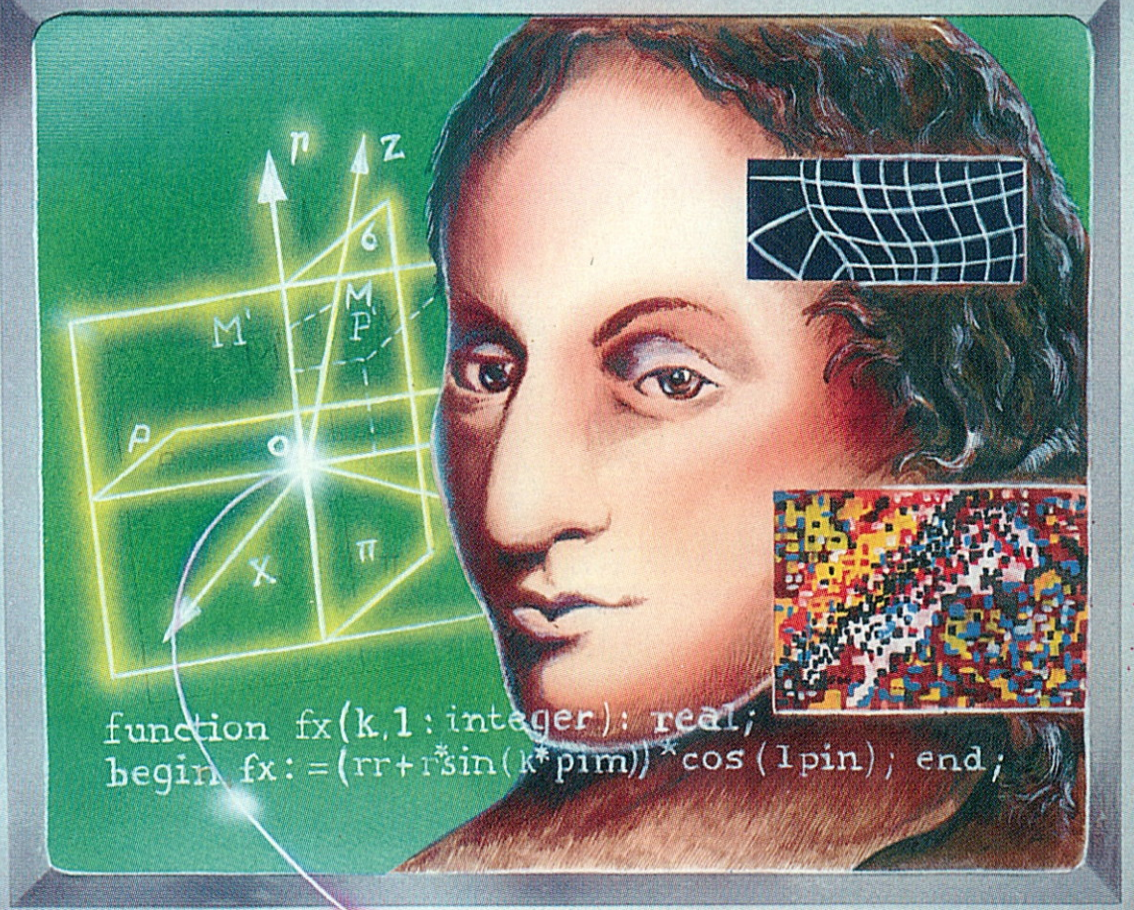
TUDOMÁNY ÉS MŰVELTSÉG

THOT

TURBO PASCAL

Kasza János



Programozási ismeretek 12–16 éveseknek

TURBO PASCAL

Programozási ismeretek 12–16 éveseknek

Tóth Könyvkereskedés és Kiadó Kft.

TÓTH KÖNYVKERESKEDÉS ÉS KIADÓ KFT.

Felelős vezető: **Tóth Csaba**
Fedélterv: **Tóth Könyvkereskedés és Kiadó Kft.**
Írta: **Kasza János**
Lektorálta: **Simon Gyula**

Tóth Könyvkereskedés és Kiadó Kft.
4034 Debrecen, Huszár Gál u. 31–33. sz.
Telefon: 06 (52) 450-861, 450-862
06-30-358-569
E-mail: tothbook@mail.matav.hu

Nyomdai munkálatok: START Rehabilitációs Vállalat és Intézményei
Nyírségi Nyomda Üzeme – 606
Felelős vezető: Balogh Zoltán vezérigazgató

BEVEZETŐ

Már régóta gondolkodtam ennek a könyvnek a megírásán. Ami visszatartott egyik oka az volt, hogy nagyon sok könyv megjelent már ebben a témában. Eddigi tapasztalataim azt mutatták, hogy a diákok nehezen indulnak a programozás néha rögös, de mégis sok örömet, sikerélményt nyújtó útján.

A könyv logikai felépítése eltérő sok hasonló társától. Egy fejezetben nem merít ki egy témát, hanem sok-sok feladaton keresztül koncentrikusan bővíti a fogalmakat. Fokozatosan nehezedő példákon keresztül szinte kézenfogva ismerkedhet meg az olvasó a programozás folyamatával. Törekedtem arra, hogy mondanivalómat érdekes példákkal illusztráljam és ezáltal mindenki számára közérthetővé tegyem. A könyv Kb. 70 db mintaprogramot tartalmaz. Csak a leglényegesebb és legfontosabb ismereteket tartalmazza, hiszen az oldalszám eleve korlátokat szabott.

A programozási feladatoknál a nyelv csak egy eszköz. A Pascal csak az egyik a sok jó tulajdonságú nyelvek között. A problémamegoldás kezdete mindig a feladathoz szükséges összefüggések meghatározása. A következő lépésben megvizsgáljuk az adott eszköz (nyelv) által kínált lehetőségeket és alkalmazzuk azt a feladat megoldására. Ha a nyelvvel kapcsolatos ismereteink bővebbek, akkor kiválaszthatjuk a probléma jellegéhez legjobban alkalmazhatót. Az általam ismertetett Pascal nyelvben is mutatok be olyan megoldási lehetőségeket, melyek nem minden programnyelvnek ismérvei (pl.: halmazok, ciklusváltozó lehetséges típusai, tömbindexek tartományai, stb.).

A könyvet használhatják a programozással most ismerkedők, de azoknak is nyújthat új ismereteket, adhat ötleteket, akik a nyelvet már alapfokon ismerik. Ajánlom a 12-16 éves korosztály számára otthoni böngészésre, illetve iskolákban az érdeklődő diákoknak szakköri feldolgozásra. A számítástechnika tanárok is hasznosan forgathatják, segítheti őket iskolai munkájukban.

a szerző

TARTALOMJEGYZÉK

Fontosabb számítástechnikai alapfogalmak

A XX. század fordulópontot jelentett a tudományok fejlődésében. A tudományok és a technika fejlődésével ugrásszerűen nőtt az ismeretek, információk mennyisége. Ezért felvetődött az a probléma, hogy a felhalmazódott információ tömeget hogyan lehet a leghatékonyabban összegyűjteni, tárolni és feldolgozni. **Az információk összegyűjtésének és feldolgozásának leghatékonyabb eszköze az elektronikus számítógép.** A számítógép tehát olyan technikai eszköz, **amely alkalmas nagy mennyiségű adatok, információk gyors feldolgozására.**

1. Az információ

Egy példa. Gondold el, hogy a Nagyerdei strandon várod egy barátnődet a bejáratnál, aki késésben volt. A találkozás 8 órára volt megbeszélve. A hangos bemondó állandóan információk tömegét közli pl.: a hullám fürdő öt perc múlva üzemel, Tóth Gyöngyit várják a Klinika felőli kapunál, a gyógymedencében labdázni tilos, szabadon hagyott értéktárgyaikért nem vállalunk felelőséget, a gyerekmedencében 6 éven aluliaknak fürödni tilos stb. Figyelted a bemondó információit? Biztosan nem. Hiszen számodra sokkal fontosabb jó ismerősöd felbukkanása. Késői érkezése megnyugvással töltött el. A késésre biztosan reagáltál valahogy (műveletvégzés).

Másképpen figyelted volna a bemondó közléseit, ha a barátnőd azt mondja: nem tudom ráérek-e. Ha elmegyek, akkor az információs stúdióból jelzem érkezésemet.

1.1. Az információ, adat fogalma

Az információ pontos fogalmát elég nehéz megfogalmazni.

Információnak nevezhetünk minden olyan **érzékelést, amelynek hírértéke van.** Az információ tehát **bizonytalanságunkat csökkenti.** A „leégett'' szó sok mindent jelenthet. Jelentése függ az adott helyzettől, ahol az elhangzott. A csengetés hangja (hangfrekvenciák sorozata), mást jelent az iskolában (szünet) és megint mást egy lakásban.

📖 **Az információ értéke egy üzenet tartalmi jelentése. Ha megfosztjuk tartalmi jelentésétől, akkor egy jelsorozatot kapunk. Ezt a jelsorozatot adatnak nevezzük.**

Az adat megjelenési formája sokféle lehet. Pl. Hang, kép, írott, nyomtatott forma, elektromos, mágneses jelek sorozata stb. Elemi adatnak nevezzük az információ megjelenési formájának azt részét, amely már **további egységekre nem bontható (Pl.: Bit).**

1.2. Az információ továbbítása

Ahhoz, hogy az információt továbbítani lehessen megfelelően át kell alakítani. Ezt a folyamatot nevezzük **kódolásnak és dekódolásnak.** A kódolás minden továbbítás nélkülözhetetlen része. Ti is használjátok az információtovábbítás legegyszerűbb eszközét, a telefont. Milyen lépésekre bontható egy telefonbeszélgetés?

- A kigondolt szavakat az ember az agy és beszélőszervek segítségével **átalakítja**, vagyis **kódolja.** Ezáltal a szavak, a hangszálak rezgése útján hallhatóvá válnak.
- A telefonban lévő mikrofon ezeket a hangokat elektromos jelekké alakítja. Tehát **újra kódolás** történik.
- A beszélőpartnerünk telefonkagylójában lévő hangszóró **visszaalakítja, dekódolja** az elektromos jeleket hangokká.
- A vonal másik végén lévő ember a hangokat újra gondolatokká alakítja, tehát szintén **dekódolja** azt .

↘ **Az információtovábbítás leegyszerűsített modellje**

Információ	kódolás	adó	csatorna	vevő	dekódoló	felhasználó

tárolás **zaj** tárolás

A rendszerben a **csatorna szolgál** a kódolt, illetve dekódolt **információk továbbítására.** A továbbítást zavarhatja, ronthatja a **csatorna zaj**. Ennek mértékét egy elemi adatra szokás kifejezni. Ha a modellt kiegészítjük olyan eszközökkel, melyek képesek az adatok tárolására és vezérlésére valamint a feldolgozott adatok megjelenítésére, akkor megközelítjük a számítógép működésének lényegét. A számítógépek kialakulásának kezdeti időszakában a csatornák adatátviteli sebessége lassú volt. Nagyarányú fejlődés az utóbbi évtizedben történt.

1.3. Kódolás, kódrendszer

Az információ átalakításához, kódolásához szükségünk van egy **jelkészletre.** Egy írásos közlemény esetében a jelkészletet az illető nyelv betűi alkotják. A Morse távíró esetében az információt két különböző jel (rövid és hosszú) sorozata jelentette. **A jelkészletet, melyből összetett információt állíthatunk össze, kódábécének nevezzük**. A kódolásához még szükségünk van szabályokra, melyeknek segítségével az adat egyértelműen kifejezhető. **A szabályrendszert és kódábécét együttesen kódrendszernek nevezzük.**

✍ Ennek megértésére nézzünk két egyszerű játékos példát! Meddig tudnál elszámolni nyolc db. gyufaszállal. Ha valaki nem gondolkodik, azonnal azt feleli, hogy **nyolcig!** Induljunk ki abból, hogy **egyesével** számolunk és minden számlálást úgy jelölünk, hogy valamely **„vízszintesen"** fekvő gyufaszálat **„függőlegesre"** állítjuk!

A **0** jelenti vízszintes, az **1** pedig a függőleges: gyufaszállakat:

számolás	K	Ó	D	O	L	Á	S
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	0	1	0
3	0	0	0	0	0	1	1
4	0	0	0	0	1	0	0
5	0	0	0	0	1	0	1
6	0	0	0	0	1	1	0
7	0	0	0	0	1	1	1
8	0	0	0	1	0	0	0
9	0	0	0	1	0	0	1
10	0	0	0	1	0	1	0
11	0	0	0	1	1	0	1
12	0	0	0	1	1	0	0

Ebben a példában a **kódábécénket két elemi adat**, a gyufák helyzete jelenti. Folytasd önállóan! Hogyan lehetne megállapítani a szabályt? Minden lépés elvégzéséhez figyeld meg az előző állapotot!

A kódrendszer megfogalmazásához még meg kell határoznunk az eljárás szabályát!

A szabály egy lehetséges megfogalmazása:

Minden gyufa vízszintes helyzetben van.
Állítsd fel jobbról számítva az elsőt!
Csináld
  Ha a felállított gyufaszáltól jobbra vannak függőleges gyufák,
  akkor tedd azokat vízszintes helyzetbe!
Amíg minden gyufa függőleges helyzetű.

A folyamat véges számú lépésben befejeződik. Például **2** gyufa esetén **háromig**, **3** esetén **hétig**, **4** esetén **tizenötig**, **5** esetén 32–1 = **harmincegyig** lehet elszámolni.

2. Számrendszerek

Ahhoz, hogy jobban megértsük a számítógép működését, foglalkoznunk kell ezzel a témával is. A későbbiekben e fogalom ismerete segítségünkre lesz a programozásban is.

2.1. Tízes számrendszer

A **tizes** számrendszerbeli számokat idegen szóval **decimális** számoknak nevezik (deci jelentése: valaminek a tízszerese). A **kódábécé: (0; 1; 2; 3; 4; 5; 6; 7; 8; 9).** Az ennél nagyobb mennyiségeket számjegyek sorozatával írtátok le. Minden számjegynek meghatározott helye van ebben a sorozatban. Ezért beszélhetünk a számjegyek **helyiértékéről** is. A helyiértékek balról jobbra haladva mindig a tízszeresére nőnek!

A helyértékek egész számok esetén

1	10	100	1000	10000	100000
10^0	10^1	10^2	10^3	10^4	10^5

📖 **Decimális számok esetén tíz db. számjegy van (0; 1 ;2; 3; 4; 5; 6; 7; 8; 9). A helyérték pedig balról jobbra haladva mindig tízszeresére növekszik. A számrendszer alapszáma a (10). A helyiérték az alapszám hatványai.**

✍ Hogyan írható fel az **alapszám és helyértékek** segítségével tízes számrendszerben például a $436_{(10)}$ és $2918_{(10)}$?

$436_{(10)} = 4*100 + 3*10 + 6*1 = 4*10^2 + 3*10^1 + 6*10^0$

$2918_{(10)} = 2*1000+9*100+1*10+8*10=2*10^3+9*10^2+1*10^1+8*10^0$

2.2. A kettes, bináris számrendszer

Hogyan „ érti „ meg a számítógép az ember (felhasználó) által készített programok utasításait? A számítógép számára fizikai felépítése alapján, legkönnyebben értelmezhető olyan állapot melynek két lehetséges értéke van. Két állapotot például kifejezhetnek a következő ellentett párok:

- **hideg** vagy meleg
- **igaz** vagy hamis
- **pozitív** vagy negatív
- **fej** vagy írás
- **van feszültség** vagy nincs feszültség
- **0** vagy 1
- **van lyukasztás** vagy nincs lyukasztás
- stb.

Egy **fizikai jel megléte jelenti az 1-et, hiánya pedig a 0-át.** A számítástechnikában ezt a **BINARY DIGIT (két állapotú jel)** angol kifejezés alapján **BIT** - nek nevezik.

📖 **Az adatokat BIT-ek sorozatával kódolhatjuk. Ennél nagyobb egység a Byte, amely nyolc biten kódolt jelsorozat. (1 Byte = 8 BIT)**

Matematikában megfelelője a kettes számrendszer, melyben csak két alaki érték (számjegy) létezik, az **1** és **0**. Ennek az elnevezése az angol nyelv alapján **BINÁRIS** számrendszer. **Tehát a számítógép BIT-ek illetve BYTE-ok sorozatát tudja csak közvetlenül értelmezni!**

Kettes számrendszerben a kódábécé-nek két eleme van, a 0 és az 1.

A helyiértékek kettőnek a hatványai. A bináris számok esetén a számrendszer alapszáma a 2.

A helyiértékek 1 Byte =8 Bit esetén:

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
1	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**

✍ Számoljuk ki az $10001_{(2)}$ és $11101_{(2)}$ (1 Byte-on jelölve: **00010001** illetve **00011101**) bináris számok decimális értékeit:

$10001_{(2)} = 1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 16 + 1 = \mathbf{17_{(10)}}$

$11101_{(2)} = 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 16 + 8 + 4 + 1 = \mathbf{29_{(10)}}$

☝ A bináris számok kimondására nem alakultak ki a magyar nyelv szabályai. Például nem mondhatjuk $101_{(2)}$ esetén, hogy százegy. Bináris számok esetén egyszerűen felsoroljuk a számjegyeket.

2.3. Tizenhatos, hexadecimális számrendszer

Alaki értékek (számjegyek) nullától tizenötig. Ezért a tíz, tizenegy, tizenkettő, tizenhárom, tizennégy, tizenöt értékekre **új jeleket kellett bevezetni:**

A 9-nél nagyobb az értékeket az ABC betűivel jelöljük (**A, B, C, D, E, F**). Így a hexadecimális számrendszer jelkészlete (számjegyeinek a jelölése) :

0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F				

Helyiérték: Mindig tizenhatszorosára változik (tizenhat hatványai)

1	16	256	4 096	65 536	1 048 578	.....
$\mathbf{16^0}$	$\mathbf{16^1}$	$\mathbf{16^2}$	$\mathbf{16^3}$	$\mathbf{16^4}$	$\mathbf{16^5}$	.....

Példa az átváltásra :

$\mathbf{A2F3_{(16)}} = \mathbf{A} * 4096 + \mathbf{2} * 256 + \mathbf{F} * 16 + \mathbf{3} * 1 =$

$= \mathbf{10} * 4096 + \mathbf{2} * 256 + \mathbf{15} * 16 + \mathbf{3} * 1 = \mathbf{41715_{(10)}}$

Tehát a $\mathbf{A2F3_{(16)}}$ hexadecimális szám tízes számrendszerben: $\mathbf{41715_{(10)}}$.

🕮 **A számrendszer alapszáma megmutatja: Hány darab számjegyünk (jelünk) van a mennyiségek kifejezéséhez. A helyiértékek hányszorosára változnak, ezért a helyiértékek az alapszám hatványai!**

2.4. Konverzió bináris és hexadecimális számokká

- Átváltás kettes számrendszerbe

✍ Hogyan lehet felírni a 43 decimális számot bináris számrendszerben?

↘ 43-at osszuk el 2-vel (az alapszámmal)

43 : **2** = **21**

1

↘ Osszuk el a kapott hányadost (*21*)2-vel

21 : 2 = **10**

1

↘ Osszuk el az új hányadost (*10*) ismét 2-vel

10 : 2 = **5**

0

↘ Osszuk el hányadost (*5*) újból 2-vel

5 : 2 = **2**

1

↘ Folytassuk tovább a műveletet!

2 : 2 = 1

0

↘ Osszuk el a hányadost (*1*) ismét 2-vel

1 : 2 = **0**

1

☝ **Az utolsó hányados 0, ezért vége az eljárásnak**!

A bináris szám számjegyei a maradékos **osztások maradékai**! Az első maradék a legkisebb helyiérték. Ezért a maradékokat a végrehajtás sorrendjével ellentétesen jegyezzük le.

Tehát $43_{(10)}$ egyenlő az $101011_{(2)}$ a bináris számmal. Egy Byte-on leírva 0010 1011

ellenőrzés:

$= 1*32 + 0*16 + 1*8 + 0*4 + 1*2 + 1*1 = 32 + 8 + 2 + 1 = 43_{(10)}$

- **Átváltás tizenhatos számrendszerbe**

A módszer hasonló mint bináris számrendszer esetén. A különbség csak az alapszám, ezért az ismételt osztásokat 16 al végezzük.

✍ Alakítsuk át a 4858 decimális számot hexadecimálissá!

↘ 4858-at osszuk el 16-al (az alapszámmal)

4858 : 16 = 303
10

↘ Osszuk el a kapott hányadost ismét 16-al

303 : 16 = 18
15

↘ Osszuk el a hányadost (18) újból 16-al

18 : 16 = 1
2

↘ Az osztást **addig ismételjük, amíg a hányados nulla** nem lesz

1 : 16 = 0
1

A maradékokat fordított sorrendbe (1, 2, 15, 10) lejegyezve:

$4858_{(10)} = 12FA_{(16)}$ ahol **A** a 10-et, **B** a 15-+öt jelenti

ellenőrzés: $12FA_{(16)} = \mathbf{1} * 16^3 + \mathbf{2} *16^{2*} + \mathbf{15}*16^1 + \mathbf{10}*16^0 =$

$=4096 + 512 + 240 + 10 = 4858_{(10)}$

Az átváltás szabályát(algoritmusát) következőképpen is meg lehet adni:

Osztandó **legyen egyenlő** a vizsgált számmal.

Csináldd!

Oszd el az osztandót a számrendszer alapszámával.

Jegyezd le a maradékot.

Osztandó **legyen egyenlő** az osztás hányadosával.

Addig, amíg a hányados egyenlő nullával.

Különböző feladatok algoritmusának a fentiek szerinti megfogalmazását a számítástechnikában **mondatszerű nyelvnek**'nevezzük! Vegyük észre hogy az osztások sorozatát többször kell **ismételni.** Az ismétlések száma függ az átváltandó tízes számrendszerbeli számtól. Az osztandó, minden egyes lépésben változik. Az osztó mindig a számrendszer alapszáma.

Azokat a feladat részleteket, melyeket többször meg kell ismételni, **ciklusoknak nevezzük**. A ciklus **befejezése a hányados = 0 feltétel igaz, vagy hamis voltától függ**. A hányados = 0 feltételt a ciklus **kilépési feltételének nevezzük**. Az **ismétlendő műveleteket**, pedig a **ciklus magjának**.

Ezeknek a megoldási terveknek a segítségével, bármilyen programnyelven megoldható a feladat. Természetesen ehhez ismerni kell az adott nyelv utasításait és a nyelv sajátosságait.

- **Átváltás tizenhatosból kettes számrendszerbe**

✍ Alakítsuk át az **C7B5** hexadecimális számot binárissá!

Az eddigiek alapján az egyik módszer lehetne az, hogy előbb átváltjuk decimálisba, majd a kapott eredményt bináris számrendszerbe. Ez két különböző algoritmus.

Egytől tizenötig a decimális számok könnyen átalakíthatók bináris számmá (0001,0010,0011.....1111). Ezért a másik módszer:

↘ A hexadecimális számot, jobbról balra haladva számjegyekre bonjuk.

↘ Minden számjegynek meghatározzuk négy biten a bináris megfelelőjét:

$C_{(16)}$=**$12_{(10)}$=$1100_{(2)}$** $7_{(16)}$=$7_{(10)}$=**$0111_{(2)}$**

$B_{(16)}$=**$11_{(10)}$=$1011_{(2)}$** $5_{(16)}$=**$5_{(10)}$=$0101_{(2)}$**

↘ Összefűzzük a kapott bináris számokat:

$C7B5_{(16)}$=1100 0111 1011 $0101_{(2)}$

A Turbo Pascal nyelv

A számítógép az **információ tárolására**, rögzítésére szolgáló gép. De ha csak ez volna a feladata, akkor megtenné egy magnetofon, vagy video is. A különbség az előbbiekhez képest az, hogy a számítógép **emberi beavatkozás nélkül** fel tudja dolgozni a kapott adatokat. Az adatok feldolgozása az elektronika fejlődésével egyre gyorsabb lett. A gép a közhiedelemmel ellentétben nem tud „gondolkodni". Ezért a feldolgozás végrehajtására csak akkor képes, ha ehhez **munkatervet kap**. Ennek a munkatervnek egy adott **kódrendszerben** való leírását programnak nevezhetjük.

1. A program fogalma

Pontosítsuk egy kicsit a program fogalmát! **A program tulajdonképpen „cselekvések", utasítások véges számú sorozata**. A programnak tartalmaznia kell a következőket:

- A feldolgozandó adatok, valamint azok tulajdonságai
- Vezérlést leíró utasítások, vagyis
  - A feladat megoldásához szükséges adatok beolvasása (**INPUT**). Ez történhet billentyűzetről, mágneses háttértárolókon tárolt állományokból.
  - Az eredményt előállító műveletek és utasítások végrehajtásának sorrendje (**FELDOLGOZÁS**)
  - Az eredményt jelölő adatok értékének kiíratása monitorra, nyomtatóval, vagy a háttértároló egy meghatározott állományába (**OUTPUT**)

Hogyan „érti" meg a számítógép az ember (a felhasználó) által készített programok utasításait? Hogyan végzi el a számítógép az adatok közötti műveleteket? A számítógép „lelke" a **processzor**, amely bonyolult áramkörök segítségével értelmezi az adatokat, utasításokat, képes néhány egyszerűbb művelet elvégzésére is. A számítógép számára fizikai felépítése alapján legkönnyebben értelmezhető állapot az, melynek **két lehetséges értéke** van. Ezért a processzor segítségével a gép **bitek sorozatát** tudja csak közvetlenül értelmezni!

- A **processzor részei:**
  - ↘ **SZÁMOLÓEGYSÉG (ALU).**

Ennek feladata egyszerűbb számtani és logikai műveletek elvégzése. A műveleteket bonyolult áramkörök segítségével végzi el. Az adatokat a számolóegységgel, két állapotú jelek segítségével közölhetjük. (0 vagy 1)

  - ↘ **VEZÉRLŐEGYSÉG (CU)**

Vezérli a központi memóriában tárolt program alapján a számolóegység működését. Irányítja a **processzor** és a **perifériák** (be- és kimeneti egységek) közötti információcserét, adatátvitelt.

  - ↘ **REGISZTEREK**

A processzor tartalmaz még **saját tárolóhelyeket** is adatok átmeneti tárolására. Ezeket nevezzük regisztereknek. Ennek segítségével gyorsabb az adatok mozgatása, a műveletek és vezérlések végrehajtása. Ha az összes információt a központi memóriában kellene megkeresni, az lassítaná a számítógép működését. A processzor az adatokat, utasításokat gyorsabban tudja ide-oda mozgatni a központi memória és saját **tárolóegységei** között.

2. A programozási nyelvek csoportosítása

2.1. Alacsony szintű programnyelvek

- **Gépi nyelv**

Hogyan tud például a gép összeadni két számot? (6 + 9).

Ennek elvégzése természetesen függ a processzor vagyis a számítógép típusától. Ebből következik, hogy **konkrét gépi nyelvű** utasítások és azok végrehajtásai csak egy adott típusú számítógépen végezhetőek el! A gépi nyelv **hardver függő!** A példa leírását nagyon leegyszerűsített formában tárgyaljuk. Ennek segítségével akarjuk érzékeltetni milyen bonyolult ennek a egyszerű problémának a végrehajtása gépi nyelven!

- ↘ **Töltsd** a processzor **regiszterébe** a **6**-ot!
- ↘ **Tedd** a központi memória (Operatív tár) adott sorszámú **(pl. 18-as) rekeszébe a regiszter tartalmát** (a 6-ot)
- ↘ **Töltsd** a processzor **regiszterébe** a **9**-et!

↘ **Add össze** a regiszter **jelenlegi** tartalmát (9) a 18-as számú rekesz (6) tartalmával!

↘ Állj meg!

Az előző példában szereplő **regisztert akkumulátornak** nevezzük. Ennek a regiszternek a segítségévelvégzi az **ALU** a műveleteket. Észrevehettük, hogy a processzor regiszterei jelen úgy működnek mint egy „átjáróház" Ki-be száguldoznak rajta keresztül az adatok a **központi memória és a processzor** között. **A regisztrerek közvetítésével** az adatok mozgatása, műveletek elvégzése sokkal gyorsabb. Tartalma lehet egy adat, egy adat helye (**címe**) a központi memóriában. A processzor fejlettsége a regiszterek számától és hosszúságától függ! **Minden regiszternek különböző feladata van.** A regiszterek nagyságát BIT-ekben mérjük (8 ; 16 ; 32....BIT). A számítógép **teljesítőképessége** tehát függ a **processzor** minőségétől!

A gépi kódú (gépi nyelvű) programban az utasításokat és az adatokat bitek segítségével írjuk le (kódoljuk). A számítógép így a programokat átalakítás nélkül, azonnal végre tudja hajtani. Ezért ezt futtatható programnak (Angol nyelven: **exe**cutable program) nevezik, amely az operációs rendszer felügyelete alatt közvetlenül indítható. Az utasítások értelmezése természetesen függ a processzor, vagyis a számítógép típusától. Ebből következik, hogy konkrét gépi nyelvű utasítások és azok végrehajtásai csak egy adott típusú számítógépen alkalmazhatók! **A gépi nyelv hardverfüggő!**

Nagy lépés volt a számítástechnikában az 1940-es évek eleje. Ekkor fogalmazta meg a magyar származású Neumann János a tárolt program elvét. Ennek lényege, hogy a program kódjai annak adataival együtt azonos módon, a központi memóriában legyen tárolva.

A processzor kb. 100 utasítást ismer, melyeket a 0 és 1 számjegyek segítségével kódolhatunk. Ezen kívül a programozáshoz ismernünk kell a memória pontos felépítését. Ennek megjegyzése, észben tartása emberfeletti feladat lenne.

Gépi nyelven programozni csak azok a szakemberek tudtak, akik jól ismerték a számítógép felépítését, vagyis a hardvert. Egy egyszerűbb szorzás művelet elvégzése is igen összetett programozási feladat.

- **Assembly nyelv**

A gépi nyelv nehézségei miatt a programozási nyelvek fejlődésének következő lépcsőfoka az Assembly nyelv volt. **Ennek lényege, hogy a műveleti utasításokat rövidítésekkel helyettesítjük.** A rövidítések az utasításokat kifejező angol szavak kezdőbetűi. Ezek jobban emlékeztetnek az utasításra és kifejezik azok jelentését. A memóriacímeket, adatokat a rövidség kedvéért tizenhatos számrendszerben adjuk meg.

Néhány példa a rövidítések jelentéseire:

LDA (Load Akkumulátor)	ADD (Additions)
Töltsd az akkumulátorba.	**Add az akkumulátor tartalmához.**
STA (Store Akkumulátor)	MOV (Move)
Helyezd át az akkumulátor tartalmát.	**Helyezd egy memóriacímre**

Az akkumulátor a processzor számolóegységéhez tartozó tárolóhely, mely az adatok átmeneti tárolására szolgál.

Természetesen **ezt a nyelvet a számítógép közvetlenül nem értette** meg. Ezért **szükség volt egy másik programra, amely lefordította az assembly programot**. Az első fordító programot gépi nyelven írták meg. **Az assembly nyelv fordítóját assemblernek nevezzük**. Az assembly nyelven írt program a forrásprogram, a lefordított gépi nyelvű formája pedig a tárgyprogram.

2.2. Magasszintű programnyelvek

A programozás fejlesztésekor több szempontot is figyelembe kellett venni. Az egyik cél az volt, hogy a programozás ne csak néhány ember kiváltsága legyen. Természetesen a fejlesztést úgy kellett végrehajtani, hogy a programozás hatékonyságát is növelni kellett. Így alakultak ki az emberi nyelvhez sokkal közelebb álló programnyelvek. Ezeket **magasszintű programnyelveknek** is nevezik. Természetesen **ezen programnyelvek megértéséhez is a gépnek szüksége van valamilyen fordítóprogramra.**

- **Compiler**

  A **program megírásához** nagy segítséget ad minden nyelv esetében egy menüvezérelt **szövegszerkesztő** (EDITOR). A program az ember számára közvetlenül értelmezhető karakterekből áll. A szövegszerkesztővel elkészített programot **forrásprogramnak** (SOURCE CODE) nevezzük. A lefordított programot **tárgyprogramnak**, tárgykódnak hívjuk (OBJECT CODE). A tárgyprogram még nincs futtatható állapotban.

A tárgyprogramot futtatható állapotba a programszerkesztő alakítja át (LINKER).

A fordítás és a programszerkesztő segítségével a folyamat eredménye egy futtatható gépi kódú (exe kiterjesztésű) program. A kész lefordított programot mágneses háttértárolón tárolhatjuk. Így későbbi végrehajtásához (futtatásához) nincs szükség arra a magasszintű nyelvre, amellyel a forrásprogramot megírtuk. **Az így működő magasszintű nyelvek fordítóját az angol megfelelő alapján COMPILER-nek nevezzük.** Compiler fordítóra példa a TURBO PASCAL nyelv.

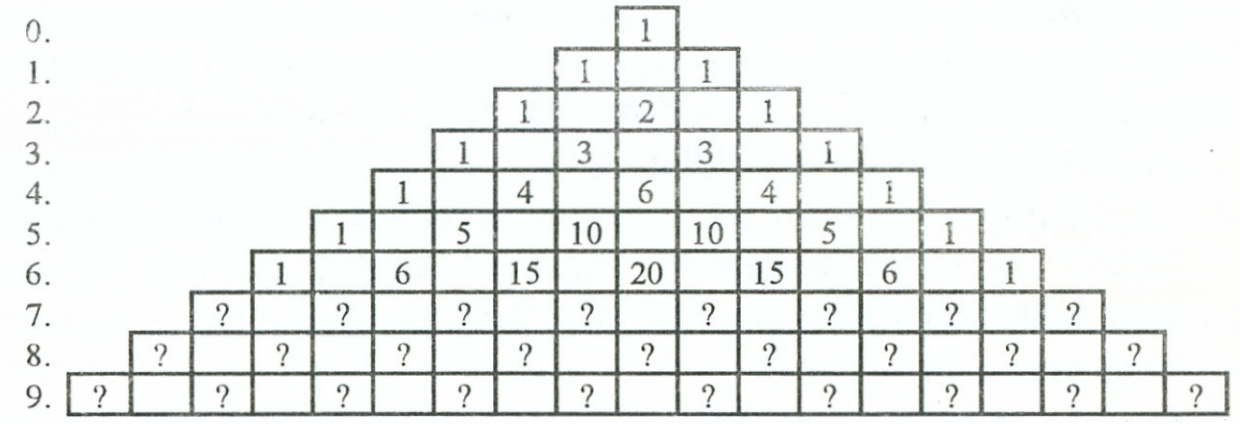
- **Interpreter** (Értelmező)

  Lényeges különbség, hogy az értelmező nem állít elő összeszerkesztett, végleges, azonnal futtatható programot. **Az értelmező a forrásprogramot utasításonként hajtja végre**. Ennek több hátránya is van. Futtatása, végrehajtása mindig feltételezi a magas szintű nyelv jelenlétét. Nagyobb a memóriaigénye, hiszen az értelmező program a forrásprogram végrehajtásáig mindvégig a memóriában tartózkodik. Az előbbiek miatt működése lassú. Erre példa néhány BASIC nyelvjárás.

3. A Pascal nyelv története

- A Pascal magas szintű programozási nyelv terveinek megalkotója egy Svájcban élő egyetemi tanár. Nevezetesen, Nicklaus Wirth a Zürichi Műszaki egyetem professzora 1968-ban készítette el a nyelv alapjait. A nyelvet Blaise Pascal-ról nevezte el, aki a XVII. század egyik legkiválóbb francia tudósa. A névadóról néhány mondat erejéig illik szólni! **Blaise Pascal** 1623. június 19-én született Clermont-Ferrandban. Apja kiváló matematikus volt, aki egyedül nevelte őt két leánytestvérével együtt. Pascal nem járt egyetemre, sőt iskolába sem. Tehetségének kibontakozása édesapjának és magántanárainak köszönhető. Pascal sok tudományban jeleskedett. Foglalkozott a **matematika, fizika** és **filozófia tudományok** szerteágazó területeivel. A matematikán belül nevéhez fűződik a **kúpszeletekkel kapcsolatos tétel**. Megalkotta a matematika akkor legfiatalabb ágának, a **valószínűség számítás**nak az alapjait. Ez teremtette meg **a matematikai statisztika, információelmélet alapjait**, amely ma is a számítógépek egyik legfőbb alkalmazási területe. Az ő nevét fémjelzi az úgynevezett **PASCAL Háromszög,** amely egy összeg, tetszőleges nem negatív egészkitevőjű hatványainak kiszámításá-

hoz ad segítséget. Ez az algoritmus programozási nyelvekkel is könnyen megoldható.



