Pásztor Attila

Algoritmizálás és programozás tankönyv az emeltszintű érettségihez

9. Összetett feladatok

Az élet ritkán olyan kegyes hozzánk, hogy a megoldandó feladat egyetlen elemi algoritmus alkalmazásával megoldható lenne. Az esetek többségében több elemi algoritmus egymás utáni alkalmazásával juthatunk el az eredményhez. Most nem általános megoldást keresünk, hanem megnézünk néhány példát

9.1. Elemi algoritmusok összeépítése

9.1. feladat: Határozzuk meg egy N elemű egészekből álló sorozat négyzetösszegét!
9.2. feladat: Ha van, adjuk meg egy N elemű egészekből álló sorozat 5. páros elemét!
9.3. feladat: Hány darab maximális eleme van egy N elemű egészekből álló sorozatnak?
9.4. feladat: Határozzuk meg egy N elemű egészekből álló sorozat páros elemeinek összegét!
Vegyük sorra a feladatokat! Az algoritmusok mellett, most a TP kódot is elkészítjük.

A 9.1. feladat megoldható úgy, hogy másolás tétellel előállítjuk az elemek négyzetét tartalmazó sorozatot, majd ezen az új sorozaton alkalmazzuk a sorozatszámítás tételt. A másolás tétellel a sorozatszámítás összeépíthető, így egy algoritmusban elvégezhető a feladat: Induljunk ki a sorozatszámítás tétel összegzésre elkészített változatából:

Algoritmus

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

Eljárás Összegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    S:=S+A(i)
  Ciklus vége
Eljárás vége
```

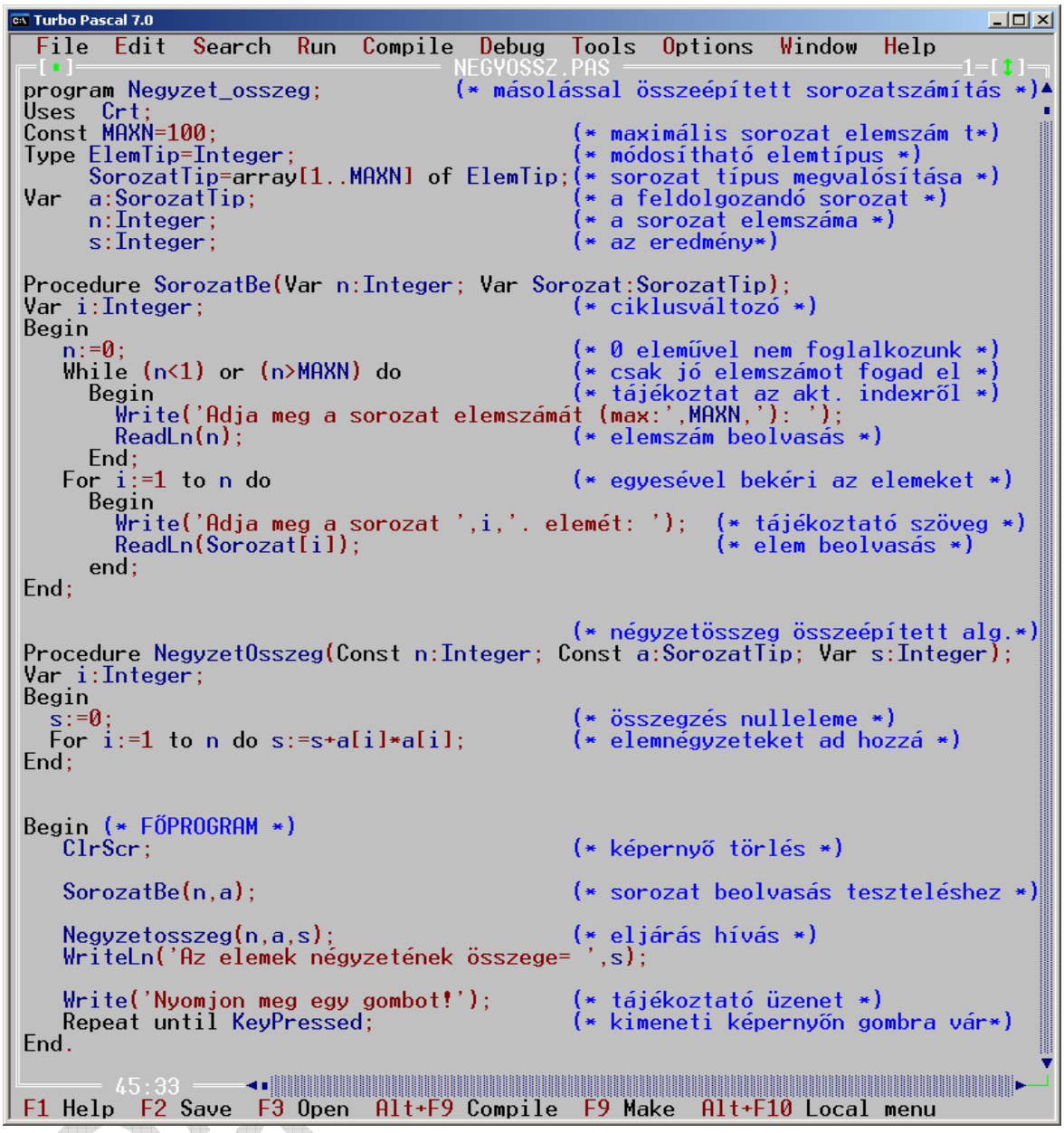
Egyetlen helyen kell módosítani: mivel nem az elemeket, hanem a négyzetüket kell összeadni, ezért az eddigi összeghez nem A(i)-t, hanem A(i)*A(i)-t kell írni. Az összeépített algoritmus:

Algoritmus

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

Eljárás Összegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    S:=S+A(i)*A(i)               [az elem négyzetét adja hozzá]
  Ciklus vége
Eljárás vége
```

