Debreceni Egyetem
Informatikai Kar

Oktatási segédanyag kialakítása webes felületen, könyvtárinformatika témakörben

Témavezető:
Dr. Boda István
Egyetemi docens

Készítette:
Tekse Attila
Informatikus könyvtáros
Szécsi Angéla
Informatikus könyvtáros

Debrecen
2009

Tartalomjegyzék

1. Bevezetés

Szakdolgozatunkban a Debreceni Egyetem Informatikai Karán lévő informatikus-könyvtáros BA képzés negyedik félévében oktatott „Programozás alapjai” című kurzus tananyagát dolgozzuk fel és készítünk belőle online oktatási segédanyagot.

Témaválasztásunkat azzal indokolhatjuk, hogy a kurzus elvégzésével rálátásunk nyílt annak nehézségeire és tapasztaltuk saját évfolyamunkon is annak következményét. Az új rendszerű felsőoktatásban a korábbi öt éves képzés három évre szűkült, így a szak elvégzésére szánt időkeretre megnőtt a tananyagok mennyisége. Ezt a tananyagot nehéz elsajátítani és megérteni rengeteg gyakorlás és odafigyelés nélkül. A feladatok gyakorlására és az anyag könnyebb megértése érdekében hozzuk létre ezt a segédanyagot a következő évfolyamok hallgatói részére. A tárgy alapvető szükségessége abból adódik, hogy a szak webprogramozói szakiránnyal folytatódik, melynek a tantervi hálójának legelső és legfontosabb alapeleme a programozás.

A kurzus során a Pascal és a Java programozási nyelvvel ismerkedhetünk meg, ezért először egy általános, majd később részletes áttekintést adunk ezekről. Az elektronikus formában elkészülő tananyag szorosan kapcsolódik napjaink új oktatási irányához, az e-learning-hez, ezért ennek történetéről, típusairól és helyzetéről átfogóbb bemutatást nyújtunk.

1.1. Programozási nyelvek

A programozási nyelv olyan utasítások sorozata, amellyel közölhetjük a számítógéppel, hogy egy adott feladatot milyen módon végezzen el.
A programozható számítógépek 1940-es évekre tehető megjelenése óta mindig szükség volt a technika, a folyamatosan megjelenő felhasználói/programozói igényeket kielégítő, egyre fejlettebb programozási nyelvek megírására, továbbfejlesztésére.
Mára számos programozási nyelv közül választhatnak a programozók, vagy programozni vágyók. Az eddig megjelent nyelvek struktúrájukból, működésükből és felhasználási módjukból adódóan különböző csoportokba sorolhatók. Az egyik ilyen csoport a fordított nyelvek, melyek fordító (compiler) használatával a forráskódból gépi kódot állítanak elő. Ezt a kódot az operációs rendszer már lényegében közvetlenül tudja futtatni. Ebbe a programnyelv-csoportba tartoznak többek között a Pascal, a C++ és a Fortran nyelvek is. Egy másik csoport az interpreteres nyelvek csoportja, ilyen programozási nyelv a Java, melyben a JVM (Java Virtual Machine) értelmezi a kódot.[1]
A „Programozás alapjai” című kurzus tananyaga a Pascal és a Java nyelvekkel foglalkozik. Ezért a következőkben ezek általános történeti, majd részletes szerkezeti és működési áttekintése következik.

[1] http://hu.wikipedia.org/wiki/Programoz%C3%A1si_nyelv
http://zeus.nyf.hu/~etelka/prognyelvfejl.doc

1.1.1 A Pascal programozási nyelv

1.1.1.1. Története

A Pascal programozási nyelvet a Zürichi Műszaki Egyetem egyik tanára, Niklaus Wirth készítette el 1968-ban. A nyelv vázlatának alapjául az Algol programozási nyelv logikája szolgált. Munkáját Blaise Pascalról, a jelentős 17. századi francia matematikusról és filozófusról nevezte el. A '60-'70-es évek fordulóján, munkatársai segítségével megalkotta eme magas szintű programozási nyelv fordítóprogramját. A nyelv standardizálása (szabványosítása) 1973-ban történt. Innen egyenes út vezetett a logikus és érthető programozási nyelv rohamos elterjedéséhez. Jelenleg is a legkedveltebb nyelvek közé tartozik a programozók körében.

A Pascal nyelv felhasználható utasításainak halmazában kezdetben csak a legegyszerűbb és legpraktikusabb parancsok voltak beleépítve. Innen adódott a nyelv puritánsága és könnyen tanulhatósága. Ám, hogy megfeleljen a kor elvárásainak, a Borland cég felkarolta a nyelvet. Új megoldásként összekapcsolta a fordítóprogramot és fejlesztői környezetet, valamint kiegészítette új utasításokkal és függvényekkel, ezzel sok kényelmi megoldást nyújtva a programozóknak. A programnak számos verziója és javítása megjelent, így a Turbo Pascal jelenleg a 7.0 változatánál tart. Így már képesek vagyunk Windows-os felületből DOS-os alkalmazások készítésére és futtatására. [2]

[2] http://hu.wikipedia.org/wiki/Pascal_(programoz%C3%A1si_nyelv)
http://indy.poliod.hu/program/Pascal/Tankonyv/Tankonyv.htm
http://prog.hu/cikkek/330/Alapok.html

Eredetileg a megírt program egy forrásfájlból áll, melynek kiterjesztése „pas”.

1.1.1.2. Általános szabályai és szerkezete

Mint minden mesterséges nyelvnek, így a Pascal-nak is megvan a saját szintaxisa (szabályrendszere), melyektől a programunk megírása közben nem szabad eltérni. Ezek az általános szabályok a következők:
A programban ékezetes karakter az azonosítók nevében nem használhatók, és a nyelv nem tesz különbséget a kis- és nagybetűk között.
Egy Pascal nyelven megírt programban minden utasítást pontosvesszővel zárunk le, (kivételt jelent ez alól a blokkot kezdő ***begin*** utasítás illetve a főprogram végét jelző ***end.*** parancs).
A Pascal nyelv egy adott programja kulcsszavakból, azonosítókból, operátorokból, változókból, értékkonstansokból, címkékből és megjegyzésekből állhat.

- A nyelv kulcsszavait, más néven alapszavait (mint például a ***program, uses, procedure, function,*** stb.) a program saját célra használja, ezeket nem módosíthatjuk, mivel ezekből épülnek fel az egyes utasítások.
- Az azonosítók a programozó által felhasználható eszközök megnevezésére szolgálnak. Az azonosító mindig betűvel kezdődik és tetszés szerint, betűvel, számmal, vagy aláhúzás jellel folytatódhat.
- A címke a programon belüli utasításokra történő hivatkozásokra szolgál. Mindig a meghívandó utasítás előtt áll és kettőspont követi.

- A megjegyzést a programozó helyezi el a forráskódban. Ez később a kódot olvasó személyek számára magyarázatul szolgálhat, mely a megértést könnyítheti. Mivoltából adódóan a program futását nem befolyásolja. A megjegyzés szövegét kapcsos zárójelek ***{...}***, vagy kerek zárójelek és csillagok közé ***(*…*)*** helyezhetjük el.
- A változók nevét, amely mindig betűvel kezdődik, a programozó adja meg, ezzel a névvel használhatjuk fel. A változók deklarálásakor a típust kötelező megadni, amely befolyásolja a felvehető értéket (például számok, vagy betűk). A változó használatához természetesen értékadás szükséges.
- Az értékkonstansok olyan önmagukat definiáló programozási eszközök, melyeknek nincs neve. Önmagukat definiálják, tehát meghatározzák saját értéküket. Jellemzőik az adattípus és az érték. (Az adattípusok részletezésére később térünk ki.)

  A konstansoknak van egy altípusa, a nevesített konstans, melynek jellemzői a név, adattípus és az érték. Nevük mindig az értéket képviseli, amely a program futása közben nem változtatható meg. A Pascal nyelvben vannak standard konstansok (például a ***true***, ***false***), de akár a programozó is definiálhat ilyen nevesített konstansokat.
- Az operátorok különböző műveletek elvégzésére alkalmasak. Az egyszerű matematikai műveletek operátorain túl (+,-,/,*) számos lehetőség közül választhatunk, például logikai operátorok (AND, OR, NOT, XOR) stb.

A Pascal program három fő részből épül fel, ezek a programfej, deklarációs rész és a programtörzs.

Programfej

A programfej a program legelső sora, amely opcionálisan tartalmazza a program azonosítóját. Ez a rész a ***program*** kulcsszóból és a programozó által megadott azonosítóból, azaz programnévből áll.

```
program pelda;
```

Deklarációs rész

Ebben a részben kerülnek deklarálásra a programban használni kívánt változók. A változók deklarálása a ***változónév:adattípus;*** paranccsal adható meg. Ez a típus a program során nem változhat meg. Szintén itt kerülnek megadásra a programban szükséges konstansok, eljárások, függvények, illetve címkék. Továbbá a deklarációs részhez tartozik a unit-ok meghívása is.

```
{ Unit-ok meghívása: }
uses unit_1, unit_2, ..., unit_n;
{ Címkék deklarálása: }
label címke_1, címke_2, ..., címke_n;
{ Konstansok deklarálása: }
const konstans1 = érték1;
      konstans2 = érték2;
{ Változók deklarálása: }
var változó1: tipus1;
    változó2: tipus2;
{ Saját függvények megírása, melyek később meghívásra kerülnek: }
function függvény1[(paraméterlista)]: tipus1;
function függvény2[(paraméterlista)]: tipus2;
{ Saját eljárások megírása, melyek később meghívásra kerülnek: }
procedure eljárás1[(paraméterlista)];
procedure eljárás2[(paraméterlista)];
```

Programtörzs vagy végrehajtási rész

A programtörzs tartalmazza a főprogramot, amely utasításokat, illetve az alprogramok (függvények, eljárások) meghívását tartalmazza. Ez a

rész mindig a program legvégén található, a ***begin*** és az ***end.*** kulcsszavak között. Valójában ez a program „lelke”, mivel az ebben foglalt utasítások oldják meg azt a feladatot, amiért a programot megírtuk.[3]

```
begin
  utasítás_1;
  utasítás_2;
  …
  utasítás_n;
end.
```

1.1.2 A Java programozási nyelv

1.1.2.1. Története

A Java programozási nyelvet a Sun Microsystems, pontosabban James Gosling és munkatársai fejlesztették ki az 1990-es évektől kezdve. Az eredetileg oak (tölgyfa) nevet viselő objektumorientált nyelv (később Java-ra átkeresztelve) a C és C++ nyelvek szintaxisát örökölte. A Sun alkalmazottjai azonban a bonyolultnak vagy nem elég megbízhatónak talált elemeket mellőzték és más nyelvek pozitív szerkezeteit emelték át, így a C és C++ nyelvnél egyszerűbb objektummodellel rendelkező nyelvet hoztak létre. A Java a Sun Microsystems védjegye.

A fejlesztés első programozók által is „kézzel fogható” eredményeként 1995. novemberében megjelent a Java Developement Kit 1.0, mely az

[3] http://indy.poliod.hu/program/Pascal/Tankonyv/Tankonyv.htm
http://www.kobakbt.hu/jegyzet/PascalPrg/pascal3.htm
http://www.sulinet.hu/inform/szakkor/jegyzet/pascal1/
http://zeus.nyf.hu/~akos/pascal/pasframe.htm
http://www.prog.ide.sk/pas.php
http://programozo.cellszaksuli.hu/download/programming/pascal/tetelsor.doc

osztálykönyvtárakat és a Java virtuális gépet tartalmazta. 2008-ig összesen nyolc verzió jelent meg. A ma legelterjedtebb változata az 1.4-es, a legújabb pedig a 7-es verziószámú változat, melynek kódneve Dolphin.[4]

1.1.2.2. Általános jellemzői

A Java nyelvnek hat fő tulajdonsága van: egyszerűség, objektumorientáltság, platformfüggetlenség, interpretáltság és dinamikusság, többszálúság és elosztottság. A továbbiakban ezen tulajdonságok részletezése következik.

Egyszerűség

A nyelv egyszerűsége a C++ nyelv leegyszerűsítéséből és abból a praktikus tulajdonságból adódik, hogy a program számon tartja a használt tárterületet, és ha olyat talál, mely a továbbiakban már szükségtelen, azt törli, vagyis felszabadítja a területet (erről a programozónak nem kell gondoskodnia).

Objektumorientáltság

Egy programozási nyelvet akkor tekintünk objektumorientáltnak, ha a következő három tulajdonság mindegyike teljesül rá: *öröklődés, polimorfizmus és zártság*.

Öröklődés: Egy osztálynak lehet leszármazott osztálya, mely örökli a szülőosztály metódusait és egyedváltozóit, valamint saját metódusokkal és egyedváltozókkal is rendelkezhet.

---

[4] http://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv)
Rogers Cadenhead: Tanuljuk meg a java programozási nyelvet 24 óra alatt, 2006
http://www.jgypk.u-szeged.hu/~devosa/dll/stud/Java/docs/JAVA.pdf
http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/ficsorl/Targyak/OOP/Segedletek/java1.pdf

Polimorfizmus (többalakúság): Egy adott leszármazott osztályban az öröklött metódusokat újradefiniálhatják, azaz nevük, pontosabban szignatúrájuk megegyezik, ám a blokkban szereplő utasítások különböznek. A polimorfizmus kizárólag a metódusokra érvényes, a megöröklött egyedváltozókra nem.

Zártság: Az osztály egyedváltozóihoz csak metódusokon keresztül férhetünk hozzá, a mezőket „elrejtjük” a felhasználó elől.

Platformfüggetlenség

A Java programok különböző hardverkonfigurációkon hasonlóan futnak le, köszönhetően annak, hogy a Java Compiler (fordító) nem egyből gépi kódra alakítja át a forráskódot, hanem egy közbülső lépéssel, egy alkalommal lefordítja bájtkódra, melyet a Java Virtual Machine (JVM) a program minden egyes futtatásakor átfordít gépi kódra. Így oldható meg az, hogy egy program csak egyszer kerül megírásra, mégis bármilyen olyan platformon futtatható, melyre telepítve van a JVM.

Interpretáltság és dinamikusság

Az interpretált programok minden információt tartalmaznak, melyek a program írásához fontosak lehetnek. A program módosítása után elég csak a megváltoztatott részeket lefordítattunk a Java Compiler-rel, ezután a program már közvetlenül futtathatóvá válik. A dinamikusság a folyamatos, futás ideje alatt történő betöltésre vonatkozik. Ez azt jelenti, hogy amennyiben szükséges, a class-loader (osztálybetöltő) futás közben tölti be a már lefordított, bájtkódú állományokat.

Többszálúság

A programok ezen tulajdonsága azt jelenti, hogy a programok nagy része több párhuzamosan futtatható vezérlési szálra (thread) osztható,

ezzel elősegítve az utasításnak a számítógép központi egysége által történő gyorsabb végrehajtását.

Elosztottság

Az elosztottság a hálózati lehetőségeket foglalja magába. Lehetőség nyílik arra, hogy a class-loader hálózatról töltsön le bájtkódú állományokat, valamint a rendszerkönyvtárban találhatóak olyan osztályok, amelyek képesek az internetes protokollok (HTTP, FTP) kezelésére is. [5]

1.1.2.3. A Java program alkotóelemei

Ha össze akarjuk hasonlítani a Java és a Pascal programot, akkor lényeges különbségeket láthatunk. A Java szerkezetileg sokkal nagyobb szabadságot ad a programozók számára (például a függvények, metódusok szabad elhelyezése vagy a változók szabad deklarálása /ezzel azonban vigyázni kell, például cikluson belüli deklarálás esetén, mivel ilyenkor a cikluson kívül nem létezik a változó/), viszont a fordító szigorú szintaxisából adódóan nagy figyelmet kell fordítani a programozás során a nyelvi szabályok betartására (például kis- és nagybetűk használata).

Egy Java program utasítások halmazából épül fel. Utasításoknak azokat a parancsokat nevezzük, amelyek meghatározzák a számítógép számára végrehajtandó feladatot vagy feladatokat. Minden utasítás pontosvesszővel (***;***) zárul. A programnak egyes részeit csoportokba, úgynevezett *utasításblokkokba* rendezhetjük, ezeket az egységeket kapcsos zárójelek (***… {utasítás_1; utasítás_2;***

[5] Rogers Cadenhead: Tanuljuk meg a java programozási nyelvet 24 óra alatt, 2006 http://www.cab.u-szeged.hu/WWW/java/kiss/javaprog.html

utasítás_n; } …) határolják. Lehetőség van a blokkok egymásba ágyazására is. A program megírásánál kifejezetten nagy figyelmet kell fordítani a zárójelek használatára, mivel ezek a jelek kötelezően párosan fordulnak elő, minden nyitótaghoz kapcsolódik egy záró tag.

A Java utasítások csoportosítása:

- deklaráló (pl.: ***String szoveg;***)
- értékadó (pl.: ***szoveg="Szia!";***)
- post- vagy prefix értéknövelő és csökkentő utasítás változatok:
  - postfix növelő (pl.: ***i++;***)
  - postfix csökkentő (pl.: ***i--;***)
  - prefix növelő (pl.: ***++i;***)
  - prefix csökkentő (pl***.: --i;***)
- metódushívó utasítás (pl.: ***System.out.println ("Szia!");***)
- példányosítás (pl.: ***szoveg=new String ("Szia!");***)
- programvezérlő utasítások (pl.: szelekció, iteráció)
- üres utasítás: ***;***

Az utasítások egy csoportját kifejezéseknek nevezzük, mivel matematikai műveleteket tartalmaznak, és valamilyen értéket adnak vissza.
A különböző utasítástípusok részletezésére később térünk ki.

Egy Java nyelven megírt program minden esetben tartalmaz egy osztálydeklarációt és (legtöbbször) egy main metódust. Az osztálydeklaráció (pl.: ***public class Programnev { … }***) a program elején szerepel, és a program elnevezésére szolgál. A deklarációban szereplő ***class*** kulcsszó jelentése osztály. Ennek az megnevezésnek az az oka, hogy a Java programokat osztályoknak is hívják. A class

után szereplő osztálynév mindig nagybetűvel kezdődik, és szigorúan megegyezik (betűről-betűre) a fájl nevével.

A main metódus a program kitüntetett része, amely végrehajtható utasításokat tartalmaz, és végrehajtáskor ez kerül elsőként feldolgozásra. Pl.: ***public static void main (String [] args){…}***

A forrásprogram különböző metódusokat (eljárásokat, függvényeket) is tartalmazhat, melyek a main metóduson kívül kerülnek megírásra, és egymásból vagy a main metódusból kell meghívni őket. A két metódus típus között az a különbség, hogy az eljárás deklarálásakor a megnevezés elé void-ot írunk, amivel azt jelezzük, hogy nem rendelkezik visszatérési értékkel míg a függvény tartalmaz egy, vagy több return utasítást, amelyekkel visszatérési értéke(ke)t adhatunk meg. A metódusok deklarálásakor jelölni kell a visszatérési érték meglétét, és ha van, annak típusát kötelező feltűntetni. (lásd a függvényes példát)

- Eljárás (nincs visszatérési érték)

  Pl.: ***public void kiir() {…}***
- Függvény (van visszatérési érték)

  Pl.: ***public int szamol() {…return i;}***

Java program írása közben lehetősége van a programozónak megjegyzések beszúrására is a program forráskódjába. Ennek köszönhetően a programunkat „beszédesebbé”, könnyebben érthetővé tehetjük. Két módja van:

- *// …* - egy sornyi megjegyzés beszúrásakor két perjellel kezdjük a sort, sorvégi jelzésre nincs szükség.
- */*…*/* - több soros megjegyzés esetén magyarázatunkat per-csillag és csillag-per jelek közé tesszük.[6]

[6] Rogers Cadenhead: Tanuljuk meg a java programozási nyelvet 24 óra alatt, 2006 http://www.cab.u-szeged.hu/WWW/java/kiss/javaprog.html

1.2. Az e-learning

Az e-learning fogalmának meghatározása nehéz feladat, mivel szorosan összekapcsolódik az infokommunikáció technikáival, velük párhuzamosan változik és bővül. Az e-learning-et leegyszerűsítve tekinthetjük az Internet vagy Intranet által nyújtott oktatási lehetőségek felhasználásának, tágabb értelemben minden olyan oktatási forma beleszámít, amely állandó oktatói jelenlét nélkül, tudás átadására alkalmazható, és valamilyen elektronikus eszköz (pl.: CD-ROM, DVD-ROM, interaktív tv, Internet) segítségével valósul meg.[7]

http://www.cab.u-szeged.hu/WWW/java/kiss/article.html
http://hu.wikibooks.org/wiki/Java_Programoz%C3%A1s/A_nyelv_jellemz%C5%91i
http://www.jgypk.u-szeged.hu/~devosa/dll/stud/Java/docs/e-Book_Java_prog_1.3.pdf
http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/ficsorl/Targyak/OOP/Segedletek/java1.pdf

[7] http://edutech.elte.hu/multiped/szst_11/szst_11.pdf
http://www.coedu.hu/domain9/files/modules/module15/2699844175F3CB3.pdf
http://www.enc.hu/1enciklopedia/fogalmi/ped/elern.htm

1.2.1 Az e-learning története

Az e-learning és a távoktatás szorosan összetartoznak. A távoktatás alapjai hosszú múltra tekintenek vissza. Fejlődésének során, mely összefügg a tömegkommunikációs eszközök kialakulásával és terjedésével, három meghatározó szakaszt különítünk el.

Az első szakasz a ***levelező oktatás***, mely a postabélyeg 1840-es angliai megjelenésével kapcsolódik össze. A találmány a világ többi országaiban néhány év késéssel jelent meg. A feltalálás helyszínéből adódóan ezt az oktatási formát elsőként Nagy-Britanniában alkalmazták. A hagyományos oktatást is helyettesítő forma több európai országban is elterjedt az első világháború kitöréséig.

A változást az 1920-as évek hozta, amikor feltalálták a rádiózást. A széles körű elterjedése után oktatási célokra is elkezdték felhasználni. 1927-ben a BBC rádiója iskolai anyagok sugárzását kezdte, ez volt az első és legfontosabb próbálkozás. A ***rádión keresztül történő oktatás***, ugyanúgy, mint a levelező oktatás, először angol területeken honosodott meg, azonban a hagyományos oktatást ez csupán kiegészítette, helyét nem tudta átvenni. Ez a korszak a második világháborúig tartott.[8]

A harmadik szakaszt a ***számítógéppel támogatott oktatás*** (CBT - Computer Based Training) 1980-tól megkezdett térhódítása jelentette. Ennek az oktatásnak és magának az e-learning-nek egyik alapfeltétele a számítógépek jelenléte a háztartásokban, mely az 1990-es évek elejére általánossá vált. Az informatika által nyújtott széles körű lehetőségeknek köszönhetően lehetővé vált a vizuális és audiovizuális élmények együttes megtapasztalása (a multimédia) a tanulás folyamatában. Ez hatalmas lépést jelentett és jelent ma is a

[8] http://mek.oszk.hu/06800/06829/06829.pdf

levelező és rádiós oktatással szemben, mivel ezek csak egy-egy érzékszervre voltak hatással. A tananyag eljuttatása a hallgatósághoz kezdetben háttértárolókon (CD-ROM, DVD-ROM) történt. Az e-learning másik létfontosságú feltétele a számítógépes hálózatok megjelenése volt. Kezdetben a helyi hálózatok (LAN – Local Area Network), majd bővülésükkel a városi (MAN – Metropolitan Area Network) és nagy kiterjedésű (WAN – Wide Area Network) hálózatok kiszélesítették az elektronikus oktatás alapját. Igazán azonban az Internet megjelenése tette lehetővé azt, hogy bármely háztartásba „betehesse a lábát" e modern oktatási forma.

Igaz, hogy még napjainkban is érvényes az a megállapítás, hogy kevés projekt indul az e-learning használatának bevezetésére. A kezdeményezések többsége is csupán a felnőttképzési, felsőoktatási intézmények oktatási rendszerében jelenik meg, kiegészítő elemként, például a debreceni Kölcsey Ferenc Református Tanítóképző Főiskola rendszerében.

Több nemzetközi oktatók és hallgatók között végzett felmérés szerint a leginkább gátló tényező az oktatók részéről a felkészületlenség, a hallgatók részéről pedig a motiváció hiánya. Természetesen érthető okokra vezethető vissza az oktatók között előforduló alulmotiváltság is, mivel nincs megfelelően kompenzálva az oktatási módszerük új elemmel történő kibővítése, se szellemi elismerés, se anyagi támogatás terén. Előfordul olyan felsőoktatási intézmény, amelyben az e-learning bevezetésének ellenérve a konzervatizmus, saját hagyományaik és egyéni oktatási szokásaik megtartása. A hallgatók körében jelenlévő pozitív hozzáállás oka viszont éppen a tanulás e módjának kényelmessége, szabadsága.

Az e-learning alkalmazásának és az ehhez szükséges támogatások, valamint feltételek teljesülésének hatása a távoktatás

minőségének és hatékonyságának növekedésében mutatkozik meg. Ez a tendencia a jövőben előreláthatóan javulni fog. [9]

1.2.1.1. Az e-learning típusai

Az e-learning-et aszerint, hogy hogyan jut el az információ a tanulóhoz, három fő típusba soroljuk: egyirányú információáramlással rendelkező, a kétirányú, interaktív kommunikációt lehetővé tevő, valamint a személyes kapcsolatot biztosító e-learning. Amennyiben az e-learning-et nem csak a távoktatás kontextusában vizsgáljuk, hanem elektronikus oktatási segédanyagként tekintünk rá, akkor a három fő típuson kívül egy negyediket is megkülönböztetünk. Ez az elektronikus segédanyagokkal kiegészülő hagyományos oktatási forma. Ezen négy típus részletezése a következőkben történik:

Egyirányú információáramlással rendelkező típus

Ehhez a típushoz a tanulónak nincs szüksége internet-hozzáféréshez, mivel az információ átadása hordozható háttértárolón történik (CD-, vagy DVD-ROM, esetleg videokazetta, hangkazetta, bár utóbbiak manapság már elavultak). Így egy internet-kapcsolat nélküli személyi számítógép is alkalmas az adott tananyag megtekintéséhez. A nagy mértékű rugalmasság ellenére nagy probléma ennél a típusnál az oktató és tanuló közötti kétirányú kommunikáció hiánya. Ez azt eredményezheti, hogy a tanuló az adott tananyagot rosszul, vagy hibásan értelmezi, kérdéseit nem tudja

[9] http://www.oki.hu/oldal.php?tipus=cikk&kod=akademia-2002-Hidvegi-elearning
http://www.hrportal.hu/index.phtml?page=article&id=61323

feltenni, továbbá hiányzik a számonkérés lehetősége. Ez a típus teljes szabadságot ad a tanuló részére.

A kommunikációt lehetővé tevő e-learning

Ebben a típusban a személyes kapcsolat hiányzik, ám valamilyen módon (később részletezzük) a tanulónak lehetősége van az oktatóval történő kommunikációs csatorna létrehozására, így biztosítván az információ (pontosabban a kérdéses témakörök magyarázatának, illetve a felmerülő kérdések megválaszolásának) megfelelő áramlását mindkét fél felé. Éppen ezért ezt a típust akkor használják az oktatásban, amikor fontos a visszacsatolás, valamint az időbeli rugalmasság. A kommunikáció az Internet által nyújtott különböző szolgáltatások igénybevételével hozható létre. Ilyen szolgáltatás például az e-mail, a chat szoba, a „virtual classroom", és a virtuális egyetem.

Személyes kapcsolatot biztosító típus

Az e-learning ezen fajtája olyan személyes kapcsolatot biztosít a tanár és a diák között, amely nem egyezik meg a hagyományos iskolai formával, mivel ritkább a személyes kontaktus. Rendszerint csak néhány konzultáció jellegű megbeszélés zajlik a tananyaggal kapcsolatban. Ez elősegíti a félreértések elkerülését, mert a tanuló már nem csak az Internetet használja (mint az előző típusban). Ezek mellett meghagyja a tanulói szabadságot, a saját időbeosztást és önállóságot, valamint költségkímélő megoldást jelent az oktatási központtól távol lakó hallgatók számára is.

Elektronikus segédanyagokkal kiegészülő hagyományos oktatási forma

Ebben a típusban a hagyományos, iskolai oktatás számítógépes közegben zajlik, így biztosítván az Internet-elérést és az elektronikus segédanyagok használatát. [10]

1.2.1.2. Az e-learning módszertana

Az e-learning módszertanát az oktatott tananyagok egymáshoz fűződő kapcsolata alapján három különböző csoportba sorolhatjuk. Ezek a következők:

- **Lineáris szerkezetű tananyagok**
  A tananyagok meghatározott sorrendben, időrend alapján követik egymást.

- **„Pókháló" szerkezetű tananyagok**
  Ebben a szerkezetben egy központi témát találhatunk, amelyhez több, kisebb tananyagrész kapcsolódik. Ezek a részek mind kapcsolatban vannak egymással is.

- **Hierarchikus szerkezetű tananyagok**
  A tananyagok alá-fölé rendeltségi viszonyban állnak egymással. Legalább egy főtémával rendelkezik a tananyag, mely tovább bontható altémákra, melyek tartalmazzák a tananyagot. A tanulás ideje alatt a fő- és altémák meghatározott sorrendjében haladnak.[11]

[10] http://www.oki.hu/oldal.php?tipus=cikk&kod=akademia-2002-Hidvegi-elearning

[11] http://www.bgk.bmf.hu/mpt/az_e-learning_szerepe_a_felnottoktatasban_es-kepzesben.pdf
http://www.hrportal.hu/index.phtml?page=article&id=61323

1.2.2 Az e-learning az Európai Unióban

Az 1957. március 25-én létrejött Római Szerződés, amely az Európai Gazdasági Közösség alapdokumentuma, nem tért ki a közösségi kompetenciák terén az oktatásra. Ez a téma az Európai Uniós alapszerződések szintjén legelőször az 1992-es Maastricht-i szerződésben jelenik meg. Ettől kezdve indul el az oktatás nemzetközi szintű összehangolásának folyamata. Ebben az időszakban az EU-s országok oktatáspolitikájában nagy szerepet kapott az e-learning, de a hétköznapokban csak elenyésző próbálkozásokról beszélhetünk. Pár évvel később a politikai életből szinte teljesen eltűnt ez a téma, helyét a terrorizmus elleni harc és a gazdasági problémák megoldására irányuló törekvések vették át. Ezzel szemben a mindennapi életben egyre nagyobb szerepet kapott az e-learning. Ellenőrzésére nagy hangsúlyt fektettek, mind uniós, mind nemzeti oldalon. Az elemző kutatásokat nagyobb részben az Európai Unió finanszírozta, ezek kiértékelésekor azonban nem minden esetben a várt eredményt kapták. Az e-learning-et kezdetben kísérleti programokban vezették be, melyek végső bevezetésük után többségükben anyagilag nem voltak fenntarthatóak. Ebben az időben jelentek meg olyan programok, melyek segítik az addig létrehozott e-learning-es projektek fenntarthatóságát és megfelelő minőségének hosszú távú biztosítását.

1991-től kezdve születtek meg olyan alapdokumentumok az Európai Unióban, melyek preferálták az e-learning-et. Az e-learning szempontjából legfontosabb stratégiai programokat és kezdeményezéseket a következőkben soroljuk fel:

- ***2000. május – eLearning initiative - Designing Tomorrow's Education, azaz E-learning – A jövő oktatásának tervezése***

Ez az első olyan stratégiai program, amely az e-learning-et EU-s magaslatokba emeli. A közös célok magukba foglalják az elektronikus oktatás alapfeltételeinek biztosítását, a tanárok továbbképzését, a közvetítő média szolgáltatásainak fejlesztését, valamint az oktatási központok közötti hatékony kommunikáció megteremtését.

- ***2001. március – eLearning Action Plan, azaz az Európai Bizottság E-learning Akcióterve***

  Ez a terv a 2001 és 2004 közötti időszakra vonatkozott és kiemelt hangsúlyt kapott benne az e-learning, amely a tervek szerint abban segíti az Európai Uniót, hogy a világ legdinamikusabban fejlődő tudásalapú gazdaságává váljon. Ennek fő feltételei a *lifelong learning*, azaz az élethosszig tartó tanulás előtérbe helyezése, az e-learning hozzáférhetővé tétele mindenki számára, valamint a magas színvonalú tananyag.

- ***2002. február 18. – Brüsszel - eLearning: Designing Tomorrow's Education. An Interim Report***

  A 2002. február előtti programok kiértékelését tartalmazza a fent említett munkaanyag. A jelentés összefoglalja a konkrét lépéseket és eredményeket, valamint egy olyan bibliográfiát tartalmaz, amely az addig megjelent e-learning-gel foglalkozó irodalmat és webhelyeket sorolja fel.

- ***2002. december – Multiannual programme (2004-2006) for the effective integration of Information and Communication Technologies in education and traming systems in Europe, azaz e-Learning Program 2004-2006. Az információs és kommunikációs technológiáknak az oktatásba és képzésbe történő integrálása***

  A program címe magába foglalja a fő irányelvet, ezért ennek részletezésére nem térünk ki.

2003. február – ***E-learning információs portál*** létrehozása az Európai Bizottság megbízásából. Tizenegy nyelven teszi közzé az e-learning-gel kapcsolatos legfontosabb információkat, ilyenek például a szakkifejezések, dokumentációk, linkgyűjtemények listája.[12]

1.2.3 Az e-learning Magyarországon

Közép-Kelet Európában az az állapot uralkodik, miszerint már az EU-s csatlakozásokat megelőző idők óta súlyos lemaradásokkal küzdenek a tagországok. Ez a lemaradás nem csak gazdasági téren értendő, hanem az oktatási területeken is. Magyarországnak is halmozottan sok kritériumnak kellett megfelelnie a csatlakozáskor, ezek között szerepelt a göteborgi konferencián, 2001. június 16-án elfogadott és aláírt akciótervben foglalt oktatásügyi, pontosabban e-learning-gel kapcsolatos, az ország területén teljesítendő feladatok sora.

- A ***Humánerőforrás-fejlesztési Operatív Program (HEFOP)*** részeként 2002-ben megjelent ***Magyarország nemzeti fejlesztési terve (2004-2006)*** foglalja magába az ország oktatásügyével kapcsolatos feladatokat, így az e-learning-et is.
- ***2002. októberében a Sulinetwork*** konferencián fogalmazódott meg *az* ***országos közoktatási elektronikus tananyag adatbázis*** létrehozásának terve.

