Ukrajna Oktatási és Tudományügyi Minisztériuma

Ungvári Nemzeti Egyetem

Ukrán—Magyar Oktatási-Tudományos Intézet

Fizika és Matematika Tanszék

Petki Katalin, Traski Viktor, Traski Natália

Informatika és programozás I

Ungvár 2019

UDC 004.43(076)
ВВК В183.4я73
П-29

RECENZENSEK:
Dr. Gecse Ferenc *professzor, a Kibernetika és Alkalmazott Matematika Tanszék vezetője*
Dr. Szlivka-Tiliscsak Hanna *docens,* V*alószinüségelmélet és Matematikai Analizis Tanszék*

Kiadását ajánlotta:

- Fizika és Matematika Tanszék
  (2019. szeptember 19-i ülésén, 2.sz. jegyzőkönyv)
- Ukrán—Magyar Oktatási-Tudományos Intézet Tudományos Tanácsa
  (2019. szeptember 24-i ülésén, 1.sz. jegyzőkönyv)

Petki K., Traski V., Traski N.
П-29 Informatika és programozás I: - Ungvár: «AUTDOR-Shark», 2019. – 99o.

Міністерство освіти та науки України

ДВНЗ «Ужгородський національний університет»

Українсько-угорський навчально-науковий інститут

Кафедра фізико-математичних дисциплін

К.П. Петкі, В.Б. Трошкі, Н.В.Трошкі

Інформатика та програмування І

Посібник

Ужгород 2019

УДК 004.43(076)
ББК В183.4я73
П-29

РЕЦЕНЗЕНТИ:

Гече Федір Елемирович – *доктор технічних наук, професор, зав. кафедрою кібернетики та прикладної математики.*

Сливка-Тилищак Ганна Іванівна – *доктор фіз.-мат. наук, доцент, доцент кафедри теорії ймовірностей та математичного аналізу.*

Рекомендовано до друку:

Кафедрою фізико-математичних дисциплін
(протокол №2 від 19 вересня 2019р.)
Вченою радою Українсько-угорського навчально-наукового інституту
(протокол №1 від 24 вересня 2019р.)

К.П. Петкі, В.Б. Трошкі, Н.В.Трошкі
П-29 Інформатика та програмування І: Посібник - Ужгород: «АУТДОР-Шарк», 2019. – 89с.

Tartalom

Bevezetés

Ez a jegyzet a Ukrán-Magyar Oktatási-Tudományos Intézet elsőéves matematika szakos hallgatók számára készült és a szak tematikáját követi. Mindegyik témához fel vannak sorolva a főbb elméleti tudnivalók az önálló felkészüléshez, példák a feladatok megoldására és az önálló munkához ajánlott feladatsor husz változata.

Az jegyzet célja: segítséget nyújtani a matematikus diákok számára az önálló munkában a "Informatika és programozás" tanulmányozása során és a megfelelő modul dolgozatok, szigorlatok és vizsgák letételének felkészülésében.

A Pascal programozási nyelv alapjait Niklaus Wirth definiálta 1971-ben. A Pascal magas szintű, általános célú, struktúrált programnyelv, az Algol és a Fortran legjobb elemeit egyesíti. Szigorú nyelv, egyszerű eszközrendszerrel, szintaktikai és szemantikai szabályokkal. Az oktatás kedvelt nyelve. A Standard Pascal kibővített mikrogépes (elsősorban IBM PC) változata a Borland cég által kifejlesztett Turbo Pascal [1].

A Pascal szabad formátumú nyelv, azaz a külalaknak (pl. sorhosszúság, bekezdések) csak a program áttekinthetősége szempontjából van jelentősége. A nyelv nem különbözteti meg a kis és nagybetűket.

Értékadás, olvasás a billentyűzetről, kiírás a képernyőre

Értékadó utasítás

változó := *kifejezés*
függvény azonosító := *kifejezés* ld. xx fejezet

Példa:

```
x := 4 * ( x + y ) + sin(alfa);
```

A kifejezés kiértékelése során előálló érték kerül a változó címére a memóriába. A kifejezések formálisan operandusokból, műveleti jelekből (operátorok) és kerek zárójeleből épülnek fel. Operandusok konstansok (pl. 2, 'szöveg'), változók (pl. x, alfa) és függvényhívások (pl. sin(alfa)) lehetnek [1].

Kifejezések kiértékelése:

- Először a zárójelben szereplő kifejezések kerülnek kiértékelésre.
- A műveletek kiértékelésének a sorrendjét a precedenciájuk szabja meg. A precedencia szintek:

1. NOT, +, -, @ (egy operandusú műveletek)
2. *, /, DIV, MOD, AND, SHL, SHR
3. +, -, OR, XOR
4. <, >, <=, >=, <>, =, IN

- Azonos prioritás esetén a balról jobbra szabály lép érvénybe, amelyet a fordító felülbírálhat az optimális kód készítése érdekében [2].

Olvasás a billentyűzetről

A Pascal nyelvben nincs input utasítás, a billentyűzetről a változókba a Read és a ReadLn eljárások segítségével olvashatunk be.

Read(v1 [,v2...])

ReadLn(v1 [,v2...])

A változók numerikus, karaktes és string (karakterlánc) típusúak lehetnek. A részletesebb leírást ld. szöveges állományok.

Példa:

```
ReadLn(fizetes);
```

A program ennél a sornál megáll, a leütött karakterek a képernyőn is megjelennek, és az Enter leütéséig szerkeszthetők. Majd a változóba

bekerül a megfelelő típusú érték. Ha a beírtak nem felelnek meg a változó típusának, akkor a program futási hibával leáll (I/O hiba lép fel).

Karakteres, karakterlánc típusú változók olvasásakor a Read használata kellemetlen félreértéseket okozhat, használjuk a ReadLn eljárást. A Read eljárás a billentyűzet pufferből kiolvassa a változónak megfelelő értéket, de nem törli a puffert, míg a ReadLn eljárás olvasás után törli a puffert. Próbáljuk ki a következő programrészletet:

```
Read(a);
Read(b);
Read(c);
WriteLn(a);
WriteLn(b);
WriteLn(c);
```

Kiírás a képernyőre

A Pascal nyelvben nincs output utasítás, a képernyőre a Write és a WriteLn eljárások segítségével írhatunk.

Write(k1 [,k2...])

WriteLn(k1 [,k2...])

Az eljárások az aktuális pozíciótól kezdődően kiírják a kifejezések értékeit. A WriteLn eljárás ezután sort emel. A kifejezések numerikus, karaktes és string (karakterlánc) és logikai típusúak lehetnek. A kiírást módosíthatjuk, mezőszélességet illetve valós kifejezés esetén a tizedesjegyek számát adhatjuk meg: Write(k[:MezSzel[:Tized]]). A mezőben jobbra igazítva, illetve a megfelelő számú tizedesjegyre kerekítva jelenik meg a kifejezés értéke [3].

Példa:

```
        Write('A dolgozó fizetése: ',
fizetes:10);
```

A Pascal program felépítése

Egy Pascal programot három részre oszthatunk:

1. Programfej
2. Programblokk - deklarációs rész (vagy: leíró rész)
3. Programblokk - végrehajtandó rész (vagy: programtörzs)

Programfej:

[**PROGRAM** *azonosító***;**]
[**USES** *azonosító* [**,***azonosító*...]**;**]

A PROGRAM fenntartott[1] szó után álló azonosító[2] lesz a programunk neve. Ez célszerűen megegyezhet a forrásprogram[3], a lemezen tárolt PAS kiterjesztésű állomány nevével. Mindez elhagyható, de használata jaánlott [3].

A USES kulcsszó után a programunk által használt egységeket soroljuk fel. A System egység, amely a leggyakrabban használt deklarációkat - konstansokat, változókat, eljárásokat, függvényeket - tartalmazza automatikusan hozzászerkesztődik a programunkhoz[4].

Deklarációs rész:

---

1 Fenntartott szó, kulcsszó. A programnyelv tulajdonít neki értelmet, amit nem lehet megváltoztatni, azaz ezeket az azonosítókat másra nem használhatjuk A Turbo Pascal kulcsszavai: AND, ASM,ARRAY, BEGIN, CASE, CONST, CONSTRUCTOR, DESTRUCTOR, DIV, END, FILE, FOR, FUNCTION, GOTO, IF, IMPLEMENTATION, IN, INLINE, NIL, NOT, OBJECT, OF, OR, PACKED, PROCEDURE, PROGRAM, RECORD, SHR, STRING, THEN, TO, TYPE, UNIT, UNTIL, USES, VAR.

2 Azonosító. A program egyes objektumait azonosítja (program, egység, eljárás, függvény, címke, konstans, típus, változó, rekordmező...). Kis- és nagybetűből, számjegyből és aláhúzásjelből állhat, de számjeggyel nem kezdődhet. A rendszer az első 63 karaktert különbözteti meg.

3 Forrásprogram. Egy magasszintű programnyelven megírt szöveg. A Turbo Pascal rendszerben kiterjesztése .pas. A forrásprogram fordítása után kapjuk a tárgykódot. A tárgykódú modulakat a szerkesztő (linker) állítja össze futtatható programmá. A Turbo Pascal renszerben a szerkesztés automatikus.

[**LABEL** *címke* [**,***címke*...]**;**]
[**TYPE** *azonosító* **=** *típus***;**...]
[**CONST** *azonosító* [**:** *típus*] **=** *konstans***;**...]
[**VAR** *azonosító* **:** *típus***;**...]
[**PROCEDURE** *azonosító* [**(***formális paraméterlista***)];**
eljárásblokk]
[**FUNCTION** *azonosító* [**(***formális paraméterlista***)**] **:** *típusazonosító***;**
függvényblokk]

A későbbiek során részletesebben tárgyaljuk a deklarációs rész egyes elemeit.

A **Var** kulcsszó után álló változódeklarációs szakaszban a programblokkban használt összes változót fel kell sorolni, és típusát megadni. A típusmegadás történhet áttételesen is, a **Type** utáni típusdeklaráció segítségével.

A konstansok[4] használata programozói munkánkat könnyítheti meg (**Const**).

Címkék[5] igen ritkán fordulnak elő egy Pascal programban (**Label**).

Végrehajtandó rész:

BEGIN
[*utasítás* [**;** *utasítás*...]]
END.

A Pascal szabad formátumú nyelv, azaz több utasítás is szerepelhet egy sorban vagy akár egy utasítást több sorra is tördelhetünk. Célszerű a

---

4 Konstansok.

Nevesített konstans: olyan programozói objektum, melynek neve címe, típusa, értéke van. Értéke állandó, a programban nem lehet megváltoztatni.

Pl.: const max = 100;

Tipizált konstans: gyakorlatilag egy változó, amelynek kezdőértéket adunk. Deklarálása a Const kulcsszó után. (Az alprogramok statikus lokális változói lehetnek, amelyek két hívás között nem veszítik el az értéküket).

Pl.: const max: integer = 100;

Konstans: Nics neve, címe. Értéke állandó.

Pl.: 100, -12.32, 'ez egy karakterlánc', ['a'..'z','.','?']

5 Címke. Egy utasítást jelölhetünk meg vele, és egy másik utasításban a címkére, azaz a megjelölt utasításra hivatkozhatunk

program olvashatóságára, áttekinthetőségére törekedni. Az egyes utasításokat **;** -vel választjuk el egymástól. (Szemben pl. a C nyelvvel, ahol minden utasítás végére ; -t kell írnunk.) [1]

A Pascal nyelv nem különbözteti meg a kis és nagy betűket (szemben a C nyelvvel).

Programunkban korlátlan hosszúságú megjegyzést[6] helyezhetünk el a { ... } illetve a (* ... *) jelek között.

A program szövegében használhatunk:

- fenntartott szavakat(pl. Program, Begin, If, Var ...)
- standard azonosítókat[7] (pl Sin, Integer...)
- azonosítókat.

6 Megjegyzés. A forrásprogram olvasójának szól, a fordító figyelmen kívűl hagyja. A program egyes részeit magyarázhatjuk, a későbbi könnyebb érthetőség érdekében.

7 Standard azonosító. A programnyelv tulajdonít neki értelmet, de azt megváltoztathatjuk, átdefiniálhatjuk. Célszerű meghagyni az eredeti funkcióját. Ilyen például: integer, sin, abs...

Egyszerű adattípusok

Egész típusok

Egy típus jellemzésénél az alábbiakat kell figyelembe venni:

a. felvehető értékek halmaza (adatábrázolás)
b. konstansai
c. végezhető műveletek
d. szabványos eljárások, függvények

a. A Pascal nyelvben bőséges választék áll a rendelkezésünkre egész típusokból

Típus	*Értéktartomány*	*Tárolás*
Byte	0..255	1 byte
ShortInt	-128..127	1 byte
Word	0..65535	2 byte
Integer	-32768..32767	2 byte
LongInt	$-2*10^9..2*10^9$	4 byte

A Byte és a Word típus esetén 8 illetve 16 biten $2^8 = 256$ (00000000-tól 11111111-ig) illetve $2^{16} = 65536$ különböző számot ábrázolhatunk kettes számrendszerben. A ShortInt, az Integer és a LongInt típusokban negatív számokat is tárolhatunk. Itt az ábrázolási tartomány egyik fele kettes komplemens kódolással[8] negatív számokat jelent [4].

b. Egész típusú konstans

Decimális egészek, pl. 25, -123
Hexadecimális egészek, pl. $33, -$A2D6

c. Végezhető műveletek

Az eredmény is egész típusú:

+,- (előjel), *, +, -
div egész osztás, pl. 13 div 3 = 4

---

8 Kettes komplemens kódolás. Negatív egész számok ábrázolására használják. Így a kivonás visszavezethető egy kettes komplemens képzésre (ami igen egyszerű) és egy összeadásra.
Egy b (egy bájtos) bináris szám kettes komplemense: (11111111 - b) + 1. A zárójelben lévő rész a b egyes komplemense.

mod maradékképzés, pl. 13 mod 3 = 1
not bitenkénti negálás, pl not 28 = -29 ; (not 00011100 = 11100011)
and bitenkénti *és*, pl. 5 and 6 = 4; (00000101 and 00000110 = 00000100)
or bitenkénti *vagy*, pl. 5 or 6 = 7; (00000101 or 00000110 = 00000111)
xor bitenkénti *kizáró vagy*, pl. 5 xor 6 = 3; (00000101 and 00000110 = 00000011)
shl bitenkénti eltolás balra, pl. 5 shl 3= 40; (00000101 shl 3 = 00101000)
shr bitenkénti eltolás jobbra, pl. 5 shr 2= 1; (00000101 shr 2= 00000001)

Az eredmény kivezet az egész számok köréből:

/ az eredmény mindig valós (6/2-t már nem egész számként kezeli a rendszer)
<, >, <=, >=, =, <> relációs műveletek, az eredmény logikai típusú
in halmaz elemvizsgálat, logikai eredmény

d. Fontosabb szabványos eljárások, függvények

Függvények:

Abs - visszatérési értéke, az argumentummal azonos típusú, az argumentum abszolút értéke.
Sqr - visszatérési értéke, az argumentummal azonos típusú, az argumentum négyzete.
Trunc - egy valós értéket egésszé csonkít a törtrész levágásával.
Round - egy valós értéket egésszé kerekít.
Ord - egy sorszámozott típusú kifejezés sorszámával tér vissza.
Pred - a paraméterét megelőző értéket adja vissza.
Succ - a paraméterét követő értéket adja vissza.
Random - egy véletlenszámot állít elő.

Eljárások:

Inc - egy változót növel.
Dec – egy változót csökkent
Str - egy numerikus értéket karakterlánccá konvertál.
Val - egy karakterláncot numerikus értéké konvertál.
Randomize - inicializálja a beépített véletlenszám generátort.

Megj.

A függvények mindig egy értéket állítanak elő (visszatérési érték), kifejezésekben hívhatjuk meg őket, pl. egy értékadó utasítás jobb oldalán; a := Abs(a) + 2. A függvények paraméteri kifejezések lehetnek, pl. Sqr(a + Round(x)).
Az eljárásokat utasításszerűen hívjuk, pl. Inc(i).

Példaprogram [1]

```
program Oszt;
uses Crt;                                {A
képernyő törlést tartalmazó unit}
var Osztando, Hanyados, Maradék: byte;
begin
  ClrScr;                                {Képernyő
törlés}
  Write('Kérem az osztandót: ');
  ReadLn(Osztando);
  Hanyados := Osztando div 3;
  Maradek := Osztando mod 3;
  WriteLn('A hányados: ', Hanyados);
  WriteLn('A maradék: ', Maradek);
  ReadLn
end.
```

Valós típusok

a. Értéktartomány, számábrázolás

A Turbo Pascalban a következő valós típusokat definiálták: Real (6 byte), Single (4 byte), Double (8 byte), Extended (10 byte), Comp (8 byte), azonban a Real típus kivételével mindegyik használatához matematikai társprocesszorra, vagy annak emulálására van szükség [1].

A Real típus ábrázolása 6 bájton, lebegőpontosan történik. Az értéktartomány:

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

A pontosság 11-12 decimális számjegy. (Ennyi értékes számjegye egy valós számnak, a többi jegyet nem ábrázolja a rendszer, pl. a 1234567890,1234567 számból a színes jegyek elvesznek.)

b. Konstansok

Pl. 5.12, -123.2313, 12.54E6, 21.12E-5

c. Műveletek

Az eredmény is valós típusú: +,- (előjel), *, /, +, -
Az eredmény logikai típusú: <, >, <=, >=, =, <>

d. Fontosabb szabványos eljárások, függvények

Függvények: **Abs, Sqr, Random, Round, Trunc**,

Sqrt – Visszatérési értéke az argumentum négyzetgyöke.
Sin – Visszatérési értéke az argumentum szinusza.
Cos – Visszatérési értéke az argumentum koszinusza.
ArcTan – Visszatérési értéke az argumentum arkusz tangense.
Exp – Visszatérési értéke: e^x, ahol *e* a természetes logaritmus alapja.
Ln – Visszatérési értéke az argumentum természetes alapú logaritmusa.
Int – Visszatérési értéke az argumentum egészrésze.
Frac – Visszatérési értéke az argumentum törtrésze.
Pi – A π értékével tér vissza.

