          GotoXy(30, wherey); write(aru.nev);
          GotoXy(60, wherey); writeln(aru.ar:10:0);
        end;
    end;
   ReadKey
end;

{Fizikai torles: azon rekordok atmasolasa egy uj allomanyba, melyek nincsenek
logikailag torolve.
 A regi áalomany torlese, az uj fajl atnevezese a regi nevure.}
procedure Surites;
var ujfile: file of TAru;
begin
  Assign(ujfile, 'ujfile');
  Rewrite(ujfile);
  Seek(bolt, 0);
  while not Eof(bolt) do
    begin
      Read(bolt, aru);
      if aru.t = false then write(ujfile, aru);
     end;
  Close(bolt);
  Erase(bolt);
  Close(ujfile);
  Rename(ujfile, 'bolt');
end;
```

```
{Foprogram, menu.}
begin
  clrscr;
  Assign(bolt, 'bolt');
  {$I-}
  Reset(bolt);
  {$I+}
  if IOResult <> 0 then Rewrite(bolt);
  repeat
    ClrScr;
    WriteLn('1. Adatbevitel');
    WriteLn('2. Modositas');
    WriteLn('3. Torles');
    WriteLn('4. Listazas');
    WriteLn('5. Vege');
    WriteLn('Valassz!');
    repeat mvalasz := readkey until mvalasz in['1'..'5'];
    case mvalasz of
      '1': bevitel;
      '2': modosit;
      '3': torles;
      '4': lista;
      '5': surites;
    end;
  until mvalasz = '5';
end.
```

Feladat: Az a.dat állomány integerekből épül fel, írasd ki a számtani átlagukat! (Ahhoz, hogy ez a program működjön, létre kell hozni az a.dat-ot!)

Megoldás:

```
program Pelda58;
uses crt;
var f:file of integer;
    ossz,x:integer;
begin
  clrscr;
     assign(f,'a.dat');
     reset(f);
     while not eof(f) do
     begin
          read(f,x);
          ossz:=ossz+x;
     end;
     Writeln('Atlag: ',ossz/filesize(f):10:2);
  readln;
end.
```

Feladatok:

1. Készíts programot, mely bekér egy mondatot, aztán ezt mondatot eltárolja egy szövegfájlban (szoveg.txt), utána pedig kiírja a szöveges állomány tartalmát a képernyőre! **(allom01)**

```
Mit szeretnel beirni a fajlba?
Ide ird be: Ezt szeretnem kiiratni!

A szoveges fajl tartalma: Ezt
szeretnem kiiratni!
```

2. Készítsünk programot, amely beolvas egy szöveges állományt (szoveg01.txt), majd egy másik állományba (szoveg02.txt) kiírja a beolvasott szöveget csupa nagybetűkkel. **(allom02)**

```
Ezt a szoveget kell atiratni masik
fajlba nagybetuvel!

EZT A SZOVEGET KELL ATIRATNI MASIK
FAJLBA NAGYBETUVEL!
```

3. Készítsünk programot, amely kiír egy menüt:

1 nevsor kiirasa
2 uj diak hozzaadasa
3 diak torlese
0 kilepes a programbol

majd a választott menüpont alapján végrehajtja az adott műveletet utána ismét kiírja ezt a menüt. A diákokról tároljuk a nevüket, születési évüket és a nemüket (fiú, lány). A diákokat a programban egy 1000 elemű tömbben tároljuk. A program elején a tömböt olvassuk be egy állományból (ha létezik már az állomány), majd a program végén mentsük ki állományba.(szoveg03.txt) **(allom03)**

```
1-istazas | 2 uj | 3-torles | 0-kilepes
valassz: 2

Add meg az 1. diak adatait!
Vezeteknev: Kiss Peter
Szuletes eve: 2002
Neme: ferfi

_____________________________________
1-istazas | 2 uj | 3-torles | 0-kilepes
valassz: 2

Add meg az 2. diak adatait!
Vezeteknev: Nagy Janos
Szuletes eve: 2003
Neme: ferfi