Magyarországon és az Európai Unióban egyaránt elmondható, hogy az elektronikus távoktatáshoz való hozzáállás javuló tendenciát mutat. Az

[12] http://ip.gallup.hu/elearning/eu.htm
http://thaleia.kodolanyi.hu/~dancso/dancso/2004.11.08/2004.11.08_DancsoT.pdf

e-learning előreláthatóan nem fogja átvenni a hagyományos oktatás szerepét, hanem azt kiegésztve *blended-learning*-ként, vagyis vegyes rendszerű tanulásként fog működni az oktatás világában. [13]

1.2.4 Az e-learning szabványok

Napjainkban ezen oktatási forma egyre elterjedtebbé válik. Sok oktatási központ, nagy- és kisvállalkozás is foglalkozik e-learning-es tananyagok kiadásával. Ahhoz, hogy a tananyagok szerkezetileg, tartalmilag és minőségileg megfeleljenek az Európai Unió dokumentumaiban foglaltaknak, általános szabványok létrehozására van szükség.

- ***2001. március - Draft Standard for Information Technology - Learning Technology – Glossary, azaz Az információs társadalom szabványai - Tanulási technológia – Glosszárium***

  Az eddig megjelent dokumentumokban megjelent összes e-learning-gel kapcsolatos szakkifejezés magyarázatát adja meg.

- ***2001. november - Standardization and Collaboration - Shaping the Future of eEurope, azaz Szabványosítás és együttműködés - az eEurope jövőjének megformálása.***

  Ez a dokumentum három EU-s szabványosítási szervezet *(European Committee for Standardization (CEN) European Committee for Electrotechnical Standardization (CENELEC) European Telecommunications Standards Institute (ETSI))* munkáját mutatja be.

---

[13] www.elek.co.hu/dpi/moodledata/46/Az_e-learning.ppt
http://ip.gallup.hu/elearning/mo.htm

- ***2002. április – eLearning Standards Report, azaz Jelentés az e-learning szabványokról.***

  A fenti dokumentum értékelést ad a szabványosítás addigi előrehaladtáról, bemutatja az azt végző intézményeket, valamint előrevetíti a folyamatban levő szabványosítással kapcsolatos munkálatokat.
- Az összes e-learning-gel kapcsolatos szabvány fellelhető az Interneten, az **Online Catalogue of European Standards**-ban, vagyis az európai uniós szabványok katalógusában.
- Magyarországon az e-learning egységes szabványosítását **a Magyar Tartalomipari Szövetségre (MATISZ) és az Informatikai Vállalkozások Szövetségére (IVSZ)** bízták.[14]

[14] http://ip.gallup.hu/elearning/szabv.htm
http://www.oki.hu/oldal.php?tipus=cikk&kod=akademia-2002-Hidvegi-elearning

1.2.5 Előnyök és hátrányok

Végül vegyük sorra az e-learning pozitív és negatív tulajdonságait az oktatásban résztvevők szerint.

Az oktatási intézmény számára

Előnyök	***Hátrányok***
• Kevesebb kiadás a humán erőforrás és az utazási költségek tekintetében • Megbízható információátadás	• Jelentős kiadás az e-learning tananyagok bevezetésekor • Az informatikai alapok megteremtése nehézséget jelenthet • Az e-learning-et nem lehet bármilyen témájú tananyag közvetítésére alkalmazni

Az oktatók számára

Előnyök	***Hátrányok***
• Rugalmasság: a tanítás nem helyhez kötött • A tananyag frissítése egyszerűbb, mint a hagyományos papíralapú tankönyveké, jegyzeteké • A tanulók munkájának és előrehaladásának nyomon követése egyszerű	• A tananyag elkészítésének időigénye nagy • Csak kellő informatikai képzettséggel rendelkező oktatók képesek a rendszerek megfelelő használatára • Az oktatási intézmény jobban tudja ellenőrizni az oktató munkáját (bár ez bizonyos mértékben előnyt is jelent, pl. minőség)

A tanulók számára

Előnyök	***Hátrányok***
• Rugalmasságot jelent a tananyag időbeli és térbeli rendelkezésre állásában (bárhol, bármikor használható) • A tanuló felveheti saját tanulási ritmusát • A tananyagot a tanuló bármeddig (a képzés ideje alatt biztosan) tárolhatja • Általában lehetőség van visszacsatolásra az oktatók és tanulótársak részéről	• Hordozható számítógép hiányában a tankönyv sokkal nagyobb rugalmasságot jelent • Nem minden esetben van lehetőség az azonnali visszacsatolásra az oktatók részéről • Hiányzik a személyes kapcsolat az oktatás menetéből, így az személytelenné válik • Nem mindenki számára engedhető meg egy számítógép megvétele, illetve előfordulhat a technológiával szembeni idegenkedés is

Az e-learning történelmi áttekintése és elemzése után elmondhatjuk, hogy korunk oktatásának egyik legmodernebb kiegészítő irányvonala, mely hátrányai ellenére, a rugalmasság mellett mindenki számára lehetőséget nyújt a folyamatos fejlődésre és az élethosszig tartó tanulásra.[15]

[15] http://mek.oszk.hu/06800/06829/06829.pdf
http://www.elek.co.hu/dpi/moodledata/46/Az_e-learning.ppt
http://en.wikipedia.org/wiki/E-learning
http://tmt.omikk.bme.hu/show_news.html?id=4255&issue_id=467
http://www.coedu.hu/domain9/files/modules/module15/2699844175F3CB3.pdf
http://www.eduweb.hu/pdf/e_learning_tan.pdf
http://www.edu-online.eu/hu/letoltes.php?fid=tartalomsor/271
http://edutech.elte.hu/multiped/szst_11/szst_11.pdf
http://www.hrportal.hu/index.phtml?page=article&id=61323
http://www.bgk.bmf.hu/mpt/az_e-learning_szerepe_a_felnottoktatasban_es-kepzesben.pdf
http://www.efeb.hu/felnottkepzes-e-learning

2. A tananyag feldolgozása

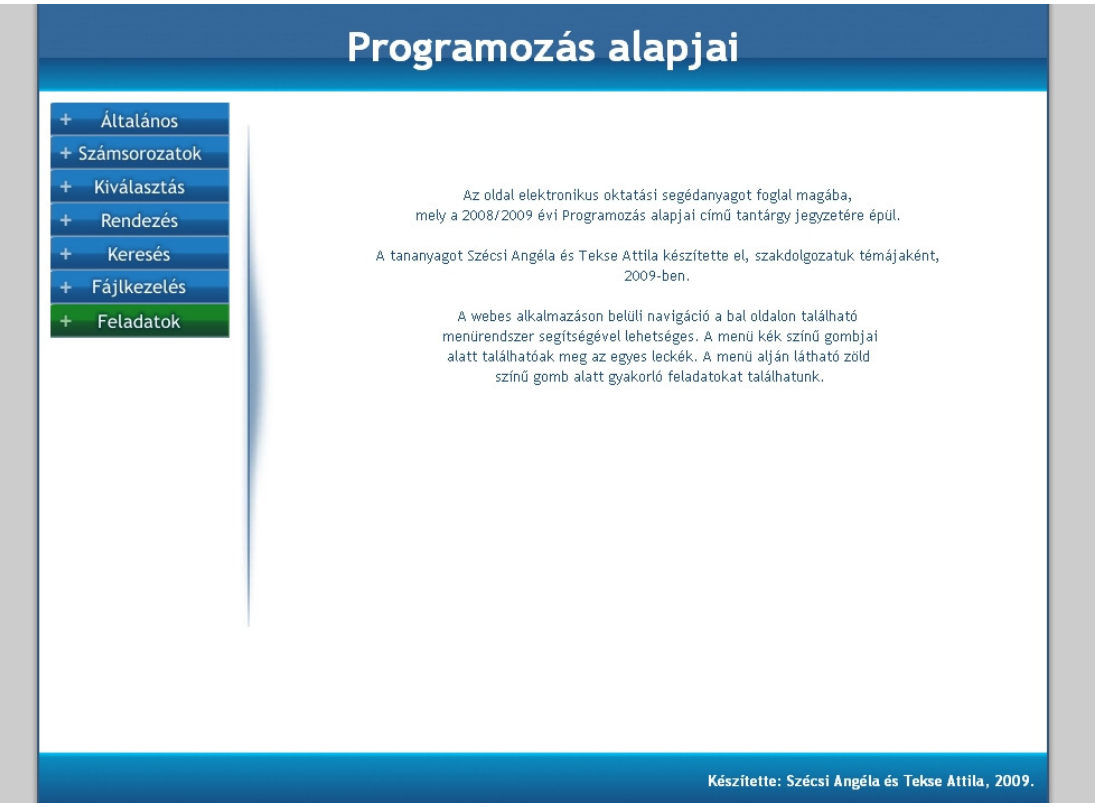
Az elektronikus tananyag megjelenítéséhez egy weblaprendszert hoztunk létre, mely létrehozásánál nagy hangsúlyt fektettünk a felhasználhatóságra és az egyszerű kezelhetőségre, ezért a weblapot kettő, jól elkülönülő részre osztottuk. A baloldalon található menürendszer segítségével könnyű a tájékozódás a webes alkalmazáson belül, a felület jobb oldalára a tartalom töltődik be.

A menüpontok a tananyag fejezetcímeit takarják. A menüsort színek alapján két részre osztottuk. Kékkel jelöltük a tananyag részeit, zölddel pedig a „Feladatok” részt, melyben a felhasználónak lehetősége van tudásának ellenőrzésére, illetve az egyes programok kipróbálására.

Az „Általános” részben a Java és Pascal programozási nyelven megírt programokra vonatkozó összefoglalást adunk, mely magában foglalja például a változódeklarációt, a vezérlési szerkezeteket és általános szabályokat. Ebben a részben tárgyaljuk továbbá a folyamatábrák részeit, azok ábrázolási módját, fogalmát is.
Az általános részt követő menüpontokban már a konkrét programok magyarázatára, azok folyamatábrában történő ábrázolására térünk ki.

A következőkben az egyes részek tartalmát, illetve megvalósításuk módját, weboldalon történő megjelenítésüket mutatjuk be.

(A tananyag webes formájának forráskódja a CD-mellékleten található meg.)



A weblaprendszer nyitóoldala

2.1. Általános áttekintés

2.1.1 A Java programozási nyelv alapjai

Változók lehetséges típusai

Típus	**Érték**
boolean (logikai, igaz vagy hamis)	`true, false`
int (egész szám)	`10, 24`
double (valós szám)	`10.5`
char (karakter)	`'c'`
String (szöveges)	`"szoveg"`
stb...	

Változók, tömbök deklarálása

A **változók deklarálása** a változó típusának és nevének megadásából áll. Deklaráció esetén lehetséges értékadás is. A a deklarációkkal azonban vigyáznunk kell, ha egy metódusban tesszük azt meg, akkor az a változó csak az adott metóduson belül érhető el.

Példa deklarációra:

`int i;` - egyész típusú, i nevű változó deklarálása

`boolean f=true;` - boolean típusú, f nevű, true értékkel rendelkező változó deklarálása

Tömbök deklarálása

A tömbök olyan homogén adatszerkezetek, amelyek meghatározott számú elemet tartalmazhatnak. A homogén tulajdonság azt jelenti, hogy csak azonos típusú elemeket adhatunk meg tömbelemeknek.

Példa deklarációra:

`int[] tomb=new int[10]` - így létrehoztunk egy 10 elemű, tomb nevű, egész típusú tömböt

A tömb elemeinek így adhatunk értéket:

`tomb[0]=20;` - Java-ban a tömb elemeinek indexe nullával kezdődik, így egy 10 elemű tömb esetében az utolsó tömbindex a 9.

Változók értékének növelése és csökkentése

Változók értékének változtatását többféle képpen is megadhatjuk, ha eggyel kívánjuk növelni vagy csökkenteni, akkor azt így tehetjük meg:

`valtozo=valtozo+1;` vagy `valtozo=valtozo-1;`

Ezzel egyenértékű a `valtozo++;` utasítás, csökkentéskor a `valtozo--;` utasítást használjuk.

Az értékmódosítás egy módja a **`++valtozo;`** vagy **`--valtozo;`** utasítás is.
Az, hogy melyik utasítást hazsnáljuk attól függ, hogy utólag vagy előbb szeretnénk növelni a változó értékét. Ha az utóbbiról van szó akkor a **`++valtozo;`** vagy **`--valtozo;`** utasításokat alkalmazzuk, ha az előbbiről, akkor pedig a **`valtozo++;`** vagy **`valtozo--;`** utasításokat.

Ha nem eggyel, hanem meghatározott értékkel szeretnénk módosítani a változó értékét, akkor azt a következő módokon tehetjük meg:

`valtozo=valtozo+5;` vagy **`valtozo=valtozo-5;`**

Vagy az előbb ismertetett megoldás elvével:

`valtozo+=5;` vagy **`valtozo-=5;`**

Feltételes utasítások megadásakor használható műveleti jelek és jelentésük

== Egyenlőség vizsgálata. (Az egyszerű egyenlőségjel értékadást jelent.)

!= Az egyenlőség tagadása, azaz nem egyenlő

< A bal oldalon lévő érték kisebb, mint a jobboldali

> A bal oldalon lévő érték nagyobb, mint a jobboldali

<= A bal oldalon lévő érték kisebb vagy egyenlő, mint a jobboldali, tehát már megengedett az egyenlőség is

>= A bal oldalon lévő érték nagyobb vagy egyenlő, mint a jobboldali, már megengedett az egyenlőség is

Feltételes utasítások logikai operátorokkal egészíthetőek ki, vagy kapcsolhatóak össze.

Logikai operátorok:

&& Az **ÉS** operátor jele.

Pl.: `((i>4) && (i<=20))` - Ez a feltétel csak akkor teljesül, ha az i nagyobb, mint 4 ÉS kisebb vagy egyenlő 20-nál.

|| A **VAGY** operátor jele.

Pl.: `((x<10) || (x==1))` - Ez a feltétel VAGY akkor teljesül, ha az x kisebb, mint 10, VAGY ha az x egyenlő eggyel.

! A **NEM** operátor jele.

Pl.: `!(b<20)` - Ez a feltétel akkor teljesül, ha a b NEM kisebb, mint 20.

Vezérlési szerkezetek

- Ciklusok
  - for iklus
  - while ciklus
  - do - while ciklus
  - Kilépés a ciklusból (break)
- Elágazások
  - Egyirányú elágazások
  - Kétirányú elágazások
  - Többirányú elágazások

Ciklusok

A for ciklus

Rögzített lépésszámú ciklus. Akkor hazsnáljuk, ha tudjuk, hogy a ciklus hányszor fog lefutni.

A for ciklus felépítése:

```
      for (előkészítő_rész; ltételes_rész;
növekményes_rész;){
         ciklusmag
         }
```

Az előkészítő részben adjuk meg a ciklusváltozó kezdőértékét. A feltételes részben vizsgáljuk azt, hogy a ciklusváltozó csökkentett vagy növelt értéke elérte-e már a kívánt értéket. A növekményes részben változtatjuk (növeljük vagy csökkentjük) a változó értékét.

A while ciklus

A while ciklus előltesztelő ciklus, tehát ha a ciklusfeltétel nem teljesül, a ciklusmag nem fut le. Így előforulhat az is, hogy a ciklus egyszer sem fut le.

A while ciklus felépítése:

```
      while (ciklusfeltétel){
         ciklusmag
         }
```

A ciklus addig fut, amíg a ciklusfeltétel teljesül. (A lehetséges ciklusfeltételeket, azaz logikai kifejezéseket fentebb részleteztük.)

A do - while ciklus

A do - while ciklus hátultesztelő ciklus, tehát attól függetlenül, hogy a ciklusfeltétel teljesül-e vagy sem, a ciklusmag mindenképpen lefut egyszer.

A do - while ciklus felépítése:

```
      do{
         ciklusmag
         }while(ciklusfeltétel);
```

A ciklus addig fut, amíg a ciklusfeltétel teljesül.

Kilépés a ciklusból (break)

A ciklusból történő kilépés a **`break;`** utasítással történik. Ennek köszönhetően a ciklusból "kiugrunk" és az utána következő utasítással folytatódik a program futása.

Elágazások

Egyirányú elágazások (if)

Az egyirányú elágazások felépítése:

```
if(logikai_kifejezés){
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
   }
```

Az if feltételes utasításban szereplő utasítások csak akkor futnak le, ha a logikai kifejezés teljesül. A kapcsos zárójelek közötti részt utasításblokknak nevezzük, és a benne szereplő utasítások egy utasításnak felelnek meg. A logikai kifejezés megadása után, ha csak egy utasítás áll, a kapcsos zárójelek elhagyhatóak.

Kétirányú elágazások (if - else)

A kétirányú elágazások felépítése:

```
if(logikai_kifejezés){
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
   }
else{
   utasítás_1;
   utasítás_2;
   ...
```

```
    utasítás_n;
    }
```

Ha a logikai kifejezések után csak egy utasítás szerepel, a kapcsoszárójelek elhagyhatóak. A feltételes utasítások egymásba is ágyazhatóak.

Az egymásba ágyazott kétirányú elágazások felépítése:

```
if(logikai_kifejezés){
    utasítás_1;
    utasítás_2;
    ...
    utasítás_n;
    }
else if(logikai_kifejezés){
    utasítás_1;
    utasítás_2;
    ...
    utasítás_n;
    }
    else{
        utasítás_1;
        utasítás_2;
        ...
        utasítás_n;
        }
```

A feltételes utasítások bármilyen mélységben egymásba ágyazthatóak.

Többirányú elágazások (switch - case)

A többirányú elágazásokat akkor hazsnáljuk, amikor egy változó többféle értéket vehet fel, és a különböző esetekben más és más utasítás hajtódjon végre.

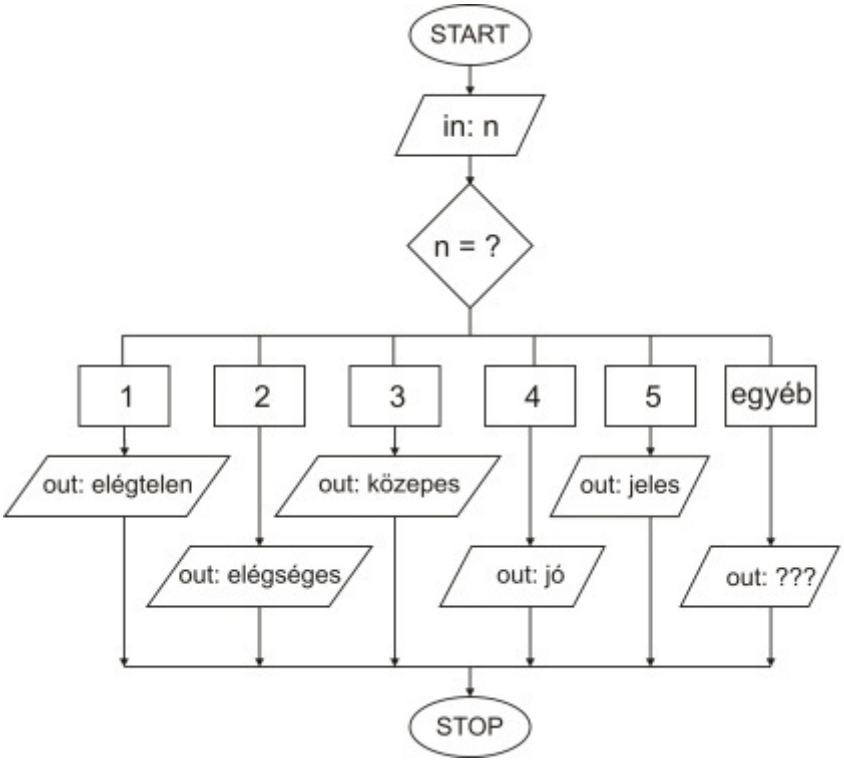
A többirányú elágazások felépítése:

```
switch(változó){
    case érték_1: utasítás_1; break;
    case érték_2: utasítás_2; break;
    ...
    default: utasítás_n;
```

```
}
```

A case-ek után adjuk meg a változó által felvehető lehetséges értéket majd kettősponttal elválasztva a végrehajtandó utasítást. Ezután egy **break;** utasítás következik, melynek köszönhetően átugunk a switch - case között lévő további utasításokon. A break utasítás elhagyható, de a programunk futását gyorsítja, hiszen felesleges a többi lehetőséget vizsgálni, ha már az egyiknél egyezést találtunk. A többirányú elágazás esetén van egy olyan speciális lehetőségünk, amely minden egyéb nem megadott lehetőség esetén következik be, ezt a **default: utasítás;** paranccsal tehetjük meg.

Példát csak erre az elágazástípusra mutatunk be, mivel ez talán a legbonyolultabb, illetve a további programokban sem fog szerepelni ennek alkalmazása.
Erre példaként egy olyan programot írunk, amely billentyűzetről bekér egy érdemjegyet, és azt kiírja szövegesen.
A hozzá tartozó folyamatábra a következő:




```
import java.io.*;
public class Jegyek{
   public static int in() throws Exception{
/* Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza. Később a programok során részletesen is magyarázásra kerül a függvény. (A függvényekről később esik szó.) */
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      String s=x.readLine();
      int i=Integer.parseInt(s);
      return i;
      }
   public static void main(String[] args) throws
Exception{
      int n;
      System.out.print("Érdemjegy: ");
      n=in();
/* Billentyűzetről bekérjük az érdemjegyet szám formájában, eltároljok az n változóban, majd a switch - case többirányú elágazás segítségével megvizsgálva azt kiírjuk annak szöveges megfelelőjét. */
      switch(n){
         case 1: System.out.println("Elégtelen"); break;
         case 2: System.out.println("Elégséges"); break;
         case 3: System.out.println("Közepes"); break;
         case 4: System.out.println("Jó"); break;
         case 5: System.out.println("Jeles"); break;
```

```
            default: System.out.println("???");
            }
        }
    }
```

Metódusok

- Eljárások
- Függvények

Az metódusoknak két típusa van: statikus (osztályszintű), vagy nem statikus metódusok. A "Programozás alapjai" c. kurzus során mi csak a statikus metódusokat alkalmazzuk.

Eljárások

A statikus eljárások felépítése:

```
      public static void
eljárás_neve(formális_paraméterlista){
            eljárás_törzsrésze
            }
```

A public szó az eljárás láthatóságát, hozzáférhetőségét biztosítja más programok számára.

A static jelenti azt, hogy az eljárás statikus azaz osztályszintű. Ez biztosítja számunkra, hogy minden metódusban (függetlenül attól hogy statikus-e vagy sem) példányosítás nélkül legyen elérhető. A példányosításról ebben afélévebn nem esik szó, mivel minden egyes metódust statikusnak definiálunk.

A void jelzi, hogy nem lesz a metódusnak visszatérési értéke, tehát eljárásról van szó.

Az eljárás neve után mindig szerepel a dupla kerek zárójel (), mely között a formális paraméterlista helyezkedik el. A formális paraméterlista a bemenő paraméterek nevét és típusát tartalmazza. Egy eljárásnak nem feltétlenül van bemenő paramétere, azok száma

lehet 0, 1 vagy több is. Ha többet is megadunk, azokat vesszővel választjuk el egymástól.

Függvények

A függvényt a visszatérési érték különbözteti meg az eljárástól.

A statikus függvények felépítése:

```
     public static visszatérési_érték_típusa
függvény_neve(formális_paramlista){
          függvény_törzsrésze
          return visszatérési_érték;
          }
```

A visszatérési érték típusát a függvény nevének megadása előtt kell megadnunk, ez lehet egésztípus, logikai, azaz boolean típus, String típus, satöbbi. A formális paraméterlista megadására ugyanaz vonatkozik, mint az eljárások esetében.

A visszatérési értéket a **`return visszatérési_érték;`** utasítás adja vissza. Return-t mindig tartalmaznia kell egy függvénynek, és nem szabad olyan feltételes utasításba helyeznünk, ami lehetséges, hogy nem teljesül, és így a **`return`** utasítás nem kerül végrehajtásra.

A metódusok a Java programok main metódusában kerülnek meghívásra. A bemenő paraméterek ekkor kapnak értéket. Mikor megadjuk egy metódus meghívásakor a paramétereket, azt aktuális paraméterlistának nevezzük.

2.1.2 A Pascal programozási nyelv alapjai

Változók lehetséges típusai

Típus	**Érték**
boolean (logikai, igaz vagy hamis)	`true, false`
integer (egész szám)	`-32768 és 32767 közötti egész szám`
double (valós szám)	`10.5`
char (karakter)	`'c'`
string (szöveges)	`'szoveg' (maximum 255 karakter)`
stb...	

Változók, tömbök deklarálása

A **változók deklarálása** a változó nevének majd kettőspont után a típus megadásából áll. A deklarációkkal azonban vigyáznunk kell, ha egy metódusban tesszük azt meg, akkor az a változó csak az adott metóduson belül érhető el.

Példa deklarációra:

`var i:integer;` - egyész típusú, i nevű változó deklarálása

`var f,t:boolean;` - boolean típusú, f és t nevű változók deklarálása (vesszővel elválasztva megadhatjuk egy sorban is az azonos tíupsú elemeket a deklírációban)

Tömbök deklarálása

A tömbök olyan homogén adatszerkezetek, amelyek meghatározott számú elemet tartalmazhatnak. A homogén tulajdonság azt jelenti, hogy csak azonos típusú elemeket adhatunk meg tömbelemeknek.

Példa deklarációra:

`tomb:array [1..10] of integer;` - így létrehoztunk egy 10 elemű, tomb nevű, egész típusú tömböt

A tömb elemeinek így adhatunk értéket:

`tomb[1]=20;` - Java-val ellentétben itt a tömb elemeinek indexe eggyel kezdődik.

Változók értékének növelése és csökkentése

Változók értékének változtatását többféle képpen is megadhatjuk, ha eggyel kívánjuk növelni vagy csökkenteni, akkor azt így tehetjük meg:

`valtozo=valtozo+1;` vagy **`valtozo=valtozo-1;`**

Ezzel egyenértékű a **`inc(valtozo);`** utasítás, csökkentéskor a **`dec(valtozo);`** utasítást használjuk.

Feltételes utasítások megadásakor használható műveleti jelek és jelentésük

= Egyenlőség vizsgálata.

< A bal oldalon lévő érték kisebb, mint a jobboldali

> A bal oldalon lévő érték nagyobb, mint a jobboldali

<= A bal oldalon lévő érték kisebb vagy egyenlő, mint a jobboldali, tehát már megengedett az egyenlőség is

>= A bal oldalon lévő érték nagyobb vagy egyenlő, mint a jobboldali, már megengedett az egyenlőség is

Feltételes utasítások logikai operátorokkal egészíthetőek ki, vagy kapcsolhatóak össze.

Logikai operátorok:

and Az **ÉS** operátor jele.

Pl.: `((i>4) and (i<=20))` - Ez a feltétel csak akkor teljesül, ha az i nagyobb, mint 4 ÉS kisebb vagy egyenlő 20-nál.

or A **VAGY** operátor jele.

Pl.: `((x<10) or (x==1))` - Ez a feltétel VAGY akkor teljesül, ha az x kisebb, mint 10, VAGY ha az x egyenlő eggyel.

not A **NEM** operátor jele.

Pl.: `not(b<20)` - Ez a feltétel akkor teljesül, ha a b NEM kisebb, mint 20.

Vezérlési szerkezetek

- Ciklusok
  - for iklus
  - while ciklus
  - repeat - until ciklus
  - Kilépés a ciklusból (break)
- Elágazások
  - Egyirányú elágazások
  - Kétirányú elágazások
  - Többirányú elágazások

Ciklusok

A for ciklus

Rögzített lépésszámú ciklus. Akkor hazsnáljuk, ha tudjuk, hogy a ciklus hányszor fog lefutni.

A for ciklus felépítése:

```
      for ciklusváltozó:=kezdőérték to
változó_maximális_értéke do begin
         utasítás_1;
         utasítás_2;          ...
         utasítás_n;
         end;
```

A ciklus **do** utáni rész hajtódig végre mindaddig, amíg a változó el nem éri a megadott maximális értéket. A változó csak egész típusú lehet. A **to** jelenti az eggyel történő léptetést, növelését a változó értékének minden egyes lefutáskor eggyel. Ha csökkenteni szeretnénk akkor a to-t **downto**-ra kell cserélnünk, így:

```
      for ciklusváltozó:=kezdőérték downto
változó_minimális_értéke do begin
         utasítás_1;
         utasítás_2;
         ...
         utasítás_n;
         end;
```

Ha csak egy utasításból áll a ciklusmag a **begin** és **end;** elhagyható, ekkor az utasítást egyszerűen pontosvesszővel zárjuk.

A while ciklus

A while ciklus előltesztelő ciklus, tehát ha a ciklusfeltétel nem teljesül, a ciklusmag nem fut le. Így előforulhat az is, hogy a ciklus egyszer sem fut le.

A while ciklus felépítése:

```
      while (ciklusfeltétel) do begin
         ciklusmag
         end;
```

A ciklus addig fut, amíg a ciklusfeltétel teljesül. (A lehetséges ciklusfeltételeket, azaz logikai kifejezéseket fentebb részleteztük.) Ha a ciklusmag csak egy utasításból áll, akkor a **begin** és az **end;** elhagyható.

A repeat - until ciklus

A repeat - until ciklus hátultesztelő ciklus, tehát attól függetlenül, hogy a ciklusfeltétel teljesül-e vagy sem, a ciklusmag mindenképpen lefut egyszer.

A repeat- until ciklus felépítése:

```
repeat
   ciklusmag
   until(kilépési_feltétel);
```

A ciklus addig fur, amíg a ciklusfeltétel már NEM teljesül, ez a kilépési feltétel lényege.

Kilépés a ciklusból (break)

A ciklusból történő kilépés a **break;** utasítással történik. Ennek köszönhetően a ciklusból "kiugrunk" és az utána következő utasítással folytatódik a program futása.

Elágazások

Egyirányú elágazások (if)

Az egyirányú elágazások felépítése:

```
if(logikai_kifejezés) then begin
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
```

```
end;
```

Az if feltételes utasításban szereplő utasítások csak akkor futnak le, ha a logikai kifejezés teljesül. A logikai kifejezés megadása után, ha csak egy utasítás áll, a **begin** és az **end;** elhagyható.

Kétirányú elágazások (if - else)

A kétirányú elágazások felépítése:

```
if(logikai_kifejezés) then begin
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
   end
else begin
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
   end;
```

Ha a logikai kifejezések után csak egy utasítás szerepel, a **begin** és az **end;** elhagyható. Az if-es kifejezést lezáró end után nem szabad pontosvesszőt tennünk, illetve ha end nincs, azaz csak egy utasítás van, azt sem zárhatjuk le pontosvesszővel, ha van utána else. A feltételes utasítások egymásba is ágyazhatóak.

Az egymásba ágyazott kétirányú elágazások felépítése:

```
if(logikai_kifejezés) then begin
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
   end
else if(logikai_kifejezés) then begin
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
   end
```

```
else begin
   utasítás_1;
   utasítás_2;
   ...
   utasítás_n;
   end;
```

A feltételes utasítások bármilyen mélységben egymásba ágyazhatóak.

Többirányú elágazások (case)

A többirányú elágazásokat akkor hazsnáljuk, amikor egy változó többféle értéket vehet fel, és a különböző esetekben más és más utasítás hajtódjon végre.

A többirányú elágazások felépítése:

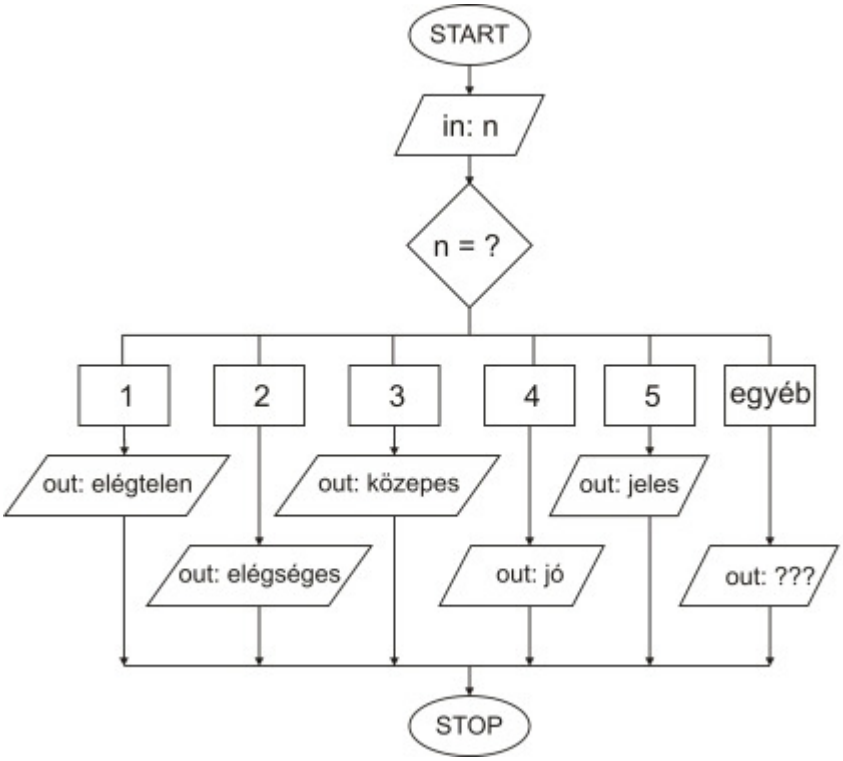
```
case(változó) of
   érték_1: utasítás_1; end;
   érték_2: utasítás_2; end;
   ...
   else: utasítás_n; end;
```

Az of után adjuk meg a változó által felvehető lehetséges értéket majd kettősponttal elválasztva a végrehajtandó utasítást. Ezután egy **`end;`** utasítás következik, melynek köszönhetően átugunk a case-ben lévő további utasításokon. A többirányú elágazás esetén van egy olyan speciális lehetőségünk, amely minden egyéb nem megadott lehetőség esetén következik be, ezt az **`else: utasítás;`** paranccsal tehetjük meg.

Példát csak erre az elágazástípusra mutatunk be, mivel ez talán a legbonyolultabb, illetve a további programokban sem fog szerepelni ennek alkalmazása.

Erre példaként egy olyan programot írunk, amely billentyűzetről bekér egy érdemjegyet, és azt kiírja szövegesen.