Vegyük elő a 6. fejezetben elkészített keretprogramot, és készítsünk egy eljárást benne **„Negyzetosszeg”** néven! A főprogramban olvastassunk be egy sorozatot (billentyűzetről), hívjuk meg a négyzetösszeget előállító eljárást, és írassuk ki az eredményt. A program kódja:



A 9.2. feladatban addig kell páros elemet keresni, amíg az ötödiket meg nem találjuk, vagy elértük a sorozat végét. Ez a feladat tehát a megszámolás és a keresés egymásra építésével oldható meg. Induljunk ki a keresés algoritmusából (a párosság vizsgálat is benne van):

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]

Eljárás Kereses(Konstans N,A, Változó: VAN,SORSZ,ELEM):
  I:=1
  Ciklus amíg i<=N és Maradék(A(i),2)<>0
    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.
```

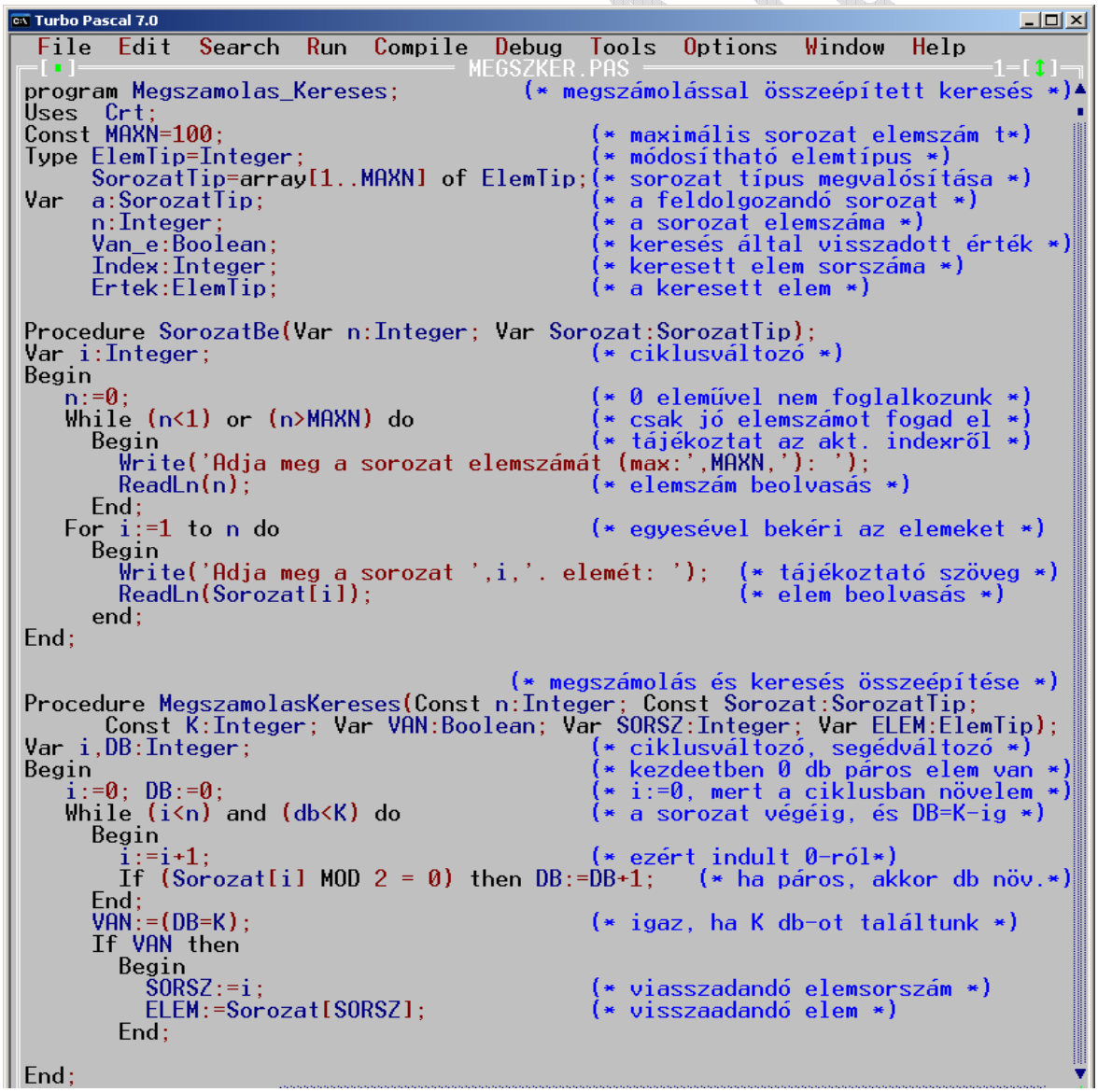
Alakítsuk át az algoritmust úgy, hogy a jó elemeket számolja, és csak az ötödikig keressen:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      K: egész                   [az ennyiedik páros elemet keressük]
      VAN: logikai               [az eredmény]
      SORSZ: egész               [a megfelelő elem sorszáma, ha van ]
      ELEM: ElemTípus            [a megfelelő elem, ha van]

Eljárás Megszámolás_Keresés(Konstans N,A,K Változó: VAN,SORSZ,ELEM):
  I:=0
  Ciklus amíg i<=N és DB<K
    i:=i+1                [ha a vizsgált elem nem jó, továbbmegyek]
    Ha Maradék(A(i),2)=0 akkor DB:=DB+1
  Ciklus vége
  VAN:=(DB=K)                    [a K-adik párosnál állt meg, akkor igaz]
  Ha VAN akkor SORSZ:=i : ELEM:=A(i)
  Elágazás vége
Eljárás vége.
```

Készítsük el ismét a TP kódot is, a 6. fejezetben kidolgozott keretprogram alapján!