Eljárások: **Str, Val, Randomize**

Példaprogram [1]

```
program Valos_Pelda;
uses Crt;
var alap, kitevo, hatvany: real;
begin
  ClrScr;
  Write('Kérem a hatvány alapját: ');
  ReadLn(alap);
  Write('Kérem a hatvány kitevőjét: ');
  ReadLn(kitevo);
  hatvany := Exp(kitevo * Ln(alap)); {ld.
logaritmus definiciója, logaritmus azonosságok}
  WriteLn('A hatvány értéke: ', hatvany:10:2);
  ReadLn
end.
```

A karakteres - Char típus

a. Értéktartomány, adatábrázolás

Egy bájtos típus, tehát 2^8 = 256 különböző érték , az ASCII kódrendszer 256 elemének a tárolására képes. A karakter típusú változó egy ASCII kódot tartalmaz. Pl. ha a változóhoz tartozó memóriarekesz értéke 65, akkor mivel változónk típusa Char, ezt a rendszer 'A' betűként értelmezi [3].

b. Konstansok

Formája: 'A', '*', '4', #65 (ez utóbbi a 65-ös ASCII kodú karaktert, azaz 'A'-t jelenti).
Lehetnek: betűk, számjegyek, írásjelek, speciális karakterek (pl. '@', '#', '$', ...), vezérlő karakterek (pl. a gyakran használt #27 - Escape), egyéb karakterek (az ASCII kódtábla 128-255 része, pl. 'é').

c. Műveletek

Relációs műveletek: <, >, <=, >=, =, <> (az eredmény természetesen logikai típusú, a karakter ASCII kódja határozza meg. Pl. 'A' < 'B' igaz, 'A' = 'a' hamis.) **in** halmaz elemvizsgálat, logikai eredmény (ld. halmaz adattípus)

d. Fontosabb szabványos eljárások, függvények

Függvények: Ord, Chr, UpCase, Pred, Succ
Eljárások: Inc, Dec

Példaprogram [1]

```
program Kisbetu;
uses Crt;
var Nagy, Kis: char;
begin
  ClrScr;
  Write('Kérek egy nagybetűt: ');
  ReadLn(Nagy);
  Kis := Chr( Ord(Nagy) + 32 );
        {egy nagybetűt kisbetűvé konvertál}
  WriteLn(Kis);
  ReadLn
end.
```

A logikai - Boolean típus

a. Értéktartomány, adatábrázolás

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

b. Konstansok

Két előredefiniált konstans: *True* (igaz), *False* (hamis)

c. Műveletek

Az eddig tanult típusokra értelmezett relációs operátorok (valamint az in halmazművelet) mindig logikai értéket állítanak elő [5].

Logikai műveletek: **not**, **and**, **or**, **xor** (az operandusok logikai típusúak). A műveletek igazságtáblái:

NOT		**AND**			**OR**			**XOR**		
A	**not** A	A	B	A **and** B	A	B	A **or** B	A	B	A **xor** B
False	*True*	*False*	*False*	*False*	*False*	*False*	*False*	*False*	*False*	*False*
True	*False*	*False*	*True*	*False*	*False*	*True*	*True*	*False*	*True*	*True*
		True	*False*	*False*	*True*	*False*	*True*	*True*	*False*	*True*
		True	*True*	*True*	*True*	*True*	*True*	*True*	*True*	*False*

Pl. egy logikai kifejezés: (x > 4) and (x < 10) or not b , ahol *b* egy Boolean változó.

d. Logikai értéket előállító függvények

Odd - az Odd (X) függvény értéke True, ha X páratlan szám.

Eof – ha az állomány mutató a fájl végén (az utolsó elem mögött) áll, akkor az értéke True.

Eoln - ha az állomány mutató a sor végén áll, akkor az értéke True.

Példaprogram [1]

```
program Logikai;
uses Crt;
var b: boolean;
    a: byte;
begin
  ClrScr;
  Write('Kérek egy számot: ');
  ReadLn(a);
  b := a mod 3 = 0;
  if b then
    WriteLn(a,' osztható 3-mal)
  else
    WriteLn(a,' nem osztható 3-mal);
  ReadLn
end.
```

Felsorolt típus

a. Értékek, adatábrázolás

A típus megadásakor fel kell sorolnom a lehetséges értékeit. Ezek csak azonosítók lehetnek.

A konstansok a felsorolás sorrendjében sorszámot kapnak 0-tól kezdődően. Tarolás a konstansok számától függően 1 vagy 2 bájton történik. [8]

Pl.

var tantargy: (magyar, matek, fizika, tesi);

...

tantargy := matek;

b. Konstansok

Konstansai a felsorolásban szereplő azonosítók.

c. Műveletek

Relációs műveletek:<, >, <=, >=, =, <> (az eredmény természetesen logikai típusú, a felsorolt típusú érték sorszáma határozza meg. Példánkban: *matek* < *tesi* igaz.) **in** halmaz elemvizsgálat, logikai eredmény (ld. halmaz adattípus)

d. Függvények, eljárások

A sorszámozás alapján értelmezhetőek az alábbi
függvények:**Ord, Pred, Succ**;
eljárások: **Inc, Dec**.

Intervallum típus

Egy már létező sorszámozott típus egy intervalluma. Szintén nekünk kell definiálnunk.

Az adatábrázolás, műveletek, függvények, eljárások megegyeznek az eredeti típuséval [11].

Pl.

```
  var  nap: (hetfo, kedd, szerda, csutortok,
pentek, szombat, vasarnap); {felsorolt típus}
       munkanap: hetfo..pentek;
       betu: 'A'..'Z';
       szamjegy: '0'..'9';
```

A felsorolt típus használata a program tesztelésekor hasznos lehet. (A fenti példában *szamjegy := 'e'* fordítási hibát, *szamjegy* beolvasásakor az 'e' karakter pedig futási hibát eredményez ($R+ fordítási direktíva esetén).) [1]

Az egyszerű típusok csoportosítása

- Sorszámozott típusok
  - egészek
  - logikai
  - karakter
  - felsorolt
  - intervallum
- Valós típusok

Sorszámozott típusok: minden lehetséges értékhez hozzárendelhető egy sorszám, az értékek a sorszám szerint rendezettek.

Ciklusok - iterációk

Segítségével utasítás(ok) ismételten végrehajható(k). Részei az utasítást (utasításokat) tartalmazó ciklusmag és az ismételt folytatást vagy befejezést vezérlő rész.

A Pascal nyelv három ciklust definiál: két feltételes ciklust valamint egy előírt lépésszámú (más néven számláló vagy léptető) ciklust [12].

A WHILE ciklus

Az utasítás szintaktikája:

WHILE *feltétel* **DO** *utasítás*

Kezdőfeltételes ciklus. Amíg a feltétel igaz, addig ismétli az utasítást, ha hamis, akkor a program következő utasítására ugrik. A feltétel egy logikai (boolean) kifejezés. Ha több utasítás szerepel a ciklus magjában, akkor **Begin** - **End** ún. utasítás zárójelet kell alkalmaznunk. Ha a feltétel már először sem teljesül, akkor a ciklusmag egyszer sem kerül végrehajtásra (üres ciklus), ha pedig a feltétel soha nem vesz fel hamis értéket, akkor a program végtelen ciklusba kerül.

Tipikus használata: a ciklus végrehajtása attól függ, hogy van-e még feldolgozandó adat, az ismétlések számát előre nem ismerjük, akár 0 is lehet [5].

Példa: Képezzük a billentyűzetről érkező pozitív számok összegét, a számsorozat végét a 0 vagy egy negatív szám jelezze [1].

Megoldás

Az ilyen típusú feladatok általános megoldási sémája:

- *az első adat előállítása (pl. beolvasása)*
- *ciklusfej*
- *az adat feldolgozása*
- *a következő adat előállítása (pl. beolvasása)*
- *ciklus vége*

```
program Osszeg_szamitas;
uses Crt;
var adat, osszeg: integer;
begin
  ClrScr;
  osszeg := 0;
  ReadLn(adat);
  while adat > 0 do
```

```
    begin
      osszeg := osszeg + adat;
      ReadLn(adat)
    end;
  WriteLn('Az összeg: ',osszeg);
  ReadLn
end.
```

A REPEAT ciklus

Az utasítás szintaktikája:

REPEAT [*utasítás* [**;** *utasítás*...]] **UNTIL** *feltétel*

Végfeltételes ciklus. Az utasítás(ok) végrehajtását meg kell ismételni, ha a feltétel hamis. Ha a feltétel igaz, a program a ciklus utáni utasítással folytatódik.

A ciklusmag legalább egyszer végrehajtódik. A ciklusmagot a **Repeat** - **Until** kulcsszavak fogják közre, nem kell **Begin** - **End** utasítás zárójelet használnunk.

A While és a Repeat ciklust hasonló típusú feladatok megoldására használhatjuk. Esetleg az egyik egy kicsit kényelmesebb megoldást nyújt.

Az előző példa megoldása Repeat-Until ciklus alkalmazásával [1].

```
program Osszeg_szamitas;
uses Crt;
var adat, osszeg: integer;
begin
  ClrScr;
  osszeg := 0;
  ReadLn(adat);
  repeat
    osszeg := osszeg + adat;
    ReadLn(adat)
  until adat <= 0;
  WriteLn('Az összeg: ',osszeg);
  ReadLn
end.
```

A FOR ciklus

Az utasitás szintaktikája:

FOR *ciklusváltozó* **:=** *kezdőérték* **TO** / **DOWNTO** *végérték* **DO** *utasítás*

ahol a *ciklusváltozó* sorszámozott típusú változó hivatkozás, a *kezdőérték* és a *végérték* pedig sorszámozott típusú kifejezések.

A *ciklusváltozó* a *kezdőértéktől* a *végértékig* egyesével nő (**To**) vagy csökken (**DownTo**). Az *utasítás* (vagy a **Begin** - **End** utasítás zárójelek közé zárt utasításcsoport) a ciklusváltozó minden értékénél végrehajtódik. Elöltesztelő ciklus, így ha a *kezdőérték* > *végérték* (**To** esetén) vagy a *kezdőérték* < *végérték* (**DownTo** esetén), akkor a ciklusmag egyszer sem kerül végrehajtásra (üres ciklus).

A ciklusváltozó értékét a ciklusmagban felhasználhatjuk, de nem változtathatjuk meg [5].

Tipikus használata: az ismétlések száma a ciklusba való belépés előtt már ismert vagy kiszámítható.

Példák [1]:

1. Írjunk ki N darab csillagot a képernyőre!

```
program Csillag;
uses Crt;
var i, n: byte;
begin
  ClrScr;
  ReadLn(n);
  for i := 1 to n do
    Write('*');
  ReadLn
end.
```

2. Számoljuk ki N! értékét! [1]

```
Program Faktorialis;
Uses Crt;
Var i, n: Byte;
    fakt: LongInt;
Begin
  ClrScr;
```

```
  ReadLn(n);
  fakt := 1;
  For i := 2 to n do
    fakt := fakt * i;
  WriteLn(n,'! = ',fakt);
  ReadLn
End.
```

Elágazások - szelekciók

A ciklus mellett a másik alapvető algoritmus elem az elágazás vagy szelekció, amelyben lehetőség van több tevékenység közül egyet kiválasztani.

Az IF utasítás

Az utasítás szintaktikája:
IF *feltétel* **THEN** *utasítás1* [**ELSE** *utasítás2*]
ahol a *feltétel* egy logikai kifejezés.

Ha a *feltétel* igaz, akkor az *utasítás1* hajtódik végre, egyébként az *utasítás2*. Az **Else** ág elhagyható, ilyenkor az *utasítás1* kimarad, a program a következő utasítással folytatódik. Egy ágon több utasítás is végrehajtható a **Begin** - **End** utasítás zárójelek alkalmazásával. Vigyázzunk, az **Else** előtt a pontosvessző szintaktikai hiba, mivel azzal lezárjuk a teljes **If** utasítást! [12]

Példa:

```
        ReadLn(Oszto);
        ReadLn(Osztando);
        if Oszto <> 0 then
          Hanyados := Osztando / Oszto
        else
          WriteLn('0-val való osztás, nincs
értelmezve.');
```

Akár az *utasítás1* vagy az *utasítás2* is lehet újabb **If** utasítás, és ezáltal többirányú elágazást is megvalósíthatunk.

If utasítások egymásba ágyazása [1], [6], [7]

```
if x > 0 then
  WriteLn('Pozitív)
else
```

```
  if x = 0 then
    WriteLn('Nulla')    {A három ág közül egy
hajtódik végre.}
  else
    Writeln('Negatív');
{A problémát az alábbi módon is kódolhatjuk,
 mivel a három feltétel közül egyszerre csak egy
áll fenn:}
if x > 0 then
  WriteLn('Pozitív);
if x = 0 then
  WriteLn('Nulla');
if x < 0 then
  Writeln('Negatív');
{Az alábbi viszont hibás megoldás:}
if x > 0 then
  WriteLn('Pozitív);
if x = 0 then
  WriteLn('Nulla')
else
  Writeln('Negatív');
{Vigyázzunk, az Else mindig a legutolsó Then
párja!}
if a > 0 then
  if Not Odd(a) then
    Writeln('Pozitív, páros')
  else
    Writeln('Pozitív, páratlan');
if a > 0 then
 begin
  if Not Odd(a) then
    Writeln('Pozitív, páros')
 end
  else
    Writeln('Nem pozitív');
```

A CASE utasítás

Az utasítás szintaktikája:

CASE *szelektor* **OF**
állandó [**..***állandó*] [,*állandó*[**..***állandó*]...] **:** *utasítás*;
[*állandó* [**..***állandó*] [,*állandó*[**..***állandó*]...] **:** *utasítás*;
[**ELSE** *utasítás*]
END

ahol a *szelektor* egy sorszámozott típusú kifejezés. Abban az ágban lévő *utasítás* (vagy **Begin** - **End** közé zárt utasításcsoport) hajtódik végre, ahol a szelektor értéke megegyezik az egyik *állandóval* vagy beleesik az egyik megadott tartományba. Ha ez egyik esetre sem teljesül, akkor az **Else** ágra kerül a vezérlés. Ez utóbbi elhagyható [4].

Példaprogram [1]

```
program CasePl;
uses Crt;
var  Kar: char;
begin
  ClrScr;
  ReadLn(kar);
  case Kar of
    'A'..'Z', 'a'..'z': WriteLn('betű');
    '0'..'9':           WriteLn('számjegy');
    '.', ',', '!', '?': WriteLn('írásjel');
    else                WriteLn('egyéb
karakter')
  end;
  ReadLn
end.
```

Karakterlánc típus valamint a strukturált (összetett) típusok

A karakterlánc - String típus

Deklarálása: **STRING***[***[***maxhossz***]***]*

ahol 1 <= maxhossz <= 255, ha elhagyjuk, maxhossz = 255. Dinamikus hosszúságú karaktersorozat: hossza futás közben változhat, elemei Char típusúak [1].

Karakterlánc konstans: 'aposztrófok közötti szöveg'

Hivatkozhatunk a karakterlánc egy elemére (amely Char típusú):

*azonosító***[***index***]**, pl. karlanc[5].

A karakterlánc típusú változó értékét megváltoztathatjuk értékadással, valamint beolvasással. Értékét kiírathatjuk a képernyőre.

Pl. karlanc := 'hahó'

Műveletek:

+ összefűzés, pl. file_spec := utvonal + file_nev;

Relációs műveletek: <, >, <=, >=, =, <> (az eredmény természetesen logikai típusú, az első nem egyenlő karakter ASCII kódja határozza meg. Pl. 'ATTILA' < 'ALFONZ' hamis.)

Adatábrázolás:

A rendszer a karakterek ASCII kódját tárolja *maxhossz*+1 bájton. A karakterlánc 0. bájtja egy tartalmazza a karakterlánc hosszát, **Ord**(karlanc[0]) = **Length**(karlanc).

Szabványos függvények, eljárások:
Függvények:
Length - a karakterlánc dinamikus hosszát adja vissza.
Pos - megkeresi a karakter első előfordulását a karakterláncban, és visszaküld egy egész értéket, amely az S. szorszáma.
Copy - visszaad egy sztring egy részét.
Concat – összekapcsol több sort.
Eljárások: **Str, Val**,
Delete – törli a sztring egy részét.
Insert - egy szöveget egy sztringbe helyez.

Lehetőleg a string műveleteket és függvényeket használjuk a karakterláncok kezelésére, a karakterenkénti hozzáférést körültekintően végezzük [4].

Példa [1]:
Olvassunk be egy modatot, és írjuk ki csupa nagybetűvel!

```
program Nagybetu;
uses Crt;
var s: string;
    i: byte;
begin
  ClrScr;
  ReadLn(s);
  for i := 1 to Length(s) do
    s[i] := Upcase(s[i]);
  WriteLn(s);
  ReadLn
end.
```

A tömb - Array típus

A tömb strukturált (összetett adattípus).

Deklarálása: **ARRAY[***indextípus [,indextípus...]***] OF** *elemtípus*

Pl.
vektor: array[1..10] of integer; {Tíz elemű egydimenziós tömb.}

matrix: array[1..5, 1..4] of integer; {5 sorból és 4 oszlopból álló kétdimenziós tömb, mátrix, elemei integer típusúak.}

A tömb olyan adatcsoport, melynek elemei azonos típusúak, az elemek száma rögzített (statikus méretű), sorrendjük kötött (indexelés), az elemekhez közvetlenül hozzáférhetünk. Meghatározása: név, dimenzió, elemeinek típusa, indexeinek típusa és tartománya [3].

Az indextípus csak sorszámozott típus lehet (kivéve Longint), többnyire intervallum típus,

pl.

array[1..10] of real {általában egész típus intervalluma}
array['a'..'z'] of real {ritkábban karakteres típus intervalluma}
array[byte] of real {még ritkábban egy előre definiált típus}.
Többdimenziós tömbök: a tömb elemei maguk is tömbök,

pl.
m: array[1..10, 1..20] of integer,
vagy
m: array[1..10] of array[1..20] of integer.
A hivatkozás a tömb egy elemére az index segítségével történik,

pl.
vektor[3] {a vektor nevű tömb 3. eleme, Integer típusú},

matrix[2, 3] vagy matrix[2][3] {a matrix nevű kétdimenziós tömb 2. sorának 3. eleme}.

Műveletek:

Két azonos típusú tömbre az értékadás és az egyenlőség vizsgálat megengedett. Egy vektor bemásolható a mátrix egy sorába.