A hozzá tartozó folyamatábra a következő:




```
program jegyek;
uses crt;
var jegy:integer;
BEGIN
   write ('Érdemjegy: ');
   readln(jegy);
```
{ Billentyűzetről bekérük az érdemjegyet szám fomájában, eltároljuk a jegy változóban. Ezután a többirányú elágazás segítségével megvizsgáljuk, hogy milyen értéke van, és az ahhoz tartozó utasítást végrehajtja a program. }
```
   case(jegy) of
      1: writeln("Elégtelen"); end;
      2: writeln("Elégséges"); end;
      3: writeln("Közepes"); end;
      4: writeln("Jó"); end;
      5: writeln("Jeles"); end;
      else: writeln("???"); end;
END.
```

Metódusok

- Eljárások
- Függvények

Eljárások

Az eljárások felépítése:

```
procedure eljárás_neve(formális_paraméterlista)
   begin
      eljárás_törzsrésze
      end;
```

A procedure jelzi, hogy eljárásról van szó, tehát nem lesz a metódusnak visszatérési értéke.

Az eljárás neve után mindig szerepel a dupla kerek zárójel (), mely között a formális paraméterlista helyezkedik el. A formális paraméterlista a bemenő paraméterek nevét és típusát tartalmazza. Egy eljárásnak nem feltétlenül van bemenő paramétere, azok száma lehet 0, 1 vagy több is. Ha többet is megadunk, azokat pontosvesszővel választjuk el egymástól.

Függvények

A függvényt a visszatérési érték különbözteti meg az eljárástól.

A függvények felépítése:

```
     function
függvény_neve(formális_paramlista):visszatérési_érték_típ
usa;
         begin
            függvény_törzsrésze
            függvény_neve:=viszatérési_érték;
            end;
```

A visszatérési érték típusát a formális paraméterlista után kell megadnunk, ez lehet egésztípus, logikai, azaz boolean típus, String típus, satöbbi. A formális paraméterlista megadására ugyanaz vonatkozik, mint az eljárások esetében.

A visszatérési értéket a `függvény_neve:=visszatérési_érték;` utasítás adja vissza. Ezt a részt mindig tartalmaznia kell egy

függvénynek, és nem szabad olyan feltételes utasításba helyeznünk, ami lehetséges, hogy nem teljesül, és így az érték visszaadása nem kerül végrehajtásra.

A metódusok a Pascal nyelven írt programok törzsrészében kerülnek meghívásra. A bemenő paraméterek ekkor kapnak értéket. Mikor megadjuk egy metódus meghívásakor a paramétereket, azt aktuális paraméterlistának nevezzük.

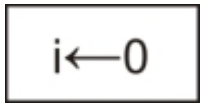
2.1.3 Folyamatábrák

A folyamatábra

Folyamatábrákat a számítógépes programok tervezésénél készítjük. Az ábrák segítik az áttekinthetőséget, és a későbbi hibák kijavítását. Alapelve az, hogy egy feladat megoldásához szükséges lépéseket önálló egységeknek tekinti, és azokat a lépések típusától függően különböző geometriai alakzatokban ábrázolja.
Az alakzatokat nyilakkal kötjük össze, melyek a haladás irányát jelzik.
A folyamatábra tehát a program logikai felépítésének grafikus megjelenítése, legnagyobb jelentősége pedig a programtervezés segítése.

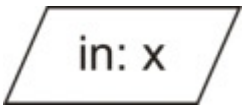
A folyamatábra elemei

- A program kezdetét jelöli.

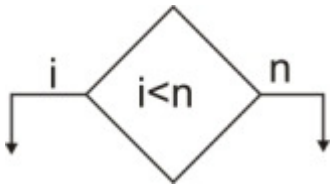


- Értékadás, általános műveletek

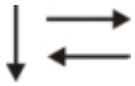
végrehajtását jelöli.



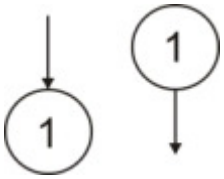
- Input, output (azaz beviteli és kiviteli) műveleteket jelöl.



- Elágazás, vizsgálat. Két iránya van, egy igaz és egy hamis ág.



- Az egységeket nyilakkal, folyamatvonalakkal kötjük össze, ezek jelölik a haladás irányát.



- A folyamatábra rajzolásakor előfordulhat, hogy az nem fér el egybefüggően. A csatlakozást így szokás jelölni.

- A program végét jelöli.

2.2. Programok

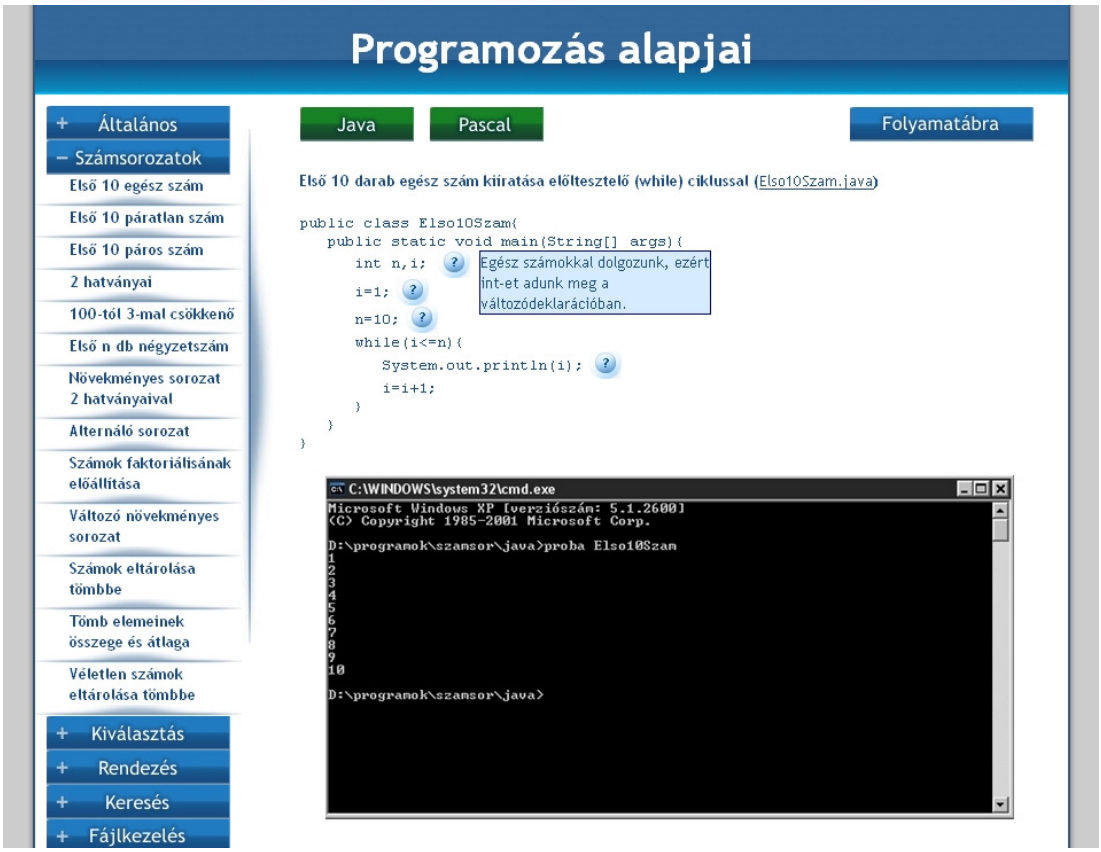
A Programozás alapjai című kurzus során tanult programokat öt nagyobb csoportba foglaltuk, melyek a következők: számsorozatok, kiválasztás, rendezés, keresés, fájlkezelés. Ezek a következő alfejezetekben kerülnek bemutatásra.

Minden egyes program Java és Pascal nyelven is elérhető, ezek között a tartalmi rész tetején található „Java” és „Pascal” feliratú gombok segítségével válthatunk, alapértelmezetten az egyes programok kiválasztásakor a Java nyelven megírt programok jelennek meg. A „Folyamatábra” gombra kattintva az egyes algoritmusokhoz tartozó folyamatábrák új ablakban jelennek meg, így a felhasználó együtt láthatja az összekapcsolódó részeket. A megértés könnyítése érdekében képernyőképeket is készítettünk, így az egyes programok eredménye is látható.

A programokat magyarázatokkal egészítettük ki, melyek a kék kérőjelre kattintva jelennek meg, illetve újabb kattintáskor eltűnnek. Ezt a weboldalon rétegek és JavaScript segítségével oldottuk meg. A következő kódrészlet, illetve kép ezt illusztrálja:

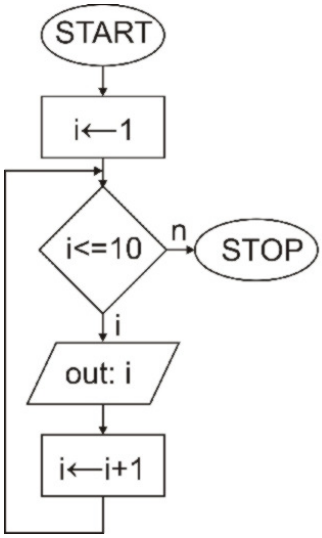
```
…
<SCRIPT>
function reteg(index){
if (document.getElementById(index).style.visibility=='hidden')
{document.getElementById(index).style.visibility='visible';}
else{
document.getElementById(index).style.visibility='hidden';}
}
</SCRIPT>
…
```

Ez a script a forráskódok fejrészében helyezkedik el. Ehhez természetesen kapcsolódnak **`id`**-vel azonosított **`span`** tagok, melynek css formázása alapesetben a következő: **`style="visibility:false; position:absolute;"`**.

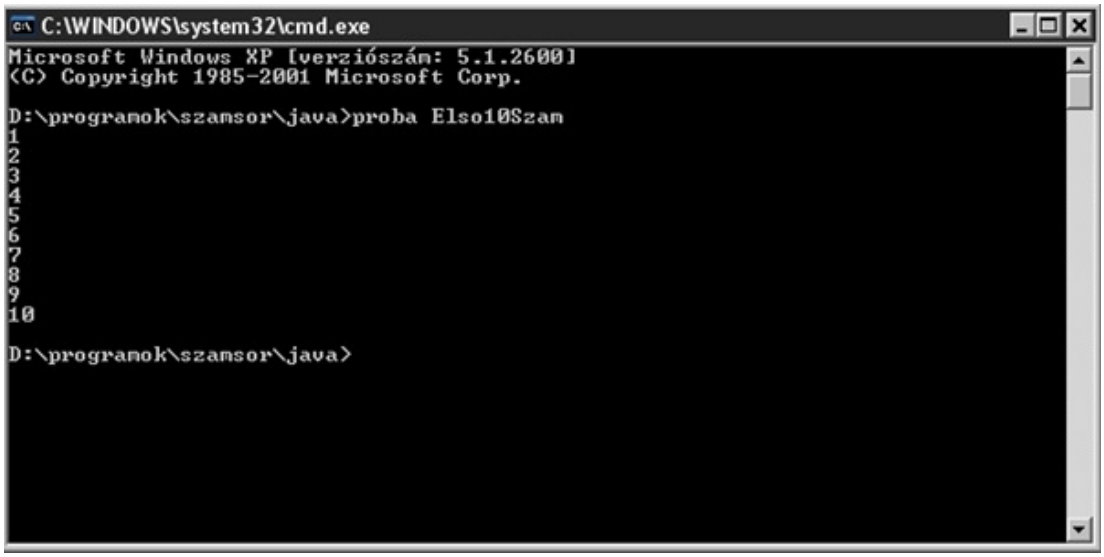


2.2.1 Számsorozatok

Első 10 darab egész szám kiíratása előltesztelő (while) ciklussal (Elso10Szam.java)

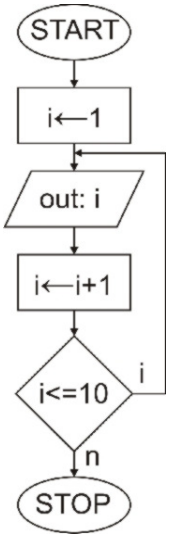



```
public class Elso10Szam{
   public static void main(String[] args){
      int n,i;
/*Egész számokkal dolgozunk, ezért int-et adunk meg a változódeklarációban.*/
      i=1;
/*i = ciklusváltozó, valamint a kiirandó szám. */
      n=10;
/*n = sorozatelemek száma. */
      while(i<=n){
         System.out.println(i);
/*Kiírjuk az i aktuális értékétés a println metódussal, mely a kiírás után sort is tör. */
         i=i+1;
      }
   }
}
```



Első 10 darab egész szám kiíratása for ciklussal (Elso10SzamV1.java)

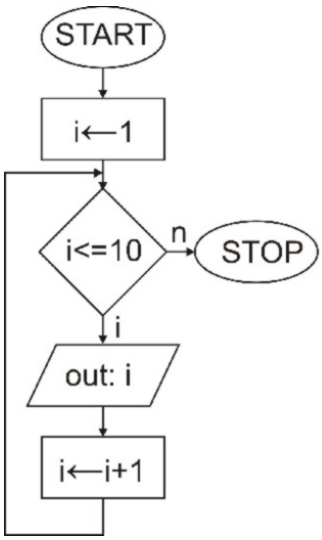
```
public class Elso10SzamV1{
   public static void main(String[] args){
      int n,i;
      n=10;
      for(i=1;i<=n;i=i+1){
         System.out.println(i);
      }
   }
}
```



Első 10 darab egész szám kiíratása hátultesztelő (do - while) ciklussal (Elso10SzamV2.java)

```
public class Elso10SzamV2{
   public static void main(String[] args){
      int n,i;
      n=10;
      i=1;
      do{
         System.out.println(i);
         i=i+1;
      }while(i<=n);
   }
}
```

Első 10 darab egész szám kiíratása előltesztelő (while) ciklussal (Elso10Szam.pas)



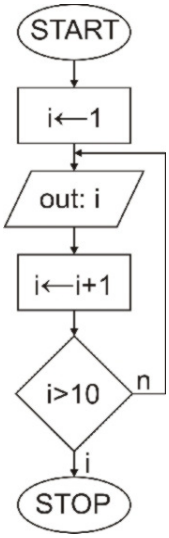

```
program Elso10Szam;
uses crt;
var i, n:integer;
{Egész számokkal dolgozunk, ezért integer-t adunk meg a változódeklarációban.}
BEGIN
   n:=10;
{n = sorozatelemek száma}
   i:=1;
{i = ciklusváltozó, valamint a kiirandó szám}
   while(i<=n) do begin
      writeln(i);
{Kiírjuk az i aktuális értékét a writeln metódussal, mely a kiírás után sort is tör.}
      i:=i+1;
```

```
      end;
END.
```

Első 10 darab egész szám kiíratása for ciklussal (Elso10SzamV1.pas)

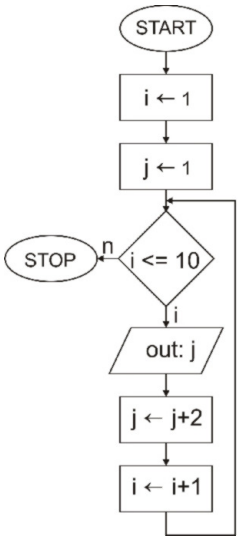
```
program Elso10SzamV1;
uses crt;
var i, n:integer;
BEGIN
   n:=10;
   for i:=1 to n do begin
{A Pascal for ciklusában nem kell megadni a ciklusváltozó növelését, mivel ez magában foglalja.}
      writeln(i);
      end;
END.
```

Első 10 darab egész szám kiíratása hátultesztelő (repeat - until) ciklussal (Elso10SzamV2.pas)

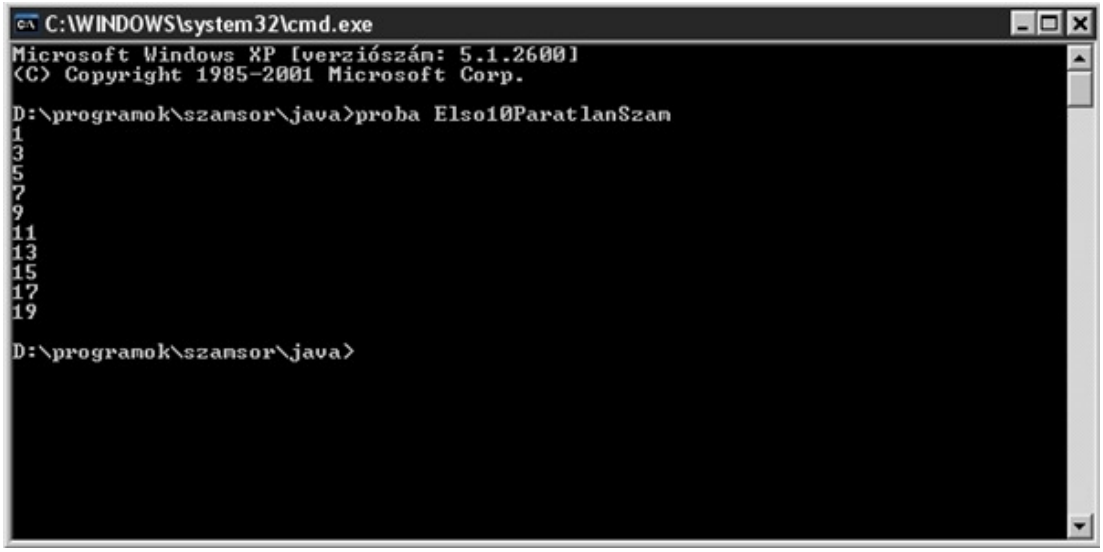



```
program Elso10SzamV2;
uses crt;
var i, n:integer;
BEGIN
   n:=10;
   i:=1;
   repeat
      writeln(i);
      i:=i+1;
   until(i>n);
```
{A Pascal hátultesztelő ciklusánál megfordul a ciklusfeltétel, mivel itt kilépési feltételt kell megadni. }
```
END.
```

Első 10 darab páratlan szám kiíratása előltesztelő (while) ciklussal (Elso10ParatlanSzam.java)



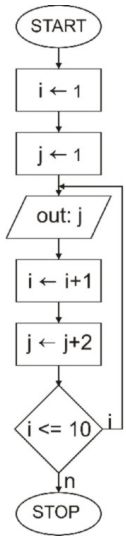

```
public class Elso10ParatlanSzam{
   public static void main(String[] args){
      int n,i,a;
      n=10;
/*n = sorozatelemek száma */
      i=1;
/*i = ciklusváltozó */
      a=1;
/*a = kiirandó érték */
      while(i<=n){
         System.out.println(a);
/*Kiírjuk az a változó aktuális értékét.*/
         a=a+2;
/*Megnöveljük az a változó értékét 2-vel, így a következő páratlan számot kapjuk.*/
         i=i+1;
      }
   }
}
```



Első 10 darab páratlan szám kiíratása for ciklussal (Elso10ParatlanSzamV1.java)

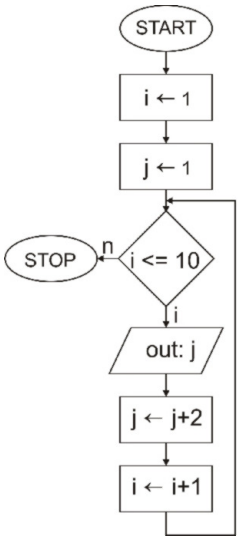
```
public class Elso10ParatlanSzamV1{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      a=1;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a+2;
      }
   }
}
```

Első 10 darab páratlan szám kiíratása hátultesztelő (do - while) ciklussal (Elso10ParatlanSzamV2.java)



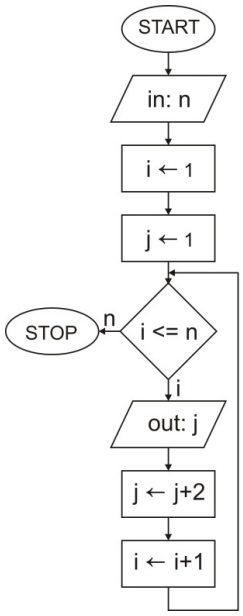

```
public class Elso10ParatlanSzamV2{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=1;
      do{
         System.out.println(a);
         a=a+2;
         i=i+1;
      }while(i<=n);
   }
}
```

Első 10 darab páratlan szám kiíratása előltesztelő (while) ciklussal (Elso10ParatlanSzam.pas)




```
program Elso10ParatlanSzam;
uses crt;
var i,n,a:integer;
BEGIN
   n:=10;
{n = sorozatelemek száma }
   i:=1;
{i = ciklusváltozó }
   a:=1;
{a = kiirandó érték }
   while(i<=n) do begin
      writeln(a);
{Kiírjuk az a változó aktuális értékét.}
      a:=a+2;
{Megnöveljük az a változó értékét 2-vel, így a következő páratlan számot kapjuk.}
      i:=i+1;
      end;
END.
```

Első n darab páratlan szám kiíratása előltesztelő (while) ciklussal (ElsoNParatlanSzam.pas)

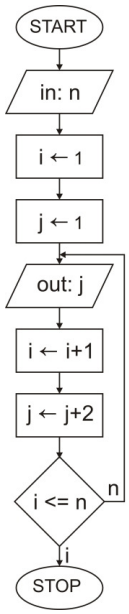



```
program ElsoNParatlanSzam;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
```
{Kiírjuk a képernyőre az "n:=" szöveget, így jelezvén, hogy számot kérünk be.}
```
   readln(n);
```
{Beolvassuk a billentyűzetről az értéket, amely az ENTER billentyű lenyomása után kerül eltárolásra. Ezután ezzel az értékkel dolgozunk tovább a ciklusban.}
```
   i:=1;
   a:=1;
   while(i<=n) do begin
      writeln(a);
      a:=a+2;
      i:=i+1;
      end;
END.
```

Első n darab páratlan szám kiíratása for ciklussal (ElsoNParatlanSzamV1.pas)

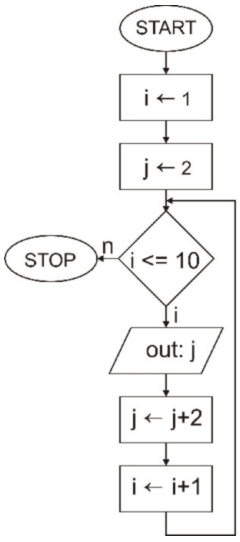
```
program ElsoNParatlanSzamV1;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=1;
   for i:=1 to n do begin
      writeln(a);
      a:=a+2;
      end;
END.
```

Első n darab páratlan szám kiíratása hátultesztelő (repeat - until) ciklussal (ElsoNParatlanSzamV2.pas)

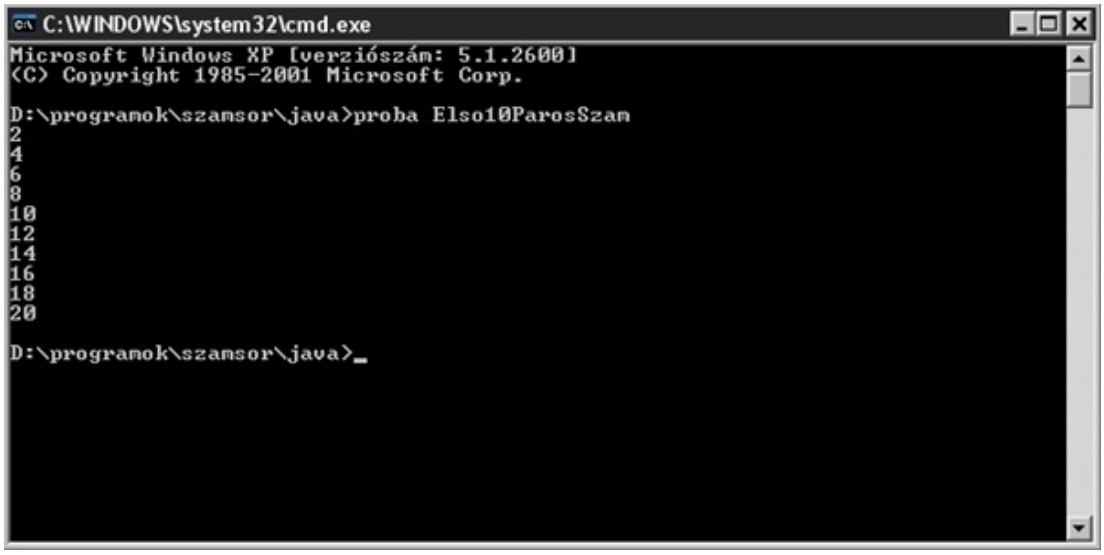



```
program ElsoNParatlanSzamV2;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   repeat
      writeln(a);
      a:=a+2;
      i:=i+1;
   until(i>n);
END.
```

Első 10 darab páros szám kiíratása előltesztelő (while) ciklussal (Elso10ParosSzam.java)

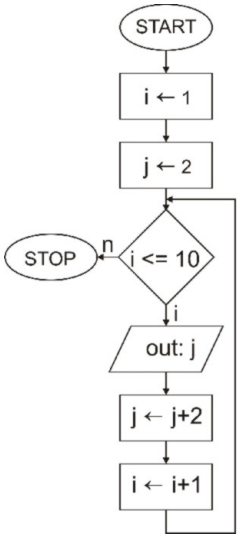



```
public class Elso10ParosSzam{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=2;
/*a = kiírandó érték, ami jelen esetben 2, mivel ez az első pozitív páros szám*/
      while(i<=n){
         System.out.println(a);
         a=a+2;
/*Megnöveljük az a változó értékét 2-vel, így a következő páros számot kapjuk.*/
         i=i+1;
      }
   }
}
```



Első 10 darab páros szám kiíratása for ciklussal

(Elso10ParosSzamV1.java)

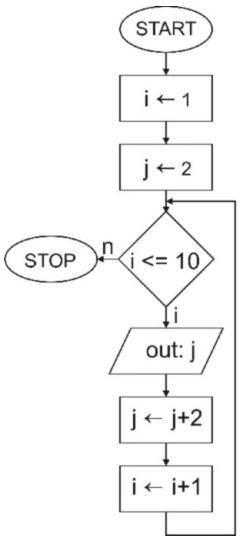



```
public class Elso10ParosSzamV1{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      a=2;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a+2;
      }
   }
}
```

Első 10 darab páros szám kiíratása hátultesztelő (do - while) ciklussal (Elso10ParosSzamV2.java)

```
public class Elso10ParosSzamV2{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=2;
      do{
         System.out.println(a);
         a=a+2;
         i=i+1;
      }while(i<=n);
   }
}
```

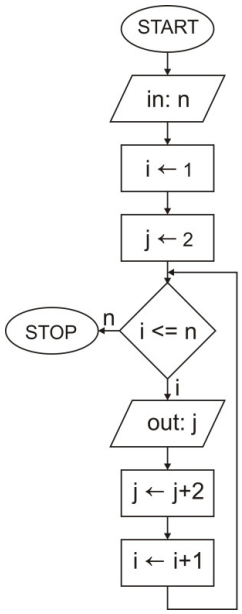
Első 10 darab páros szám kiíratása előltesztelő (while) ciklussal (Elso10ParosSzam.pas)




```
program Elso10ParosSzam;
uses crt;
var i,n,a:integer;
BEGIN
   n:=10;
   i:=1;
   a:=2;
```

```
{a = kiírandó érték, ami jelen esetben 2, mivel ez az első pozitív páros szám}
   while(i<=n) do begin
      writeln(a);
      a:=a+2;
{Megnöveljük az a változó értékét 2-vel, így a következő páros számot kapjuk.}
      i:=i+1;
      end;
END.
```

Első n darab páros szám kiíratása előltesztelő (while) ciklussal (ElsoNParosSzam.pas)




```
program ElsoNParosSzam;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=2;
   while(i<=n) do begin
      writeln(a);
      a:=a+2;
      i:=i+1;
      end;
END.
```

Első n darab páros szám kiíratása for ciklussal (ElsoNParosSzamV1.pas)

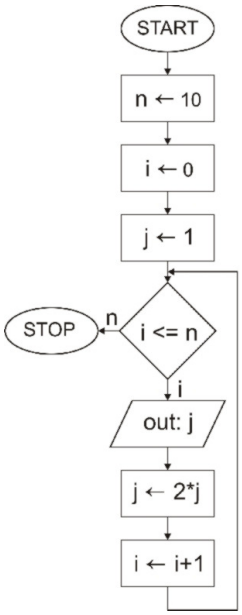
```
program ElsoNParosSzamV1;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=2;
   for i:=1 to n do begin
      writeln(a);
      a:=a+2;
      end;
END.
```

Első n darab páros szám kiíratása hátultesztelő (repeat - until) ciklussal (ElsoNParosSzamV2.pas)

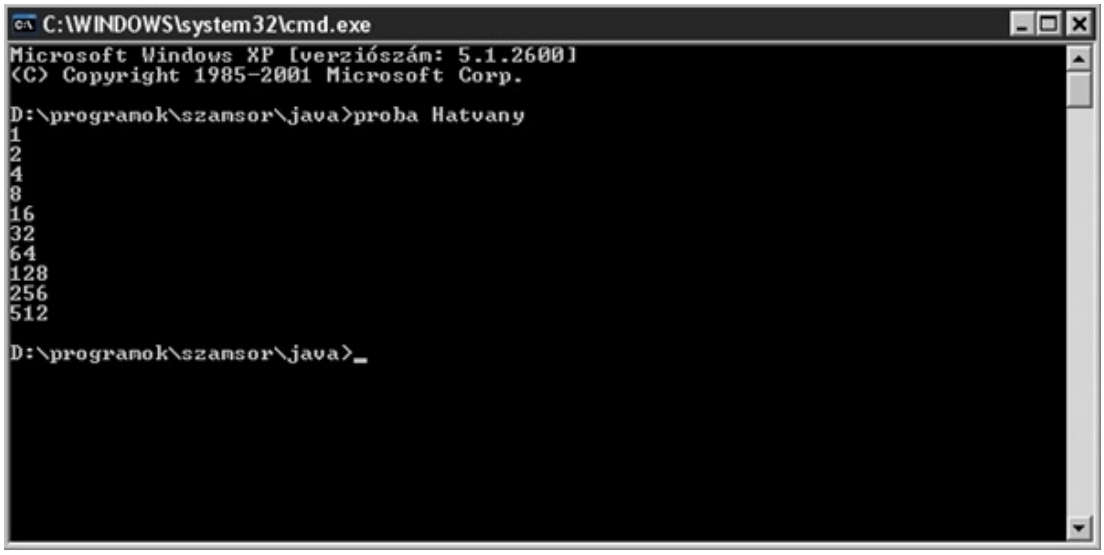
```
program ElsoNParosSzamV2;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=2;
   repeat
      writeln(a);
```

```
      a:=a+2;
      i:=i+1;
   until(i>n);
END.
```

Kettő hatványainak kiíratása előltesztelő (while) ciklussal (Hatvany.java)




```
public class Hatvany{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=1;
      while(i<=n){
         System.out.println(a);
         a=a*2;
/*Mivel hatványozunk, az a változó értékét minden ciklusban megszorozzuk
2-vel.*/
         i=i+1;
      }
   }
}
```



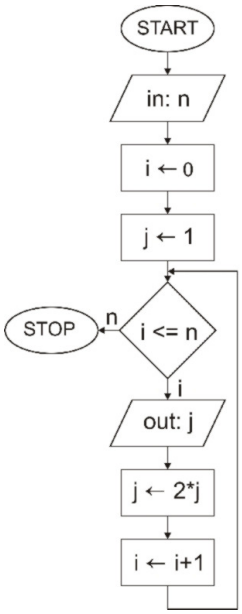
Kettő hatványainak kiíratása for ciklussal (HatvanyV1.java)

```
public class HatvanyV1{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      a=1;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a*2;
      }
   }
}
```

Kettő hatványainak kiíratása hátultesztelő (do - while) ciklussal (HatvanyV2.java)

```
public class HatvanyV2{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=1;
      do{
         System.out.println(a);
         a=a*2;
         i=i+1;
      }while(i<=n);
   }
}
```

Kettő hatványainak kiíratása előltesztelő (while) ciklussal (Hatvany.pas)




```
program Hatvany;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   while(i<=n) do begin
      writeln(a);
      a:=a*2;
{Mivel hatványozunk, az a változó értékét minden ciklusban megszorozzuk 2-vel.}
      i:=i+1;
      end;
END.
```

Kettő hatványainak kiíratása for ciklussal (HatvanyV1.pas)

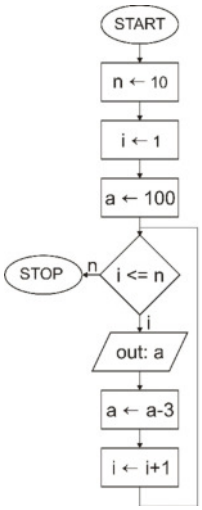
```
program HatvanyV1;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=1;
   for i:=1 to n do begin
      writeln(a);
      a:=a*2;
      end;
END.
```

Kettő hatványainak kiíratása hátultesztelő (repeat - until) ciklussal (HatvanyV2.pas)

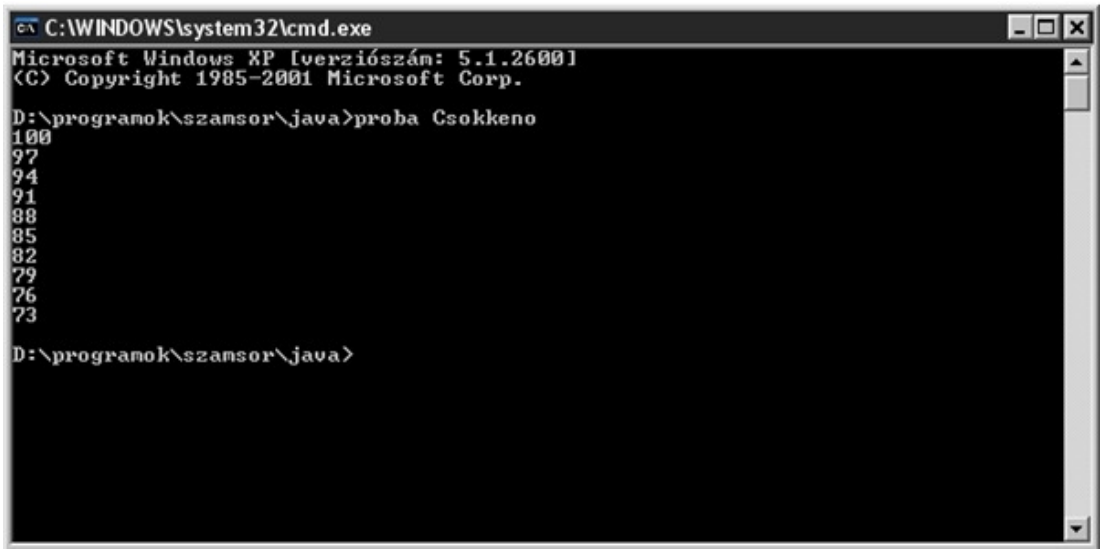
```
program HatvanyV2;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   repeat
      writeln(a);
      a:=a*2;
      i:=i+1;
```

```
   until(i>n);
END.
```

