Pásztor Attila

Algoritmizálás és programozás tankönyv az emeltszintű érettségihez

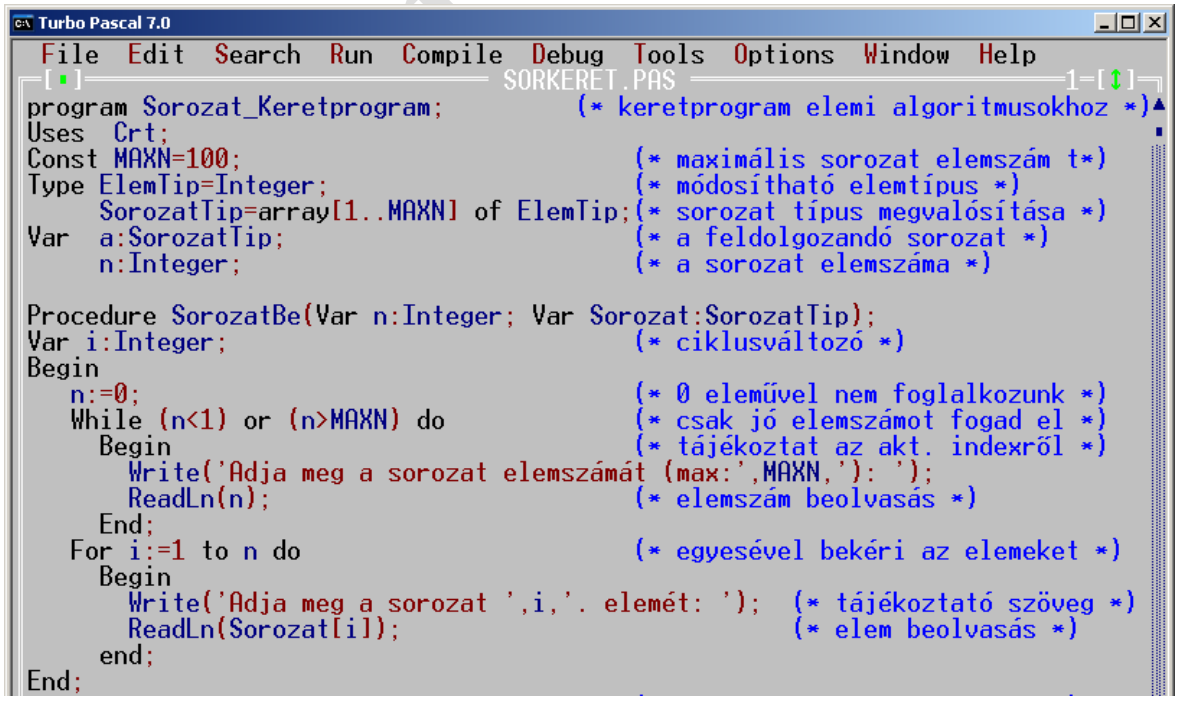
6. Elemi algoritmusok I

Az elemi algoritmusokat programozási tételeknek is szokták nevezni, mert ezek olyan algoritmusok, amelyek gyakran előforduló feladatokra adnak biztosan jó megoldást. Elegendő tehát felismerni, hogy a feladat melyik tétel használatát igényli, és azután annak algoritmusát lehet használni.
Ebben a fejezetben (a keretprogram elkészítése után) 6 olyan algoritmust ismerünk meg, amely sorozathoz egyetlen értéket rendel. Minden algoritmust példával fogok bevezetni.

6.1. Keretprogram

Mielőtt elkezdenénk az elemi algoritmusok ismertetését, megalkotunk egy keretprogramot, melynek főprogramja a képernyőtörlést és a billentyű megnyomásra várást tartalmazza csupán, és megtalálható benne egy sorozat beolvasó és sorozat kiíró eljárás. A sorozat beolvasó eljárás már korábban elkészült, természetesen azt fogjuk használni. Ez az eljárás kéri be a tényleges elemszámot is. Természetesen most is kell a sorozat elemszára egy technikai maximum a deklaráció miatt.

A **SorozatKi** eljárás a sorozat elemeit külön sorba írja ki, ez egy kevéssé takarékos megoldás, kis elemszámra jó. Előnye, hogy az elemek típusának megváltoztatására érzéketlen.
A **SorozatKi1** eljárás annyi elemet ír ki egy sorban, amennyi kifér (vesszővel elválasztva). Az utolsó elem után pontot tesz. Így takarékosabban bánunk a képernyővel. Az eljárás számokra fog működni, az egy sorba kiférő elemeket úgy határozhatjuk meg, hogy a számokat szöveggé alakítjuk, és a szöveg hosszából kiszámoljuk, hogy melyik oszlopra írunk.
Természetesen a két kiíró rutin közül elegendő az egyiket használni egy konkrét példánál.
Mivel a keretprogram jelentős részének algoritmusait a korábbiakban tárgyaltuk, így most csak a pascal kódot közlöm:



```
                                        (* sorozat elemeinek kiírása *)
Procedure SorozatKi(Const n:Integer; Const Sorozat:SorozatTip);
Var i:Integer;                          (* ciklusváltozó *)
Begin
  For i:=1 to n do WriteLn(Sorozat[i]);   (* minden elem külön sorban *)
End;

                                        (* számsorozat takarékos kiírása *)
Procedure SorozatKi1(Const n:Integer; Const Sorozat:SorozatTip);
Var i:Integer;                          (* ciklusváltozó *)
    o:Integer;                          (* oszlop, ahová írtam *)
    s:String;                           (* szöveggé alakítom a számot *)
Begin
   o:=0;                                (* eddig ezen az oszlopon állok *)
   For i:=1 to n do                     (* végigmegyek a sorozat elemein *)
     Begin
       Str(Sorozat[i],s);               (* elemet szöveggé alakítok *)
       o:=o+Length(s)+1;                (* hol állnék, ha kiírom *)
       if o>79 then                     (* nem fér ki ebbe a sorba *)
         Begin
           WriteLn;                     (* új sor elejére megyek *)
           o:=Length(s)+1;              (* kiírás után itt fogok állni *)
           Write(s);                    (* elem kiírás *)
           If i<>n then Write(',')      (* ha nem az utolsó elem, akkor , *)
                   else WriteLn('.');   (* ha az utolsó elem, akkor pont *)
         end
       else                             (* kifér az elem *)
         Begin
           Write(s);                    (* elem kiírása *)
           If i<>n then Write(',')      (* ha nem az utolsó, akkor vessző *)
                   else WriteLn('.');   (* ha az utolsó, akkor pont *)
         end;
     end; (* For *)
End;

Begin (* FŐPROGRAM *)
   ClrScr;                              (* képernyő törlés *)

   SorozatBe(n,a);                      (* sorozat beolvasás teszteléshez *)

   SorozatKi1(n,a);                     (* helytakarékos kiírás *)
   SorozatKi(n,a);                      (* sorozat kiírás *)

   Write('Nyomjon meg egy gombot!');    (* tájékoztató üzenet *)
   Repeat until KeyPressed;             (* kimeneti képernyőn gombra vár*)
End.
 70:37
F1 Help  F2 Save  F3 Open  Alt+F9 Compile  F9 Make  Alt+F10 Local menu
```