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

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

   MegszamolasKereses(n,a,5,Van_e,Index,Ertek);  (* eljárás hívás *)
                                            (* eredmény kijelzés *)
   If Van_e then WriteLn('5. páros elem sorszáma: ',Index,' értéke: ',Ertek)
            else WriteLn('Nincs 5. páros elem.');

   Write('Nyomjon meg egy gombot!');        (* tájékoztató üzenet *)
   Repeat until KeyPressed;                 (* kimeneti képernyőn gombra vár*)
End.
                29:35
F1 Help  F2 Save  F3 Open  Alt+F9 Compile  F9 Make  Alt+F10 Local menu
```

Ellenőrizzük a programot a feltételt kielégítő és ki nem elégítő sorozattal is.

A 9.3. feladat megoldható úgy, hogy megkeresem az elemek maximumát (maximumkiválasztás), majd megszámolom, hogy hány ilyen értékű elem van benne (megszámolás). A két algoritmus azonban összeépíthető: a maximumkiválasztás algoritmusa során az aktuális maximum beállításakor egy darabszám változót állítsunk 1-re. Ha az aktuális maximummal megegyező elemet találunk, akkor növeljük a darabszámot, ha nagyobb elemet találunk, akkor ez lesz az új maximum és a darabszám ismét 1. Nézzük meg először a maximumkiválasztás algoritmusát!

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.
```

Írjuk bele az azonos maximális elemek megszámolását (a sorszámnak nincs értelme)!

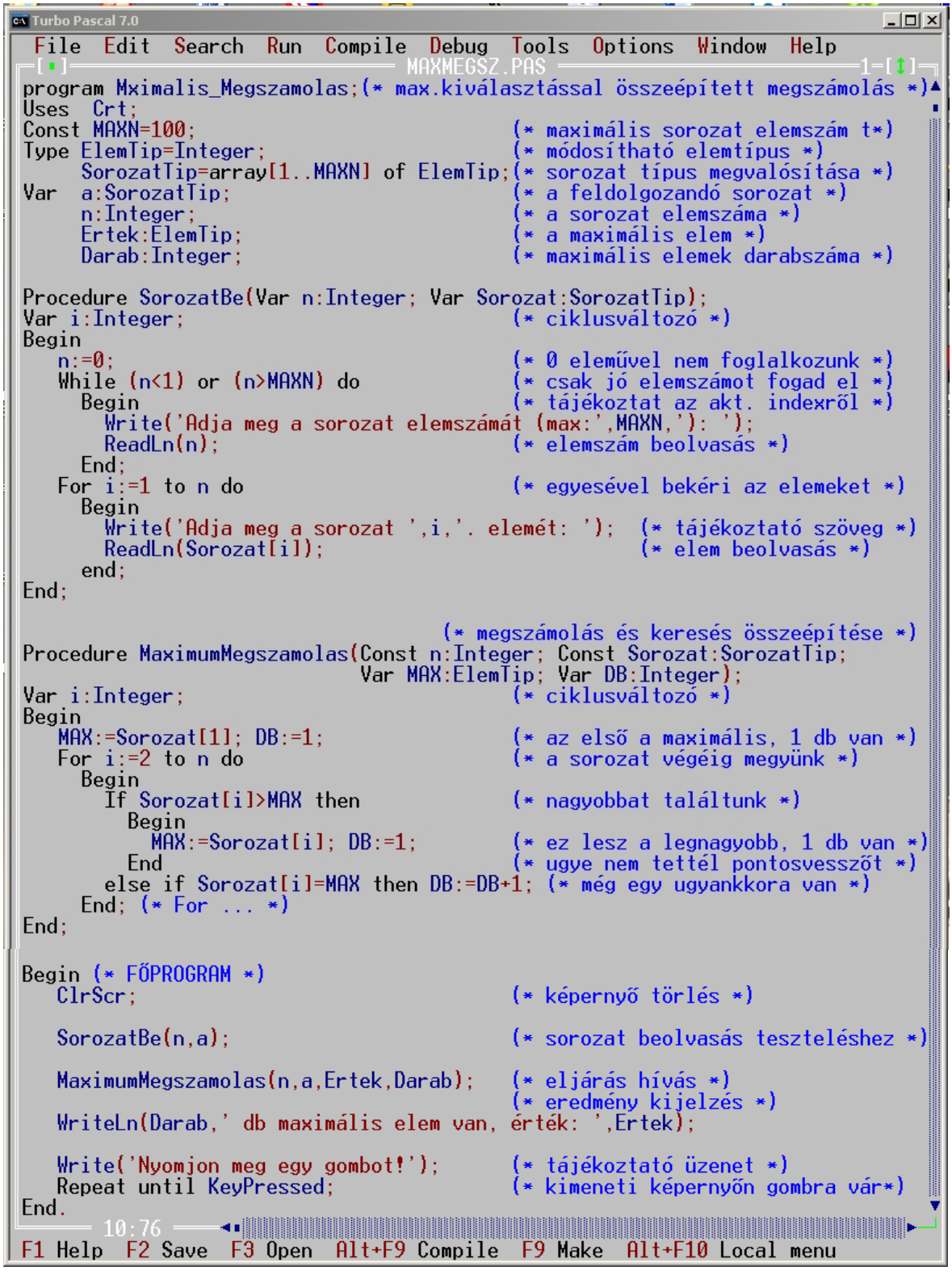
Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma, pozitív]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      MAX: ElemTípus             [a másik eredmény: az érték]
      DB: egész                  [a maximális elemek darabszáma]