100-tól kezdődően 3-mal csökkenő számsorozat kiíratása előltesztelő (while) ciklussal (Csokkeno.java)




```
public class Csokkeno{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=100;
/*a = 100, melyet minden körben csökkentünk */
      while(i<=n){
         System.out.println(a);
         a=a-3;
/*3-mal csökkentjük az a változó értékét. */
         i=i+1;
      }
   }
}
```



100-tól kezdődően 3-mal csökkenő számsorozat kiíratása for ciklussal (CsokkenoV1.java)

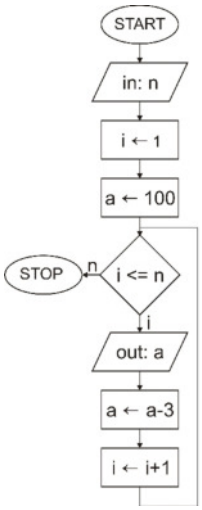
```
public class CsokkenoV1{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      a=100;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a-3;
      }
   }
}
```

100-tól kezdődően 3-mal csökkenő számsorozat kiíratása hátultesztelő (do - while) ciklussal (CsokkenoV2.java)

```
public class CsokkenoV2{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=100;
      do{
         System.out.println(a);
         a=a-3;
         i=i+1;
      }while(i<=n);
```

```
    }
}
```

100-tól kezdődően 3-mal csökkenő számsorozat kiíratása előltesztelő (while) ciklussal (Csokkeno.pas)




```
program Csokkeno;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=100;
{a = 100, melyet minden körben csökkentünk}
   while(i<=n) do begin
      writeln(a);
      a:=a-3;
{3-mal csökkentjük az a változó értékét. }
      i:=i+1;
      end;
END.
```

100-tól kezdődően 3-mal csökkenő számsorozat kiíratása for ciklussal (CsokkenoV1.pas)

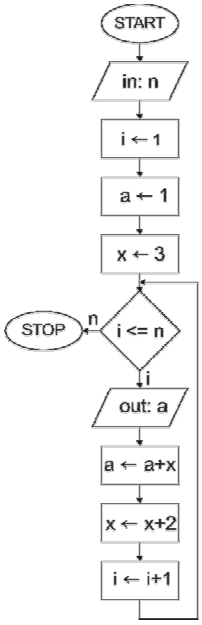
```
program CsokkenoV1;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=100;
   for i:=1 to n do begin
      writeln(a);
      a:=a-3;
      end;
END.
```

100-tól kezdődően 3-mal csökkenő számsorozat kiíratása hátultesztelő (repeat - until) ciklussal (CsokkenoV2.pas)

```
program CsokkenoV2;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=100;
   repeat
      writeln(a);
```

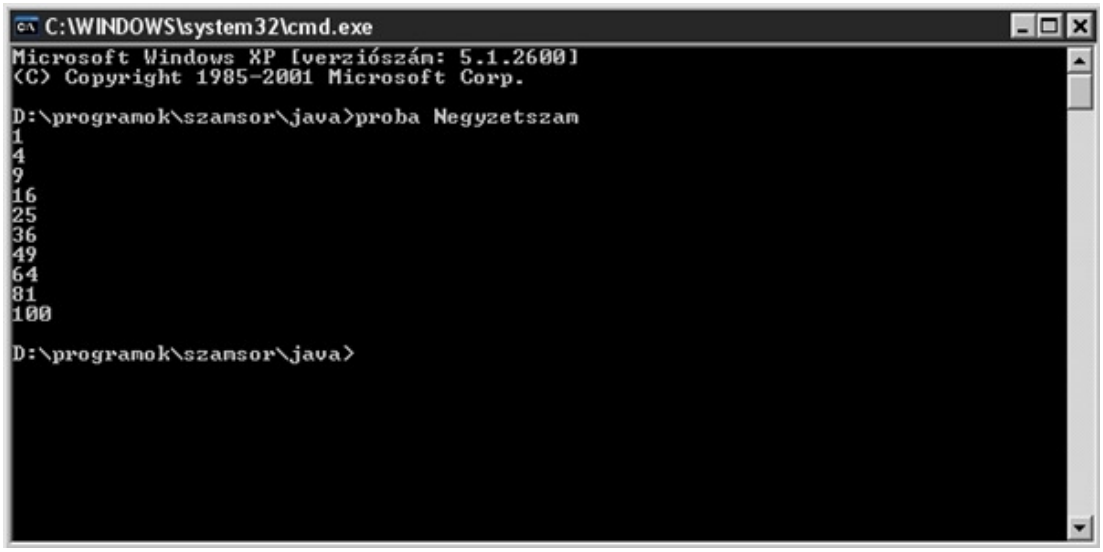
```
      a:=a-3;
      i:=i+1;
   until(i>n);
END.
```

Első 10 darab négyzetszám előltesztelő (while) ciklussal (Negyzetszam.java)




```
public class Negyzetszam{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      i=1;
      a=1;
/*a = kiírandó szám, azaz sorozatelem */
      k=3;
/*k = növekmény, amit mindig növelni kell 2-vel */
      while(i<=n){
         System.out.println(a);
         a=a+k;
/*a-hoz hozzáadjuk az aktuális növekményt */
         i=i+1;
         k=k+2;
/*A növekményt növeljük 2-vel.*/
      }
   }
```

```
}
```



Első 10 darab négyzetszám for ciklussal (NegyzetszamV1.java)

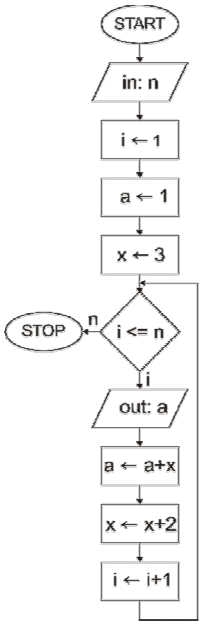
```
public class NegyzetszamV1{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      a=1;
      k=3;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a+k;
         k=k+2;
      }
   }
}
```

Első 10 darab négyzetszám hátultesztelő (do - while) ciklussal (NegyzetszamV2.java)

```
public class NegyzetszamV2{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      i=1;
      a=1;
      k=3;
      do{
```

```
            System.out.println(a);
            a=a+k;
            i=i+1;
            k=k+2;
         }while(i<=n);
   }
}
```

Első n darab négyzetszám előltesztelő (while) ciklussal (Negyzetszam.pas)




```
program Negyzetszam;
uses crt;
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
{a = kiírandó szám, azaz sorozatelem }
   k:=3;
{k = növekmény, amit mindig növelni kell 2-vel }
   while(i<=n) do begin
      writeln(a);
      a:=a+k;
{a-hoz hozzáadjuk az aktuális növekményt }
```

```
        k:=k+2;
{A növekményt növeljük 2-vel. }
        i:=i+1;
        end;
END.
```

Első n darab négyzetszám for ciklussal (NegyzetszamV1.pas)

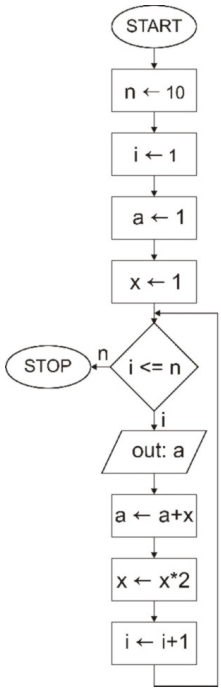
```
program NegyzetszamV1;
uses crt;
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=1;
   k:=3;
   for i:=1 to n do begin
      writeln(a);
      a:=a+k;
      k:=k+2;
      end;
END.
```

Első n darab négyzetszám hátultesztelő (repeat - until) ciklussal (NegyzetszamV2.pas)

```
program NegyzetszamV2;
uses crt;
```

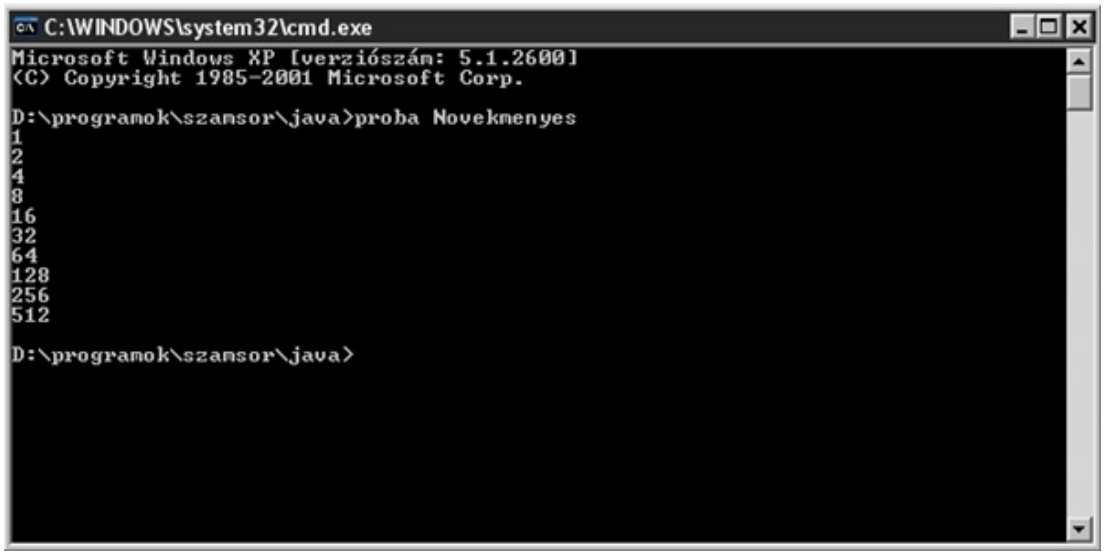
```
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   k:=3;
   repeat
      writeln(a);
      a:=a+k;
      k:=k+2;
      i:=i+1;
   until(i>n);
END.
```

Növekményes sorozat kettő hatványaival, előltesztelő (while) ciklussal (Novekmenyes.java)




```
public class Novekmenyes{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      i=1;
      a=1;
```

```
        k=1;
/*k = változó növekmény */
        while(i<=n){
            System.out.println(a);
            a=a+k;
/*Hozzáadjuk az a változóhoz a változó növekményt.*/
            i=i+1;
            k=k*2;
/*A változó növekményt szorozzuk 2-vel.*/
        }
    }
}
```



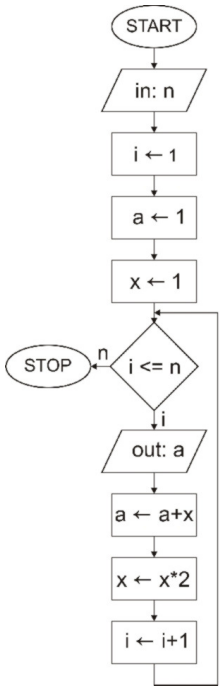
Növekményes sorozat kettő hatványaival, for ciklussal (NovekmenyesV1.java)

```
public class NovekmenyesV1{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      a=1;
      k=1;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a+k;
         k=k*2;
      }
   }
}
```

Növekményes sorozat kettő hatványaival, hátultesztelő (do - while) ciklussal (NovekmenyesV2.java)

```
public class NovekmenyesV2{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      i=1;
      a=1;
      k=1;
      do{
         System.out.println(a);
         a=a+k;
         i=i+1;
         k=k*2;
      }while(i<=n);
   }
}
```

Növekményes sorozat kettő hatványaival, előltesztelő (while) ciklussal (Novekmenyes.pas)



```
program Novekmenyes;
uses crt;
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   k:=1;
{k = változó növekmény }
   while(i<=n) do begin
      writeln(a);
      a:=a+k;
{Hozzáadjuk az a változóhoz a változó növekményt.}
      k:=k*2;
{A változó növekményt szorozzuk 2-vel. }
      i:=i+1;
      end;
END.
```

Növekményes sorozat kettő hatványaival, for ciklussal (NovekmenyesV1.pas)

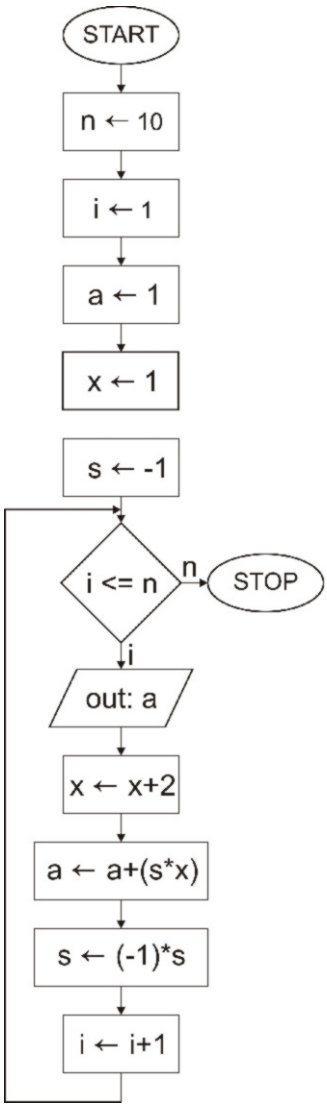
```
program NovekmenyesV1;
uses crt;
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
```

```
   a:=1;
   k:=1;
   for i:=1 to n do begin
      writeln(a);
      a:=a+k;
      k:=k*2;
      end;
END.
```

Növekményes sorozat kettő hatványaival, hátultesztelő (repeat - until) ciklussal (NovekmenyesV2.pas)

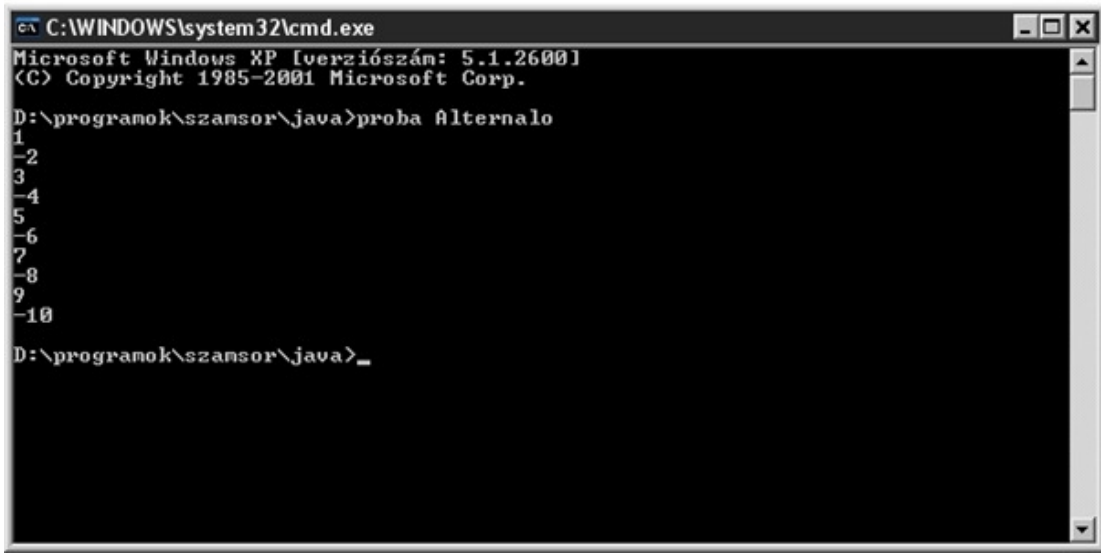
```
program NovekmenyesV2;
uses crt;
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   k:=1;
   repeat
      writeln(a);
      a:=a+k;
      k:=k*2;
      i:=i+1;
   until(i>n);
END.
```

Alternáló sorozat, előltesztelő (while) ciklussal (Alternalo.java)




```
public class Alternalo{
   public static void main(String[] args){
      int n,i,a,x,s;
      n=10;
      i=1;
      a=1;
      s=-1;
/*Ezzel a változóval módosítjuk az érték előjelét.*/
      x=1;
/*x = változó növekmény */
      while(i<=n){
         System.out.println(a);
         x=x+2;
/*A növekményt növeljük 2-vel. Ez az előjelváltáshoz szükséges összeadás
miatt kell.*/
```

```
            a=a+x*s;
/*A sorozatelem aktuális értékéhez hozzáadjuk a változó előjelű növekményt.
/*
            s=s*(-1);
/*Megváltoztatjuk az s változó előjelét.*/
            i=i+1;
            }
        }
    }
```



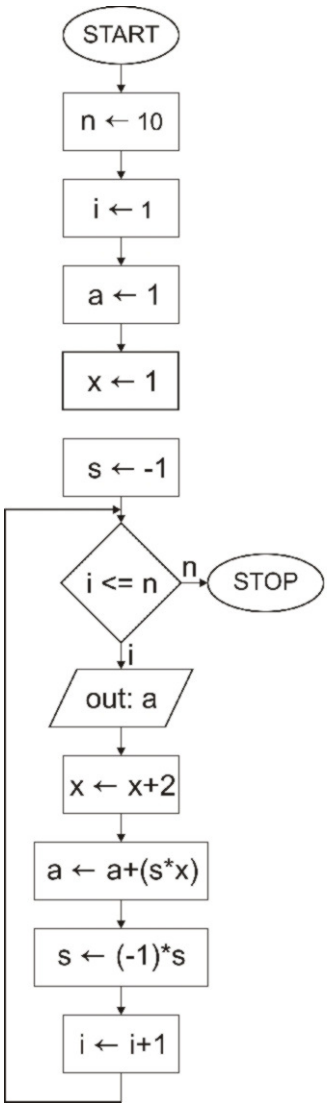
Alternáló sorozat, for ciklussal (AlternaloV1.java)

```
public class AlternaloV1{
   public static void main(String[] args){
      int n,i,a,s,x;
      n=10;
      a=1;
      s=-1;
      x=1;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         x=x+2;
         a=a+x*s;
         s=s*(-1);
         }
      }
   }
```

Alternáló sorozat, hátultesztelő (do - while) ciklussal (AlternaloV2.java)

```
public class AlternaloV2{
   public static void main(String[] args){
      int n,i,a,s,x;
      n=10;
      i=1;
      a=1;
      s=-1;
      x=1;
      do{
         System.out.println(a);
         x=x+2;
         a=a+x*s;
         s=s*(-1);
         i=i+1;
         }while(i<=n);
      }
   }
```

Alternáló sorozat, előltesztelő (while) ciklussal (Alternalo.pas)




```
program Alternalo;
uses crt;
var i,a,s,x,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   s:=-1;
{Ezzel a változóval módosítjuk az érték előjelét. }
   x:=1;
{x = változó növekmény }
   while(i<=n) do begin
      writeln(a);
      x:=x+2;
```

```
{A növekményt növeljük 2-vel. Ez az előjelváltáshoz szükséges összeadás
miatt kell. }
        a:=a+s*x;
{A sorozatelem aktuális értékéhez hozzáadjuk a változó előjelű növekményt.}
        s:=s*(-1);
{Megváltoztatjuk az s változó előjelét.}
        i:=i+1;
        end;
END.
```

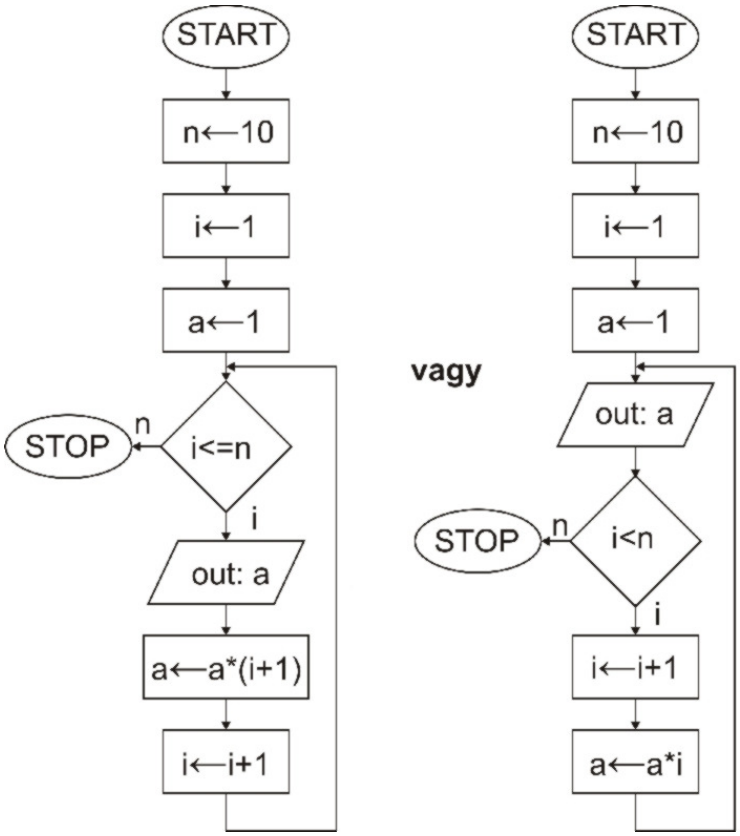
Alternáló sorozat, for ciklussal (AlternaloV1.pas)

```
program AlternaloV1;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=1;
   s:=-1;
   x:=1;
   for i:=1 to n do begin
      writeln(a);
      x:=x+2;
      a:=a+s*x;
      s:=s*(-1);
      end;
END.
```

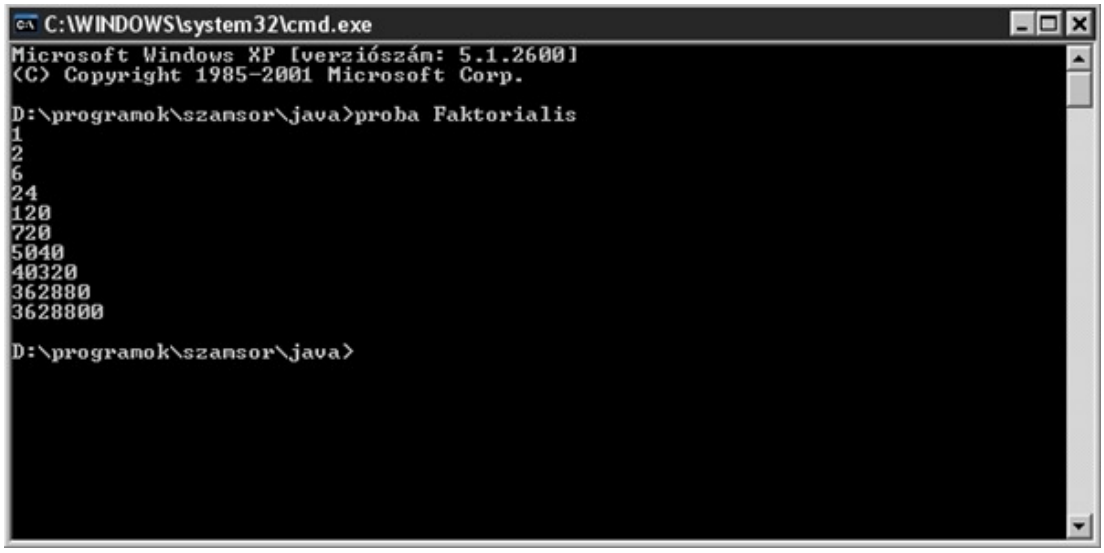
Alternáló sorozat, hátultesztelő (repeat - until) ciklussal (AlternaloV2.pas)

```
program AlternaloV2;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   s:=-1;
   x:=1;
   repeat
      writeln(a);
      x:=x+2;
      a:=a+s*x;
      s:=s*(-1);
      i:=i+1;
   until(i>n);
END.
```

Számok faktoriálisának kiíratása előltesztelő (while) ciklussal (Faktorialis.java)




```
public class Faktorialis{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=1;
      while(i<=n){
         System.out.println(a);
         a=a*(i+1);
/*A sorozatelem értékét beszorozzuk a ciklusváltozónál egyel nagyobb
értékkel. Ezt lerövidthetnénk úgy, hogy a ciklusváltozó növelése után
hajtanánk végre a sorozatelem előállítását (ekkor már csak azt írnánk, hogy
a=a*i;).*/
         i=i+1;
      }
   }
}
```



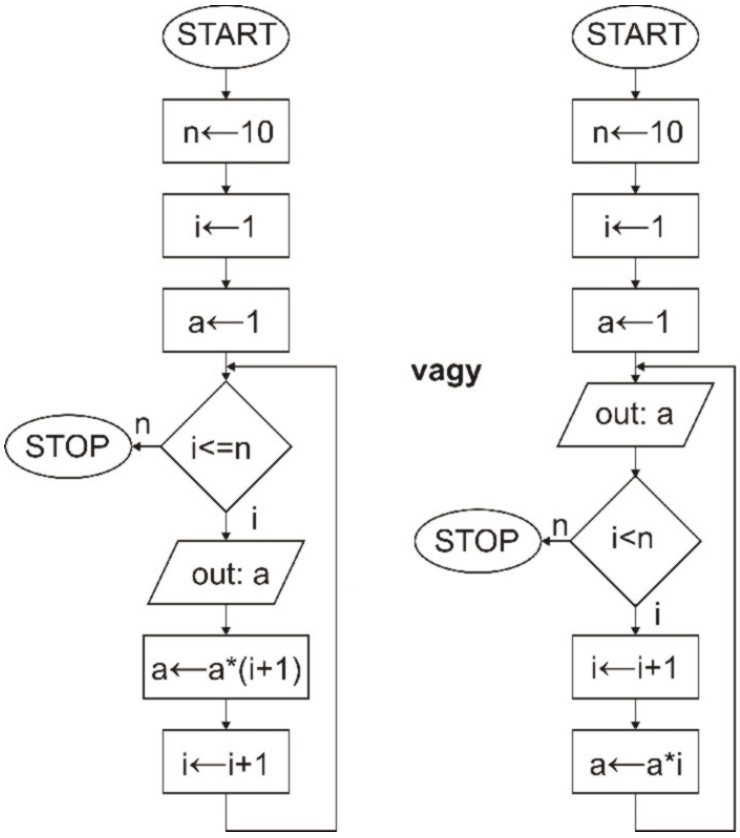
Számok faktoriálisának kiíratása for ciklussal (FaktorialisV1.java)

```
public class FaktorialisV1{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      a=1;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a*(i+1);
      }
   }
}
```

Számok faktoriálisának kiíratása hátultesztelő (do - while) ciklussal (FaktorialisV2.java)

```
public class FaktorialisV2{
   public static void main(String[] args){
      int n,i,a;
      n=10;
      i=1;
      a=1;
      do{
         System.out.println(a);
         a=a*(i+1);
         i=i+1;
      }while(i<=n);
   }
}
```

Számok faktoriálisának kiíratása előltesztelő (while) ciklussal (Faktorialis.pas)




```
program Faktorialis;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   while(i<=n) do begin
      writeln(a);
      a:=a*(i+1);
```

*{A sorozatelem értékét beszorozzuk a ciklusváltozónál egyel nagyobb értékkel. Ezt lerövidthetnénk úgy, hogy a ciklusváltozó növelése után hajtanánk végre a sorozatelem előállítását (ekkor már csak azt írnánk, hogy a=a*i;). }*

```
      i:=i+1;
```

```
      end;
END.
```

Számok faktoriálisának kiíratása for ciklussal (FaktorialisV1.pas)

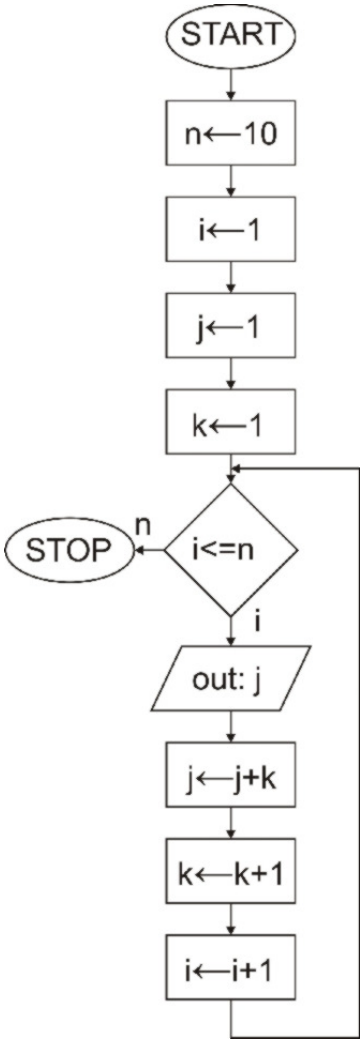
```
program FaktorialisV1;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=1;
   for i:=1 to n do begin
      writeln(a);
      a:=a*(i+1);
      end;
END.
```

Számok faktoriálisának kiíratása hátultesztelő (repeat - until) ciklussal (FaktorialisV2.pas)

```
program FaktorialisV2;
uses crt;
var i,a,n:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
```

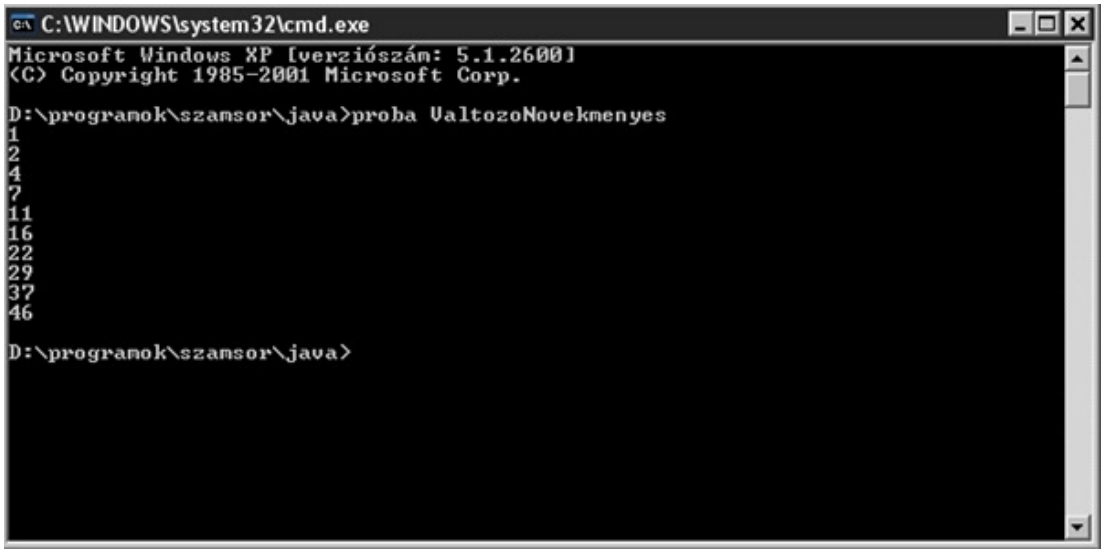
```
   repeat
      writeln(a);
      a:=a*(i+1);
      i:=i+1;
   until(i>n);
END.
```

Változó növekményes sorozat, előltesztelő (while) ciklussal (ValtozoNovekmenyes.java)




```
public class ValtozoNovekmenyes{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
```

```
        i=1;
        a=1;
        k=1;
/*k = változó növekmény */
        while(i<=n){
            System.out.println(a);
            a=a+k;
/*A sorozatelem értékét megnöveljük a változó növekménnyel. */
            k=k+1;
/*A növekményt mindig növeljük egyel. */
            i=i+1;
        }
    }
}
```



Változó növekményes sorozat, for ciklussal (ValtozoNovekmenyesV1.java)

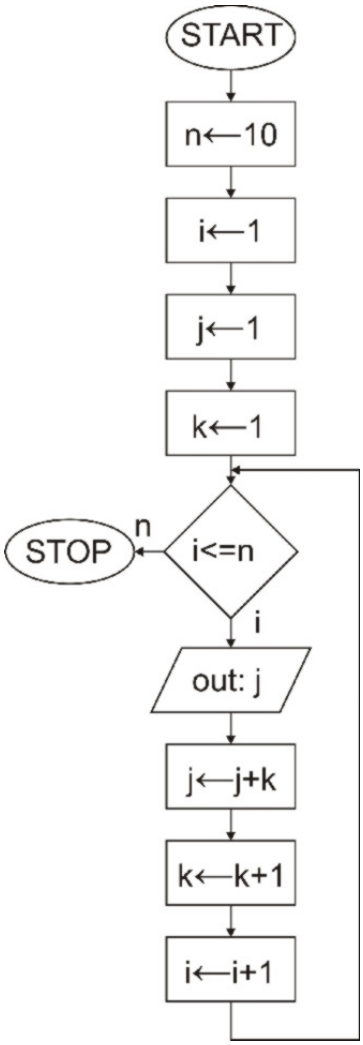
```
public class ValtozoNovekmenyesV1{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      a=1;
      k=1;
      for(i=1;i<=n;i=i+1){
         System.out.println(a);
         a=a+k;
         k=k+1;
      }
   }
```

```
}
```

Változó növekményes sorozat, hátultesztelő (do - while) ciklussal (ValtozoNovekmenyesV2.java)

```
public class ValtozoNovekmenyesV2{
   public static void main(String[] args){
      int n,i,a,k;
      n=10;
      i=1;
      a=1;
      k=1;
      do{
         System.out.println(a);
         a=a+k;
         k=k+1;
         i=i+1;
      }while(i<=n);
   }
}
```

Változó növekményes sorozat, előltesztelő (while) ciklussal (ValtozoNovekmenyes.pas)




```
program ValtozoNovekmenyes;
uses crt;
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   k:=1;
{k = változó növekmény }
   while(i<=n) do begin
      writeln(a);
      a:=a+k;
{A sorozatelem értékét megnöveljük a változó növekménnyel. }
      k:=k+1;
```

```
{A növekményt mindig növeljük eggyel. }
      i:=i+1;
      end;
END.
```

Változó növekményes sorozat, for ciklussal (ValtozoNovekmenyesV1.pas)