A tömbökkel végzett műveletek során többnyire a for ciklust használjuk, úgy hogy a ciklusváltozót végigléptetjük a tömb indextartományán.

Pl. egy a vektor nevű tömb beolvasása a billentyűzetről:

```
for i := 1 to 10 do
  begin
    Write('Kérem a tömb ',i,'. elemét: ');
    ReadLn(t[i])
  end;
```

Tömb típusú konstans:

Tömb konstanst csak a **Const** deklarációs részben adhatunk meg tipizált konstansként. Az elemeket zárójelben, vesszővel elválasztva kell felsorolnunk.

Pl.
const T1 : array[1..3, 1..4] of byte = ((1, 3, 4, 1), (2, 3, 4, 2), (1, 6, 3, 5));

Adatábrázolás:

A rendszer a tömböt sorfolytonosan tárolja a memóriában. A foglalt terület az elemek száma szorozva egy elem méretével.

Tömb típus deklarálása:

Tömb típusú változók használatakor célszerű először a **Type** típusdeklarációs részben a megfelelő típusokat deklarálni, majd ezeket a saját típusainkat használhatjuk a változók deklarálásakor (a Var után). A Pascal nyelv logikája ezt az áttételes deklarálást támogatja, és bizonyos esetekben ez nem is kerülhető meg **[1]**.

Pl.

```
type VektorTip = array[1..4] of integer;
     MatrixTip = array[1..5] of VektorTip;
var  v1, v2: VektorTip;
     m1, m2: MatrixTip;
```

A fenti példában elvégezhető a következő értékadás: pl. m1[2] := v1.

Példák [1]:

1. Töltsünk fel egy 10 elemű integer tömböt. Számítsuk ki az elemek számtani átlagát.

```
program Atlagsz;
uses Crt;
type TombTip = array [1..10] of integer;
var     t: TombTip;
        atlag: real;
        osszeg, i: integer;
begin
  ClrScr;
  {beolvasás}
  for i := 1 to 10 do
    begin
      Write('Kérem a tömb ',i,'. elemét: ');
      ReadLn(t[i])
    end;
  {átlagszámítás}
  for i := 1 to 10 do
      osszeg := osszeg+t[i];
  atlag := osszeg/10;
  WriteLn('A tömb elemeinek számtani átlaga: ',
atlag:10:2);
  ReadLn
end.
```

2. Állítsuk elő a Fibonacci sorozat (1, 1, 2, 3, 5, 8, 13...) első 20 elemét.

```
program Fibonacci;
uses Crt;
type TombTip = array [1..20] of integer;
var  t: TombTip ;
     i: integer;
begin
  t[1] := 1;
  t[2] := 1;
  for i := 3 to 20 do
     t[i] := t[i-2] + t[i-1];
  for i := 1 to 20 do
     Write(t[i], '  ');
  ReadLn
end.
```

A rekord - Record típus

A rekord strukturált adattípus, amelyben különböző típusú adatokat (mezőket) fogunk össze.

Deklarálása:

Deklarálásakor a Record és End kulcsszavak között fel kell sorolnunk az egyes mezők neveit valamint típusait. A rekord tartalmazhat egy változó részt is, amely a fix rész egy mezőjétől (szelektor mező) függően más-más adatokat tartalmazhat [5].

RECORD
[*mezőlista*;]
[**CASE** *szelektormező***:** *sorszámozott típus* **OF**
*állandók***: (***mezőlista***)**
[*állandók***:** *(mezőlista)*...]]
END

ahol mezőlista:
mezőazonosító [**,***mezőazonosító*...]**:** *típus*
[**;***mezőazonosító* [**,***mezőazonosító*...]**:** *típus*...]

Példák:

```
1.
type DolgozoTip = record
                    Nev: string;
                    Hazas: boolean;
                    Fizetes: real;
                  end;
var Dolgozok: array[1..50]of DolgozoTip;
    d1, d2: DolgozoTip;
2.
type RekTip = record
                Nev: string[70];
                Lakohely: string[70]
                case Nagykoru: boolean of
                  True: (Gyerek_Szam: byte);
                  False: (Anyja_Neve:
string[70];
                          Apja_Neve:
string[70]);
              end;
```

Hivatkozás a rekord egy mezőjére:

*rekordazonosító***.***mezőazonosító*

Pl.

d1.Nev

Dolgozok[5].Hazas

A With utasítás

Ha egy programrészletben gyakran hivatkozunk egy (vagy több) rekord mezőire, akkor a With utasítással ez leegyszerűsíthető, a rekordazonosító elhagyható.

Szintaktikája: **WITH** *rekordazonosító* [**,** *rekordazonosító*] **DO** *utasítás*

Pl.

```
with d1 do
  begin
    ReadLn(Nev);
    ReadLn(Fizetes)
  end;
```

Ha több rekordazonosítót sorolunk fel, akkor az utolsónak a legnagyobb a prioritása [12].

Rekord típusú konstans:

Hasonlóan a tömb konstanshoz csak tipizált konstans kezdőértékeként adhatjuk meg a **Const** deklarációs részben.

Pl.

```
const Origo: record
               x, y: integer;
             end = (x: 320; y: 240);
```

Adatábrázolás:

A memóriában elfoglalt hely egyszerűen a mezők helyfoglalásának az összege (fix rész + legnagyobb változó rész).

A halmaz - Set típus

A programozásban a halmaz azonos típusú különböző elemek összességét jelenti. A halmazt az elemek felsorolásával adhatjuk meg. Az elemek rendezetlenek. Az összetett adattípusokhoz soroljuk, bár a halmaz egy elemére nem tudunk hivatkozni [11].

Deklarálása: **SET OF** *alaptípus*

ahol az *alaptípus* csak olyan sorszámozott típus lehet, amelynek maximálisan 256 eleme van.

Halmaz típusú konstans:

Szögletes zárójelben a halmaz elemeit (az alaptípussal megegyező típusú konstansokat) vagy azok intervallumait felsoroljuk.

Pl.

```
const Betuk = ['A'..'Z', 'a'..'z'];

H1 := [1, 4, 6, 8..12]
H2 := [] {üres halmaz}
```

A programunkban egy halmaznak a halmaz konstanstól egy kicsit különböző **halmazkonstruktorral** is értéket adhatunk. Itt a szögletes zárójelben felsorolt kifejezések változókat is tartalmazhatnak.

Pl.

```
k := '?';
H := [k, '!','.'];
```

Műveletek:
* metszet
+ egyesítés
- különbség
Logikai típusú eredményt szolgáltatnak:
= egyenlőség
<> különbözőség
<=, >= tartalmazás (részhalmaz)
IN elemvizsgálat

Adatábrázolás:

Egy lehetséges halmazelemnek egy bit felel meg a memóriában. Ha az lehetséges elem benne van a halmazban, akkor a bit 1, ellenkező esetben 0. Gyakran az első és az utolsó bájt olyan biteket is tartalmazhat, amelyek nem vesznek részt a tárolásban.

Alprogramok

Az alprogram olyan utasítások csoport, amelyet a program bizonyos pontjairól aktivizálhatunk. Az alprogramokat a deklarációs részben kell megírnunk (**procedure, function** kulcsszavak után). Az alprogramok tartalmazhatnak újabb alprogramokat (egymásba ágyazás) [1].

Akkor használjuk őket, ha

- bizonyos tevékenység többször előfordul a programban,
- egy nagy programot tagolni, strukturálni szeretnénk [3].

Két fajtája van:

- eljárás (procedure): egy tevékenységcsoportot hajt végre, utasításszerűen hívjuk (ld. standard eljárások),
- függvény (function): feladata egy érték előállítása, hívásakor neve operandusként egy kifejezésben szerepelhet (ld. standard eljárások).

Már eddig is sok szabványos eljárást és függvényt használtunk, melyeket a különböző egységekben deklaráltak. Nézzük, hogyan készíthetünk saját alprogramokat!

Eljárás - Procedure

a, Szerkezete:

Hasonló a programéhoz.

Eljárás fej:	**PROCEDURE** *azonosító* [**(** *formális paraméter lista* **)**]**;**
Deklarációs rész:	**Label**... **Const**... **Type**... **Var**...

	Procedure...
	Function...
Végrehajtandó rész:	**Begin** *utasítások* **End**;

ahol, formális paraméter lista:
[**Var**] *azonosító* [**,** *azonosító*...] **:** *típusazonosító* [**;** [**Var**] *azonosító* [**,** *azonosító*...] **:** *típusazonosító*...]

Például:

```
procedure Teglalap(a, b: integer; var t, k:
integer);
  begin
    t := a * b;
    k := 2 * (a + b)
  end;
```

b, Az eljárás hívása [12]:

azonosító [**(***aktuális paraméter lista***)**]

ahol az aktuális paraméter lista elemei kifejezések vagy változók lehetnek (ld. paraméterátadás) egymástól vesszővel elválasztva.

Pl. *Teglalap(5, 4, Ter, Ker)*

c, Az eljárások hatásköre [11]:

A Pascal nyelv befelé struktúrált, az alprogramokat egymásba ágyazhatjuk (az alprogram deklarációs részében is lehet alprogram). Ezért fontos tisztán látnunk, hogy hogy a főprogramból illetve egy alprogramból mely eljárásokat hívhatjuk meg:

- A program illetve egy eljárás meghívhatja (ismeri) azokat az alprogramokat, melyeket a program vagy az adott alprogram deklarációs részében deklaráltunk, de azok alprogramjait már nem. - Egy alprogram meghívhatja az ugyanazon deklarációs részben (, ahol őt deklaráltuk) korábban deklarált alprogramokat.
- Egy alprogram meghívhatja az őt tartalmazó eljárásokat.
- Egy alprogram meghívhatja saját magát.

d, Paraméterek [8]:

A paraméterek az eljárás és az őt hívó programrész közötti adatcserét, kommunikációt szolgálják. A formális paraméterekkel írjuk le az alprogram tevékenységét. Híváskor ezek helyére konkrét objektumokat, aktuális paramétereket írunk.

Az aktuális és a formális paramétereknek meg kell egyezniük számban, sorrendben és típusban.

Vigyázzunk, hogy a formális paraméterek típusának megadásakor csak típusazonosítót használhatunk, így pl. a következő eljárásfej hibás: *procedure elj (t: array[1..10] of real)*;

A paraméterátadás két féle módon történhet:

- *Érték szerinti paraméter átadás* (a deklarációban a formális paraméter előtt nincs **Var**)

Ekkor az aktuális paraméter értéke kerül át a formális paraméterbe. Az eljárás minden egyes hívásakor a rendszer tárterületet rendel a verem memóriában a formális paraméterekhez, és ide másolja be az aktuális paraméterek értékeit. Az eljárás végeztével ez a terület felszabadul. Az aktuális paraméter értékét az eljárás nem változtathatja meg, így ez csak *bemenő paraméter*.

Az aktuális paraméter kifejezés lehet.

- *Cím szerinti paraméter átadás* (a deklarációban a formális paraméter elé **Var** -t írunk)

Az aktuális paraméter címe kerül át a formális paraméterhez, ha változik a formális paraméter, akkor változik az aktuális is. Ezáltal egyaránt használhatjuk *be- és kimenő paraméterként* is [1].

Az aktuális paraméter csak változó lehet.

e, Lokális és globális változók [4]

Egy eljárásban deklarált változókat ezen eljárás lokális változóinak nevezzük. Ezek a program más részein nem ismertek. (Így különböző eljárásokban előfordulhatnak azonos nevű változók, amelyeknek azonban semmi közük egymáshoz.) A lokális változókhoz (az eljárás paramétereihez hasonlóan) a rendszer a veremben rendel tárterületet, dinamikus módon, azaz csak akkor van címe a változóknak, ha az eljáráson van a vezérlés. A lokális változók értéke az eljárás két hívása között elvész.

Egy eljárásra nézve globális változó egy őt tartalmazó eljárásban vagy a főprogramban deklarált változó. Ezt az eljárás ismeri, hacsak nem deklaráltunk egy vele azonos nevű lokális változót vagy az eljárásnak nincs egy vele azonos nevű paramétere. Ekkor a lokális változó "eltakarja" a globálisat. A globális változó értéke természetesen nem vész el. (Abban az esetben használhatunk egy a lokálissal azonos nevű globális változót, ha az a főprogram változója. Ekkor a program nevével kell minősítenünk a globális változót: *programnév.változónév*.)

Lokális és globális változók – példaprogram [1]

```
program Pelda;

var a, b, c: integer;
    x, y, z: real;

procedure p1(x:real);          {a p1 nem ismeri,
nem használhatja a főprogram x változóját}
  var a: array[1..10]of real;
      s: string;               {lokális változó}
  begin
    {globális változók: b, c, y, z}
  end;

procedure p2;
  begin
   {imeri a főprogram összes változóját, mint
globális változót, nem ismer a p1 s változóját}
  end;

begin

end.
```

f, Információ csere [5]

Összefoglalva elmonhatjuk, hogy egy alprogram kétféle módon kommunikálhat az őt hívó programegységgel,

- a paramétereken keresztül,
- a globális változók segítségével.

Kommunikáció paraméterek illetve globális változók segítségével - példaprogram

1. A program egy globális változóban (tömbben) tárolja az adatokat, melyeket az eljárások feldolgoznak **[1]**.

```
program Globalis;

uses Crt;
type TOszt=array[1..50] of string;
var Osztaly: TOszt;
```

```
    Letszam: integer;
    c: char;

procedure Beiras;
  var i: integer;
  begin
    ClrScr;
    Write('Hány név lesz: ');
    ReadLn(Letszam);
    for i := 1 to Letszam do
      begin
        Write('Az ',i,'. név: ');
        ReadLn(Osztaly[i])
      end
  end;

procedure Rendezes;
  var i, j: integer;
      seged: string;
  begin
    for i := 1 to Letszam-1 do
      for j := i + 1 to Letszam do
        if Osztaly[i] > Osztaly[j] then
          begin
            seged := Osztaly[i];
            Osztaly[i] := Osztaly[j];
            Osztaly[j] := seged
          end
  end;

procedure Listazas;
  var i: integer;
  begin
    ClrScr;
    for i:=1 to Letszam do
      WriteLn(Osztaly[i]);
    ReadKey
  end;

begin
  repeat
    ClrScr;
    WriteLn('1. Nevek beírása');
```

```
    WriteLn('2. Névsorba rendezés');
    WriteLn('3. Listázás');
    WriteLn('4. Vége');
    repeat c := ReadKey until c in ['1'..'4',
#27];
    case c of
      '1': Beiras;
      '2': Rendezes;
      '3': Listazas;
    end;
  until (c = '4') or (c = #27)
end.
```

2. Az egyes eljárások több adatsorral (tömbbel) is dolgoznak, paramétereken keresztül kapják meg a feldolgozandó tömböt, illetve adják vissza a főprogramnak **[1]**.

```
program Parameterek;
uses Crt;
type TNaplo = array[1..50] of real;
var A_oszt, B_oszt: Tnaplo;
    A_letsz, B_letsz: integer;

procedure Beolvas(var Oszt: Tnaplo; var Letsz:
integer);
  var i: integer;
  begin
    Write('Hány tanuló van: ');
    readln(Letsz);
    for i := 1 to Letsz do
      begin
        Write('Az ',i,'. átlaga: ');
        ReadLn(Oszt[i])
      end;
    WriteLn;
  end;

function Atlag(Oszt: Tnaplo; Letsz: integer):
real;
  var i: integer;
      sum: real;
  begin
```

```
    sum := 0;
    for i := 1 to Letsz do
      sum := sum + Oszt[i];
    Atlag:= sum / Letsz
  end;

begin
  ClrScr;
  WriteLn('Irja be az A osztály tanulóinak
átlagait: ');
  Beolvas(A_oszt, A_letsz);
  WriteLn('Irja be az B osztály tanulóinak
átlagait: ');
  Beolvas(B_oszt, B_letsz);
  WriteLn('Az A osztály átlaga: ', Atlag(A_oszt,
A_letsz):4:2 );
  WriteLn( 'A B osztály átlaga: ', Atlag(B_oszt,
B_letsz):4:2 );
  ReadKey;
end.
```

Függvény - Function

A függvény feladata egy érték előállítása. Ezt az értéket a függvény nevéhez rendeljük, a függvény törzsében kell szerepelni legalább egy értékadó utasításnak, amelyben a függvény neve a baloldalon áll. (Vigyázzunk, ha a jobboldalon szerepeltetjük a függvény nevét, akkor az már rekurziót jelent [2].

A függvényt egy kifejezésben hívhatjuk meg, pl. egy értékadó utasítás jobboldalán.

Szerkezete megegyezik az eljárásával azzal a különbséggel, hogy még meg kell határoznunk a viszatérési érték típusát is. Így a függvény feje:

FUNCTION *azonosító* [**(** *formális paraméter lista* **)**]**:** *típusazonosító***;**

ahol a típusazonosító csak csak sorszámozott, valós, karakterlánc vagy mutató lehet.

Pl.

```
function Tangens(Alfa: real): real;
```

```
begin
  if cos(Alfa) <> 0 then
    Tangens := Sin(Alfa) / Cos(Alfa)
end;
```

Rekurzió

Ha egy alprogram saját magát meghívja, akkor rekurzióról beszélünk. Megkülönböztethetünk közvetlen és közvetetten rekurziót.

A rekurzió alkalmazásának egyik területe, amikor úgy oldunk meg egy problémát, hogy visszavezetjük egy egyszerűbb esetre, majd ezt addig folytatjuk, míg el nem jutunk a triviális esetig. A módszer a matematikai indukción alapszik [3].

A megoldás lépései:

1. Megkeressük azt a legegyszerűbb esetet, ahol a megoldás már magától értetődő - triviális eset. Ekkor áll le a rekurzív hívások sorozata.
2. Megvizsgáljuk, hogy ismételt egyszrűsítésekkel hogyan juthatunk el a triviális esethez. (Az általános esetet visszavezetjük az eggyel egyszerűbbre.)