Eljárás MaximumMegszámolás(Konstans N,A, Változó MAX:ElemTípus, DB:egész):
  MAX:=A(1) : DB:=1
  Ciklus i:=2-től N-ig
    Elágazás
      A(i)>MAX esetén: MAX:=A(i) : DB:=1     [nagyobbat találtunk]
      A(i)=MAX esetén: DB:=DB+1              [ugyanakkorát találtam]
    Elágazás vége
  Ciklus vége
Eljárás vége.
```

Készítsük el a TP kódot a keretprogram felhasználásával!

A Turbo Pascal kód:



Ellenőrizzük a program működését többféle bemenő sorozattal (1 db maximális elem, több maximális elem,…)!

A 9.4. feladatban az N elemű egészekből álló sorozat páros elemeinek összegét meghatározhatjuk úgy, hogy kiválogatjuk a páros elemeket (kiválogatás) és összeadjuk azokat (sorozatszámítás). Megoldhatjuk a feladatot úgy is, hogy a sorozatszámítás tétel belsejében csak a megfelelő elemeket összegezzük, azaz összeépítjük a kiválogatást a sorozatszámítással.

A sorozatszámítás algoritmusa összegzésre:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]
Eljárás Összegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    S:=S+A(i)
  Ciklus vége
Eljárás vége
```

Módosítsuk az algoritmus ciklusmagját úgy, hogy csak akkor adja hozzá az eddigi összeghez az új elemet, ha az pozitív.

Az összeépített algoritmus általánosan:

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]
Függvény:
      Jó: ElemTípus -> logikai   [az elem megfelelő-e]
Eljárás KiválogatásÖsszegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    Ha Jó(A(i)) akkor S:=S+A(i)
  Ciklus vége
Eljárás vége
```

A „Jó” függvény a konkrét példára:

Alg.

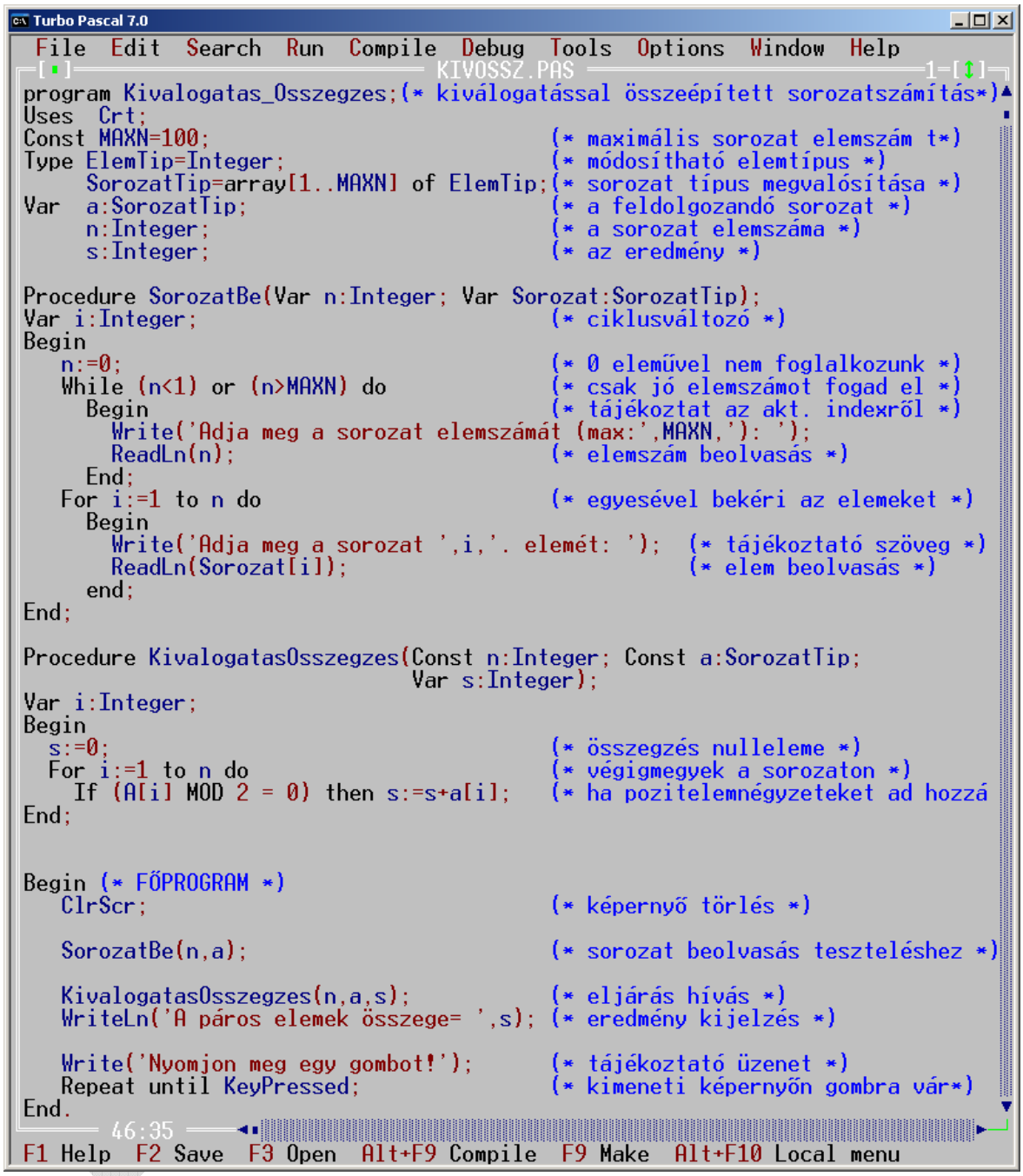
```
Függvény Jó(Konstans E:egész): logikai;
           [ilyen esetben fölösleges a függvény, az eljárásban kezelhető]
  Jó:=(Maradék(E,2)=0)
Eljárás vége.
```

Az egyszerűbb algoritmus érdekében az eljárás ciklusmagjába írjuk be a „Jó” függvényt!

Algoritmus