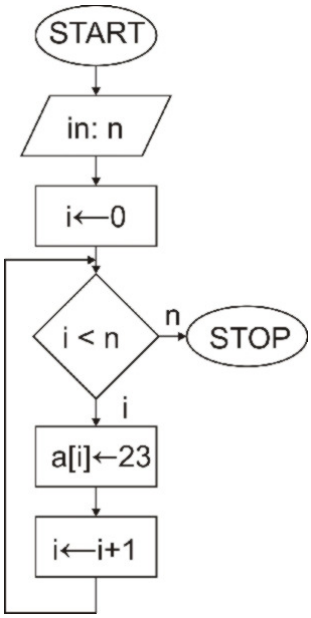
```
program ValtozoNovekmenyesV1;
uses crt;
var i,a,n,k:integer;
BEGIN
   write('n:=');
   readln(n);
   a:=1;
   k:=1;
   for i:=1 to n do begin
      writeln(a);
      a:=a+k;
      k:=k+1;
      end;
END.
```

Változó növekményes sorozat, hátultesztelő (repeat - until) ciklussal (ValtozoNovekmenyesV2.pas)

```
program ValtozoNovekmenyesV2;
uses crt;
var i,a,n,k:integer;
```

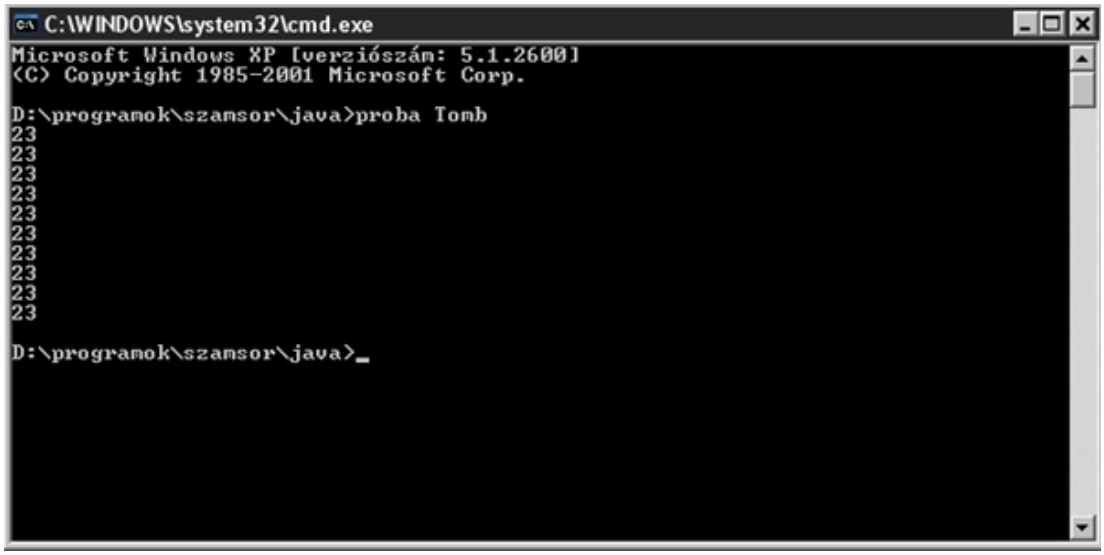
```
BEGIN
   write('n:=');
   readln(n);
   i:=1;
   a:=1;
   k:=1;
   repeat
      writeln(a);
      a:=a+k;
      k:=k+1;
      i:=i+1;
   until(i>n);
END.
```

Számok eltárolása tömbbe, majd az elemek kiíratása (Tomb.java)

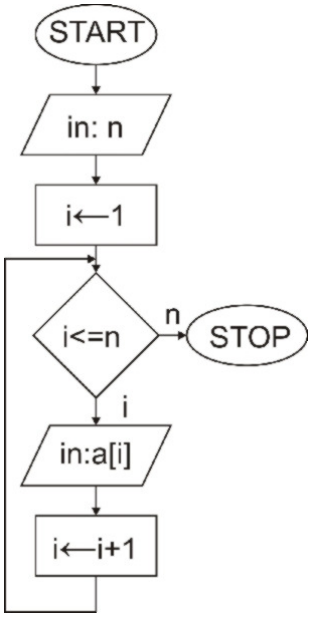



```
public class Tomb{
   public static void main(String[] args){
      int n,i;
      int[] a=new int[10];
/*Egész típusú, a nevű, 10 elemű tömb deklarálása.*/
      i=0;
/*Azért 0-ra álltjuk, mivel a java a tömbökben mindig 0-ról kezdi a számlálást,
azaz az első tömbindex 0. */
      n=10;
      while(i<n){
```

```
            a[i]=23;
/*A 23-mas számot eltároljuk az a tömb i-edik elemében, így a tömböt
feltöltjük 10 db 23-mas számmal. */
            i=i+1;
        }

        i=0;
/*Kinullázzuk a használt ciklusváltozó értékét, hogy újra használhassuk.*/
        while(i<n){
            System.out.println(a[i]);
/*Egy ciklus segtségével kiírjuk a képernyőre a tömb elemeit külön sorban. */
            i=i+1;
            }
        }
    }
```

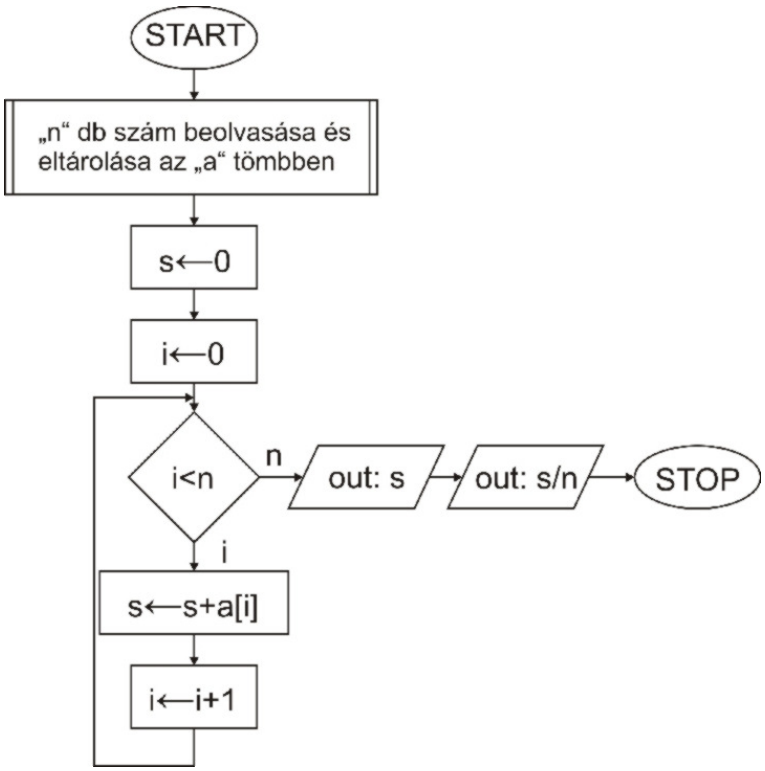


Számok eltárolása tömbbe, majd az elemek kiíratása (Tomb.pas)




```
program Tomb;
uses crt;
var i,n:integer;
    a:array [1..30] of integer;
{Egész típusú, a nevű, 30 elemű tömb deklarálása. }
BEGIN
   write('n:=');
   readln(n);
{Bekérjük a tömbelemek számát (természetesen ez csak kevesebb lehet, mint 30). }
   i:=1;
   while(i<=n) do begin
      write('a[',i,']:=');
{Kiíráskor vesszővel kapcsolhatjuk össze a kiírandó szöveget és változókat. Szöveg kiírásakor ügyeljünk az aposztrófok meglétére.}
      readln(a[i]);
{Beolvassuk a billentyűzetről az a nevű tömb i-edik elemét. }
      i:=i+1;
      end;

   i:=1;
{1-re állítjuk a használt ciklusváltozó értékét. }
   while(i<=n) do begin
{A ciklus segítségével soronként kiírjuk a képernyőre az a tömb elemeit. }
      writeln(a[i]);
      i:=i+1;
      end;
END.
```

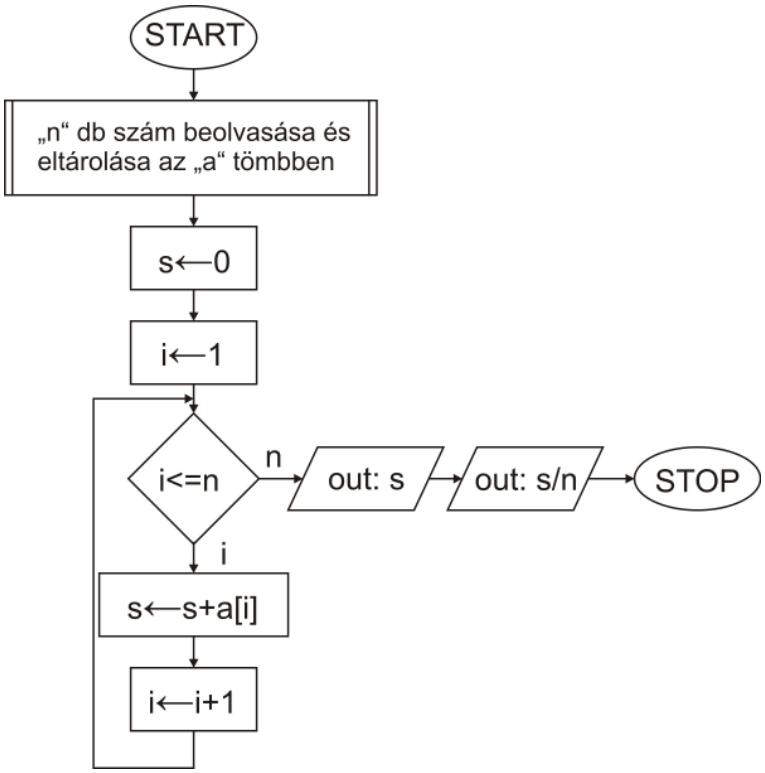
Tömb elemeinek összege és átlaga (OsszegAtlag.java)




```
public class OsszegAtlag{
   public static void main(String[] args){
```

```
        int n,i,s;
        int[] a=new int[10];
        i=0;
        n=10;
        while(i<n){
/*Feltöltjük a tömböt 10 db 23-mas számmal. */
            a[i]=23;
            i=i+1;
        }
        s=0;
/*s-ben fogjuk tárolni a részösszegeket, ezért kezdjük 0-ról */
        i=0;
        while(i<n){
            s=s+a[i];
/*A ciklus segítségével végigmegyünk a tömb elemein és minden tömbelemet
hozzáadunk az s-hez. */
            i=i+1;
            }
        System.out.println("A tomb elemeinek osszege= "+s);
/*  Kiíráskor összeadás jellel kapcsolhatjuk össze a kiírandó szöveget és
változókat. Szöveg kiírásakor ügyeljünk az idézőjelek meglétére. */
        System.out.println("A tomb elemeinek atlaga=
"+s/n);
/*Kiírjuk a tömb elemeinek átlagát (elosztjuk az összeget a tömb elemeinek
számával). */
        }
    }
```

Tömb elemeinek összege és átlaga (OsszegAtlag.pas)



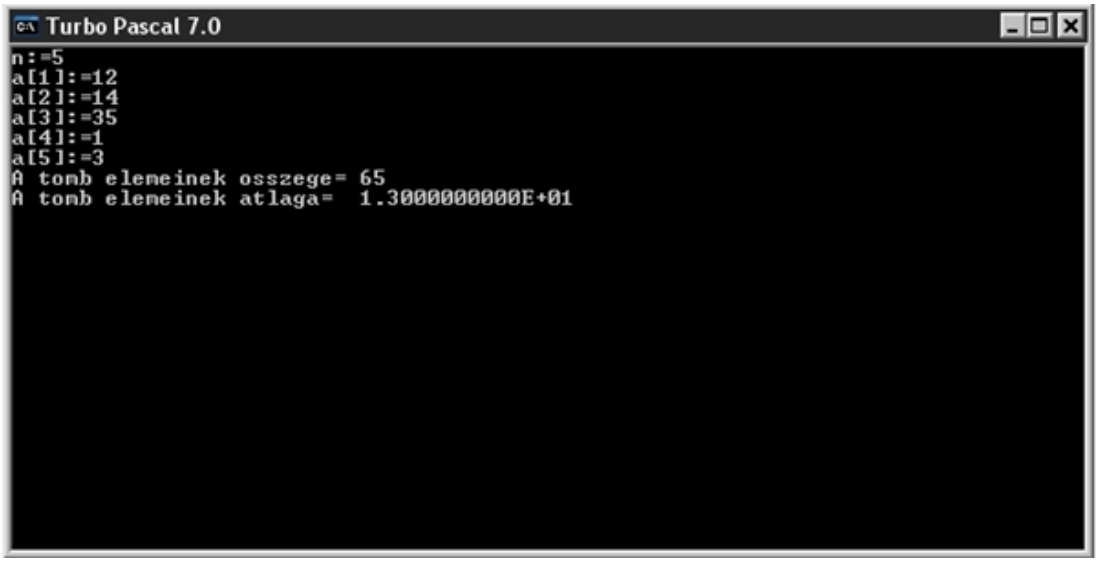

```
program OsszegAtlag;
uses crt;
var i,n,s:integer;
    a:array [1..30] of integer;
BEGIN
   write('n:=');
   readln(n);
```
{Bekérjük a tömbelemek számát (természetesen ez csak kevesebb lehet, mint 30). }
```
   i:=1;
   while(i<=n) do begin
```
{Végigmegyünk a tömb elemein és beolvassuk a billentyűzetről az a nevű tömb i-edik elemét. }
```
      write('a[',i,']:=');
      readln(a[i]);
      i:=i+1;
      end;

   s:=0;
```
{s-ben fogjuk tárolni a részösszegeket, ezért kezdjük 0-ról }
```
   i:=1;
   while(i<=n) do begin
```

{A ciklus segítségével végigmegyünk a tömb elemein és minden tömbelemet hozzáadunk az s-hez }

```
       s:=s+a[i];
       i:=i+1;
       end;

   writeln('A tomb elemeinek osszege= ',s);
   writeln('A tomb elemeinek atlaga= ',s/n);
```

{Kiírjuk a tömb elemeinek átlagát (elosztjuk az összeget a tömb elemeinek számával). }

```
END.
```



Véletlen számok eltárolása tömbbe, majd az elemek kiíratása (Veletlen.java)

```
public class Veletlen {
    public static void main(String[]args){
      int i,n;
      int[] a=new int[100];
```

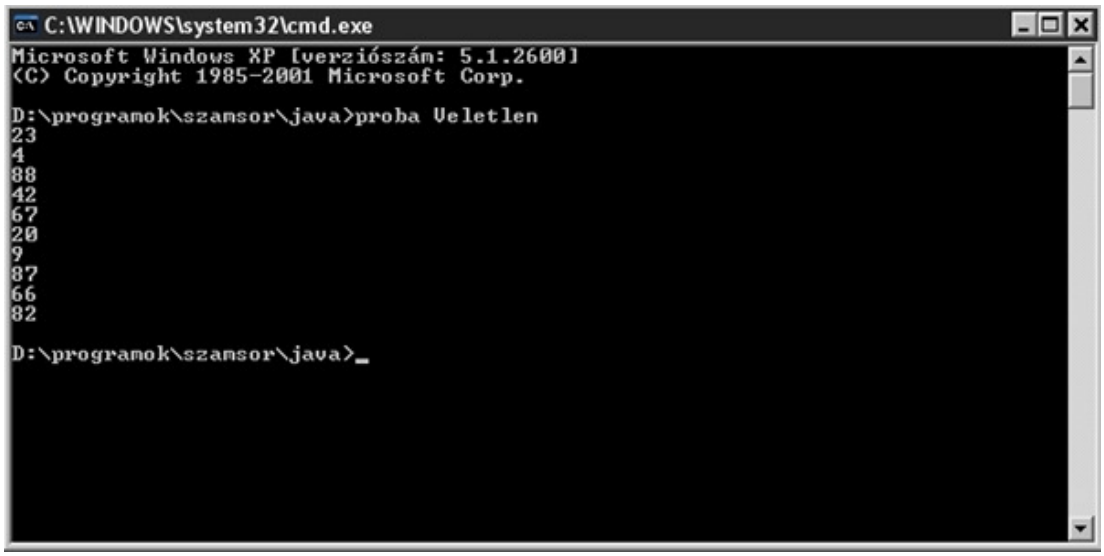
*/*Létrehozunk egy 100 elemű, egész számokból álló tömböt, melyet a-nak nevezünk el.*/*

```
      n=10;
      for (i=0;i<n;i++) {
         a[i]=(int)(90*Math.random()+1);
```

*/*Véletlen számmal töltjük fel a tömb i-edik elemét. Véletlen számot a Math osztály random() függvényével állítunk elő. Ezt 90-el szorozzuk,hogy 0 és 89*

közötti számokat kapjunk, majd hozzáadunk 1-et, hogy 1 és 90 közötti számok legyenek. Ezek viszont egyelőre nem egész számok, ezért típuskényszerítést hajtunk végre az (int) paranccsal./*

```
            }

        i=0;
        while(i<n){
/*Ciklus segítségével kiírjuk a tömb elemeit. */
            System.out.println(a[i]);
            i=i+1;
            }
        }
    }
```



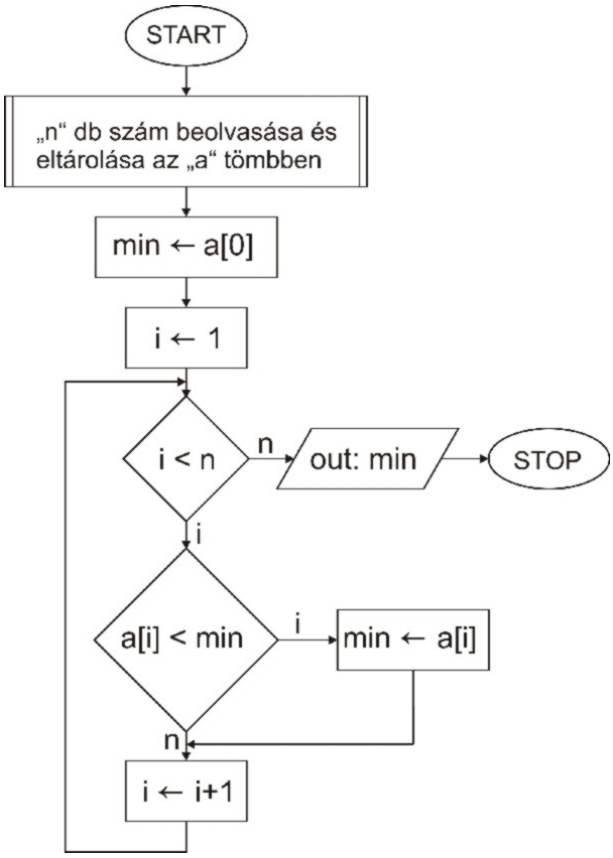
Véletlen számok eltárolása tömbbe, majd az elemek kiíratása (Veletlen.pas)

```
program Veletlen;
uses crt;
var i,n:integer;
    a:array [1..100] of integer;
{Létrehozunk egy 100 elemű, egész számokból álló tömböt, melyet a-nak nevezünk el. }
BEGIN
   write('n:=');
   readln(n);
```

```
{Beolvassuk a tömb elemeinek számát. }
   i:=1;
   while(i<=n) do begin
      a[i]:=trunc(90*random+1);
{Véletlen számokkal töltjük fel a tömb i-edik elemét. Véletlen számot a
random függvénnyel állítunk elő. Ezt 90-el szorozzuk,hogy 0 és 89 közötti
számokat kapjunk, majd hozzáadunk 1-et, hogy 1 és 90 közötti számok
legyenek. Ezek viszont egyenlőre nem egész számok, ezért a trunc
paranccsal levágjuk a szám törtrészét és egész számmá alakítjuk. }
      i:=i+1;
      end;

   i:=1;
   while(i<=n) do begin
{Ciklus segítségével kiírjuk a tömb elemeit. }
      writeln(a[i]);
      i:=i+1;
      end;
END.
```

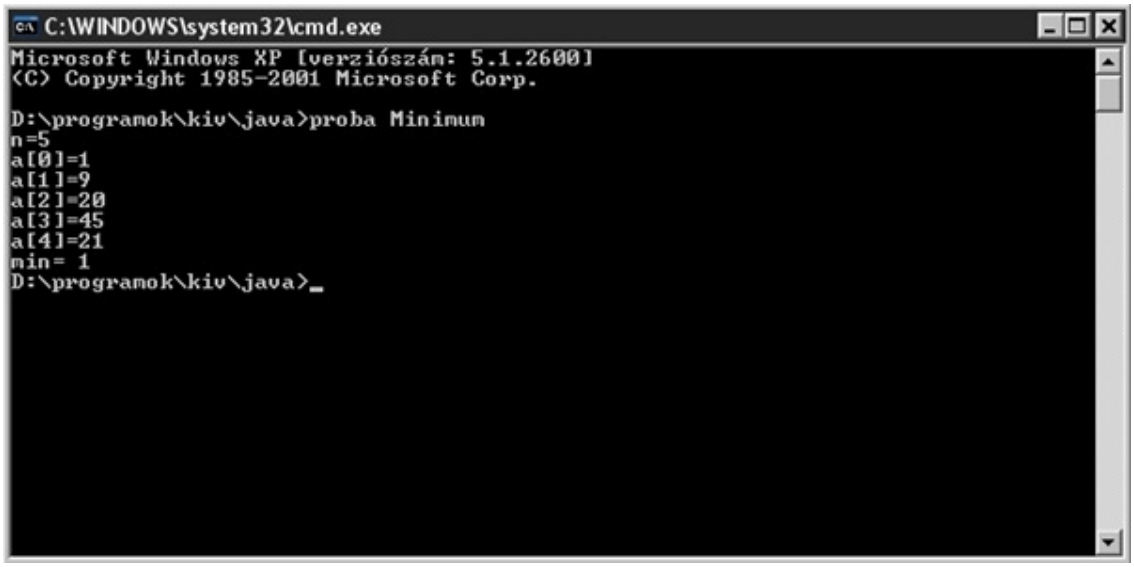
2.2.2 Kiválasztási algoritmusok

Tömb feltöltése n darab bekért számmal, és ezek közül a legkisebb elem kiválasztása (Minimum.java)

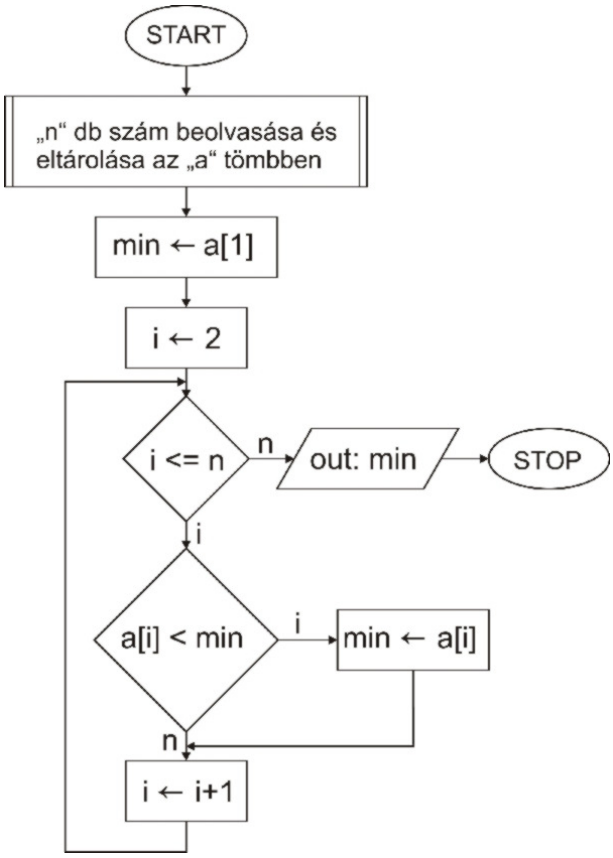



```
import java.io.*;
```
/ Importáljuk az io csomagba tartozó osztályokat. Erre a billentyűzetről történő beolvasás miatt van szükségünk. */*
```
public class Minimum{
   public static int in() throws Exception{
```
/ Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza. A throws Exception arra szolgál, hogy az előforduló hibákat eldobjuk. */*
```
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
```
/ Ahhoz, hogy sorokat olvassunk be, szükségünk van csatornák megnyitására. Jelen esetben létrehozunk egy x nevű LineNumberReader csatornát, beleágyazunk egy InputStreamReader-t, azaz bemeneti csatornát, amely a System.in, azaz billentyűzetről beolvasást azonosítja. */*
```
      String s=x.readLine();
```
/ Létrehozzuk az s nevű, szöveges változót, amelybe 1 sor beolvasott adat kerül. Ez bármilyen karakter lehet, szám és betű is. */*

```
        int i=Integer.parseInt(s);
/* Létrehozunk egy i nevű egész típusú változót, amelybe az s változó kerül
egész számmá alakítva. Kivételkezelésre épp azért van szükség nekünk,
mivel nem csak számok szerepelhetnek az s változóban, azaz a beolvasott
szövegben. */
        return i;
/* A függvény visszaadja az i változó értékét. */
      }
   public static void main(String[] args) throws
Exception{
/* Azért van itt is szükség kivételkezelésre, mivel a main metódusban kerül
meghívásra az in() függvény. */
        int n,i,min;
        int[] a=new int[10];
        System.out.print("n=");
        n=in();
/* Bekérjük a tömbelemek számát, amely nem lehet nagyobb 10-nél, mivel az
a tömb 10 eleműnek lett deklarálva. */
        i=0;
        while (i<n){
/*Feltöltjük az a tömböt az in() függvény segítségével. */
            System.out.print("a["+i+"]=");
            a[i]=in();
            i=i+1;
             }
        min=a[0];
/* A fent egész számnak deklarált min változóba beletesszük az a tömb 0.
indexű elemének értékét. */
        i=1;
/* Mivel a tömb 0. indexű elemét már beleraktuk a min nevű változóba, ezért
az 1. indexű elemmel kezdjük a következő ciklust. */
        while (i<n){
            if (a[i]<min){
/* Amennyiben a min változó értékénél kisebb értékű tömbelemet találunk,
azt beletesszük a min-be. */
                min=a[i];
                }
            i=i+1;
            }
        System.out.print("min= "+min);
        }
   }
```



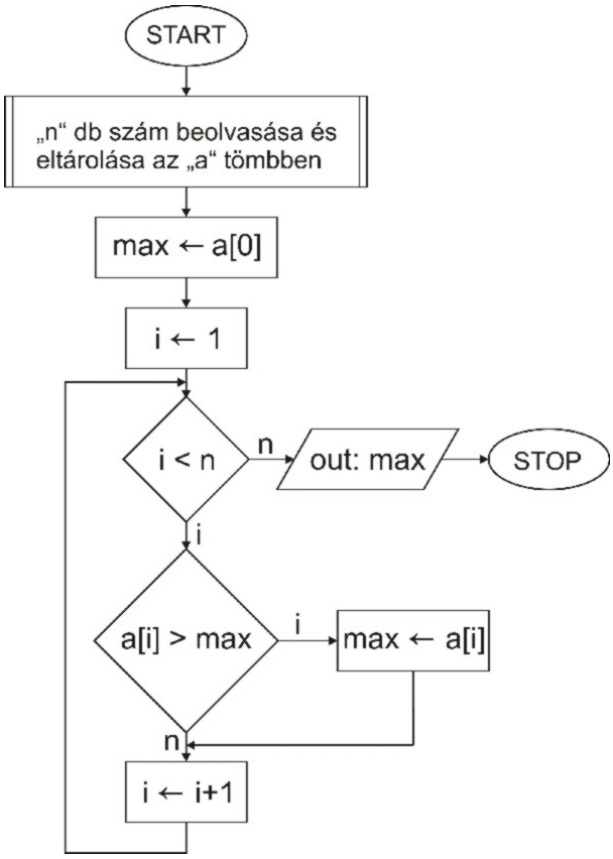
Tömb feltöltése n darab bekért számmal, és ezek közül a legkisebb elem kiválasztása (Minimum.pas)




```
program Minimum;
uses crt;
var i,n,min:integer;
```

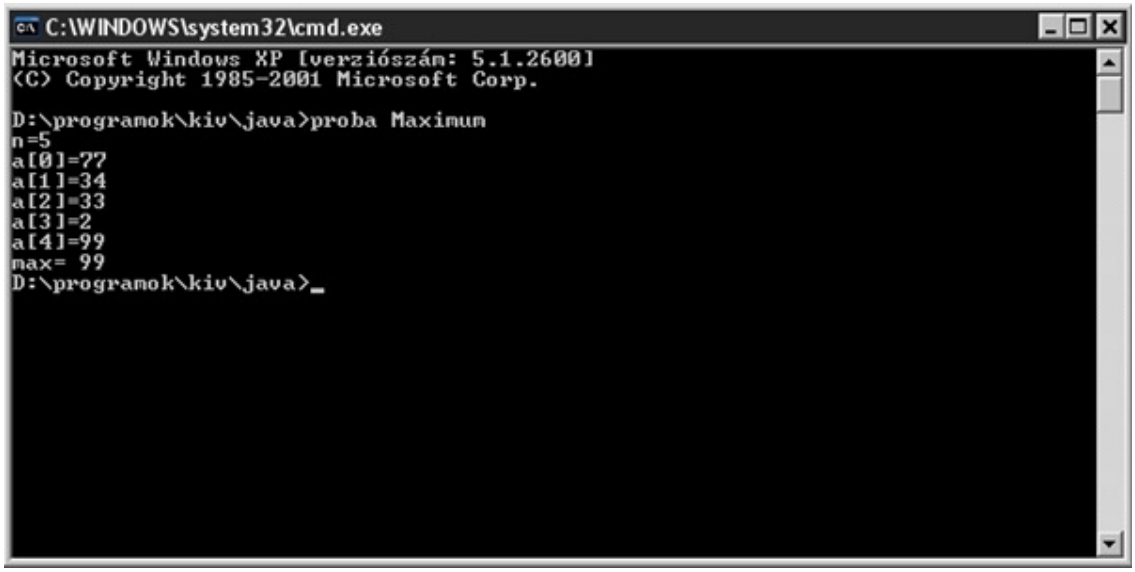
```
    a:array [1..10] of integer;
BEGIN
   write ('n:=');
   readln(n);
```
{Bekérjük a tömbelemek számát, amely nem lehet nagyobb 10-nél, mivel az a tömb 10 eleműnek lett deklarálva. }
```
   i:=1;
   while (i<=n) do begin
```
{A ciklus segítségével feltöltjük a tömböt. }
```
      write ('a[',i,']:=');
      readln (a[i]);
```
{Beolvassuk az a tömb i-edik elemét. }
```
      i:=i+1;
      end;
   min:=a[1];
```
{A fent egész számnak deklarált min változóba beletesszük az a tömb 1. elemének értékét. }
```
   i:=2;
```
{Mivel a tömb 1. indexű elemét már beleraktuk a min nevű változóba, ezért a 2. indexű elemmel kezdjük a következő ciklust. }
```
   while (i<=n) do begin
      if(min>a[i]) then begin
         min:=a[i];
```
{Amennyiben a min változó értékénél kisebb értékű tömbelemet találunk, azt beletesszük a min-be. }
```
         end;
      i:=i+1;
      end;
   writeln('A legkisebb szam= ',min);
END.
```

Tömb feltöltése n darab bekért számmal, és ezek közül a legnagyobb elem kiválasztása (Maximum.java)

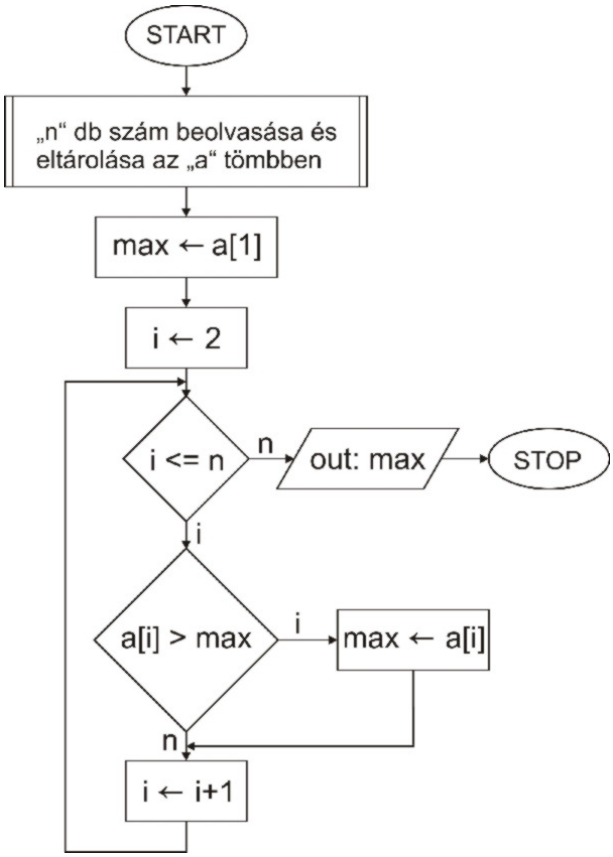



```
import java.io.*;
public class Maximum{
   public static int in() throws Exception{
/* Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza. */
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      String s=x.readLine();
      int i=Integer.parseInt(s);
      return i;
     }
   public static void main(String[] args) throws
Exception{
      int n,i,max;
      int[] a=new int[10];
      System.out.print("n=");
      n=in();
/* Bekérjük a tömbelemek számát, amely nem lehet nagyobb 10-nél, mivel az
a tömb 10 eleműnek lett deklarálva. */
      i=0;
```

```
       while (i<n){
          System.out.print("a["+i+"]=");
/* Feltöltjük az a tömböt az in() függvény segítségével. */
          a[i]=in();
          i=i+1;
           }
    max=a[0];
/* A fent egész számnak deklarált max változóba beletesszük az a tömb 0. indexű elemének értékét. */
    i=1;
    while (i<n){
       if (a[i]>max){
/* Amennyiben a max változó értékénél nagyobb értékű tömbelemet találunk, azt beletesszük a max-ba. */
       max=a[i];
    }
     i=i+1;
    }
    System.out.print("max= "+max);
    }
}
```