Ebben a fejezetben a továbbiakban már nem közlök képernyőképet a Turbo Pascal szerkesztői ablakáról, csupán a tartalmát adom meg.

Ügyeljünk arra, hogy az eljárások deklarációja megelőzze felhasználásukat!

Fontos lenne, hogy minden programot önállóan készíts el, ezzel szerezheted meg a szükséges gyakorlatot!

6.2. Sorozatszámítás

6.2.1. feladat: Egy kosárlabda csapat játékosai magasságainak ismeretében adjuk meg az átlagmagasságot!
6.2.2. feladat: Egy koldus kapott adományaiból határozzuk meg az összbevételét!
6.2.3.feladat: Határozzuk meg egy sorozat elemeinek szorzatát!

A feladatokban közös, hogy a kért értéket az egész sorozaton értelmezett függvény adja (átlag, összeg, szorzat). Ezt a függvényt azonban felbonthatjuk úgy, hogy i-1 elemre ismerve az értékét hogyan határozhatjuk meg i elemre az értékét. Meg kell találnunk tehát azt a függvényt, amelyik i-1 elemhez tartozó értékből és az i-edik elemből meghatározza az eredeti függvény értékét i elemre. Ez általában nagyon egyszerű (a fenti példákban összeadás, szorzás de lehet unió, metszet, egymásután írás,…)

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]

Eljárás Sorozatszámítás(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    S:=f(S,A(i)) [f számolja a függvény értékét az i-1-re meglévő értékből]
  Ciklus vége
Eljárás vége
```

Turbo Pascal kód

```
Procedure SorozatSzamitas(Const N:Integer; Const A:Sorozat;
                          Var s:ElemTípus);
Var i:Integer;
Begin
  s:=S0;                   (* S0 konkrét érték, vagy globális változó *)
  For i:=1 to N do
    Begin
      s:=fuggv(s,Sorozat[i]);    (* fuggv függvényt meg kell írni, *)
    End;                         (* általában függvény sem kell, csak *)
End;                             (* egy operátor: +, *, stb. *)
```

6.2.1.feladat:

Turbo Pascal kód

```
Procedure Atlagmagassag(Const N:Integer; Const A:Sorozat;
                          Var s:ElemTípus);
Var i:Integer;
Begin
  s:=0;
  For i:=1 to N do
    Begin
      s:=s+Sorozat[i];
    End;
  s:=s/N (* feltételezve, hogy az osztás értelmezett *)
End;
```

6.2.2. feladat:

Turbo Pascal kód

```
Procedure Osszbevetel(Const N:Integer; Const A:Sorozat;
                          Var s:ElemTípus);
Var i:Integer;
Begin
  s:=0;
  For i:=1 to N do
    Begin
      s:=s+Sorozat[i];
    End;
End;
```

A 6.2.3. feladat:

Turbo Pascal kód

```
Procedure ElemSzorzat(Const N:Integer; Const A:Sorozat;
                      Var s:ElemTípus);
Var i:Integer;
Begin
  s:=1;                    (* legalább 1-elemű a sorozat *)
  For i:=1 to N do         (* ezért nem ad rossz erredményt *)
    Begin
      s:=s*Sorozat[i];
    End;
End;
```

6.3. Eldöntés

6.3.1. feladat: Döntsük el egy szövegről, hogy van-e benne kérdőjel!
6.3.2. feladat: Döntsük el egy tanuló év végi osztályzatai alapján, hogy kitűnő-e!
6.3.3. feladat: A napi középhőmérsékletekről döntsük el, hogy emelkednek-e!

A sorozathoz rendelt érték most egy logikai érték (igaz/igen, vagy hamis/nem). Mindegyik elemről el lehet dönteni, hogy megfelel-e az elvárt tulajdonságnak.
A feladatok eszerint két csoportba sorolhatók, az egyikben azt kell eldönteni, hogy a sorozatban létezik-e adott tulajdonságú elem, a másikban pedig azt, hogy mindegyik elem ilyen tulajdonságú-e. Az elem tulajdonságának vizsgálatát egy külön függvénnyel fogjuk megvalósítani:

Nézzük az első esetet:
Ha találtunk egy megfelelő tulajdonságú elemet, akkor már nem kell a sorozatot tovább vizsgálni. Haladunk tehát a sorozatban addig, amíg az elem rossz és nem értünk a sorozat végére. Ha túlmentünk a sorozat végén, akkor nincs megfelelő elem.

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      VAN: logikai               [az eredmény]
Függvény:
      Jó: ElemTípus -> logikai   [az elem megfelelő-e]
Eljárás Eldöntés(Konstans N,A, Változó: VAN):
  I:=1
  Ciklus amíg i<=N és nem Jó(A(i))
    i:=i+1                 [ha a vizsgált elem nem jó, továbbmegyek]
  Ciklus vége
  VAN:=(i<=N)              [az i<=N reláció logikai értéke lesz az eredmény]
Eljárás vége
```

TP kód

```
Procedure Eldontes(Const N:Integer; Const A:Sorozat;
                   Var VAN:Boolean);
Var i:Integer;
Begin
  i:=1;
  While (i<=N) and not(Jo(A[i])) do i:=i+1;(*precedencia miatt zárójelezés*)
  VAN:=(i<=N);
End;
```

A **Jó(elem:ElemTípus):logikai** függvény a feladattól függ, általánosan nem írható fel, de általában igen egyszerű.