```
Változó:
      N:egész                    [a sorozat elemszáma ]
      A:Tömb(1..N:ElemTípus)     [N elemű sorozat]
      S: ElemTípus               [egyetlen érték, mely ElemTípusú]
      S0: ElemTípus              [a művelet nulleleme]
Eljárás KiválogatásÖsszegzés(Konstans N,A, Változó: S):
  S:=S0
  Ciklus i:=1-től N-ig
    Ha Maradék(A(i),2)=0 akkor S:=S+A(i)
  Ciklus vége
Eljárás vége
```

Készítsük el ezt a programot is! A kód a következő oldalon látható.



Ellenőrizd a programot csak páros, csak páratlan, illetve páros és páratlan elemeket tartalmazó sorozatra is!

9.2. Összefoglalás

Ebben a fejezetben elemi algoritmusok egymás utáni végrehajtásával megoldható feladatok algoritmusát és pascal kódját készítettük el az elemi algoritmusok összeépítésével.

Természetesen az érettségin, illetve versenyen nem feltétlenül kell hatékony algoritmust készíteni összeépítéssel, de néha az összeépítés jelentősen csökkentheti a kódolás idejét és a memória felhasználást.

9.3. Gyakorló feladatok

1. Egy N elemű sorozatban tároljuk az iskolai kosárlabda csapat játékosainak adatait. Egy-egy játékos adata egy-egy rekordban van (mezszám, név, magasság, életkor). Adjunk választ a következő kérdésekre:
2. Adjuk meg a 16 évnél fiatalabb játékosok átlagéletkorát!
3. Adjuk meg a mezszám szerint rendezett sorozatból a harmadik 180 cm-nél magasabb játékos nevét!
4. A legkisebb magasságú játékosból hány egyforma van?

10. Rekurzió

Az általános iskolai informatika órákon Logo programnyelven készítettünk olyan eljárásokat, amelyek önmagukat hívták meg – más paraméterekkel, és így készítettünk sormintákat, szép rajzokat (fraktálokat).
A matematika órán is tanultunk olyan függvényekről, sorozatokról, amelyek i-edik elemét az i-1 elemre előállított értékből számítjuk ki. Ilyen feladatokkal fogunk most foglalkozni, azaz rekurzívan specifikált függvények algoritmusát és TP kódját fogjuk elkészíteni.

10.1. A faktoriális függvény

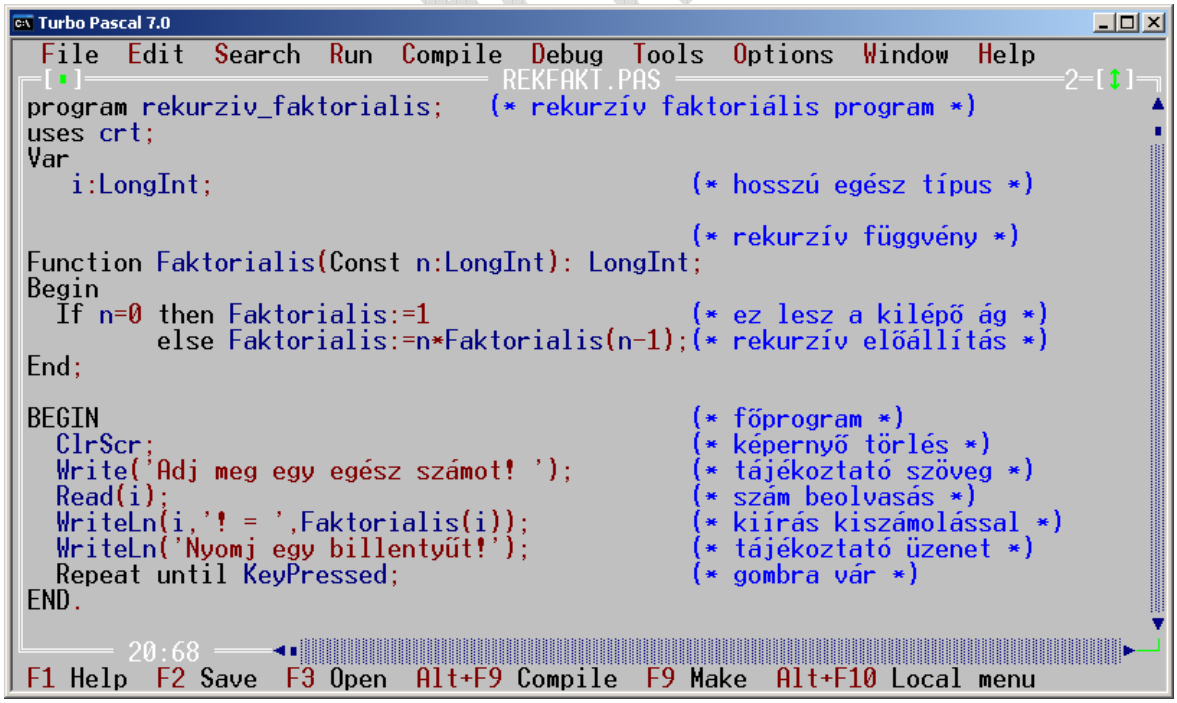
A faktoriális függvény rekurzív megadása:

$$n! = \begin{cases} 1 & ha\ n = 0 \\ n * (n-1)! & ha\ n > 0 \end{cases}$$

Írjunk erre algoritmust!