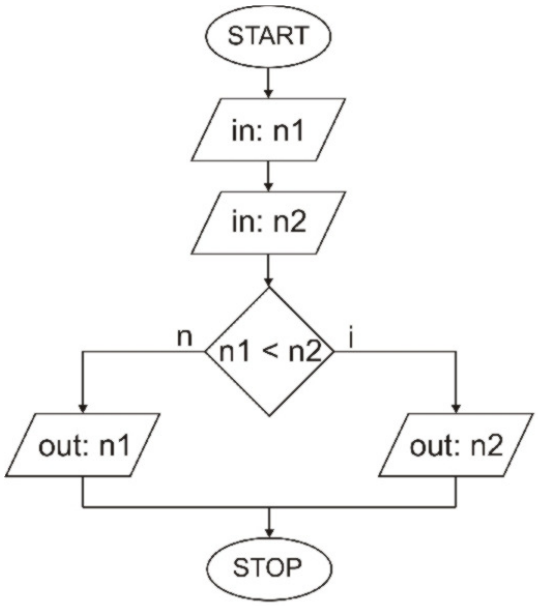
Tömb feltöltése n darab bekért számmal, és ezek közül a legnagyobb elem kiválasztása (Maximum.pas)




```
program Maximum;
uses crt;
var i,n,max:integer;
    a:array [1..10] of integer;
BEGIN
   write ('n:=');
   readln(n);
```
{Bekérjük a tömbelemek számát, amely nem lehet nagyobb 10-nél, mivel az a tömb 10 eleműnek lett deklarálva. }
```
   i:=1;
   while (i<=n) do begin
```
{A ciklus segítségével feltöltjük a tömböt. }
```
      write ('a[',i,']:=');
      readln (a[i]);
```
{Beolvassuk az a tömb i-edik elemét. }
```
      i:=i+1;
      end;
   max:=a[1];
```
{A fent egész számnak deklarált max változóba beletesszük az a tömb 1. elemének értékét. }
```
   i:=2;
```

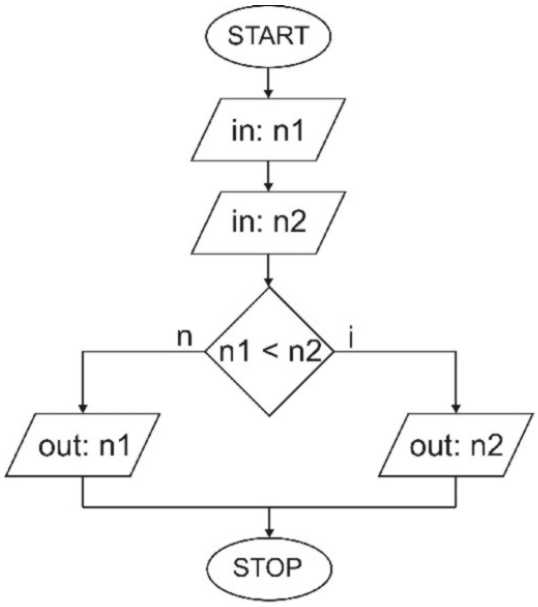
```
   while (i<=n) do begin
      if(max<a[i]) then begin
         max:=a[i];
{Amennyiben a max változó értékénél nagyobb értékű tömbelemet találunk,
azt beletesszük a max-ba. }
         end;
      i:=i+1;
      end;
   writeln('A legnagyobb szam= ',max);
END.
```

Két szám bekérése és összehasonlítása (Hasonlit.java)

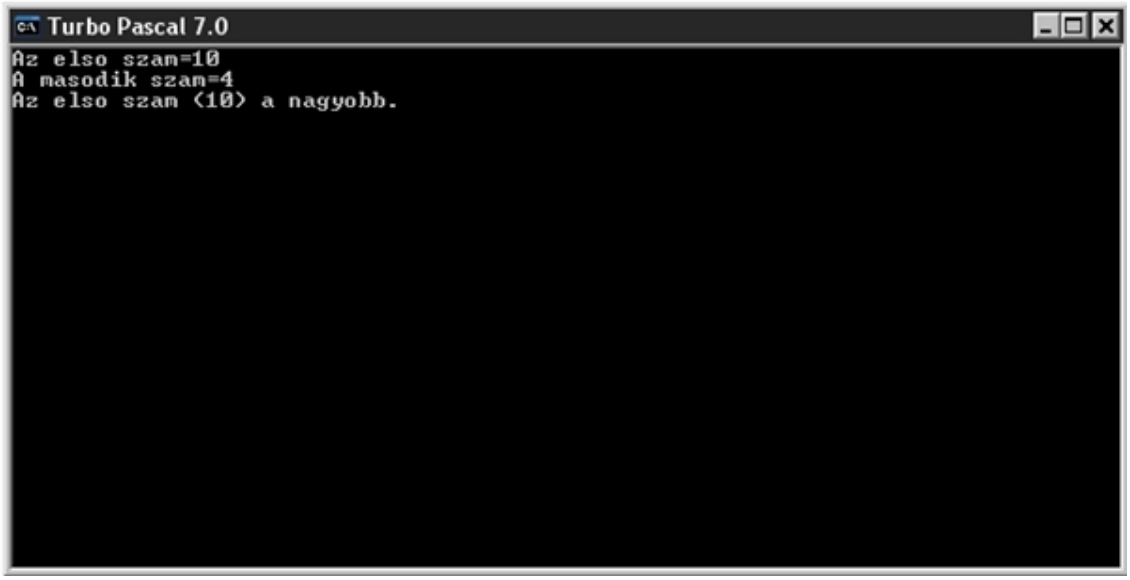


```
import java.io.*;
public class Hasonlit{
   public static int in() throws Exception{
/* Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza. */
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      String s=x.readLine();
      int i=Integer.parseInt(s);
      return i;
     }
   public static void main(String[] args) throws
Exception{
      int n1,n2;
      System.out.print("Az elso szam=");
      n1=in();
/* Bekérjük az első számot és eltároljuk n1-ben. */
      System.out.print("Az masodik szam=");
      n2=in();
/* Bekérjük a második számot és eltároljuk n2-ben. */
      if (n1<n2){
/* Megvizsgáljuk, hogy az első szám kisebb e a másodiknál, ha igen, akkor
kiírjuk, hogy az n2 a nagyobb, egyébként pedig, hogy az n1 a nagyobb. */
         System.out.println("A masodik szam ("+n2+") a
nagyobb.");
         }
      else {
         System.out.println("Az elso szam ("+n1+") a
nagyobb.");
         }
   }
}
```

Két szám bekérése és összehasonlítása (Hasonlit.pas)

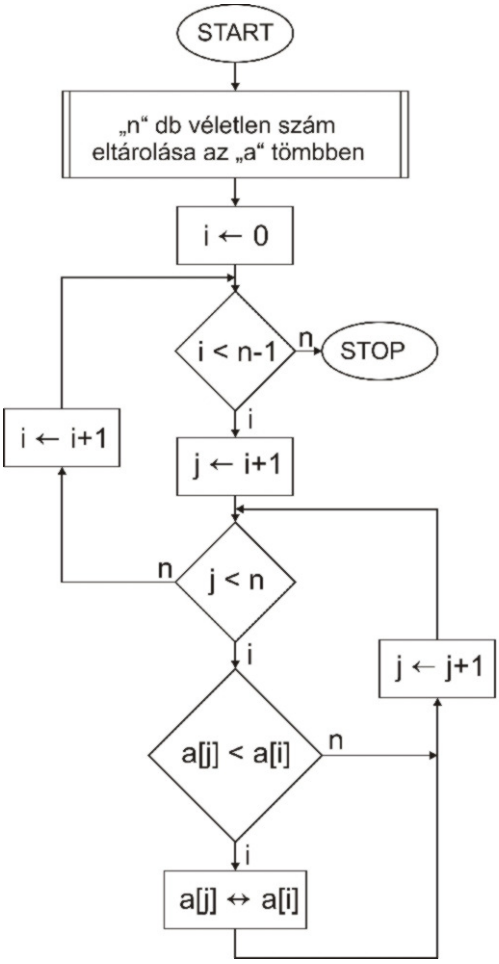



```
program Hasonlit;
uses crt;
var n1,n2:integer;
BEGIN
   write('Az elso szam=');
   readln(n1);
{Bekérjük az első számot és eltároljuk n1-ben. }
   write('A masodik szam=');
   readln(n2);
{Bekérjük a második számot és eltároljuk n2-ben. }
   if (n1<n2) then begin
{Megvizsgáljuk, hogy az első szám kisebb e a másodiknál, ha igen, akkor
kiírjuk, hogy az n2 a nagyobb, egyébként pedig, hogy az n1 a nagyobb. }
      writeln ('A masodik szam (',n2,') a nagyobb.');
      end
{Kétirányú elágazás esetén az else előtti end-et nem zárjuk le ;-vel. }
   else begin
      writeln('Az elso szam (',n1,') a nagyobb.');
      end;
END.
```


Turbo Pascal 7.0
Az elso szam=10
A masodik szam=4
Az elso szam (10) a nagyobb.

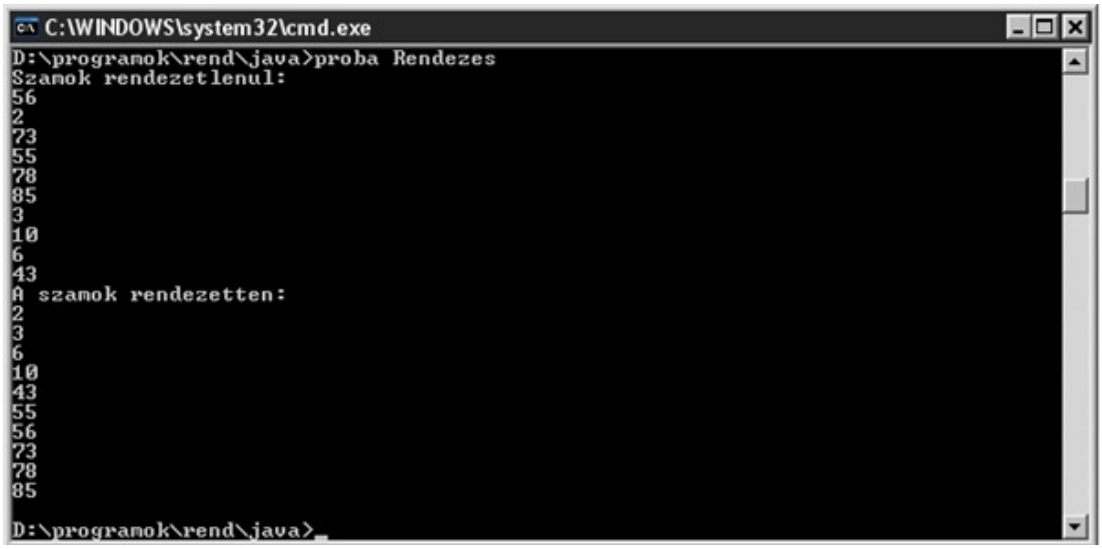
2.2.3 Rendezési algoritmusok

Tömb feltöltése véletlen számokkal és az elemek elrendezése legkisebb elem kiválasztásával (Rendezes.java)

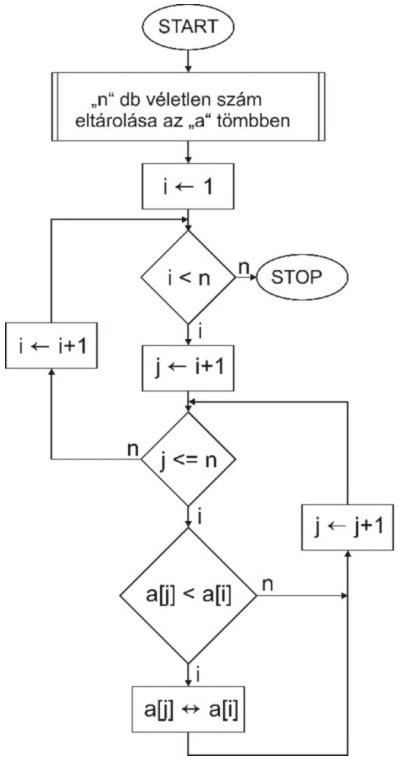



```
public class Rendezes{
   public static void main(String[] args){
      int n,i,j,x;
      int[] a=new int[10];
      n=10;
      i=0;
      while (i<n){  Feltöltjük a tömböt n darab véletlen
számmal.
         a[i]=(int)(90*Math.random()+1);
         i=i+1;
         }
      i=0;
```

```
        System.out.println("Szamok rendezetlenul:");
        while (i<n){
           System.out.println(a[i]);
           i=i+1;
           }
        i=0;
/* A tömb első eleme Java-ban mindig a 0. indexxel rendelkezik. */
        while (i+1<n){
/* A tömb aktuális i-edik eleméhez hasonlítjuk a tömb összes többi elemét. Ha a sorrend nem megfeleleő, akkor cserélünk. Amennyiben elérünk a tömb végére, akkor a következő indexű elemet hasonlítjuk az utána lévőkhöz. Ezt addig folytatjuk, míg végig nem érünk a tömbön. */
           j=i+1;
           while(j<n){
              if (a[j]<a[i]){
/* Ha azt észleljük, hogy a vizsgált elemnél kisebb elemet találtunk, akkor a kettő értékét megcseréljük. */
                 x=a[j];
                 a[j]=a[i];
                 a[i]=x;
                 }
              j=j+1;
              }
           i=i+1;
           }
        i=0;
        System.out.println("A szamok rendezetten:");
        while(i<n){
           System.out.println(a[i]);
           i=i+1;
           }
        }
     }
```



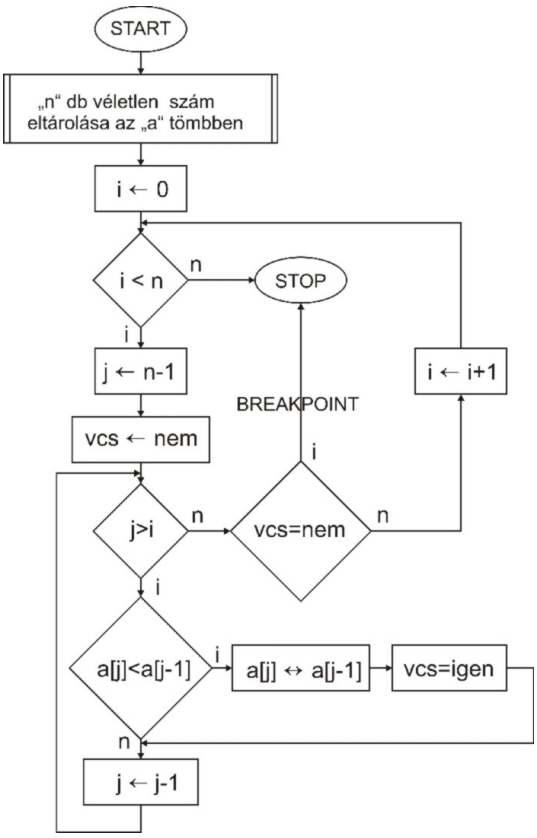
Tömb feltöltése véletlen számokkal és az elemek elrendezése legkisebb elem kiválasztásával (Rendezes.pas)




```
program Rendezes;
uses crt;
var i,n,x,j:integer;
    a:array [1..100] of integer;
BEGIN
   clrscr;
   i:=1;
   write('Hany szam legyen a tombben? ');
   readln(n);
   while (i<=n) do begin
{Feltöltjük a tömböt n darab véletlen számmal.}
      a[i]:=trunc(90*random+1);
      i:=i+1;
      end;

   writeln('A szamok rendezes elott:');
   i:=1;
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme: ',a[i]);
```

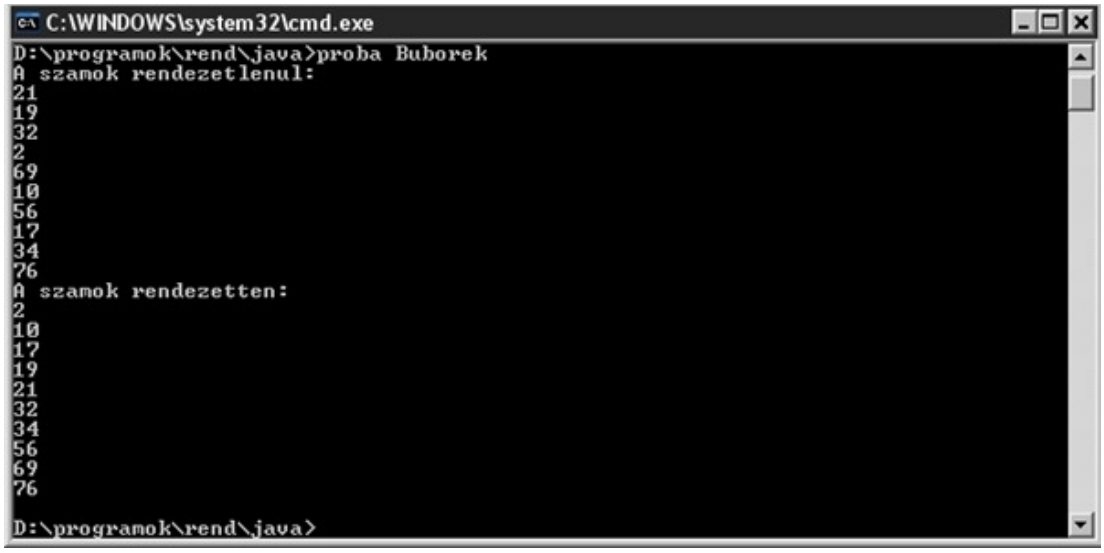
```
      i:=i+1;
      end;

   i:=1;
   while (i<n) do begin
```
{A tömb aktuális i-edik eleméhez hasonlítjuk a tömb összes többi elemét. Ha a sorrend nem megfeleleő, akkor cserélünk. Amennyiben elérünk a tömb végére, akkor a következő indexű elemet hasonlítjuk az utána lévőkhöz. Ezt addig folytatjuk, míg végig nem érünk a tömbön.}
```
      j:=i+1;
      while (j<=n) do begin
         if(a[j]<a[i]) then begin
```
{Ha azt észleljük, hogy a vizsgált elemnél kisebb elemet találtunk, akkor a kettő értékét megcseréljük.}
```
            x:=a[j];
            a[j]:=a[i];
            a[i]:=x;
            end;
         j:=j+1;
         end;
      i:=i+1;
      end;

   writeln('');
   writeln('A szamok rendezes utan:');
   i:=1;
   while (i<=n) do begin
      writeln(a[i]);
      i:=i+1;
      end;
   readkey;
END.
```

Tömb feltöltése véletlen számokkal és az elemek elrendezése buborék rendezéssel (Buborek.java)

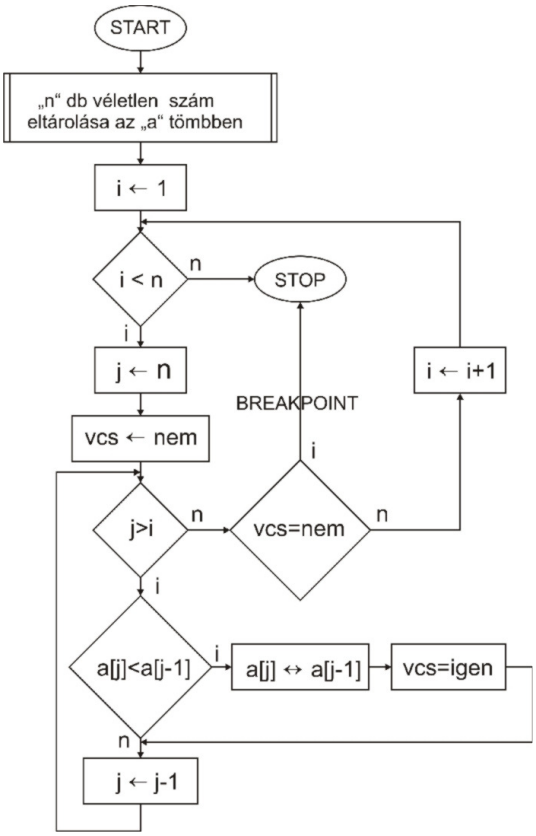



```
public class Buborek{
   public static void main(String[]args){
      int n,i,j,x;
      boolean vcs;
```
/ boolean típusú, azaz true vagy false értéket felvevő vcs változó lértehozása. Ez a "van csere"-t fogja jelenteni. */*
```
      int[] a=new int[50];
      n=10;
      i=0;
      while (i<n){
         a[i]=(int)((90*Math.random())+1);
         i=i+1;
         }
      System.out.println("A szamok rendezetlenul:");
      i=0;
      while(i<n){
```

```
            System.out.println(a[i]);
            i=i+1;
            }
        i=0;
        while (i<n){
/* Tömb rendezése buborék rendezéssel, úgy, hogy a tömb legnagyobb
indexű elemétől lefelé haladva megvizsgáljuk meg azt, hogy az aktuális
elemnél kisebb e az őt megelőző. Ha igen, cserélünk. */
            j=n-1;
/*A legnagyobb indexű elemet kezdjük vizsgálni. */
            vcs=false;
/*Még nincs csere. */
            while (j>i){
/*Addig vizsgáljuk az elemet, amíg az indexe nagyobb az i-nél, azaz a
megvizsgált elemek számánál. */
                if(a[j]<a[j-1]){
/*Ha az aktuálisan vizsgált elemnél kisebb az előtte lévő, akkor megcseréljük
őket, valamint beállítjuk, hogy történt csere, azaz vcs=true; */
                    x=a[j-1];
                    a[j-1]=a[j];
                    a[j]=x;
                    vcs=true;
                    }
                j=j-1;
                }
            if (vcs==false) break;
/* Ha nem történt csere, azaz a vizsgált elemnél nem találtunk kisebbet,
akkor nem kell tovább vizsgálódni. */
            i=i+1;
            }
        System.out.println("A szamok rendezetten:");
        i=0;
        while(i<n){
            System.out.println(a[i]);
            i=i+1;
            }
        }
    }
```



Tömb feltöltése véletlen számokkal és az elemek elrendezése buborék rendezéssel (Buborek.pas)

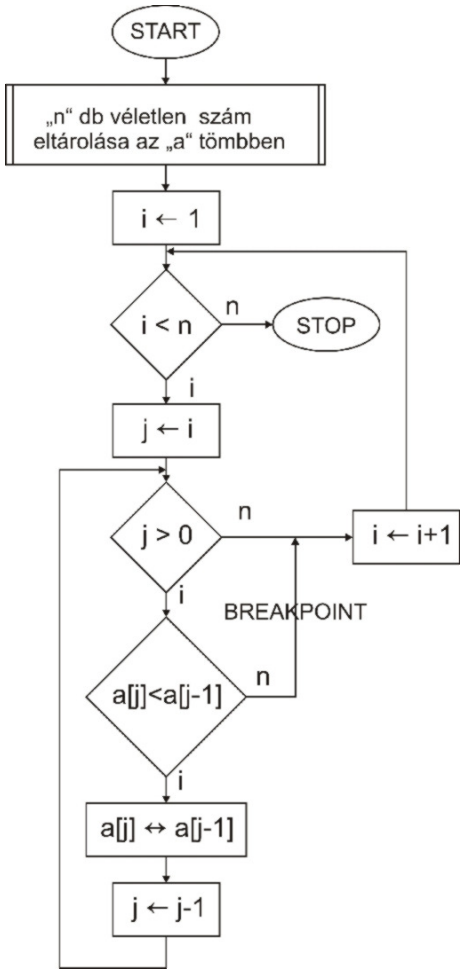



```
program Buborek;
uses crt;
var i,n,j,x:integer;
    vcs:boolean;
```

```
{boolean típusú, azaz true vagy false értéket felvevő vcs változó lértehozása.
Ez a "van csere"-t fogja jelenteni.}
    a:array [1..100] of integer;
BEGIN
   clrscr;
   i:=1;
   write('Hany szam legyen a tombben? ');
   readln(n);
   while(i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      end;
   writeln('A szamok rendezes elott:');
   i:=1;
   while(i<=n) do begin
      writeln('A tomb ',i,'. eleme: ',a[i]);
      i:=i+1;
      end;

   i:=1;
   while (i<n) do begin;
{Tömb rendezése buborék rendezéssel, úgy, hogy a tömb legnagyobb
indexű elemétől lefelé haladva megvizsgáljuk meg azt, hogy az aktuális
elemnél kisebb e az őt megelőző. Ha igen, cserélünk. }
      j:=n;
{A legnagyobb indexű elemet kezdjük vizsgálni. }
      vcs:=false;
{Még nincs csere.}
      while (j>i) do begin
{Addig vizsgáljuk az elemet, amíg az indexe nagyobb az i-nél, azaz a
megvizsgált elemek számánál. }
         if(a[j]<a[j-1]) then begin
{Ha az aktuálisan vizsgált elemnél kisebb az előtte lévő, akkor megcseréljük
őket, valamint beállítjuk, hogy történt csere, azaz vcs:=true; }
            x:=a[j];
            a[j]:=a[j-1];
            a[j-1]:=x;
            vcs:=true
            end;
         j:=j-1;
         end;
      if (vcs=false) then break;
{Ha nem történt csere, azaz a vizsgált elemnél nem találtunk kisebbet, akkor
nem kell tovább vizsgálódni. }
      i:=i+1;
      end;
   writeln('');
   writeln('A szamok rendezes utan:');
   i:=1;
```

```
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme: ',a[i]);
      i:=i+1;
      end;
   readkey;
END.
```

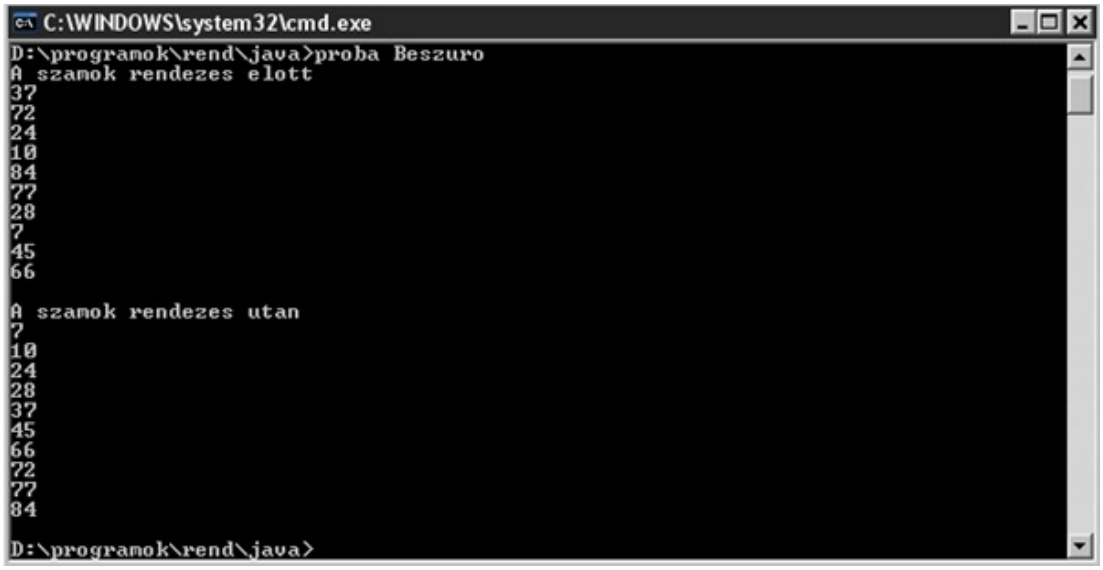
Tömb feltöltése véletlen számokkal és az elemek elrendezése beszúró rendezéssel, eljárások használatával (Beszuro.java)



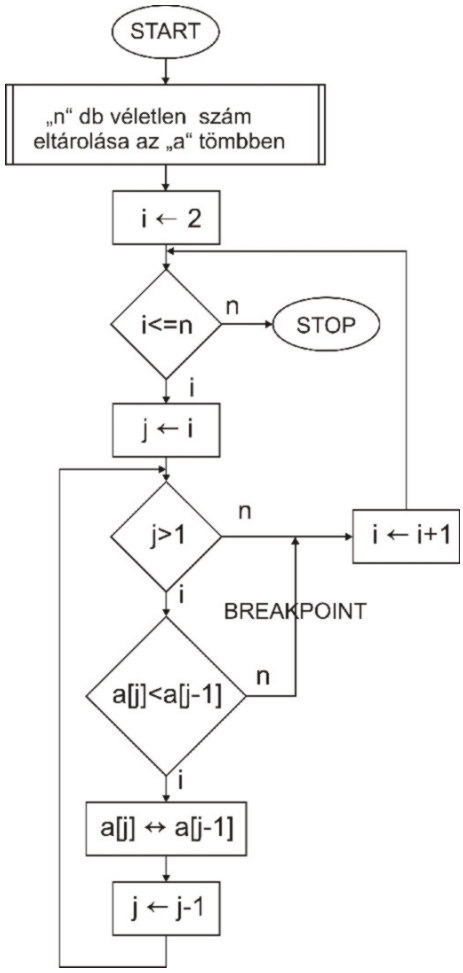

```
public class Beszuro{
   public static int n=10;
   public static int[] a=new int[10];

   public static void feltolt(){
      int i;
      i=0;
      while(i<n){
         a[i]=(int)(90*Math.random())+1;
         i=i+1;
         }
      }

   public static void kiir(){
      int i;
      i=0;
      while (i<n){
         System.out.println(a[i]);
```

```
        i=i+1;
        }
    }

  public static void cserel(int i,int j){
    int x;
    x=a[j];
    a[j]=a[i];
    a[i]=x;
    }

  public static void rendez(){
/* Tömb elemeinek rendezése beszúró rendezéssel, úgy, hogy elölről kezdve
végigmegyünk a tömb elemein és megvizsgáljuk, hogy az aktuális elem
kisebb e, mint a sorban azt megelező elem. Ha igen, akkor az aktuális
elemet megcseréljük a másikkal és újra megvizsgáljuk, hogy az őt megelező
elem kisebb e nála. Ez a cserélgetés addig megy, amíg az aktuálisan vizsgált
elem meg nem találja a helyét, azaz nincs nála kisebb a listában. */
    int i,j;
    i=1;
/* A második elemnél kezdjük a vizsgálatot, hiszen annak már van megelőző
eleme. */
    while (i<n){
/*Minden tömbelemet megvizsgálunk. */
        j=i;
/*Beállítjuk a j változóba az aktuálisan vizsgált elem indexét. */
        while (j>0){
/* Elértük e már az első tömbelemet a vizsgálódás idején. Ha nem, folytatódik
a vizsgálat. */
            if(a[j]<a[j-1]){
/* Megvizsgáljuk, hogy az aktuális tömbelem kisebb e az őt megelőzőnél, ha
igen, cserélünk. Ha ez a feltétel nem teljesül, akkor az aktuális tömbelem
előtt csak kisebb elemek vannak,így nincs szükség további vizsgálatra,  ezért
megszakítjuk a ciklust. */
               cserel(j,j-1);
               }
            else break;
            j=j-1;
            }
        i=i+1;
        }
    }

  public static void main(String[] args){
    feltolt();
    System.out.println("A szamok rendezes elott");
    kiir();
    System.out.println(" ");
    rendez();
```

```
        System.out.println("A szamok rendezes utan");
        kiir();
        }
    }
```



Tömb feltöltése véletlen számokkal és az elemek elrendezése beszúró rendezéssel, eljárások használatával (Beszuro.pas)

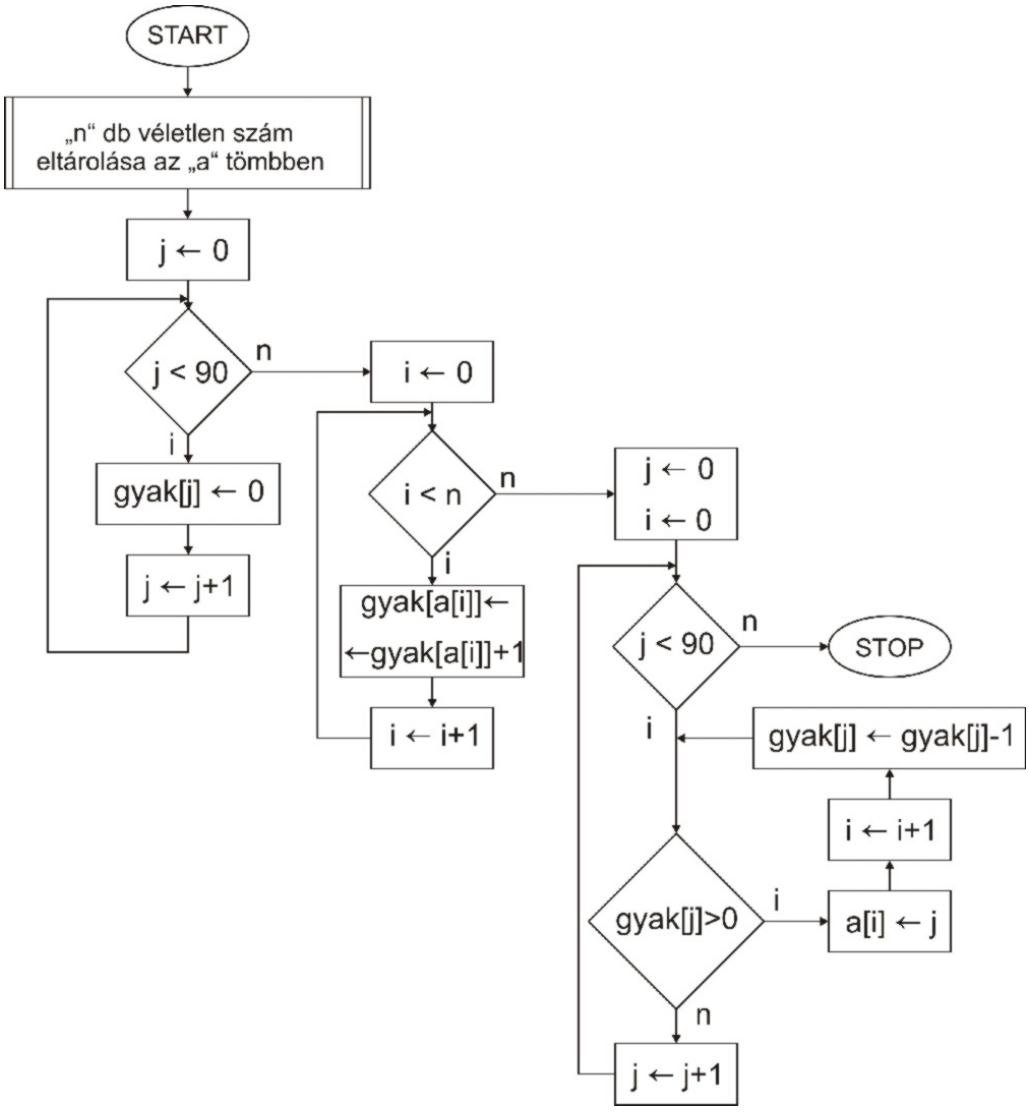



```
program Beszuro;
uses crt;
var i,n:integer;
    a:array [1..50] of integer;

procedure kiir;
begin
   for i:=1 to n do begin
      writeln(a[i]);
      end;
   end;

procedure cserel(mit,mire:integer);
var x:integer;
begin
   x:=a[mit];
   a[mit]:=a[mire];
```

```
   a[mire]:=x;
   end;

procedure beszuras(k:integer);
{Tömb elemeinek rendezése beszúró rendezéssel, úgy, hogy megvizsgáljuk,
hogy az eljárásban az aktuális bemenő elem kisebb e, mint a tömbben az azt
megelező elem. Ha igen, akkor az aktuális elemet megcseréljük a másikkal
és újra megvizsgáljuk, hogy az őt megelező elem kisebb e nála. Ez a
cserélgetés addig megy, amíg az aktuálisan vizsgált elem meg nem találja a
helyét, azaz nincs nála kisebb a listában. }
begin
   while(k>1) do begin
{Elértük e már az első tömbelemet a vizsgálódás idején. Ha nem, folytatódik
a vizsgálat. }
      if (a[k]<a[k-1]) then
{Megvizsgáljuk, hogy az aktuális tömbelem kisebb e az őt megelőzőnél, ha
igen, cserélünk. Ha ez a feltétel nem teljesül, akkor az aktuális tömbelem
előtt csak kisebb elemek vannak,így nincs szükség további vizsgálatra, ezért
megszakítjuk a ciklust. }
         cserel (k,k-1)
      else break;
      dec(k);
{Csökkentjük a k értékét 1-el (helyette írhatnák, hogy k:=k-1;) }
      end;
   end;

BEGIN
   clrscr;
   randomize;
   n:=10;
   writeln('A tomb elemei rendezes elott:');
   for i:=1 to n do begin
      a[i]:=trunc(90*random)+1;
      writeln(a[i]);
      end;
   writeln('');
   writeln('A tomb elemei rendezes utan:');
   i:=2;
   while (i<=n) do begin
      beszuras(i);
      i:=i+1;
      end;
   kiir;
  readkey;
END.
```