Egy kicsit elrejtve ugyan, de ennek is vannak számítástechnikai vonatkozásai! Összefüggést találhatsz a Byte fogalmaával kapcsolatban is.

Történeti kutatások is bizonyítják, hogy Pascal készítette el az első működőképes számológépet. Ennek segítségével a legegyszerűbb, négy alapművelet volt elvégezhető. Még sok példával folytathatnánk e zseniális tudósnak a munkásságát. Fejezzük be egy nagyon találó filozófiai idézetével: Az ember *„semmi a végtelenhez képest és minden a semmihez képest, valahol a semmi és a minden közt, a középen"*. Ezek után érthető a névadó kiválasztása.

A pascal nyelv fejlődése: Wirth professzor megelégelte a hagyományos programnyelvek hiányosságait. Tervezői szándéka az volt, hogy fegyelmezett munkára ösztökélje a programozókat. **1970-ben készült el az első fordítóprogram a nyelvre.** Nem sokkal ezután megjelent az Standard Pascal nyelv, mely azóta is különböző fejlesztéseknek az alapja. Az első változatok nagygépekre készültek. A nyelv alkalmazkodó képessége akkor mutatkozott meg, amikor megjelentek az első személyi számítógépek. Népszerűsége így annak is köszönhető, hogy szinte minden típusú processzoron használható volt. Különböző típusokra, így pl. COMMODORE gépekre is készültek értelmezők. További fejlődést az IBM személyi számítógépek megjelenése tette lehetővé. A TURBO PASCAL elődjét egy dán programozó írta, melyet a Borland Intézet fejlesztett tovább az Egyesült Államokban. **A nyelv bővítése napjainkban is folytatódik.**

3.1. A nyelv néhány kiemelkedő tulajdonsága:

- **Rendelkezik saját szövegszerkesztővel**, mellyel elkészíthetjük forrásprogramunkat.

- **Tartalmaz fordító és futtató rendszert**, mellyel könnyen létrehozható a forrásprogram operációs rendszerből is azonnal értelmezhető, indítható változata.
- **Segítséget ad a program javítására és tesztelésére, beépített hibakereső és nyomkövető** segítségével.
- Rendszerkörnyezetével, saját egységek készítésével **lehetőséget ad a nyelv fejlesztésére.**

3.2. A Turbo Pascal nyelv fontosabb állományai

- `turbo.exe`

  A szoftver egyik legfontosabb programja. Ez az állomány egy **keretrendszer. Tartalmaz egy több ablakos szövegszerkesztőt**, melynek segítségével párhuzamosan akár több program is szerkeszthető. Ebbe az állományba van beépítve a **fordító** (compiler), amely a forrásprogramból tárgykódú programot készít. Ezzel még nincs kész a teljes fordítás! Futtatható állapotba a programszerkesztő (LINKER) alakítja át. Így lesz saját általunk elkészített programunkból a rendszer jelenléte nélkül végrehajtható gépi nyelvű program. A programszerkesztőt nem szabad összetéveszteni a szövegszerkesztővel. Segít még a programozónak a program kijavításához egy hibakereső, nyomkövető is.

  Tehát a Turbo Pascal programnyelv a `turbo.exe` azonosítójú állománnyal indítható.

  Hol található ez a file? Ez függ a felhasználótól vagyis Tőled, aki például telepítette a rendszert. Elfelejtettük a könyvtár nevet? Ez nem okoz problémát. Megkeresheted pl. a Norton Commander segítségével, vagy akár egy megfelelő DOS paranccsal is. Ha az elérési út automatikusan az `autoexec.bat` állományban meg van adva akkor nincs probléma.

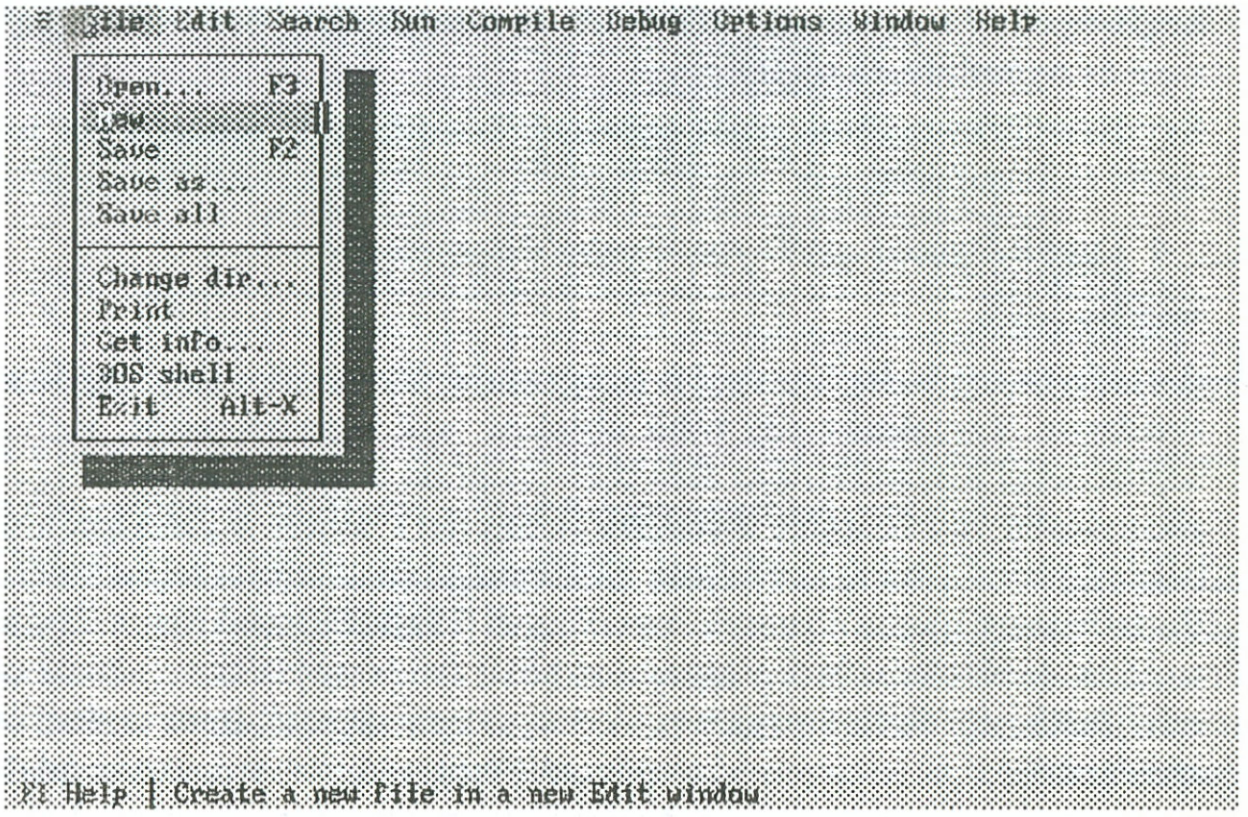
  Tegyük fel hogy az elérési út: **c:\tp60**

  Így a rendszer indítása: **c:\tp60\turbo.exe**

✍ Indítás előtt készíts a Pascal könyvtáradban egy tanulprg könyvtárt, ahová kimentheted programjaidat!

✍ Keresd meg te is a gépeden az előbb említett állományt! Ha megtaláltad indítsd is el a rendszert! Indítás után **<F10>** billentyűvel lépj be a menürendszerbe és a kurzor mozgató billentyűkkel válaszd ki **File** főmenüt!

Másik lehetőség a belépésre az **<Alt>+<F>**. Általános szabály bármelyik menü aktualizálására **<Alt>+<a menüpont kezdőbetűje>**. Ha ez sikerült, akkor a következő kép jelenik meg:



A kurzorunk éppen a **File/New** parancson található, mely funkció jelentése az ablak legalsó sorában angolul megtalálható. A státuszsorban lévő „Create a new file in a new Edit window" angol kifejezés jelentése: Új állomány (program) létrehozása a szövegszerkesztő ablakban.

<Enter> billentyűvel válaszolva akár indulhat is a programírás. A menürendszerek néhány fontosabb feladatára még visszatérünk!

- `turbo.tph`

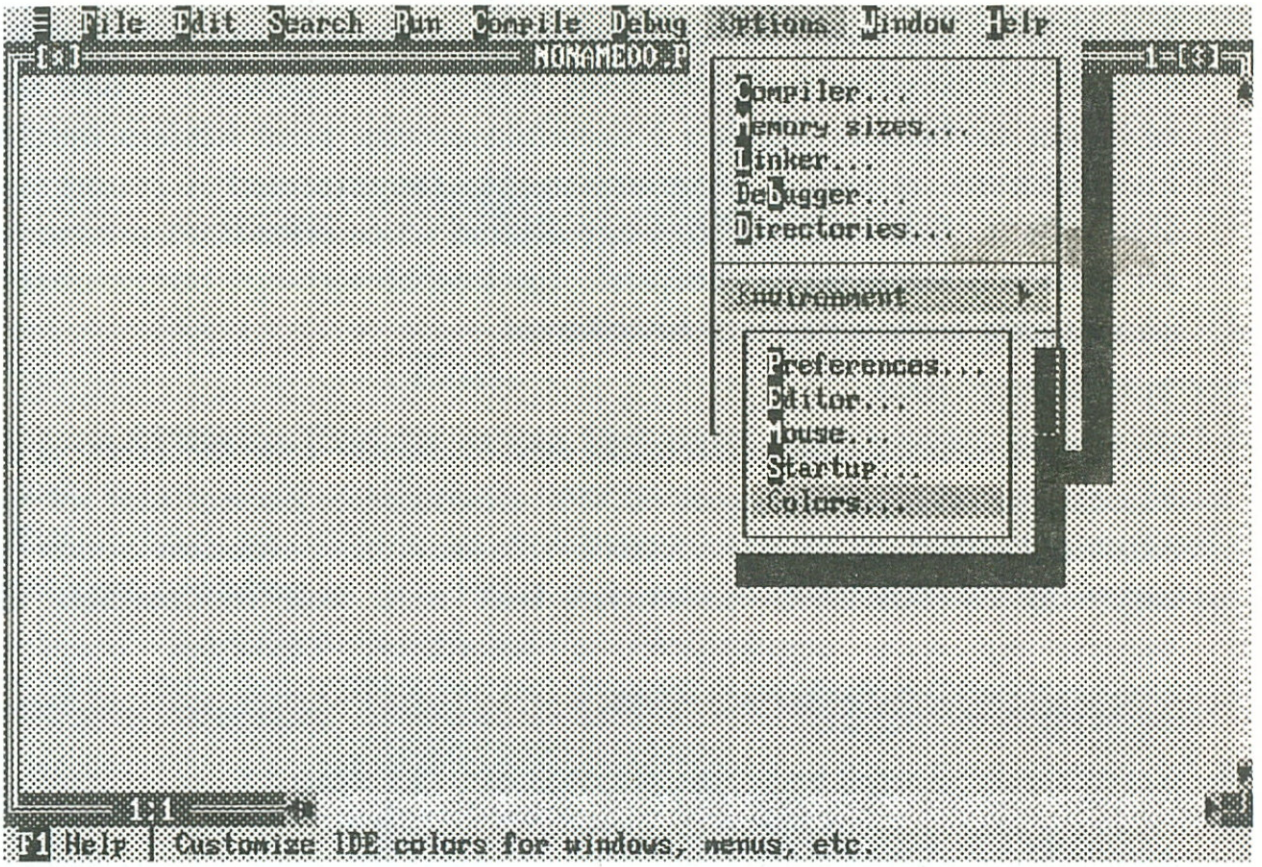
A program írása közben ez az állomány bármikor **segítséget nyújt**, ha van valamilyen problémánk. Ha a **programban hiba van**, akkor az **<F1>** billentyű segítségével tájékoztatót kapunk annak lehetséges okairól.

Ha elfelejtettük egy **utasítás pontos használatát**, akkor a **<Ctrl>+<F1>** lenyomásával kapunk erre vonatkozó ismertetést. Ez a billentyűkombináció a kurzor pozíciójához tartozó utasítás használatáról **tájékoztat bennünket.**

- `turbo.tp`

A Pascal lehetőséget ad különböző beállítások elmentésére, rögzítésére. Ezt minden szoftvernek illik tudnia.

A szerkesztő által kínált változtatási módokat, környezeti beállításokat az **Options** menüpont segítségével hajtjuk végre. Indítás után az <**Alt**>+<**O**> billentyűkombinációval a következő képet kapjuk.



A lehetőségek felsorolása hosszú és fárasztó lenne. Ha kedved és alkalmad van hozzá, akkor nyugodtan kísérletezhetsz. Nézzünk ebből néhány fontosabb példát!

- Directories

  Ezzel állíthatjuk be a **rendszer fontosabb állományainak az elérési útvonalát**. Például itt adhatjuk meg, hogy a lefordított program a mágneslemez melyik könyvtárába kerüljön.

- **Environment** (környezeti beállítások)

  Segítségével lehetőség van a **szövegszerkesztő színeinek** és az **egér kezelésének** beállítására, programok automatikus biztonsági mentésére, stb.

↘ Save Options

Ezzel a menüponttal **mentjük ki a rendszer változtatásait** a `turbo.tp` állományba, melyet a programnyelv indításakor ugyanebből a file-ból olvas ki.

- `turbo.tpl` (rendszerkönyvtár)

Ha ez a file nem lenne, akkor a rendszer csak egy félkarú óriás lenne. **E nélkül** az állomány nélkül **egyetlen programot sem írhatnánk**, hiszen ez tartalmazza a Pascal nyelv utasításait, eljárásait. **Több önálló egységet tartalmaz**. Ezekre használata esetén, mint később meglátjuk hivatkoznunk is kell a programban. Ha megszereted a programozást akkor Te is eljutsz odáig, hogy egy általad írt és lefordított programegységet ebbe a könyvtárba „beépíts". Így később egy-egy bonyolultabb eljárásra **elég csak a nevével hivatkozni**.

- `tpumover.exe`

Ha a programozásban eljutottunk egy bizonyos szintre, akkor a nyelv lehetőséget ad saját egységek (**UNIT**) készítésére. Ebben már jól kipróbált, működőképes és összetettebb feladatok megoldására képes programrészletek lehetnek. Ha ezeket már nem kell módosítani, akkor `tpumover.exe` segítségével a lefordított egységek beépíthetők `turbo.tpl` rendszerkönyvtárba!

4. Kezdeti lépések a programozásban

A programnyelv megszeretéséhez, elsajátításához nagyon sok gyakorlás és kitartás kell, természetesen számítógép mellett. A könyvben található tájékoztató – a nyelv lehetőségeit távolról sem meríti ki –, csak egy kis ízelítőt ad. Remélem, hogy a leírtak alapján kedvet kapsz arra, hogy ennél sokkal több ismeretet szerezz a Pascal nyelvről.

4.1. Egy egyszerű program felépítése

✍ Indítsd el a rendszert az előző fejezetben ismertetett módon!

Ha a szerkesztő nem megnyitott ablakkal jelentkezik be, akkor a **File/New** menüponttal (**<Alt>+<F>**) hozz létre egy szerkesztőablakot! Már a kezdet kezdetén gondolom felvetődik a kérdés, **hogy milyen utasítással kezdődik és végződik a program**? Ha angol nyelvet tanulsz, akkor sokkal könnyebben meg fogod tanulni a programnyelvet mint mások, **a nyelv**

szókészlete ugyanis angol szavak, kifejezések, vagy azok rövidítései. Ha magyarul értene a fordítónk, akkor

Kezdődik

.....

Vége.

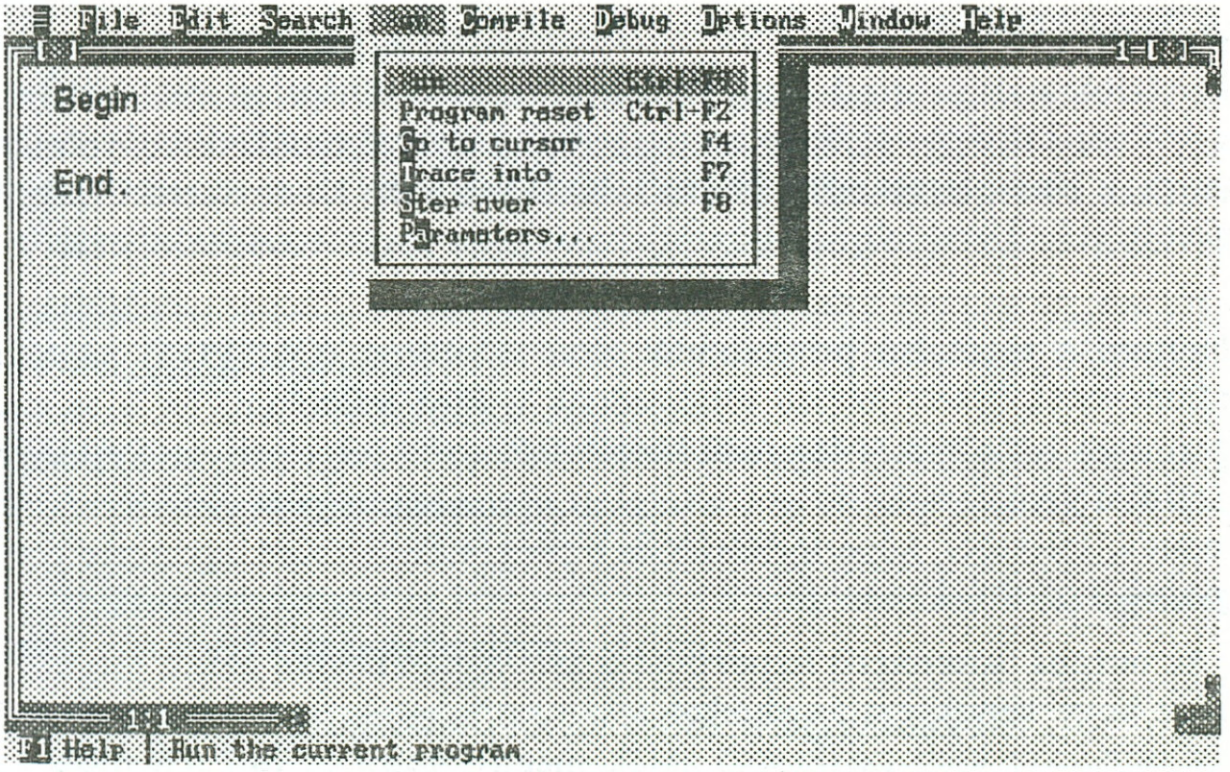
szavak közé zárva írhatnánk meg programunkat. Ennek megfelelője angolul:

Begin

.....

End.

Nagyon fontos az End szó végén lévő pont. Mint később meglátjuk az End szót másra is használhatjuk, persze más írásjelet téve utána. Tehát mondhatjuk hogy az End jelentését környezete dönti el! Erre egyszerű példa a magyar nyelvben is van. Például a „nap" szó jelentését egy mondatban, a körülötte szereplő szavak határozzák meg.



Programunk már működőképes, csak az égvilágon semmit nem csinál.

✍ Futtasd le a programot a **File/Run** menüponttal! Az indítás helyettesíthető a **<Ctrl>+<F9>** billentyűkombinációval.

☝ Vigyázz a CONTROL billentyűre, mert ha nem tartod nyomva akkor csak fordítás történik. (Ne ijedj meg ha ez nem a leírtak szerint sikerült!)

Mit vettél észre a program végrehajtásakor? Gondolom semmit. Ez már jó jel, mivel a program végrehajtotta a „semmit". Ha láthatóan történik valami akkor a fordító üzenne, hogy valami hiba van a programunkban. Ezt kipróbálhatod, ha a például az **End** szó után lévő pontot letörlöd!

✍ Töröld le pontot és indítsd el ismét a programot!

A fordító intelligensen kiírta, hogy „Error 10:Unexpected end of file" vagyis a „hiba kódja 10: hiányzik a program vége"! Hasonló hibaüzenetet kapsz, ha a **Begin** szó hiányzik, vagy nem helyesen írtad le.

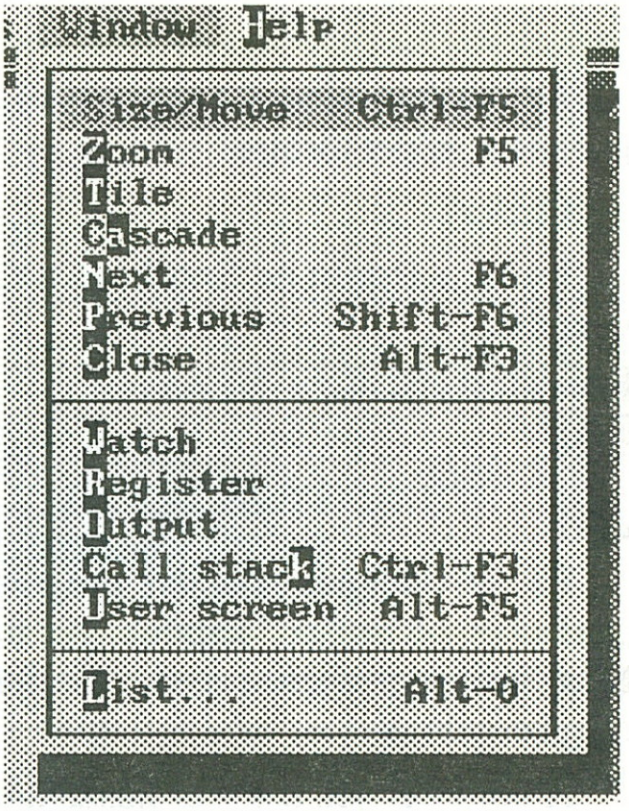
Minden nyelvnek jól meghatározott szabálya, nyelvtana van. A fordító minden futás közben megvizsgálja, hogy programunk megfelel-e a nyelv logikájának. Ezt **szintaktikai ellenőrzésnek** nevezzük. Sok esetben a fordító sem tudhatja a hiba pontos okát. Ekkor a rendszer HELP-je segítséget ad a hiba lehetséges okainak felsorolásával.

Gondolom alig várod, hogy csináljunk már olyan programot amelynek van valami látszata is. Ilyen szempontból a Pascal tanulása bonyolultabb mint pl. a Basic nyelv volt. Ebben a nyelvben még az egyszerűbb programokat is át kell gondolni, mert **nagyon szigorú a nyelv szintaktikája**.

Ha sok programot futtatunk egymás után, akkor a képernyőn nehéz kiválasztani a legutolsó programunk által előállított (OUTPUT) adatokat, eredményeket. Ezért jobban eligazodunk a kiírt adatokon, ha a monitor képernyőjét letöröljük. Még nem beszéltünk arról, hogy **milyen képernyőre küldi a program az output adatokat**. Remélem te sem gondoltad, hogy az adatok kiíratása, képernyőtörlés és a programírás mind egyszerre a szövegszerkesztő ablakban történik.

Az output (kimeneti) képernyőt a szövegszerkesztő ablak eltakarja. Ahhoz hogy lássuk ezt az ablakot sok lehetőségünk van. Már említettük, hogy egyszerre több szerkesztő ablak megnyitására is van lehetőség. Az ablakok közötti váltás az **<F6>** billentyüvel történik.

Jelen esetben van egy OUTPUT ablakunk is, melyet a program szerkesztő ablakához igazíthatunk. Ennek beállítása érdekében aktivizáljuk a **Window** menüpontot az <**Alt**>+<**W**> billentyűkkel! Az itt lévő ablak jelenik meg. Kérd ezután az **Output**, majd a **Tile** menüpontot. Ennek hatására a Pascal szövegszerkesztőjének mérete csökken és alatta megjelenik egy másik ablak, melyre a program a kiíratandó adatokat fogja megjeleníteni. A teljes Output ablakot, a **User Screen** (felhasználói képernyő) <**Alt**>+<**F5**> segítségével nézheted meg.



✍ Folytassuk programunkat nevének megadásával és a képernyőtörléssel!

4.2. A program nevének megadása

A programok elején a programunknak valamilyen nevet szokás adni. Ez a **Program** szóval történik. Ez után kell beírni a **program nevét**, melyet Te választhatsz meg. A név után pontos vesszőt (;) kell írni. Fontos még, hogy a név csak az **angol ABC betűit, számjegyeket és aláhúzás karaktert tartalmazhat**, de nem kezdődhet számjegyet jelölő karakterrel!

például:

Program `tisztit;`

☝ A program nevének megadása **nem kötelező**! A névadás arra is jó, hogy később emlékezzünk arra, hogy milyen feladat elvégzésére írtuk azt.
A programban megadott név mint látni fogod, **nem** feltétlen **ugyanaz, mint amellyel a mágneslemezen tároljuk.**

✍ Egészítsd ki a programod elejét a név megadásával!

4.3. Képernyőtörlés

Az utasítás az angol kifejezésnek megfelelően (**Clear Screen**) melynek jelentése „képet törölni"

ClrScr

Írd be a Begin és End közé a ClrScr utasítást, és tegyél utána pontos vesszőt!

```
Program tisztit;
Begin
     ClrScr;
End.
```

Ha az eddigiek alapján programunkat futtatjuk, akkor **hibaüzenetet** *„unknown identifier"* kapunk. Az üzenet jelentése *„ismeretlen azonosító"*. Most látszik a nyelv szigorú szerkezete. Az azonosító fogalmáról később még beszélünk.

A könyv elején sokszor használjuk az „**utasítás**" szót, melyhez egyelőre tartjuk magunkat. E helyett a helyes szóhasználat sok esetben az „**eljárás**" lenne, amely pontosan meghatározható, összetettebb feladat elvégzésére képes. Ez gépi szinten sok-sok elemi szintű műveletek algoritmusa, melyet a fordító önálló egészként tud kezelni. Erre példa a **ClrScr**, hiszen ez is egy eljárás.

4.4. Beépített egységek használata

A fordítót fel kell előre készíteni sok utasítás értelmezésére. A feladat kezeléséért a `turbo.tpl` állomány a felelős. Fordításkor ez a file azonnal betöltődik a memóriába. Az előbb említett állományban alapesetben öt különálló egység található. Ezek mindegyike egy-egy jól meghatározott problémakör kezelésére hivatott. Minden egység (unit) több, összetett probléma elvégzésére képes egy megfelelő utasítás segítségével.

A legfontosabb az úgynevezett rendszer (SYSTEM) egység. Ez az egyetlen, aminek a nevére nem kell hivatkozni. Külön egység végzi pl. a nyomtató (PRINTER) irányítását, míg egy másik kötelessége a képernyő és a billentyűzet (CRT) kezelése.

Te is csodálkoznál, ha azt a parancsot kapnád, hogy „töröld le". Mit töröljek le? Az ablakot vagy a mosolyt az arcodról? A következőképpen már egyértelműbb lenne.

Használd az ablakot;

Kezd

Törlés;

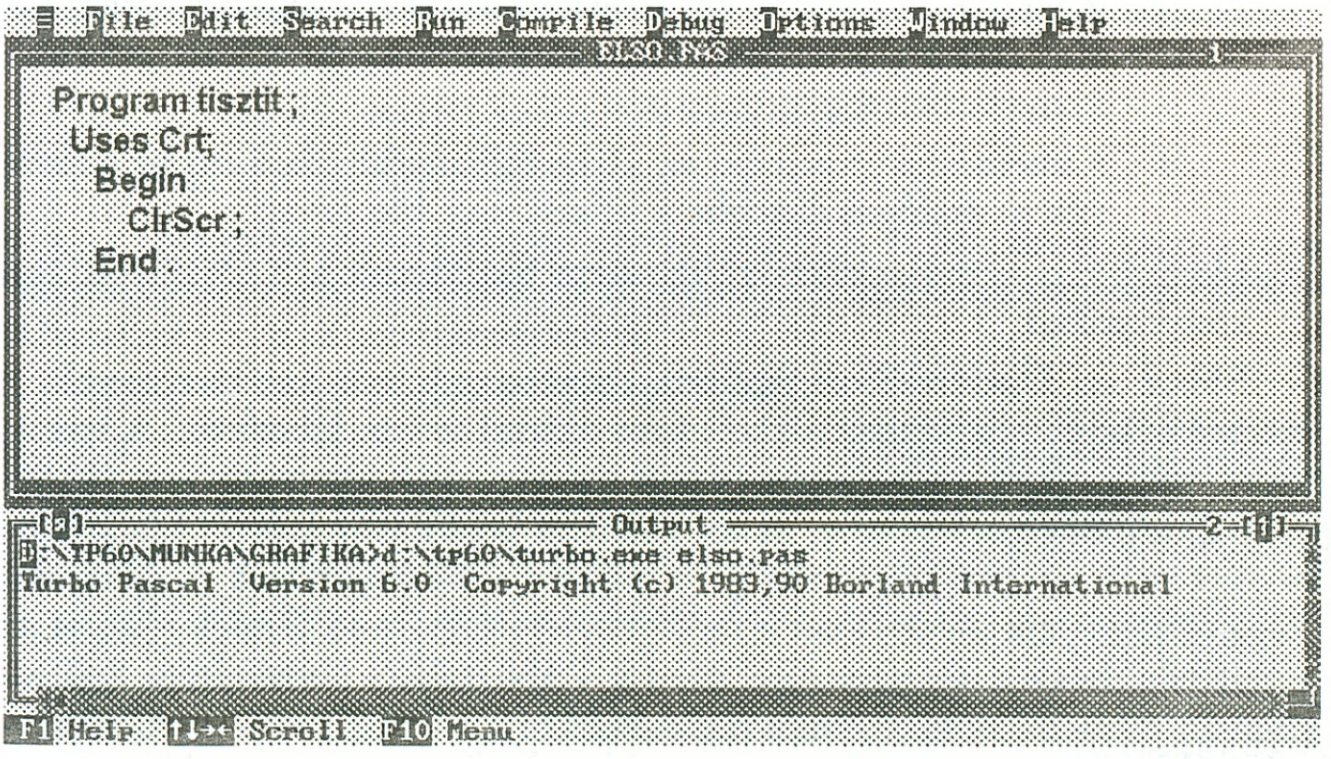
Vége.

Remélem kitaláltad, hogy miről van szó! Komolyabbra fordítva a szót, a **ClrScr** utasítás az előbb említett rendszerkönyvtár CRT nevű egységébe van beépítve. A fordító viszont csak akkor ismeri fel a **ClrScr** szó igazi tartalmát (képernyőtörlés), ha megmondjuk a problémakör gazdáját, CRT.

A használjuk szó angol megfelelője **Uses**. Egészítsd ki tehát az eddigieket a program neve utáni sorban a következővel:

Uses Crt;

Ha az eddigi feladatokat jól végezted el, akkor a szerkesztőablakban a következő kép látható. A Te képernyőd output ablaka természetesen más, hiszen a rendszer indítása a Pascal könyvtár neve és elérési útja gépenként változó.



Kezdő, de még haladó programozóval is gyakran előfordul, hogy elfelejtkezik egy fontos jelről. Ez a karakter a pontos vessző (;)! Erről a szintaktikai hibáról a Pascal mindig értesít a „*Expected (;)*" hibaüzenettel. Ho-

va is kell tennünk? Ennek pontos megfogalmazása elég nehéz. Első megközelítésben egyezzünk meg abban, **hogy minden utasítás után pontos vesszőt kell tenni**! Egy utasítást nyelvtanilag nem mindig egyetlen szó fejez ki. Így a pontosvessző szerepe hasonló, mint a magyar nyelvben a mondatvégi jel.

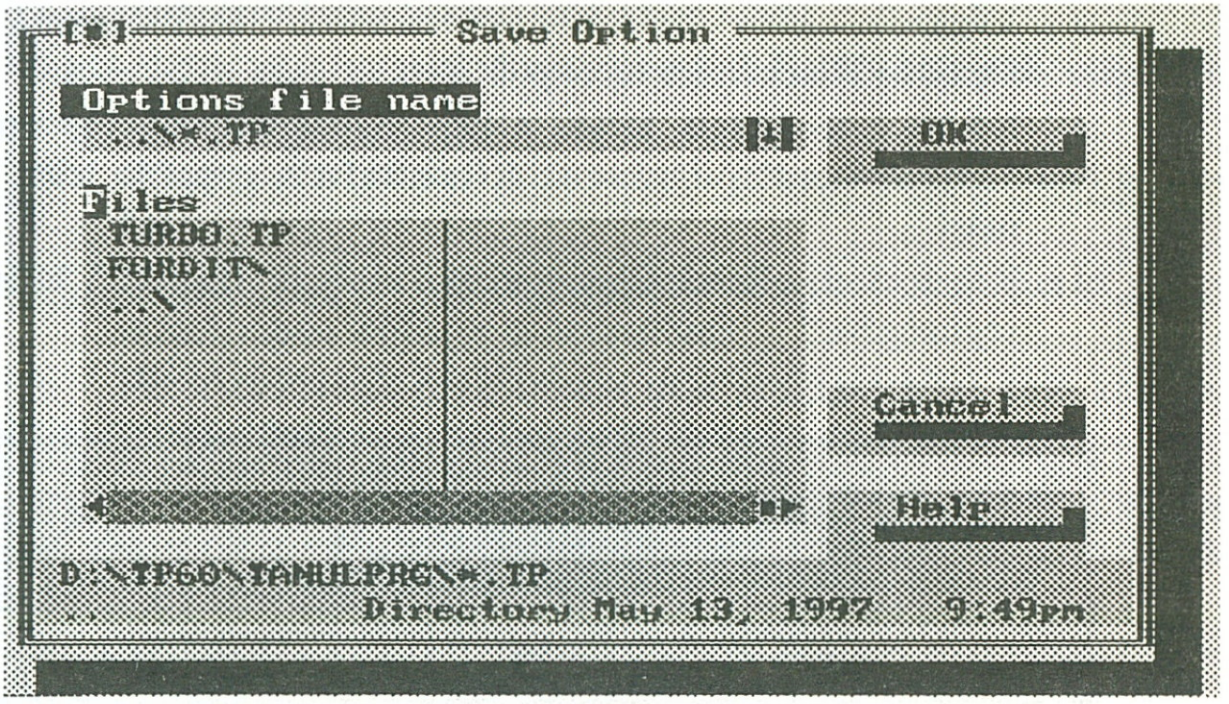
> 🕮 **A szerkesztő egy sorában akár írhatunk több utasítást is, vagy egy utasítás kerülhet akár két sorba is. A fordító (;) jel segítségével fog több utasítást szétválasztani.**

✍ Indítsd el programot! Az eredmény magáért beszél.

Mivel rend a lelke mindennek, fejezzük be eddigi munkánkat a következőkkel:

- Tároljuk a háttértárolón programunkat a Te általad már régebben elkészített TANULPRG alkönyvtárába, `elso.pas` néven!
  - ↘ Először tegyük aktívvá a TANULPRG könyvtárat!
  - ↘ Válaszd ki a **File/Change Dir** menüt <**Alt**>+<**F**> és a <**C**> billentyűkkel! Keressd meg **Directory Name** párbeszédablakban az említett könyvtárnevet. (Ha ezt végrehajtottad, akkor minden lemezre írás művelet automatikusan ebben a könyvtárban fog történni.)
  - ↘ Kimentéshez használd az <**F2**> billentyűt! Majd írd be a **Save file as** dialógus dobozba a `elso.pas` file nevet!
- Mentsd ki a szövegszerkesztő jelenlegi beállításait a TANULPRG alkönyvtárába!