```
Függvény Faktoriális(n):
  Ha n=0 akkor Faktoriális:=1
         Különben Faktoriális:=Faktoriális(n-1)
  Elágazás vége
Függvény vége.
```

Készítsük al a TP kódot!



Próbáld ki 15-re (15!=2004310016), 17-re (17!=-288522240), tehát már túlcsordult. Negatív számra pedig „stack overflow” hibaüzenetet ad, hiszen végtelen sokszor próbálja önmagát meghívni, mert sohasem éri el n az 1-et. Az utóbbi hiba kivédhető, ha csak nem negatív számra hívjuk meg a függvényt.

10.2. Fibonacci-számok

Érdekes a feladat háttere: Fibonacci azt vizsgálta, hogy egy nyúlpár adott idő alatt mekkora létszámra növekszik, ha az utódok is „mindent megtesznek” a létszám növekedése érdekében. Szerinte egy új generáció növekedését az előző két generáció gyerekei teszik ki (azaz minden nyúlpár egyszerre éppen két utóddal járul hozzá a népességhez, és ezt két egymás utáni alkalommal „gyakorolja”, azaz a harmadik alkalom előtt már fazékba kerülnek.

A Fibonacci-számok definiciója:

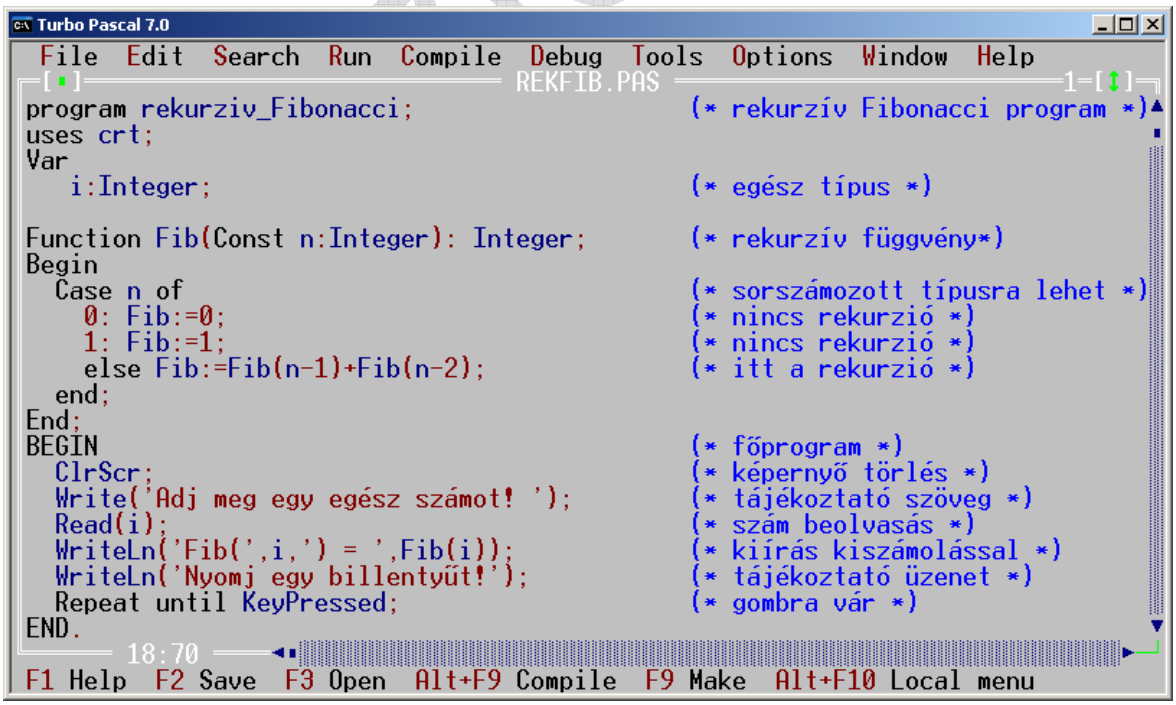
$$Fib(n) = \begin{cases} 0 & ha\ n = 0 \\ 1 & ha\ n = 1 \\ Fib(n-1) + Fib(n-2) & ha\ n > 1 \end{cases}$$

A sorozat néhány első tagja: 0,1,1,2,3,4,5,13,21,34,55,89,144,…

Az algoritmus:

```
Függvény Fib(n):
  Elágazás
    n=0 esetén: Fib:=0
    n=1 esetén: Fib:=1
    egyébként:  Fib:=Fib(n-1)+Fib(n-2)
  Elágazás vége
Függvény vége.
```

Készítsük el a Turbo Pascal kódot!



Megjegyzem, hogy ez a program sem tartalmaz védelmet negatív számok megadására!

10.3. Egy elemi algoritmus rekurzívan

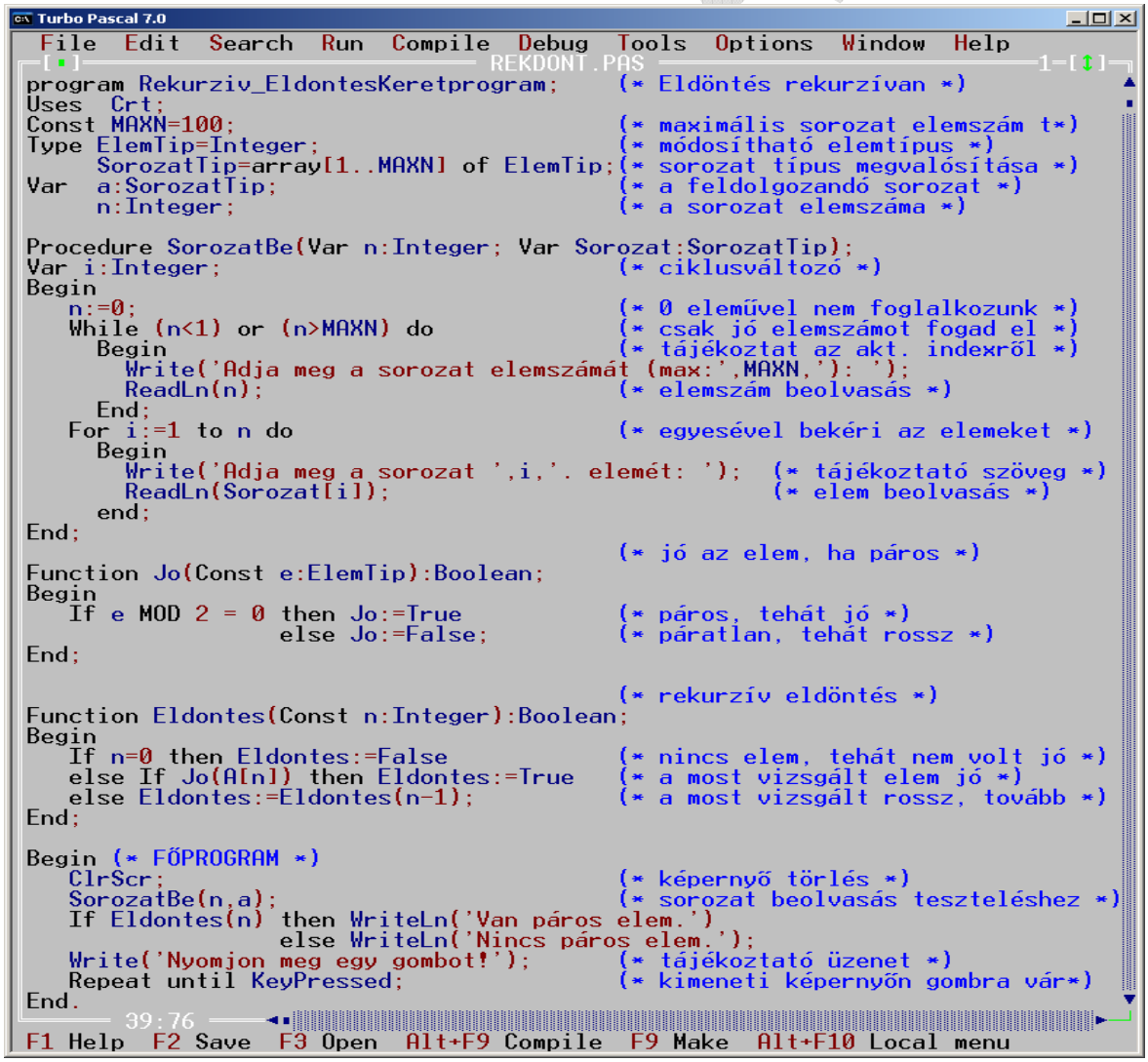
Készítsük el az **eldöntés** algoritmusát rekurzívan! Ha a most vizsgált elem a nulladik, akkor nyilvánvalóan hamis a kimenő érték (egyetlen elem sem volt „jó” tulajdonságú. Ha n nagyobb, mint nulla, és a most vizsgált elem „jó” tulajdonságú, akkor igaz a kimeneti érték, egyébként pedig az előző elemre kell meghívni az algoritmust.
Az eldöntés tétel rekurzív specifikációja (az elemek az A sorozatban vannak:

$$Eldöntés(n) = \begin{cases} hamis & ha\ n = 0 \\ igaz & ha\ n > 0\ és\ Jó(A(n)) \\ Eldöntés(n-1) & egyébként \end{cases}$$

Az eldöntés rekurzív algoritmusa (az algoritmust a sorozat elemszámával kell meghívni):

```
Függvény Eldönt(n):
  Elágazás
    n=0       esetén: Eldönt:=hamis
    Jó(A(n)) esetén: Eldönt:=igaz
    egyébként:       Eldönt(n-1)
  Elágazás vége
Függvény vége.
```

TP kód a keretprogramra építve, példaként páros elem létezését kell eldönteni:



10.4. Gyorsrendezés (QuickSort)

A 8. fejezet 7. pontjában többféle rendező algoritmust ismertünk meg. A leghatékonyabb rendezést azonban nem tudtam bemutatni, mert annak algoritmusa rekurzív.
A módszer a következő: Válogassuk szét a sorozatot úgy, hogy az aktuális elem előtt (tőle balra) a nála nem nagyobb elemek legyenek, míg utána (tőle jobbra) a nála nagyobbak. Mindkét oldalon ezt ismételjük addig, amíg egy elemű lesz a rendezendő sorozat. Szétválogatásra a helyben szétválogatás algoritmusát használjuk.

Az algoritmus:

Algoritmus

```
Változó:
      E:egész                   [aktuális sorozat első elemének indexe]
      U:egész                   [aktuális sorozat utolsó elemének indexe]
      A:Tömb(1..N:ElemTípus)    [N elemű bemenő sorozat]
      K:egész                   [a szétválogatás határa lesz majd]