A 6.3.1. feladatnál kihasználhatjuk, hogy a szöveg típusú változó karakterek sorozataként kezelhető, tehát vizsgálható, hogy az i-edik karaktere kérdőjel-e:

Algoritmus

```
Függvény Jó(Konstans E:ElemTípus): logikai
      [Az ElemTípus most karakter ]
      [A karakterről vizsgálható, hogy kérdőjel-e ]
  Jó:=(E='?');
Függvény vége.
```

TP kód

```
Function Jo(Const E:ElemTipus): Boolean;
      [mejegyzés az algoritmusnál]
  Jo:=(E='?');
End;
```

A 6.3.2. feladat felfogható úgy, hogy el kell dönteni, hogy nem létezik nem megfelelő tulajdonságú elem, azaz nincs egyetlen nem ötös jegye sem. A megoldás egyszerűem adódik: két helyen kell tagadást bevezetni. Egyszerre adom meg az eljárást és a függvényt:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      VAN: logikai               [az eredmény]
Függvény Rossz(Konstans E:ElemTípus): logikai
  Rossz:=(E<5);
Függvény vége.
Eljárás Eldöntés(Konstans N,A, Változó: VAN):
  I:=1
  Ciklus amíg i<=N és Rossz(A(i))
    i:=i+1               [ha a vizsgált elem nem jó, továbbmegyek]
  Ciklus vége
  VAN:=(i>N)       [az i>N reláció logikai értéke lesz az eredmény]
Eljárás vége
```

TP kód

```
Function Rossz(Const E:ElemTipus): Boolean;
      [mejegyzés az algoritmusnál]
  Rossz:=(E<5);
End;
Procedure Eldontes(Const N:Integer; Const A:Sorozat;
                   Var VAN:Boolean);
Var i:Integer;
Begin
  i:=1;
  While (i<=N) and Rossz(A[i]) do i:=i+1;(*precedencia miatt zárójelezés*)
  VAN:=(i<=N);
End;
```

A 6.3.3. feladatban nem csupán az összes elemre kell teljesülnie valamilyen tulajdonságnak, hanem egy elem „jósága” egy, vagy több másik elemmel összehasonlítva adható meg. Ilyenkor nem érdemes külön függvényt írni a jóságra. A konkrét feladatban egy elem akkor jó, ha a megelőző kisebb nála (persze a másodiktól indulva), tehát a megoldás:

Algoritmus

```
Eljárás Eldöntés(Konstans N,A, Változó: VAN):
  I:=2
  Ciklus amíg i<=N és (A(i-1)<A(i))
    i:=i+1               [ha a vizsgált elem nem jó, továbbmegyek]
  Ciklus vége
  VAN:=(i>N)       [az i>N reláció logikai értéke lesz az eredmény]
Eljárás vége
```

TP kód

```
Procedure Eldontes(Const N:Integer; Const A:Sorozat;
                   Var VAN:Boolean);
Var i:Integer;
Begin
  i:=2;
  While (i<=N) and (A[i-1]<A[i]) do i:=i+1;(*precedencia miatt zárójelezés*)
  VAN:=(i>N);
End;
```

6.4. Kiválasztás

6.4.1. feladat: Adjuk meg egy természetes számokat tartalmazó sorozat páros elemét, ha tudjuk, hogy biztosan van ilyen!
6.4.2. feladat: Adjuk meg, hogy egy karakterek sorozataként ábrázolt szövegben hányadik karakter szóköz, ha tudjuk, hogy biztosan van!

Ha ezekre a feladatokra az eldöntés tételt alkalmaznánk, akkor mindig igaz választ kapnánk. Most azonban többre vagyunk kíváncsiak:
A 6.4.1. feladatban meg kell adnunk egy megfelelő elemet.
A 6.4.2. feladatban meg kell adnunk egy megfelelő elem sorszámát.
Általában előnyösebb a sorszámot meghatározni, mert abból az elem mindig meghatározható (fordítva általában nem). Nem kell mást tennünk, mint az eldöntés tételt kicsit átalakítani: ha jó elemet talált, akkor a sorszámot (értéket) adja eredményül.

A most közölt változat a sorszámot és az értéket is adja (a szükségtelen kihagyható):

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      SORSZ: egész               [az egyik eredmény]
      ELEM: ElemTípus            [a másik eredmény]

Függvény:
      Jó: ElemTípus -> logikai   [az elem megfelelő-e]

Eljárás Kiválasztás(Konstans N,A, Változó: SORSZ,ELEM):
  I:=1
  Ciklus amíg nem Jó(A(i))
    i:=i+1                [ha a vizsgált elem nem jó, továbbmegyek]
  Ciklus vége
  SORSZ:=i : ELEM:=A(i)   [a sorszám és az érték is kell]
Eljárás vége
```

TP kód

```
Procedure Kivalasztas(Const N:Integer; Const A:Sorozat;
                      Var SORSZ:Integer; ELEM:ElemTipus);
Var i:Integer;
Begin
  i:=1;                              (*not lehet operátor és lehet függvény*)
  While  not(Jo(A[i])) do i:=i+1;    (*precedencia miatt zárójelezés*)
  SORSZ:=i; ELEM:=A[i];
End;
```

A 6.4.1. feladat „Jó” függvénye:

Alg.

```
Függvény Jó(Konstans E:ElemTípus): logikai
      [páros, ha kettővel osztva 0 maradékot ad]
  Jó:=(Maradák(E,2)=0);
Függvény vége.
```

TP kód

```
Function Jo(Const E:ElemTipus): Boolean;
      [mejegyzés az algoritmusnál]
  Jo:=(E DIV 2 = 0);
End;
```

A 6.4.2. feladat „Jó” függvénye:

Alg.

```
Függvény Jó(Konstans E:ElemTípus): logikai
      [Az ElemTípus most karakter, az adott karaktert vizsgáljuk]
  Jó:=(E='?');
Függvény vége.
```

TP kód

```
Function Jo(Const E:ElemTipus): Boolean;
      [mejegyzés az algoritmusnál]
  Jo:=(E='?');
End;
```

6.5. (Lineáris) keresés

6.5.1. feladat: Egy egészekből álló sorozatnak adjuk meg egy negatív elemét, ha van!
6.5.2. feladat: Adjuk meg, hogy egy szövegben hányadik karakter kérdőjel, ha van!

Ezek a feladatok az előző két feladat kérdéseit tartalmazzák: van-e megfelelő elem, és ha van, akkor melyik az. Itt is kíváncsiak lehetünk az értékre (6.5.1.) és a sorszámra is (6.5.2.), az általánosság kedvéért most is mindkét választ megadjuk. A megoldásban induljunk ki az eldöntés tételből, és amennyiben van megfelelő elem, akkor határozzuk meg annak sorszámát és értékét is: Az elnevezésben a „lineáris” szó arra utal, hogy az első elemtől kezdve végigmegyünk a sorozaton ameddig szükséges – lehetséges, hogy az utolsó elemig. Bizonyos esetekre léteznek ennél hatékonyabb keresési algoritmusok is.

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      VAN: logikai               [az eredmény]
      SORSZ: egész               [a megfelelő elem sorszáma, ha van ]
      ELEM: ElemTípus            [a megfelelő elem, ha van]
Függvény:
      Jó: ElemTípus -> logikai   [az elem megfelelő-e]
Eljárás Kereses(Konstans N,A, Változó: VAN,SORSZ,ELEM):
  I:=1
  Ciklus amíg i<=N és nem Jó(A(i))
    i:=i+1                [ha a vizsgált elem nem jó, továbbmegyek]
  Ciklus vége
  VAN:=(i<=N)             [az i<=N reláció logikai értéke lesz az eredmény]
  Ha VAN akkor SORSZ:=i : ELEM:=A(i)
  Elágazás vége
Eljárás vége.
```

TP kód

```
Procedure Kereses(Const N:Integer; Const A:Sorozat;
                  Var VAN:Boolean; Var SORSZ:Integer; Var ELEM:ElemTipus);
Var i:Integer;
Begin
  i:=1;
  While (i<=N) and not(Jo(A[i])) do i:=i+1;  [precedencia miatt zárójelezés]
  VAN:=(i<=N);
  If VAN then
    Begin
      SORSZ:=i; ELEM:=A[i];
    End;
End;
```

A 6.5.1. feladat „Jó” függvénye:

Alg.

```
Függvény Jó(Konstans E:ElemTípus): logikai
      [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Jó:=(E<0);
Függvény vége.
```

TP kód

```
Function Jo(Const E:ElemTipus): Boolean;
      [mejegyzés az algoritmusnál]
  Jo:=(E<0);
End;
```

A 6.5.2. feladat „Jó” függvénye:

Alg.

```
Függvény Jó(Konstans E:ElemTípus): logikai
      [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Jó:=(E='?');
Függvény vége.
```

TP kód

```
Function Jo(Const E:ElemTipus): Boolean;
      [mejegyzés az algoritmusnál]
  Jo:=(E='?');
End;
```

6.6. Megszámolás

6.6.1. feladat: Adjuk meg egy egészeket tartalmazó sorozat pozitív eleminek darabszámát!
6.6.2. feladat: Adjuk meg egy szöveg magánhangzóinak számát!

Végig kell mennünk a sorozat összes elemén, és a jó tulajdonságúakat meg kell számolni (egy új „jó” elem megtalálásakor a darabszámot növelni kell). Megvalósítható lenne sorozatszámítással is: megfelelő elem esetén 1-et adva a darabszámhoz.
Az általános algoritmus:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      DB: egész                  [az eredmény]

Függvény:
      Jó: ElemTípus -> logikai  [az elem megfelelő-e]

Eljárás Megszamolas(Konstans N,A, Változó: DB):
  DB:=0
  Ciklus i:=1-től N-ig
    Ha Jó(A(i)) akkor DB:=DB+1   [ha a vizsgált elem jó, növelem DB-t]
  Ciklus vége
Eljárás vége.
```

TP kód

```
Procedure Megszamolas(Const N:Integer; Const A:Sorozat;
                      Var DB:Integer);
Var i:Integer;
Begin
  DB:=0;
  For i:=1 to N do
    If Jo(A[i]) then DB:=DB+1;
End;
```

A 6.6.1. feladat „Jó” függvénye:

Alg.

```
Függvény Jó(Konstans E:ElemTípus): logikai
      [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Jó:=(E>0);
Függvény vége.
```

TP kód

```
Function Jo(Const E:ElemTipus): Boolean;
      [mejegyzés az algoritmusnál]
  Jo:=(E>0);
End;
```

A 6.6.2. feladat „Jó” függvénye:

Alg.

```
Függvény Jó(Konstans E:ElemTípus): logikai
Konstans maganhangzok='aAeEiIoOíÍöÖüÜóÓőŐúÚéÉáÁűŰ'
  Jó:=BenneVan(E,maganhangzok);
Függvény vége.
```

TP kód

```
Function Jo(Const E:ElemTipus): Boolean;
Const maganhangzok=' aAeEiIoOíÍöÖüÜóÓőŐúÚéÉáÁűŰ';
  Jo:=(Pos(E,maganhangzok)>0);
End;
```

Emlékezzünk arra, hogy a Pos(minta,szöveg) függvény visszaadja a szöveg azon karakterének sorszámát, amelytől kezdve a minta megtalálható, nullát ad vissza, ha a minta nem illeszthető a szövegre.