Ehhez az **Options/Save options** menüpontot kell kiválasztanunk, a <**Alt**>+<**O**> és <**S**> billentyűkkel! Erre egérrel is van lehetőség. A Pascal most létre fogja hozni a `turbo.tp` és `turbo.dsk` állományokat. Ha később újra indítod a rendszer, akkor jelenlegi beállítások szerint töltődik be a szerkesztő. Így megmaradnak pl. a mostani ablakbeállítások és automatikusan megjelenik ablakodban a most megírt programod. Ennek viszont az a feltétele, hogy a `turbo.exe` file-t a TANULPRG könyvtárból kell indítanod, vagy az előbb említett állományokat a `turbo.exe-t` tartalmazó katalógusba kell áthelyezni. Ha eddig a leírtak szerint végezted el feladatodat, akkor a következő képet kellett kapnod:



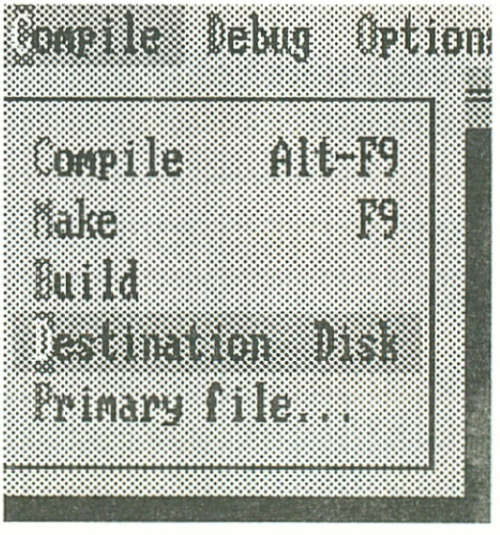
- Lépj ki a rendszerből és hozz létre egy LEFORDIT könyvtárat a TANULPRG alkönyvtárban!
  - ↘ Lépj ki a szoftverből az <**Alt**>+<**X**> billentyűkombinációkkal!
  - ↘ Hozd létre a Norton Commander segítségével a TANULPRG\ LEFORDIT könyvtárat!

  Indítsd el a Pascalt az előbb létrehozott könyvtárból!

Ha mindent az eddigiek szerint hajtottál végre, akkor indítás után két ablakod van (a szerkesztő és az output). A szerkesztőbe pedig betöltődött az `elso.pas` nevű programod.

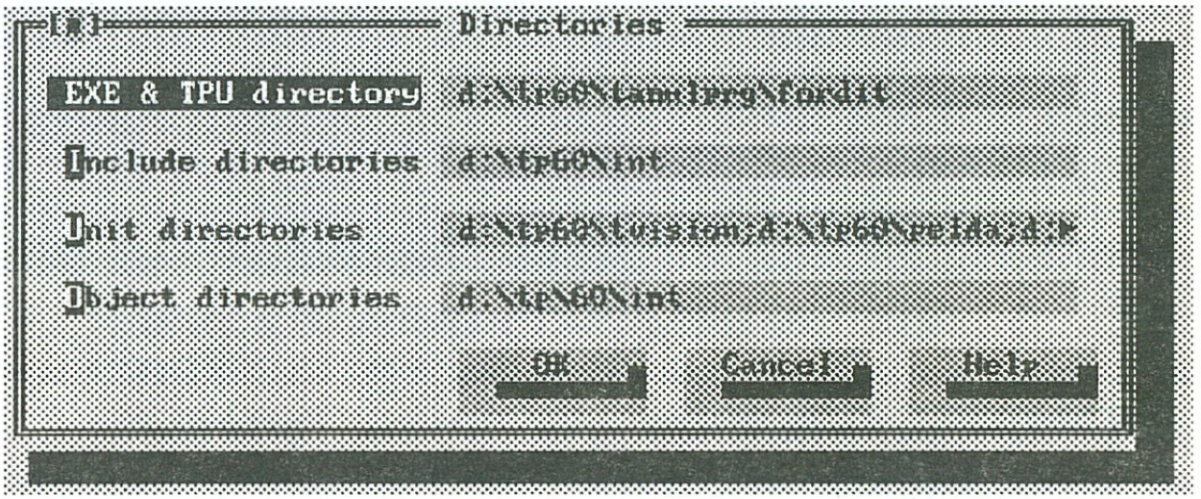
- Állítsd be, hogy a rendszer lemezre fordítson!

  Kérd a képen látható **Compile** menüt az <**Alt**>+<**C**> billentyűkkel! Majd nyomd le a <**D**> karaktert! A képen láthatod, hogy két lehetőséged van. Fordíthatsz lemezre (Disk) és memóriába (Memory). A fordítás majd az <**F9**> billentyűvel történik (Make).



- Befejezésül fordítsd le a forrásprogramot (`elso.exe`), hogy az a LEFORDIT könyvtárba kerüljön!

- Végezd el billentyűkombinációk segítségével az **Options/Directories** menüpont kérését!
- Írd be a megfelelő helyre a kép segítségével a már említett könyvtár nevet! Vigyázz az útvonalra, a te gépeden lehet hogy más!
- Készítsd el a forrásállomány fordítását a megfelelő billentyű segítségével!



- Keresd meg és indítsd el az operációs rendszerből a fordítás eredményét, az `elso.exe` nevű állományt!

4.5. A Pascal program szerkezete, belső struktúrája

Az eddigiek alapján beszélhetünk a Pascal program általános szerkezetéről. Ha megnézzük egyszerű kis programunkat akkor ez három különálló részből áll.

```
Program elso;          programfej
Uses Crt;
.....
.....                  deklarációs rész
......
Begin
......
      ClrScr;          programtörzs
......
End.
```

- A programfej tartalmazza tehát a program nevét.
- A **deklarációs rész** nagyon fontos lesz számunkra. Később még erre hivatkozni fogunk.
  - Ebben a részben kell jellemezni a programban használt adatokat
  - Itt kell megírni olyan programrészleteket, melyeket a programtörzsben többször is használunk
  - Hivatkozhatunk programunk más programokkal való kapcsolatára, stb

A **programtörzsben** kell megadni a **Begin** és **End.** között a végrehajtandó utasításokat, ebben a részben történik az egész program kerek teljes egységgé szervezése.

Input - Output műveletek

Ebben a fejezetben az **állandó adatok** kiíratásának lehetőségeivel foglalkozunk. Meglátod, maga a kiíratás egyszerű. Ennél egy kicsit nehezebb lesz az **állandó** (konstans) és **változó adatok fajtái és** azok **jelölése**. Egy jól működő program nem véletlenszerűen, hanem rendszeretően írja ki az eredményeket. Ezért ebben a fejezetben szó lesz az output adatok **szerkesztett kiíratásának a lehetőségeiről is.** Megkérdezhetnéd, hogy miért nem az adatok beolvasásával **(Input)** kezdjük? A kérdés jogos, hiszen e nélkül nem igen beszélhetünk egy feladat automatikus feldolgozásáról. A sorrendet az egyszerűség indokolja.

1. Állandó adatok

A karakterek fogalma még sokszor elő fog fordulni. Első gondolata az embernek, hogy a karakterek vizuálisan, írásos formában megjeleníthetőek. Később meglátjuk, hogy ez csak részben igaz. De mi az a string?

1.1. String és karakter konstansok

📖 **A string a szöveg szó angol megfelelője, amely karakterek végesszámú sorozata. Maximum 255 karaktert tartalmazhatnak.**

Ahhoz, hogy fordító felismerje, hogy konkrét szövegről vagy karakterekről van szó, valamilyen jelet kell tenni a szöveg elejére és végére. Magyar nyelven az idézeteket idézőjelek („) közé szoktátok írni. A Pascalban is hasonló a helyzet, csak a jel más. Ennek jelölésére a felső vessző karaktert használjuk, melyet **aposztrófnak** is szokás nevezni.

- **Karakterek** jelölése:

  'p' '(' '8' 'A'

- **Stringek** jelölése:

  'Pascal' 'Programozni tanulok'

A karakterek „megjeleníthetők" kódszámok segítségével is. Az 'A' betű kódszáma 65, 'B' betűé 66, stb. Ha egyesével folytatjuk, akkor könnyen meghatározható így az angol ABC összes nagybetűje. A karakterek

„kiírathatók'' az <**Alt**>+<**kódszám**> billentyűkkel pl.: egy szövegszerkesztőben. A kódszámot a numerikus billenyűzeten kell begépelni. A kódszámok értéke 0 és 255 közötti egész számok lehetnek. A számrendszereknél láttuk, hogy ez megegyezik 8 biten (1 Byte-n) leírható bináris számmal. A Pascal nyelvben karakterekre a # jel és ASCII kódszámuk segítségével is hivatkozhatunk. Szöveg konstanst megadhatunk az őt tartalmazó karakterek kódjainak felsorolásával is.

Foglaljuk táblázatba néhány karakter, szöveg (string) jelölésének lehetőségeit:

kódszám nélkül	**kódszámmal**
'A'	#65
'B'	#66
'BABA'	#66#65#66#65
' ' szóköz (space)	#32
'9'	#57
'19'	#49#57

Egyes kódszámok vizuálisan **nem „megjeleníthetők**'', ilyenek a **vezérlő karakterek**. Erre néhány érdekes példa a soremelés (Line Feed) **#10,** melynek kódja 10. Ennek hatására a kurzor ugyanabban az oszlopban marad, de **egy sorral lejjebb kerül**. A másik a **kocsi vissza** (Carrige Return) **#13**, amely vezérlőkarakter a kurzort az adott **sor eléjére helyezi**. Egy másik a számítógép belső hangszóróját szólaltatja meg, melynek kódolása **#7.**

1.2. Adatok kiíratása: a **Write** *és* **WriteLn** *eljárás*

Szó szerinti jelentése tükrözi feladatát. A **WriteLn** viszont két szót takar. A több szóból álló kifejezések rövidítéseit a Pascal nyelvben egybeírjuk. Ezért, hogy jobban érzékelhető legyen az összetétel minden szó kezdő betűjét nagy betűvel írjuk.

Write írni
Write Line sort írni

Eddig egyszerű is, de hiányzik az alany, vagyis mit írunk ki? A másik probléma hogyan jelöljük azt? A mit kérdésre egyszerű a válasz, tehát **a kiírandó adatokat kerek zárójelek közé tesszük! Ha egynél többet aka-**

runk kiíratni, akkor vesszővel válasszuk el őket. Szövegkonstans esetén a következő jelek kötelezőek: (' ').

*Programunk belső neve legyen: szavakat_ir1. Írjuk ki a képernyőre a következő szöveget:'**Programozni tanulok Turbo Pascal nyelven**'*

Egészítsd ki programod a következő utasítással és indítsd is el!
***Write**('Programozni tanulok Turbo Pascal nyelven');*

A főprogram mindig a képernyő azon helyétől kezdi a kiíratást, ahol a kurzorunk áll. **A ClrScr utasítás a kurzort mindig a képernyő bal felső sarkába helyezi.**

```
Program szavakat_ir1;
Uses Crt;
Begin
     ClrScr;
     Write('Programozni tanulok Turbo Pascal nyelven');
End.
```

Az utolsó előtti sorban egészítsd ki a feladatot a **ReadLn**; utasítással és indítsd el újra!

Ennek hatására a program csak az <**Enter**> billentyűre fejezi be a végrehajtást. Amíg nem nyomtuk meg ezt a billentyűt, addig láthatjuk a program output adatait a képernyőn. Erről az utasításról később még lesz szó.

1.3. A Write és WriteLn eljárások közötti különbség

Írassuk ki a képernyőre többféleképpen a szem és üveg szavakat az említett utasítások váltogatásával! Bővítsük programunkat a

Write('szem'); és

Write('üveg'); utasításokkal .

```
Program szavakat_ir2;
Uses Crt;
Begin
   ClrScr;
   Write('szem');
   Write('üveg');
   ReadLn;
```

```
End.
```

A kiíratás eredménye:

```
szemüveg
```

✍ Javítsd át a második kiíró utasítást **WriteLn**-re! Indítsd el a programot javítás után is! Próbáld ki az összes lehetséges esetet a **Write** és **WriteLn** utasítások cserélgetésével! Mit vettél észre?

Ha kipróbáltad az összes lehetséges variációt, akkor a következőt tapasztalhattad:

utasítás	a képernyő
Write('szem'); **Write**('üveg');	szemüveg
Write('szem'); **WriteLn**('üveg')	szemüveg
WriteLn('szem'); **Write**('üveg');	szem üveg
WriteLn('szem'); **WriteLn**('üveg');	szem üveg

✍ Fogalmazd meg a két utasítás közötti különbséget!

A különbség még érzékelhetőbb lesz, ha még egy szóval megtoldjuk a kiíratást. Legyen ez a szó: tok

💻 *Írassuk ki az említett három szót a következő formában!*

szemüveg

tok

A program:

```
Program szavakat_ir3;
Uses Crt;
Begin
      ClrScr;
      Write('szem');
      WriteLn('üveg');
      Write('tok');
      ReadLn;
End.
```

📖 **A Write hatására a következő kiíratás a legutoljára kiírt adat után közvetlenül folytatódik. A WriteLn a kiíratás befejezésekor a kurzor a következő sor elejére kerül, így a kiíratás legközelebb innen folytatódik.**

Példák a vezérlőkarakterek használatára a **Write** utasítással

utasítás	képernyő
Write('számítás',#10,'technika')	számítás technika
Write('szem',#10,'üveg',#10,'tok')	szem üveg tok
Write('gép',#10,'író',#10,#13,',','nő')	gép író nő

Az utolsó példa programja pl.:

```
Program vezerlo;
Uses Crt;
Begin
ClrScr;
Write('gép',#10,'író',#10, #13,'nő');
ReadLn;
End.
```

1.4. Számkonstansok típusai

- **Egész típusú.**

  Ezeknek az egész számoknak a jelölése, megegyezik a matematikában szokásos írásmóddal. Kódolásuk két Byte-on történik. A különbség az értékhatár, így például a **legkisebb és a legnagyobb** kiértékelhető **konstans egész** szám:

  -2147483648= – 2^{15} illetve **+2147483647=2^{15}–1**

☝ Az értékhatárok különbsége 2^{16}– 1, **mivel** két Byte-on (16 biten) az 1111 1111 1111 1111 bináris szám a 2^{16} – 1 decimális számnak felel meg.

- **egész típusú konstansok**

↘ **decimális** (tízes számrendszerbeli jelölés)

65535 +242 0 –128 255

↘ **hexadecimális** (tizenhatos számrendszerbeli jelölés)

Ezeket a típusokat a Pascalban a dollár($) jel jelöli

$B $1A, $13 $BF $FF -$0C

- **Valós típusú konstansok**

Ennek a véges tizedes törtek felelnek meg. A tizedes helyiértékek kezdetét nem vesszővel, hanem ponttal jelöljük.

Például:

12.0 –0.62 +145.7 15.346E2

A 15.346E2 jelentése: $15.346*10^2 = 15.346*100 = 534.6$, így már ismerős matematika óráról. (a * jel a szorzás)

✍ Írjunk programot, mely a képernyőre soronként kiírja a következő adatokat: **'+12'**, +12, **'+12.0'**, 12.0 , **'145.7'**, 145.7, **'–$2A'** –$2A 'BABA'.

A program írja ki a feladat címét is: pl.: **'Konstans adatok típusainak bemutatása:'**

Figyeld meg a képernyőn az output adatok közötti különbséget!

Mi lehet az eltérés oka?

```
Program adat_tipusok;
Uses Crt;
Begin
   ClrScr;
   Write('Konstans adatok típusainak bemutatása:');
   WriteLn('+12');
   WriteLn(+12);
   WriteLn('12.0');
   WriteLn(12.0);
   WriteLn('145.7');
   Write(145.7);
   WriteLn (-$2A);
   WriteLn ('-$2A');
   WriteLn (#66,#65,#66, #65);
```

```
    ReadLn;
End.
```

A lehetőségek széleskörűek:

sor	A képernyőn megjelenő adatok	típusa
1.	Konstans adatok típusainak bemutatása:	string
2.	+12	string
3.	12	egész
4.	12.0	string
5.	1.2000000000E+01	valós
6.	145.7	string
7.	1.4570000000E+02	valós
8.	–$2A	String
9.	–44	a hexa szám átváltva
10.	BABA	String kódokkal

- A 2. , 6. és 8. sorokban a '+12' , '145.7' , –$2A string adatok voltak, ezért minden karakterük kiíródott.
- A 3. sorban a +12 számtípusú pozitív egész szám, de a gép nem írta ki a pozitív előjelet, mint ahogy matematikában is szokás. Ha a 2. sorban '12' írattunk volna ki, akkor a képernyőn megjelenő különböző adattípusok között nem láttunk volna semmi különbséget!
- A 3. és 5. sorban kiírt értékek numerikus adatok, de ezen belül az egyik egész, a másik valós.
- A 7. sorban lévő 145.7 számtípusú (numerikus) pozitív valós típusú adat. A gép lehagyta a pozitív előjelet, helyette egy szóközt írt és normál alakban írta ki. Az 1.4570000000E+02 jelölés normál alaknak felel meg. Ezt a fogalmat ti is tanultátok már az iskolában. Az E+02 jelölés 10^2-al való szorzást jelent!
  Ezért $1.4570000000*10^2=145.70000000= =145.7$
- A 9.sorban egy negatív hexadecimális konstanst írattunk ki, melyet a Pascal fordítója nekünk decimális (tízes) számrendszerbe átalakítva írta ki, mivel **–$2A = –(2*16 + 10) = –44**
- A 10. sorban a 'BABA' stringet írtuk ki a # jel ésa 'B', 'A' karakterek kódjával.

📖 **Összefoglalva tehát: A számítógép, a karakter és szöveg konstanst a numerikus és más típusú adatoktól az aposztróf vagy kódjelek alapján különbözteti meg.** A Pascal különbözőképpen kódolja a karakter, string, egész és valós tipusú adatokat. **Ezt szaknyelven úgy is mondhatjuk, hogy mindegyiknek más a belső ábrázolása.**

1.5. A karakteres képernyő felépítése, felbontása

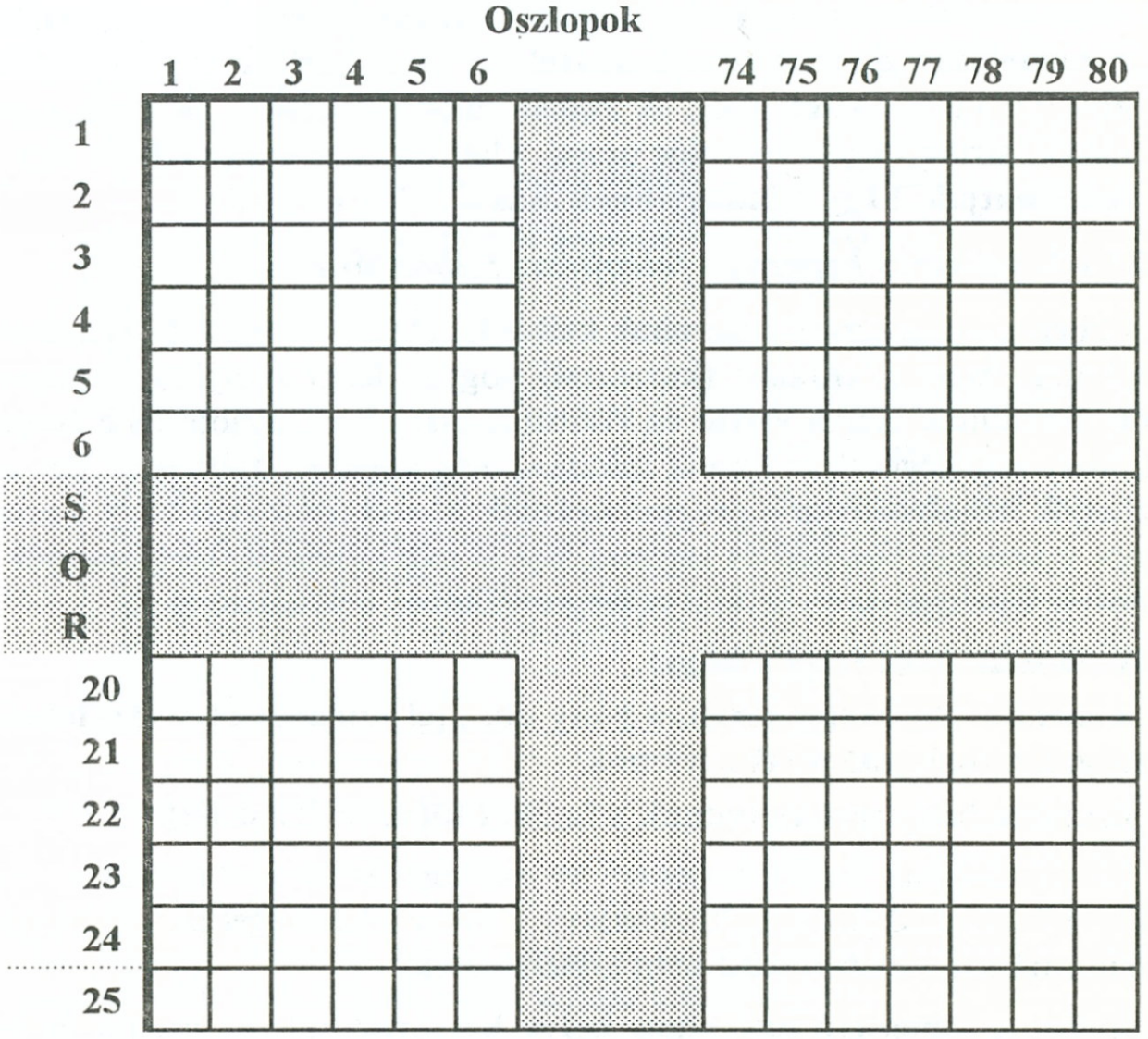
Ebben a témakörben ismernünk kell a karakteres képernyő felépítését. Meglátjuk, hogy a kiíratást (kurzorral) hogyan lehet irányítani a képernyőn. Beszélünk még a **Write** és **WriteLn** utasítások további lehetőségeiről, mely segítségével az adatok kiíratása még áttekinthetőbb lesz. Ezt nevezhetjük felhasználóbarát programozásnak is, ami azt jelenti hogy a programot másoknak készíted és segíteni kell annak kezelését. Megismerjük a változók fogalmát, azon belül a szöveges változók deklarációját.

Felbontás alatt értjük, hogy:

- **a monitor egy adott sorában hány db. „jel kiíratására'' van lehetőség**. Ezt oszlopszámnak nevezzük.
- **összesen hány sorban tudunk adatokat kiíratni** (sorszám).

Alapesetben az output képernyő **80 oszlopot és 25 sort** tartalmaz. Ennek a felbontásnak a megváltoztatására a nyelv lehetőséget ad egy megfelelő utasítással, de ezzel most nem foglalkozunk.

Tehát a képernyő egy adott helye két értékkel, az oszlop és sor számmal jellemezhető.



A fenti ábrán tanulmányozva megértheted a karakteres képernyő **felépítését**. Első látásra hasonlít a matematikai koordinátarendszerhez, sajnos a tengelyek iránya más. A matamatikában megszokottól abban tér el, hogy csak egy síknegyedet tartalmaz. A másik különbség, hogy az oszlopok és sorok számozása a **bal felső sarokban eggyel kezdődik**.

A képernyőn a kurzort irányíthatjuk a kívánt hely **oszlop és sorszámának** megadásával. **A sorrend nem cserélhető fel!**

A kurzor irányítása, a GotoXY utasítás (eljárás):

Az utasítás formája: **GotoXY**(oszlop,sor)

Példák:

GotoXY(1,1).....bal felső sarok
GotoXY(80,1).....jobb felső sarok
GotoXY(1,25).....bal alsó sarok

GotoXY(80,25) .....jobb alsó sarok

GotoXY(1,12).....a monitor középső sora

1.6. Adatok kiíratása mezőszélességben

Az eddigiek során láttuk, hogy a **Write** utasítás az output adatokat a kurzor helyétől írja ki a képernyőre. Lehetőségünk van azonban arra is, hogy megadjuk a programnak, hogy mekkora szélességű helyet foglaljon le az adatok kiíratására. Ekkor a megadott szélességű területre, jobbra igazítva írja ki az adatokat a program.

Néhány példa:

Mezőszélesség

1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.	19.	20.	21.
					s	z	á	m	í	t	á	s	t	e	c	h	n	i	k	a
							g	é	p	í	r	ó	n	ő						
	s	z	ö	v	e	g	s	z	e	r	k	e	s	z	t	é	s			

A mezőszélesség figyelembevétele mindig a kurzor pillanatnyi helyétől függ! A mezőszélesség értékét, a kiíratandó adattól kettősponttal választjuk el! A mező maximális értéke 255 lehet.

Az ennek megfelelően az előző példához tartozó Pascal utasítások:

WriteLn('számítástechnika':21);

WriteLn('gépírónő':15);

WriteLn('szövegszerkesztés':18);

Az elmondottak bemutatására legyen a feladat a következő:

Írjunk programot az előzőek felhasználásával, amely a képernyő első három sorába kiírja az KAR stringet a következő formában:

Az első sor esetében a mezőszélesség 77 oszlop

3	képernyő	80
KAR		KAR
A		A
R		K

Az feladat egyik lehetséges megoldása Pascal nyelven:

```
Program igazit;
Uses Crt;
```

```
Begin
    ClrScr;
    Write('KAR',' KAR':77);
    Write('A','A':79);
    Write('R','K':79);
    ReadLn;
End.
```

Egy másik megoldás a mezőszélességekkel:

```
Program igazit1;
Uses Crt;
Begin
    ClrScr;
    Write('KAR','KARA':78,'AR':80,'K':79);
    ReadLn;
End.
```

✍ Mi a különbség a két program között? Számolj utána a mezőszélességeknek! Oldd meg a feladatot a **GotoXY** eljárással is!

2. Változó adatok típusai

Eddigi programjainkban nem volt (külső) feldolgozandó adat, input. A most következő egyszerű kis program segítségével változó adatok beolvasásához szükséges utasítást és azok deklarációját mutatjuk be.

✍ Írjunk programot, amely billentyűzetről kér tőlünk egy tetszőleges karaktert és kiírja a képernyő 1. és 24. sor bal és jobb szélső oszlopaiba!

Mi az ismeretlen ebben a programban?

- **A számítógép előre nem tudhatja, hogy a perifériákon keresztül melyik jelet (karaktert) kell kiírnia**. A programot erre fel kell készíteni. Ezért **szükséges egy karakter típusú változó**, mellyel a programban a kiíratandó adatot jelöljük. Így az minden indításkor tetszőlegesen megválasztható.

  Legyen ennek **azonosítója:** betu

  A betu **változó segítségével a számítógép a kapott karaktert tárolni tudja a memóriában** a további feldolgozás érdekében.

- Új probléma, hogy **a fordítót a fel kell készíteni a** betu azonosítóval jelölt **karakter típusú változó fogadására**. Ennek kezdetét a Var, a típust pedig a Char utasításokkal jelöljük.

```
Var
betu: Char;
```

- Szükséges még egy utasítás, mely **segítségével** betu **változó értéket kap a** program felhasználójától **és tárolja** azt a memóriában.

```
ReadLn(betu);
```

Ennél az utasításnál várja a kiíratandó karaktert és <**Enter**> után folytatja a program végrehajtását.

☝ A programot még illik kiegészíteni tájékoztató szöveggel is. (String konstans). A használója így tudni fogja, hogy a program milyen adatot vár tőle. A program így sokkal „barátságosabb'' lesz.

Write('Kérek egy karaktert:');

A megoldás:

```
Program beolvas;
Uses Crt;
Var
    betu:Char;
Begin
    ClrScr;
    Write('Kérek egy karaktert: ');
    ReadLn(betu);
    ClrScr;
    Write(betu, betu:79);
    GotoXY(1,24);
    Write(betu, betu:79);
    ReadLn;
End.
```

✍ A program most a képernyő utolsó előtti (24.) sorába ír. Javítsd ki 25-re és úgy is próbáld ki! Mit tapasztaltál? Írd át a programot a **GotoXY** utasítás segítségével úgy, hogy ne használj a **Write** utasításokban mezőszélességet! Futtasd le úgy, hogy egynél több karaktert adsz meg! Kiírt e minden karaktert?

2.1. A változó adat fogalma

A **változó fogalmát** magyarázhatnánk szó szerinti jelentésével is. E helyett próbáljunk felsorolni szempontokat a teljesség igénye nélkül, ami alapján ilyen adatokat használunk egy probléma megoldásához.

- Olyan adatra van szükség, melynek pontos értékét a gép, a program futása közben kapja meg valamely periférián keresztül. Így minden végrehajtás más adattal történhet függetlenül attól, hogy a programon belül értéke esetleg nem fog megváltozni (de megváltozhat).
- A feldolgozandó adattal több összetett műveletet végzünk és a részeredmények tárolásához máskor is szükségünk van.
- Áttekinthetőség: ha egy végeredményhez sok fajta művelet elvégzése szükséges, akkor érdemes több lépésben végrehajtani azt. Ilyenkor a részeredményekhez szintén szükségünk van változókra.

2.2. Változó azonosítója

A változó adat azonosítója (neve)

Honnan tudja a Pascal fordítója, hogy a memória melyik „fiókjába'' helyezze el, és keresse később az adatot? A memóriahelyre azonosítóval (névvel) hivatkozhatunk! A memóriacím már a fordító dolga. Hogyan jelölhetjük az azonosítókat? A változók azonosítóját, a programozó, vagyis Te választhatod meg!

🕮 A változó jelölésének szabályai

- Az azonosító csak **az angol ABC betűi és számjegyek** lehetnek, egyéb speciális karakterek nem használhatók! Kis és nagy betűk között nem teszünk különbséget!
- Minden azonosító egy „szó'', tehát **szóköz karaktert sem tartalmazhat**. E helyett a _ karakter használható.
- **Nem kezdődhetnek számjegy karakterrel**!
- **Nem lehet használni a nyelv foglalt szavait**, melyek közül eddig a következőt ismertük meg:

  Program, Uses, Begin, End, Var

2.3. Változó beolvasása

A változók beolvasását a **ReadLn** utasítás segítségével végezhetjük el. (Read Line azaz olvass egy sort)

Használata: **ReadLn**(azonosító);

Például: **ReadLn**(neved);
ReadLn(jel);

Vedd észre, hogy itt a neved és jel egy változó azonosító, nem pedig egy szöveg konstans (String), hiszen nem használtuk az aposztróf jelet!

🕮 **Működése:**

- A program indítása után **a gép felfüggeszti a program futását, és vár**. A kívánt adatot a kurzor helyétől a billentyűzetről begépelhetjük.
- Az **adat végét az <Enter>** billentyű lenyomása fogja jelenteni.
- A példa alapján a fordító a kapott adatot neved illetve jel azonosítóval fogja jelölni, mely értékére a program **más helyein is ugyanezzel az azonosítóval hivatkozhatunk.**
- A beviteli adat mindaddig javítható, amíg nem volt **<Enter>**. Befejezésekor a kurzor a következő sor elejére kerül.
- Az utasítást használhatjuk azonosító nélkül is, ahogy az eddigi példáinkban láttad: **ReadLn**; (vár egy **<Enter>**-re)

2.4. Változók típusai és deklarálásuk

Az eddigiek során a konstans adatoknak 3 nagyobb csoportját különböztettük meg. Ez a csoportosítás igaz a változó adatokra is.

Karakter	karakterlánc	numerikus	
Char	**String**	egész **Integer**	valós **Real**

A későbbiekben az itt megadott angol kifejezéseket sokszor fogjuk használni, ezért jól jegyezd meg őket! Az osztályozásból még hiányzik az úgynevezett logikai adat. Az egész és valós típusú változók értékhatárok szempontjából további alcsoportokra bonthatóak. Ezekkel a későbbiekben foglalkozunk.

🕮 **A változók jellemzése (deklarációja), tehát a „nevével" (azonosítójával) és típusának megadásával történik a program deklarációs részében. Ennek kezdetét a Var angol rövidítés jelöli. Ezután kell megadni a változó azonosítóját és típusát kettős ponttal elválasztva. Azonos típusok felsorolása esetén veszővel választjuk el őket. Minden egyes típus deklarációját pontosvesszővel fejezzük be.**

✍ Jellemezzünk öt változót a következőképpen:

azonosító	típus
jel	karakter
neved, mondat	szöveg
maradek	egész szám
oldalhossz	tizedes tört is lehet

Példák:

```
Var
jel : Char;
neved, mondat : String;
maradek : Integer;
oldalhossz  : Real;
Begin
.
        . utasítások
.
End.
```

☝ A Var szó után nem kell semmilyen írásjelet tenni!

2.5. Konstans (állandó) adatok deklarációja

- **Példák állandókra:**
  - körrel kapcsolatban ($\pi \approx 3.14$);
  - fajhő;
  - olvadáshő;
  - gravitációs gyorsulás bolygónkon, stb.

Legegyértelműbb a π, hiszen minden körrel kapcsolatos számításnál az értéke változatlan. Fizikából a belső energia változásával kapcsolatban ismeritek pl. a fajhő fogalmát. Te is tudod, hogy ez egy adott anyagra jellemző állandó. Tehát pl. a víz szempontjából állandó, egy másik anyag esetében az értéke más.

Számítástechnikai szempontok:

A probléma jellegétől függ, hogy a program megírásánál szükséges-e használnunk állandó adatokat.

A változókról tudjuk, hogy azokat **azonosítókkal látjuk el és deklarálni kell** őket. Láttunk példát állandó (konstans) adatok jelölésére is, azo-

nosítók nélkül. A Pascal nyelvben **a konstans adatokat is jelölhetjük azonosítóval** (névvel), de akkor **azokat szintén deklarálni kell**. Ha például egy hosszú szöveget kell többször kiíratni a programmal, akkor egyszerűbb a szövegre a nevével hivatkozni.

🕮 **A konstans lényeges tulajdonsága, hogy a programon belül megadott értékét nem lehet megváltoztatni.**

Konstans adat deklarálásánál a **Var** alapszó helyett a **Const** alapszót használhatjuk, amely után szintén nem teszünk pontos vesszőt. A típust a kostans adatoknál ismertettek szerint jelölhetjük, vagyis az azonosító után= jelet teszünk, majd megadjuk a konstans értékét.

Példák konstansok deklarálására

```
Const
   olvasmany = 'A Pál utcai fiúk ';
   gravi = 9.81;
   hegy = 1014;
   soremel = #10;
   mol = 6.0E23;
```

✍ Sorold fel milyen típusú adatokat deklaráltunk?

A constans használatára egy egyszerű példa a következő:

✍ Írjuk ki a képernyő 12. és 13. sorába a következő szöveget:

Ez egy konstans string adat

Ez egy konstans string adat

A program rövidsége és áttekinthetősége érdekében az adott szöveget érdemes jelölni egy azonosítóval (névvel). Legyen az azonosító mondat.

A feladathoz szükséges deklaráció:

Const

mondat = 'Ez egy konstans string adat';

```
Program konstans;
Uses Crt;
Const
     mondat = 'Ez egy konstans string adat';
Begin
     ClrScr;
```

```
    GotoXY(1,12);
    Write(mondat, #10, mondat);
End.
```

Látható, hogy a kétszer szereplő mondat azonosító helyébe nagyobb munka lett volna beírni a hosszú szöveget.

*Írj Programot, amely: Megkérdezi valakinek a nevét, melyet **rögzít egy String változóban**. Az életkorát beolvassa egy **egésztípusú változóba**. A képernyő 12. sorába **kiírja a nevét és utána** 'a nevem, programozni tanulok.' stringet. A 13. sorban kiírja az éveinek a számát és utána 'éves vagyok.' szöveget.*

Elemezzük a feladatot:

- Bemenő adatok: A **név** és az **életkor**. Az Output képernyőn megjelenő adatok pl.:

  Tóth Péter a nevem, programozni tanulok.

  13 éves vagyok.

A program egy lehetséges megoldása:

```
Program mi_a_neved;
Uses Crt;
Var
    kora : Integer;
    neved : String;
Const
    kiir1=' a nevem, programozni tanulok.';
    kiir2=' éves vagyok. ';
Begin
    ClrScr;
    Write('Írd be a neved: ');
    ReadLn(neved);
    Write('Hány éves vagy? ');
    ReadLn(kora);
    GotoXY(1,12);
    WriteLn(neved,kiir1);
    Write(kora,kiir2);
    ReadLn;
End.
```

Értékadó utasítás

1. Műveletek adatokkal

A programozásban **az adatok közötti műveleti jeleket operátoroknak nevezik. A műveletben szereplő adatokat, amelyek között azt el kell végezni, operandusnak nevezzük.**

Alapvető kérdés, hogy **egy művelet milyen adattípusok között végezhető el?** A másik probléma **milyen típus lesz a művelet eredménye?** A felvetett kérdések érthetőek lesznek, ha megismerünk néhány egyszerűbb műveletet.