_____________________________________
1-istazas | 2 uj | 3-torles | 0-kilepes
valassz: 3

Melyik rekordot toroljem (1-2): 2

_____________________________________
1-istazas | 2 uj | 3-torles | 0-kilepes
valassz: 1

A(z) 1. diak Kiss Peter, 2002-ben szuletett a
neme ferfi
 A(z) 2. diak   ,0-ben szuletett atlaga 0
 …
1-istazas | 2 uj | 3-torles | 0-kilepes
valassz: 0
 fajlbairas…
```

4. A szoveg04.txt állományban egy feladatsor van, kérdések és válaszok felváltva egymás után. Minden kérdés illetve válasz új sorban kezdődik. A kérdések egy számjeggyel kezdődnek, és kérdőjellel fejeződnek be. Készítsünk két új szöveges állományt úgy, hogy az egyik csak a kérdéseket, a másik pedig csak a válaszokat tartalmazza. (kerdes.txt; valasz.txt) **(allom04)**

```
Nyomj ENTER-t hogy lefusson a
program!
```

5. Hozz létre egy szam.dat állományt, írasd bele a prímeket 1000-ig, és irasd ki a képernyőre is! **(allom05)**

```
2   3   5   7  11  13  17  19  23
29  …
                   983  991  997
```

18 Dos unit, rendezési algoritmusok

- dos unit
- rendezési algoritmusok

18.1 Dos unit

A dos unit segítségével többek között lekérhetjük az aktuális dátumot és idõt. A dátumot a **getdate** eljárás segítségével kérhetjük le. A parancsnak négy paramétere van. Ebbe a négy paraméterként megadott változóba adja vissza az eljárás az aktuális évet, hónapot, napot és azt, hogy a hét melyik napja van éppen (0-vasárnap, 1-hétfõ, ...).

Hasonlóan működik az idõ lekérésére is a **gettime** eljárás. Paraméterként itt is négy változót kell megadnunk, melyekben visszakapjuk az aktuális órát, percet, másodpercet és századmásodpercet. Ennek az eljárásnak a segítségével fogjuk késõbb lemérni az egyes rendezési algoritmusok futásának idõtartamát.

```
program Pelda59;
uses dos crt;
var d1,d2,d3,d4,t1,t2,t3,t4:word;
begin
clrscr;
 getdate(d1,d2,d3,d4);
 gettime(t1,t2,t3,t4);
 write('Mai datum: ',d1,'. ',d2,'. ',d3,'. ');
 case d4 of
  0: writeln('vasarnap');
  1: writeln('hetfo');
  2: writeln('kedd');
  3: writeln('szerda');
  4: writeln('csutortok');
  5: writeln('pentek');
  6: writeln('szombat')
 end;
 writeln('Ido: ',t1,':',t2,':',t3,'.',t4);
readln;
end.
```

18.2 Rendezési algoritmusok

Adott egy tömb, melyben véletlenszerű számok vannak. Feladatunk, hogy rendezzük ezeket a számokat növekvõ sorrendbe különbözõ algoritmusok segítségével. A rendezés idejét minden esetben mérjük le századmásodpercekben.

Egyszerű rendezés (SimpleSort)

Ez a programilag legegyszerűbb rendezési algoritmus. Mindegyik elemet összehasonlítjuk az összes utána következő elemmel két egymásba ágyazott ciklus segítségével. Ha az éppen összehasonlított két elem nem jó sorrendben van, akkor azokat rögtön ki is cseréljük.

Rendezés a legkisebb elem kiválasztásával (MinSort, SelectSort)

A módszer lényege, hogy az i. helyre (i = 1,2,3,...) sorban kiválasztjuk az i...elemekszama helyen levők közül a legkisebbet.

Rendezés a szomszédos elemek cseréjével (BubbleSort)

A módszer lényege, hogy egyesével végignézzük a szomszédos elemeket, és ha elõbb van a nagyobb, megcseréljük azokat. Így biztosan a legnagyobb helyre kerül a legnagyobb elem ("a buborék felszáll"). Ugyanezt sorban végrehajtjuk az utolsó elõtti elemig stb., egészen az elsõig. Ha közben kiderül, hogy a tömb rendezett, az eljárást befejezzük.

Rendezés beszúrásos módszerrel (InsertSort)

A módszer hasonlít a kártyák kézben való rendezéséhez. Mindig vesszük a következõ elemet, s azt a megfelelõ helyre beszúrjuk úgy, hogy a többi, már rendezett elemet eggyel feljebb toljuk.

Feladat: Mindegyik rendezés pontos algoritmusa megtalálható az alábbi programban:

```
program Pelda60;
uses crt, dos;
const elemekszama = 10000;
var a,b:array [1..elemekszama] of integer;
    ido:longint;
     i:integer;

procedure kiiras;
var i:integer;
begin
 for i:=1 to elemekszama do write(a[i],' ');
 writeln;
 writeln;
end;

procedure stopperbe;
var t1,t2,t3,t4:word;
begin
 gettime(t1,t2,t3,t4);
 ido:=t4+100*(t3+60*(t2+60*t1));
end;

procedure stopperki;
var t1,t2,t3,t4:word;
begin
 gettime(t1,t2,t3,t4);
 ido:=t4+100*(t3+60*(t2+60*t1))-ido;
end;

{ egyszeru rendezes }
procedure simplesort;
var i,j,x:integer;
begin
 for i:=1 to elemekszama-1 do
  for j:=i+1 to elemekszama do
     if a[i]>a[j] then begin
                        x:=a[i];
                        a[i]:=a[j];
                        a[j]:=x;
                       end;
end;

{ rendezes a legkisebb elem kivalasztasaval }
procedure minsort;
var i,j,min,x:integer;
begin
 for i:=1 to elemekszama-1 do
  begin
   min:=i;
   for j:=i+1 to elemekszama do
     if a[j]<a[min] then min:=j;
   x:=a[i];
   a[i]:=a[min];
   a[min]:=x;
  end;
end;
```

```
procedure bubblesort; { rendezes a szomszedos elemek cserejevel }
var i,j,x:integer;
    ok:boolean;
begin
 i:=elemekszama-1;
 ok:=false;
 while (i>=1) and (not ok) do
  begin
   ok:=true;
   for j:=1 to i do
     if a[j]>a[j+1] then begin
                         x:=a[j];
                         a[j]:=a[j+1];
                         a[j+1]:=x;
                         ok:=false;
                         end;
   dec(i);
  end;
end;
procedure insertsort; { rendezes beszurasos modszerrel }
var i,j,x,ment:integer;
begin
 for i:=2 to elemekszama do
  begin
   if a[i]<a[i-1] then begin
                       ment:=a[i];
                       j:=i;
                       repeat
                        dec(j);
                        a[j+1]:=a[j];
                       until (j=1) or (a[j-1]<=ment);
                       a[j]:=ment;
                       end;
  end;
end;
begin { foprogram }
 clrscr;
 writeln(elemekszama,' drb. elem rendezese:');
 writeln;
 randomize;
 for i:=1 to elemekszama do b[i]:=random(64000)-32000;
 { mindig a-t fogjuk rendezni, b-ben meghagyjuk a kigeneralt szamokat}
 a:=b;
 stopperbe;
 simplesort; { rendezes }
 stopperki;
 writeln('SimpleSort: ',ido,' szazadmasodperc'); { kiiras }
 a:=b;
 stopperbe;
 minsort; { rendezes }
 stopperki;
 writeln('MinSort: ',ido,' szazadmasodperc'); { kiiras }
 a:=b;
 stopperbe;
 bubblesort; { rendezes }
 stopperki;
  writeln('BubbleSort: ',ido,' szazadmasodperc'); { kiiras }
 a:=b;
 stopperbe;
 insertsort; { rendezes }
 stopperki;
 writeln('InsertSort: ',ido,' szazadmasodperc'); { kiiras }
end.
```

Feladatok:

1. Készítsünk képernyővédő animációt, amely minden másodpercben a képernyőn véletlen helyre kiírja a dátumot és a pontos időt, majd egy másodperc után eltünteti azt. **(dos01)**

```
20XX.XX.XX. pentek
15:30:22:13


        20XX.XX.XX. pentek
        15:30:23:14
```

2. Írasd ki két billentyű lenyomása alatt eltelt idõt! **(dos02)**

```
Nyomj le egy billentyut, majd meg
egyet!
Az eltelt ido: 85 szazadmasodperc.
```

3. Készítsünk programot, melyben generál egy 100 elmű tömbe 0-1000 közötti számokat. Majd sorbarendezi valamelyik rendező algoritmussal. Utána pedig kimenti egy szamok.txt szövegfájlba. **(rend01)**

```
Rendezetlen: 453; 761; 33; ..