Példák [1]:

1. Faktoriális számítás
- Triviális eset: 1! = 1
- Egyszerűsítés: N! = N*(N-1)!

Ezzel a problémát megoldottuk, már csak kódolnunk kell.

```
program Faktorialis;
uses Crt;
var N: integer;

function  FaktIter(N: integer): longint;
{Iterációs megoldás}
var i: integer;
    fakt: longint;
begin
  fakt := 1;
  for i: = 2 to N do fakt := fakt * i;
  FaktIter := Fakt
end;

function FaktRek(N: integer): longint;
{Rekurzív megoldás}
begin
```

```
  if N = 1 then
    FaktRek := 1
{Triviális eset}
  else
    FaktRek := n * FaktRek(n-1)
{Visszavezetés az egyszerűbbre}
end;

begin
  ReadLn(N);
  WriteLn(FaktIter(N));
  WriteLn(FaktRek(N));
  ReadKey
end.
```

Megj.: Bár a feladat kitűnő példa a rekurzív algoritmusra, az iterációs (ciklussal történő) megodás jobb, mivel az ismételt függvényhívások időigényesek.

2. Fibonacci sorozat (1, 1, 2, 3, 5, 8, 13...) N. eleme
- Triviális eset: az első és a második elem értéke 1.
- Egyszerűsítés: az N. elem az N-1 - edik és az N-2 - dik elemek összege **[1]**.

```
program Fibon_Rek;
uses Crt;
var n: byte;

function Fibo(n: byte): integer;
begin
  if (n = 0) or ( n = 1) then
    Fibo := 1                        {Triviális
eset}
  else
    Fibo := Fibo(n-1) + Fibo(n-2)
{Visszavezetés az egyszerűbbre}
end;

begin
  ClrScr;
  ReadLn(n);
  WriteLn(Fibo(n));
  ReadKey
end.
```

Állománykezelés

A programok a bemeneti adataikat nem csak a billentyűzetről, hanem a háttértárolókon lévő állományokból is kaphatják, valamint kimeneti adataikat a képernyőn történő megjelenítés mellett állományokban is tárolhatják. A Pascal nyelvben három összetett típus és az ezekhez kapcsolódó szabványos eljárások és függvények valósítják meg az állományok kezelését [4].

Típusos állomány

Deklarálása: **FILE OF** *alaptípus*

Összetett típus, fizikailag egy lemezes állomány. Egyforma méretű elemekből (komponensekből) áll. Az elemek számának csak a lemez mérete szab határt.

A típusos állományból való olvasás illetve az állományba való írás egysége a komponens.

Az elemekhez a rendszer sorszámot rendel 0-tól kezdődően. Az elérés szekvenciálisa (Read, Write) vagy a komponensek sorszáma szerint direkt módon történhet (az állomány mutató mozgatásával) [5].

A program mindig egy logikai állományt kezel, melyet hozzá kell rendelnünk egy fizikai állományhoz (Assign), majd használat előtt meg kell nyitnunk. A Rewrite eljárás létrehozza, és megnyitja a logikai fájlhoz rendelt fizikai állomány. Ha a fizikai fájl már létezett, akkor törli annak tartalmát. A Reset eljárással egy már létező állományt nyithatunk meg. Ekkor az állománymutató az 0. komponensre áll. (Ezért ezt az eljárást használhatjuk egy nyitott állomány elejére való ugrásra is.) Használat után a Close eljárással zárjuk le fájlunkat! [1]

A típusos állományból a Read eljárás olvas be változókba adatokat. Ügyeljünk arra, hogy a változó típusa egyezzen meg a fájl alaptípusával! Beolvasás után az állomány mutató automatikusan a következő komponensre lép (szekvenciális elérés). Egy változó (vagy kifejezés) értékét a Write eljárással írhatjuk ki egy fájlba. Hasonlóan az olvasáshoz a változó típusának meg kell egyeznie a fájl elemeinek a típusával, valamint az eljárás után az állomány mutató továbblép. Ha az állomány mutató a fájl végén (az utolsó elem mögött) áll, akkor az Eof függvény értéke True. Nézzünk egy példát a fájl szekvenciális feldolgozására:

```
Reset(f)
while not Eof(f) do
  begin
    Read(f,v);
    {a v változóban lévő adat feldolgozása}
  end;
```

Az állomány mutató direkt pozicionálását a Seek eljárás valósítja meg. A FilePos függvénnyel lekérdezhetjük az aktuális pozíciót, a FileSize függvény pedig az állomány elemeinek a számát (méretét) adja vissza.

Az I/O műveletek során nagy a hibalehetőség (pl. a lemezegység, fájl nem elérhető). Az esetleges futási hibákat tudnunk kell kezelni, ha megbízhatóan működő programot szeretnénk írni. Ha az I/O műveletek ellenőrzése aktív (ez az alapértelmezés), akkor programunk futási hibával leáll egy I/O hiba esetén. Ezért I/O műveletek ellenőrzését inaktívvá kell tennünk a {$I-} fordítási direktívával a kényes műveletek esetén. A művelet után az esetleges hiba kódját az IOResult függvénnyel kérdezhetjük le [12]. Erre egy példa:

```
    Assign(f, 'adatok.dat');
    {$I-}
    Reset(f);                   {megpróbáljuk
megnyitni a fájlt}
    {$I+}
    if IOResult <> 0 then       {ha hiba történt,
tehát a fájl nem létezik, }
      Rewrite(f);               {akkor létrehozzuk
az állományt}
```

A Truncate eljárással levághatjuk a fájl komponenseit az aktuális pozíciótól kezdődően.

Lezárt állományokra használhatjuk a Rename valamint az Erase eljárásokat a fájlok átnevezésére illetve törlésére.

Példa [1]:

1. A program egy bolt árucikkeinek adatait (név, kód, ár) tárolja és kezeli egy állományban.

```
program aruk;
uses crt;
type TAru = record          {A fájl alaptípusa.}
         kod: string;
         nev: string[15];
         ar: real;
         t: boolean;        {Ez a mező jelzi, hogy
e rekord törölt-e (logikai törlés).}
       end;

var bolt: file of TAru;
    aru: TAru;
    mkod: string;
```

```
    mvalasz: char;

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

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

{Egy rekord módosítása a fájlban.}
```

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

{Egy rekord logikai törlése: a t mezőt True
értékűre állítja,
 az ilyen rekordokat a program nem létezőnek
tekinti.
 Fizikai törlés kilépéskor.}
procedure Torles;
begin
  ClrScr;
  WriteLn('Kerem az aru kodjat!');
  ReadLn(mkod);
  if Van(mkod) then
    begin
      Seek(bolt, FilePos(bolt) - 1);
      aru.t := true;
      Write(bolt, aru);
    end
  else
    begin
      WriteLn('Nincs ilyen aru!');
      ReadKey
```

```
    end
end;

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

{Fizikai törlés: azon rekordok átmásolása egy új
állományba, melyek nincsenek logikailag törölve.
 A régi állomány törlése, az új fájl átnevezése
a régi nevére.}
procedure Surites;
var ujfile: file of TAru;
begin
  Assign(ujfile, 'ujfile');
  Rewrite(ujfile);
  Seek(bolt, 0);
  while not Eof(bolt) do
    begin
      Read(bolt, aru);
      if aru.t = false then write(ujfile, aru);
     end;
  Close(bolt);
  Erase(bolt);
  Close(ujfile);
  Rename(ujfile, 'bolt');
end;
```

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

Szöveges állomány - Text

Deklarálása: **TEXT**

A Pascal programban szöveges állományként kezelhetjük az egyszerű ASCII szövegeket. (Például a .pas kiterjesztésű forrásprogramjainkat.) A szöveges állomány változó hosszúságú sorokból áll, melyeket a sorvégjel zár le (CR/LF). Az állományt az állományvégjel zárja(^Z). Az Eoln illetve az Eof függvény értéke True, ha az aktuális pozíció egy sorvégjelen vagy az állomány végén áll. A SeekEoln illetve a SeekEof függvények az állomány következő TAB szóköz illetve TAB szóköz és sorvégjel karaktereit átugorva tájékoztatnak arról, hogy sorvégjelen illetve az állomány végén állunk-e [1].

A szöveges állományt csak szekvenciálisan érhetjük el. Az állomány csak olvasásra vagy csak írásra lehet megnyitni. Az

állományból olvasni a Read, ReadLn, illetve írni a Write, Writeln eljárásokkal tudunk. Ha az eljárásoknak a fájl azonosító paraméterét elhagyjuk, akkor az olvasás / írás az alapértelmezett input / output szöveges állományból / -ba történik, ami a billentyűzet illetve a monitor. Szöveges állományból (azaz a billentyűzetről is) olvashatunk egész, valós, karakteres és sztring típusú változokba adatokat. Az állományba az előbbi típusokon kívül még logikai értéket is kiírathatunk.

Az fizikai állományhoz az Assign eljárással rendelhetünk egy Text típusú változót, azaz a logikai állományt. A Rewrite eljárás csak írásra nyitja meg a szöveges állományt, ha nem létezett létrehozza, egyébként törli a tartalmát. A Reset eljárással csak olvasásra nyithatunk meg egy már létező fájlt. Az Append eljárás egy létező állományt nyit meg írásra, és az állománymutató a fájl végére állítja. Az állományt a Close eljárással zárhatjuk be [3].

Az I/O műveletek hibakódját az IOResult függvény adja vissza (bővebben ld. Típusos állományok).

Lezárt állományokra használhatjuk a Rename valamint az Erase eljárásokat a fájlok átnevezésére illetve törlésére.

A Fluss és a SetTextBuf eljárásokkal az írás, olvasás során a rendszer által használt átmeneti tárolóhoz (pufferhez) férhetünk hozzá.

Példa [1]:

1. A doga.txt állományban egy feladatsor van, kérdések és válaszok felváltva egymás után. Minden kérdés illetve válasz új sorban kezdődik. A kérdések egy számjeggyel kezdődnek, és kérdőjellel fejeződnek be. Készítsünk két új szöveges állományt úgy, hogy az egyik csak a kérdéseket, a másik pedig csak a válaszokat tartalmazza.

```
program Doga;
var f, k, v: text;
    s: string;
    kerd: boolean;
begin
  Assign(f, 'doga.txt');
  Assign(k, 'kerd.txt');
  Assign(v, 'val.txt');
  Reset(f);
  Rewrite(k);
  Rewrite(v);
  while not Eof(f) do
    begin
      ReadLn(f, s);
{Egy sor beolvasása a dolgozatból}
```

```
      if s[1] in ['1'..'9'] then kerd := true;
{A sor első karaktere számjegy-e (kérdés)}
      if kerd then WriteLn(k, s) else WriteLn(v,
s);  {Kiírás a megfelelő állományba}
      if s[Length(s)] = '?' then kerd := false;
{Vége-e a kérdésnek}
    end;
  Close(f);
  Close(k);
  Close(v)
end.
```

Típusnélküli állomány

Deklarálása: **FILE**

Általában gyors adatmozgatás vagy ismeretlen állomány esetén használjuk. Hasonló a típusos állományhoz, de az elemeinek nem a típusa, hanem a hossza a lényeges. A komponensek hosszát a fájl megnyitásakor adhatjuk meg (Reset, Rewrite), az alapértelmezés 128 bájt. Az állomány írható, olvasható, az elérés szekvenciálisan (BlockRead, BlockWrite eljárásokkal) vagy az elemek sorszáma szerint direkt módon történhet [4].

További függvények, eljárások: Assign, Close, Eof, Erase, FilePos, FileSize, IOResult, Rename, Seek, Truncate.

Példa:

1. Tördeljünk szét egy állományt egy kilobájt hosszúságú kisebb állományokra! **[1]**

```
program Tordel;
uses Crt;
var forras, cel: file;
    n, maradek: integer;
    s: string;
    t: array[1..1024]of byte;
begin
  Assign(forras, 'nagyfajl.arj');
  Reset(forras,1);
{Megnyitás egy bájt elemhosszúsággal}
  n := 0;
  while not Eof(forras) do
    begin
      inc(n);
      str(n,s);
```

```
      Assign(cel, 'kisfajl.'+s);
{A cél állomány hozzárendelése, a kiterjesztés a
sorszám}
      Rewrite(cel,1);
      maradek := FileSize(forras)-
FilePos(forras);  {A forrás még át nem másolt
részének a hossza}
      if maradek >= 1024 then
        begin
          BlockRead(forras, t, 1024);
          BlockWrite(cel, t, 1024);
        end
      else
        begin
          BlockRead(forras, t, maradek);
          BlockWrite(cel, t, maradek);
        end;
      Close(cel)
    end;
  Close(forras)
end.
```

Karakteres képernyő kezelése - a CRT unit

A Crt egység a karakteres képernyő, a billentyűzet valamint a hangszóró kezelését segítő függvényeket, eljárásokat tartalmazza. Mint az egységek többsége, a Crt unit is definiál konstansokat, változókat.

Színek:

A karakteres képernyő tartalma megtalálható az ún. képernyő memóriában. Itt egy karaktert két bájton tárol el a rendszer, melyek a karakter ASCII kódja (1 bájt) valamint a karakter attribútuma (1 bájt). Ez utóbbi a színinformációt hordozza, az alábbi módon:

```
7 6 5 4 3 2 1 0

V R G B I R G B
```

A 0.-3. bit a karakter tintaszínét határozza meg, R, G, B az additív színkeverés három alapszíne, I pedig az intenzitás. Például 0100 - piros, 1100 - világospiros, 0101 - lila. A 4.-6. bitek a karakter háttérszínét kódolják. Ha a 7. bit (V) egyes, akkor a karakter villog.

A fentiekből következik, hogy összesen 16 tinta- és 8 háttérszínt használhatunk. A színek kódjait könnyen kiszámolhatjuk, ezeket a megfelelő eljárásokban használhatjuk, de a könnyebb megjegyezhetőség kedvéért a Crt unit az alábbi szín konstansokat definiáljam [12].

Tinta- és háttérszínek:			**További tintaszínek:**		
Balck	0	Fekete	**DarkGray**	8	Sötétszürke
Blue	1	Kék	**LightBlue**	9	Világoskék
Green	2	Zöld	**LightGreen**	10	Világoszöld
Cyan	3	Türkiz	**LightCyan**	11	Világostürkiz
Red	4	Piros	**LightRed**	12	Világospiros
Magenta	5	Lila	**LightMagenta**	13	Világoslila
Brown	6	Barna	**Yellow**	14	Sárga
LightGray	7	Világosszürke	**White**	15	Fehér

Blink 128 Villogás

Pl: *TextColor(Lightred+Blink)*, ezzel egyenértékű: *TextColor(12 + 128)* vagy *TextColor(140)*.

Fontosabb eljárások, függvények:

Képernyőkezelés:

Függvények:

WhereX - visszaadja a vízszintes koordinátát

WhereY – visszaadja a függőleges koordinátát

Eljárások:

TextBackground - a háttérszínt állítja be.

TextColor - a szöveg színtét állítja be.

ClrScr - törli a képernyőt.

CrlEol - kurzorpozíciótól kezdve a sor végéig törli a karaktereket.

DelLine - törli a kurzort tartalmazó sort.

InsLine - egy sort helyez a kurzor pozícióba.

GotoXY - a szöveges képernyő x.oszlopába és y.sorába viszi a kursort.

Window - a képernyőn kijelöl egy ablakot, azaz amit itt beállítottál akkora területre tudsz továbbra írni. Ezután az 1,1 koordináta az ablak bal felső sarka lesz.

NormVideo - beállítja a normál karakter fényerejét.

TextMode - szöveges mód beállítása.

Billentyűzetkezelés:

Függvények:

KeyPressed - True értékkel tér vissza, ha a gomb megnyomása megtörtént.

ReadKey - beolvass egy karaktert a billentyűzetről anélkül, hogy a képernyőn megjelenne.

Hang, késleltetés:

Eljárások:

Sound - bekapcsolja a hanggenerátort.

Delay - megállítja a program végrehajtását n milliszekundumig.

NoSound - kikapcsolja a hanggenerátort.

Példa [1]:

```
1. Mozgassunk egy téglalapot (egy kis képernyőt)
benne egy szöveggel a képenyőn a kurzormozgató
billentyűk segítségével!
program CrtPl;
uses Crt;
var  x1, x2, y1, y2: byte;
     c: char;
begin
```

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

A Turbo Pascal grafikája - a GRAPH unit

Ha rajzolni szeretnénk a képernyőre, akkor azt át kell kapcsolnunk grafikus üzemmódba. A grafikus képernyőn külön-külön hozzáférhetünk az egyes képpontokhoz. Ismernünk kell (illetve bizonyos határok között meghatározhatjuk) a képernyőnk felbontását (a VGA üzemmódban a legnagyobb felbontás 640x480) valamint azt, hogy hány színt használhatunk (az előbb említett felbontásnál 16 színt). A (0, 0) képpont a képernyő bal felső sarkában található [12].

Az egység fontosabb eljárásai, függvényei, típusai, konstansai:

A grafikus képrnyő inicializálása (átkapcsolás karakteres képernyőről grafikusra), bezárása:

Eljárások: InitGraph, DetectGraph, CloseGraph, stb.

Függvények: GraphResult, stb.

Konstansok: grafikus meghajtók (Pl. Detect = 0, CGA = 1 stb.); grafikus üzemmódok (pl. VGALo, VGAMed, VGAHi stb.)

Pl. [1]:

```
    uses Graph;
    var Meghajto, Uzemmod: integer;
    begin
      Meghajto := Detect; {Automatikusan
beállítja grafikus üzemmódot a legnagyobb
felbontással.}
      InitGraph(Meghajto, Uzemmod,
'C:\TP70\BGI');  {Inicializálás}
      If GraphResult <> 0 then
        begin
          WriteLn('Grafikus hiba!');  {Nem
sikerült az inicializálás, kilépés a programból}
          ReadLn;
          Halt
        end;
      ...{Grafika használata}
      CloseGraph    {Grafikus képernyő bezárása}
    end
```

Színek:

Konstansok: 16 színű üzemmódban megegyeznek a Crt unit konstansaival.
Eljárások: SetColor, SetBkColor, stb.
Függvények: GetColor, GetBkColor, stb.

Rajz:

Típusok: LineSettingsType (ld. GetLineSettings), stb.
Konstansok: vonalstílus (pl. SolidLn, DottedLn, stb. ld. SetLineStyle), vonalvastagság (NormWidth. ThickWidth ld. ld. SetLineStyle), rajzolási mód (pl. CopyPut, XorPut, stb. ld. SetWritMode)
Eljárások: PutPixel, Line, LineTo, LineRel, Circle, Rectangle,SetLineStyle,GetLineSettings, SetWriteMode, stb.

Kitöltött rajz:

Típusok: FillSettingsType (ld. GetFillSettings), stb.
Konstansok: kitöltési stílus (pl. SolidFill, LineFill, stb. ld. SetFillStyle)
Eljárások: Bar, Bar3D, FillEllipse, FloodFill, SetFillStyle, GetFillSettings, stb.

Szöveg:

Típusok: TextSettingsType (ld. GetTextSettings), stb.
Konstansok: betűtípus, szövegállás, szövegigazítás
Eljárások: OutText, OutTextXY, SetTextStyle, GetTextSettings stb.

Kurzor:
Függvények: GetX, GetY, stb.
Eljárások: MoveTo, MoveRel, stb.

Egyéb:
Kép mentése egy változóba, visszatöltése: ImageSize, GetImage, PutImage.

Példa [1]:

Kérjük be grafikus képernyőn egy parabola és egy egyenes egyenletét. Ábrázoljuk a görbéket, és metszéspontjaikat határozzuk meg grafikusan valamint analitikusan. A program tegye lehetővé, hogy a le-ill. felfele nyíllal Y irányban mozgathassuk az egyenest, a jobbra ill. balra nyíllal a meredekségét változtathassuk, a PgUp ill. PgDn billentyűkkel pedig a koordinátarendszer méretarányát (képpont/egység) változtathassuk **[1].**

```
   program GraphPl;

uses Crt, Graph;
var  dr, mo, h, px1, px2, py1, py2: integer;
     x1, x2, y1, y2, a, b, c, d, m, n: real;
     s1, s2: string[6];
     k: char;

{Grafikus kepernyőn adatbevitel az x változóba.}
procedure Beolvas(var x: real);
  var hiba, xx: integer;
      ch: char;
      s: string;
  begin
    s[0] := #0;
    xx:= GetX;
    repeat
      ch := ReadKey;  {Egy karakter leütése.}
      SetColor(yellow);
      if (ch in ['0'..'9']) or (ch in ['.','-'])
then
        begin
          OutText(ch); {A karakter
megjelenítése, ha számjegy, előjel vagy
tizedespont.}
          s := s + ch; {A karakter hozzáfűzése.}
        end;
       {Torles}
       if (ch = #8) and (s[0] > #0) then
         begin
             MoveTo(xx, GetY); SetColor(black);
OutText(s);
             s[0] := Chr( Ord(s[0]) - 1 );
             MoveTo(xx, gety); SetColor(yellow);
OutText(s);
          end;
    until ch = #13;  {ENTER-re adatbevitel vége}
    Val(s, x, hiba);    {Az s string
konvertálása valóssá, elhelyezése az x
változóban.}
  end;
```

```
{Koordinátarendszer kirajzolása.}
procedure Koord;
  var i: integer;
  begin
    Line(0, 240, 640, 240); Line(320, 0, 320,
480);
    Line(315, 5, 320, 0); Line(320, 0, 325, 5);
    Line(635, 235, 640, 240); Line(635, 245,
640, 240);
    for i := 0 to 640 div h do Line(h * i, 238,
h * i, 242);
    for i := 1 to 480 div h do Line(318, h * i,
322, h * i);
    OutTextXY(624, 228, 'x'); OutTextXY(325, 10,
'y');
  end;

{A parabola és az egyenes kirajzolása.}
procedure FvRajzolas;
  var px, py: integer;
      x, y: real;
  begin
    for px := 0 to 639 do      {Az x tengely
minden pixelénél a függvényértékek  }
      begin                    { kiszámítása,
megjelenítése.}
        x := (px - 320) / h;   {A pixelértékhez
x független változó érték rendelése.}
        y := a*x*x + b*x + c;  {Az y függő
változó kiszámítása.}
        py:=Round(-h*y + 240); {Az y függő
változóhoz pixel érték rendelése.}
        PutPixel(px, py, red); {Megjelenítés}
        y:=m*x + n;
        py:=Round(-h*y + 240);
        PutPixel(px, py, green);
      end;
  end;

{Az egyenes és a parabola metszési egyenletének
a megoldása.}
procedure Metszes;
  begin
```

```
    d := sqr(b-m) - 4*a*(c-n);
    if d < 0.0 then OutTextXY(50, 50, 'Nincs
gy”k!')
    else
      if d > 0.0 then
        begin
          x1 := ( -(b-m) + Sqrt(d) ) /2/a;
          x2 := ( -(b-m) - Sqrt(d) ) /2/a;
          y1 := m*x1 + n;
          y2 := m*x2 + n;
          px1 := Round(h*x1 + 320);
          px2 := Round(h*x2 + 320);
          py1 := Round(-h*y1 + 240);
          py2 := Round(-h*y2 + 240);
          SetLineStyle(1, 0, 1);
          Line(px1, py1, px1, 240);
          Line(px2, py2, px2, 240);
          Str(x1:6:2, s1);
          Str(x2:6:2, s2);
          OutTextXY(50, 50, 'A k,t gy”k:'+s1+' ;
'+s2)
        end
      else
        begin
          x1 := -(b-m) /2/a;
          y1 := m*x1 + n;
          px1 := Round(h*x1 + 320);
          py1 := Round(-h*y1 + 240);
          SetLineStyle(1, 0, 1);
          Line(px1, py1, px1, 240);
          Str(x1:6:2, s1);
          OutTextXY(50, 50, 'A gy”k:'+s1)
        end;
  end;

{A függvények és a metszéspontok törlése.}
procedure Torles;
  var px, py: integer;
      x, y: real;
  begin
    for px := 0 to 640 do
      begin
        x := (px-320) / h;
```

```
        y := m*x + n;
        py := Round(-h*y + 240);
        PutPixel(px, py, black);
        y := a*x*x + b*x + c;
        py := Round(-h*y + 240);
        PutPixel(px, py, black);
      end;
    SetColor(black);
    SetLineStyle(0, 0, 1);
    Koord;
    if d < 0 then begin SetColor(0);
OutTextXY(50, 50, 'Nincs gy”k!') end
    else
      if d > 0 then
        begin
          SetColor(black);
          Line(px1, py1, px1, 240);
          Line(px2, py2, px2, 240);
          OutTextXY(50, 50, 'A k,t gy”k:'+s1+' ;
'+s2)
        end
      else
        begin
          SetColor(black);
          Line(px1, py1, px1, 240);
          OutTextXY(50, 50, 'A gy”k:'+s1)
        end;
  end;

begin
  {Inicializálás}
  h := 40;  {A koordinátarendszer egysége 40
képpont.}
  dr := 0;
  InitGraph(dr, mo, 'c:\bp\bgi');
  SetColor(yellow);
  SetLineStyle(0, 0, 1);
  Koord;
  {A parabola és az egyenes egyenletének
beolvasása.}
  MoveTo(50,  40); OutText('A parabola
egyenlete:');
  MoveTo(50,  70); OutText('a: '); Beolvas(a);
```

```
  MoveTo(50,  90); OutText('b: '); Beolvas(b);
  MoveTo(50, 110); OutText('c: '); Beolvas(c);
  MoveTo(50, 140); OutText('Az egyenes
egyenlete:');
  MoveTo(50, 170); OutText('m: '); Beolvas(m);
  MoveTo(50, 190); OutText('n:  '); Beolvas(n);
  ClearDevice;

  {Ismétlés az ESC bill. leütéséig.}
  repeat
    setcolor(yellow);
    setlinestyle(0,0,1);
    Koord;
    FvRajzolas;
    Metszes;
    k := ReadKey;    {Várakozás egy billentyű
leütésére, kód beolvasása.}
    if k = #0 then   {Kettős kódú billentyű
(kurzormozgató)}
      begin
        k := ReadKey;  {Második kód beolvasása.}
        Torles;
        case k of
          #72: n := n + 0.2;      {Felfele nyíl:
az egyenes eltolása y irányban felfelé.}
          #80: n := n - 0.2;      {Lefele nyíl:
az egyenes eltolása y irányban lefelé.}
          #77: m := m + 0.1;      {Jobbra nyíl:
az egyenes meredekségének növelése.}
          #75: m := m - 0.1;      {Balra nyíl:
az egyenes meredekségének csökkentése.}
          #73: h := h * 2;        {PgUp: lépték
növelése.}
          #81: h := Round(h / 2); {PgDn: lépték
csökkentése.}
        end; {case}
      end {if}
  until k = #27;         {Kilépés ESC-re.}
  CloseGraph;
end.
```

Mutatók

Típusos mutató

Deklarálása: *azonosító*: ^*alaptípus*

Pl. Var p1, p2: ^real;

A mutató egy memóriacímet (4 bájtos: szegmens, ofszet) tartalmaz, egy *alaptípusú* változóra mutat. A mutatóhoz futás közben rendelhetünk memóriacímet és így tárterületet (dinamikus adattárolás).

A New eljárás a heap memóriában foglalterületet a mutatott változó számára, a Dispose eljárás pedig felszabadítja a lefoglalt területet. Így lehetővé válik, hogy egy nagy helyfoglalású adatstruktúra számára csak akkor kössünk le memóriát, amikor használjuk a változót. Nézzünk erre egy példát:

```
    type TTomb = array[1..1000]of real;
    var t: TTomb;  {A rendszer már a program
idításakor lefoglal 6000 bájtot az
adatszegmensben}
        pt: ^TTomb;{A rendszer a program
idításakor csak 4 bájtot foglal le a mutató
számára}
    begin
      ...
      New(pt);     {A heap-ben létrejön a
mutatott változó}
      ...          {Használhatom a pt által
mutatott változót (pt^)}
      Dispose(pt)  {A terület felszabadul a
heap-ben}
    end.
```

A mutatót ráirányíthatjuk egy az alaptípusával megegyező típusú változóra a @ operátorral vagy az Addr függvénnyel. Pl. pt := @t; vagy pt := Addr(t).

A Ptr függvénnyel a mutatónak egy tetszőleges memóriacímet adhatunk értékül.

A negyedik lehetőség arra, hogy egy mutatóhoz egy memóriacímet rendeljünk: értékadó utasítás egy vele azonos alaptípusú mutatóval.

Hivatkozás a mutatott változóra: *mutató-azonosító*^
(pl.: pt^[12] := 1)

Mutató típusú konstans: Nil.
A Nil nem mutat sehová. (Pl. láncolt lista végének a jelzésére használhatjuk).

Műveletek:
Címe: @
Egyenlőség vizsgálat: =, <>

További szabványos eljárások, függvények:
Eljárások: Mark, Release
Függvények: MaxAvail, MemAvail, Ofs, Seg

Láncolt listák

A különböző típusú láncolt listák nagyon fontos adatszerkezetek a számítástechnikában. Az egyes adatelemek (rekordok) közötti csatolást mutatókkal valósíthatjuk meg, így a rekordnak van egy mutató típusú mezője, melynek alaptípusa maga a rekord. Az alábbi példában figyeljük meg, hogy a mutató típus deklarálásánál olyan azonosítót használunk, amely a programunkban csak később szerepel. (A Pascal ebben az egy esetben engedi ezt meg.) [12]

```
type Mutato = ^Adatelem;
     Adatelem = record
       Adat: real;
       Kovetkezo: Mutato;
     end;
var Elso, Uj, Aktualis: Mutato;
```

Példák [1]:

1. Fűzzük fel láncolt listára a billentyűzetről beolvasott számokat. Írassuk ki a listát!

```
program Lista1;
uses Crt;
type TMutato = ^TAdatElem;
     TAdatElem = record
       Adat: integer;
       Kovetkezo: TMutato;
     end;
var  Elso, Aktualis, Uj: TMutato;
     a: integer;
begin
  ClrScr;
```

```
  ReadLn(a);
  Elso := nil;
  while a <> 0 do
    begin
      New(Uj);                          {Az Uj
mutatóhoz memóriaterület rendelése}
      Uj^.Adat := a;
      Uj^.Kovetkezo := nil;
      if Elso = nil then
        Elso:=uj                        {Első
rekord, az Elso mutat az Uj-ra}
      else
        Aktualis^.Kovetkezo := Uj;  {Az
Aktualis-hoz csatoljuk az Uj-at}
      Aktualis := Uj;                {Az Aktualis
léptetése}
      ReadLn(a)
  end;
  {A lista megjelenítése}
  WriteLn;
  Aktualis := Elso;
  while Aktualis <> nil do
    begin
      WriteLn(Aktualis^.Adat);
      Aktualis := Aktualis^.Kovetkezo
    end;
  ReadKey
end.
```

2. Fűzzük fel rendezett láncolt listára a billentyűzetről beolvasott számokat. Írassuk ki a listát!

```
program Lista2;
uses Crt;
type TMutato = ^TAdatElem;
     TAdatElem = record
       Adat: integer;
       Kovetkezo: TMutato;
     end;
var  Elso, Aktualis, Uj, Elozo: TMutato;
     Szam: integer;
begin
  ClrScr;
```

```
  Elso := nil;
  ReadLn(Szam);
  while Szam <> 0 do
    begin
      New(Uj);
      Uj^.Adat := Szam;
      Uj^.Kovetkezo := nil;
      {Az Uj rekord helyének a megkeresése a
rendezett listában}
      Aktualis := Elso;
      while (Aktualis <> nil) and
(Aktualis^.Adat < Uj^.Adat) do
        begin
          Elozo := Aktualis;
          Aktualis := Aktualis^.Kovetkezo;
        end;
      {Az Uj rekord beillesztése a listába}
      if Aktualis = Elso then   {Az Elso elem
lesz}
        Elso := Uj
      else                      {Az Elozo mögé,
az Aktualis elé}
        Elozo^.Kovetkezo := Uj;
      Uj^.Kovetkezo := Aktualis;
      ReadLn(Szam)
    end;
  {A lista megjelenítése}
  WriteLn;
  Aktualis := Elso;
  while Aktualis<>nil do
    begin
      WriteLn(Aktualis^.Adat);
      Aktualis := Aktualis^.Kovetkezo;
    end;
  readkey
end.
```

Típusnélküli mutató - Pointer

Deklarálása: POINTER

Egy négy bájtos memóriacímet tartalmaz. A mutatott változónak nincs típusa. A típusnélküli műveleteknél használjuk (pl. adatmozgatás).

A GetMem eljárással foglalhatunk le egy megadott méretű területet a heap-ben a mutatónk számára. Ha a memóriaterületre már nincs szükségünk a FreeMem eljárással azt felszabadíthatjuk [1].

A mutatót ráirányíthatjuk akármilyen címre vagy változóra a @ operátorral vagy az Addr függvénnyel. Pl. p := @tomb1; vagy p := Addr(tomb1) [8].

A Ptr függvénnyel a mutatónak egy tetszőleges memóriacímet adhatunk értékül.

Egy mutató értékadó utasítással egy másik mutató címét is felveheti.

A mutatott változóhoz rendelhetünk típust: *típus***(***mutató-azonosító***^)**.

Hivatkozás a mutatott változóra: *mutató-azonosító*^

Mutató típusú konstans: Nil.

Műveletek:

Címe: @

Egyenlőség vizsgálat: =, <>

További szabványos függvények: MaxAvail, MemAvail, Ofs, Seg.

Példa [1]:

```
1. Mentsünk el egy garfikát egy típus nélküli
állományba, majd olvassuk vissza!
program KepMent;
uses Crt, Graph;
var f: file;          {A típusnélküli állomány,
amelybe mentjük a képet}
    kep: pointer;     {A kép memóriába való
elmentéséhez szükséges mutató}
    d, m: integer;
    meret: word;
    grd, grm, a, b, c, px, py, py1, i: integer;
    x, y: real;
begin
  {Grafika kirajzolása (két függvény
ábrázolása)}
  d := detect;
```

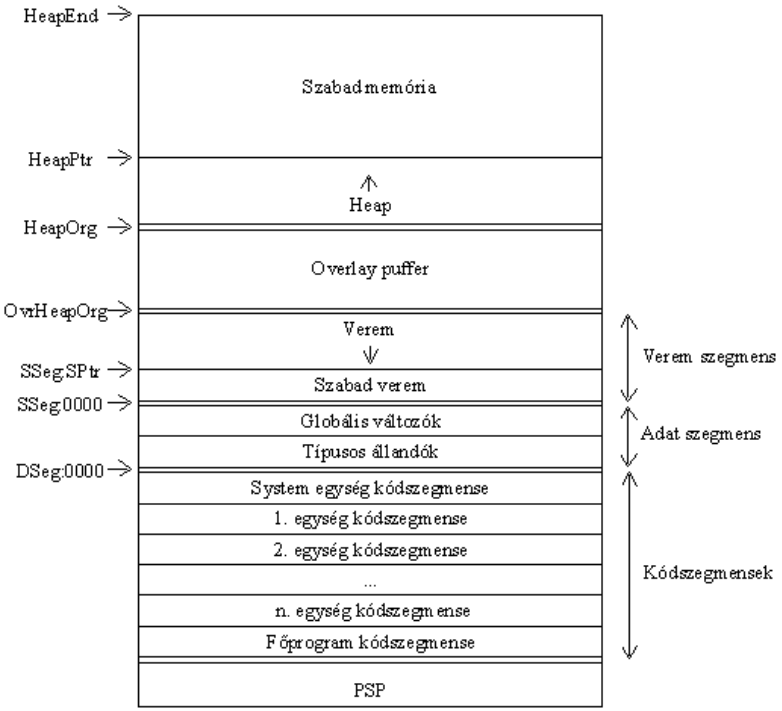
```
  InitGraph(d, m, 'c:\bp\bgi');
  a := 1;
  b := -2;
  c := -3;
  for i := 0 to 640 do PutPixel(i,240,14);
  for i := 0 to 64 do begin
PutPixel(10*i,241,14); PutPixel(10*i,239,14)
end;
  for i := 0 to 480 do PutPixel(320,i,14);
  for i := 0 to 48 do begin
PutPixel(319,10*i,14); PutPixel(321,10*i,14)
end;
  for i := -320 to 320 do
    begin
      x := i/10;
      y := 5*Sin(x/4);
      px := Round(10*x + 320);
      py := Round(-10*y + 240);
      py1 := Round(-10*x + 240);
      if (py < 480) and (py > 0) then
PutPixel(px,py,15);
      if (py1 < 480) and (py1 > 0) then
PutPixel(px,py1,15)
    end;