1.1. Tetszőleges numerikus adatok között végezhető műveletek

műveletek	**műveleti jelek**
összeadás	+
kivonás	-
szorzás	*
valós osztás	/

✍ Írjunk programot, amely két egésztípusú változó között elvégzi az osztás alapműveletet!

A program ''beszédes" változata:

```
Program osztas2;
Uses Crt;
Var
adat1,adat2:Integer;
Begin
     WriteLn('Kérem az első számot');
     ClrScr;
     ReadLn(adat1);
     WriteLn('Kérem az második számot');
     ReadLn(adat1);
     WriteLn('A két szám hányadosa = ',adat1/adat2);
End.
```

Futtasd le többször, különböző adatokkal! Pl: az input adatok legyenek 7 és 2, illetve 8 és 2. Milyen típusú lesz az eredmény? Foglaljuk táblázatba a lehetséges eseteket!

bemenő adat típusa		művelet	eredmény típusa
egész	egész	+	egész
egész	valós	+	valós
valós	valós	+	valós
egész	egész	–	egész
egész	valós	–	valós
valós	valós	–	valós
egész	egész	*	egész
egész	valós	*	valós
valós	valós	*	valós
egész	egész	/	valós
egész	valós	/	valós
valós	valós	/	valós

A táblázatból kiderül, hogy matematikai szempontból a 8/2 eredménye egész szám. A Pascal viszont ezt az eredményt tizedes törtként kezeli, vagyis valós adatként.

Példák:

a 7/2	eredménye	3.5000000000E+00
a 70/2	eredménye	3.5000000000E+01
a 8/2	eredménye	4.0000000000E+00

Az eddigiek alapján megállapíthatjuk, hogy a / jellel jelölt osztás eredménye Pascal nyelvben mindig valós típusú. A gép ezt normál alakban írja ki. Ezért ezt a műveletet valós osztásnak nevezzük.

Erre a „kellemetlenségre" sokszor fel kell készülni programkészítés esetén!

Hogyan lehetne a képernyőn megjelent normál alakot kellemesebb formában kiíratni, megváltoztatni? Erre a sok lehetőség közül az egyik a **Write** utasítás. A **Write** utasítással nemcsak a mezőszélességet adhatjuk meg. **Valós típusú adatok esetén meg lehet határozni a kiíratandó tizedes jegyek számát.**

Néhány példa:

Write(3.82:0:1);	3	.	8			
Write(5.92:3:0);			6			
Write(7.467:0:2);	7	.	4	7		
Write(9.3852:6:3);		9	.	3	8	5

🕮 **Az adat utáni első szám a mezőszélesség, a második pedig a tizedes jegyek száma. A mezőszélességbe a tizedespontot is bele kell számolni.**

✍ Írj programot a táblázatban írt adatok kiíratására és próbáld ki más adatokkal is! Milyen matematikai szabályt veszel észre?

1.2. Műveletek egész típusok között

Egésztípusú adatok közötti művelek közül kettővel fogunk foglalkozni.

Ezeket az operátorokat csak egész típusú adatok között lehet használni és az eredmény is egésztípusú lesz!

- **egész osztás**

  Ilyenkor az eredmény mindig egész szám volt. Másképpen kifejezve a hányados egészrészét számoljuk ki.

  Az egész osztást a DIV operátor jelöli.

  Példák:

  8 div 2=4 ; 17 div 2=8 ; 15 div 4=3 ; 29 div 5=5

- **maradékképzés**

  A maradékképzés operátora: MOD

  Ahogy az elnevezés mutatja, két egésztípus hányadosának a maradékát adja eredményül.

  Példák:

  8 mod 2=0 ; 17 mod 2=1 ; 15 mod 4= 3 ; 29 mod 5=4

1.3. Művelet karakterláncokkal

Az ilyen típusok között az eddig bemutatott műveletek nem végezhetők el! A string típusú adatokkal egy műveletet ismerhetünk meg. Szemlélete-

sen, hatása alapján **összefűzésnek** is nevezhetjük. **A művelet jele:** + (nem tévesztendő össze az összeadással)

Példák konstans karakterláncokkal:

kifejezés	*a kiértékelés eredménye*
'szem'+'üveg'	'szemüveg'
'üveg'+'szem'	'üvegszem'
'üveg'+'pohár'	'üvegpohár'
'pohár'+'üveg'	'pohárüveg'
'malom'+'kő'	'malomkő'
'kő'+'malom'	'kőmalom'
'432'+'11'	'43211'
'11'+'432'	'11432'

A példákon látszik, hogy jelen esetben a plusz jel egészen mást jelent, mint amit eddig megszoktatok. Az eredmény alapján megállapítható, hogy a **művelet nem felcserélhető**. Ennek alapján nézzünk egy egyszerű kis programot, amely bemutatja tulajdonságait.

Írjunk programot, amely a billentyűzetről beolvas két adatot string változókba. Az output adat legyen a képernyő következő soraiban a stringek összefűzése különböző sorrendben.

```
Program osszefuz;
Uses Crt;
Var
   adat1,adat2:String;
Begin
      ClrScr;
      Write('Kérek egy szöveget: ');
      ReadLn(adat1);
      Write(' Kérek egy másik szöveget:');
      ReadLn(adat2);
      WriteLn(adat1+ adat2);
      Write(adat2+ adat1);
      ReadLn
End.
```

Próbáld ki úgy a programot, hogy az input adatok az előző táblázat szerint legyenek!

2. Étékadó utasítás

A kifejezés leegyszerűsítve műveletek (operátorok), azonosítók (konstansok, változók, stb.), zárójelek sorozata a nyelv szabályait figyelembe véve.

2.1. Példák kifejezésekre:

- (8 mod 3)/2
- 2*(a+b)
- (alfa mod 180)/2
- c*m*deltaT
- 4*a
- 180-(alfa+beta) ..... stb.

A kifejezésekben, a műveletek elvégzésének sorrendje hasonló mint a matematikában.

🕮 **Precedencia szabály:** Egy kifejezésen belül megadja a különböző **operátorok végrehajtási sorrendjét. Egy sorszámhoz tartozó műveletek egyenrangúak.** Ebben az esetben a műveletek **végrehajtása balról jobbra történik.**

1.	zárójel			
2.	*	/	DIV	MOD
3.	+	-		

A kifejezések kiértékelése a műveletek sorrendje szerint történik, lépéről lépésre. Minden eredménynek a művelettől függően lesz egy konkrét típusa. Ha ez a típus és a következő művelet típusa összeegyeztethető, akkor a kiértékelés tovább folytatódik. Ezért nagyon fontos, **hogy a részeredmény típusának egyeznie kell a következő típusművelettel!**

Kifejezés	kiértékelése	eredmény
2+5*(3+6)	2+5*9=2+45	47
2+5/2	2+2.5	4.5000000000E+00
2+9/6 Mod 4	2+1.5 mod 4	**nem összeegyeztethető** típusok
2+8/(11 Mod 6)	2+8/5=2+1.6	3.6000000000E+00
10 Div 4/2	2/2	1.0000000000E+00
5/2 Div 10	2.5 div 10	**nem összeegyeztethető** típusok

☝ Ha egy kifejezés nem kiértékelhető, akkor a Pascal nyelv fordítója hibaüzenetet ad. Ennek jelentése: **Az operátor nem értelmezhető az adatok típusai között.**

✍ Számolj utána a fenti táblázatben lévő konstans kifejezéseknek! Határozd meg azt a részeredményt, amely miatt a soron következő művelet nem egyeztethető (nem végezhető el).

2.2. Értékadó utasítás általános alakja

jele: := olvasása: **legyen egyenlő**

utasítás: változó:=kifejezés;

Az értékadó utasítás hatására a Pascal kiértékeli az előbb elmondottak alapján a kifejezés értékét. Ezután ezt az értéket kapja a változó.

Példák:

```
kerulet := 2*(a+b);                belsoenergia := c*m*deltaT
ertek := 2+5/(6 mod 4)             terulet := a*b
kocka := 6*a*a                     gamma :=180-(alfa+beta)
```

☝ Észrevehettétek, hogy eddig is megpróbáltunk (ékezetes betű nélkül) olyan változó azonosítókat használni, amely kifejezi tartalmát!

Próbáld meg az előző értékadó utasítások alapján meghatározni, hogy milyen számításokra lehetne használni!

💻 *Számítsuk ki egy 32 fős osztály buszköltségét egy kirándulásra személyenként! Bemenő adatok: busz kilométerenkénti ára és a kirándulás helyének távolsága.*

- **Állandó adat**: a létszám
- **Változó adatok:** a távolság, buszhasználat ára kilométerenként és az egy főre jutó költség.

Az adatok jelölése: letszam, kilometer, forin, fizet

Ennek alapján a kifejezés: forint*kilometer/letszam

Az értékadó utasítás: fizet:= forint*kilometer/letszam;

```
Program busz;
Uses Crt;
Var
forint,kilometer:Integer;
fizet:Real;
```

```
Const
letszam=32;
Begin
     ClrScr;
     WriteLn('Mennyibe kerül a busz kilométerenként? ');
     ReadLn(forint);
     WriteLn('Milyen messze van a kirándulóhely?');
     ReadLn(kilometer);
     fizet := forint*kilometer/letszam;
     WriteLn('A buszköltség személyenként : ' , fizet );
     ReadLn
End.
```

Írjunk programot, amely meghatározza a víz belső energia változását.

Állandó adat: a fajhő (4.2 kJ / kg °C), **azonosítója** legyen faj

Input adatok: kezdeti és vég állapotának hőmérséklete, a víz tömege.

Változói legyenek kezdet, veg, tomeg

Output adat a belső energia változása, változója energia

$\Delta E_b = c*m*\Delta T$ összefüggés alapján, az értékadó utasítás:

energia:=faj*tomeg*(veg-kezdet);

```
Program energia;
Uses Crt;
Var
   kezdet, veg, energia,tomeg:Real;
Const faj=4.2;
Begin
   ClrScr;
   Write('Mennyi a víz tömege? ');
   ReadLn(tomeg);
   Write('Kérem a kezdeti hőmérsékletet:');
   ReadLn(kezdet);
   Write('Hány fokra melegítettük fel? ');
   ReadLn(veg);
   energia:=faj*tomeg*(veg-kezdet);
   Write('A víz belső energia változása:');
```

```
    Write(energia:0:2,' KJ');
    ReadLn;
End.
```

A program fizikai szempontból csak akkor működik jól, ha a kezdeti hőmérséklet 0 °C-nál nagyobb, a végállapot hőmérséklete 100 °C pedig kisebb. A 0 °C és 100 °C határeset. Ezekben az esetekben már figyelembe kell venni a számításnál a olvadáshőt illetve a forráshőt is. A bemenő adat csak 0-nál nagyobb de 100-nál kisebb valóstípus lehet!

Alakítsd át a programot, hogy csak a fizikai szempontból helyes adatokra végezze el a számítást!

2.3. Egy hasznos függvény: Length( )

Egy String (szöveg) típusú adatban előforduló karakterek számát adja meg. A zárójelek között csak Sring konstans vagy Sring változók szerepelhetnek. Értéke mindig egész típusú lesz!

Példák:

Length('ablak') = 5
Length('1.23') = 4
Length('számítástechnika') = 16
Length('-23400') = 6
Length('+547412') = 7

Értékadó utasításban is szerepelhet a Length függvény. Pl.:

a kar_szam:= Length('számítástechnika') div 6; értékadó utasításban a kar_szam változó értéke 2 lesz. (16 div 6)

Írjunk programot, amely egy string változóba tárolja a nevedet és kiírja a hosszát.

A feladatban az input adatot a neved azonosítójú string változóban tároljuk. A szöveg hosszát a Length függvény segítségével számítjuk ki egy értékadó utasítással. Ennek értékét tároljuk kar_szam nevű változóban.

Az értékadó utasítás: kar_szam:=Length(neved);

```
Program karakterhossz;
Uses Crt;
Var
    kar_szam:Integer;
    neved:String;
```

```
Begin
    ClrScr;
    Write('Kérem a nevedet, megmondom milyen hosszú!');
    ReadLn(neved);
    kar_szam:=Length(neved);
    Write(kar_szam,' darab karaktert tartalmaz.');
    ReadLn;
End.
```

Írjunk programot, amely egy string változóba beolvassa a nevedet és kiírja a képernyő közepére.

```
Program kozepre;
Uses Crt;
Var
   hova, hossz : Integer;
   neved : String;
Begin
   ClrScr;
   Write('Hogy hívnak?');
   ReadLn( neved );
   hossz:=Length(szoveg);
   hova:=(80-hossz) div 2;
   GotoXY(hova,12);
   Write(szoveg);
   ReadLn;
End.
```

Futtasd különböző adatokkal!

Programelágazások

1. Kétágú elágazások - If..Then..Else

Legyenek a program input adatai tetszőleged valós számok. Számolja ki a hányadosukat és írja ki a képernyőre. Vizsgálja meg, hogy az osztás elvégezhető-e?

Nullával való osztásnál program megszakítás történne, hiszen ekkor művelet nem értelmezhető!

A lényeget a következő képen fogalmazhatjuk meg:

Ha szam2 egyenlő 0 **akkor** írd ki 'Az osztás értelmetlen'

különben írd ki szam1/szam2

A **ha..akkor..különben** szavak angol **(If..Then.. Else)** megfelelőjét behelyettesítve:

If szam2=0 **Then WriteLn**('Az osztás értelmetlen')

Else WriteLn(szam1/szam2)

A **feltétel** szam2=0 logikai kifejezés igaz vagy hamis volta határozza meg a szöveg kiírását vagy a művelet elvégzését.

```
Program zero;
Uses Crt;
Var
    szam1,szam2 : Real;
Begin
    ClrScr;
    Write('Kérem a számokat: ');
    ReadLn( szam1);
    ReadLn( szam2);
    If szam2=0 Then WriteIn('Az osztás értelmetlen!')
      Else WriteIn('Eredmény:',szam1/szam2 );
    ReadLn;
End.
```

Ennek alapján a feltételes utasítás általános alakja:

If feltétel **THEN** utasítás1 **ELSE** utasítás2;

A végrehajtás szabálya: ha a feltétel

- **igaz,** akkor **utasitás1** kerül végrehajtásra
- **hamis**, akkor **utasitás2** végrehajtása következik
- folytatódik a program

Az **Else ág** nem kötelező. Ha a feladat nem kívánja meg, el is hagyható.

1.1. Összehasonlítási műveletek

Jele	jelentése
=	egyenlő
<	kisebb
>	nagyobb
<=	kisebb vagy egyenlő
>=	nagyobb vagy egyenlő
<>	nem egyenlő

Az összehasonlítás operátorai is szerepelhetnek kifejezésekben. A kiértékelésben ez kerül legutoljára végrehajtásra. A hasonlító műveletek **eredménye** minden esetben egy **logikai érték. Értéke vagy igaz vagy hamis**. Azokat a kifejezéseket, **melyekben hasonlító operátorok is szerepelnek, logikai kifejezéseknek nevezzük**. Ezzel az adattípussal később részletesebben is foglalkozunk.

Egyszerű példák logikai kifejezésekre konstans adatokkal:

kifejezés	érték	kifejezés	érték
5*2 > 3	igaz	6 div 4 > 1	hamis
18 < 12	hamis	18 mod 3 = 0	igaz
5<>4	igaz	37 div 12 = 5	hamis
5<=6	igaz	6 mod 2 > 0	hamis
5 mod 2 = 1	igaz	Length('ablak')=5	igaz

Vizsgálja meg a program egy egész típusú változó értékét! Írja ki szöveggel a nullához viszonyított értékét (negatív, nem negatív vagy nullával egyenlő)!

```
Program elojel;
Uses Crt;
Var
   szam:Integer;
Begin
   ClrScr;
   Write('Add meg a számot:');
   ReadLn(szam);
   If szam<0 THEN Writeln('A szám negatív!');
   If szam=0 THEN Writeln('A szám nulla!');
   If szam>0 THEN Writeln('A szám pozitív!');
   ReadLn;
End.
```

1.2. Változók cseréje

Bemenő adat két egész szám. Döntsük el hogy az egyik bemenő adat osztója-e a másiknak?

- A feladathoz a Mod műveletet és a most ismertetett feltételes utasítást használjuk! Az előző táblázat 4. sora alapján a 18 osztható 3-al, mivel a maradék nullával egyenlő : 18 Mod 3 = 0
- Így ha a két változó osztando és oszto, akkor a feladatban a feltételes utasításban a feltétel: osztando mod oszto = 0
- Ha az osztando<oszto, megkell cserélnünk a két változó értékét, máskülönben a vizsgálatnak nincs értelme.

Az osztando:=oszto értékadó utasítás nem vezetne megoldásra, mivel az osztando változó értékét elveszítenénk! Ez megfordítva is igaz.

Hogyan lehetne felcserélni a két változó (osztando és oszto) értékét? Miként cserélhetünk ki két különböző pohárban lévő folyadékot?

A programozásban is meg van erre a lehetőség. Tehát kell egy harmadik változó! Legyen ez a változó: ment

Az értékadó utasítások:

ment:=osztando; osztando:=oszto; oszto:=ment;

1.3. Összetett utasítások (utasítás zárójel)

Az IF... Then...Else feltételes utasításban az igaz és hamis ághoz is egyetlen utasítást írtunk. Honnan fogja tudni a fordító hogy a cseréhez szükséges három utasítás mindegyikét el kell végezni?

🕮 **Több utasítást a gép a csak akkor hajt végre pl. egy feltételes ágban, ha BEGIN.....END; szavak között soroljuk fel őket. Ezt nevezzük utasítás zárójelnek.**

```
Begin
    ut1;ut2;ut3;
End;
```

☝ Ilyenkor az **End után nem szabad pontot tenni**, hiszen ez a program végét jelentené!

Ha az osztando<oszto feltétel igaz akkor végre kell hajtani a két változó cseréjét:

```
If osztando<oszto Then
    Begin
        ment:=osztando;
        osztando:=oszto;
        oszto:=ment;
    End;
```

utasításokkal bővül programunk.

☝ Ha az egyik input adat a billentyűzetről a nulla értéket kapja, akkor a fordító hibaüzenettel megszakítja a program végrehajtását. A hiba oka osztás nullával. A probléma elkerülhető egy újabb feltételes utasítással. Egy jó program nem fogad el olyan adatokat, amely a feladatnak nem megfelelő. Az egyik legnehezebb feladat a programokat az ilyen adatokra felkészíteni.

```
Program oszthatosag;
Uses Crt;
Var
   osztando,ment,oszto:Integer;
Begin
```

```
  ClrScr;
  Write('A vizsgált szám: ');
  ReadLn(osztando);
  Write('Osztó: ');
  ReadLn(oszto);
  If osztando<oszto Then
    Begin
      ment:=osztando;
      osztando:=oszto;
      oszto:=ment;
    End;
  If osztando mod oszto = 0 THEN
    WriteIn(oszto,' osztója a ',osztando)
       ELSE WriteIn(osztando,'- nak(nek) nem osztója a ',oszto);
  ReadLn;
End.
```

Készítsünk programot, amely egy egész számnak kiírja a nagyobbik páros szomszédját! ***Az input adat azonosítója*** *:* szam

Az input adatunkat vagy **eggyel** vagy **kettővel** kell növelni, attól függően hogy **páros** vagy **páratlan!**

Az értékadó utasítások: szam:=szam+1; illetve szam:=szam+2;

```
Program paros;
Uses Crt;
Var
  szam:Integer;
Begin
  ClrScr;
  Write('A vizsgált szám: ');
  ReadLn(szam);
  If szam mod 2 = 0 THEN szam:=szam+2
     ELSE szam:=szam+1;
  WriteIn('A szomszédos páros szám: ',szam);
  ReadLn;
End.
```

2. Feltételes utasítás egymásba ágyazása

A feltételes utasítás **Then (igaz) és Else (hamis) ága** egyaránt **tartalmazhat további feltételes utasításokat.** Használata a probléma jellegétől függően nem mindig kerülhető el. Ennek bemutatását az indokolja, hogy ugyanazt a feladatot bizonyos esetekben meg lehet oldani egyszerűbben is. Ehhez azonban további ismeretek szükségesek (pl.: Logikai adattípusok, műveletek).

Az előző részben már megoldottuk három feltételes utasítással a következő példát. Ezért egy kis segítséggel a megoldást rád bízzuk.

✍ Egész típusú input adatról döntsük el, hogy negatív, nulla vagy pozitív!

Magyarul leírva az egymásba skatulyázott feltételes utasításokat:

Ha a szám<0 **akkor írd ki** 'negatív'
különben ha a szám=0 **írd ki** 'nulla'
különben írd ki 'pozitív';

Sok probléma esetében nehezen kerülhető el az egymásba ágyazott **If** utasítás. A következő feladatot másképpen megoldani csak logikai műveletek segítségével lehet.

💻 *Legyenek az input adatok egy háromszög oldalai. Írjunk programot, amely eldönti, hogy a háromszög megszerkeszthető-e. Írjuk ki a megfelelő szöveget a képernyőre.*

Állítás: Egy háromszög **akkor és csakis akkor szerkeszthető meg, ha bármelyik két oldalának összege nagyobb mint a harmadik oldal.** Ha az oldalakat a, b, c jelöli, akkor az **a+b>c, a+c>b, b+c>a** feltételeknek **egyszerre kell teljesülnie**.

A megoldás érdekében alakítsuk át feltételünket: Mikor nem szerkeszthető meg a háromszög?

Állítás: Egy háromszög csak akkor **nem szerkeszthető** meg, ha **valamelyik két oldalának összege nem nagyobb mint a harmadik oldal** hosszúsága. Így, ha az **a+b<=c, a+c<=b, b+c<=a** feltételek közül **legalább az egyik igaz**, akkor a háromszög nem szerkeszthető meg.

Ha a+b<=c **akkor** írd ki 'nem szerkeszthető meg.'
különben ha a+c<=b **akkor** írd ki 'nem szerkeszthető meg.')
különben ha b+c<=a **akkor** írd ki 'nem szerkeszthető meg.'
különben írd ki 'megszerkeszthető.');

```
Program haromszog;
Uses crt;
Var
   a,b,c:Real;
Const
  mondat='nem szerkeszthető meg.';
Begin
   Clrscr;
   Write('Kérem a háromszög oldalait: ');
   ReadLn(a,b,c);
   Write('A háromszög ');
   If a+b<=c Then Write(mondat)
     Else If a+c<=b Then Write(mondat)
           Else If b+c<=a Then Write(mondat)
                 Else Write('megszerkeszthető.');
   ReadLn;
End.
```

A ReadLn utasításról még el kell mondanunk, hogy segítségével egyszerre több változó értékét is beolvashatunk. Ilyenkor az azonosítókat vesszővel kell elválasztani egymástól. A billentyűzetről megadott input adatok közé szóközöket kell tenni.

A logikai kifejezésekről már tettünk említést a hasonlító operátorokkal kapcsolatban. **Logikai kifejezés** értéke mindössze kétféle lehet, **igaz** illetve **hamis**. Angol megfelelője **True** illetve **False**.

2.1. Logikai műveletek (operátorok)

Ezek közül csak a három legfontosabb művelettel foglalkozunk, nevezetesen: **Not** (nem), **And** (és), **Or** (vagy)

A következő táblázatok tartalmazzák az egyes műveletek lehetséges eredményeit.

Not	True	False
	False	True

And	True	False
True	True	False
False	False	False

Or	True	False
True	True	True
False	True	False

2.2. Logikai kifejezések kiértékelése

Logikai kifejezést, kifejezések **összehasonlításakor** kaphatunk. Pl.:

kifejezés	érték
(8 div 5>3) **Or** (18 mod 6 =0)	**True**
(8 div 5>3) **And** (8 mod 2=0)	**False**
(**Length**('Pascal')=6) **And Not**(5<=6)	**False**
Not(12 div 3)<>18 mod 7	**True**
(8 div 5<3) **Or** (8 mod 2=0) **And Not**(5<6)	**True**

Példaképpen határozzuk meg az utolsó értékét:

(8 div 5<3) **Or** (8 mod 2=0) **And Not**(5<6) =

= (1<3) **Or** igaz **And Not** igaz = igaz **Or** igaz **And** hamis =

= True **Or** True **And** False = True **Or** False= True

☝ Ha az utolsó lépésben a kiértékelést balról jobbra végeztük volna el, akkor helytelen eredményt kapunk.

📖 **Precedencia szabály:**
Az eddig tanult operátorok végrehajtásának sorrendje a kifejezésekben

I. ( ) : Először a zárójelben lévő kifejezések értékelődnek ki

II. Függvények

A mi példáinkban pl.: a Length függvény

III. előjelek, Not (egy operátort tartalmazó műveletek)

IV. Div, Mod, And, * , /

V. +, –, Or

VI. Relációk

☝ **Azonos sorszámú operátorok között a végrehajtás balról jobbra történik!**

✍ Ellenőrizd a táblázat eredményeit! Értékeld ki lépésenként a műveletek sorrendjének betartásával az előző táblázat kifejezéseinek az értékeit! **Írasd ki az értéküket** a monitorra!

Ha az eddigiek megértése nem okozott nagyobb problémát, akkor a következő program megértése sem lesz nehéz.

💻 *Legyen **input adatunk integer** típusú. Ha **kettővel osztható,** de **hárommal nem,** akkor írassuk ki, hogy 'A feltétel igaz', ellenkező esetben 'A feltétel hamis'.*

Legyen programunkban a két logikai munkaváltozónk azonosítója kettovel és harommal. Ezekben tároljuk az oszthatóság logikai értékét. A két változót szintén deklarálnunk kell:

🕮 **A logikai változókat a Boolean alapszóval deklaráljuk.**

```
Var
   kettovel és harommal : Boolean;
Const igaz =True;
      hamis = False;
```

A feladatban szerelplő feltétel csak akkor lesz igaz, ha a (szam **mod** 2=0) **And Not** (szam **mod** 3=0) logikai kifejezés igaz.

```
Program logikai;
Uses Crt;
Var
   szam:Integer;
   kettovel, harommal:Boolean;
Begin
   ClrScr;
   Write('Kérem a számot: ');
   ReadLn(szam);
   Write('A feltétel ');
   kettovel:=szam mod 2=0;
   harommal:=szam mod 3=0;
   If kettovel And Not harommal Then Write('igaz.')
     Else Write('hamis.');
   ReadLn;
End.
```

Mindenki ismeri iskolai tanulmányaiból Pitagoras tételét. Feladatunkban összekapcsoljuk a tételt az előbbi fejezet háromszöggel kapcsolatos példájával.

Legyenek az input adatok egy háromszög oldalai. Döntse el a program, hogy a háromszög derékszögű-e. Vizsgálja meg azt is, hogy az adatok helyesek voltak-e és írjon ki ennek megfelelő szöveget!

Soroljuk fel a program feladatait:

- Adatellenőrzés:
  - A háromszög megszerkeszthető-e?

    A mostani programban a **szerkesztéshez** a szükséges és elégséges feltételt adjuk meg és ennek alapján írjuk meg a forrásprogramot. A feltételes utasításunkban mind három feltételnek (**a+b>c, a+c>b, b+c>a**) egyszerre kell teljesülnie. Ebben az esetben az **And** logikai műveletet fogjuk használni. Ennek értékékét egy helyes azonosítójú logikai változó tárolja:

    helyes:= **(a+b>c) And (a+c>b) And (b+c>a)**
  - A háromszög derékszögű-e?

    Nem elég csak két oldal esetében vizsgálni az összefüggéseket. A háromszög akkor és csakis akkor derékszögű, ha van olyan két oldala, melyekre teljesül Pitagoras tétele. Ezért az **Or** logikai műveletet fogjuk használni. Ezt a pitagoras változóban tároljuk.

    pitagoras := $(a^2 + b^2 = c^2)$ **Or** $(a^2 + c^2 = b^2)$ **Or** $(b^2 + c^2 = a^2)$
  - A két feltétel egymástól nem független, ezért csak a két elágazás egymásba ágyazásával oldható meg! Magyarosan leírva:

    Ha nem helyes **akkor** írd ki 'helytelen adat'

    különben

    Ha pitagoras **akkor** írd ki 'derékszögű'

    különben írd ki 'nem derékszögű';

Mivel a hatványozás **ismételt szorzás,** ezért pl.: az a^2 **helyett a*a** műveletet fogjuk használni.

```
Program haromszog2;
Uses crt;
Var
   pitagoras,helyes : Boolean;
```

```
    a,b,c : Real;
Const kiir='derékszögű háromszög';
Begin
    Clrscr;
    Write('Kérem oldalakat: ');
    ReadLn(a,b,c);
    helyes:=(a+b>c) And (a+c>b) And (b+c>a);
    pitagoras:=(a*a+b*b=c*c) Or (a*a+c*c=b*b) Or (c*c+b*b=a*a);
    If Not helyes Then Write('Helytelen adat')
       Else
          If pitagoras Then Write(kiir) Else Write('Nem ',kiir);
    ReadLn;
End.
```

☝ A Pascal nyelvben van olyan függvény is, amely kiszámítja tetszőleges numerikus adatnak (bemenő paraméter) a négyzetét. Ez a függvény az **SQR.**

Pl.: **Sqr**(-2)=4; **Sqr**(3)=9; **Sqr**(2)=4; **Sqr**(2.5)=6.25; **Sqr**(a)= a^2

3. Többirányú elágazás

Mielőtt folytatnánk a programelágazásokat, meg kell ismernünk a **numerikus (számtípusú) adatok további csoportosításának lehetőségeit**. Nem utolsó szempont, hogy az adatokra a fordító **mennyi helyet (Byte-ot) készít elő a memóriában**.

A valós adatok közül a **Real csak egy az öt típus között**. A többi négy típus használatáról a fordítót külön tájékoztatni kell „kapcsolók" segítségével. Ezek használatához matematikai társprocesszor, vagy annak emulátora szükséges.

Az eddigi feltételes utasításokkal kapcsolatba észrevehettük, hogy több feltétel esetén **nagyon sok If...Then...Else** szerkezetet kellett használnunk. **Összetett feltételek** esetén pedig **logikai műveletekre** is volt szükség. Bizonyos esetekben **ezek elkerülhetőek**. A **többirányú elágazásnak is lesznek korlátai**. Célszerű **használatát** – mint példáinkon is látni fogjuk –, a **megoldandó feladat** sugallhatja.

3.1. Többágú szelekció – Case

A fenti új fogalom bevezetésére legyen egy egyszerű példa.

Egy labdarugó mérkőzésről az 1,X,2 bemenő adatok alapján írassuk ki a képernyőre szövegesen, hogy a hazai csapat szempontjából milyen lett az eredmény!

Az eddigi ismereteink alapján három feltételes utasítással, vagy egymásba ágyazott szerkezettel lehetne megoldani.

```
Program foci1;
uses Crt;
   Var
    tipp:Char;
    Const k i='A hazai csapat';
Begin
   ClrScr;
   Write('Add meg a tippet:');
   ReadLn(tipp);
   Case tipp Of
          '1': Then WriteLn(ki,' győzött.');
          'X': Then WriteLn(ki,' döntetlent ért el.');
          '2': Then WriteLn(ki,' vereséget szenvedett.');
              Else WriteLn('Az adat nem értékelhető');
   End;
   ReadIn;
End..
```

Ennél a szelekciónál a program rendre összehasonlítja a **tipp változó értékét az '1', 'X', '2' karakterekkel.** Ennek alapján írja ki a képernyőre a megfelelő szöveget. Ha az összehasonlítás minden esetben **hamis,** akkor az **Else ág** kerül végrehajtásra.

A többirányú, szelektív elágazást a következőképpen írhatjuk le. A szelekció magyarul válogatást jelent.

```
Case szelektor Of
    állandók1:utasítások1;
    állandók2:utasítások2;
    állandók3:utasítások3;
    ..... ;
    ..... ;
       Else utasítások;
End;
```

Fontosabb szabályok:

- A **szelektor** kifejezés kiértékelésének eredménye az eddig tanult típusok közül csak **egész, karakter vagy logikai típus lehet**. Tehát valós és string nem lehet az eredmény. A szelektor természetesen **lehet egyetlen változó vagy konstans**.
- **A szelektorhoz** csak vele **azonos típusú állandókat** vagy konstans kifejezéseket lehet hasonlítani!
- Ha egy ághoz **több konstans** tartozik, akkor azokat **vesszővel választjuk el** egymástól.
- Ha egy ághoz **több végrehajtandó utasítás** tartozik, akkor azokat **utasítás zárójelek** között soroljuk fel **(Begin.....End;)**.
- Az Else ágban összetett utasítások esetén szintén használni kell az utasítás zárójelet. Az **Else ág** el is hagyható!
- A **Case, Of** szavak fenntartott szavak, tehát azonosítóként nem használhatók!

Működése:

A fordító az állandók1, állandók2,.....értékeit **összehasonlítja a szelektor kifejezéssel.** Ha az **összehasonlítás logikailag igaz**, akkor a **vezérlés** a szelekciót lezáró **End**-re **adódik** (a szelekció befejeződik). **Ezért maximum csak egy feltételes ágat hajt csak végre.** Ha egyetlen összehasonlítás sem igaz, akkor a további vezérlés az **Else ág** feladata. Ha **nem volt Else ág**, akkor a feltételektől függően az is lehet, hogy egyetlen feltételes részt sem hajt végre.

*Írjunk programot, amely két valós típusú számmal elvégzi a négy alapművelet (+ – * /) valamelyikét. A program a műveleti jel alapján válasszon. Az eredményt tároljuk egy változóban és két tizedes helyiérték pontossággal írassuk ki.*

Input: egyik, masik valós és jel karakter típusú változók.

A programban a szelektív elágazást fogjuk használni. A szelektor a jel karakter típusú változó lesz. A hasonlító konstansok, pedig a műveleti jelek. Illik még arra is figyelni, hogy ha masik változó értéke nulla, akkor ne végezzük el az osztást.

```
Program alapmuvelet;
Uses Crt;
Var
   jel:Char;
   egyik,masik:Real;
Begin
   ClrScr;
   Write('Milyen műveletet végezzek el? (+,-,*,/): ');
   ReadLn(jel);
   Write('Kérem a két számot: ');
   ReadLn(egyik,masik);
   Write('Eredmény: ');
   Case jel Of
     '+': Write(egyik+masik:0:2);
     '-': Write(egyik-masik:0:2);
     '*': Write(egyik*masik:0:2);
     '/': If masik=0 Then
             Write('nincs. A nullával való osztás értelmetlen.')
                Else Write(egyik/masik:0:2);
   End;
   ReadLn;
End.
```

Input adat legyen Byte típusú, amely a nap valamelyik órája. Szelektáljunk az idő szerint és írjunk ki ezekre az időpontokra megfelelő tevékenységet (pl. 12 óra ebéd). A konstansok legyenek 7, 8, 10. 12. 14, 15, 19, 22. Ha az input adat nem esik ezen értékek közé, akkor erről is tájékoztasson a program (pl.: Az időpont értéke nem lehet nagyobb mint 24).

idő (feltétel)	A képernyőn megjelenő információ
mennyi=7	Reggeli
mennyi=8	Valószínű, hogy elkéstél az iskolából.
mennyi=10	Tízórai
mennyi=12	Ebéd.
mennyi=14	Tanulnod kell.
mennyi=15	Ha éhes vagy uzsonnázz.

mennyi=19	Vacsora idő.
mennyi=22	Aludnod kell.
mennyi>24	Nem ismered az órát
Ha az előzőek nem teljesülnek	Nem vagyok mindentudó

```
Program napszak;
Uses Crt;
Var mennyi:Byte;
Begin
   Clrscr;
   WriteLn('Hány óra van?');
   WriteLn('Megmondom milyen tevékenység tartozik hozzá: ');
   ReadLn(mennyi);
   Case mennyi Of
       7  :Write('Reggeli.');
       8  :Write('Valószínű, hogy elkéstél az iskolából.');
       10 :Write('Tízórai.');
       12 :Write('Ebéd.');
       14 :Write('Tanulnod kell.');
       15 :Write('Ha éhes vagy, uzsonázz!');
       19 :Write('Vacsoraidő.');
       22 :Write('Aludnod kell.');
       Else
           If Mennyi>24 Then Write('Nem ismered az órát!')
           Else Write('Nem vagyok mindentudó.');
   End;
   ReadLn;
End.
```

3.2. Résztartomány-típus

- **Egésztípusú adatok csoportosítása:**

Öt típust különböztetünk meg, melyek közül már egyet (**Integer**) tanultunk. Foglaljuk táblázatba előjel és azon belül nagyságrend szerint növekvő sorrendben.