6.7. Maximumkiválasztás

6.7.1. feladat: Egy sorozatban tároljuk a boltunk napi bevételeit. Mennyi volt a legnagyobb bevétel?
6.7.2. feladat: Egy osztály tanulóinak nevét tároljuk egy sorozatban. Adjuk meg, hogy a névsorban utolsó tanulót!

Ezeknél a feladatoknál is felmerül, hogy két dologra lehetünk kíváncsiak: a maximális elem sorszámára (ebből az érték mindig megkapható), vagy a maximális elem értékére. Az algoritmust úgy készítjük el, hogy mindkét kérdésre választ adjon.
Az eddigi algoritmusok üres sorozatra is helyesen működtek volna (a kiválasztás természetesen nem, mert nem lehet megfelelő elem, ha elem sincs). A szélsőérték feladatoknak akkor van értelmük, ha a sorozat legalább 1 elemű!

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      MAXSORSZ: egész            [az egyik eredmény: a sorszám]
      MAX: ElemTípus             [a másik eredmény: az érték]

Eljárás MaximumKiválasztás(Konstans N,A, Változó: MAXSORSZ:egész;
                           Változó MAX:ElemTípus):
  MAXSORSZ:=1 : MAX:=A(1)
  Ciklus i:=2-től N-ig
    Ha A(i)>MAX akkor MAXSORSZ:=i      [ha a vizsgált nagyobb, akkor]
                      MAX:=A(i)        [ezt tárolom]
  Ciklus vége
Eljárás vége.
```

TP kód