Eljárás QuickSort(Változó A,E,U):
  Szétválogat(A,E,U,K)
  Ha K-E>1 akkor QuickSort(A,E,K-1)
  Ha U-K>1 akkor QuickSort(A,K+1,U)
Eljárás vége

Eljárás Szétválogat(Változó A,E,U,K):
  K:=E : L:=U : X:=A(K)
  Ciklus amíg K<L
    Ciklus amíg K<L és A(L)>=X
        L:=L-1
    Ciklus vége
    Ha K<L akkor A(K):=A(L) : K:=K+1
                 Ciklus amíg K<L és A(K)<=X
                    K:=K+1
                 Ciklus vége
                 Ha K<L akkor A(L):=A(K) : L:=L-1
    Elágazás vége
  Ciklus vége
  A(K):=X
Eljárás vége.
```

Az eljárás meghívása:
QuickSort(A,1,N)

Az algoritmus tovább finomítható lenne, ha a szétválogatás nem az első elemet választaná, hanem becslés alapján választana egy „közbülsőt”.

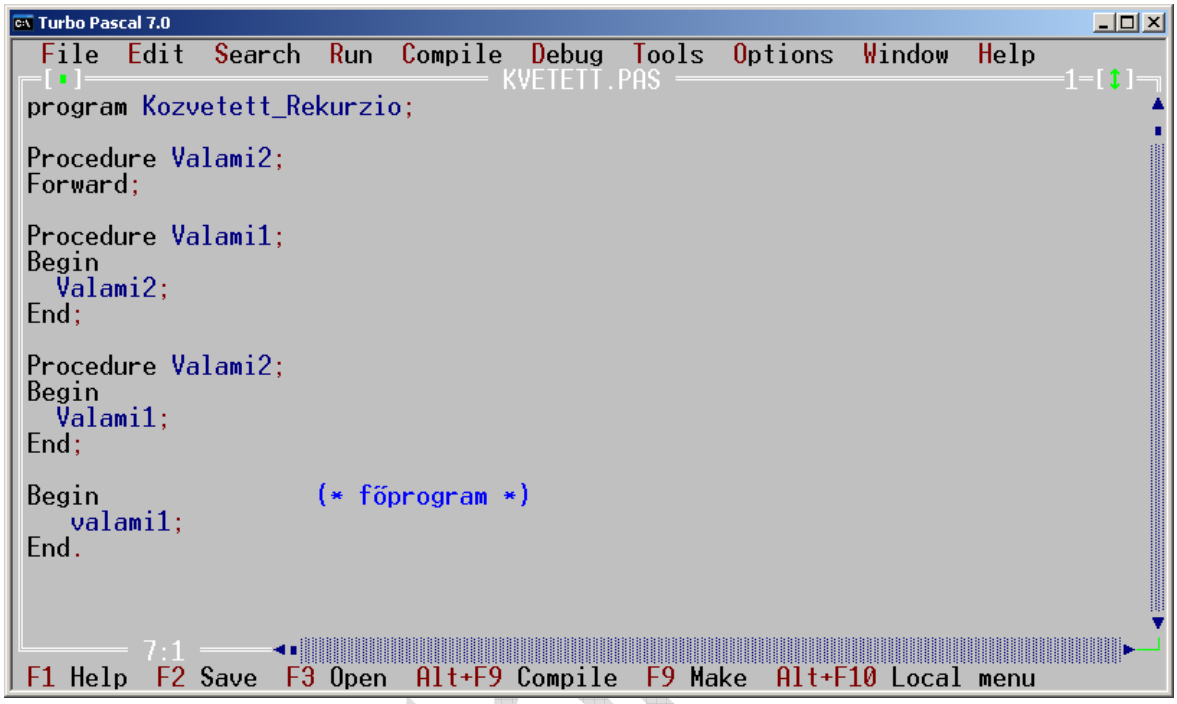
10.5. Közvetett rekurzió

Az eddigi rekurzív eljárások, függvények önmagukat hívták meg, ez a közvetlen rekurzió. Elképzelhető olyan eset, hogy az egyik eljárás meghívja a másikat, és a másik eljárás meghívja az egyiket. Ez a közvetett rekurzió.

Algoritmus szintjén a közvetett rekurzió semmilyen különlegességet nem tartalmaz, de a Turbo Pascal megvalósításnak van egy specialitása: Minden eljárásnak amit használunk előtte deklaráltnak kell lennie. Ez nyilvánvalóan most nem oldható meg. A forrásszövegben másodikként megírt eljárást az első eljárás előtt meg kell adni úgy, hogy a deklarációs

és utasítás rész helyére a Forward szót kell írni. Ebből tudni fogja a fordító, hogy az adott eljárás (függvény) később lesz megadva.

Egy példa a program szerkezetére (nem érdemes végrehajtani, így megállási feltétel nélkül végtelen rekurziót valósít meg).



10.6. Összefoglalás

Ebben a fejezetben rekurzívan megadott matematikai példák algoritmusát készítettük el.
Egy elemi programozási tételt is megadtunk rekurzívan (a többit is meg lehet).
Megismertünk egy rekurzív rendezési algoritmust (QuickSort).
Megtanultuk a közvetett rekurzió megvalósítási technikáját TP környezetben.

10.7. Gyakorló feladatok

1. Keress egy a könyvben nem tárgyalt rekurzív függvényt, vagy sorozatot, és készítsd el algoritmusát és TP kódját!
2. Valósíts meg az ismertetetten kívül még két elemi algoritmust rekurzívan (algoritmus és kód is)!
3. Illeszd be a gyorsrendezés algoritmusát az előző fejezetben elkészített rendezéseket összehasonlító programba. Nézd meg, hogy valóban gyorsabb-e a többinél?