Típus deklaráció	méret (Byte)	értékhatár	jelentése	
ShortInt	1	–128..127	rövid egész	előjeles
Integer	2	–32768..32767	egész	előjeles
LongInt	4	-2^{31}.. $2^{31}-1$	hosszú egész	előjeles
Byte	1	0..255		előjel nélküli
Word	2	0..65535	szó	előjel nélküli

- Példák résztartományokra:

résztartomány	*értékei*
–8..0	–8, –7, –6, –5, –4, –2, –3, –1,0
'0'..'9'	számjegy karakterek
'a'..'z'	Az angol ABC kisbetűi
'A'..'Z'	Az angol ABC nagybetűi
10..99	Kétjegyű pozitív egész számok
–999.. –100, 100..999	Háromjegyű egész számok
–9999.. –1000	négyjegyű negatív egész számok
#65..#68	karakterkódok: 65, 66, 67, 68
	az A, B, C, D betűk
'a'..'c' , 'k'..'o'	Betűk: a, b, c, k, l, m, n, o

A tartományba **beletartozik a kezdő és végérték is**. A két adat közé **két pontot** kell tenni. Ha a kifejezett résztartomány nem folytonos, akkor több résztartománnyal fejezzük ki, vesszővel elválasztva őket. Értékei a **LongInt** kivételével **egész, karakter és logikai típusú** adatok lehetnek.

*Kérjünk billentyűzetről egy karaktert. Döntsük el róla hogy az angol ABC **kisbetűje, nagybetűje, számjegy vagy egyéb jel.***

A felhasználható résztartományok:

Karakterek	Résztartományok jelölése
kisbetűk:	**'a'..'z'**
nagybetűk:	**'A'..'Z'**
számjegyek:	**'0'..'9'**

Az **egyéb jel** információt az **Else** ág segítségével írjuk ki.

```
Program karakter_szelekt;
Uses crt;
Var ch:Char;
Begin
    Clrscr;
    Write('Kérek egy karaktert: ');
    ReadLn(ch);
    Write('Ez a karakter ');
    Case ch Of
        'a'..'z':Write('Kisbetű.');
        'A'..'Z':Write('Nagybetű.');
        '0'..'9':Write('Számjegy.');
        Else Write('Egyéb jel.');
    End;
    ReadLn;
End.
```

Bemenő adat egy Integer típusú szám. Értékeljük a szerint, hogy egyjegyű, kétjegyű, háromjegyű, négyjegyű.

```
Program szamjegy;
Uses Crt;
Var szam: Integer;
Begin
    ClrScr;
    Write('Adja meg a számot: ');
    ReadLn(szam);
    Write('Ez a szám: ');
    Case szam Of
          -9..9: Write('Egyjegyű');
          -99..-10,10..99: Write('Kétjegyű');
          -999..-100,100..999: Write('Háromjegyű');
          -9999..-1000,1000..9999: Write('Négyjegyű');
    End;
    ReadLn;
End.
```

Ciklusok I.

A továbbiakban egy magasabb szintű programszervezési móddal foglalkozunk, nevezetesen az ismétlődő problémák egyik megoldásának lehetőségeivel. A ciklusok szemléletes bemutatatásának érdekében további utasításokat, függvényeket és adatszerkezeteket is meg kell ismernünk!

1. A számláló (FOR) ciklus

Már sokszor előfordultak a következő fogalmak: **algoritmus, ciklus**. Az első fogalom jelentése leegyszerűsítve: egy **probléma megoldásának általános megfogalmazása** egy adott eszközcsoportra (pl.: számítógép), függetlenül a csoporton belüli (programnyelv) konkrét lehetőségektől.

A ciklus tevékenységek sorozatának ismétlése. Erre számtalan példát lehetne említeni a köznapi életből is (pl.: napi szükséges étkezések, napszakok, évszakok váltakozása, bioritmus stb.).

Mikor fejeződnek be ezek a ciklusok? Ha jól belegondolsz akkor ez a körülményektől is függ. A Földön egy évben 12 hónapot különböztetünk meg. Ez állandóan ismétlődik (**számláló ciklus**). Más körülmények között, ez bizonyos **feltételektől** (pl.: csillagászati összefüggések) **függ.**

> 🕮 **A számláló ciklus adott számú lépésben hajtja végre ugyanazt az utasításcsoportot.**

Írjuk le az algoritmusát 1 - 8 -ig számok négyzetének az előállítására.

A probléma mondatszerű leírása: pl.: nyolcig

```
ciklus szam:=1-től 8-ig csináld
kezd
    negyzet:=szam*szam
    írd ki negyzet
ciklus vége
```

Közelítsük a megfogalmazást Pascal nyelven.

```
For szam:= 1 To 8 Do
Begin
negyzet:=szam*szam;
WriteLn(negyzet);
End;
```

A szavak jelentése: **For** (-tól), **To** (-ig), **Do** (csináld)

ciklusváltozó kezdőérték végérték

For	azonosító	:=	kifejezés1	**To**	kifejezés2	**Do**
Begin						
.....						
utasítás1;						
utasítás2;	ciklusmag (amely ismétlődik)					
utasítás3;						
.....						
End;	ciklus vége					

Következő programunk alapja az előzőző probléma:

A ciklus pontos működésére a következő példa után visszatérünk!

*Legyen **két input** adatunk, melyek **egész számok**. Írjunk programot mely kiírja az ebbe a tartományba tartozó összes egész szám négyzetét. A kiíratást mezőszélességek segítségével formázzuk.*

A programunkban használt változó azonosítók:

azonosító	típusa	azonosítók feladata
k	egész: **Integer**	A számláló ciklus **változója**
kezdi	egész: **Integer**	**Input** adat és a **ciklusunk kezdőértéke**
befejezi	egész: **Integer**	**Input** adat és a For **ciklus végértéke**
negyzet	egész: **Word**	A négyzetszámok **tárolása**

```
Program negyzetszamok;
Uses Crt;
Var
   k, kezdi, befejezi : Integer;
   negyzet : Word;
Begin
```

```
    ClrScr;
    Write('Melyik számmal kezdjem? ');
    ReadLn(kezdi);
    Write('Melyik legyen az utolsó szám? ');
    ReadLn(befejezi);
    For k := kezdi To befejezi Do
        Begin
          negyzet := k*k;
          Write(negyzet:20);
        End;
    ReadIn;
End.
```

A negyzet változó **értéke mindig nem negatív,** ezért matematikai szempontok miatt deklaráltuk **Word** típusra. Számítástechnikai szempontból (**nagyobb pozitív értékhatár**) lehetne LongInt és Real típusú is. Az utóbbi esetben, ha ezen nem igazítunk, normál alakban kapjuk az eredményt. **Ha nagyobb** számokra kipróbálod (pl.: a végérték **256**), akkor a program egyrészt **görgeti a képernyőt** (szépséghiba), másrészt matematikailag **nem megfelelő értékeket ír ki (adathiba).** Ennek oka, hogy ilyenkor az eredmény nagyobb mint a Word tartomány felső határa.

📖 **A For ciklus működése vázlatosan:**

- **A** kifejezés1 **és** kifejezés2 **kiértékelése.** A mi példánkban a kifejezések mindegyike egyetlen változó volt (kezdi és befejezi). Természetesen konstansok is lehetnek.
- **Típusegyeztetés**, különböző típusok esetén hibaüzenet. A ciklusváltozó felveszi a kezdő értéket
- **Vizsgálat,** vagyis a fordító **összehasonlítja** a **ciklusváltozó** és **kifejezés2** (végérték) értékét.
- Ha a **ciklusváltozó nem nagyobb mint a végérték, akkor végrehajtja** a ciklusmagban lévő utasításokat, **ellenkező esetben kilép** a ciklusból.
- Ha a **ciklusváltozó kisebb mint a végérték, akkor növeli eggyel** a **ciklusváltozót** (lépteti), vagyis veszi a soron következő értéket

A For ciklus tehát az előltesztelő ciklusok csoportjába tartozik. A ciklus végét az **End;** alapszó jelezi.

☝ **Ha a For ciklus magjában egyetlen utasítás** van, akkor a ciklusmagot közrefogó (**Begin...End**;) **elhagyható**. Ilyenkor a **Do**-hoz legközelebb lévő **pontosvessző** jelzi a ciklus végét. A **For, To, Do** a fordító számára **fenntartott szavak**, tehát azonosítóként nem lehet használni.

Írassuk ki a képernyőre egy húsz karakternél nem hosszabb szöveget a következőképpen:

- A képernyő teljes szélességében ahányszor kifér
- Minden kiírás egy sorral lejjebb kezdődjön, de ugyanabban az oszlopban mint ahol a kurzor előzőleg volt. pl.:

SZÖVEG
SZÖVEG
SZÖVEG
SZÖVEG
.....
SZÖVEG

```
Program csusztatva;
Uses Crt;
Var
    nev:String[20];
    i, hanyszor:Byte
    ClrScr;
    Write('Kérem a nevet: ');
    ReadLn(nev);
    hanyszor:=80 div Length(nev);
    For i := 1 To hanyszor Do
      Write(nev,#10);
    ReadIn;
End.
```

A kiíratást végző **For** ciklus **egyetlen utasítást** tartalmaz. A képernyő 80 oszlop szélességű és a Length(nev) a kiíratandó karakterek számát adja meg. Így a hanyszor:=80 div Length(nev) értékadó utasítással lehet kiszámolni a ciklus végértékét. A #10 karakter hajtja végre a soremelést.

Az előző programban is van egy eddig nem ismeretetett deklaráció (**String;** helyett **String[20];**). **Mindkettő helyes**, de a második takarékosabb (az **adat mérete miatt**). A feladatban feltételként szerepelt, hogy a

string hossza nem több mint **20 karakter**. A fordítót tájékoztathatjuk **[ ] zárójelek** között **a karakterek maximális számáról**. Ebben az esetben hiába adtunk meg hosszabb string adatot, a ReadLn utasítással a beolvasott változóba csak az első 20 db karakter lesz tárolva. **Ellenkező esetben a fordító** a kelleténél nagyobb **méretet** foglal le a memóriában.

☝ Ha a **Do után közvetlenül pontosvessző van ,** akkor a ciklusmag üres. Ilyenkor az utána lévő programrészletet csak **egyszer hajtja végre**. Jelen esetben nem kellett használnunk a **Begin...End; utasítás zárójelet**, mivel a ciklusmag egyetlen utasítást tartalmazott.

2. Képernyő, billentyűzet kezelése

2.1. Képernyőszínek és ablak beállítása

- **Képernyőszínek:** TextColor, TextBackGround eljárások

Szokás szerint kezdjük az angol kifejezések magyar jelentésével:

Text	szöveg
Color	szín
BackGround	háttér

A jelentése is mutatja, hogy segítségével a kiírt **karakter** színe(**TextColor**) illetve a karakter **háttérszíne** állítható (**TextBackGround.**

Ez még persze kevés. A bemenő paraméterek (amit utána kerek zárójelbe írunk) a színeket adják meg. Összesen tizenhat féle szín lehetséges, melyet megadhatunk szám konstansokkal (0-tól 15-ig), vagy a neki megfelelő angol szavakkal (konstans azonosítók).

Foglaljunk táblázatba a 16 lehetséges eset közül néhányat:

☝ Aki nem tud angolul az számokkal is helyettesítheti a színeket.

konstans azonosítókkal	**konstans**	**hatása**
TextBackGround(**black**)	0	háttér fekete
Textcolor(**blue**)	1	írás színe sötétkék
TextBack**Ground(green)**	2	háttér zöld
Textcolor(**red**)	4	írás színe piros
TextBackGround(**white**)	15	háttér fehér
Textcolor(**Brown**)	6	írás színe barna
TextBackGround(**Yellow**)	14	háttér sárga

Természetesen szín paraméternek nemcsak konstansokat lehet használni, hanem Byte típusú változókat is.

A **TextColor** egyszerűen a megadott **színre állítja a kurzort**, az írás színét. **A TextBackGround** a karakterek mögötti színt változtatja. A **képernyőtörlés(ClrScr)** utasítás midig a legutóbb beállított háttérszínnel törli le a képernyőt (Mintha festékes szivaccsal törölnéd a táblát). pl.:

TextBackGround(blue)
ClrScr
TextBackGround(yellow)
Textcolor(red)

programrészlet hatására három féle színt látunk: Az egész képernyő kék lesz, az írás háttere sárga, a karakterek színe pedig piros.

✍ Az előbbiek alapján milyen lesz az egész képernyő színe, ha azt ismét letöröljük?

💻 *A bemenő adat legyen egy név, amely **maximum húsz karakter**. Írjuk ki a nev változót **piros színű képernyő** közepére, úgy, hogy a **karakterek színe fekete**, **háttere** pedig **zöld** legyen.*

✍ A fenti példa és a táblázat segítségével írd fel a színeket beállító utasításokat!
Melyik függvény számítja ki a karakterlánc hosszát?
Hány oszlopa van alapesetben a képernyőnek?
Hogyan lehet kiszámítani a karakterhosszból és a képernyő oszlopainak a számából, hogy a kiíratás melyik oszlopban kezdődjön? Melyik utasítás helyezi a kurzort egy adott pozícióra?

```
  Program szinek;
Uses crt;
Var
     hova:Byte;
     nev:String[20];
Begin
     Clrscr;
     Write('Kérem a karakterláncot! ');
     ReadLn(nev);
     TextBackGround(red);
     ClrScr;
     TextBackGround(green);
     TextColor(black);
```

```
    hova:=(80-Length(nev)) div 2;
    GotoXY(hova,12);
    Write(' ',nev,' ');
    ReadLn;
End.
```

✍ **Próbáld ki** a programot húsznál hosszabb karakterláncokra is!

- **Ablakok létrehozása:** Window eljárás

  Window(oszlop1,sor1,oszlop2,sor2)

Segítségével a karakteres képernyő méretét lehet beállítani. Bemenő paraméterei lehetnek konstansok, változók, kifejezések. Az oszlop1 és sor1 az ablak bal felső, az oszlop2 és sor2 a jobb alsó pozícióinak az értékeit jelöli.

Példák:

Window(1,1,80,25)	a teljes karakteres képernyő
Window(1,1,40,25)	a képernyő bal oldali fele
Window(41,1,80,25)	a képernyő jobb oldali fele
Window(1,1,80,5)	a képernyő felső ötöde

💻 *Osszuk fel a képernyőt függőlegesen két ablakra. Az egyik háttere legyen piros, a másiké zöld. Írjunk ki az ablakokba fehér színnel egy szöveget.*

```
Program ablakok;
Uses Crt;
   Const mondat='Ez a képernyő ';
Begin
   TextColor(White);
   Window(1,1,40,25);
   TextBackGround(red);
   ClrScr;
   Write(mondat,'bal oldala');
   TextBackGround(green);
   Window(41,1,80,25);
   ClrScr;
   Write(mondat,'jobb oldala');
   ReadLn;
```

```
End.
```

A képernyő beállítások minen esetben arra az ablakra vonatkoznak, amelyet legutóbb hoztunk létre. A Window eljárás, a kurzort mindig az aktuális ablak bal felső sarkába helyezi. Ha sok ablakot nyitottunk meg és azokat felváltva használjuk, akkor a kurzor pozíciókat minden esetben tárolni kell. Ellenkező esetben a kiírást nem lehet vezérelni.

2.2. Olvasás billentyűzetről a Readkey függvénnyel

Változtassuk meg a képernyő háttérszínét az összes lehetséges módon.

Mielőtt elkezdenénk programunk írását, beszéljük meg a további lehetőségeket. A karakteres képernyő háttérszínét, mint láttuk **0-tól 15-ig terjedő numerikus konstansokkal** is jellemezhetjük. Tehát kézenfekvő **ezekkel az értékekkel egy For ciklus szervezése**. Gondoskodnunk kell a **cikluson belül egy megszakításról** (program felfüggesztéséről) **is**. Ellenkező esetben a képernyőszínekből nem látnánk semmit. Erre megoldás volt eddig a paraméter nélküli ReadLn eljárás, amely egy <**Enter**> lenyomására várt. E helyett van egy sokkal jobb, a **Readkey függvény**. **Használata** nagyon sokat **segít a billentyűzet kezelésében, figyeltetésében.** A program után az előzőekre visszatérünk.

Értéke **karaktertípusú,** így egy értékadó utasítás jobb oldalán szereplő azonosító ugyanezt az értéket kapja.

```
Var varokoz:Char
varakoz:=Readkey;
```

Működése: Az előbbi értékadó utasításhoz érve a **fordító megszakítja a program futását**. A program csak **akkor folytatódik, ha lenyomunk egy billentyűt.** Az ennek megfelelő értéket példánk alapján a varakoz **karakter azonosító veszi fel**. A karakterről **a monitoron nem kapunk tájékoztatást** („vakon gépelünk"). Ha szükség van a kiíratására, akkor azt megtehetjük a **Write** utasítással.

Ezzel a kiegészítéssel a kész program:

```
Program hatterszinek;
Uses crt;
Var
    szinek:Byte;
```

```
    varakoz:Char;
Begin
    ClrScr;
    For szinek:=1 to 15 Do
    Begin
    TextBackGround(szinek);
    ClrScr;
    varakoz:=ReadKey;
    End;
End.
```

☝ Ha programunk rosszul szervezett („sokat szemetel") az megzavarhatja az előbbiek szerinti működést. Ugyanis a fordító minden lenyomott billentyű **kódját egy pufferbe teszi** („lyukas zsákba"). Az előbb említett függvény ezt a billentyűzet puffert vizsgálja meg. Ha ott talál karakterkódot, azt kiolvassa, így a program várakozás nélkül folytatódik. A pufferből midig a legelőször bekerült kódot veszi ki. Ha üres a puffer akkor ténylegesen vár egy billentyű lenyomására. A probléma természetesen kezelhető, ha megismered a ciklusok szervezésének további lehetőségeit.

2.3. A WhereX és WhereY függvények

Segítségükkel lekérdezhetjük a képernyőn legutoljára kiírt karakter utáni oszlopszámot és sorszámot (a where jelentése *hol*). A képernyő kezelése akár a ClrScr vagy GotoXY utasításokkal könnyedén megvalósítható.

A **WhereX** az **oszlopszámot** (függőleges), **WhereY** az **sorszámot** (vízszintes) adja. **Típusa Byte**. Nagyon sok output adat esetén a képernyőn lévő információt nem tudjuk elolvasni, mivel a képernyő „felfelé" mozog. Ilyenkor a megoldás az, hogy a kurzor helyétől függően (**Wherey**) felfüggesztjük a programot (**ReadKey**), és rendet teszünk (**ClrScr**).

Az ilyen jellegű feladatok megoldására mindig használható a következő kis programrészlet:

Ha kurzor a 25. sorba kerül, akkor várakozás majd képernyőtörlés

```
If WhereY=25 Then
    Begin
```

```
    Ch:=ReadKey;
    Clrscr;
  End;
```

Egészítsd ki a **program negyzetszamok** *For ciklusát ezzel a programrészlettel!*

A ciklusváltozó és a For ciklus kifejezéseinek értéke **nem lehet sem valós, sem string típusú.**

Az eddigi példáinkban láttuk, hogy a ciklusváltozó, kezdőérték és végérték is **egész típusú adatok** voltak. A pascal érdekessége, hogy a tanul típusok közül **karakter vagy logikai típusúak is** lehetnek. Ez könnyen érthető, hiszen a nyelv ezeket az adatokat kódszámukkal kezeli. A kódszámok kölcsönösen egyértelműen meghatározzák az illető adatot. Így a For ciklus léptethető, hiszen minden ilyen típus esetén meghatározható az előző és következő adat is (sorszámozott). Tehát az **egész, karakter és logikai típusú adatokat** összefoglaló néven **sorszámozott típusnak is nevezhetjük**.

Írjuk ki az angol ABC összes kisbetűjét.(A kódrendszerben a kisbetűk egymásután következnek)

```
Program angol_abc;
Uses crt;
   Var betu:Char;
Begin
   ClrScr;
   WriteLn;
   For betu:='a' To 'z' Do
      Write(betu,' ');
   ReadLn;
End.
```

Alakítsd át a For ciklust úgy, hogy nagybetűket írjon ki! Hogyan jelölhetjük a konstans karaktereket kódszámuk segítségével?

Írjuk ki a képernyőre az összes ASCII karaktereket! Ehhez csak az előzőleg ismertetett program 8. sorában módosítanod kell a ***For ciklust*** *a következőképpen:*

For betu:=#0 **To** #255 **Do**

Természetesen a képernyőn nem látunk minden karaktert, hiszen a kiírt karakterek között vannak pl.: a vezérlő karakterek is.

Hogy jobban láthatóak legyenek a kiírt karakterek, egészítsük ki a módosított For ciklust várakozással is.

```
Program ASCII_karakterek1;
Uses crt;
   Var
   varakoz,betu:Char;
Begin
ClrScr;
   WriteLn('Az összes karakter :');
   For betu:=#1 To #255 Do
       Begin
         Write(betu,' ');
         varakoz:=ReadKey;
       End;
End.
```

A For ciklusban a **Readkey** függvénnyel a billentyűlenyomást figyeltetjük. Így minden karaktert egy tetszőleges **billentyű leütése után fog kiírni**. Ha jó a szemed és a füled, akkor a vezérlőkarakterek hatását érzékelni fogod!

2.4. A Chr és Ord függvények

Az eddigiek során láthattuk, hogy a **karakter konstansokat # jel és kódszámuk** segítségével is jelölhetjük. Ezután a jel után természetesen nem írhatunk változó azonosítót vagy kifejezést!

Konstans karakterek jelölési módjai					
'A'	'a'	'3'	'+'	'd'	'D'
#65	#97	#51	#43	#100	#68
#$41	#$61	#$33	#$2B	#$64	#$44
Chr(65)	Chr(97)	Chr(51)	Chr(43)	Chr(100	Chr(68)

A Chr függvény segítségével egész (Byte) típusú változók értékeivel is meghatározható a neki megfelelő karakter. Bemutatására tekintsük a következő táblázatot. A Byte típusú változónk azonosítója legyen: kod karakter azonosító pedig betu.

betu:=Chr(kod)	
kod változó értéke	**betu** változó értéke
65	'A'
97	'a'
51	'3'
43	'+'
10	soremelés

Tehát a Chr(kifejezés) függvény bemenő paramétere mindig egy Byte típusú érték, eredménye pedig a kódnak megfelelő karakter.

Legyen két Byte típusú input adatunk. Írassuk ki azon karaktereket, melyek kódja a két érték közötti tartományba esik!

```
Program Ascii_karakterek2;
Uses crt;
Var
   kod, kodalso, kodfelso:Byte;
   varakoz, betu:Char;
Begin
   ClrScr;
   Write('Kérem az alsó kódot : ');
   ReadLn(kodalso);
   Write('Kérem az felső kódot :');
   ReadLn(kodfelso);
   WriteLn('A karakterek:');
   For kod:=kodalso To kodfelso Do;
      Begin
         betu:=Chr(kod);
         Write(betu,' ');
         varakoz:=ReadKey;
      End;
End.
```

Az **Ord** működése **fordított a Chr** függvényhez viszonyítva. Ha bemenő paramétere **Byte típus érték**, akkor **kiértékelésének eredménye karaktertípusú**. A következő táblázat bemutatja, hogy hogyan lehet segítségével karakterkódokat meghatározni. Legyen: **kod Byte**, a **betu karakter** típusú azonosító.

kod:=Ord(betu)	
betu változó értéke	kod változó értéke
'A'	65
'a'	97
'3'	51
'+'	43
<Esc>	27
<Enter>	13

A Chr függvény értéke mindig karakter típusú, ez az **Ord** függvény paraméterének megfelel. Az **Ord** függvény nem pontosan a megfordítottja a Chr függvénynek. Annál többet tud. Úgy is fogalmazhatnánk, hogy van olyan bemenő értéke (paramétere), melyet fordítva átalakítani nem tud a **Chr** függvény. **Az Ord függvény karakterhez a kódját, egésztípushoz önmagát, logikai típushoz 0 vagy 1-et rendel.**

Pl.: logikai paraméterekre az Ord függvény esetén

Ord(False)=0 **Ord**(True)=1 **Ord**(6 **mod** 2 = 0)= 1

Ord(4 < 3)=0 **Ord**(4 > 3)=1 **Ord**(16 **div** 5 = 2)= 0

A logikai típusra való alkalmazása sok feladat esetében egyszerűbb megoldásokra ad lehetőséget.

Ezután a kis kitérő után nézzünk egy példát az Ord függvény használatára.

*Írassuk ki az **angol ABC nagy betűinek a kódját**. A kiíratásnál minden kód előtt legyen ott a megfelelő betű is. A betűket 2-es, a kódszámokat 3-as mezőszélességbe írjuk a képernyőre.*

```
Program ABC_kod;
Uses crt;
Var
    kod:Byte;
    varakoz,betu:Char;
```

```
Begin
    ClrScr;
    WriteLn('Az angol ABC nagybetűinek kódjai :');
    For betu:='A' To 'Z' Do
        Begin
          kod:=Ord(betu);
          Write(betu:2,kod:3);
        End;
    varakoz:=ReadKey;
End.
```

✍ Alakítsd át a programot úgy hogy a **kis betűk kódjait** írja ki!

✍ Alakítsd át a programot úgy hogy az angol ABC első tíz nagy betűjének **kódját** írja ki!

3. String típusú adatok kezelése

3.1. Stringek összehasonlítása

Itt csak szövegtípusú adatok tömbként való kezelését mutatjuk be. Emlékeztetésül elevenítsük fel, hogy mit tanultunk a String adatokról:

- **A String AscII kódú(0-255) karakterek sorozata**
- **Két szöveg azonos ha karakterei rendre megegyeznek.**

Hogyan történik a stringek összehasonlítása? Az első a karakterek száma. Ha ez **különböző**, akkor nem azonosak. **Ha megegyezik**, akkor (mint az elsős diák aki most tanul olvasni) **az egyik String elemeit egyenként hasonlítja a másik String összes karaktereihez.** Ha az **összehasonlítás logikai eredménye minden esetben igaz**, akkor a **két karakterlánc egymással azonos.**

Az algoritmus lépései hasonlítanak két egymásba ágyazott For ciklushoz.

<>	nem azonos
=	azonos
<	„névsor" szerint (ASCII kód) növekvő sorrendben
>	„névsor" szerint (ASCII kód) csökkenő sorrendben

Vannak olyan állítások, amelyek természetesek számunkra.

True	'ló'<>'ól'	'Bab'=#66'ab'	'kutya'= 'kutya'
False	'keze'<'ezek'	'kapa'='kap'	'Te'>'Ti'

A fordító **a String változók között** az előbbiekben említett bonyolult **algoritmusokat ő maga elvégzi**. Ehhez a nyelv is felhasználja, hogy a string karaktereihez sorszám rendeltető. Ezáltal nekünk is **lehetőségünk van a karakterlánc minden elemének egyedi kezelésére**.

🕮 **Foglaljuk össze a karakterláncokkal kapcsolatos ismereteinket:**

- **Stringek** hossza megállapítható a **Length függvény** segítségével:

  Length('kutya')=5; **Length**('Budapest')=8

- A deklarációnál lehetőségünk van **megadni a String változó hosszát. Ellenkező esetben a fordító 255 karaktert** feltételez és ennek megfelelően történik a memóriában a helyfoglalás. **A hosszúságot [ ] zárójelek között adjuk meg.**

  Var utca: String[20]; Var mondat: String;

  Az egyik esetben az utca **változó hossza 20 vagy annál kisebb**. A másik példában a mondat **változó hossza nem nagyobb mint 255.**

3.2. Hivatkozás a String elemeire

Hogyan hivatkozhatunk a BUDAPEST szónak pl. az ötödik karakterére? Legyen az előbb említett **string változó** azonosítója lakhely. Az **ötödik betűjét** (karakterét) a **lakhely[5] változó jelöli.** Ennek értéke most a 'P' betű.

Tehát a [ ] zárójelen belül megadhatjuk, hogy hányadik karakteréről van szó.

Tekintsük a következő táblázatot:

String	*hivatkozás egy elemre*	*Érték*
'DEBRECEN'	lakhely[4]	'R'
'Szeged'	lakhely[1]	'S'
'Kaposvár'	lakhely[3]	'p'
'Békéscsaba'	lakhely[5]	's'
'Hatvan'	lakhely[6]	'n'

A [ ] zárójelben megadott **értéket indexnek** nevezzük. **Az index nemcsak konstans szám lehet. Lehet változó** azonosító is. Ilyenkor a változó

értéke szabja meg, hogy a karakterlánc melyik sorszámú eleméről van szó.

3.3. Használatuk értékadó utasításokban

Példáinkban legyen szo egy változó, melynek kiinduló értéke: 'Makó'. A szo[k] a k-adik elemét fejezi ki a String-nek. Mivel a szo string változó volt, ezért a szo[k] karakter változó. Így szerepelhet az értékadó utasítás bal oldalán is. A táblázat minden sorában a szo String értéke az előző sor szerint értendő:

Az index értéke:	*értékadó utasítások*	*a String értéke*
-	szo:= 'Makó';	'Makó'
k:=3;	szo[k] := 'r';	'Maró'
k:=1;	szo[k] := 'K';	'Karó'
k:=3;	szo[k] := 't';	'Kató'
k:=4;	szo[k] := 'i';	'Kati'
k:=4;	szo[k] := szo[k-2];	'Kata'
k:=3;	szo[k] := 'p';	'Kapa'

A szo[k]:=szo[k-2] magyarázata: Az index értéke négy. Ezért az előző sor szerinti **'Kati'** stringnek a **4. betűjét ('i')** kicseréli **ugyanennek a szónak** a (4-2) **2. betűjére ('a').** A példán látható, hogy az **index lehet kifejezés** is. Természetesen az **index értéke** (Byte típus) **nem lehet több** a **szöveg** (String) **hosszánál.**

✍Találj ki hasonló szójátékot és foglald táblázatba!

3.4. For ciklus String adatokkal

Most következő feladatainkban az **Input adat** változója mindig **string típusú** lesz.

💻*Egy szövegben a szavakat szóköz (Space) választja el. Írjunk programot, amely a* ***szóközöket '*' karakterekre cseréli ki.***

A feladat főbb lépései:

- A szöveg karakterekre bontása a szöveg hossza alapján For ciklus segítségével.
- A string-ben lévő elemek hasonlítása a ' ' karakterhez.
- Találat esetén a karakterek cseréje.

```
Program csere;
Uses Crt;
    Var  szoveg: String;
    elemszam: Byte;
Begin
    Clrscr;
    WriteLn('Kérem a karakterláncot: ');
    ReadLn(szoveg);
    For elemszam:=1 To Length(szoveg) Do
        If szoveg[elemszam]=' ' Then  szoveg[elemszam]:='*';
        WriteLn(szoveg);
    ReadLn;
End.
```

*Egy szövegben a **szavakat szóköz (<Space>)** választja el. Írjunk programot, amely ennek alapján a szöveget **szavakra bontva kiírja** a képernyőre!*

```
Program szavak;
Uses Crt;
Var
    mondat: String;
    elemszam: Byte;
    bill: Char;
Begin
    Clrscr;
    WriteLn('Kérem a mondatot: ');
    ReadLn(mondat);
    For elemszam:=1 To Length(mondat) Do
        If mondat[elemszam]<>' ' Then
        Write(mondat[elemszam])
            Else WriteLn;
    bill:=ReadKey;
End.
```

Egészítsd ki tetszésed szerint további tájékoztatóval a felhasználó számára! Alakítsd át a programot, hogy a feltételes utasí-

tásban az **összehasonlítás az = jel legyen! Vigyázz a kiíratásra!**

Az probléma lényegére (szavakra bontás) még visszatérünk. Jelen esetben a mondat szavaival nem végeztünk semmilyen műveletet, ezért **nem is tároltuk a memóriában**. Az ilyen problémák megoldásához fogjuk megismerni **tetszőleges típusú tömb kezelését**.

Input egy 20 karakternél nem hosszabb karakterlánc. Írjuk ki a karaktereit soronként. Minden karakternek más legyen a színe.

A program fontosabb lépései:

- A háttérszín beállítása
- A string hosszának kiszámítása és tárolása egy változóban
  - ↘ Ciklus **egytől** a string **hosszáig**
    - A **szín** változó **kiszámítása** maradék képzéssel(0-15)
    - **Karakterszín** váltogatása
    - A **ciklusváltozó** szerinti **karakter** kiíratása és soremelés
  - Ciklus vége
- Várakoztatás <Enter> leütéséig

Egyeztesd a pascal nyelv szókészletével az előbbiekben leírt algoritmust! Futtasd le programodat 16 hosszúságú karakterlánccal, számold meg kiírt-e minden karaktert a program!

```
Program soronkent;
Uses Crt;
Var
   nev: String[20];
   elemszam, hossz, szin:Byte;
Begin
   TextBackGround(black);
   ClrScr;
   Write('Kérem a nevet: ');
   ReadLn(nev);
   hossz:=Length(nev);
   For elemszam := 1 To hossz Do
      Begin
         szin:=(elemszam mod 16);
         TextColor(szin);
```

```
        WriteLn(nev[elemszam]);
    End;
  ReadLn;
End.
```

Biztosan észrevetted, hogy **16. karaktert nem láttad a képernyőn**. Ennek az oka, hogy a **Mod művelet értéke nulla** volt, amely a **fekete színnek** a számértéke. **Hogyan lehetne ezen változtatni?** Sokféleképpen. Az egyik lehetséges megoldás az **Ord függvény használata**.

Számoljuk meg egy string változóban lévő kicsi 'ly' karakterek számát!

Néhány dolgot előzetesen a programhoz:

A **'ly'** karakterek számát **szamol:=szamol+1** értékadó utasítással határozzuk meg. A furcsa a számodra az lehet, hogy az utasítás **mindkét oldalán ugyanaz a változó áll**. Ha visszaemlékszel a **kifejezések kiértékelésére és az értékadó utasítások működésére**, akkor érthetővé válik:

- A program elején a **szamol:=0** miatt a **változó értéke nulla**.
- A For ciklusban a ciklusmag minden egyes végrehajtáskor az értékadó utasítás **jobb oldalán álló kifejezés kiértékelődik** (a szamol változó értéke nő eggyel). Ezután az **érték átadódik a bal oldalon** (szamol) **lévő változónak.** Tehát ez az értékadó utasítás **ugyanannak a memóriacímnek az értékét növeli 1-el.** Hasonló értékadó utasítással még fogunk találkozni.
- Mivel a **szamol** változó a **program munkaváltozója**, ezért értékét a program elején **kezdőértékre kell beállítani.** Ez a mi esetünkben nulla (szamol:=0;). Egyes Pascal nyelvek (pl.: Turbo Pascal 6.0) indításkor a program változóinak értékét nem törlik a memóriában. **Ilyenkor** minden futtatáskor **bizonytalan a változó kezdőértéke,** amely programunk **eredményét rossz irányba befolyásolná.**

A kezdőértéket beállíthatjuk másképpen is. A jelen esetben deklarálhatnánk a **szamol változót úgynevezett tipizált konstansként is.** Ez azt jelenti hogy a deklarációban **kezdőértéket adunk neki.** De a konstansokkal kapcsolatban azt mondtuk, hogy **konstans azonosító nem állhat értékadó utasítás bal oldalán**!

A tipizált konstans egy kezdőértékkel rendelkező változó! Így szerepelhet az értékadó utasítások baloldalán is.