  Assign(f, 'kepment.dat');
  Rewrite(f);
  {A képet négy részletben tudjuk elmenteni}
  meret := ImageSize(0, 0, 319, 239); {A
negyedkép mérete}
  GetMem(kep, meret);
{Helyfoglalás a típusnélküli mutatónak a heap-
ben}
  GetImage(0, 0, 319, 239, kep^);     {Negyedkép
elmentése a mutatott területre}
  BlockWrite(f, kep^, meret div 128); {A
memóriaterület elmentése a fájlba}
  GetImage(320, 0, 639, 239, kep^);
  BlockWrite(f, kep^, meret div 128);
  GetImage(0, 240, 319, 479, kep^);
  BlockWrite(f, kep^, meret div 128);
  GetImage(320, 240, 639, 479, kep^);
  BlockWrite(f, kep^, meret div 128);
```

```
  FreeMem(kep, meret);                {A
memóriaterület felszabadítása}

  {A kép visszaolvasása}
  ReadKey;
  ClearDevice;
  ReadKey;
  Reset(f);
  GetMem(kep, meret);
{Helyfoglalás a típusnélküli mutatónak a heap-
ben}
  BlockRead(f, kep^, meret div 128);
{Adatmozgatás a fájlból a lefoglalt
memóriaterületre}
  PutImage(0, 0, kep^, copyput);      {A
negyedkép kirajzolása}
  BlockRead(f, kep^, meret div 128);
  PutImage(320, 0, kep^, copyput);
  BlockRead(f, kep^, meret div 128);
  PutImage(0, 240, kep^, copyput);
  BlockRead(f, kep^, meret div 128);
  PutImage(320, 240, kep^, copyput);
  FreeMem(kep, meret);                {A
memóriaterület felszabadítása}

  ReadKey;
end.
```

1 Memóriakezelés

1.1 Memória felosztás

Az alábbi ábrán a Turbo Pascal memóriatérképét láthatjuk. A memória logikai egységeinek az elérését a System unit által deklarált változók (PrefixSeg, HeapOrg, HeapPtr, HeapEnd, OvrHeapOrg, OvrHeapEnd) és függvények (Dseg, SSeg, SPtr) segítik [1].



PSP (Program Segment Prefix)

Az DOS hozza létre a program betöltésekor.

Kódszegmensek

A főprogramnak és minden unitnak van egy kódszegmense. A System egység kódszegmense minden prograhoz automatikusan hozzászerkesztődik. Ezután (fordított sorrendben) következnek a Uses kulcsszó után megadott unitok kódszegmensei. Egy kódszegmens

mérete legfeljebb 64 kB lehet. Nagy program írásakor könnyen ütközhetünk ebbe a korlátba. Ekkor főprogramunk méretét csökkenthetjük saját unitok készítésével, melyeknek mérete természetesen külön-külön 64 kB lehet [11].

Adatszegmens

Programunknak egy adatszegmense van, amely tartalmazza a főprogram és az összes unit globális deklarációit (típusos konstansok, változók). Maximális mérete 64 kB lehet. Ha ez nem elegendő a heap memóriát (ld. mutatók) vagy a verem memóriát (lokális változók) használhatjuk adataink tárolására.

Veremszegmens

LIFO (Last In First Out) szervezésű memória, az utolsónak beírt adatot olvashatom ki belőle először. A renndszer itt tárolja az alprogramok visszatérési címeit, paramétereit, lokális változóit. Méretét a $M fordítási direktívával megadhatjuk, maximálisan 64kB [8].

Heap

Futás közben a heap-ben foglalhatunk le illetve szabadíthatunk fel memóriát a mutatók számára. Alapértelmezés szerint a lefoglalja az egész maradék memóriát. Méretét a $M fordítási direktívával korlátozhatjuk [5].

1.2 Közvetlen memóriahozzáférés

a, A System unitban definiált tömbök segítségével a memória bármely bájtját, szavát (2 bájt) dupla szavát (4 bájt) kiolvashatjuk illetve felülírhatjuk:

Mem[szegmens**:**eltolás]
MemW[szegmens**:**eltolás]
MemL[szegmens**:**eltolás]

Az alábbi példa a képernyő 5. sorának 10. oszlopába kiír egy szines A betűt (ld CRT unit).

```
    var sor, oszlop: byte;
    begin
      sor := 5;
      oszlop := 10;
      Mem[$B800:sor*160+oszlop*2] := Ord('A');
{$B800 a képernyőmemória kezdőcíme}
```

```
      Mem[$B800:sor*160+oszlop*2+1] := $14
{Kék háttéren piros betű}
    end.
```

b, Az **Absolute** direktíva segítségével egy változóhoz közvetlenül rendelhetünk rendelhetünk memóriacímet a deklaráció során. Ez lehet egy abszolút memóriacím (szegmens:eltolás), vagy egy másik változó címe.

Az előző példa egy másik megoldása:

```
    var kepernyo: array[1..25,1..80,1..2]of byte
absolute $B800:0;
    begin
      kepernyo[5, 10, 1] := Ord('A');
      kepernyo[5, 10, 2] := $14
    end.
```

Az alábbi példaprogram a beolvasott sztring hosszát írja ki:

```
    uses Crt;
    var s: string;
        b: byte absolute s;
    begin
      ReadLn(s);
      WriteLn(b);  {A sztring 1. (0. indexű)
bájtját írja ki, mint egy egész számot}
    end.
```

1.3 A Seg és az Ofs függvények

A Seg függvény egy változó vagy egy alprogram kezdőcímének a szegmenscímét, míg az Ofs függvény az ofszet (eltolási) címét adja vissza [1].

Például az alábbi program a *d* és *i* integer változók (2-2 bájt) valamint az *r* rekord (2+2+6 bájt) együttes méretét (azaz 14-et) írja ki bájtokban:

```
    uses Crt;
    var d: integer;
        r: record
             a, b: integer;
             x: real;
         end;
        i, j: integer;
    begin
      WriteLn(Ofs(j) - Ofs(d));
    end.
```

2 Kapcsolat a DOS-szal, a Dos egység

Már a System unit is tartalmaz olyan eljárásokat és függvényeket, melyek a háttértárolók kezelését segítik. Ezek közül az már jó néhányat megismertünk az Állománykezelés fejezetben [4].

További eljárások: ChDir, MkDir, RmDir, GetDir

A Dos egység további segítséget nyújt az állománykezeléshez, valamint a rendszerközeli programozáshoz. A teljesség igénye nélkül nézzünk néhány területet:

Állománykezelés

A FindFirst a paramétereiben meghatározott fájlok közül megkeresi az elsőt, és jellemzőit elhelyezi a (unit által definiált) SearchRec típusú paraméterében. A SearchRec deklarációja:

```
    SearchRec = record
      Fill: array[1..21] of byte;   {A DOS
számára foglalt}
      Attr: byte;                   {Az állomány
attribútuma}
      Time: longint;                {Létrehozási
idő (pakoltan)}
      Size: longint;                {Az állomány
mérete}
      Name: string[12];             {Az állomány
neve}
    end;
```

A FindNext az előzőleg FindFirst eljárással definiált fájlok közül a következőt keresi ki. Mint a Dos unit legtöbb eljárása, így ezen két eljárás is a DosError változóban adja vissza az információt a végrehajtás sikerességéről (értéke 0, ha talált állományt).

Megj.: További eljárások: FSplit, GetFAttr, SetFAttr; függvények: FExpand, FSearch.

Példa [1]:

```
Írjunk programot, amely az aktuális könyvtár pas
kiterjesztésű állományaiban keres egy szót. A
program írja ki, hogy mely állományok és ott
mely sorokban fordul elő a keresett szó!
program Eger;
```

```
uses Graph, Dos, Crt;
var gd, gm, i, j: integer;
    x, y, x1, y1, x2, y2: word;
    r: registers;

procedure Init;
  begin
    r.ax := 0;
    Intr($33, r);
  end;

procedure Be;
  begin
    r.ax := 1;
    Intr($33, r);
  end;

procedure Ki;
  begin
    r.ax := 2;
    Intr($33, r);
  end;

function LeBal: boolean;
  begin
    r.ax := 3;
    Intr($33, r);
    LeBal := r.bx=1;
  end;

function LeJobb: boolean;
  begin
    r.ax := 3;
    Intr($33, r);
    LeJobb := r.bx=2;
  end;

function XKoor: integer;
  begin
    r.ax := 3;
    Intr($33, r);
    XKoor := r.cx;
  end;
```

```
function YKoor: integer;
  begin
    r.ax := 3;
    Intr($33, r);
    YKoor := r.dx;
  end;

begin
  gd := detect;
  InitGraph(gd, gm, 'c:\bp\bgi');
  Init;
  Be;
  {Vonalhuzás}
  repeat
    repeat until LeBal or LeJobb;  {Várakozás}
    Ki;                            {Rajzolás
közben az egér kikapcsolása}
    MoveTo(XKoor, YKoor);          {Rajzolás}
    repeat
      LineTo(XKoor, YKoor)
    until not LeBal;               {Rajzolás
vége}
    Be;
  until LeJobb;                    {Vonalhúzás
vége}

  Delay(500);
  {Téglalapok rajzolása}
  SetWriteMode(xorput);
  repeat
    repeat  until LeBal or LeJobb;
    Ki;
    x1 := XKoor; y1:= YKoor;
    x2 := XKoor; y2 := YKoor;
    repeat
      if (XKoor <> x2) or (YKoor <> y2) then
        begin
         Rectangle(x1, y1, x2, y2);
         x2 := XKoor; y2 := YKoor;
         Rectangle(x1, y1, x2, y2)
        end
    until not LeBal;
```

```
    Be;
  until Lejobb;

end.
```

A rendszerdátum és idő kezelése

A GetDate és a GetTime eljárásokkal az aktuális dátumot és időt kérdezhetjük le, míg a SetDate és SetTime eljárásokkal beállíthatjuk azokat. Például az aktuális idő kiírása:

```
uses Dos;
var ora, perc, mperc, szazadmp: word;
begin
  GetTime(ora, perc, mperc, szazadmp);
  WriteLn(ora,':',perc,':',mperc)
end.
```

Megj.: A GetFTime, SetFTime, PackTime, UnPackTime eljárások egy állomány utolsó módosításának az idejét kezelik. Ezen eljárások használják a unit által definiált DateTime típust .

Megszakítások kezelése

Az Intr eljárás meghívja a paramétereként megadott megszakítást. A megszakítás működését a mikroprocesszor regisztereinek a beállításával szabályozhatjuk, valamint a megszakítás eredményét a regiszterekben kapjuk vissza. A regiszterekhez való hozzáférést a unit által deklarált Registers típus segíti. Az alábbi programocska kiír 10 piros csillagot kék háttéren a képernyőre a képernyőkezelő $10 (hexadecimális 10) BIOS megszakítás segítségével:

```
uses Dos;
var r: registers;
  begin
  r.ah := $9;         {A megszakítás $9
funkciója (karakterek megjelenítése)}
  r.al := ord('*');   {A karakter kódja}
  r.bl := $14;        {A karakter attribútuma
(szín, háttérszín)}
  r.cx := 10;         {Az ismétlések száma}
  Intr($10, r)        {A 10h megszakítás
hívása a fent beállított regiszterekkel}
end.
```

Megj.: Az MsDos eljárás a Dos függvényeket megvalósító $21 megszakítást hívja meg. A GetIntVec illetve SetIntVec eljárással egy megszakításvektort kérdezhetünk le illetve állíthatunk be.

Külső program indítása

Az Exec eljárással futtathatunk egy külső programot. A {$M} fordítási direktívával a heap felső határát le kell csökkenteni úgy, hogy a külső program beférjen a memóriába. Az eljárás előtt és után hívjuk meg a SwapVectors eljárást. Például:

```
    {$M 16000, 0, 0}     {A verem, a heap
minimális és a heap maximális méretének
beállítása}
    uses Crt, Dos;
    begin
      ClrScr;
      SwapVectors;
      Exec('c:\command.com', '/c dir *.pas');
{command.com esetén az első parancssor
paraméter}
      SwapVectors;
{/c kell hogy legyen}
      ReadKey;
      SwapVectors;
      Exec('c:\vc\vc.com', '');
      SwapVectors;
    end.
```

3 Az egér programozása

A BIOS $33 megszakítása tartalmazza az egétkezelő rutinokat. Ezen rutinok bemeneteit és kimeneteit a mikroporcesszor regiszterei alkotják, melyeket a DOS unit Registers típusának segítségével érhetünk el. A megszakítás egyes funkcióit az AX regiszter béállításával érhetjük el. (Akár az első négy funkcióval már kényelmes egérkezelés valósítható meg.) [12]

Az egyes rutinok leírása.

Egérkezelés - $33 megszakítás [1]

```
0.
 Leírás: Az egér inicializálása
 Input : AX  0000h
 Output: Egér státusz
         AX 0000h egér hardver v. meghajtó prg.
nincs installálva
            ffffh sikeres installálás
         BX egérgombok száma
            0000h nem kétgombos
            0002h kétgombos
            0003h Mouse Systems v. kompatibilis
egér

1.
 Leírás: Az egérkurzor megjelenítése.
 Input : AX 0001h
 Output: nincs

2.
 Leírás: Az egérkurzor elrejtése.
 Input : AX 0002h
 Output: nincs

3.
 Leírás: A egérkurzor pozíciójának és az
egérgombok állapotának lekérdezése
 Input : AX 0003h
 Output: BX egérgombok státusza
            0. bit baloldali gomb (1 lenyomva 0
felengedve)
            1. bit jobboldali gomb
```

```
            2. bit középső gomb
         CX x koordináta
         DX y koordináta

4.
 Leírás: Az egér mozgatása adott pontra
 Input : AX 0004h
         CX x koordináta
         DX y koordináta
 Output:

5.
 Leírás: A gomb lenyomásainak száma.
 Input : AX 0005h
         BX a vizsgálandó gomb
            1 baloldali gomb
            2 jobboldali gomb
            4 középső gomb
 Output: AX egérgombok státusza
            0. bit baloldali gomb (1 lenyomva 0
felengedve)
            1. bit jobboldali gomb
            2. bit középső gomb
         BX A figyelt gomb lenyomásainak a száma
az utolsó hívás óta
         CX x koordináta
         DX y koordináta

6.
 Leírás: A gomb felengedéseine száma.
 Input : AX 0006h
         BX a vizsgálandó gomb
            1 baloldali gomb
            2 jobboldali gomb
            4 középső gomb
 Output: AX egérgombok státusza
            0. bit baloldali gomb (1 lenyomva 0
felengedve)
            1. bit jobboldali gomb
            2. bit középső gomb
         BX A figyelt gomb felengedéseinek a
száma az utolsó hívás óta
         CX x koordináta
```

```
         DX y koordináta

7.
 Leírás: A kurzor vizszintes mozgásának
korlátozása.
 Input : AX 0007h
         CX min. x koordináta
         DX max. x koordináta
 Output:

8.
 Leírás: A kurzor függőleges mozgásának
korlátozása.
 Input : AX 0008h
         CX min. y koordináta
         DX max. y koordináta
 Output:

9.
 Leírás: Grafikus módban az egér és a
képernyőmaszk definiálása
 Input : AX 0009h
         BX az egér célpontjának x koordinátája
a maszkban
         CX az egér célpontjának y koordinátája
a maszkban
         ES:DX pointer az egér és a
képernyőmaszkot tartalmazó pufferre
 Output:

10.
 Leírás: Szöveges módban az egér és a
képernyőmaszk definiálása
 Input : AX 000ah
         BX az egérkurzor típusa
            0 szoftver kurzor
            1 hardver kurzor
         CX AND maszk (ha BX=0) vagy a kurzor
első sora (ha BX=1)
         CX OR maszk (ha BX=0) vagy a kurzor
utolsó sora (ha BX=1)
 Output:
```

```
11.
 Leírás: Az egér elmozdulása az utolsó hívás óta
mickey-ben (1 mickey 1/200 inch)
 Input : AX 000bh
 Output: CX vízszintes elmozdulás
         DX függőleges elmozdulás

13.
 Leírás: A fényceruza emuláció bekapcsolása
 Input : AX 000dh
 Output:

14.
 Leírás: A fényceruza emuláció kikapcsolása
 Input : AX 000eh
 Output:


15.
 Leírás: Lépésköz beállítása
 Input : AX 000fh
         CX vízszintes irányban
         DX függőleges irányban
 Output:

16.
 Leírás: A kurzor megjelenítésének letiltása a
monitor meghatározott részén
 Input : AX 0010h
 Output:
```

Célszerű az egyes egérkezelési teendőkre eljárásokat és függvényeket írni (akár egy külön unit-ban elhelyezni), majd ezeket használni a programban. Ezt láthatjuk az alábbi példában is.

Példa [1]:

```
Rajzolhassunk az egérrel (balgomb), a jobboldali
gomb lenyomása után téglalapokat rajzolhassunk,
újabb jobbgomb, kilépés a programból.
program Eger;
uses Graph, Dos, Crt;
var gd, gm, i, j: integer;
```

```
    x, y, x1, y1, x2, y2: word;
    r: registers;

procedure Init;
  begin
    r.ax := 0;
    Intr($33, r);
  end;

procedure Be;
  begin
    r.ax := 1;
    Intr($33, r);
  end;

procedure Ki;
  begin
    r.ax := 2;
    Intr($33, r);
  end;

function LeBal: boolean;
  begin
    r.ax := 3;
    Intr($33, r);
    LeBal := r.bx=1;
  end;

function LeJobb: boolean;
  begin
    r.ax := 3;
    Intr($33, r);
    LeJobb := r.bx=2;
  end;

function XKoor: integer;
  begin
    r.ax := 3;
    Intr($33, r);
    XKoor := r.cx;
  end;

function YKoor: integer;
  begin
```

```
    r.ax := 3;
    Intr($33, r);
    YKoor := r.dx;
  end;

begin
  gd := detect;
  InitGraph(gd, gm, 'c:\bp\bgi');
  Init;
  Be;
  {Vonalhuzás}
  repeat
    repeat until LeBal or LeJobb;  {Várakozás}
    Ki;                            {Rajzolás
közben az egér kikapcsolása}
    MoveTo(XKoor, YKoor);          {Rajzolás}
    repeat
      LineTo(XKoor, YKoor)
    until not LeBal;            {Rajzolás vége}
    Be;
  until LeJobb;                {Vonalhúzás vége}

  Delay(500);
  {Téglalapok rajzolása}
  SetWriteMode(xorput);
  repeat
    repeat  until LeBal or LeJobb;
    Ki;
    x1 := XKoor; y1:= YKoor;
    x2 := XKoor; y2 := YKoor;
    repeat
      if (XKoor <> x2) or (YKoor <> y2) then
        begin
         Rectangle(x1, y1, x2, y2);
         x2 := XKoor; y2 := YKoor;
         Rectangle(x1, y1, x2, y2)
        end
    until not LeBal;
    Be;
  until Lejobb;

end.
```

Saját unit készítése

Az egységek (unitok) előre lefordított programmodulok. Általában egy adott területhez tartozó eljárásokat, függvényeket tartalmaznak, illetve deklarálják az általuk használt konstansokat, típusokat, változókat. Mivel a kódszegmens maximálisan 64 kB lehet, így programunk nagysága is korlátozott. Ha elértük a határt (kb. 2-3000 programsor), akkor programunk egyes részeit saját unitokban helyezhetjük el, melyek külön-külön szintén 64 kB méretűek lehetnek [1].

Az egység felépítése:

Egységfej:

Unit *azonosító***;**

Az azonosítót kell megadnunk a Uses kulcsszó után abban a programban vagy egységben, ahol a unitot használni szeretnénk, továbbá az azonosító legyen a neve az elmentett forrásnyelvű unitnak.

Illesztő rész:

INTERFACE
[**USES** *azonosító* [**,***azonosító*...]**;**]

Továbbá globális deklarációk (konstansok, típusok, címkék, változók, eljárások, függvények), melyeket az egységet használó programokban illetve egységekben is elérhetünk. Az eljárásoknak, függvényeknek itt csak a fejlécei szerepelnek.

Kifejtő rész:

IMPLEMENTATION
[**USES** *azonosító* [**,***azonosító*...]**;**]

Továbbá egység hatáskörű deklarációk (konstansok, típusok, címkék, változók, eljárások, függvények), melyeket csak ebben az egységben érhetünk el. Itt fejtjük ki az Interface részben deklarált eljárásokat, függvényeket is [11].

Inicializáló rész:

[**BEGIN**
[*utasítás* [**;** *utasítás*...]]
END.

A főprogram első utasítása előtt egyszer végrehajtódik, elhagyható.

Ha több egység egymást kölcsönösen használja, akkor mindegyikben a többi egység nevét az implementációs rész Uses kulcsszava után kell megadni.

Ha az egységet elkészítettük, .pas kiterjesztéssel metjük lemezre. Az egységet le kell fordítanunk (a fordítást lemezre kérjük) [12]. A lefordított egység kiterjesztése .tpu lesz.

Példa [1]:

Az egérkezelést megvalósító rutinokat lehelyezhetjük egy unitban.