Turbo Pascal 7.0
A tomb elemei rendezes elott:
44
64
55
9
3
16
6
28
42
1
A tomb elemei rendezes utan:
1
3
6
9
16
28
42
44
55
64

Tömb feltöltése véletlen számokkal és az elemek elrendezése terítő rendezéssel, eljárások használatával (Terito.java)

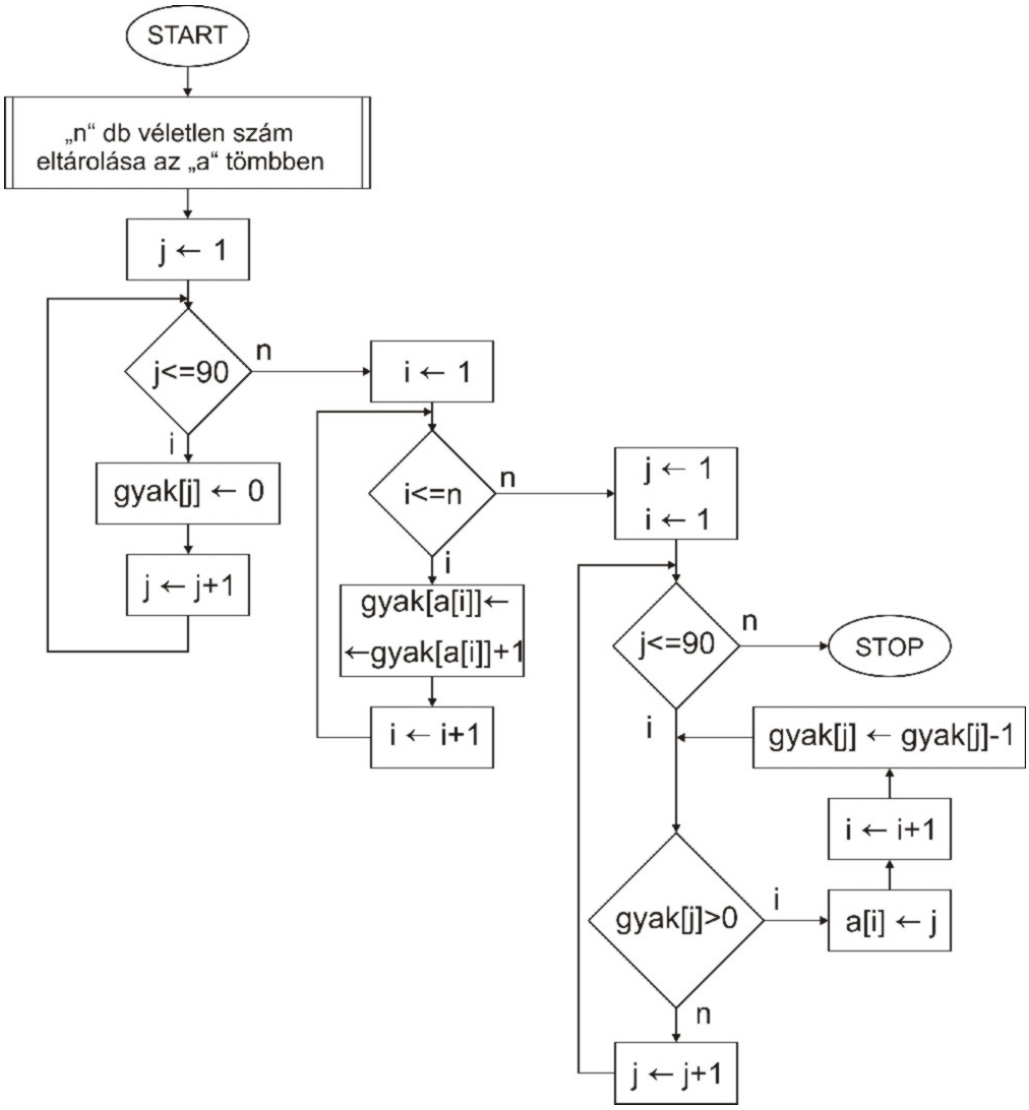



```
public class Terito{
   public static int n=10;
   public static int[] a=new int[n];
   public static int[] gyak=new int[90];

   public static void feltolt(){
      int i=0;
      while(i<n){
         a[i]=(int)(90*Math.random())+1;
         i=i+1;
         }
      }
```

```
   public static void kiir(){
      int i=0;
      while(i<n){
         System.out.println(a[i]);
         i=i+1;
         }
      }

   public static void terito(){
      int i,j;
      j=0;
      while(j<90){
/* Tömb rendezése terítő eljárással, úgy, hogy létrehozunk egy tömböt, mely
elemeit kinullázzuk. Figyeljük meg, hogy a gyak tömb elemeinek száma
megegyezik az a tömbben előforduló elemek maximális értékével, azaz 90-
el. A gyak tömbben az elemek gyakoriságát tároljuk el. */
         gyak[j]=0;
         j=j+1;
         }
      i=0;
      while(i<n){
/* A ciklus segítségével feljegyezzük a gyak tömbbe az a  tömbben előforduló
értékeket, úgy, hogy a gyak tömb indexét beállítjuk az a tömb elemének
értékére és a gyak tömb aktuális elemének értékét megnöveljük. */
         gyak[a[i]]=gyak[a[i]]+1;
         i=i+1;
         }
      i=0;
      j=0;
      while(j<90){
/* Visszaadjuk az a tömbbe az értékeket. */
         if(gyak[j]>0){
/* Végigmegyünk a gyak tömbön és megvizsgáljuk, hogy az aktuális indexű
elem értéke mennyi, mivel ez az előfordulás száma. Amekkora értékű,
annyiszor helyezzük el az indexét az a tömbben. A ciklusban ezt az értéket
csökkentjük, az a tömb indexét növeljük, így egymás után kerülnek az
elemek tárolásra. */
            a[i]=j;
            i=i+1;
            gyak[j]=gyak[j]-1;
            }
         j=j+1;
         }
      }

   public static void main (String[] args){
      feltolt();
      System.out.println("A szamok rendezes elott:");
      kiir();
```

```
        System.out.println(" ");
        terito();
        System.out.println("A szamok rendezes utan:");
        kiir();
        }
    }
```

Tömb feltöltése véletlen számokkal és az elemek elrendezése terítő rendezéssel, eljárások használatával (Terito.pas)

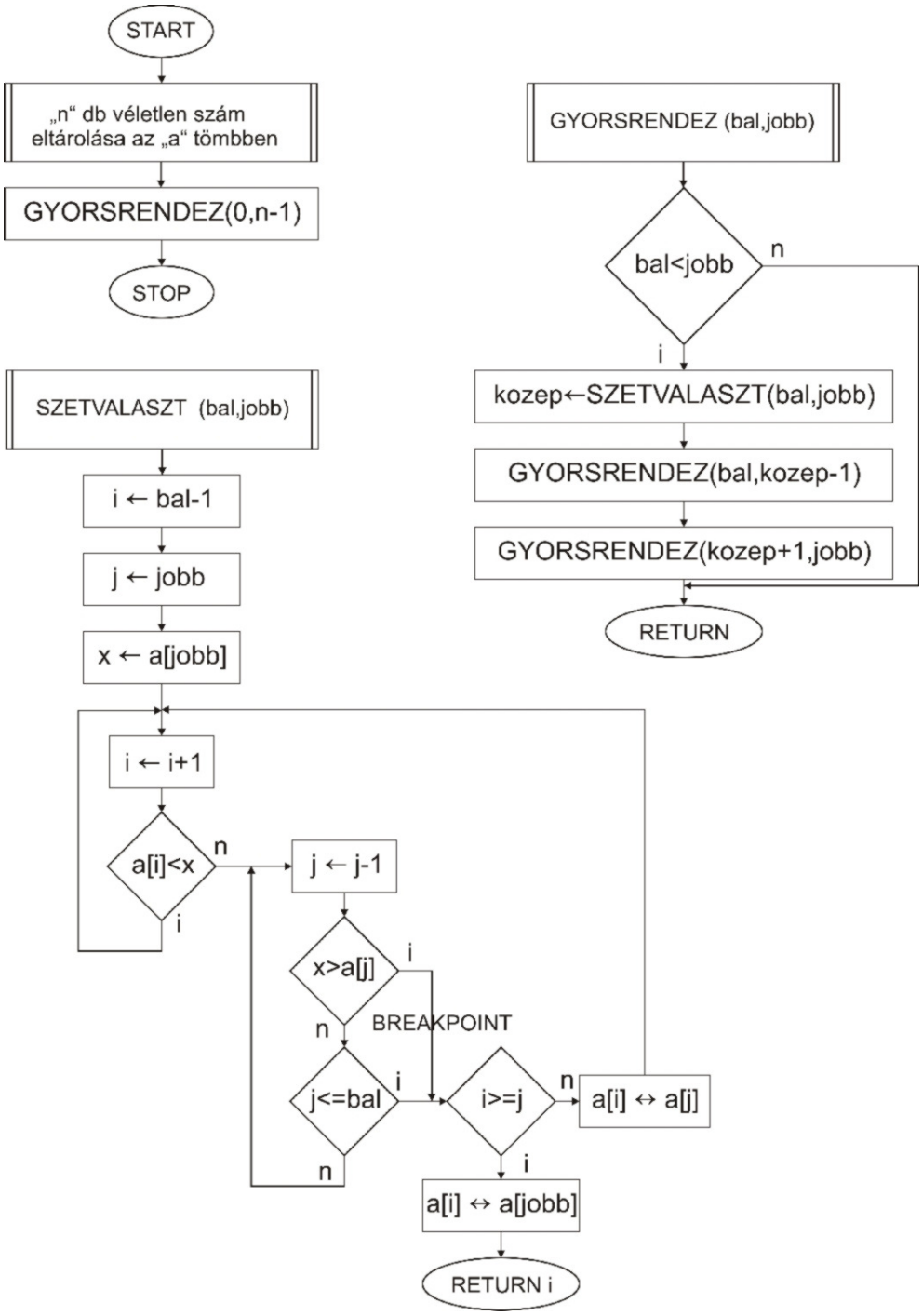



```
program Terito;
uses crt;
var i,j,n,m:integer;
    a:array [1..50] of integer;
    gyak:array [1..90] of integer;

procedure feltolt;
var i:integer;
begin
   for i:=1 to n do begin
      a[i]:=trunc(90*random)+1;
      end;
   end;
```

```
procedure kiir;
var i:integer;
begin
   for i:=1 to n do begin
      writeln(a[i]);
      end;
   end;

BEGIN
   clrscr;
   randomize;
   n:=10;
   feltolt;
   writeln('A szamok rendezes elott:');
   kiir;
   for j:=1 to 90 do begin
{Tömb rendezése terítő eljárással, úgy, hogy létrehozunk egy tömböt, mely
elemeit kinullázzuk. Figyeljük meg, hogy a gyak tömb elemeinek száma
megegyezik az a tömbben előforduló elemek maximális értékével, azaz 90-
el. A gyak tömbben az elemek gyakoriságát tároljuk el. }
      gyak[j]:=0;
      end;
   for i:=1 to n do begin
{A ciklus segítségével feljegyezzük a gyak tömbbe az a tömbben előforduló
értékeket, úgy, hogy a gyak tömb indexét beállítjuk az a tömb elemének
értékére és a gyak tömb aktuális elemének értékét megnöveljük. }
      inc(gyak[a[i]]);
      end;
   i:=1;
   for j:=1 to 90 do begin
{Visszaadjuk az a tömbbe az értékeket. }
      while (gyak[j]>0) do begin
{Végigmegyünk a gyak tömbön és megvizsgáljuk, hogy az aktuális indexű
elem értéke mennyi, mivel ez az előfordulás száma. Amekkora értékű,
annyiszor helyezzük el az indexét az a tömbben. A ciklusban ezt az értéket
csökkentjük, az a tömb indexét növeljük, így egymás után kerülnek az
elemek tárolásra. }
         a[i]:=j;
         inc(i);
         dec(gyak[j]);
         end;
      end;
   writeln('');
   writeln('A szamok rendezes utan:');
   kiir;
   readkey;
END.
```


Turbo Pascal 7.0
A szamok rendezes elott:
16
61
19
59
64
36
75
29
17
37
A szamok rendezes utan:
16
17
19
29
36
37
59
61
64
75

Tömb feltöltése véletlen számokkal és az elemek elrendezése gyorsrendezéssel, eljárások használatával (Gyorsrendezes.java)

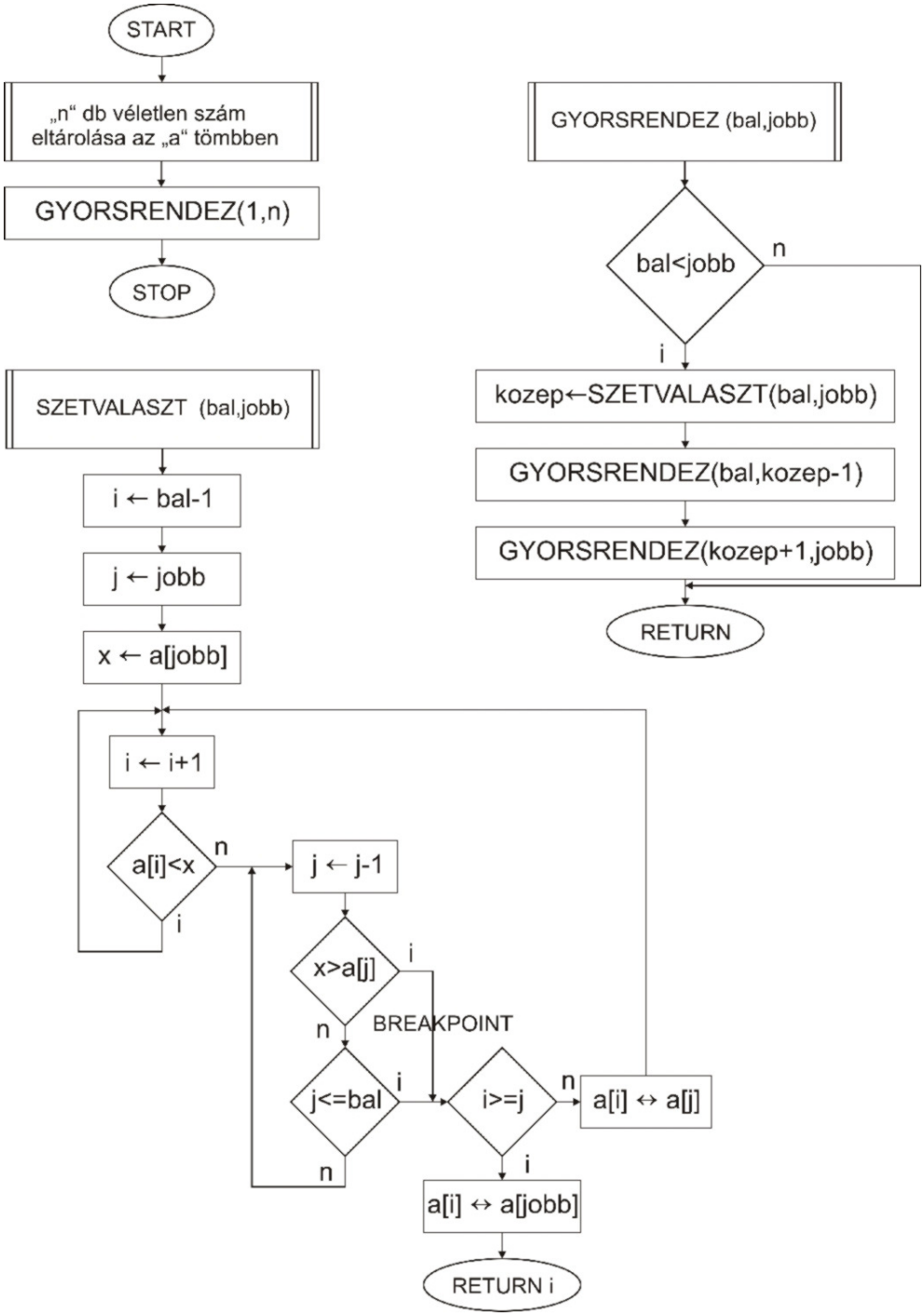



```
public class Gyorsrendezes{
   public static int n=10;
   public static int[] a=new int[n];
```
/ Létrehozunk egy n elemű, a nevű, egész típusú tömböt. */*

```
   public static void feltolt(){
/* Véletlen számokkal feltöltjük az a tömböt. */
      int i=0;
      while(i<n){
         a[i]=(int)(90*Math.random())+1;
         i=i+1;
         }
      }

   public static void kiir(){
/* A tömb elemeinek kiírása eljárással. */
      int i=0;
      while(i<n){
         System.out.println(a[i]);
         i=i+1;
         }
      }

   public static void cserel(int i,int j){
/* Cserélő eljárás, két egész típusú bemenő paraméterrel. */
      int x;
      x=a[j];
      a[j]=a[i];
      a[i]=x;
      }

   public static int szetvalaszt(int bal,int jobb){
/* Visszaad egy olyan pozíciót, melytől balra csak kisebb, jobbra csak
nagyobb számok helyezkednek el. A függvény során addig cserélgetjük a
tömbelemeket amíg a középponti elemhez képest ez a helyzet ki nem alakul.
Végül egy egész típusú i változóban visszaadjuk a középponti elem
helyzetét. */
      int i,j,x;
      i=bal-1;
      j=jobb;
      x=a[jobb];
      do{
         do{
            i=i+1;
            }while(x>a[i]);
/* Balról keresi az x-nél nagyobb elemet és eltárolja az indexét i-ben. */
         do{
            j=j-1;
            if(a[j]<x) break;
            }while(j>bal);
/* Keresi jobbról a kisebb elemet majd eltárolja az indexét j-ben. */
         if(i>=j) break;
         else cserel(i,j);
```

```
/* Amíg a két index nem ugyanaz, vagy nincs átfedés, addig cserél, ha
egyenlőek vagy van átfedés, akkor pedig break-kel kilépünk a ciklusból. */
            }while(true);
        cserel(i,jobb);
/* Mivel van átfedés, ezért megvan a középponti elem, ekkor ezt helyezzük
középre. */
        return i;
/* Visszaadja a középponti elem pozícióját. */
        }

    public static void gyorsrendez(int bal,int jobb){
        int kozep;
        if(bal<jobb){
            kozep=szetvalaszt(bal,jobb);
/* kozep-be beletesszük a középponti elem indexét, majd rekurzívan
meghívjuk a gyorsrendez eljárást a középponti elemtől balra és jobbra
helyezkedő elemekre is. */
            gyorsrendez(bal,kozep-1);
            gyorsrendez(kozep+1,jobb);
            }
        }

    public static void main(String[] args){
        feltolt();
        System.out.println("A szamok rendezes elott:");
        kiir();
        System.out.println(" ");
        gyorsrendez(0,n-1);
        System.out.println("A szamok rendezes utan:");
        kiir();
        }
    }
```

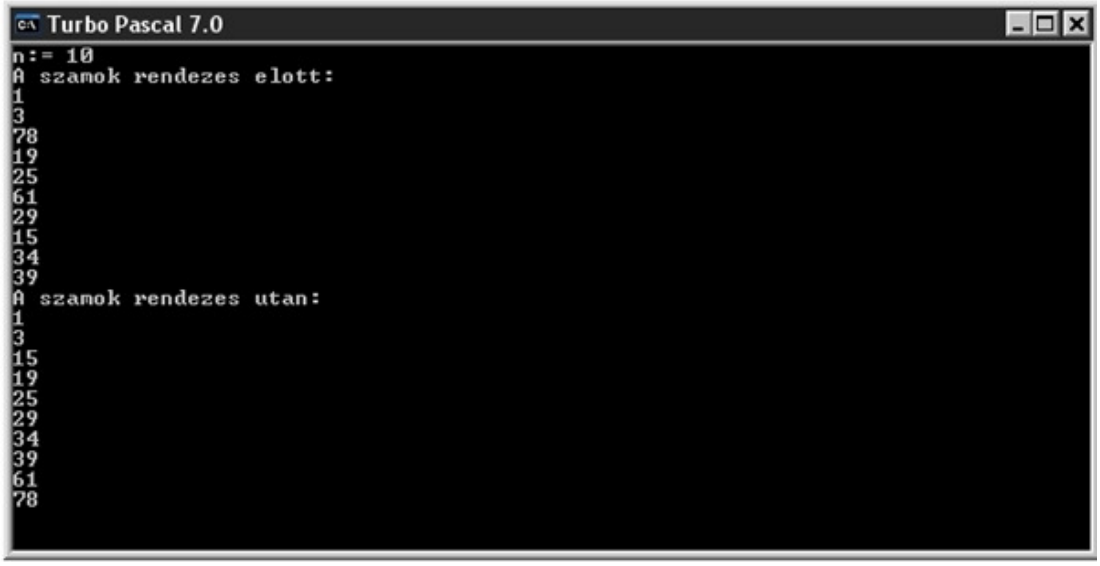
Tömb feltöltése véletlen számokkal és az elemek elrendezése gyorsrendezéssel, eljárások használatával (Gyorsrendezes.pas)



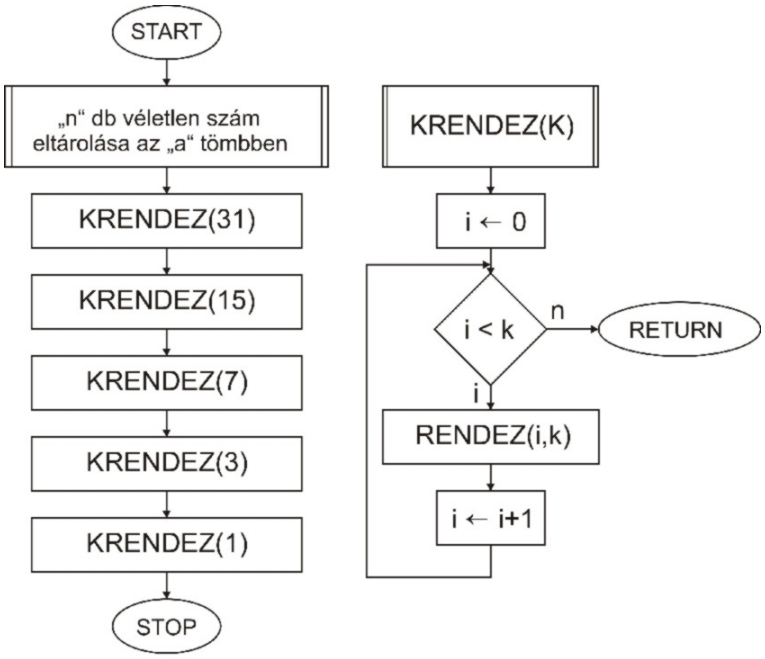

```
program Gyorsrendezes;
uses crt;
```

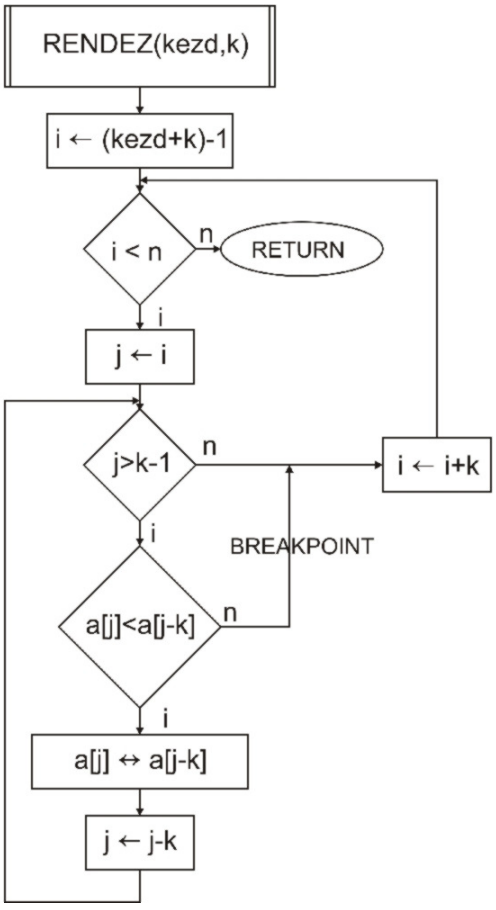
```
var n:integer;
    a:array [1..300] of integer;
{ Létrehozunk egy 300 elemű, a nevű, egész típusú tömböt. }

procedure cserel(mit,mire:integer);
{Cserélő eljárás, két egész típusú bemenő paraméterrel. }
var x:integer;
begin
   x:=a[mit];
   a[mit]:=a[mire];
   a[mire]:=x;
   end;

procedure feltolt;
{Véletlen számokkal feltöltjük az a tömböt. }
var i:integer;
begin
   for i:=1 to n do begin
      a[i]:=trunc(90*random)+1;
      end;
   end;

procedure kiir;
{A tömb elemeinek kiírása eljárással. }
var i:integer;
begin
   for i:=1 to n do begin
      writeln(a[i]);
      end;
   end;

function szetvalaszt(bal,jobb:integer):integer;
{ Visszaad egy olyan pozíciót, melytől balra csak kisebb, jobbra csak
nagyobb számok helyezkednek el. A függvény során addig cserélgetjük a
tömbelemeket amíg a középponti elemhez képest ez a helyzet ki nem alakul.
Végül egy egész típusú i változóban visszaadjuk a középponti elem
helyzetét. }
var i,j,x:integer;
begin
   i:=bal-1;
   j:=jobb;
   x:=a[jobb];
   repeat
      repeat inc(i);
{Balról keresi az x-nél nagyobb elemet és eltárolja az indexét i-ben. }
         until (a[i]>=x);
      repeat dec(j);
         if(a[j]<x) then break;
         until (j<=bal);
```

```
{Keresi jobbról a kisebb elemet majd eltárolja az indexét j-ben. }
      if (i>=j) then break
      else cserel(i,j);
{Amíg a két index nem ugyanaz, vagy nincs átfedés, addig cserél, ha
egyenlőek vagy van átfedés, akkor pedig break-kel kilépünk a ciklusból. }
      until(false);
   cserel(i,jobb);
{Mivel van átfedés, ezért megvan a középponti elem, ekkor ezt helyezzük
középre. }
   szetvalaszt:=i;
{Visszaadja a középponti elem pozícióját. }
   end;

procedure gyorsrendez(bal,jobb:integer);
var kozep:integer;
begin
   if(bal<jobb) then begin
      kozep:=szetvalaszt(bal,jobb);
{kozep-be beletesszük a középponti elem indexét, majd rekurzívan
meghívjuk a gyorsrendez eljárást a középponti elemtől balra és jobbra
helyezkedő elemekre is. }
      gyorsrendez(bal,kozep-1);
      gyorsrendez(kozep+1,jobb);
      end;
   end;

BEGIN
   clrscr;
   write('n:= ');
   readln(n);
   feltolt;
   writeln('A szamok rendezes elott:');
   kiir;
   writeln('A szamok rendezes utan:');
   gyorsrendez(1,n);
   kiir;
   readkey;
END.
```


Turbo Pascal 7.0
n:= 10
A szamok rendezes elott:
1
3
78
19
25
61
29
15
34
39
A szamok rendezes utan:
1
3
15
19
25
29
34
39
61
78

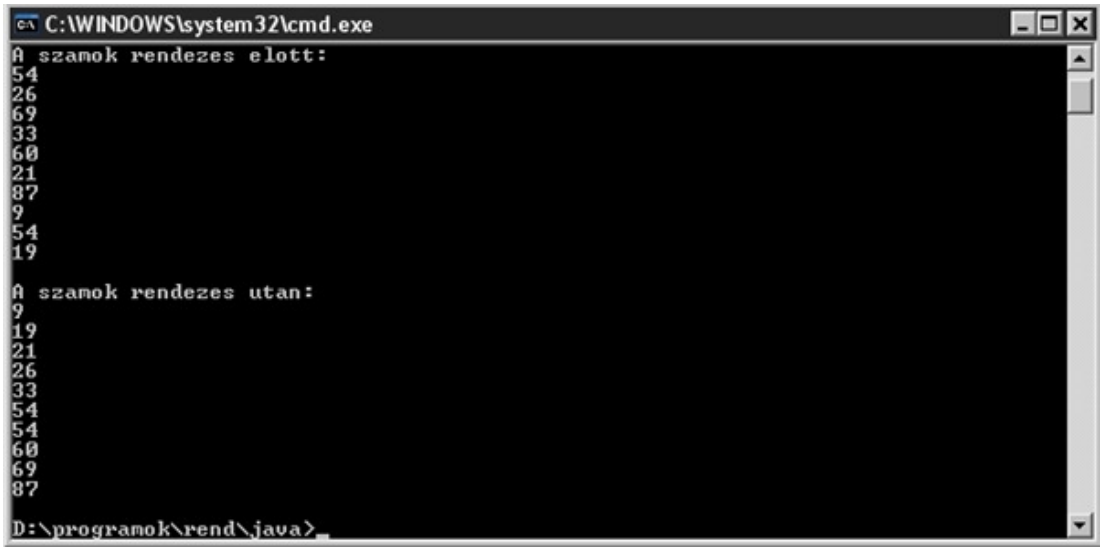
Tömb feltöltése véletlen számokkal és az elemek elrendezése shell-rendezéssel, eljárások használatával (Shellrendezes.java)



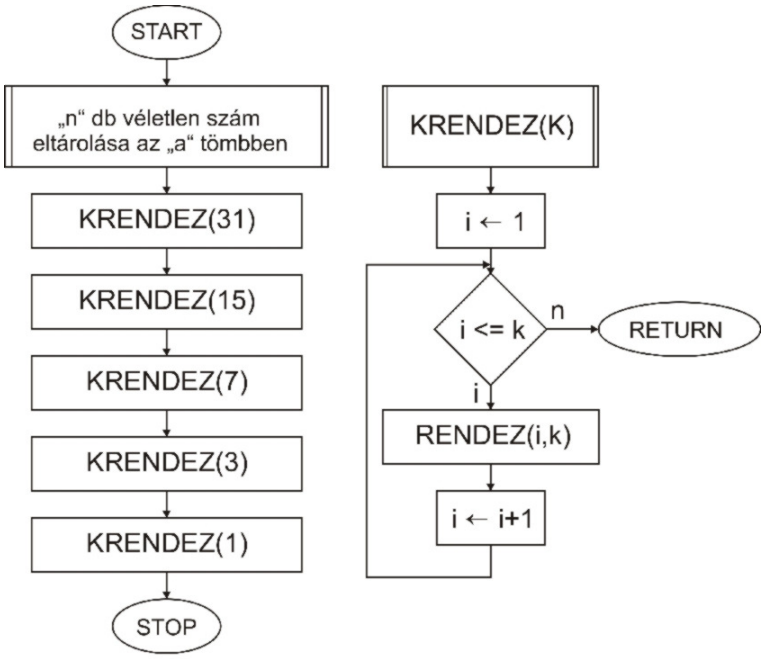


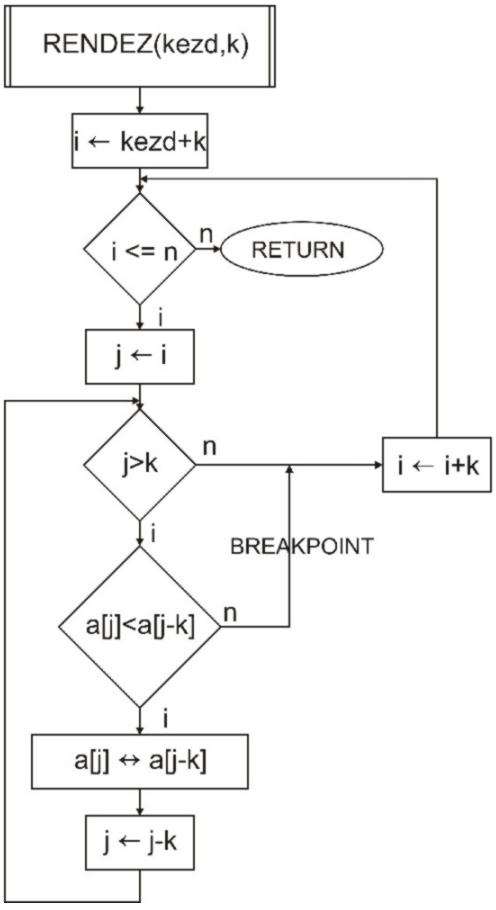
```
public class Shellrendezes{
   public static int n=10;
   public static int[] a=new int[n];

   public static void feltolt(){
      int i=0;
      while(i<n){
         a[i]=(int)(90*Math.random())+1;
         i=i+1;
         }
      }

   public static void kiir(){
      int i=0;
      while(i<n){
         System.out.println(a[i]);
         i=i+1;
         }
      }

   public static void cserel(int i,int j){
      int x;
      x=a[j];
      a[j]=a[i];
      a[i]=x;
      }

   public static void rendez(int kezd,int k){
/* A rendező eljárás, mellyel minden k-adik elemet összehasonlítjuk, szükség
esetén cseréljük. */
      int i,j;
      i=(kezd+k)-1;
      while(i<n){
/* Hogy ha a átlépjük a tömb elemeinek számát, akkor a ciklus megáll. */
         j=i;
         while(j>k-1){
            if(a[j]<a[j-k]){
               cserel(j,j-k);
               }
            else break;
            j=j-k;
            }
         i=i+k;
         }
      }

   public static void krendez(int k){
```

```
/* Előrendezés eljárása. A benne lévő ciklus segítségével fogjuk meghívni a
rendezést minden elemre. */
        int i;
        i=0;
        while(i<k){
/* Végigmegyünk minden tömbelemen. */
            rendez(i,k);
/* Meghívjuk az aktuális indexű elemre az aktuális lépésű (31,15...)
előrendezést. */
            i=i+1;
            }
        }

    public static void shell(){
        krendez(31);
/* Előrendezéseket hajtunk végre, így gyorsítva a rendezési folyamatot. Az,
hogy először a 31., majd a 15., 7., 3. és 1. elemeket rendezzük,
kísérletezések során alakult ki. Ez a rendezés nagy elemszámú tömbök,
vagy sorozatokat esetén ideális. Természetesen minden szám elrendezésre
kerül. */
        krendez(15);
        krendez(7);
        krendez(3);
        krendez(1);
        }

    public static void main(String[] args){
        feltolt();
        System.out.println("A szamok rendezes elott:");
        kiir();
        System.out.println(" ");
        shell();
        System.out.println("A szamok rendezes utan:");
        kiir();
        }
    }
```