A deklaráció:

Const szamol : **Byte**=0;

Általában: **Const azonosító : típus=kezdőérték;**

A Turbo Pascal 7.0 a programban használt változó értékeket minden végrehajtás kezdetén automatikusan nulla kezdőértékre állítja.. Ettől függetlenül általános szabály programozásban, hogy a feladat eredményeinek a meghatározásához használt változókat kezdőértékre kell állítani. A mi esetünkben pl.: a szoveg változót ez nem érinti, hiszen a ReadLn utasítással mindig új értéket kap.

```
Program Ly_szamol;
Uses Crt;
Var
   szoveg: String;
   k,szamol: Byte;
Begin
   szamol:=0;
   Clrscr;
   Write('A vizsgált karakter sorozat: ');
   ReadLn(szoveg);
   For k:=1 To Length(szoveg)-1 Do
      If (szoveg[k]='l') And (szoveg[k+1]='y')
      Then szamol:=szamol+1;
   WriteLn('A ly betűk száma: ',szamol);
   ReadLn;
End.
```

A programot tanulmányozva válaszolj a következő kérdésekre:

- A ciklusváltozó végértéke miért kisebb eggyel mint a string hossza?
- Milyen adatot jelölnek a szoveg[k] és szoveg[k+1] változók?
- Milyen művelet az And?
- Milyen esetekben igaz vagy hamis a (szoveg[k]='l') And (szoveg[k+1]='y') logikai kifejezés értéke?
- Milyen utasítás alkotja a For ciklus magját? Hogyan működik ez?

A következő feladat az előző logikáján alapul. A feladatot ezért nem is magyarázzuk. Feltehetsz saját magad is a feladattal kapcsolatos kérdéseket, melyeket közösen megbeszéltek.

Gyakran eltévesztjük a számítógép billentyűzetén a 'k' és 'l' karaktereket, hiszen egymás mellett vannak. Ebből következően a 'ly' helyett 'ky' stringet gépelünk. Ezért a feladat:

Írjunk programot, amely egy tetszőleges input karaktersorozatban a 'ky' stringet 'ly' stringre cseréli.

```
Program ky_ly_csere;
Uses Crt;
Var
     bill: Char
     szoveg: String;
     elem: Byte;
Begin
     Clrscr;
     Write('Mit javítsak ki: ');
     ReadLn(szoveg);
     For elem:=1 To Length(szoveg) Do
          If (szoveg[elem]='k') And (szoveg[elem+1]='y')
          Then szoveg[elem]:='l';
     WriteLn('A javitott szöveg: ',szoveg);
     bill:=ReadKey;
End.
```

3.5. A csökkenő számláló ciklus

Mint a neve is mutatja, a **ciklusváltozó** nem +1-el hanem **-1-el változik**. Ilyenkor a fordító a ciklust **csak akkor hajtja végre**, ha a **kezdőérték nem nagyobb** (kisebb vagy egyenlő) **mint a végérték**! **Szintaktikája** (formája) **hasonló** mint az eddig növekvő For ciklusé. Csupán **egyetlen alapszót kell kicserélni**. Nevezetesen a **To helyett** a **DownTo** alapszót használjuk. Down magyar jelentése: lefelé

```
For változó:=kifejezés1 DownTo kifejezés2 Do
     Begin
          utasítás1;
          utasítás3;
          .....
     End;
```

*Ennek bemutatására legyen a példa a következő: Egy Stringnek a karaktereit az eredetihez képest **írassuk ki fordított sorrendben.** A string hosszúsága 20-nál nem nagyobbra legyen deklarálva.*

pl.: káró **órák**
pala **alap**

```
Program vissza;
Uses Crt;
Var
     nev:String[20];
      k,hossz:Byte;
Begin
     ClrScr;
     Write('Hogy hívnak? ');
     ReadLn(nev);
     hossz:=Length(nev);
     For k :=hossz DownTo 1 Do
        Write(nev[k]);
     ReadIn;
End.
```

Vedd észre, hogy a megfordított **stringet nem tároltuk külön egy másik változóban!**

A string és karakter típusú témakörünket távolról sem merítettük ki az eddig közölt ismeretekkel. A példák alapján ennek a fejezetnek adhattuk volna a Játék a betűkkel címet is. Azért a háttérben komolyabb alkalmazási lehetőségek rejlenek. Ehhez persze még több ismeretre lenne szükség (pl.: szöveges állományok kezelése).

*Témakörünket fejezzük be szintén egy **egyszerű játékos programmal**, melyet további bővítésre, szépítésre és finomításra ajánlunk! A programot magyarázat nélkül közöljük!*

- **Mozgassunk** a képernyő 12. sorában egy maximum húsz karakterből álló stringet a következőképpen.
  - ↘ Az input értékadás után töröljük le képernyőt.
  - ↘ Írassuk ki a 12. sorban jobbra igazítva.

- **Mozgassuk karakterenként** (az elsővel kezdve) ugyanebben a sorban jobbról balra.
- **Befejezés** ha a string képernyőn **jobbra lett igazítva**.

```
Program mozog;
Uses crt;
Var
     bill:Char;
     hossz,k,m:Byte;
     szoveg:String[20];
Begin
     TextBackGround(black);
     TextColor(White);
     Clrscr;
     Write('Kérem a szöveget: ');
     ReadLn(szoveg);
     ClrScr;
     Hossz:=Length(szoveg);
     GotoXY(1,12);
     Write(szoveg:80);
          For m:=1 To hossz Do
               For k:=79-hossz+m DownTo m do
                    Begin
                         GotoXY(k,12);
                         Write(szoveg[m],' ');
                         Delay(100);
                    End;
     bill:=ReadKey;
End.
```

Annyit hozzáfűzünk, hogy a mozgatás nem állandó képernyő törléssel van megoldva. A képernyő törlés ismétlésével **állandó villogást látnánk**!

A **Delay** utasítás a programot 100 milliszekundum ideig késlelteti. Ezt a számítógépek **gyorsaságától függően érdemes változtatni**.

A programban **For ciklusok egymásba ágyazása** is szerepel. Ezekkel a következő fejezetben foglalkozunk.

For ciklusok alkalmazása

1. Műveletek eredményeinek gyűjtése

Ebben a fejezetben bemutatjuk a For ciklus további alkalmazásait. Megmutatjuk két adattípus (numerikus és string) kölcsönös átalakításának lehetőségét is. Az adatok átalakításának alkalmazása sokszor elkerülhetetlen, vagy nélküle a feladat megoldása sokkal nehezebb. Foglalkoznunk kell még néhány alapvető számítástechnikai algoritmussal is. Ezek alkalmazásában az ismertetett ciklusokat összetettebb formában kell majd alkalmazni és szükség lesz további új fogalomra is (Tömb).

Az előző fejezetekben találkoztunk olyan értékadó utasításokkal, ahol mindkét oldalon ugyanaz a változó azonosító szerepelt (pl: db:=db+1;).

Mit jelentenek az alábbi értékadó utasítások?

hatvany:=hatvany*alap; **kuka:**=kuka+szemet; **szam:**=szam+7;

A három eset a következőképpen értelmezhető: A jobb oldalon álló változó a régi értéket, a bal oldali pedig a művelet elvégzése utáni új értéket jelenti.

induló értékek

hatvany:=1; alap:=4	kuka:='zöld'; szemet:='bab';	szam:=2;

ciklus háromszor

hatvany:=hatvany*alap;	**kuka:**=kuka+szemet;	**szam:**=szam+7;

a ciklusban felvett értéke lépésenként

hatvany=4	**kuka='zöldbab'**	**szam=9**
hatvany=16	**kuka='zöldbabbab'**	**szam=16**
hatvany=64	**kuka='zöldbabbabbab'**	**szam=23**

A példákon tehát érzékelhető, hogy mindhárom esetben „gyűjtöttük" a **műveletek eredményét a memóriában** a **hatvany, kuka, szam változók** segítségével.

Input adatok: egy számtani sorozat első eleme (elso), az utána következő tagok száma (db) és a szomszédos tagok különbsége (novel)! Határozzuk meg a sorozat többi elemét!

pl.: Ha az Input adatok (–3, 4, 5), akkor az Output eredmény (1, 5, 9, 13, 17).

- Ha a bemenő adatok (–8, 6, 7), akkor mi lesz a program eredménye?
- Az előző táblázat alapján fogalmazd a programban végrehajtandó ciklus lépéseket?

Az feladat algoritmusa az ismételt összeadások elvégzésén alapszik. Az összegzést a **tagok változó** segítségével végezzük a

tagok:=tagok+**novel** értékadó utasítással.

```
Program sorozat;
Uses Crt;
Var
   k,db:Byte;
   elso,novel,tagok:Integer;
Begin
     ClrScr;
     Write('Kérem a sorozat eksô tagját: ');
     ReadLn(elso);
     Write('Adja meg a szomszédos tagok közötti különbséget: ');
     ReadLn(novel);
     Write('Hány tagját kéri a sorozatnak? ');
     ReadLn(db);
     ClrScr;
     tagok:=elso;
     For k := 1 To db Do
          Begin
               Write(tagok:20);
               tagok:=tagok+novel;
          End;
     ReadIn;
End.
```

Válaszolj a következő kérdésekre:

- Melyik változóban történik az összegzés?
- Miért nem lehet a db változót valós típusúra deklarálni?
- Milyen input adatok esetén nem működik helyesen a program? Próbáld ki? Mi lehet az oka?
- Alakítsd át a programot úgy, hogy a sorozat tagjai tizedes törtek is lehessenek!

Készítsünk programot amely elvégzi a hatványozás műveletét, ha az alap tetszőleges egész szám, a kitevő tetszőleges nem negatív egész szám

A hatványozás bizonyos feltételek esetén (ha a kitevő nem negatív egész szám) ismételt szorzásként értelmezhető. A feladatunk ezeknek a feltételeknek megfelel: pl.: $-5^3 = (-5)*(-5)*(-5) = -125$

A programban felhasználjuk az előző táblázat **hatvany:**=hatvany*alap; értékadó utasítását.

```
Program hatvanyozas;
Uses Crt;
Var
   alap:LongInt;
   k, kitevo:Byte;
   ch : Char
Const hatvany: LongInt=1;
Begin
   ClrScr;
   WriteLn('Hatványozás ismételt szorzással.');
   Write('Alap: ');
   ReadLn(alap);
   Write('Kitevő: ');
   ReadLn(kitevo);
   For k:=1 To kitevo Do
     hatvany:=hatvany*alap;
   Write('A hatványozás eredménye:',hatvany);
   ch:=ReadKey;
End.
```

☝ Megjegyzés: **A hatványozást egyetlen értékadó utasítással is el lehet végezni két függvény segíttségével(Exp és Ln).** Ha gimnáziumba jársz, biztos tanultad, hogy $a^x = e^{x * Ln(a)}$ .Az előző program változóit használva: hatvany: = Exp(kitevo *Ln(alap)). Ennek megértéséhez matematikai ismeretekre van szükség.

2. Numerikus és string adatok konvertálása

2.1. Az Str eljárás

Az Str utasítás lehetőséget ad egész vagy valós típusú adatok String adatokra való átalakítására. Ennek nagyon sok haszna van pl. az input adatok ellenőrzésénél. Erre a lehetőségre még visszatérünk. Segítségével bizonyos típusú feladatok egyszerűbben oldhatók meg. Általános alakja: **STR(szam**, szamstring); ahol a szam egy numerikus (egész vagy valós) típusú változó vagy konstans. A szamstring csak string változó lehet. Nézünk néhány példát numerikus konstansokkal:

az eljárás használata	*az adat* string változó értéke
Str(1.32, adat)	adat=' 1.3200000000E+00'
Str(1.9856:8:3, adat)	adat=' 1.986'
Str(-432, adat)	adat='-432'
Str(39, adat)	adat='39'
Str(+78, adat)	adat='78'

A string változóba mindig jobbra igazítva kerülnek az adatok. Egész típus esetén a pozitív előjel nem kerül bele a karakterláncba. A mezőszélességgel valós értékek esetén a tizedes helyiértékek számát a Write utasításhoz hasonlóan meg lehet adni.

Használatára ügyelj, mivel két paramétert tartalmaz. Az első, amelyet át kell alakítani. A másodikban pedig az átalakítás eredményét tároljuk. Az eljárásokat értékadó utasításban vagy kifejezésekben zárt formája miatt nem használhatjuk.

2.2. A Val eljárás

Általános alakja: **Val(szamstring, szam, kod);** ahol a szamstring egy string típusú változó vagy konstans. A szam csak numerikus változó lehet, működése az előzőnek az ellentettje. Tehát stringet alakít numerikus értékké. Az átváltásnál viszont lehet probléma. Minden string adat nem

alakítható át. Pl.: ('1.3w'). A kod változó (csak Integer típusú lehet) az átalakítás sikerét ellenőrzi. Ha átalakítható volt, akkor **értéke nulla lesz**. Ellenkező esetben a string azon karakterének sorszáma, ahol az átváltás sikertelen volt. Meglátjuk, hogy ez jól felhasználható input adatok ellenőrzésére.

Néhány példa:

az eljárás használata	*a szám* **változó értéke**	*a kód változó értéke*
Val('1.32',szam, kod)	szam=1.32	kod=0
Val('1.32s',szam, kod)	szam=?	kod=4 (sikertelen)
Val('-432',szam, kod)	szam=-432	kod=0
Val('+47',szam, kod)	szam=47	kod=0
Val('16',szam, kod)	szam='39'	kod=0
Val('ü23',szam, kod)	szam=?	kod=1 (sikertelen)

Olvassunk be egy tetszőleges egész számot egy Integer változóba. A programnak meg kell határoznia a számjegyek összegét.

A program főbb lépései:

A változót string adattá alakítjuk
For ciklus a string hosszáig
For ciklussal karakterekre bontjuk
A kapott számjegy karaktereket számmá alakítjuk
Összegezzük a számjegyeket, amelyek már egész típusúak

```
Program szamjegyek_osszege
Uses Crt;
Var
   osszeg,szam,kod:Integer;
   szamjegy,k:Byte;
   szamstr:String;
Begin
   ClrScr;
   osszeg:=0;
   Write('Kerem a szamot: ');
   ReadLn(szam);
   Str(szam,szamstr);
   For k:=1 To Length(szamstr) Do
      Begin
```

```
        Val(szamstr[k],szamjegy,kod);
        osszeg:=osszeg+szamjegy;
    End;
  WriteLn('A szamjegyek osszeg: ',osszeg);
  ReadLn;
End.
```

Negatív szám esetén is működik, pedig a programnak van egy szépséghibája. Ugyanis a '-' karaktert nem tudja számmá alakítani. Ez nem szakítja meg a program futását és nem „ rontja" el az osszeg változó értékét. Javítani lehetne, ha a ciklus kezdő értéke: k:=1+ **Ord(szam<0) lenne.**

✍ Módosítsd úgy, hogy külön számolja ki a páros és páratlan sorszámú számjegyek összegét.

2.3. For ciklusok egymásba ágyazása

Egy for ciklus tartalmazhat egy másik for ciklust is. Hogyan működnek ezek együttesen? A külső ciklus vezérli a belső működését. Ha pl.: a külső 5-ször hajt végre valamilyen tevékenységet, a belső pedig 3-szor, akkor a belső tevékenységét ötször ismétli, vagyis 15-ször fogja végrehajtani.

💻 *Írassuk ki az a, b, c és d, e, f, g, h karaktereket a következő formában:*

képernyő

a defgh
b defgh
c defgh

```
Program ciklusban_a_ciklus
Uses Crt;
   Var k,m:Char;
Begin
   ClrScr;
```

```
    For m:='a' To 'c' Do                    külső ciklus kezdete
      Begin
        Write(m);
          For k:='d' To 'h' Do              belső ciklus
          Write(k);                         belső ciklus vége
        WriteLn;
      End;                                  külső ciklus vége
    ReadLn;
End.
```

A program megértéséhez, próbálj válaszolni a következő kérdésekre:

- A **write(m) és writeLn** utasítások melyik ciklushoz tartoznak?
- Mit ír ki a program ha **write(m) utasítást kicseréljük writeLn(m)** utasításra.?
- Mi lesz az eredmény, ha **writeLn utasítás helyett write** utasításra?
- Hogyan működik **writeLn(k)** utasítás esetén?

Írjunk programot, amely megkeresi azt a háromjegyű pozitív egész számot, melyben a számjegyek szorzata 24!

```
Program haromjegyu;
Uses Crt;
Var
    szamjegy, k:Byte
    szorzat,szam: Longint;
    szamstr: String;
    kod: integer;
Begin
    ClrScr;
      For szam:=100 To 999 Do
      Begin
        Str(szam,szamstr);
        szorzat:=1;
        For k:=1 To Length(szamstr) Do
            begin
                Val(szamstr[k],szamjegy,kod);
                szorzat:=szorzat*szamjegy;
            End;
        If szorzat=24 Then Write(szam,' ');
```

```
    End;
End.
```

✍ Alakítsd át a programot úgy, hogy azokat a számokat szűrje ki, amelyekre a feltétel: a számjegyek szorzata 12-nél nagyobb, de 20-nál kisebb!

3. Véletlen számok: a Random függvény

A Random függvény a gépi megszakítás segítségével egy számot állít elő. Az előállítandó szám határait a bemenő paraméterrel a programban változtathatjuk. Ez lehet konstans is vagy kifejezés. Típusaik az egészek közül a nem negatívak (Byte,Word) lehetnek. **Példák:**

értékadás	*érték*
Random(25)	0 <=érték <25
Random(25)+1	0 < érték <=25
Random(90)	0 < érték <90
Random	0 <= érték <1 (valós)
Random(101)	0 <= érték <101
Random(101)-50	-50 <= érték <=50

Ha nem használunk paramétert, akkor értéke nullánál nem kisebb és egynél kisebb valós adat. A **véletlen számok előállításához**, még szükség van a **Randomize** eljáráshoz.(a program elején egyszer kell használni)

Sorsoltassunk a géppel 0-nál nagyobb és 90-nél nem nagyobb egész számokat! Írassuk ki az eredményt a képernyőre!

```
Program veletlen1;
Uses Crt;
Var lotto,j:Byte;
Begin
   Randomize;
   ClrScr;
   For j:=1 To 5 Do
      Begin
         lotto:=Random(90)+1;
         Write(lotto:4)
```

```
    End;
  Readln;
End.
```

Előfordulhat, hogy a program egy számot többször is kisorsol. Ennek a problémának a megoldására még visszatérünk. Ezzel kapcsolatban nagyon jól használhatók a halmaztípusú adatok.

✍ A következő feladat megoldása előtt próbáljunk választ adni a következőkre:

- Melyik utasítással lehet állítani a képernyőszíneket?
- Hány féle színbeállítás lehetséges?
- Hogyan és melyik műveletet kell használni, hogy egy tetszőleges számot 15-tel osztva a lehetséges maradékokat kapjuk?
- Melyik az függvény, amely „ vakon" beolvas egy karaktert?
- Hogyan lehet a kurzort a képernyő adott helyére irányítani?
- A karakteres képernyőnek hány darab oszlopa és sora van?
- Mennyi karaktert lehet összesen kiíratni a képernyőre?
- A képernyő melyik sorára kell vigyázni kiíratásnál és miért?

💻 *Olvassunk be a billentyűzetről egy karaktert, és írjuk ki a képernyőre véletlenszerűen 60-szor! A képernyő fekete, a karakterek fehér színűek legyenek. A karakterek lehetséges háttérszínét is változtassuk a oszlopszámhoz viszonyítva, de vigyázzunk arra, hogy háttérszíne ne egyezzen meg az írás színével.*

```
Program veletlen2;
Uses Crt;
Var
   kar: Char;
   szin, oszlop, sor, j: Byte;
Begin
   Randomize;
   TextBackGround(0); TextColor(15);
   ClrScr;
   kar:=ReadKey;
```

```
    For j:=1 To 60 Do
      Begin
        oszlop:=Random(80)+1;
        sor:=Random(25);
        GotoXY(oszlop,sor);
        szin:=(oszlop div 15);
        TextBackGround(szin);
        Write(kar);
      End;
    ReadIn;
End.
```

4. Tömb típus

A For ciklus segítségével elő tudtunk állítani nagyobb mennyiségű Output adatot is. De a feldolgozás után az értéküket nem tudtuk tárolni esetleges további feldolgozás céljából. Lehetséges olyan feladat melyben az input adataink több elemet is tartalmazhatnak és az adathalmaz minden egyedével ugyanazokat a műveleteket akarjuk elvégezni. Az input folyamatok elvégzésének egyik lehetősége a For ciklus. De hogyan tároljuk, szervezzük meg párhuzamos feldolgozását pl. 100 adatnak. Deklaráljunk száz változót? Nem.

🕮 A Tömb: A tömb tehát **logikailag összetartozó (azonos típusú egyedek)** adatcsoport. A matematikai értelembe vett általános **halmazfogalomtól az különbözteti meg, hogy minden elemére egy sorszámmal tudunk hivatkozni.**

Az osztályban például a naplóban lévő sorszám alapján valakinek a neve egyértelműen megállapítható. Hogyan? Legyen az azonosító pl.: **naplo**

Eddig csak egyetlen változó azonosítója. Ha megmondjuk, hogy hányadik eleméről van szó akkor már egyértelműbb. Pl.: **naplo[31].** De lehet, hogy nincs is ennyi tanuló. A másik probléma az adat tulajdonsága. A **naplo[31]** alapján gondolhatnánk a 31**. tanuló tanulmányi átlagára** is.

🕮 **Tehát tömb esetében meg kell adni a tömb azonosítóját, az elemek számát és típusát.**

példa deklarációra:

```
Var
```

naplo:Array[1..31] OF String

Of (az angolban birtokos szerkezet), **array**(tömb). Lefordítva magyarra a **naplo azonosítójú tömbnek maximálisan 31 eleme** lehetséges és **minden elem String típusú.**

4.1. Tömbök deklarálása és indexelése

```
Var
azonosító:Array[ alsó..felső ] Of egyedtípus;
```

Az **elemek számát résztartománnyal adjuk meg. A résztartomány alsó és felső határait indexeknek nevezzük. Az index határok típusai egész, karakter és logikai konstans**. Az első és utolsó elem sorszámát a deklarációban két pont választja el. A tömb elemeinek a típusa az eddig megismert típusok bármelyike lehet.

☝ Az Of és Array a fenntartott szavakhoz tartoznak, tehát azonosítóként nem lehet használni!

Példák:

```
var
    egeszek  : array[1..10]   Of integer ;
    mondat   : array[1..20]   Of String ;
    nevek    : array[1..20]   Of String[12] ;
    korok    : array[1..30]   Of Byte ;
    jelek    : array[1..40]   Of Char ;
    tizedes  : array[1..50]   Of Real ;
```

A fenti példák közül a nevek tömbnek **20 db eleme lehet és minden eleme karakterlánc**. A lehet szó nagyon fontos mert a programban nem biztos, hogy a tömb minden eleme értéket kapott.

✍ **Fogalmazd meg az** előző deklarációk jelentéseit (milyen típusú elemekből áll és mennyi az elemek maximális száma)!

Nézzünk egy példát a fentiek közül néhány névvel:

index	A tömb elemei	hivatkozás egy elemre	értéke
1	'Kati'	nevek[3];	'Kati'
2	'Jóska'	nevek[5];	'Péter'

3	'Pista'	nevek[2];	'Jóska'
4	'Ákos'	nevek[5][2;]	'é'
5	'Péter'	nevek[1][1];	'K'

Az utolsó két sorban egy furcsa hivatkozást találsz. Ha eszedbe jut, hogy a **karakterlánc elemeit hogyan lehet indexelni,** akkor nem is olyan érthetetlen. pl.: a **nevek[5][2]**; a nevek tömb ötödik elemének (**'Péter'**) a második karaktere az **'é'**.

indexek	A tömb elemei	hivatkozás egy elemre	értéke
1	'Eger'	adat[5] +adat[1];	'lakhely:Eger'
2	'1985-04-03'	adat[3][2]:='7'	'1785-04-03'
3	'-231'	adat[4][1]:=''	'óra'
4	'Móra'	adat[3][1]:='+'	'+231'
5	'lakhely:'	adat[4][3]:=''	'Móka'

A tömb elemek között természetesen műveletek is végezhetőek és **elemeire változókkal, vagy kifejezésekkel is lehet hivatkozni. Az elemek sorszámát index számnak is szokás nevezni.**

A tömböket vektorként, mátrixként is szokták emlegetni a matematikai vonatkozásai miatt. Mi továbbra is maradunk a tömb elnevezésnél.

4.2. Tömbök beolvasása

Input adat legyen a beolvasandó adatok száma. Írjunk programot amely a megadott számú elemet tárolja egy Integer típusú tömbbe!

A program megoldására az eddig tanult **For ciklus kínálkozik**. A **ciklusváltozó** kezdőértékét **egynek válasszuk**, a végérték pedig az **input adat** (elemszam azonosító). A ciklusban a beolvasást a ReadLn utasítás végzi, mely hatására a **betomb** (azonosítójú) tömb változóban tárolódik. Miközben a ciklusváltozó mindig eggyel nő, a tömb elemeire az **index változóval hivatkozunk.**

```
Program tomb_be;
Uses Crt
Var
```

```
    betomb: Array[1..100] Of Integer;
    bill:Char;
    elemszam, index:Byte;
Begin
    ClrScr;
    Write('Hany db. elem lesz? ');
    ReadLn(elemszam);
    For index:=1 To elemszam Do
        Begin
          Write(index,'.','eleme: ');
          ReadLn(betomb[index]);
        End;
    bill:=ReadKey;
End.
```

Már volt róla szó, hogy a programozási feladatokban a feldolgozáshoz szükséges változókat kezdő értékre kell beállítani. Ezt tömbök esetén is megtehetjük egy For ciklussal melynek végértéke a tömb felső index határa.

Input adat String adatok száma. Írjunk programot amely a megadott számú elemet tárolja egy tömbbe! Írassuk ki a tömb elemeit a beolvasással fordított sorrendbe!

Alapesetben a kiíratás logikája ugyanaz mint a beolvasásé.

```
Program be_ki_forditva
Uses Crt;
Var
    betomb: Array[1..100] Of String;
    bill:Char;
    darab,index:Byte;
    Const kiir='A beolvasott elemek forditott sorrendbe';
Begin
    ClrScr;
    Write('Hany db. elem lesz? ');
    ReadLn(darab);
    For index:=1 To darab Do
        Begin
            Write(index,'.','eleme: ');
            ReadLn(betomb[index]);
```

```
    End;
  WriteLn(kiir);
  For index:=darab DownTo 1 Do
    Write(betomb[index]:10);
  bill:=ReadKey;
End.
```

✍ Módosítsd a programot, hogy a szavakat is fordított sorrendbe írja ki.

4.3. Példák tömb adatok kezelésére

💻 *Egy input Stringben a szavak végét egy szóköz karakter jelöli. Bontsuk a stringet szavakra és tároljuk egy tömbben. A program feltételezi, hogy a string elején és végén nincsenek szóköz karakterek!*

A program megoldásának főbb lépései:

- Deklarációk
- A tömb indexének beállítása kezdőértékre
- A tömb elemeinek feltöltése üres karakterekkel
- Beolvasás
- Feldolgozás

Ciklus a **szöveg hossza szerint**
  Ha az adott számú karakter **nem szóköz, akkor**
  a karakter **hozzáfűzése** az **m indexű tömb elemhez**,
    különben az **index** szám(m) **növelése 1-el**.
Ciklus **vége**
A tömb elemeinek kiíratása For ciklussal

```
Program szavakra;
Uses Crt;
Var
  gyujtszoveg: String,
  m,k: Byte;
  szavak: Array[1..20] Of String;
Begin
  Clrscr; m:=1;
  For k:=1 To 100 Do szavak[k]:='';
```

```
    ReadLn(szoveg);
    For k:=1 To Length(szoveg) Do
        If szoveg[k]<>' '
        Then szavak[m]:=szavak[m]+szoveg[k]
            Else m:=m+1;
    For k:=1 To m Do WriteLn(szavak[k]);
End.
```

✍ Alakítsd át a programot:

- A szavakat karakterenként és a szöveget szavanként fordított sorrendbe írja ki.
- Írd át a feltételes utasításban a <> operátort = összehasonlításra és javítsd ki a programot ennek megfelelően!
- Melyik változó a legfontosabb a tömb szempontjából?
- Mi a szerepe a **szavak[m]:=szavak[m]+szoveg[k]** értékadó utasításnak?
- Deklaráld az **m** változót kezdőértékkel megadott változóként!

☝ Bemutatunk még egy deklarációs lehetőséget, amely bizonyos feladattípusoknál jól alkalmazható. Említettük, hogy az **index lehet karakter tartomány** is. Mint említettük ezek sorszámozott típusú adatok, mivel mindegyikhez egyértelműen tartozik egy ASCII kód. Ezért pl.:

🕮 **Var**

nagy:Array['A'..'Z'] Of egyedtípus

kicsi: Array['a'..'z'] Of egyedtípus

is helyes deklaráció. Természetesen a hivatkozás ilyenkor csak karakter típusú adatokkal lehetséges.

Pl.: legyen kar karaktertípusú változó:

- nagy[kar] A tömb elemet a kar változó határozza meg
- nagy['Z'] A tömb utolsó eleme
- nagy['C'] A tömb harmadik eleme
- kicsi['b'] A tömb második eleme
- kicsi['a'] A tömb első eleme
- kicsi[kar] A tömb elemet a kar változó határozza meg

Ezzel a lehetőséggel élve stringeket az első betűje szerint könnyen csoportosíthatunk. Ezzel kapcsolatos következő példánk:

Olvassunk be egy tömbbe neveket! A tömb elemeinek száma szintén input adat. Az angol abc figyelembe vételével számoljuk meg, hogy hány név kezdődik A, B, C.....Z betűkkel. A program ne tegyen különbséget kicsi és nagy betűk között. Csak a nullától különböző eredményeket kell kiíratni a hozzátartozó karakterekkel együtt!

Mielőtt elkezdenénk programunkat, meg kell ismernünk egy új függvényt. Ez a függvény a kisbetűket alakítja nagybetűkké.

Használata: Upcase(kar) ahol a kar karakter típusú változó vagy konstans. Ez csak a 'a'..'z' tartományra vonatkozik. Más karakterek értékén nem változtat. Értéke szintén karakter.

pl.: Upcase('a')= 'A'; Upcase('#98')='B' Upcase('w')= 'W';

```
Program elso_betu;
Uses Crt;
Var
  ch : Char;
  k,db : Byte;
  kezdo : Array['A'..'Z'] Of Byte;
  nevek : Array[1..100] Of String;
Begin
   For k:=1 To 100 Do nevek[k]:='';
   For ch := 'A' To 'Z' Do kezdo[ch]:=0;
   ClrScr;
   Write('Hány Db. név lesz? ');
   ReadLn(Db);
   WriteLn('Kérem neveket! : ');
   For k:=1 To db Do
      Begin
         ReadLn(nevek[k]);
         ch := Upcase(nevek[k][1]);
         kezdo[ch]) := kezdo[ch]+1;
      End;
   For ch := 'A' To 'Z' Do
      If kezdo[ch]<>0 Then
```

```
    WriteLn(Upcase(ch),' betűvel kezdődőek: ',kezdo[ch]);
  ch:=ReadKey;
End.
```

☝ A magyar abc ékezetes karaktereire példánk programja csak módosítva működik. Ennek oka, hogy a kódtáblában nem folytonosan helyezkednek el. A módosítás a következő:

- Indíts el egy szoftvert, amely átírja magyarosítja a kódtáblát. pl.: **multikey**
- Az 'A'..'Z tartományokat javítsd ki #1..#255 tartományra

🕮 A program rövid ismertetése:

- Deklarációk
- Tömbök elemeinek beállítása kezdőértékre
- A string tömb felső indexének beolvasása
- Számláló ciklus a tömb elemeinek száma szerint
  - ↘ A tömb elemeinek a beolvasása
  - ↘ Értékadó utasítás, amely meghatározza és nagybetűsíti a tömb elemeinek kezdőbetűjét (**ch := Upcase**(nevek[k][1]);)
  - ↘ A betűket számláló tömb növelése eggyel(kezdo[ch]) := kezdo[ch]+1;)
  - ↘
- Számláló ciklus az eredmény kiíratására

✍ Ha a betűk számát meghatározó tömbünket nem a fentiek szerint deklaráljuk, akkor a megoldás csak két egymásba ágyazott For ciklussal lehetséges. Próbáld megoldani így is a feladatot!

4.4. A minimum kiválasztás módszere

A minimum kiválasztás módszere

Adott egy rendezetlen adathalmaz. Ki kell választanunk a legkisebb elemét. Ha az elemek egy tömb elemei, akkor könnyebb dolgunk van, mivel az adathalmaz minden eleméhez hozzá van rendelve az indexszám.

Tároljuk egy min változóban a tömb első elemét. Összehasonlítjuk, hogy ez nagyobb-e, mint a második elem. Ha **nagyobb,** akkor ez lesz **min**

változó új értéke. Megint összehasonlítás, de a harmadik elemmel és a feltétel szerint értékváltoztatás ha az logikai kiértékelés True.

A táblázat érzékelteti a tömb elemeivel való összehasonlítás lépéseit. Szemlélteti, hogy a min változónak mikor változott meg az értéke. Ebben az algoritmusban a tömb elemeinek az értékeit változatlanul hagytuk.

a ciklus lépései	minimum	kezdő érték					
	14	hasonlítások a tömb elemeihez					
1. csere	**12**	14	**12**	16	8	2	
2. nincs csere	12	14	12	16	8	2	
3. csere	**8**	14	12	16	**8**	2	
4. csere	**2**	14	12	16	8	**2**	

Az összehasonlításokat egy for ciklussal lehet elvégezni. Mivel a ciklus előtt a min változó a tömb első eleme lesz, ezért az összehasonlítást a tömbindex második elemétől kell kezdeni. A maximum keresés ugyanígy történik csak a feltétel az új értékadáshoz max<tömb[k].

Mondatszerűen leírva az algoritmust:

min=tomb(1)
max=tomb(1)
ciklus 2-től indexig
    Ha min>tomb(index) **akkor** min:= tomb(index)
    Ha max<tomb(index) **akkor** max:= tomb(index)
ciklus vége

Olvassunk be egy tömb változóba 50 db egynél nagyobb és ezernél nem kisebb véletlen számot és írassuk ki a képernyőre! Keressük meg a legkisebb és legnagyobb elemét!

```
Program min_max;
Uses Crt;
Var
   min,max:Word;
   veletlen: Array[1..50] Of Word;
   bill:Char;
   index:Byte;
```

```
   Const kiir='Veletlen szamok(1-1000): ';
Begin
   ClrScr;
   WriteLn(kiir);
   Randomize;
   For index:=1 To 50 Do
      Begin
         veletlen[index]:=Random(1000)+1;
         Write(veletlen[index]:8);
      End;
   min:=veletlen[1];
   max:=veletlen[1];
   For index:=2 To 50 Do
      Begin
         If min>veletlen[index] Then min:=veletlen[index];
         If max< veletlen[index] Then max:=veletlen[index];
      End;
   WriteLn('Legkisebb: ',min);
   WriteLn('Lenagyobb: ',max);
   bill:=ReadKey;
End.
```

4.5. Rendezés minimum kiválasztással

Az előbbi módszeren alapul az adatoknak növekvő vagy csökkenő sorrendbe való rendezése. Az egyik több lépésben való **minimum keresés** a másik pedig **maximum keresés**. Ha az előző algoritmust tovább folytatnánk akkor minden lépésben tárolhatnánk az eredményeket egy rendmin és rendmax tömbben. A feldolgozás végén három tömbbel rendelkeznénk. (rendmax, rendmin és veletlen)

A példánkban a rendezetlen tömbben fogjuk tárolni (a tömbelemek cserélgetésével) a lépések szerinti összehasonlítás eredményeit. Természetesen így egyszerre csak egyféleképpen lehet rendezni (növekvő vagy csökkenő).