```
unit EgerUnit;

interface

  procedure Init;
  procedure Be;
  procedure Ki;
  function LeBal: boolean;
  function LeJobb: boolean;
  function XKoor: integer;
  function YKoor: integer;

implementation

uses Dos;
var  r: registers;

procedure Init;
  begin
    r.ax := 0;
    Intr($33, r);
  end;

procedure Be;
  begin
    r.ax := 1;
    Intr($33, r);
  end;

procedure Ki;
  begin
    r.ax := 2;
    Intr($33, r);
  end;
```

```
function LeBal: boolean;
  begin
    r.ax := 3;
    Intr($33, r);
    LeBal := r.bx=1;
  end;

function LeJobb: boolean;
  begin
    r.ax := 3;
    Intr($33, r);
    LeJobb := r.bx=2;
  end;

function XKoor: integer;
  begin
    r.ax := 3;
    Intr($33, r);
    XKoor := r.cx;
  end;

function YKoor: integer;
  begin
    r.ax := 3;
    Intr($33, r);
    YKoor := r.dx;
  end;

end.
```

Gyakorlati feladatok

I. Egy elágazó struktúrával rendelkezö folyamatábrát szerkeszen az összetett függvény értékéi kiszámítására, majd készítsen egy programot, amely az algoritmust realizálja (a függvény argumentum értékét a billentyűzetről kell megadni) [6], [7], [9], [10].

Valtozatok	Függvény	Valtozatok	Függvény
1	$y=\begin{cases} x-2 & x>2.5 \\ 1+x^2 & 0\le x\le 2.5 \\ x\ln\lvert\cos(x)\rvert & x<0 \end{cases}$	2	$y=\begin{cases} \sin(2.3x-1) & x>2.5 \\ 1-3\ln\lvert 1-x\rvert & 0<=x<=2.5 \\ \frac{x^2}{2-x} & x<0 \end{cases}$
3	$y=\begin{cases} \sqrt{(tg(x^2-1))} & x>1 \\ -2x & 0\le x\le 1 \\ e^{\cos(x)} & x<0 \end{cases}$	4	$y=\begin{cases} x^2-3+2.5x^2 & x>12.5 \\ e^x+5+\cos(0.001x) & 0\le x\le 12.5 \\ x^2 & x<0 \end{cases}$
5	$y=\begin{cases} 1+\sqrt{\lvert\cos(x)\rvert} & x>1 \\ x+1 & -0.5\le x\le 1 \\ 1-x^2 & x<-0.5 \end{cases}$	6	$y=\begin{cases} 2.5\cdot x^3+6\cdot x^2-30 & x>1.5 \\ x+1 & 0\le x\le 1.5 \\ x & x<0 \end{cases}$
7	$y=\begin{cases} 1+x & x>14.5 \\ e^{-x} & 3\le x\le 14.5 \\ \cos(x) & x<3 \end{cases}$	8	$y=\begin{cases} \ln\lvert 1+x\rvert & x>3.8 \\ e^{-x} & 2.8\le x\le 3.8 \\ \cos(x) & x<2.8 \end{cases}$
9	$y=\begin{cases} 1+\sqrt{\cos(x)} & x>4 \\ x+1 & 0\le x\le 4 \\ 1-x^2 & x<0 \end{cases}$	10	$y=\begin{cases} e^{-(x+8)} & x>3.61 \\ 1 & 0\le x\le 3.61 \\ \frac{x}{5} & x<0 \end{cases}$
11	$y=\begin{cases} x & x>1.5 \\ 2x^2\sqrt{\lvert\cos(2x)\rvert} & 0\le x\le 1.5 \\ e^{-\cos(3x)} & x<0 \end{cases}$	12	$y=\begin{cases} 1-\sqrt{\cos(2x)} & x>2.5 \\ x^2-x & 1\le x\le 2.5 \\ 1+x^2 & x<1 \end{cases}$

13	$y=\begin{cases}2x & x>4.5\\ 1-\ln\lvert 1-x^2\rvert & 0\le x\le 4.5\\ e^{-x} & x<0\end{cases}$	14	$y=\begin{cases}\sqrt{\ln(x^2-1)} & x>2\\ -2x^3 & 0\le x\le 2\\ e^{\sin(x)} & x<0\end{cases}$
15	$y=\begin{cases}1-\frac{2x^3}{1-x^2} & x>3.5\\ \sqrt{\cos(2x-1)} & 0\le x\le 3.5\\ e^{-\cos(2x)} & x<0\end{cases}$	16	$y=\begin{cases}x+1 & x>2.5\\ 1-x^5 & 0\le x\le 2.5\\ x+\ln\lvert\sin(x)\rvert & x<0\end{cases}$
17	$y=\begin{cases}x-2 & x>2.5\\ 1+x^2 & 0\le x\le 2.5\\ x\ln\lvert\cos(x)\rvert & x<0\end{cases}$	18	$y=\begin{cases}1+3x & x>4.5\\ e^{-2x} & 1\le x\le 4.5\\ \cos(2x) & x<1\end{cases}$
19	$y=\begin{cases}\sqrt{\lvert tg(x^2-1)\rvert} & x>4\\ -2x & 0\le x\le 4\\ e^{\cos(x)} & x<0\end{cases}$	20	$y=\begin{cases}e^{(x+2)} & x>1\\ 1-2x & -1\le x\le 1\\ -\frac{2x^3-3}{5} & x<-1\end{cases}$

II. Hozzon létre egy folyamatábrát és egy programot az egyes feladatok végrehajtására a case operátor segítségével. Minden esetben gondoskodjon a forrásadatok helyességének ellenőrzéséről. Helytelen adatok beírásakor hibaüzenetet kell megjeleníteni [6] ,[7], [9], [10].

1. Hónapszámot adjuk meg (január 1. - január, 2. - február, ...). Nyomtassa ki a megfelelő évszak nevét ("tél", "tavasz" stb.).
2. Tekintettel a hónap (1 - január 2 - február, ...). Nyomtatás a napok száma az adott hónapban nem szökőév (azaz a február 28 nap).
3. Adott egy egész szám a 0 és 9 közötti tartományban van. Nyomtasson egy karakterláncot - a megfelelő szám nevének magyar nyelvű neve (0 - „nulla”, 1 - „egy”, 2 - „kettő”, ...).
4. Adott egy egész szám 1-től 5-ig terjed. Vezese ki - a megfelelő értékelés verbális leírása (1 - „rossz”, 2 - „nem kielégítő”, 3 - „kielégítő”, 4 - „jó”, 5 - „kiváló”).
5. A számok számtani műveletei a következőképpen vannak számozva: 1 - összeadás, 2 - kivonás, 3 - szorzás, 4 - osztás. Figyelembe véve a művelet számát és két számot A és B (A, B ≠0). Hajtsa végre a megadott műveletet a számokon és jelenítse meg az eredményt.

6. A hossz mértékegységei a következőképpen vannak számozva: 1 - deciméter, 2 - kilométer, 3 - méter, 4 - milliméter, 5 - centiméter. Tekintettel a hosszúság egységeinek számára és az L szegmens hosszára ezekben az egységekben (valós szám). Nyomtassa ki ennek a szegmensnek a hosszát méterben.
7. A tömegegységek a következőképpen vannak számozva: 1 kilogramm, 2 milligramm, 3 gramm, 4 tonna, 5 mázsa. Tekintettel a tömegegységek számára és az M testtömeg értékére ezekben az egységekben (valós szám). Nyomtassa ki az adott test tömegét kilogrammban.
8. Állítson össze egy olyan programot, amely a személy életkora szerint (egész számként beírva a billentyűzetről) meghatározza a korosztályhoz tartozó tartozást: 0-13 - fiú; 14-től 20-ig - fiatal férfi; 21-től 70-ig - férfi; 70 év felett - egy idős ember.
9. A lokátor egy adott pontra van irányítva („É” északra, „Ny” nyugatra, „D” délre, „K” keletre) és három parancs egyikét képes elvégezni: -1 - forduljon balra, 2 - forduljon jobbra, 3 - 180 fokos fordulat. A C karakter - a helymeghatározó kezdeti tájolása és az N szám - megadott parancs. A parancs futtatása után nyomtassa ki a lokátor tájolását.
10. A kör elemei a következőképpen vannak számozva: 1 - sugár (R), 2 - átmérő (D), 3 - hossz (L), 4 - kör terület (S). Figyelembe véve ezen elemek számát és jelentését. Nyomtassa ki a kör többi elemének értékeit (ugyanabban a sorrendben). A π értékhez használja a Pi állandót.
11. Adott egy egyenlő szárú derékszögű háromszög, amelynek számozva vannak a következő értékei: 1 - befogó (a), 2 - átfogója (c), 3 - magasság, az átfogóra (h), 4 - a terület (S). Adva van ezen elemek egyike. Nyomtassa ki ennek a háromszögnek a többi elemét (ugyanabban a sorrendben).
12. Két egész számot adunk: D (nap) és M (hónap), amelyek meghatározzák a nem szökő év helyes dátumát. Nyomtassa ki a megadott dátumot megelőző dátum D és M értékeit (például megadva D = 1 M = 1, kimenet D = 31 M = 12; megadva D = 1 M = 3, kimenet D = 28 M = 2; megadva D = 15) M = 12-t kell levonni D = 14 M = 12).
13. Két egész számot adunk: D (nap) és M (hónap), amelyek meghatározzák a nem szökő év helyes dátumát. Nyomtassa ki a megadott időpontot követő dátum D és M értékeit (például megadva D = 1 M = 1, kimenet D = 2 M = 1; megadva D = 31 M = 12, kimenet D = 1 M = 1; megadott D = 28 M = 2 le kell vonni D = 1 M = 3).

14. Adott egy egész szám a 20-69 tartományban, hogy meghatározzák a életkor (években). Nyomtasson egy sort - a megadott életkor szóbeli leírását, biztosítva a szám helyes összehangolását az „év” szóval, például: 20 - „húsz év”, 33 - „harminchárom év”, 41 - „negyven egy év”.
15. Egy egész szám a 100 és 999 közötti tartományban van. Nyomtasson egy sort - ennek a számnak a verbális leírása, például: 256 - "kétszázötven hat", 804 - "nyolcszáznégy".
16. Készítsen egy programot, hogy meghatározza a hónapokban levő napok számát, ha megadva van: N-hónapszám - egész szám 1-től 12-ig, A egész szám 1-ha a szökőév és 0 egyébként.
17. Hozzon létre egy programot, amely kiszámítja a geometriai ábra területét. Az ábra típusát a (c) szimbólum határozza meg: O - kör, T - egyenlő szárú derékszögű háromszög és K - négyzet. A szimbólum után beírt egész szám meghatározza a megfelelő elemet a terület kiszámításához (a körnek a sugára, a háromszögnek a befogó hossza, egy négyzetnek az oldal hossza).
18. Állitson össze olyan programot, amely a hónap sorszáma alapján meghatározza, mely évszakhoz tartozik.
19. Állitson össze olyan programot, amely kinyomtatja az évfolyamot a félév száma szerint, amelyhez a megadott félév tartozik (1. és 2. félév - 1 év, 3. és 4. félév - 2 év, stb.).
20. Adva van egy n – egész szám, amely megfelel a geometriai ábra szögeinek. Készítsen egy programot, amely az megadott n szám szerint kinyomtatja az ábra nevét (például n = 3 - háromszög , n = 5 - ötszög, n> 8 - sokszög). Ha 2-nél kisebb számot ír be, hibaüzenet jelenyen meg.

III. Egy elágazó struktúrával rendelkezö folyamatábrát szerkeszen az összetett függvény értékéi kiszámítására, majd készítsen egy programot, amely az algoritmust realizálja (a függvény argumentum értékét az alábi táblázatböl vegye) [6] ,[7], [9], [10].

Változatok	Függvény		Az argumentum tartománya	Lépték
1	$y = \begin{cases} x - 2 \\ 1 + x^2 \\ x \ln\lvert\cos(x)\rvert \end{cases}$	$x > 2.5$ $0 \le x \le 2.5$ $x < 0$	[-π; π]	π/10
2	$y = \begin{cases} \sin(2.3x - 1) \\ 1 - 3\ln\lvert 1 - x\rvert \\ \frac{x^2}{2 - x} \end{cases}$	$x > 2.5$ $0 <= x <= 2.5$ $x < 0$	[-π/5; 9π/5]	π/3
3	$y = \begin{cases} \sqrt{(tg(x^2 - 1))} \\ -2x \\ e^{\cos(x)} \end{cases}$	$x > 1$ $0 \le x \le 1$ $x < 0$	[-1; 1.5]	0.5
4	$y = \begin{cases} x^2 - 3 + 2.5x^2 \\ e^x + 5 + \cos(0.001x) \\ x^2 \end{cases}$	$x>12.5$ $0 \le x \le 12.5$ $x < 0$	[-5; 10]	0.55
5	$y = \begin{cases} 1 + \sqrt{\lvert\cos(x)\rvert} \\ x + 1 \\ 1 - x^2 \end{cases}$	$x > 1$ $-0.5 \le x \le 1$ $x < -0.5$	[-1.5; 1.5]	0.25
6	$y = \begin{cases} 2.5 \cdot x^3 + 6 \cdot x^2 - 30 \\ x + 1 \\ x \end{cases}$	$x > 1.5$ $0 \le x \le 1.5$ $x < 0$	[-2; 3]	0.5

7	$y = \begin{cases} 1 + x & x > 14.5 \\ e^{-x} & 3 \le x \le 14.5 \\ \cos(x) & x < 3 \end{cases}$		[-1; 15]	1
8	$y = \begin{cases} \ln\lvert 1 + x \rvert & x > 3.8 \\ e^{-x} & 2.8 \le x \le 3.8 \\ \cos(x) & x < 2.8 \end{cases}$		[0; 5]	0.5
9	$y = \begin{cases} 1 + \sqrt{\cos(x)} & x > 4 \\ x + 1 & 0 \le x \le 4 \\ 1 - x^2 & x < 0 \end{cases}$		[-1;4.5]	0.25
10	$y = \begin{cases} e^{-(x+8)} & x > 3.61 \\ 1 & 0 \le x \le 3.61 \\ \frac{x}{5} & x < 0 \end{cases}$		[-π; 2π]	π/5
11	$y = \begin{cases} x & x > 1.5 \\ 2x^2\sqrt{\lvert\cos(2x)\rvert} & 0 \le x \le 1.5 \\ e^{-\cos(3x)} & x < 0 \end{cases}$		[-1; 3]	0.5
12	$y = \begin{cases} 1 - \sqrt{\cos(2x)} & x > 2.5 \\ x^2 - x & 1 \le x \le 2.5 \\ 1 + x^2 & x < 1 \end{cases}$		[0; 3]	0.3
13	$y = \begin{cases} 2x & x > 4.5 \\ 1 - \ln\lvert 1 - x^2 \rvert & 0 \le x \le 4.5 \\ e^{-x} & x < 0 \end{cases}$		[-0.5; 5]	0.5
14	$y = \begin{cases} \sqrt{\ln(x^2 - 1)} & x > 2 \\ -2x^3 & 0 \le x \le 2 \\ e^{\sin(x)} & x < 0 \end{cases}$		[-π/2; π]	π/5

15	$y=\begin{cases}1-\dfrac{2x^3}{1-x^2} & x>3.5\\ \sqrt{\cos(2x-1)} & 0\le x\le 3.5\\ e^{-\cos(2x)} & x<0\end{cases}$		[-0.5; 4.5]	0.25
16	$y=\begin{cases}x+1 & x>2.5\\ 1-x^5 & 0\le x\le 2.5\\ x+\ln\lvert\sin(x)\rvert & x<0\end{cases}$		[-π; 2π]	π/5
17	$y=\begin{cases}x-2 & x>2.5\\ 1+x^2 & 0\le x\le 2.5\\ x\ln\lvert\cos(x)\rvert & x<0\end{cases}$		[-π/2; 2π]	π/4
18	$y=\begin{cases}1+3x & x>4.5\\ e^{-2x} & 1\le x\le 4.5\\ \cos(2x) & x<1\end{cases}$		[-π/2; 2π]	π/5
19	$y=\begin{cases}\sqrt{\lvert tg(x^2-1)\rvert} & x>4\\ -2x & 0\le x\le 4\\ e^{\cos(x)} & x<0\end{cases}$		[-2; 5]	0.5
20	$y=\begin{cases}e^{(x+2)} & x>1\\ 1-2x & -1\le x\le 1\\ -\dfrac{2x^3-3}{5} & x<-1\end{cases}$		[0.8; 2.5]	0.1

IV. Hozzon létre egy programot, amely egész számokat tartalmazó egydimenziós tömbök adatait dolgozza fel. A feldolgozás során használja az elemek permutációit új tömbök létrehozása nélkül. A forrás tömb kitöltése a véletlen számok generátorával végezze. A forrás és a feldolgozott tömböt jelenítse meg [6] ,[7], [9], [10].

1. Korrigálja az A = (a_1, a_2, ..., a_n) tömböt úgy, hogy a tömb elejére írja azt a csoportot, amely a legtöbb egymást követő pozitív elemet tartalmazza. A tömb elemeit a billentyűzetről adja be.
2. Az A = (a_1, a_2, ..., a_n) tömb minden nullával egyenlő elemét helyezze át közvetlenül a tömb maximális eleme után. A tömb elemeit a billentyűzetről adja be.
3. Az A = (a_1, a_2, ..., a_n) tömb minden negatív elemét helyezze át a tömb végébe.
4. Az A = (a_1, a_2, ..., a_n) tömbben távolítsa el a pozitív elemek utolsó csoportját. A csoporton ugyanazon jelű egymást követő elemeinek értünk, amelyek száma legalább 2 vagy több.
5. Az A = (a_1, a_2, ..., a_n) tömbben helyezze át azokat a pozitív elemeket, amelyek a minimális pozitív elem előtt találhatók, a tömb végébe.
6. Az A = (a_1, a_2, ..., a_n) tömbben távolítsa el az egymást követő negatív elemeket amelyek a tömb minimális eleme után találhatók.
7. Az A = (a_1, a_2, ..., a_n) tömbben távolítsa el az összes negatív elemet amelyek a tömb minimális eleme előtt találhatók.
8. Az A = (a_1, a_2, ..., a_n) tömbben távolítson el minden elemet, amely kisebb, mint a tömb azon eleme amely maximálistól balra található.