Rendezve: 12; 33; …
```

4. A nev.txt text típusú állományban nevek vannak soronként (ha nincs ilyen fájl, akkor hozzál létre), írd át úgy, hogy rendezve legyenek! (max. 10 névvel) **.(rend02)**

```
Agoston Eszter
Horvath Eva
Horvath Zoltan
Kovacs Peter
…
Zelk Sandor
```

5. Készítsünk programot, amely rendezi egy rekord típusú tömbben levő diákok neveit növekvő sorrendbe. A tömb tartalmazza a diákok nevein kívül a születési évüket és születési helyüket is. **(rend03)**

```
1 ADATBEVITEL | 2 LISTAZAS | 0 KILEPES
Valassz: 2
Abert Tamas     - 1998 - Szombathely
Andor Eva       - 2001 - Budapest
Belso Peter     - 2003 - Sarvar
…
Zente Zsolt     - 1999 - Szombathely
```

19 Összefoglaló, fogalmak

- program nev; {a program indítása, nevének megadása}
- var {változók deklarálása}
- begin {főprogram, alprogramrész kezdete}
- end; end. {alprogram vége; főprogram vége}
- clrscr {képernyőtörlés}
- write; writeln; {kiíratás a képrnyőre; kiíratás a képernyőre sortöréssel)
- mod {két egész szám osztásának maradéka}
- div {két szám hányadosának egész része}
- read; readln; {beolvasás billentyűzetről}
- integer {egész típus}
- string {szöveg}
- real {valós szám}
- boolean {igen; nem;}
- byte {0-255 egész szám}
- char {egy karakter}
- sqr() {négyzetreemelés}
- sqrt() {gyökvonás}
- for do {ciklus}
- downto {visszafelé számol}
- length() {a változó hossza}
- upcase() {nagybetűssé teszi}
- chr() {egy karakter kiíratása az ASCII kód alapján}
- ord() {egy karakter ASCII kódjának kiíratása}
- inc() {függvényt (increase); egyel növeli az értéket}
- dec() {egyel csökkenti az értéket}
- and; or; not; xor; {logikai műveletek}
- if then {ciklus}
- case of {ciklus}
- while do {elöltesztelő ciklus}
- repeat until {ciklus}
- randomize; {véletlen szám generálásához meg kell adni a program elején}
- random() {véletlen szám generálása}
- array of {tömbök megadása}
- const {állandók megadása}
- procedure {eljárás}
- function {függvény}

- delay() {késleltetés ms –ban)
- gotoxy() {pozícióba ugrás)
- keypressed {boolean változó}
- readkey {karakter beírására vár}
- textbackground {hátérszín megadása}
- textcolor {betűszín megadása}
- window {belső ablak rajzolása}
- sound(); nosound; {hang be- és kikapcsolása}
- abs() {abszolút érték}
- type {típus megadása}
- record {record típus}
- set of {halmaz típus}
- text {szöveges állományok deklarálása, text típusként}
- assign {a parancs hozzárndel egy állományhoz egy fájlt}
- reset {megnyitja az állományt olvasásra}
- rewrite {megnyitja az állományt írásra}
- halt {befejeződik a program futása}
- eof {end of file – a file végére}
- close {bezárjuk az állományt}
- file {típus nélküli állomány deklarálása}
- blockread {típus nélküli állományból olvasás}
- blockwrite {típus nélküli állományba írás}
- sizeof {függvény – mennyi bájton van tárolva az adott változó}
- seek {ugrás megadott fájlon bellüli recordra ugrik}
- filepos **{**függvény - megadja, hogy a direkt file hányadik rekordjánál tartunk}
- getdate {eljárás – aktuális idátum lekérése}
- gettime {eljárás – aktuális idő lekérése}

Forrás: Internet ()