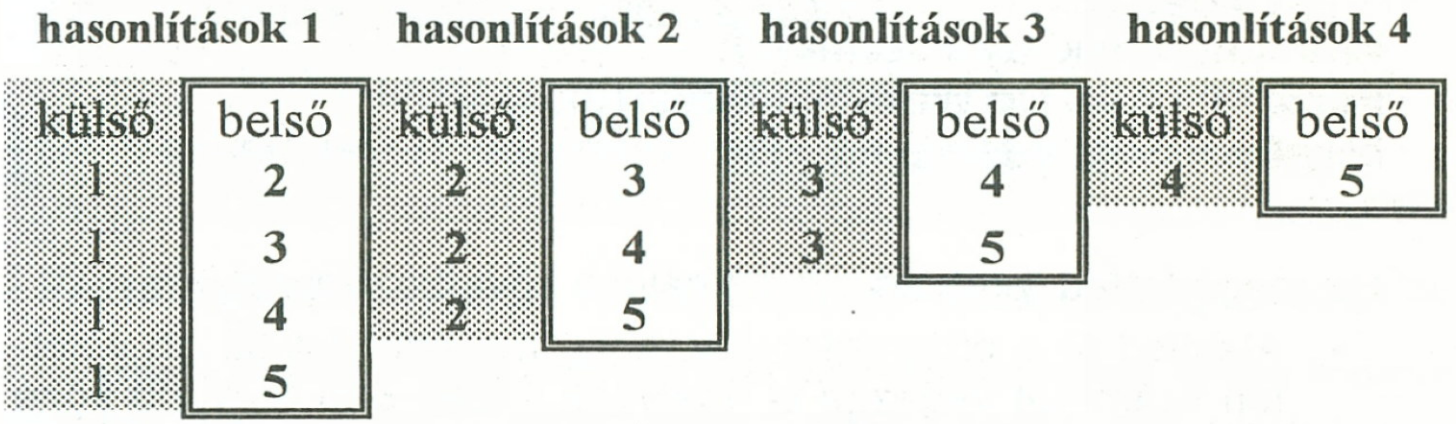
Jelöljük a for ciklus ciklusváltozóját: **Cv**-nek

A növekvő sorrendbe rendezés lépései:

cikus 1-től (**elemszám -1**)
ciklus (**külső ciklus változó+1**)-tól **elemszámig**

Ha a **külső Cv** szerinti tömbelem **nagyobb** mint a **belső Cv** szerinti, **akkor** tömbelemek cseréje
ciklus vége
ciklus vége

Lényege: a tömb első elemét hasonlítjuk az összes többihez, ha nagyobb akkor csere. A másodikat hasonlítjuk az utána következőkhöz, ha nagyobb akkor csere. Folytatva a módszert, az utolsó lépésben az utolsó előtti tömbelemet viszonyítjuk az utolsóhoz. A következő ábra mutatja 5 elem esetén az összehasonlítások lépéseit. A cserékhez szükségünk van egy változóra, melyben a tömb elemeit csere esetén tároljuk (ment).



Olvassunk be tömbbe megadott számú egésztípusú adatot és rendezzük növekvő sorrendbe!

```
Program rendez_minimummal;
Uses crt;
Var
   bill:Char;
   ment:Integer;
   rendez:array[1..900] of Integer;
   db,x,k:Byte;
Begin
   Clrscr;
   For x:=1 To 100 Do rendez[x]:=0;
    Write('Hány számot rendezzek? ');
   ReadIn(db);
   For x:=1 To db Do
```

```
    Begin
      Write(x,'.szam: ');
      ReadLn(rendez[x]);
    End;
  For k:=1 To db-1 Do
    For x:=k+1 To db Do
      If rendez[k]>rendez[x] Then
        Begin
          ment:=rendez[x];
          rendez[x]:=rendez[k];
          rendez[k]:=ment;
        End;
  WriteLn('A rendezett szamok:');
  For x:=1 To db Do Write(rendez[x]:10);
  bill:=ReadKey;
End.
```

✍ Ha megértetted, próbáld megcsinálni a következő programokat!

- Alakítsd át a programot úgy, hogy a rendezést 50 db véletlen számmal végezze el, melynek értéke 500 és 1000 közé esik!
- Módosítsd úgy, hogy egyszerre növekvő és csökkenő sorrendbe rendezzen két másik tömbbe tárolva az eredményt!
- Javítsd ki a deklarációt úgy, hogy stringeket rendezzen a program!

A névsorba rendezésnél szintén ugyanez a módszer. A fordító szerencsére a karaktereket minden tömbelem esetében rendre összehasonlítja. A probléma csak a magyar ékezetes betűkkel van, ugyanis ezek a karakter kódtáblában a kódszám szerint nem magyar abc sorrendbe helyezkednek el.

4.6. Kezdőértékkel rendelkező tömbök

Az egyszerű adattípusok esetében már beszéltünk **a konstans azonosítók deklarációjának lehetőségeiről.(Konstans és tipizált konstans)** Ha egy valós típusú tömb elemeit a programban a feldolgozás előtt nullázni akarjuk, akkor erre természetes a For ciklus használata. Más jellegű adatok esetén (ha pl.: a kezdőérték a hónapok nevei), akkor ez nem alkalmazható.

A tömb elemeinek kezdőértéket a deklarációban is adhatunk.

Példák deklarációra:

```
Const
egeszkezdo: array[1..2] Of integer=(2, -43);
nevekkezdo : array[1..3] Of String=('Hétfő','Kedd','Szerda');
korkezdo  : array[1..4] Of  Byte=(5, 34, 7, 76);
jelkonskezdo: array[1..3] Of Char=('2; ;'3', '5' ,#65);
tizedeskezdo: array[1..5] Of Real=(0.2, 2.34, -3.0, 234, 53);
```

✍ Határozd meg az előző példák alapján a következő tömb elemeket értékeit:

egeszkezdo[2];	egeszkezdo[1];	nevekkezdo[3] [1]
nevekkezdo[1];	jelkonskezdo[3];	nevekkezdo[1][2]
korkezdo[3];	korkezdo[4];	tizedeskezdo[3]

🕮 **A kezdőértékkel rendelkező Tömb deklarálása általánosan:**

Var

azonosító:**Array**[also..felso] **Of** típus=($adat_1$,$adat_2$...$adat_n$)

[also..felso] a tömb **méretét** adja meg résztartomány típusú konstansokkal kifejezve. Az **($adat_1$, $adat_2$...$adat_n$)** segítségével soroljuk fel a konstansokat. Vigyázni kell, hogy a felsorolás ugyanannyi db. elemet tartalmazzon a méretben megadott tartomány.

💻 *A bemenő adat legyen egy pozitív egész szám, mely nem nagyobb 31-nél. Ennek alapján* ***írja ki a program azokat a hónap neveket, amelyek ugyanennyi naposak.***

```
Program honap;
Uses Crt;
Const
ido: Array[1..12] Of Byte = (31,28,31,30,31,30,31,31,30,31,30,31);
honap:Array[1..12] Of String=('január','február','marcius','április',
          'május','június','július','augusztus','szeptember','október',
          'november','december');
Var  napszam, k: Byte;
Begin
   ClrScr;
   Write('Adja meg a hónap napjainak a számát: ');
   ReadLn(napszam);
```

```
    WriteLn('A következ hónapok ',napszam,' naposak:');
    For k:=1 To napszam Do
      If napszam=ido[k] Then WriteLn(honap[k]);
    ReadLn;
End.
```

A program lényege:

- Kezdőértékekkel rendelkező tömb deklarációja
- For ciklussal összehasonlítjuk az **ido** tömb elemeit az input adattal.
- Ha az összehasonlítás igaz, akkor kiíratjuk a **honap** tömb elemeit.

✍ Egészítsd ki a programot:

- Írjon ki nem megfelelő adat esetén erre vonatkozó üzenetet!
- Bővítsd még egy input adattal, az évszámával is! Ezután a program működjön szökőévekre is. Így ha szökőév van akkor a februárra a helyes input a 29.

💻 *Készítsünk címletező programot: Egy input adat alapján írja ki, hogyan lehet kifizetni egy összeget a lehető legkevesebb pénz egységekkel. Természetesen azokat a címleteket amelyből 0 db kell ne írja ki.*

```
Program cimletezo;
Uses Crt;
Var
  k:Byte;
  penz: LongInt;
  ch : Char;
Const
      cimlet:Array[1..10] Of
      LongInt=(5000,1000,500,100,50,20,10,5,2,1);
Begin
    ClrScr;
    Write('Kérem az összeget: ');
    ReadLn(penz);
    WriteLn;
    For k:=1 To 10 Do
       Begin
          If penz Div cimlet[k]=0 Then
```

```
            WriteLn(penz Div cimlet[k],' db. ',cimlet[k]:4,' forintos');
            penz:=penz Mod cimlet[k];
        End;
    ch:=ReadKey;
End.
```

A for ciklus a következő lépéseket fogja ismételni.

- A penz és cimlet változók közötti egész osztások végrehajtása
- Ha az eredmény nem nulla, akkor kiíratás.
- A penz változó új értékének kiszámítása egész osztással.

Készísünk programot, amely egy karakterláncban megszámolja az angol abc magánhangzóit.

```
Program maganhangkeres;
Uses Crt;
Var
    mondat : String;
    ch : Char;
    k,i : Integer;
Const
    angolmgh:Array[1..5] Of Char = ('A','E','I','O','U',);
    gyujt :Byte=0;
Begin
    ClrScr;
    Write('Kérem a mondatot! : ');
    ReadLn(mondat);
    For k:=1 To Length(mondat) Do
        For i:=1 To 5 Do
        If Upcase(mondat[k])=angolmgh[i] Then
        gyujt:= gyujt+1;
    Write('Magánhangzók száma: ',gyujt);
    ch:=ReadKey;
    End.
```

Reméljük, hogy az eddigiek szerint elegendő a következő magyarázat:

- a programtörzs nűködése

Ciklus1 a string hossza szerint

Ciklus2 a tömbelemek száma szerint

Ha a sztring karakter nagybetűs változata = a tömbelemmel

akkor a gyujt változö növelése eggyel

Ciklus2 vége

Ciklus1 vége

✍ Ennek alapján írj programot, amely külön-külön minden magánhangzót megszámol és különbözőnek tekinti a kicsi és nagybetűket!

Ciklusok II.

A For ciklussal kapcsolatban láttuk, hogy a ciklus végrehajtása egy meghatározott lépésszámban történt. Legtöbb feladat viszont olyan, hogy a **ciklust egy logikai feltételtől függően kell végrehajtani**. Erre a For ciklus nem alkalmas. Döntés hozatalt már megismertük az If utasítással kapcsolatban. A tanult formájában nem volt alkalmas ciklus szervezésre. Most már elárulhatjuk, hogy az If utasítással kapcsolatban nem említettük a feltételes ugró utasítást. (ennek használatával a programszervezés lépései nehezen követhetők).

1. A Repeat (ismét) ciklus

Egy adat feltételhez kötött beolvasásán keresztül magyarázzuk meg.

Olvassunk be olyan, egész típusú adatot amely (-12)-nél nagyobb de (30) -nál kisebb!

Ismételd

olvass be szamot

addig míg **szam>-12 és szam<30**

Behelyettesítve a nyelv alapszavait:

Repeat

ReadLn(szam);

Until (szam>-12) And (szam<30)

Általánosítva egy kicsit a ciklus:

Repeat

.....

utasítások

.....

Until logikai kifejezés

Működésének lényege: A Repeat és Until alapszavak közötti utasítást **addig ismétli, amíg a logikai kifejezés (feltétel) hamis**. Így a ciklusnak akkor van vége ha a feltétel igaz lesz (megfelelő volt az input adat).

A Repeat ciklus hátul tesztelő, tehát a ciklust egyszer mindenképpen végrehajtja.

Ezután a kis bevezető után nézzük programunkat:

☝ A (szam>-12) And (szam<30) logikai kifejezésben a zárójelek nagyon fontosak. Ellenkező esetben a kiértékelés más sorrendbe történne és a kiértékelés hiba miatt elakadna.

```
Program hatultesztelo1;
Uses crt;
Var
   szam:Integer;
   ch:Char;
Begin
   Clrscr;
   Repeat
      Write('Kérek egy szamot (-12<szamot<30): ');
      Readln(szam);
   Until (szam>-12) And (szam<30);
   WriteLn('Na végre');
   ch:=Readkey;
End.
```

2. A While ciklus

☝ Csak olyan Input adatot fogadjunk el, amely nullánál nem kisebb és 12-nél nem nagyobb.

Ha a feltételt **tagadjuk**, akkor az input adat akkor **nem megfelelő** ha szám nullánál kisebb (szam<0) **vagy** tizenkettőnél nagyobb (szam>12). Logikai kifejezéssel felírva: **(szam<0) Or (szam>12)**

A probléma megoldása:

Ciklus amíg(**While**) **(szam<0) Or (szam>12)** csináld(**Do**)
    csináld(**Begin**)
    beolvasás
ciklus vége(**End**)

Pascal nyelven általánosan felírva:

```
While feltétel Do
Begin
    beolvasás
End;
```

📖 **A While ciklus elöltesztelő. Belépés csak akkor történik, ha a feltétel igaz. Ha a logikai kifejezés értéke hamis, akkor kilép a ciklusból.**

☝ A mi példánkban a feltétel akkor igaz, ha az adat nem megfelelő. Ha erről elfelejtkezünk, akkor ez problémát okozhat. Ha a ciklus előtt nem lenne beolvasás, akkor szam változónak az értéke nulla. Mivel a kifejezés logikai kiértékelése ekkor hamis lenne a ciklus végrehajtása nem történne meg.

```
Program eloltesztelo
Uses crt;
Var
   szam:Integer;
   ch:Char;
   Const kiir='Kérek egy (0<=számot<=12): ';
Begin
   Clrscr;
   Write(kiir);
   Readln(szam);
   While (szam<0) Or (szam>12) Do
      Begin
         WriteLn('A szám nem megfelelő ');
         Write(kiir);
         Readln(szam);
      End;
   ch:=Readkey;
End.
```

✍ Oldd meg Repeat ciklussal is!

💻 *Olvassunk be String adatokat egy tömbbe! A beolvasás végét a csillag karakter jelezze!*

A megoldás főbb lépései:

- A tömb feltöltése üres string adatokkal('*')
- A változó nullázása, mellyel számoltatjuk a beolvasott elemek számát
- A repeat ciklusban a tömb elemek számlálása és a végjel ellenőrzése.

```
Program be_stringtomb;
Uses crt;
Var
   tombstr:Array[1..100] of String;
   k:Byte;
   ch:Char;
Begin
     Clrscr;
     Writeln;
     For k:=1 To 100 Do tombstr[k]:='';
     k:=0;
     Repeat
          k:=k+1;
          Write('Kérem a(z) ',k,'. elemet: ');
          Readln(tombstr[k]);
     Until tombstr[k]='*';
     ch:=Readkey;
End.
```

☝ A fenti módszerrel való beolvasás esetén az utolsó karakter a végjel. Ez további feldolgozáshoz nem szükséges. Ha a tömb elemekkel tovább dolgozunk, akkor a felső elemhatár eggyel kevesebb lesz mint annak a változónak az értéke, amellyel a tömb elemeket számoltuk.

✍ Hogyan lehetne javítani a feladaton úgy, hogy a k változó értéke pontosan megegyezzen az értékelendő tömb elemek számával?

✍ Oldd meg a feladatot előtesztelő ciklussal is! Próbáld megadni a beolvasást és az elemek számlálását úgy, hogy a változó értéke pontosan annyi legyen mint az értékelendő tömbelemek száma.

3. Adatok beolvasásának ellenőrzése

Numerikus adatok beolvasásánál futási hibát okoz, ha a billentyűzetről nem megfelelő adatot adunk meg.

Ennek ellenőrzésére az egyik módszer a már megismert **Val és Str eljárás használata**. Segítségükkel az adatot karakterláncként olvassuk be. Az input művelet után a **Val** eljárással ezt **numerikus adattá átalakítjuk.** Végül egy **Repeat** vagy **While** ciklussal megvizsgálhatjuk az átalakítás sikerét.

Olvassunk be billentyűzetről egy valóstípusú adatot. A műveletet addig ismételjük, amíg az adat helytelen.

```
Program IO_vizsga;
uses crt;
var
   szam:Real;
   kod:integer;
   szamstr:string;
   ch:char;
   Const kiir = 'Kérek egy számot: ';
Begin
   szamstr:=''; kod:=0;
   Clrscr; Write(kiir);
   readln(szamstr);
   Val(szamstr,szam,kod);
   While kod<>0  do
      Begin
         Write(kiir);
         ReadLn(szamstr);
         Val(szamstr, szam ,kod);
      End;
   ch:=ReadKey;
End.
```

Működése:

- A valós adatot karakterláncként olvassuk be
- Beolvasás után átalakítjuk a Val eljárás segítségével valós adattá.
- Beolvasás amíg kod<>0 kifejezés értéke igaz.

✍ Oldd meg a feladatot Repeat ciklussal is!

Az ellenőrzés másik lehetősége **IoResult függvény** használata.

Értéke sikeres fordítás esetén nulla. Ennek használatához szükség van a{$I+} és {$I-} kapcsolókra. **Ha be van kapcsolva {$I+} akkor hiba esetén leáll a program. Kikapcsolt állapotban {$I-} a programunkban megvizsgálhatjuk az Ioresult függvény értékét. Ha ez nulla akkor a numerikus input adat helyes volt**.

💻 *Olvassunk be numerikus adatot egy tömbbe a 0 végjelig! A beolvasást addig ismételjük, amíg az adat nem megfelelő! A program írja ki a beolvasandó adat index számát is!*

```
Program IO_vizsga1;
Uses crt;
Var
  tomb:Array[1..100] of Integer;
  k:Byte;
  ch:char;
{$i-}
Begin
    Clrscr;
    For k:=1 To 100 Do tomb[k]:=0;
    k:=0;
    WriteLn;
    Repeat
        k:=k+1;
        Repeat
            Gotoxy(1,k);
            Delline;
            Write('Kérem a(z) ',k,'. elemet: ');
            ReadLn(tomb[k]);
        Until Ioresult=0;
    Until tomb[k]=0;
    ch:=ReadKey;
End.
```

A program lényege:

A feladatot két egymásba ágyazott Repeat ciklussal oldjuk meg. A külső Repeat ciklus kilépési feltétele az adott elemű tömb nulla értéke. Ebben a

ciklusban számoljuk a tömbelemeket is. A belső ciklus az **Ioresult=0** feltétel szerint ismétli a beolvasást.

A program még tartalmaz egy szépítő elemet, nevezetesen sokszori beolvasás után ne legyen a képernyőn a sok helytelen adat. Ezt a **GotoXY és DelLine eljárások** oldják meg.

📖 **A DelLine segítségével a kurzor aktuális sorát tudjuk letörölni.**

4. Ciklusok alkalmazása

4.1. Egy trükk a Chr és Ord függvénnyekkel

Programunk váltson egy tízes számrendszerbeli számot tetszőleges számrendszerbe! Input adataink két Byte típusú szám. Az egyik az átalakítandó adat, a másik a számrendszer alapszáma.

Input:11, 2	Output:	1011
Input:43, 16	Output:	2B
Input:, 25, 8	Output:	31

Legyenek input változóink szam és alap, két változó hányadosa pedig hanyados.

Algoritmusunk lényege:

Ciklus

szam és alap változók között egész osztás

meghatározzuk a maradékot

gyűjtjük a maradékot egy string változóban

a szam változó legyen egyenlő hanyados változóval

amíg a hanyados = 0

```
Program szamrendszer;
Uses Crt;
Var
   hanyados, szam : LongInt;
   maradek, alap,kod : Byte;
   gyujt : String;
   ch : Char;
Begin
```

```
    ClrScr;
    gyujt:='';
    Write('A tizes számrendszerbeli szám: ':32);
    ReadLn(szam);
    GotoXY(2,9);
    Write('A számrendszer alapszáma: ':32);
    Read(alap);
    Repeat
        hanyados:=szam Div alap;
        maradek:=szam Mod alap;
        szam:=hanyados;
        kod:=48+maradek+Ord(maradek>9)*7;
        gyujt:=Chr(kod)+gyujt;
    Until hanyados=0;
    GotoXY(2,15);
    WriteLn(alap:13,'. számrendszerben: ',gyujt);
    ch:=Readkey;
End.
```

Ha a számrendszer alapszáma tíznél nagyobb, akkor a maradékok (az illető számrendszerben a számjegyek) csak betűkkel írhatók le. Lásd a hexadecimális számokat. Ha az alapszámot korlátozzuk (pl. csak tizenhatos számrendszerig működjön a program), akkor egy tömbben tárolhatnánk az

'1', '3', '4', '5', '6', '7', '8', '9', 'A', ' B', 'C', 'D', 'E', 'F'

karaktereket. Erre pontosan alkalmasak a tipizált konstans vagy a karakter résztartománnyal deklarált tömbök.

☝ Hasonlítsuk össze pl.: az **'0'..'9' és 'A'..'F'** résztartományokban lévő karakterek ASCII kódjait a lehetséges maradékokkal.

Ha a maradek<=9, akkor

maradék	**'0'**	**'1'**	**'2'**	**'3'**	**'4'**	**'5'**	**'6'**	**'7'**	**'8'**	**'9'**
kód	48	49	50	51	52	53	54	55	56	57

pl.: Chr(0+48)= **'0'** ; Chr(1+48)= **'1'**... Chr(9+48)= **'9'**

Így a számjegyeknek megfelelő karakter: Chr(maradek+48)

Ha a maradek>9, akkor

karakter	'A'	'B'	'C'	'D'	'E'	'F'
kód	65	66	67	68	69	70
maradék	10	11	12	13	14	15

Chr(10+55)= **'A'**; Chr(11+55)= **'B'**... Chr(15+55)= **'F'**

Így a számjegyeknek megfelelő karakter: Chr(maradek+55)

Tehát az egyik esetben **7-tel** többet kell hozzáadni a **48-as** kódszámhoz.

Erre a az esetre használható az Ord függvény, amely értéke logikai kifejezések esetén 1 vagy 0. **Tehát az Ord(maradek>9)*7 művelet eredménye az egyik esetben nulla, a másik esetben hét.**

Ezért a számjegyeknek megfelelő ASCII kód, kiszámítható és gyűjthető a következő értékadó utasításokkal:

```
kod:=48+maradek+Ord(maradek>9)*7;
gyujt:=Chr(kod)+gyujt;
```

amely a megoldás kulcsa.

4.2. Rendezés Repeat ciklus segítségével

A Repeat és For ciklus segítségével a rendezés a következőképpen oldható meg.:

A tömbelemeket összehasonlítjuk egy for ciklussal, amely mindig a szomszédos elemeket hasonlítja.(1-2, 2-3, 4-5, ...k-1,k). Ha az összehasonlítás feltétele igaz akkor az adatok cseréje történik és egy logikai változónak igaz (True) értéket adunk. Ezzel figyeltetjük, hogy volt-e csere. A For cikluson kívül ennek az értékét hamisra állítjuk. Az egészet egy Repeat ciklussal figyeltetjük. A Repeat ciklusból akkor lép ki a program, ha a Not csere logikai kifejezés értéke igaz. Ez azt jelenti, hogy az utolsó lépésben nem volt csere, tehát tömbünk rendezett. A többi programrészlet már ismerős számotokra.

```
Program rendez;
Uses crt;
Var
   csere:Boolean;
   ment:Word;
   rendez:Array[1..100] of Word;
   hany,x:Byte;
```

```
Begin
   Clrscr;
   Write('Hány számot rendezzek?');
   ReadIn(hany);
   For x:=1 To 100 Do rendez[x]:=0;
   For x:=1 To hany Do
      Begin
         Write(x,'.szam: ');
         ReadLn(rendez[x]);
      End;
   Repeat
         csere:=false;
         For x:=1 To hany-1 Do
            If rendez[x]>rendez[x+1] Then
               Begin
                  ment:=rendez[x];
                  rendez[x]:=rendez[x+1];
                  rendez[x+1]:=ment;
                  csere:=true;
               End;
   Until not csere;
   WriteLn('A rendezett szamok:')
   For x:=1 To hany Do
   Write(rendez[x]:20);
   ReadIn;
End.
```

5. Halmazok

A halmaz egy olyan **logikailag összetartozó egyedek összessége**, melyben az **elemek típusai megegyeznek**. A halmaz lévő egyedek a halmaz elemeinek nevezzük. A halmaz **elemeire hivatkozni nem tudunk** (nem indexelhetőek), hiszen nincsen sorrendjük.

5.1. Halmazok deklarációja

A halmaz elemei Byte, karakter és logikai típusúak. Ebből következik hogy maximum 256 eleme lehet. Nézzük először hogyan kell az ilyen adattípusokat deklarálni.

- Változók

Var	
szamok: Set Of Byte; karakter: Set Of Char;	pozitív számok 0-tól 255-ig ASCII karakterek

- Konstansok
  elemeiket [ ] zárójelek köztt soroljuk fel

Const	

nyolc=[1..8];
mgh=['a' , 'e' ,'i' ,'o' ,'u'];
vegyes=['a' ,'e' ,'j' ,'k' , 'l' ,'m', 'n'];
kevert=[0, 4, 12];
	tartomány:(1,2,3,4,5,6,7,8)

- Tipizált(kezdőértékkel rendelkező) konstansok

Const	

tipp:Set Of Byte=[12, 25, 54, 67, 88];
nagy_abc:Set Of Char=['A'.. 'Z'];
lotto> Set Of Byte =[5, 25, 67, 73, 88];
nincs_eleme Set Of Byte =[ ];
	az angol abc nagy betűi
	
	üres halmaz

5.2. Halmazműveletek és kifejezések

A műveleteket a fenti deklarációk alapján mutatjuk be:

- **Két halmaz egyesítése, jele: +**
  karakter:= mgh+vegyes **akkor** karakter =['a' , 'e' ,'i' ,'o' ,'u','j'..'n']
  szamok:=nyolc+kevert akkor szamok=[1..8,12]
- **Két halmaz metszete jele ***
  karakter:= karakter *['a' , 'e'] akkor karakter =['a' , 'e']
  szamok:=nyolc*tipp akkor szamok=[ ]
  lotoi= mgh*vegyes akkor karakter =['a' , 'e']
  nyert:=tipp*lotto akkor nyert:=[25,67]
- Hasonlítások:

↘ tartalmazás, jele: **In**

[a] In mgh	True
[z] In nagy_abc	False

↘ részhalmaz: <=

['a', j'..'l'] <=vegyes	True
mgh <=nagy_abc	False

Halmazok általános deklarációja:

```
Var
azonosító: Set Of elemtípus;
Const azonosító=[elem konstansok];
Const azonosító: Set Of elemtípus=[elem konstansok];
```

5.3. Példák halmazokra

Írjunk programot, mely megállapítja egy szó szótagjainak a számát!

A feladat halmazokkal megoldva nem nehéz. Csak a halmazokkal kapcsolatos furcsa jelöléseket kell megtanulni.

```
Program szotag;
Uses Crt;
Var
    Szo: String;
    I,K,Szotag: Byte;
Const
char=['A','a','Á','á','E','e','É','é','I','i','Í','í','O','o',
'Ó','ó','Ö','ö','Ő','ő','U','u','Ú','ú','Ü','ü','Ű','ű'];
Begin
    ClrScr;
    Szo := '';
    Szotag := 0;
    WriteLn('Adjon meg egy szót!');
```

```
    ReadLn(Szo);
    For I := 1 To Length(Szo) Do
        If Szo[i] In char Then Szotag:=Szotag+1;
        WriteLn('A begépelt szó ',Szotag,' db. szótagból áll.');
    ReadLn;
End.
```

A megoldás lényege: Egy For ciklus segítségével a string hossza alapján minden karaktert az In(benne) oprátor alapján megnézzük, hogy benne van-e az mgh konstans halmazban. Ha igen akkor egy nullával indított változót eggyel mindig megnövelünk.

Készítsünk lottó programot! A program véletlenszerűen sorsoljon ki öt lottó számot. Vigyázzunk arra, hogy a kihúzott számok különbözőek legyenek. Input adat legyen öt szám amelyet a program kezelője ad meg. Írja ki a gép, hogy hány találata volt az illetőnek és mik volt ezek a számok!

A program halmazok segítségével oldható meg a legkönnyebben:

```
Program halmazok;
Uses Crt;
Var nyert,tiedtipp,
geptipp: Set Of 1..90;
fogad,lottoszam,i:Byte;
Begin
  Clrscr;
  tiedtipp:=[ ];geptipp:=[ ];                    adatok kezdőértéke
  WriteLn('Lottó sorsoáas. Tipppeljen');
  Randomize;
  For i:=1 to 5 do
  Begin
    Repeat
      lottoszam:=Random(90)+1;                   sorsolás ismétlése
    Until Not(lottoszam In geptipp);             ha már volt ilyen
    Write(i,'. tippp: ');
    ReadIn(fogad);                               fogadó tippjei
```

```
    tiedtipp:=tiedtipp+[fogad];
    geptipp:=geptipp+[lottoszam];
  End;
  nyert:=geptipp*tiedtipp;
  Write('A kihúzott számok:');
  For i:=1 to 90 Do
  If (i In geptipp) Then Write(i:3);
  WriteLn;
  If nyert=[ ] Then WriteLn('Nem nyert!')
     Else
       Begin
         Write('  A nyerő szamok:');
         For i:=1 To 90 Do
           If (i In nyert) then Write(i:3);
       End;
  ReadIn;
End.
```

tippek gyűjtése a halmazokban

találat halmaz

a gép tipjeinek kíratása

az eltalált számok kiírása

Input adat egy String. Készítsünk programot, amely minden magánhangzó után szúrjon be egy 'v' karaktert és utána ugyanazt a magánhangzót!

Pl.: iskola helyett iviskovolava

pl.: **Insert**('sok','moadik',2)='mosakodik'

```
Program madarnyelv
Uses Crt
Var
  mondat,oriz : String;
  ch : Char;
  k,d : Byte;
Const
mgh=['A','a','Á','á','E','e','É','é','I','i','Í','í','O','o',
'Ó','ó','Ö','ö','Ő','ő','U','u','Ú','ú','Ü','ü','Ű','ű'];
d:=0;
ClrScr;
Write('Kérem a szót! : ');
```

```
ReadLn(mondat);
oriz:=mondat;
For k:=1 To Length(mondat) Do
   Begin
     ch:=mondat[k];
     If mondat[k] In mgh Then
       Begin
       Insert('v'+ch,oriz,k+1+d);
       d:=d+2;
       End;
   End;
WriteLn(oriz);
ch:=ReadKey;
End.
```

A program egy új eljárást tartalmazott: **Insert**

📖 **Az Insert eljárás egy karakterláncot szúr be egy másik karakterláncba egy adott helytől. Általánosan: Insert(mit,mibe,honnan), ahol mit, mibe String azonosítók és honnan pedig Byte típusú azonosító.**

A 'v' karaktert és a magánhangzót a szóból leválasztott karakter után kell beszúrni. Ennek helyét a k+1 kifejezés jelöli. Minden beszúrásnál az oriz változó kettővel hosszabb lesz. Ezért a következő helyet mindig növelni kell ezzel az értékkel. Ezt a célt szolgálja a d:=d+2; értékadó utasítás. Tehát a beszúrás helyét az eredeti szó karaktereinek helyéhez viszonyítva a k+1+d kifejezés értéke adja meg.

Input adat egy szó. A program döntse el, hogy milyen magas, mély vagy vegyes hangrendű

```
Program hangrend;
Uses Crt;
Var
  ossz,gyujt: Set Of Char;
  szo : String;
  ch : Char;
  k : Integer;
Const
magas=['E','e','É','é','I','i','Í','í','Ö','ö','Ő','ő','Ü','ü','ű','Ű'];
```

```
mely=['A','a','Á','á','O','o','Ó','ó','U','u','Ú','ú'];
Begin
ossz:=magas+mely;
gyujt:=[ ];
ClrScr;
Write('Kérem a szót! : ');
ReadLn(szo);
For k:=1 To Length(szo) Do
      If szo[k] In ossz Then gyujt:=gyujt+[szo[k]];
If gyujt<=magas Then WriteLn('Magas hangrendű')
  Else
     If gyujt<=mely  Then WriteLn('Mély hangrendű')
       Else WriteLn('Vegyes hangrendű');
ch:=ReadKey;
End.
```

A megoldás lényege: A feladatot halmazműveletekkel és egymásba ágyazott feltételes utasításokkal oldottuk meg.

Ciklus egytől a szo változó hosszáig

Ha a szo változó karaktere magánhangzó

akkor gyűjtjük egy gyujt azonosítójú halmazban

Ciklus vége

Ha a gyujt halmaz részhalmaza a magas halmaznak

akkor magas hangrendű

különben ha a mely halmaz részhalmaza **akkor** mély hangrendű

különben vegyes hangrendű;

Eljárások, függvények

1. Alprogram fogalma és fajtái

Eddigi programjainkban is észrevehettük, hogy bizonyos **műveleteket többször végre kellett hajtani**. Előfordult az is, hogy egyes **programrészletek nem sokban különböztek egymástól**. Ha programunk túl **terjengős, akkor logikailag áttekinthetetlenné válik**. Ez megnehezíti a program javítását, tesztelését. Ennek megkönnyítésére a Pascal lehetőséget ad arra, hogy a logikailag jól szétválasztható **programrészleteket egységes egészként kezeljük.** Az így megírt modulokat egyenként tesztelhetjük. Az ismétlődő részeket elegendő egyszer megírnunk, többszöri alkalmazása így könnyen megvalósítható.

📖 **Az alprogram tehát logikailag összetartozó utasításcsoportok halmaza, valamely összetett tevékenység végrehajtására. Végrehajtásához, a főprogram bármely helyén elegendő azonosítójával hivatkozni rá. Az alprogramokat a főprogram deklarációs részében adhatjuk meg.**

Az alprogramok és a főprogram között **értékátadással input-output műveletek is történhetnek**. A Pascal két fajta alprogramot ismer, az **eljárást és a függvényeket.**

2. Eljárások

2.1. Paraméter nélküli eljárás

Írjunk ki a képernyőre 100 db véletlen számot, egytől tízezerig. Ha a képernyő megtelt, akkor függesszük fel a program futását. A program az <ESC> billentyűre folytatódjon és írjon ki a 25. sorba erre vonatkozó tájékoztatást!

☝ Az <ESC> billentyű decimális kódja 27, így ezt a karaktert #27-tel jelölhetjük. A képernyő kezeléséhez a már ismert Readkey függvényre és WhereY eljárásra lesz szükségünk.

Eljárásunk feladata, hogy ha kurzor aktuális sora 25, akkor programunk végrehajtását felfüggessze.

```
Program nincs_parameter;
Uses Crt;
Var
   k:Byte;
   Procedure varakoz;
   Var
     ch:Char;
     Begin
        Write('Tovább:<ESC>');
        Repeat
           ch := ReadKey;
        Until ch=#27;
        Clrscr;
     End;
   Begin
   ClrScr;
   Randomize;
   For k := 1 To 100 Do
      Begin
         If WhereY=25 Then varakoz;
         WriteLn(Random(10000)+1);
      End;
   ReadLn;
End.
```

Eljárás feje
Deklarációs rész
Eljárás kezdete
Végrehajtandó rész
(Eljárás törzse)
Eljárás vége

Eljárásunkra a varakoz azonosítóval hivatkozhatunk. Főprogramunk For ciklusa írja ki a véletlen számokat. Ha a kurzor vízszintes pozíciója 25 akkor a kiíratás előtt meghívja a varakoz eljárást.

Az eljárás működése:

Eljárás kezdődik

- String kiíratás
  Ismételd
    egy karakter olvasása billentyűzetről
  Amíg a kod =27
- képernyőtörlés

Eljárás vége

Az eljárás meghívása után programunk vezérlése visszaadódik a hívott hely utáni utasításra. Előző példánkban, a varakoz eljárás nem vett át értéket a főprogramtól. Ezeket **paraméter nélküli eljárásoknak nevezzük**.

2.2. Az eljárás általános alakja

Az eljárások pontos működésének leírása előtt, nézzünk még egy példát:

Programunk feladata: Osszuk fel a képernyőt két ablakra, melyeknek magassága 11 sor. Az ablakokat vegye körül piros szegély, háttere pedig fekete legyen. Írjuk ki az ablakokba a következő szövegeket:

'Ez a felső ablak.' illetve 'Ez az alsó ablak.'

```
Program ablakok;
Uses Crt;
   Var
   szavak:String;
   Procedure ablak(s1,s2:Byte;mondat:String);
   Var oszlop1,oszlop2:Byte;
   Begin
       oszlop1 := 2;oszlop2 := 79;
       Window(oszlop1,s1,oszlop2,s2);
       TextColor(White);
       TextBackGround(black);ClrScr;
       Write(mondat);
   End;
Begin
   TextBackGround(red);ClrScr;
   szavak := 'Ez a felső ablak';
   ablak(2,12,szavak);
   szavak := 'Ez az alsó ablak';
   ablak(14,24,szavak);
   ReadLn;
End.
```

Az ablakok létrehozásához kétszer kell végrehajtani a Window beépített eljárást. Tehát az ablak eljárást kétszer fogjuk meghívni.