9. 9.Az A = (a_1, a_2, ..., a_n) tömbbe illesszen be egy új elemet a tömb negatív elemei közül a legnagyobb után.
10. Az A = (a_1, a_2, ..., a_n) tömbben távolítson el minden elemet a minimális pozitív és a maximális negatív elem között.
11. Az A = (a_1, a_2, ..., a_n) tömbben távolítson el minden pozitív elemet, amelyek páros sorszámmal rendelkeznek és a tömb minimális elemét követik.
12. Az A = (a_1, a_2, ..., a_n) tömbben az összes pozitív elemet, a második pozitívtól kezdve, helyezze át a tömb végébe.

13. Az A = (a_1, a_2, ..., a_n) egydimenziós tömbben cserélje ki a legnagyobb számú egyenlő elemet tartalmazó csoportot ennek a tömbnek a maximális elemére. A beállítás után a tömb kevesebb elemet tartalmazhat, mint korábban. Írja be a tömb elemeit a billentyűzetről.
14. Az A = (a_1, a_2, ..., a_n) egydimenziós tömbben helyezze át az elemek azon csoportját, amelyek a tömb legtöbb egymást követő negatív elemét tartalmazza a tömb végébe. Írja be a tömb elemeit a billentyűzetről.
15. Az A = (a_1, a_2, ..., a_n) egydimenziós tömbben helyezze át az összes negatív elemet, amelynek páratlan sorszámmal rendelkeznek a tömb végébe, vagyis tegye az utolsó elemek helyére.
16. Egydimenziós tömbben A = (a_1, a_2, ..., a_n) cserélje ki az összes elemcsoportot, amely több, mint 3 egymást követő negatív elemet tartalmaz a maximális elemmel. Írja be a tömb elemeit a billentyűzetről.
17. Egy egydimenziós tömbben A = (a_1, a_2, ..., a_n) minden pozitív elemet amely páros sorozatszámmal rendelkezik helyezze át a tömb elejére.
18. Az A = (a_1, a_2, ..., a_n) egydimenziós tömbben cserélje ki a legnagyobb számú egyenlő elemet tartalmazó csoportot ennek a tömbnek a maximális elemére. Írja be a tömb elemeit a billentyűzetről.
19. Az A = (a_1, a_2, ..., a_n) egydimenziós tömbben távolítsa el az összes negatív elemet, amelyek a pozitív elemek között helyezkednek el.
20. Egydimenziós A = (a_1, a_2, ..., a_n) tömbből távolítsa el azt a csoportot, amelyben a legtöbb egymást követő pozitív elem van. Írja be a tömb elemeit a billentyűzetről.

V. Hozzon létre egy programot, amely egész számokat tartalmazó mátrixok adatait dolgozza fel. A feldolgozás során használja az elemek permutációit új tömbök létrehozása nélkül. A forrás tömb kitöltése a véletlen számok generátorával végezze. A forrás és a feldolgozott tömböt jelenítse meg [6] ,[7], [9], [10].

1. Adott egy A($n \times (n + 1)$) mátrix és két egydimenziós tömb X = (x_1, ..., x_{n+1}) és Y = (y_1, ..., y_{n+1}), valamint a két természetes szám p és q. Hozzon létre egy új $(n + 1) \times (n + 2)$ méretű mátrixot úgy, hogy a p számú sor után beilleszt egy új sort az A mátrixba $x_1, x_2, ..., x_{n+1}$ elemekkel, majd beilleszt a q számú oszlop után egy új oszlopot $y_1, y_2, ..., y_{n+1}$ elemekkel.
2. Adott A = ($a_1, a_2, ..., a_{10}$) tömb és a B($n \times n$) mátrix. Cserélje ki nullákra a mátrix azon elemeit, amelyek számára az indexek összege páros és megegyeznek az A tömb elemeivel.
3. Adott A = ($a_1, a_2, ..., a_{10}$) tömb és a B($n \times n$) mátrix. A mátrix első sorának elemei növekvő sorrendben vannak rendezve. Szerkesszen egy új, $n \times (n + 1)$ méretű mátrixot az A tömb elemeinek új oszlopának beillesztésével az eredeti mátrixba úgy, hogy a mátrix első sorának rendezése maragyon meg.
4. Adott egy A($n \times m$) mátrix. Szerkesszen egy új mátrixot, amelyet a következő kapuk: az oszlopok átrendezésével: az első az utolsó, a második az utolsó előtti stb.
5. Adott egy A($n \times m$) mátrix, valamint p és q egész számot. Alakítsa át az A mátrixot úgy, hogy az eredeti p sorszámmal rendelkező sor kövesse az eredeti q sorszámút, megőrizve a fennmaradó sorok sorrendjét.
6. Adott egy A($n \times n$) mátrix. Keresse meg és nyomtassa ki azt a mátrix azon sorát, amely a legtöbb páros számot tartalmazza
7. Adott egy A($n \times m$) mátrix. Rendezze át a mátrix sorait következő képen: a sorok átrendezésével - az első az utolsóval, a második az utolsó előttivel stb. A mátrix megengedett transzformációját két sor és két oszlop permutációjának nevezzük.
8. Adott egy n rendű négyzetmátrix. Engedélyezett transzformációk segítségével érje le, hogy a mátrix legkisebb abszolút értékel rendelkező eleme a mátrix jobb alsó sarkában kerüljön.

9. Adott egy A($n \times m$) mátrix. Alakítsa át a mátrixot oly módon, hogy törölje azt a sort és oszlopot, amelyeknek a metszéspontjában a legnagyobb abszolút értékű elem található.
10. Adott egy A($n \times n$) mátrix, amelynek nincsenek megegyező elemei. Keresse meg a legnagyobb elemet a fő- és az oldalátlón, és cserélje meg őket.
11. Alakítson ki egy egy dimenziós tömböt azokból a negatív elemekből amelyek abban a sorban találhatók ahol az A($n \times n$) mátrix legkisebb eleme.
12. Alakítson ki egy egy dimenziós tömböt azokból a pozitív elemekből amelyek abban a sorban találhatók ahol az A($n \times n$) mátrix legnagyobb eleme.
13. Az A ($2 \times n$) mátrix segítségével n pontot definiálunk a síkban úgy, hogy $a_{1,j}$, $a_{2,j}$ a j pont koordinátái. A pontokat szegmensek páronként kötik össze. Keresse meg a legnagyobb szegmens hosszát, és jelenítse meg annak koordinátáit.
14. Adott egy A($n \times n$) mátrix. Számítsa ki az $x_1x_n + x_2x_{n-1} + ... + x_nx_1$ értéket, ahol x_k a mátrix k oszlopának legnagyobb értéke.
15. Adott egy A($n \times n$) mátrix. Keresse meg a sor- és oszlopszámot, amelyek metszéspontjában van egy mátrix legnagyobb eleme található.
16. Távolítsa el az A($m \times n$) mátrixból a legnagyobb mennyiségű nullát tartalmazó sort.
17. Adott egy ($n \times m$) mátrix. Javítsa ki ezt a mátrixot azáltal, hogy törli a sort és az oszlopot, amelyeknek a metszéspontjában található a legmagasabb abszolút értékű elem.
18. Adott egy ($n \times n$) mátrix. Szerkesszen egy egydimenziós tömböt e mátrix pozitív elemeiről, amelyek a fő átló felett találhatók.
19. Adott egy ($n \times n$) mátrix. Szerkesszen egy egydimenziós tömböt e mátrix negatív elemeiről, amelyek a fő átló alatt találhatók.
20. Adott egy egész számokat tartalmazó ($n \times n$) mátrix. Keresse meg a mátrix minden sorában a legkisebb elemeket, és számítsa ki a páros számok mennyiségét közöttük.

Pót feladatok

1. Számok bekérése, sorrendben való kiíratása.
2. Osztás adott pontossággal.
3. Maximumelem kiválasztás tömbből.
4. Buborékrendezés.
5. Lotto – legtöbbször kisorsolt elem.
6. Fájlkezelés – a rejtö.txt fájl tartalmát átmásolni egy másik fájlba.
7. Menürajzolás.
8. Vonal, kör, rúd kirajzoltatása.
9. Egy pontba érintkező koncentrikus körök kirajzolása.
10. Mátrix szorzása (tükrözése).
11. Bekért személyi adatok fájlba mentése.
12. Alakzatok kirajzolása bekért paraméterek szerint.
13. Írjunk programot, amely beolvas két természetes számot, majd kiírja a két szám hányadosát és maradékát az alábbi formában. A program az adatok beolvasása után hagyjon ki egy üres sort.
14. Készítsünk programot, amely bekér egy egész számot, majd mindaddig kér be további egész számokat, amíg nem adjuk meg a 0-t. A program határozza meg és írja ki a beadott egész számok közül a legnagyobbat.
15. Készítsünk programot, amely ki fogja kérdezni a matematikát (két szám összeadását, kivonását és szorzását az <1,10> intervallumból). A két számot és a műveletet a számítógép véletlenszerűen válassza ki. A program akkor fejeződjön be, ha a felhasználó 10 példát kiszámolt helyesen. Rossz válasz esetén kérdezze újra ugyanazt a példát. A program végén írjuk ki az eredményességet százalékokban.
16. Készítsünk programot, amely bekér egész számokat mindaddig, amíg nem adjuk meg a 0-t. A program határozza meg és írja ki a beadott egész számok közül a legkisebbet és a legnagyobbat. (A 0-t ne számítsa bele a beadott számokba, ez csak a bevitel végét jelzi.) A számok beolvasását a 0 végjelig repeat .. until ciklus segítségével valósítsuk meg!
17. Olvassunk be egész számokat 0 végjelig egy maximum 100 elemű tömbbe (a tömböt 100 eleműre deklaráljuk, de csak az elejéből használjunk annyi elemet, amennyit a felhasználó a nulla végjelig beír). A beolvasás után írjuk ki a számokat a beolvasás sorrendjében majd fordítva.
18. Olvassunk be egész számokat 0 végjelig egy maximum 100 elemű tömbbe (a tömböt 100 eleműre deklaráljuk, de csak az

elejéből használjunk annyi elemet, amennyit a felhasználó a nulla végjelig beír).

- Írjuk ki a számokat a beolvasás sorrendjében.
- Írjuk ki az elemek közül a legkisebbet és a legnagyobbat, tömbindexükkel együtt.
- Írjuk ki az elemeket fordított sorrendben.

19. Olvassunk be egész számokat egy 20 elemű tömbbe, majd kérjünk be egy egész számot. Keressük meg a tömbben az első ilyen egész számot, majd írjuk ki a tömbindexét. Ha a tömbben nincs ilyen szám, írjuk ki, hogy a beolvasott szám nincs a tömbben.
20. Állítsunk elő egy 50 elemű tömböt véletlen egész számokból (0-tól 90-ig terjedő számok legyenek).
    - Írjuk ki a kigenerált tömböt a képernyőre.
    - Számítsuk ki az elemek összegét és számtani középértékét.
    - Olvassunk be egy 0 és 9 közötti egész számot, majd határozzuk meg, hogy a tömbben ez a szám hányszor fordul elő.
21. Olvassunk be egy N egész számot (1<=N<=10), majd egy NxN-es kétdimenziós tömbbe generáljunk véletlen egész számokat 10-tol 99-ig. (A tömböt 10x10-esre deklaráljuk, de ennek csak az NxN-es részét használjuk.)
22. Keressünk olyan p prímeket, amelyekre a) p+10 b) p+14 c) p^2+2 is prímszám!
23. Mersenne-prímnek nevezzük a 2^p-1 alakú prímszámot, ha p prím. Keressünk ilyen alakú összetett számot!
24. Fermat-féle számoknak szokás nevezni a 2^{2n+}1 alakú számokat. Keressük meg közülük a prímszámokat!
25. Négyesiker prímeknek nevezzük azokat a számokat, amelyekre p, p+2, p+6 és p+8 is prímszám. Keressünk adott intervallumban ilyen négyesikreket! Miért nem szerepel a sorban a p+4?
26. Keressük meg az összes olyan n természetes számot, amelyekre az n+1, n+3, n+7, n+9, n+13 és n+15 számok mindegyike prímszám! Ha az utolsó feltételt elhagyjuk, akkor találunk-e további számot?
27. Keressünk olyan n természetes számot, amelyre n^2-3 osztható 1-nél nagyobb négyzetszámmal!
28. Keressünk olyan természetes számot, ami után legalább 12 összetett szám következik!

29. Keressünk olyan természetes számpárokat, amelyeknek az euklideszi algoritmusuk 2,3,... lépésből áll!
30. Keressünk olyan természetes számpárt, amelynek legkisebb közös többszöröse egy adott d számmal nagyobb a legnagyobb közös osztónál! Milyen szám lehet a d?
31. Keressük meg adott számig a legtöbb osztójú természetes számot!
32. Erősen összetett számnak nevezzük azokat a természetes számokat, amelyek osztói száma több, mint bármely náluk kisebb természetes szám osztói száma. Készítsünk adott N-ig erősen összetett számot kereső programot!
33. Határozzuk meg adott intervallumban, hogy a számok hány százaléka esetében kisebb a valódi osztók összege a számnál!
34. Suryanarayana nevezte el nagyon tökéletesnek azt a természetes számot, amelyre teljesül, hogy az osztóinak összegének osztóinak összege a szám kétszerese. Keressünk ilyen számokat!
35. Egy számot k-szorosan tökéletesnek nevezünk, ha a nála kisebb pozitív osztóinak összege a szám k-szorosa. Keressünk 2, 3, 4, 5-szörösen tökéletes számokat adott intervallumban! (Ez ideig nem találtak 7-nél nagyobb tökéletességű számot!)

Irodalom jegyzetek

1. http://zeus.nyf.hu/~akos/pascal/pasframe.htm
2. http://prog.ide.sk/pas.php
3. http://progizas123.atw.hu/programozas.htm
4. http://ep128.hu/Ep_Konyv/Szabvanyos_Pascal.htm
5. https://studfile.net/preview/412168/
6. Turbo Pascal: решение сложных задач. – СПб.: БХВ-Петербург, 2006. –208 с.
7. Чеснокова О. В. Турбо Паскаль 7.0. Полное руководство. – М.: НТ Пресс, 2005. – 320 с.
8. Рапаков Г. Г., Ржеуцкая С. Ю. Программирование на языке Pascal. –СПб.: БХВ-Петербург, 2004. – 480 с.
9. Долинский М. С. Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач. – СПб.: Питер, 2005. – 237 с.
10. Культин Н. Turbo Pascal в задачах и примерах. – СПб.: БХВ-Петербург 2010. – 256 с.
11. Моргун А. Справочник по Turbo Pascal для студентов. – Вильямс, 2006. – 608 с.
12. Фаронов В. Turbo Pascal. – СПб.: Питер, 2016. – 368 с.

Ajánlott irodalom

1. Angster Erzsébet: **Programozás tankönyv I.- II**.Akadémia nyomda, Martonvásár, 1999.
2. Baga Edit: **Delphi másképp** Akadémia nyomda, Martonvásár, 1999.
3. Benkő Tiborné: **Programozási feladatok és algoritmusok Delphi rendszerben** Computer Books, Budapest, 2002.
4. Ray Lischner: **Delphi kézikönyv** Kossuth Kiadó, 2001.
5. Paul Kimmel: **Delphi 6 fejlesztők kézikönyve** Panem Kft. , Budapest, 2002.
6. Dr. Fercsik János: **Programozás – Delphi** Dunaújvárosi Főiskola - jegyzet, 2001.
7. Marco Cantú: **Delphi 7 mesteri szinten I. –II.** Kiskapu Kft, 2003.