```
Procedure MaximumKivalasztas(Const N:Integer; Const A:Sorozat;
                             Var MAXSORSZ:Integer; Var MAX:ElemTipus);
Var i:Integer;
Begin
  MAXSORSZ:=1; MAX:=A[1];
  For i:=2 to N do
    If A[i]>MAX then
      Begin
        MAXSORSZ:=i; MAX:=A[i];
      End;
End;
```

Megjegyzések:

- Ha a legkisebb elemet keressük, akkor értelemszerűem a relációjelet kell megfordítani.
- Az eldöntés, kiválasztás, keresés tételhez hasonlóan ez is az első megfelelő elemet találja meg.

Gondold át, hogyan kellene módosítani az algoritmust ahhoz, hogy az utolsó megfelelőt adja meg!

6.8. Összefoglalás

Első lépésként elkészítettük a sorozatok beolvasására és kiírására alkalmas keretprogramot.

Ezt követően elemi algoritmusokat írtuk fel és kódoltunk TP nyelven. Az elemi algoritmusok jelentősége az, hogy tudjuk róluk, hogy az adott feladatra biztosan jó megoldást nyújtanak. Így elegendő meghatározni, hogy a feladat megoldásához melyik programozási tétel kell, és már kész is megoldás.

Az első hat elemi algoritmus (programozási tétel) egy sorozathoz egyetlen értéket rendelt. Az algoritmusok elnevezése igen beszédes:

- sorozatszámítás (összegzés, szorzat,…),
- eldöntés,
- kiválasztás,
- keresés,
- megszámolás,
- maximumkiválasztás (szélsőérték kiválasztás).

Próbáld meg az algoritmusokat memorizálni!

6.9. Gyakorló feladatok

Készítsd el a következő feladatsort TP nyelven!

Adott egy N tagú kosárlabda csapat. Testmagasságukat egy N elemű sorozatban tároljuk. A játékosok mezszáma 1 és N közötti, és az adatok a sorozatban a mezszám szerinti indexen találhatók.

1. Milyen magas lenne a „torony”, ha egymás fejére állnának?
2. Van-e közöttük 170 cm-nél kisebb játékos?
3. Tudjuk, hogy van 2m-nél magasabb játékos. Milyen magas, és mi a mezszáma?
4. Ha van 190 cm és 195 cm közötti magasságú játékos, adjuk meg egy ilyen játékos mezszámát és magasságát!
5. Hány 190 cm-nél magasabb játékos van?
6. Adjuk meg a legkisebb és a legnagyobb játékos mezszámát és magasságát!

Valós számokat tárolunk egy N elemű sorozatban. Készíts programot az alábbi feladatokra!

1. Mennyi a számok szorzata?
2. Melyik a legnagyobb érték?
3. Add meg egy páros elemének sorszámát!
4. Hány 5-tel osztható elem van?
5. Van-e 0 elem a sorozatban?

7. Fájlkezelés

Az előző hat programozási tétel kipróbálásánál is láthattuk, hogy sokelemű sorozatok bevitelét kényelmetlen a billentyűzetről elvégezni. A monitoron megjelenő kimenetet fel kell írni, hogy megmaradjon. A megoldás kézenfekvő: olvassuk az adatokat fájlból és írjuk az eredményt is fájlba.
A következő részben csak az érettségihez és a versenyekhez legfontosabb fájlkezelési ismereteket tanuljuk meg, kifejezetten a Turbo Pascal nyelv tulajdonságaira építve.

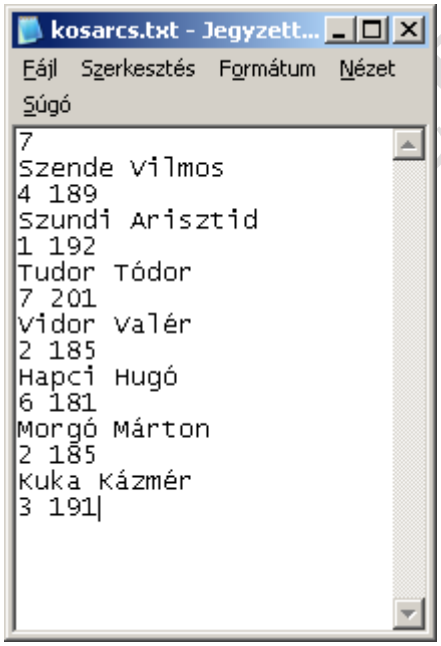
7.1. Adatok beolvasása szekvenciális szövegfájlból

A szekvenciálisan (sorban) olvasható szövegfájlok használatának vitathatatlan előnye, hogy bármilyen egyszerű szövegszerkesztő programmal (jegyzettömb, a TP editora,…) elkészíthető az állomány.

Vegyük a következő példát: Az iskolai kosárlabda csapat tagjainak adatait egy kosarcs.txt nevű szövegfájlban tároljuk a következő módon:

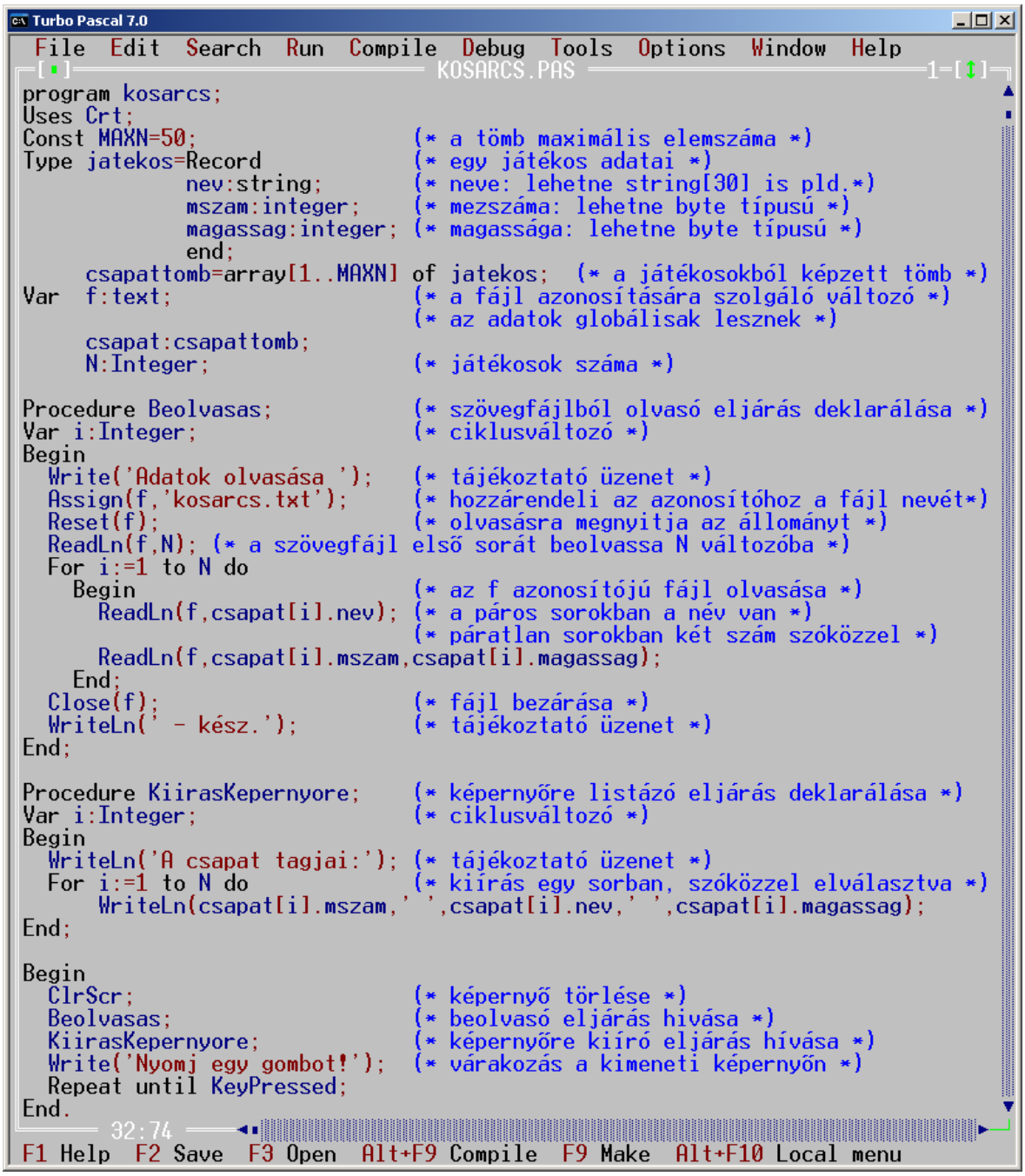
- Az első sorban szerepel a játékosok száma,
- A páros sorokban a játékos neve, majd alatta
- A páratlan sorokban a játékos mezszáma és szóközzel elválasztva a magassága (cm-ben) található.

Íme egy lehetséges állománykép:



Olvassuk be a fájl adatait!
Az adatokat tömbben szeretnénk tárolni, a tömb egy eleme egy játékos mindhárom adatát tartalmazza, tehát rekord lesz.
A beolvasást követően listázzuk ki az adatokat úgy, hogy egy sorban jelenjen meg a játékos mezszáma, neve és magassága.

Feltételezhetjük (az érettségin és a versenyeken is), hogy a szövegfájl helyesen van kitöltve, így annak szerkezetét nem kell ellenőrizni. Garantáljuk, hogy legfeljebb 50 játékos van. A megoldást igyekeztem sok megjegyzéssel ellátni:



A kimeneti képernyő – hasznos, mert ellenőrizhető a beolvasás helyessége:

```
Adatok olvasása  - kész.
A csapat tagjai:
4 Szende Vilmos 189
1 Szundi Arisztid 192
7 Tudor T˘dor 201
2 Vidor ValÚr 185
6 Hapci Hug˘ 181
2 Morg˘ Mßrton 185
3 Kuka KßzmÚr 191
Nyomj egy gombot!
```

Észrevételek:

- A kimeneti képernyőn látszik, hogy a Windows Jegyzettömb karakterkódolása nem egyezik meg a DOS-os Turbo Pascal karakterkódolásával. Érdemes tehát kerülni az ékezetes betűket.
- A megfelelő adattípus kiválasztására nagy figyelmet kell fordítani! Ha megadják az egyes adatok ábrázolását, akkor azt követni kell. Most nem tudtuk, hogy a nevek milyen hosszúak lehetnek, ezért legfeljebb 255 karakter hosszúsági szöveges változóban tároljuk azokat – vélhetően feleslegesen. A játékosok mezszámára is megadhatnak korlátozást (például legfeljebb két számjegyű), és akkor már az egybájtos előjel nélküli egész (byte) is használható. A magasság adat is tárolható így (feltételezve, hogy nincs 255 cm-nél magasabb játékos).

A szövegfájlból olvasás legfontosabb elemei:

Lennie kell egy text típusú változónak (f) a fájlnév hozzárendelésére!
Assign(f,'eleresi_ut\fajl_nev');
hozzárendeli a fájl nevét az azonosítóhoz, az elérési út elhagyható, ha a szövegfájl az aktuális mappában van.
Reset(f);
olvasásra megnyitja az állományt (és az elején áll).
ReadLn(f,szovegtipusu_valtozo);
a sor végéig olvas a változóba (a sorvég karaktereket is).
Read(f,szovegtipusu_valtozo);
a sor végéig olvas a változóba (a sorvég karaktereket nem).
ReadLn(f,szamtipusu_valtozo);
a változóba beolvassa az adott sorban lévő számot..
ReadLn(f,szamtipusu_valtozok);
több szám van egy sorban szóközzel, vagy tabulátorral elválasztva.
Read(f,szamtipusu_valtozo);
egy számot olvas be (szóközig, vagy tabulátorig, vagy sorvégig – de azt nem).
Close(f);
Bezárja az állományt.

Hogyan készítsünk mi szöveges bemenő állományt?

- Számokat külön sorba írhatunk (ENTER-t ütve minden szám után).
- Egy sorban szerepelhet több szám is szóközzel, vagy tabulátorral elválasztva.
- Törekedjünk arra – ha megtehetjük – hogy szöveget külön sorban szerepeltessünk

Azt láttuk, hogy számok többen is lehetnek egy sorban. Vegyíthetők a szövegek és a számok egy sorban? A válasz igen, de több eset lehetséges:

A számok a szöveg előtt vannak, a számokat szóköz választja el egymástól és a szövegtől.

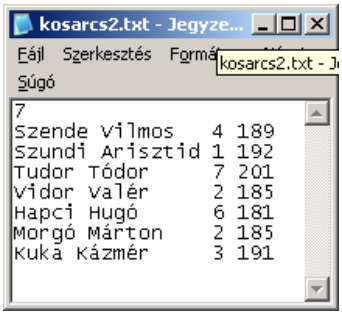
Az első sor beolvasása ReadLn(N);

A többi sor beolvasása:
ReadLn(f,csapat[i].mszam,csapat[i].magassag,csapat[i].nev);

Sajnos a név minden esetben tartalmazni fog egy fölösleges szóközt az első karakteren, melyet törölni kell:

Delete(csapat[i].nev,1,1);

Szeretnénk úgy feltölteni a fájlt, hogy a név legyen legelöl. Semmi akadálya, ha tudjuk, hogy hány karakteren adjuk meg a nevet (fix oszlopszélességet feltételezve).



A mellékelt példában a nevet 17 karakteren adjuk meg, és azt követően jön a két szám.
Ez is beolvasható az alábbi módon (az első sorral már nem foglalkozunk):
A nev mező string[17] típusú legyen!

Beolvasás:
ReadLn(f, csapat[i].nev, csapat[i].mszam,csapat[i].magassag);

A nevek „szóközmentesítése” így történhet:

```
For i:=1 to N do
  Begin
    While Copy(csapat[i].nev,Length(csapat[i].nev,1)=' ' do
        Delete(csapat[i].nev,Length(csapat[i].nev,1);
  End;
```

Adatok mezőszélességének ismeretében másként is beolvashatjuk azokat. A név mezőszélessége legyen 17 karakter, a mezszám mezőszélessége 3 karakter, a magasság mezőszélessége 3 karakter.

A beolvasás (egyetlen sorra):

```
ReadLn(f,sor);  (* egy teljes sort beolvasunk *)
csapat[i].nev:=Copy(sor,1,17);
While Copy(csapat[i].nev,Length(csapat[i].nev,1)=' ' do
        Delete(csapat[i].nev,Length(csapat[i].nev,1);
Val(Copy(sor,18,3),csapat[i].mszam,hibakod);
Val(Copy(sor,21,3),csapat[i].mszam,hibakod);
```

Amennyiben az adatok valamilyen ismert karakterrel vannak elválasztva, akkor egy sor beolvasása:

```
ReadLn(f,sor);   (* egy teljes sort olvasunk be *)
p:=Pos(',',sor);
csapat[i].nev:=Copy(sor,1,p-1);
Delete(sor,1,k);
p:=Pos(',',sor);
Val(Copy(sor,1,p-1),csapat[i].mszam,hibakod);
Delete(sor,1,k);
Val(sor, csapat[i].magassag,hibakod);
```

Ha nem ismerjük a beolvasandó sorok számát, akkor fájl végéig olvashatunk:

```
While not eof(f) do
  Begin
    ReadLn(f,sor);
    (* a sor feldolgozása*)
  End;
```

Egy karakter beolvasása:

```
Read(f,karaktertipusu_valtozo);
```

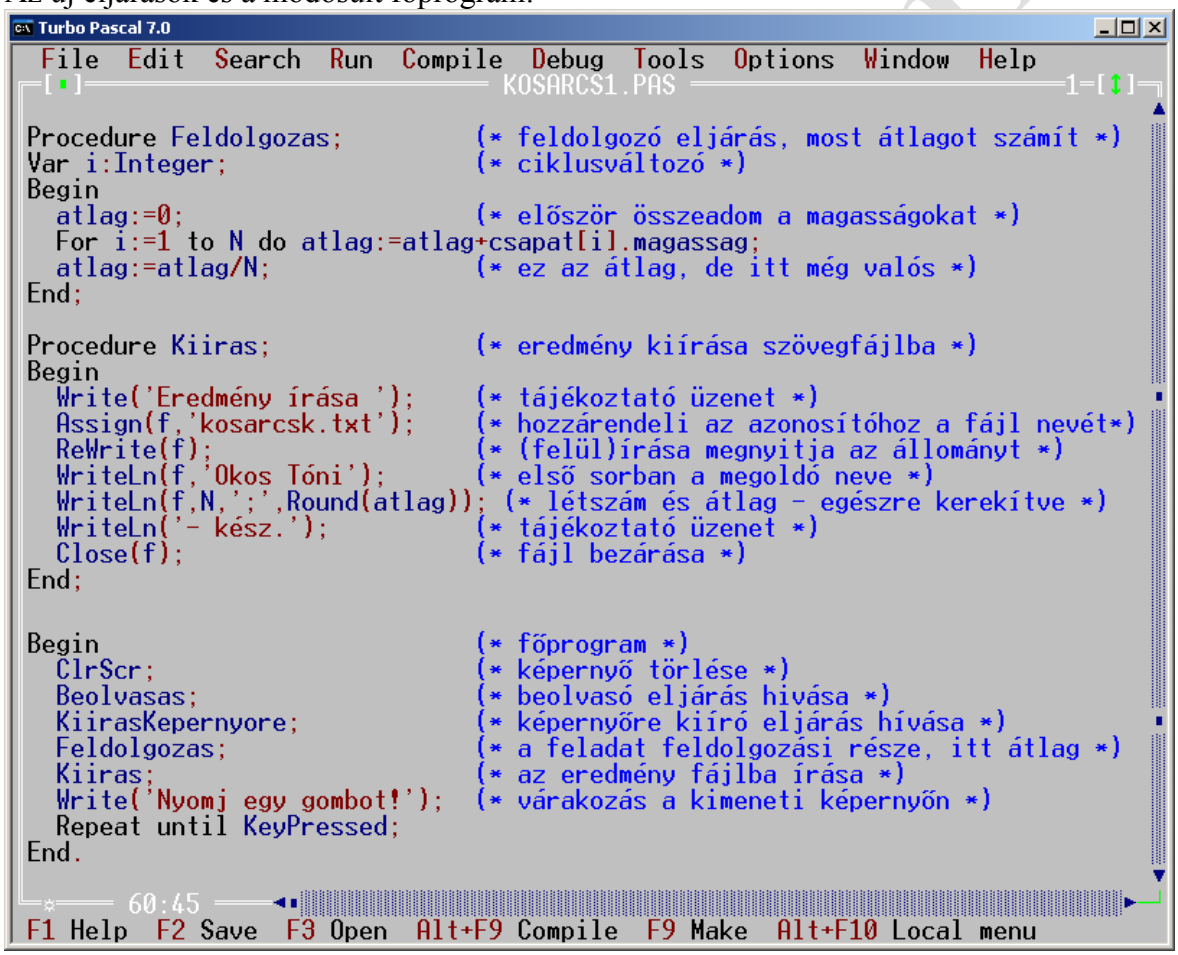
7.2. Adatok írása szövegfájlba

Tegyük fel, hogy az adatbeolvasásra használt példában az a feladat, hogy számoljuk ki a csapattagok átlagmagasságát, és egy kosarcsk.txt nevű egyszerű szöveges állomány első sora tartalmazza a megoldó nevét (Okos Tóni), második sorában pedig pontosvesszővel elválasztva legyen ott a játékosok száma és a magasságuk átlaga.
A korábbi programot egészítsük ki az alábbiakkal:
A főprogram előtt készítsünk egy új eljárást (**Feldolgozas**), mely kiszámolja az átlagot egy **atlag** nevű globális változóban (a változót is deklarálni kell. Készítsünk új eljárást **Kiiras** néven a fájlba írásra.

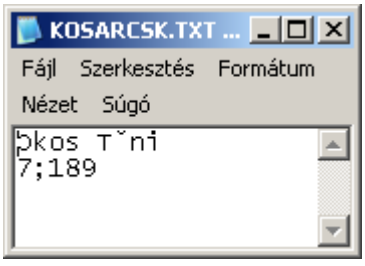
A főprogram deklarációs része kiegészül ezzel:

```
Var  atlag:Integer;              (* a magasságok átlaga egészre kerekítve *)
```

Az új eljárások és a módosult főprogram:



A kimenő szövegfájl tartalma:



A tartalma megfelel a feladat kiírásának.

Az ékezetes betűkkel most is gond van, érdemes használatukat kerülni (olvashatóbb lesz a szöveg.

Ha a kimeneti fájl nem újonnan készintendő, hanem egy meglévőhöz kell hozzáfűzni, akkor a ReWrite(f); helyett az alábbi használandó:

```
Append(f);          (* megnyitás a fájl végére hozzáfűzéshez *)
```

7.3. Összefoglalás

Nagymennyiségű tesztadat beolvasását nem ésszerű billentyűzetről elvégezni, erre a célra leggyakrabban szekvenciális szövegfájlt használunk.

Megismertük a TP fájlkezelő műveleteit:

- Assign(f,fájlnév);
- Reset(f);
- ReWrite(f);
- Append(f);
- Close(f);

A számok és szövegek elhelyezkedésének (az elválasztó karakternek) függvényében több beolvasási módszert is megismertünk.

7.4. Gyakorló feladatok

Készítsd el az első 6 programozási tételhez készített keretprogramot úgy, hogy fájlból vegye az adatokat:

- Az első sorban az elemek száma van, a többi sorban az elemek.
- Az első sorban az elemek száma van, a második sorban szóközzel elválasztva az elemek (első változatban legfeljebb 255 karakter hosszúságban, második változatban legfeljebb 1000 karakter hosszúságban).