Az ablakok közötti különbség a vízszintes pozíciókat meghatározó sorok száma és a kiíratandó szöveg. Ennek alapján eljárásunk input adatai:

- az ablak bal felső sarkának sorszáma
- az ablak jobb alsó sarkának sorszáma
- a szavak változó értéke

A program lényege:

- A teljes karakteres képernyő háttérszínének beállítása pirosra
- értékadás a mondat stringnek
- ablak eljárás hívása a 2 és 12 konstansokkal és a szavak változó aktuális értékével
- újabb értékadás a szavak stringnek
- ablak eljárás ismételt hívása a 14 és 24 konstansokkal és a szavak változó aktuális értékével

Az ablak eljárás működésének lényege:

- értékátadás az s1, s2 formális paramétereknek
- Az ablak oszlop pozícióinak beállítása
- ablak helyének meghatározása
- Írás színének beállítása fehérre
- az ablak háttérszínének megváltoztatása feketére

Az eljárás általános formája a következőképpen határozható meg:

```
Procedure eljárás_azonosító (paraméterlista)
   Deklarációk
   ..........
   ..........
   ..........
   Begin
      .........
      utasítás1; utasítás2;....
      .........
   End;
```

Eljárás feje

Deklarációs rész

Eljárás törzse

📖 **Az eljárás fejét a procedure fenntartott szó vezeti be, melyet az eljárás neve követ.**

- Az eljárás azonosító után zárójelben soroljuk fel **a formális paramétereket**, melyek a főprogram és alprogram közötti **értékátadást közvetítik**
- Azonos típusú paramétereket **vesszővel** választjuk el
- Az utolsó paraméter után **kettőspontot téve meg kell adni azok típusát**
- Különböző típusokat a **pontosvesszővel** különböztetjük meg.
- A paraméterlista **elhagyható** (paraméter nélküli eljárások)

2.3. Eljárások hívása

- **Formális paraméterek**

  Előző példánkban az eljárás feje (s1,s2:**Byte**, mondat:**String**) volt. Az s1, s2, mondat azonosítókat **formális paramétereknek** nevezzük. Az eljárásoknak a **formális paraméterek segítségével adhatunk bemenő értékeket.**

- **Aktuális paraméterek**

  A főprogramunkban az eljárásunkat az ablak159 (2,14,szavak) és ablak(14,24,szavak) utasításokkal aktivizáltuk. A zárójelben lévő konstansokat és változót **aktuális paramétereknek** hívjuk.

📖 **Eljárások hívása általánosan:**
eljárás_azonosító(aktuális paraméterlista)
Az eljárás fejében lévő formális paraméterek híváskor átveszik az aktuális paraméterek értékeit (paraméterátadás). Az eljárásban az értékekre a formális paraméterekkel lehet hivatkozni.

☝ A formális és aktuális paraméterek számának, sorrendjének és típusának meg kell egyeznie.

3. Paraméterátadás módjai

Az eljárásoknak a formális paraméterek közvetítésével érteket adhatunk át. Az paraméterátadásnak két fajtáját különböztetjük meg:

- **Érték szerinti**
- **Cím szerinti** értékátadás

3.1. Érték szerinti paraméterátadás

Az előző példánkban az érték szerinti paraméterátadásra láttunk példát. Mindhárom formális paraméter (s1,s2,mondat) esetében érték szerinti paraméter átadás történt. Az aktuális paraméterek ebben az esetben lehetnek konstansok, változók és kifejezések. Kifejezések esetén működése azok kiértékelésével kezdődik.

Működésének lényege:

📖 Csak **egyirányú értékátadást közvetít**, mégpedig az **alprogram felé**. Az eljárás hívásakor a **fordító lefoglal egy memóriacímet a formális paraméter számára.** A lefoglalt **memóriaterületet tárolja** az **aktuális paraméter értékét**. Erre az értékre **az alprogramban a formális paraméter azonosítójával hivatkozhatunk**. Az eljárás befejezésekor ez a memóriaterület felszabadul. Akárhogyan változik meg az alprogramban a formális paraméter értéke, ez **nincs hatással az aktuális paraméterre.** Ebben az esetben az **eljárás fejében** az azonosítók előtt **nem használjuk a Var kulcsszót**.

3.2. Cím szerinti paraméterátadás

Ennek megértésére nézzük a következő példát:

💻 *Készítsünk programot, mely két eljárást tartalmaz. Az egyik esetben **érték** szerinti, a másik esetben **cím szerinti** paraméter átadás történjen. A paraméterek String típusúak legyenek. **Hívjuk** mindkét eljárást **ugyanazzal a szöveggel** és az alprogramokban **változtassuk meg a formális paramétereket** ugyanarra a szövegre.*

```
Program ertekatadasok;
Uses Crt;
   Var
   szavak:String;
   Procedure figyel(Var mondat:String);
     Begin
       mondat := 'Figyeltem arra amit mondtál.';
     End;
```

```
Procedure nem_figyel(mondat:String);
  Begin
    mondat := 'Figyeltem arra, amit mondtál';
  End;
Begin
  Clrscr;
  szavak := 'Mondtam valamit, figyeltél rám?';
  figyel(szavak);
  WriteLn(szavak);
  szavak := 'Mondtam valamit, figyeltél rám?';
  nem_figyel(szavak);
  Write(szavak);
  ReadLn;
End.
```

	A monitoron megjelenő szöveg
szavak értéke	Mondtam valamit, figyeltél rám?
figyel(szavak)	
szavak értéke	Figyeltem arra, amit mondtál.
szavak értéke	Mondtam valamit, figyeltél rám?
nem_figyel(szavak)	
szavak értéke	Mondtam valamit, figyeltél rám?

Programunkban a figyel eljárásnál cím szerinti, a nem_figyel eljárásnál érték szerinti paraméterátadás történik. **Mindkét alprogramot** a szavak változóval jelölt karaktersorozattal hívjuk, melynek **tartalma a 'Mondtam valamit, figyeltél rám?'** szöveg. Az eljárásokban a **formális paraméterek értékét** egyformán **a** 'Figyeltem arra, amit mondtál' **szövegre változtatjuk.**

Az eljárás hívása után a főprogramban az aktuális paraméterek (szavak) értéke az **első esetben megváltozott**, míg a **második esetben változatlan maradt.**

A cím szerinti paraméterátadás lényege:

🕮 A program indításakor a fordító **memóriacímet foglal az aktuális paraméternek.** Cím szerinti paraméterátadásnál, az eljárás **hívásakor a**

formális paraméter felveszi az aktuális paraméter címét. Így nem jött létre új változó, csupán **ugyanarra memóriaterületre két névvel hivatkozunk.** Ha az eljárás törzsben a formális paraméter értéke megváltozik, akkor ez által **az aktuális paraméter értéke is változni fog. Tehát hívás után az aktuális paraméter értéke a formális paraméter értékével fog megegyezni.**

☝ Az előbbiek szerint cím szerinti paraméterátadásnál az aktuális paraméter nem lehet sem kifejezés, sem konstans. **Csak vátozó azonosító lehet**. Az ilyen típusú értékadásokat az eljárás törzsben a **Var kulcsszóval jelöljük** a formális paraméterek előtt.

3.3. Globális és lokális változók, érvényességi kör

Mielőtt megbeszélnénk a fogalmat, nézzünk egy példaprogramot:

Készítsünk eljárás segítségével programot, amely egy kezdőértékkel és végértékkel megadott számoknak kiszámolja a négyzetösszegét. Pl.: ha input adatunk -2 és 3, akkor az output $(-2)^2+(-1)^2+(-0)^2+1^2+2^2+3^2 = 4+1+0+1+4+9=19$

A program lényege:

- A főprogram
  - ↘ globális változók: a program input adatainak és az eljárás aktuális paraméterének deklarálása
  - ↘ Input adatok beolvasása
  - ↘ a problémát megoldó eljárás hívása cím szerinti paraméterátadással
  - ↘ az eredmény kiíratása
- A négyzetösszeget előállító eljárás
  - ↘ negyzet_sum eljárás fejében cím szerinti formális paraméter megadása
  - ↘ az eljárás helyi (lokális) változójának deklarálása. Ez lesz a For ciklus ciklusváltozója
  - ↘ Az eljárás törzse

    For ciklus segítségével az input adatok tartománya szerint összegezi a négyzetszámokat. Az összeg értékét az eljárás befejezésekor az aktuális paraméter jelöli.

```
Program negyzetszamok;
Uses Crt;
Var
   kezdi,befejezi,
   osszeg:LongInt;
   Procedure negyzet_sum(Var
gyujt:LongInt);
      Var
         k:Integer;
      Begin
         gyujt:=0;
         For k := kezdi To befejezi Do
           Begin
              gyujt:=gyujt+k*k;
              Write(k * k)
              If k<befejezi Then Write('+')
           End;
      End;
   Begin
   ClrScr;
   Write('Mettől kezdjem? ');
   ReadLn(kezdi);
   Write('Melyik legyen az utolso szám? ');
   ReadLn(befejezi);
   negyzet_sum(osszeg);
   Write(' = ',osszeg);
   ReadLn;
End.
```

globális változók

lokális változók: a **k** és **gyujt**

☝ A **lokális és globális** változó fogalmát óvatosan kell megközelíteni. A **lokális** változó csak a program **egy adott helyén ismert**, míg a **globális** változót **a fordító több blokkban is felismeri**. Ezt nevezzük érvényességi körnek. Példáink túl egyszerűek voltak a fogalom pontos megfogalmazásához. Tehát a fogalom egyes esetekben relatív, vagyis viszonylagos.

🕮 **Lokális változók fogalma: Egy kisebb szerkezeti egység változója (pl. eljárás) lokális változó, az őt tartalmazó nagyobb szerkezeti egység szempontjából.**

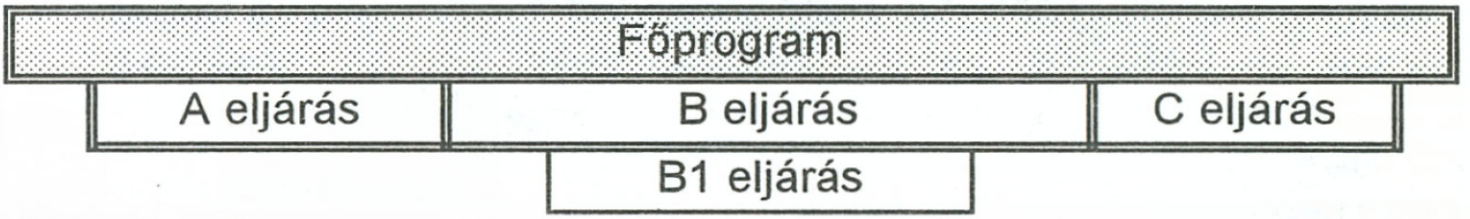
- **Egyértelműen lokális változók**: pl.: Minden eljárás formális paraméteri és deklarációs részben megadott változói az őt tartalmazó főprogram számára. Ezek a változók a főprogram számára ismeretlenek.
- **Egyértelműen globális változók**: pl.: Minden a főprogramban deklarált változó a főprogramban deklarált eljárások számára. Ezek a változók az eljárások számára ismertek.
- **Globális és lokális változók viszonylagossága**

☝ **Alprogramok** a feltételes utasításokhoz hasonlóan **egymásba ágyazhatóak**. Tehát ennek a problémának a megértéséhez foglalkoznunk kell ezzel a lehetőséggel is.

Az alábbi ábra egy lehetőséget szemléltetne:

↘ Főprogram tartalmazza az A, B, C eljárásokat

↘ B eljárás tartalmazza a B1 eljárást



Lehetőségek:

1. A **B1** eljárás minden változója és formális paramétere lokális
2. A **B** eljárás minden változója és formális paramétere **lokális** a **főprogram szempontjából és globálisa B1-nek.**
3. Az **A** és **C** eljárások minden változója és formális paramétere lokális változó.

☝ Hosszabb programok esetén vigyázni kell a globális változók használatára. Az eljárásokban használt segéd változókat lehetőleg az alprogramban kell deklarálni. Így a főprogramban, ha ugyanazt a változó nevet használjuk, akkor nem érhet bennünket kellemetlen meglepetés.

3.4. Felhasználói típus

Eddigi feladatainkban láthattuk, hogy az eljárásban lévő paraméterek egyszerű típusúak voltak. Összetett típusok esetén (pl.: tömb, halmaz) a paraméterátadásra más módszert kell alkalmazni. A Pascal nyelvben lehetőség van saját típusok létrehozására.

Hogyan hozhatnánk létre egy olyan tömböt, melynek elemei halmazok?

pl.:

```
Type halmaz = Set Of Char;
Var kar_tomb : Array[1..10] Of halmaz;
```

Ebben a példában egy olyan tömböt deklaráltunk, melynek maximum tíz eleme lehet, és minden tömbelem egy karakter halmaz.

Új típus létrehozása a Type kulcsszóval történik. A kulcsszó után a létrehozandó típus azonosítóját adjuk meg, majd egyenlőségjel után egy létező típust. Általánosan:

Type azonosító = meglévő_típus;

Ennek alapján a **Var alapszóval már az új típust is használhatjuk** a deklarációkban.

Állítson elő programunk véletlen számokat egytől ezerig! Input adatként olvassuk be a darabszámot! Írjunk eljárásokat, melyek feladatai:

- *létrehozza, kiírja és tárolja a véletlen számokat további feldolgozásra*
- *rendezi növekvő sorrendbe*
- *kiírja a rendezett számokat a képernyőre*

Mindhárom eljárásnak a formális és aktuális paramétere tömb típusú adat lesz. A paraméter típusát csak új típus létrehozásával adhatjuk meg.

Az új típust a **Type adat = Array[1..100] Of Word; típus deklarációval** hozzuk létre. Az eljárások fejében ezután már használhatjuk az adatbe típusazonosítót. Aktuális paraméterként a főprogramban szintén ugyanilyen típust kell deklarálni. (Var ebbe : adatbe)

A program működésének rövid ismertetése:

- A véletlen számok mennyiségének beolvasása
- A veletlen eljárás hívása. Az aktuális paraméter az ebbe változó mely adatbe **típusú.** A paraméterátadás jelen esetben cím szerinti. Ezért az eljárás befejezésekor az ebbe **változó fogja tartalmazni** a **rendezetlen véletlen számokat**. Itt történik a feldolgozandó adatok kiíratása is.

- A rendez eljárást szintén az ebbe aktuális paraméterrel hívjuk, szintén cím szerinti értékátadással. Az eljárás a már megismert módszer szerint elvégzi a rendezést. Kilépéskor az ebbe változó már **a rendezett adatokat fogja tartalmazni**.
- A program befejezése előtt a kiir eljárást hívjuk. A paraméterátadás érték szerinti. Ennek oka, hogy ez az eljárás már nem változtatja az ebbe változó tartalmát.

```
Uses Crt;
Type
   adatbe = Array[1..100] Of Word;
Var
   ebbe : adatbe;
   db,k  : Byte;
   ch : Char;
   Procedure veletlen(Var veletlentomb:adatbe);
     Begin
       Randomize;
       For k := 1 To 100 Do veletlentomb[k] := 0;
       For k := 1 To db Do
         Begin
           veletlentomb[k] := Random(10000);
           Write(veletlentomb[k]:8);
         End;
       ch := ReadKey;
     End;
   Procedure rendez(Var rendezetlen:adatbe);
   Var
     csere : Boolean;
     ment:Word;
     Begin
       Repeat
         csere := False;
         For k := 1 To db-1 Do
           Begin
             If rendezetlen[k]>rendezetlen[k+1] Then
               Begin
                 ment := rendezetlen[k];
                 rendezetlen[k] := rendezetlen[k+1];
```

```
                rendezetlen[k+1] := ment;
                csere := True;
              End;
          End;
        Until csere=False;
      End;
    Procedure kiir(rendezett:adatbe);
      Begin
        WriteLn;WriteLn('A rendezett számok:');
        For k := 1 To db Do
          Write(rendezett[k]:8);
      End;
Begin
    ClrScr;
    WriteLn('Hány db. véletlen számot rendezzek:');
    ReadLn(db);
    WriteLn('A rendezetlen számok:');
    veletlen(ebbe);
    rendez(ebbe);
    kiir(ebbe);
    ch := ReadKey;
End.
```

4. Függvények

A függvények fogalma könyvünkben már sokszor előfordult. Matematika órán is nagyon sokszor hallottál már róla. A Pascal nyelvnek már ismertettük néhány beépített(szabványos) függvényét. Közös tulajdonságuk, hogy valamilyen értéket szolgáltattak vissza. Ilyenek voltak pl.: **Length, Chr, Ord, WhereX, WhereY, ReadKey, Upcase** stb.

A nyelv lehetőséget ad saját felhasználói függvények készítésére. Ezek sokban hasonlítanak az eljárásokhoz.

Függvények általános alakja:

```
Fuction azonosító (paraméterlista):függvény_típus
   Deklarációk
   ..........
   ..........
   ..........
   Begin
      .........
      utasítás1; utasítás2;....
      .........
   End;
```

függvény feje

Deklarációs rész

függvény törzse

Mi tehát a lényeges különbség?

- A formális paraméterek után kettősponttal elválasztva meg kell adni a függvény által visszaadott érték típusát
- A függvény törzsében legalább egyszer kell szerepelnie egy olyan értékadó utasításnak, melynek bal oldalán a függvény azonosítója áll.
- A függvények mindig szolgáltatnak vissza valamilyen értéket
- Csak egy éréket adhat vissza
- A bemenő paraméter típusa különbözhet a függvény által visszaadott értéktől
- A függvény szerepelhet kifejezésekben is

A következő példánkban függvényünk feladata fordítottja lesz a már ismert **Upcase** függvényhez viszonyítva.

Input adatunk legyen egy karakterlánc, azonosítója **mondat**. *Készítsünk programot, amely a* **mondat** *String angol abc szerinti nagybetűit kisbetűkre cseréli. A karakterek kis betűsítését függvény eljárással oldjuk meg!*

```
Program fuggveny;
Uses crt;
Var
 mondat:String;
 k:Byte;
   Function kisbetu(betu:char):char;
      Var
      kod:byte;
```

```
    Begin
        kod := Ord(betu)+ 32*Ord(betu In ['A'..'Z']);
        kisbetu := Chr(kod);
    End;
Begin
ClrScr;
Write('Kérem a mondatot: ');
ReadLn(mondat);
WriteLn;
For k := 1 To Length(mondat) Do
mondat[k] := kisbetu(mondat[k]);
ReadLn;
End.
```

A főprogram főbb lépései:

- A karakterlánc inputja
- Ciklus 1-től a szöveg hosszáig
  a mondat karakterlánc megfelelő karakterének cseréje kisbetűre a kisbetu függvény hívásával
  Ciklus vége
- A megváltozott string kiíratása

Függvényünk feladata, hogy a karakterlánc megfelelő karakterét kisbetűvé alakítsa. Ezt csak akkor kell végrehajtania, ha a karakter eleme a ['A'..'Z'] halmazban. Talán ennek a megoldása nehezebb, mint maga a függvény fogalma. Első lépésben az ember egy feltételes utasításra és For ciklus használatára gondol karakter résztartománnyal. Nézzünk más lehetőséget:

A másik lehetőség a karakterek ASCII kódja. Az 'a'..'z' és 'A'..'Z' (#97..#122 és #65..#95) karakterek kódjai között a különbség rendre 32.

Ezért a megoldás feltételes utasítás segítségével a Pascal Chr és Ord függvényei alapján:

```
kod:=Ord(betu)
If betu In ['A'..'Z'] Then kisbetu := Chr(kod+32)
    Else kisbetu := betu;
```

Programunkban egy másik megoldást mutattunk be, amellyel a feltételes utasítás kikerülhető. Erre a trükkre már volt példa az előzőekben. Nevezetesen az Ord függvény értéke logikai kifejezések esetén 1 vagy 0.

- Ord(**betu In ['A'..'Z']= 1** ha a karakter angol nagybetű
- Ord(**betu In ['A'..'Z']= 0** ha a karakter nem angol nagybetű

Ezért a kod := **Ord**(betu)+ 32***Ord**(betu **In** ['A'..'Z']); értékadó utasítás, ha változtatás szükséges, akkor a megfelelő kódot számolja ki. Ellenkező esetben a karakteren nem változtat. Feladatunkban a függvény azonosítója a kisbetu volt. A függvény értékét a kisbetu:=Chr(kod); utasítással számoltuk ki.

Ezután a kitérő után térjünk rá a függvény fogalmának pontosítására:

```
Procedure f_azonosító (paraméterlista)
   Deklarációk
   ..........
   ..........
   ..........
   Begin
      .........
      utasítás1; utasítás2;...
      .........
   End;
```

Eljárás feje

Deklarációs rész (lokális változók)

Eljárás törzse

4.1. Egy példa a rekurzióra

Alprogramok egy másik alprogramot is hívhatnak. Ilyenkor a végrehajtást mindig a legutoljára hívott alprogram kezdi. Hívás után a vezérlés visszaadódik a hívott helyre. Ha a hívó hely szintén alprogram volt akkor ennek befejezése után ennek a hívó helyén folytatódik a program végrehajtása. Pl.: Tegyük fel, hogy A, B, C, D egymásba ágyazott alprogramok és A hívja B-t, B hívja C-t, C hívja D-t. A végrehajtás a D-vel kezdődik, majd C-vel folytatódik....., és legutoljára az A fejezi be végrehajtást a hívó helytől kezdődően.

Rekurzióról akkor beszélünk, ha egy eljárás önmagát hívja. Példa válogatja, hogy érdemes-e használni. Nézzünk egy példát a hatványozással kapcsolatban. A számláló ciklusokkal ismerkedve ezt a feladatot már megoldottuk.

Számoljuk ki egy egész szám pozitív kitevős hatványát rekurzív hívással, ha input adataink a hatvány alap és a kitevő.

A régebbi megoldásunk ismételt szorzáson alapult. Egy for ciklus segítségével a szorzat:=szorzat

```
Program rekurzio;
Uses crt;
Var
 al,kit:Word;
  ertek:LongInt;
Function hatvany(Var alap,kitevo:Word):LongInt;
    Begin
      If kitevo=0 Then hatvany := 1
      Else hatvany := alap*hatvany(alap,kitevo-1);
    End;
Begin
 Clrscr;
 WriteLn('Kérem az alapot és kitevőt : ');
 ReadLn(al,kit);
 ertek := hatvany(al,kit);
 WriteLn(ertek);
 ReadLn;
End.
```

A fuggvényünk azonosítója legyen hatvany.

A rekurzív hívást az

```
If kitevo=0 Then hatvany := 1
      Else hatvany := hatvany(alap,kitevo-1)*alap;
```

programrészlet végzi. Játsszuk el 2 és 4 esetén!

- A hívást a főprogram kezdi ertek:=hatvany(2,4)

- A függvény törzsében lévő feltételes szerkezet hatására az Else ág hajtódik végre:
- Újra hívja önmagát, de a Then ág még mindig hamis, így újra az Else ág
- Még mindig az Else ág
- Utolsó hívás
- Végül a Then ág

```
hatvany := 2*hatvany(2, 3)
hatvany := 2*hatvany(2, 2)
hatvany := 2*hatvany(2, 1)
hatvany:= 2*hatvany(2, 0)
hatvany:=1
```

A visszatérés a hívott helyre fordított sorrendbe történik (alulról felfelé). Ha visszahelyettesítjük az értékátadásokat:

akkor hatvany:=1*2*2*2*2=16 így a főprogramban az ertek:=hatvany(2,4) alapján az ertek valtozó 16.

Állományok kezelésének alapjai

Eddig az I/O műveleteket a billentyűzet és a monitor között hajtottuk végre. Nagyobb mennyiségű adatok feldolgozása esetén ez eléggé fáradságos. A másik szempont, hogy a feldolgozás után az Output adatokat meg szeretnénk őrizni egy későbbi feldolgozás érdekében. A tárolásra a mágneses háttértárolókat használhatjuk. Az állományok típusai megegyeznek az őt tartalmazó adatelemek vagy logikai adatcsoportok típusaival. Ebben a témakörben mi csak a szöveges típusú állományokkal foglalkozunk.

1. Szöveges állományok szerkezete, deklarációja

Minden szöveges állomány ASCII karakterek sorozatából áll.

Az állomány nagyobb egysége a sor. Minden sor végét a már ismertetett vezérlőkarakterek (#13#10) Carriage Return And Line Feed jelzik.

Az állomány végét szintén egy karakter jelzi mégpedig a <CTRL>+<Z>. Ennek a kódja: #26

A Pascal az állományok(file) esetében két azonosítót különböztet meg:

- **Logikai filenév**

  Ez a programban használt azonosító. A programban az állományra minden művelet esetén ezzel az azonosítóval hivatkozunk. Ezt a deklarációs részben adjuk meg.

  - Szöveges állományok deklarációja a Var alapszóval történik

    Var logikai_azonosító:Text;

    pl.: Var dolgozat:Text;

- **Fizikai filenév**

  Ennek segítségével történik az állomány tárolása a háttértárolókon. Ez tartalmazhatja a könyvtárszerkezetet, az elérési utat is.

  pl.: c:\Winword\doga.txt

- **logikai és fizikai filenév egymáshoz rendelése**

  Ahhoz, hogy a Pascal az állományokkal műveleteket tudjon végezni, a fizikai névhez hozzá kell rendelni egy logikai azonosítót. Ezután a fel-

dolgozás után már csak ezt a nevet kell használni. **Erre az Assign eljárást használjuk.** A fizikai filenévnek string típusú adatnak kell lennie.

Assign(logikai azonosító, fizikai azonosító)

pl.: Assign(dolgozat,'c:\Winword\doga.txt')

2. Állományok létrehozása, írása

Nézzük a következő példát:

Input adatunk billentyűzetről a file fizikai neve. Írjunk az állomány minden sorába szövegesen **sorszámneveket** *és utána a '* **sora az állománynak'** *karakterláncot.*

Ehhez a következő ismeretekre van szükségünk:

1. Új szövegfile létrehozása a **Rewrite** eljárással történik:

   Rewrite(fileváltozó) **pl.: Rewrite(dolgozat)**

2. Írás állományba: Ez a már ismertetett Write és WriteLn eljárásokkal történik, amelyeket eddig csak egy paraméterrel használtunk. Most kiegészítjük még egy paraméterrel, a logikai filenévvel. Az egyik sorfolytonosan a másik soronként ír az állományba.

   Write(fileváltozó, változó) Write(dolgozat, jegyek)

   WriteLn(fileváltozó, változó) Write(dolgozat,tantargy)

3. Az állománnyal végzett műveletek a program futás közben a memóriába történnek. A program befejezésekor az állományokat le kell zárni. Ez a **Close** eljárással történik.

   Close*(fileváltozó)* **Close***(dolgozat)*

Ennek alapján tekintsük a kész programot:

```
Program letrehoz;
Uses Crt;
var
   nev : String;
   f : Text;
   sor : Integer;
```

```
Const
    szamstr : Array[1..7] of String=('Első','Második',
    'Harmadik','Negyedik', 'Ötödik','Hatodik','Hetedik');
    mondat=' sora az állománynak!';
    Procedure ir;
        Begin
            For sor:=1 To 7 Do
            Begin
            WriteLn(f,szamstr[sor]+mondat);
            End;
        End;
Begin
    Clrscr;
    Write('Kérem a file nevét: ');
    ReadLn(nev);
    Assign(f,nev);
    Rewrite(f);
    ir;
    Close(f);
End.
```

Programunkban az input adatokat egy tipizált konstans tömbbe tároljuk. A file fizikai nevét billentyűzetről kéri a program. Az ir eljárás végzi a file műveletet, nevezetesen a tömb elemeinek és konstans string írását soronként. Ezt egy For ciklussal végezzük el.

☝ A Rewrite eljárás a már létező file tartalmát törli!

3. Olvasás állományokból

Írjuk ki az előbb létrehozott állományunkat a monitorra, soronként

- **File megnyitása olvasásra**

  A művelet a **Reset** eljárással történik

  Reset(fileváltozó) **Reset(dolgozat)**

- **Olvasás állományból soronként**

  A már ismertetett **ReadLn** eljárással történik:

  ReadLn(fileváltozó, Stringváltozó) **ReadLn(f,sor)**

Ha string paramétert nem írunk, akkor egy "üres sort" olvas és az állomány mutatója a következő sor elejére áll. Ez azt jelenti, hogy a következő olvasás innen kezdődik.

ReadLn(fileváltozó) **ReadLn(f)**

Ha a második paraméter karakterváltozó, akkor csak a sor első karakterét olvassa be. **ReadLn(fileváltozó,karakterváltozó)**

- **File vége jel, olvasás befejezése**

  Az olvasás befejezését csak akkor tudjuk végrehajtani, ha értesülünk a file végéről. A már említett Close eljárás a file végére egy #26 karaktert tesz a file végére. A probléma ennek segítségével is lekezelhető. Szerencsére az **Eof** függvény segít a probléma megoldásában. Ha ennek értéke True, akkor ez a file végét jelzi. **Eof(fileváltozó)**

Az eddigiek alapján programunk:

```
Program olvas1;
Uses Crt;
var
   nev, sor : String;
   f : Text;
     Procedure monitor;
     Begin
        WriteLn;
        While Not Eof(f) Do
        Begin
        ReadLn(f,sor);
        WriteLn(sor);
        End;
     End;
Begin
   Clrscr;
   Write('Kérem a file nevét: ');
   ReadLn(nev);
   Assign(f,nev);
   Reset(f);
   monitor;
   Close(f);
   ReadLn;
End.
```

Az olvasást és a képernyőre való kiíratást a monitor eljárás végzi. Ennek legfontosabb része, hogy **a műveletet a file vége jelre abba kell hagyni**. Ez egy elöl tesztelő ciklussal oldható meg.

ciklus **amíg nincs állomány vége**
    olvasás az állományból
    kiírás a monitorra
ciklus vége

Erre a megoldásra a While ciklus kínálkozik:

```
While Not Eof(file változó) Do
  Begin
    műveletek
  End;
```

4. Megnyitás hozzáírásra

Nyissuk meg a már előzőleg létrehozott állományt! Egy tipizált konstans tömb segítségével fűzzünk a végéhez még hat sort!

A szöveges állományok esetében az állományba való beszúrás nehézkes. A file mutatóval nem tudunk tetszőlegesen mozogni. A mozgást csak soronként tudjuk végrehajtani, egy paraméteres ReadLn eljárással. Beszúrás csak egy másik állományba való átmozgatással és visszamozgatásával valósítható meg.

Programunkban az új eljárás az **Append(fileváltozó).** Ez a megnyitás után a file mutatót az állomány végére helyezi. Innen következhet az írás művelete.

```
Program fuz;
Uses Crt;
var
   nev : String;
   f : Text;
   sor : Integer;
Const
   szamstr : Array[1..6] of String=('Nyolcadik','Kilencedik',
   'Tizedik','Tizenegyedik','Tizenkettedik','Tizenharmadik');
mondat =' sora az állománynak!';
Procedure hozzair;
   Begin
```

```
        For sor:=1 To 6 Do
        Begin
        WriteLn(f,szamstr[sor]+mondat);
        End;
    End;
Begin
Clrscr;
    Write('Kérem a file nevét: ');
    ReadIn(nev);
    Assign(f,nev);
    Append(f);
    hozzair;
    Close(f);
End.
```

5. Olvasás karakterenként

Folytassuk a file kezelést a legutoljára létrehozott állományunkkal. Ha ezeket a programokat egymás után futtatod, akkor ellenőrizni tudod mindegyik működését.

Nyissuk meg az állományt olvasásra! Írassuk ki a monitorra az állomány minden sorának első szavát! A szavakat egy szóköz karakter választja el.

- **Olvasás karakterenként a Read eljárással történik:**
  Read(fileváltozó,karakterváltozó)
- **Soremelés a ReadLn eljárással**
  Ilyenkor az eljárásnak csak egy paramétere van, nevezetesen az olvasandó állomány logikai azonosítója
  ReadLn(fileváltozó)

A megoldás lehetne az is, hogy soronként olvasunk egy string változóba. Ezután leválasztjuk a string karaktereit a legközelebbi szóköz karakterig.
A feladatot másképpen oldjuk meg.

```
Program olvas2;
Uses Crt;
```

```
var
    nev, szo : String;
    f : Text;
    betu : Char;
    Procedure monitor;
        Begin
          WriteLn;
          While Not Eof(f) Do
            Begin
              szo:= '';
              betu:=#0;
                While betu<>' ' Do
                  Begin
                    Read(f,betu);
                    szo := szo+betu;
                  End;
              WriteLn(szo);
              ReadLn(f);
            End;
        End;
Begin
    Clrscr;
    Write('Kérem a file nevét: ');
    ReadLn(nev);
    Assign(f,nev);
    Reset(f);
    monitor;
    Close(f);
    ReadLn;
End.
```

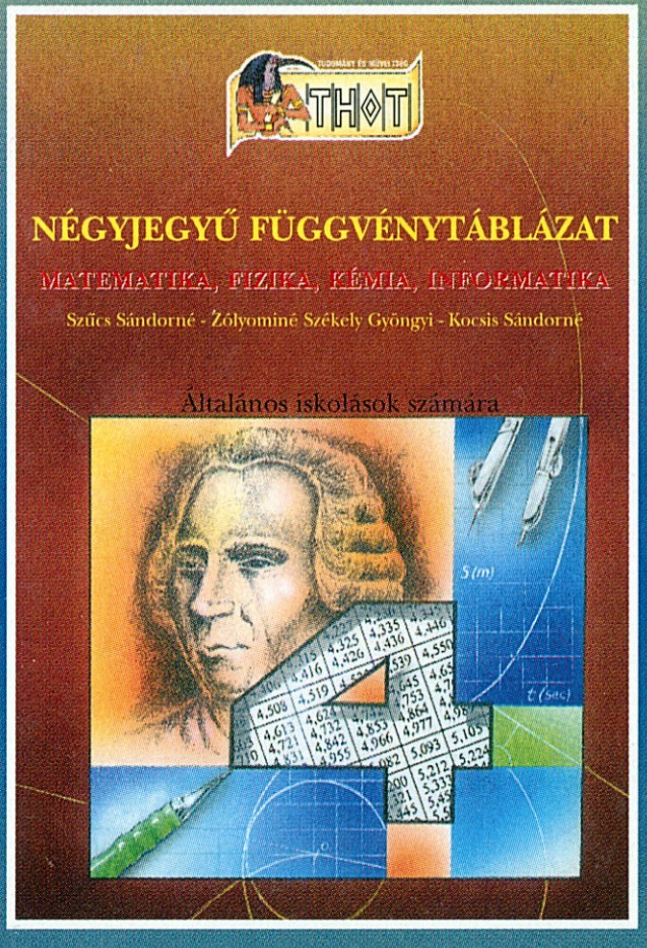
Az állományból karakterenként olvassunk a legközelebbi szóközig. Közben hozzáfűzzük a karaktereket egy string változóhoz. Kiíratjuk a string változót a monitorra. Ha ez megtörtént, akkor az olvasást egy új sorban kezdjük. Az egész eljárást folytatjuk File vége jelig. Programunkban ezt a monitor eljárás végzi.

A probléma megoldása két egymásba ágyazott While ciklus.

```
Ciklus amíg nincs file vége jel
    a string változó tartalmának törlése
    a karakterváltozó tartalmának törlése
    ciklus amíg a karakter nem szóköz
        karakter beolvasása az állományból
        karakterek összefűzése a string változóba
    ciklus vége
    a string változó kiíratása
    új sor kezdése
ciklus vége
```

TUDOMÁNY ÉS MŰVELTSÉG

THOT



9 789639 081376

TÓTH KÖNYVKERESKEDÉS ÉS KIADÓ KFT. 4034 Debrecen, Huszár Gál u. 31-33.
Tel.: (06-52)-450-861; (06-30) 358-569