C:\WINDOWS\system32\cmd.exe
A szamok rendezes elott:
54
26
69
33
60
21
87
9
54
19
A szamok rendezes utan:
9
19
21
26
33
54
54
60
69
87
D:\programok\rend\java>

Tömb feltöltése véletlen számokkal és az elemek elrendezése shell-rendezéssel, eljárások használatával (Shellrendezes.pas)



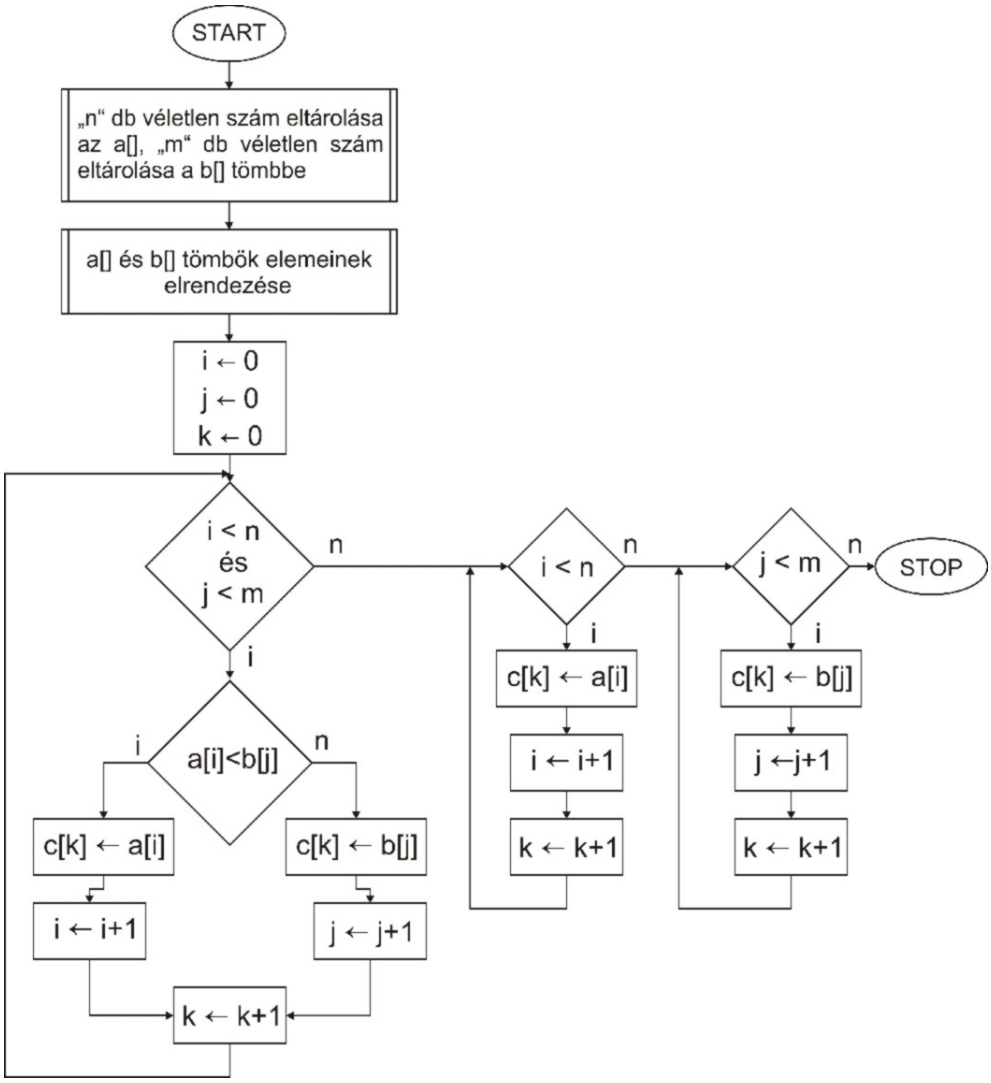


```
program Shellrendezes;
uses crt;
var n:integer;
    a:array [1..300] of integer;

procedure cserel(mit,mire:integer);
var x:integer;
begin
   x:=a[mit];
   a[mit]:=a[mire];
   a[mire]:=x;
   end;

procedure feltolt;
var i:integer;
begin
   for i:=1 to n do begin
      a[i]:=trunc(90*random)+1;
      end;
   end;

procedure kiir;
var i:integer;
begin
   for i:=1 to n do begin
      writeln(a[i]);
      end;
   end;

procedure rendez(kezd,k:integer);
{A rendező eljárás, mellyel minden k-adik elemet összehasonlítjuk, szükség
esetén cseréljük. }
var i,j:integer;
begin
   i:=kezd+k;
   while(i<=n) do begin
{Hogy ha a átlépjük a tömb elemeinek számát, akkor a ciklus megáll. }
      j:=i;
      while (j>k) do begin
         if (a[j]<a[j-k]) then begin
            cserel(j,j-k);
            j:=j-k;
            end
         else break;
         end;
      i:=i+k;
      end;
   end;
```

```
procedure krendez(k:integer);
{Előrendezés eljárása. A benne lévő ciklus segítségével fogjuk meghívni a
rendezést minden elemre. }
var i:integer;
begin
   i:=1;
   while (i<=k) do begin
{Végigmegyünk minden tömbelemen. }
      rendez(i,k);
{Meghívjuk az aktuális indexű elemre az aktuális lépésű (31,15...)
előrendezést. }
      i:=i+1;
      end;
   end;

procedure shell;
begin
   krendez(31);
{Előrendezéseket hajtunk végre, így gyorsítva a rendezési folyamatot. Az,
hogy először a 31., majd a 15., 7., 3. és 1. elemeket rendezzük,
kisérletezések során alakult ki. Ez a rendezés nagy elemszámú tömbök,
vagy sorozatokat esetén ideális. Természetesen minden szám elrendezésre
kerül. }
   krendez(15);
   krendez(7);
   krendez(3);
   krendez(1);
   end;

BEGIN
   clrscr;
   write('n:= ');
   readln(n);
   feltolt;
   writeln('A szamok rendezes elott:');
   kiir;
   writeln('');
   writeln('A szamok rendezes utan:');
   shell;
   kiir;
   readkey;
END.
```


Turbo Pascal 7.0
n:= 10
A szamok rendezes elott:
1
3
78
19
25
61
29
15
34
39
A szamok rendezes utan:
1
3
15
19
25
29
34
39
61
78

Két tömb feltöltése véletlen számokkal, ezek elrendezése, majd összefésülése és eltárolása egy újabb tömbbe (Fesules.java)

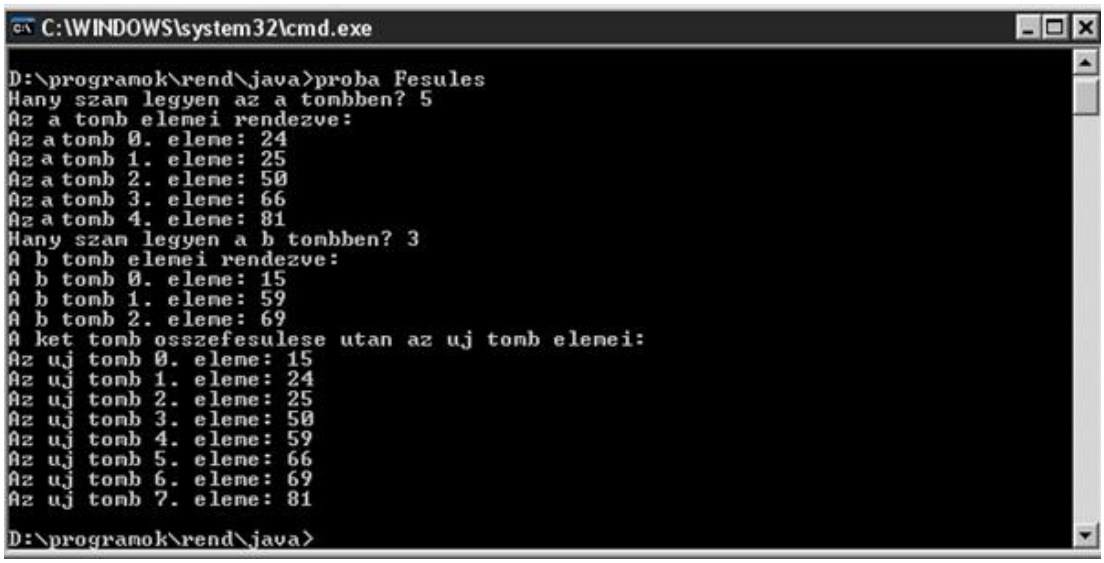



```
import java.io.*;
public class Fesules {
   public static int in() throws Exception {
/* Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza. */
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      String s=x.readLine();
      int i=Integer.parseInt(s);
      return i;
      }
   public static void main(String[] args) throws
Exception{

      int i,j,n,k,x,h,g,m;
      int[] a=new int[100];
```

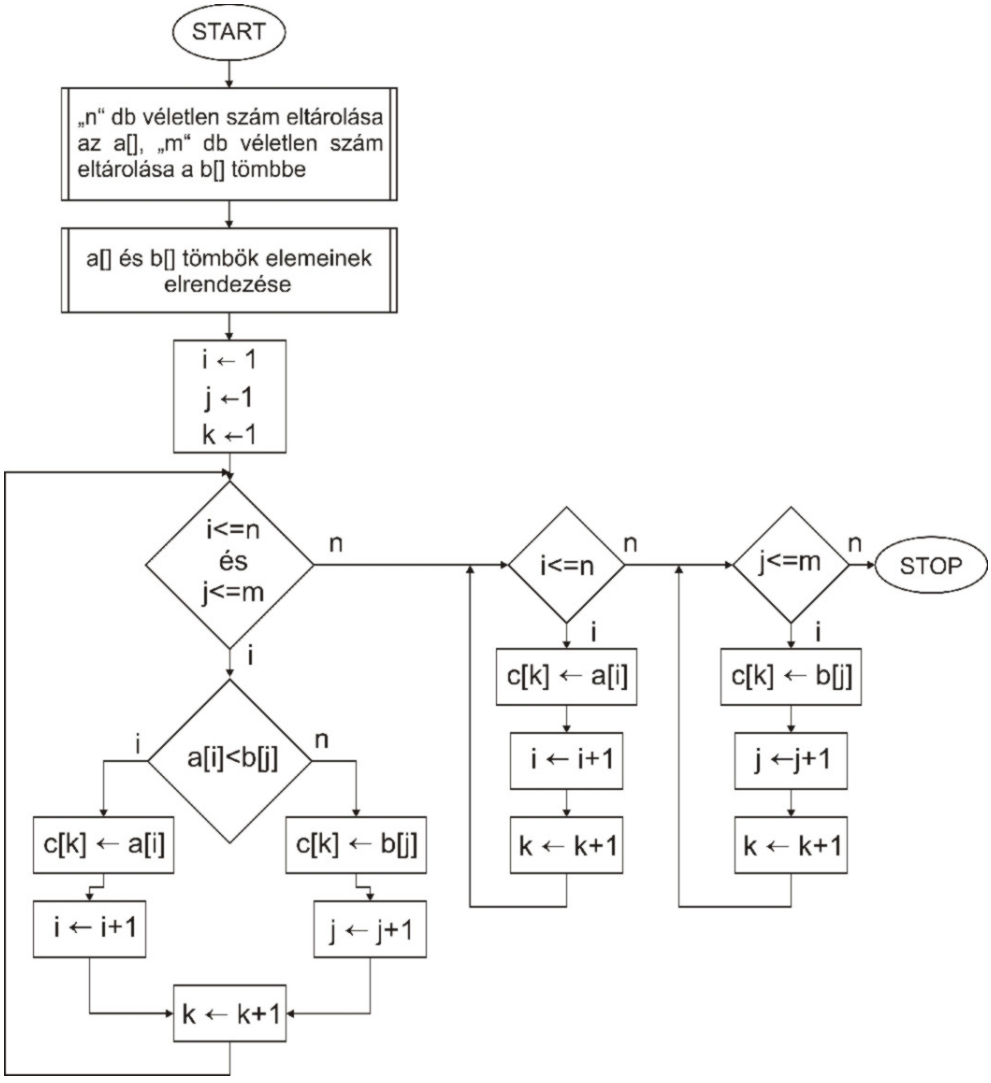
```
/* Létrehozunk három tömböt, az első kettőt fésüljük össze a harmadikba.
Ezért a 3. tömbnek legalább annyi eleműnek kell lennie, mint az első
kettőnek együtt. */
      int[] b=new int[100];
      int[] c=new int[200];

      System.out.print("Hany szam legyen az a tombben?
");
      n=in();
      i=0;
      while (i<n){
/* Véletlen számokkal feltöltjük az a tömböt. */
         a[i]=(int)(90*Math.random()+1);
         i=i+1;
         }

      i=0;
      while(i+1<n) {
/* Elrendezzük az a tömb elemeit, legkisebb alapján történő rendezéssel.
Természetesen itt bármilyen rendezést használhatunk. */
         g=i+1;
         while(g<n){
            if (a[g]<a[i]){
               x=a[g];
               a[g]=a[i];
               a[i]=x;
               }
            g=g+1;
            }
         i=i+1;
         }

      i=0;
      System.out.println("Az a tomb elemei rendezve:");
      while(i<n) {
         System.out.println("Az a tomb "+i+". eleme:
"+a[i]);
         i=i+1;
         }


      System.out.print("Hany szam legyen a b tombben? ");
      m=in();
      j=0;
      while (j<m){
/* Véletlen számokkal feltöltjük a b tömböt. */
         b[j]=(int)(90*Math.random()+1);
         j=j+1;
         }
```

```
      j=0;
      while(j+1<m) {
/* Elrendezzük a b tömb elemeit, legkisebb alapján történő rendezéssel.
Természetesen itt bármilyen rendezést használhatunk. */
         h=j+1;
         while(h<m){
         if (b[h]<b[j]){
            x=b[h];
            b[h]=b[j];
            b[j]=x;
            }
         h=h+1;
         }
      j=j+1;
      }

      j=0;
      System.out.println("A b tomb elemei rendezve:");
      while(j<m) {
         System.out.println("A b tomb "+j+". eleme:
"+b[j]);
         j=j+1;
         }

  //a[] és b[] tömb elemeinek összefésülése következik
      i=0;
      j=0;
      k=0;
      while (i<n && j<m){
/* Amíg mindkét összefésülendő tömbben van további elem, addig hajtódik
végre a ciklus. */
         if (a[i]<b[j]){
/* Ha az a tömb i-edik eleme kisebb, mint a b tömb j-edik eleme, akkor az a
tomb i-edik elemét rakja bele a c tömb aktuális elemébe. */
            c[k]=a[i];
            i=i+1;
            }
         else{
/* Ellenkező esetben a b tömb aktuális elemének értéke kerül a c tömb
aktuális elemébe. */
            c[k]=b[j];
            j=j+1;
            }
         k=k+1;
/* A c tömb következő elemére lépünk át. */
         }

      while (i<n){
```

```
/* Ez a ciklus akkor lép életbe, ha a b tömb kevesebb elemű, mint az a tömb.
Az a tömb maradék elemeit sorra belepakolja a c tömbbe. */
            c[k]=a[i];
            i=i+1;
            k=k+1;
            }
        while (j<m){
/ *Ez a ciklus akkor lép életbe, ha az a tömb kevesebb elemű, mint a b tömb.
A b tömb maradék elemeit sorra belepakolja a c tömbbe. */
            c[k]=b[j];
            j=j+1;
            k=k+1;
            }

        k=0;
        System.out.println("A ket tomb osszefesulese utan
az uj tomb elemei:");
        while (k<(n+m)){
/* Kiírjuk a c tömb elemeit. A ciklusfeltételben szereplő összeadás arra utal,
hogy annyi szám van a c tömbben, mint az a-ban és b-ben együtt. */
            System.out.println("Az uj tomb "+k+". eleme:
"+c[k]);
            k=k+1;
            }
        }
    }
```



Két tömb feltöltése véletlen számokkal, ezek elrendezése, majd összefésülése és eltárolása egy újabb tömbbe, eljárások használatával (Fesules.pas)

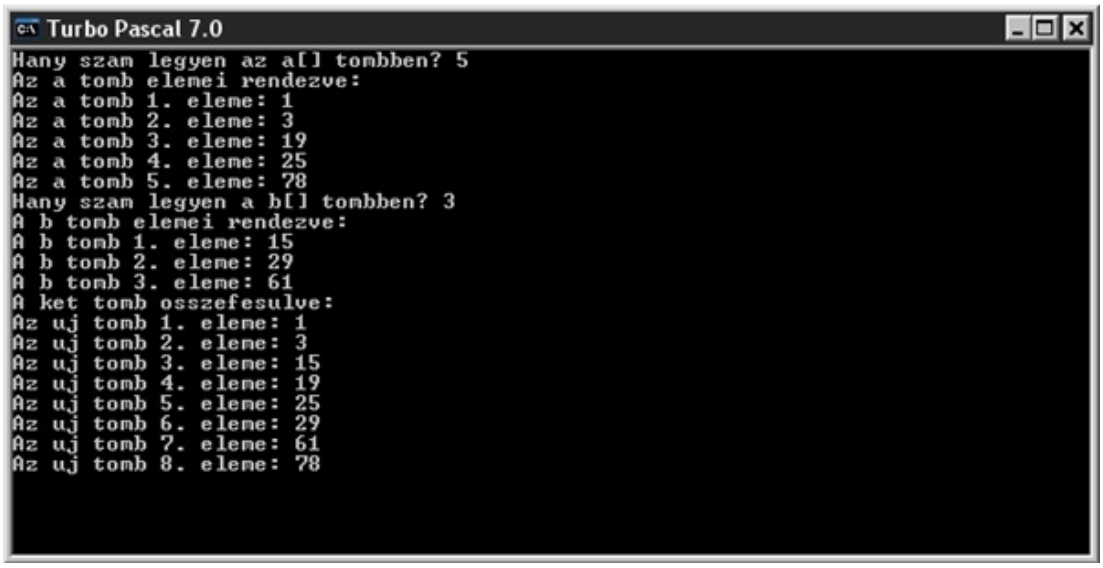



```
program fesules;
uses crt;
var i,j,n,k,x,h,m:integer;
    a:array [1..100] of integer;
```
{Létrehozunk három tömböt, az első kettőt fésüljük össze a harmadikba. Ezért a 3. tömbnek legalább annyi eleműnek kell lennie, mint az első kettőnek együtt. }
```
    b:array [1..100] of integer;
    c:array [1..200] of integer;

procedure tombfeltoltes1;
```
{Véletlen számokkal feltöltjük az a tömböt. }

```
begin
   i:=1;
   write('Hany szam legyen az a[] tombben? ');
   readln(n);
   while (i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      end;
   end;

procedure tombfeltoltes2;
{Véletlen számokkal feltöltjük az b tömböt. }
begin
   j:=1;
   write('Hany szam legyen a b[] tombben? ');
   readln(m);
   while(j<=m) do begin
      b[j]:=trunc(90*random+1);
      j:=j+1;
      end;
   end;

procedure rendezes1;
{Elrendezzük az a tömb elemeit, legkisebb alapján történő rendezéssel.
Természetesen itt bármilyen rendezést használhatunk. }
begin
   i:=1;
   while (i<n) do begin
      j:=i+1;
      while(j<=n) do begin
         if (a[j]<a[i]) then begin
            x:=a[j];
            a[j]:=a[i];
            a[i]:=x;
            end;
         j:=j+1;
         end;
      i:=i+1;
      end;
   writeln('Az a tomb elemei rendezve:');
   i:=1;
   while(i<=n) do begin
      writeln('Az a tomb ',i,'. eleme: ',a[i]);
      i:=i+1;
      end;
   end;

procedure rendezes2;
```

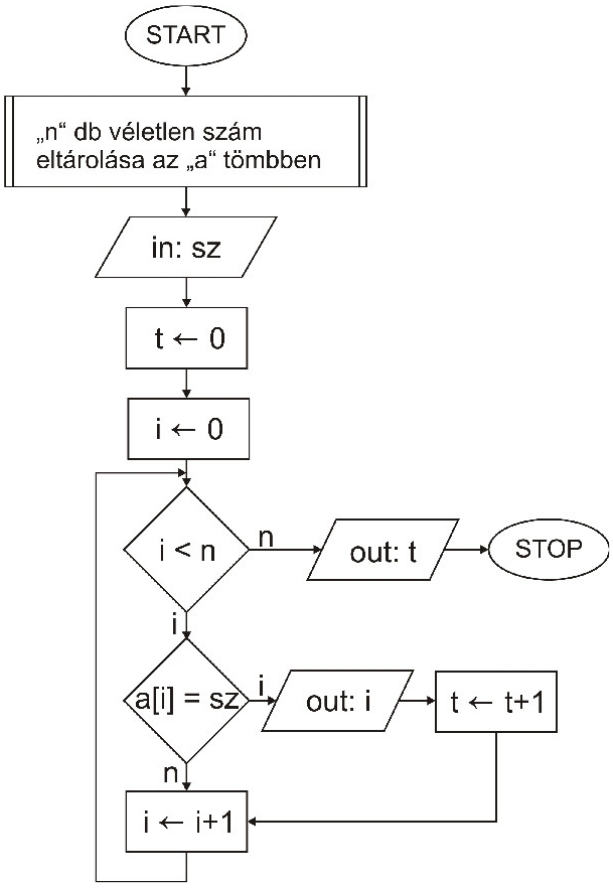
```
{Elrendezzük a b tömb elemeit, legkisebb alapján történő rendezéssel.
Természetesen itt bármilyen rendezést használhatunk. }
begin
   j:=1;
   while (j<m) do begin
      h:=j+1;
      while (h<=m) do begin
         if (b[h]<b[j]) then begin
            x:=b[h];
            b[h]:=b[j];
            b[j]:=x;
            end;
         h:=h+1;
         end;
      j:=j+1;
      end;
   writeln ('A b tomb elemei rendezve:');
   j:=1;
   while (j<=m) do begin
      writeln ('A b tomb ',j,'. eleme: ',b[j]);
      j:=j+1;
      end;
   end;

procedure osszefesules;
begin
   i:=1;
   j:=1;
   k:=1;
   while ((i<=n) and (j<=m)) do begin
{Amíg mindkét összefésülendő tömbben van további elem, addig hajtódik
végre a ciklus. }
      if (a[i]<b[j]) then begin
{Ha az a tömb i-edik eleme kisebb, mint a b tömb j-edik eleme, akkor az a
tömb i-edik elemét rakja bele a c tömb aktuális elemébe. }
         c[k]:=a[i];
         i:=i+1;
         end
      else begin
{Ellenkező esetben a b tömb aktuális elemének értéke kerül a c tömb aktuális
elemébe. }
         c[k]:=b[j];
         j:=j+1;
         end;
      k:=k+1;
{A c tömb következő elemére lépünk át. }
      end;

   while (i<=n) do begin
```

```
{Ez a ciklus akkor lép életbe, ha a b tömb kevesebb elemű, mint az a tömb.
Az a tömb maradék elemeit sorra belepakolja a c tömbbe. }
      c[k]:=a[i];
      i:=i+1;
      k:=k+1;
      end;

   while (j<=m) do begin
{Ez a ciklus akkor lép életbe, ha az a tömb kevesebb elemű, mint a b tömb. A
b tömb maradék elemeit sorra belepakolja a c tömbbe. }
      c[k]:=b[j];
      j:=j+1;
      k:=k+1;
      end;
   end;

procedure kiiras;
begin
   k:=1;
   writeln('A ket tomb osszefesulve:');
   while (k<=(n+m)) do begin
{Kiírjuk a c tömb elemeit. A ciklusfeltételben szereplő összeadás arra utal,
hogy annyi szám van a c tömbben, mint az a-ban és b-ben együtt. }
      writeln('Az uj tomb ',k,'. eleme: ',c[k]);
      k:=k+1;
      end;
   end;

BEGIN
   clrscr;
   tombfeltoltes1;
   rendezes1;
   tombfeltoltes2;
   rendezes2;
   osszefesules;
   kiiras;
   readkey;
END.
```


Turbo Pascal 7.0
Hany szam legyen az a[] tombben? 5
Az a tomb elemei rendezve:
Az a tomb 1. eleme: 1
Az a tomb 2. eleme: 3
Az a tomb 3. eleme: 19
Az a tomb 4. eleme: 25
Az a tomb 5. eleme: 78
Hany szam legyen a b[] tombben? 3
A b tomb elemei rendezve:
A b tomb 1. eleme: 15
A b tomb 2. eleme: 29
A b tomb 3. eleme: 61
A ket tomb osszefesulve:
Az uj tomb 1. eleme: 1
Az uj tomb 2. eleme: 3
Az uj tomb 3. eleme: 15
Az uj tomb 4. eleme: 19
Az uj tomb 5. eleme: 25
Az uj tomb 6. eleme: 29
Az uj tomb 7. eleme: 61
Az uj tomb 8. eleme: 78

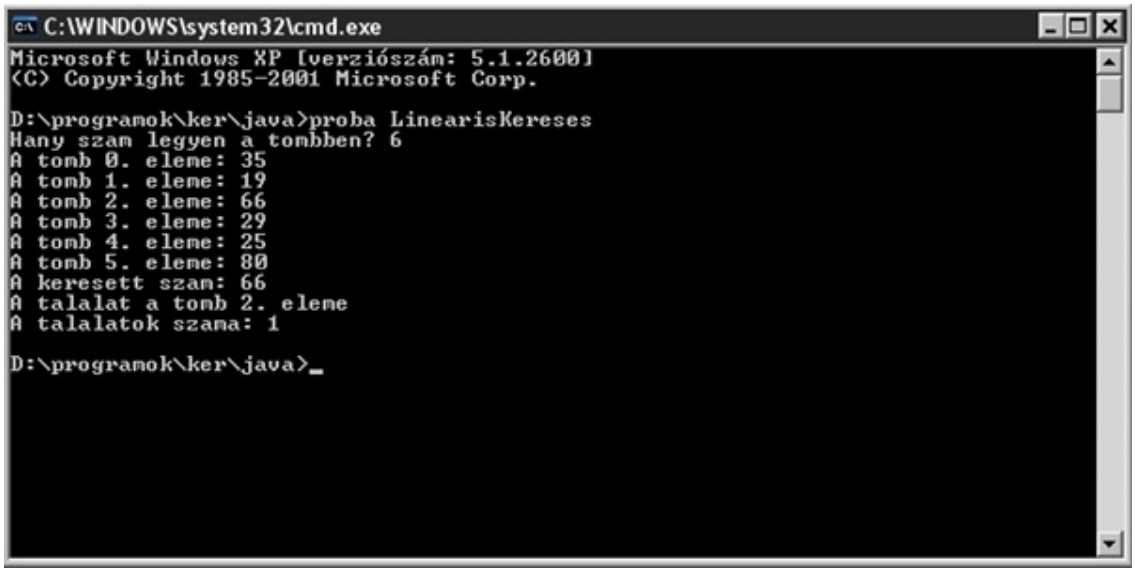
2.2.4 Keresési algoritmusok

Tömb feltöltése véletlen számokkal, majd egy bekért szám keresése a tömbben lineáris kereséssel (LinearisKereses.java)

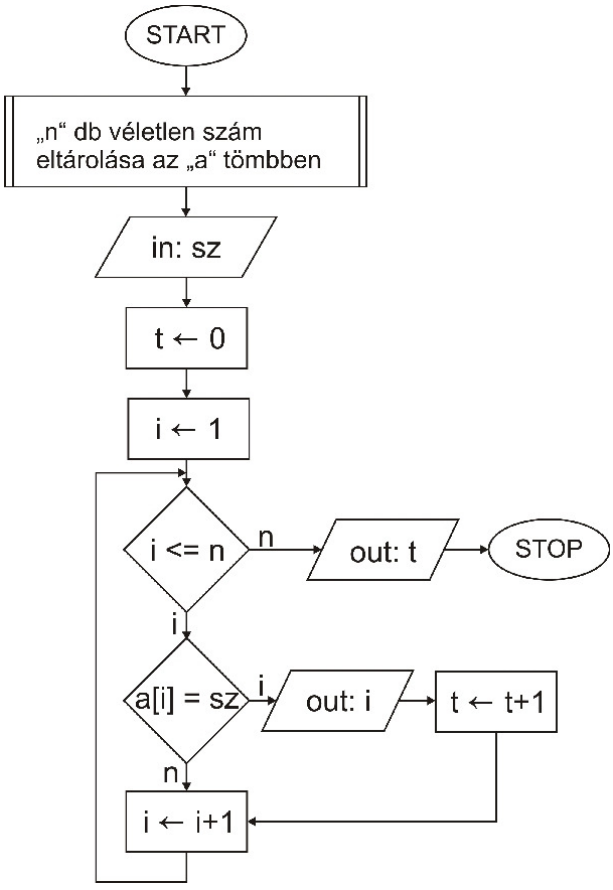



```
import java.io.*;
public class LinearisKereses {
   public static int in() throws Exception {
/* Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza. */
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      String s=x.readLine();
      int i=Integer.parseInt(s);
      return i;
      }
   public static void main(String[] args) throws
Exception{

      int i,n,sz,t;
```

```
      int[] a=new int[100];

      System.out.print("Hany szam legyen a tombben? ");
      n=in();
      i=0;
      while (i<n){
         a[i]=(int)(90*Math.random()+1);
         i++;
         }

      i=0;
      while(i<n) {
         System.out.println("A tomb "+i+". eleme:
"+a[i]);
         i++;
         }

      System.out.print("A keresett szam: ");
      sz=in();
/* Bekérjük a keresett számot. */
      t=0;
/* t = találatok száma, melyet kezdetben 0-ra állítunk */
      i=0;
      while (i<n){
         if(a[i]==sz){
/* Sorba megyünk a tömb elemein és megvizsgáljuk, hogy megegyezik e a
keresett számmal. Ha igen, növeljük a találatok számát 1-el és kiírjuk a
képernyőre, hogy melyik elem egyezik a keresett számmal. Java-ban, ha
egyenlőséget szeretnénk vizsgálni feltétel megadásakor, akkor azt dupla
egyenlőségjellel tehetjük meg. */
         t++;
         System.out.println("A talalat a tomb "+i+".
eleme");
         }
      i++;
      }
      System.out.println("A talalatok szama: "+t);
      }
   }
```

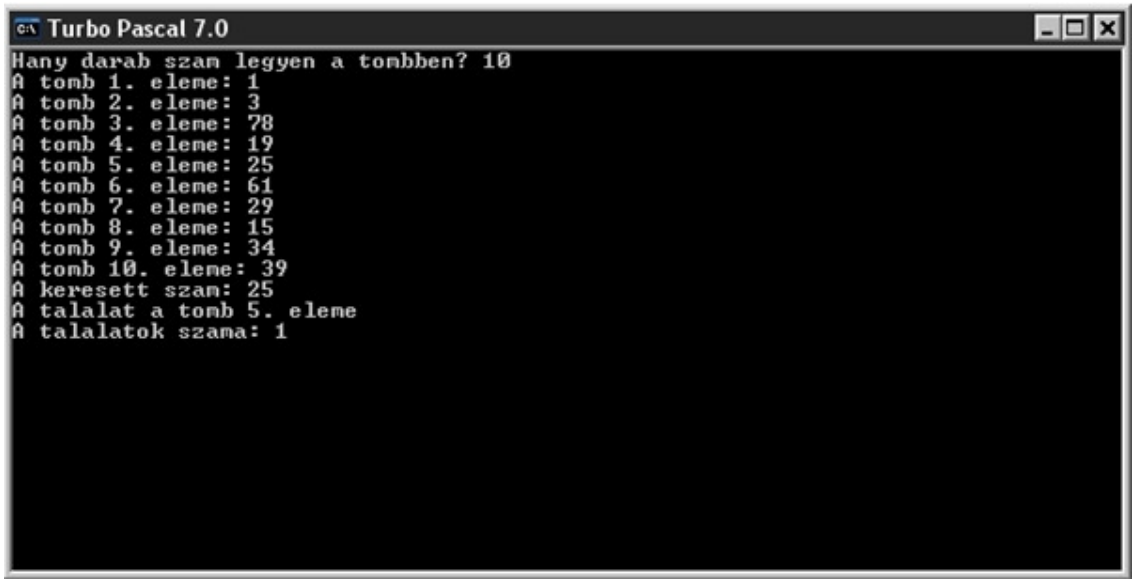


Tömb feltöltése véletlen számokkal, majd egy bekért szám keresése a tömbben lineáris kereséssel, eljárások használatával (LinearisKereses.pas)

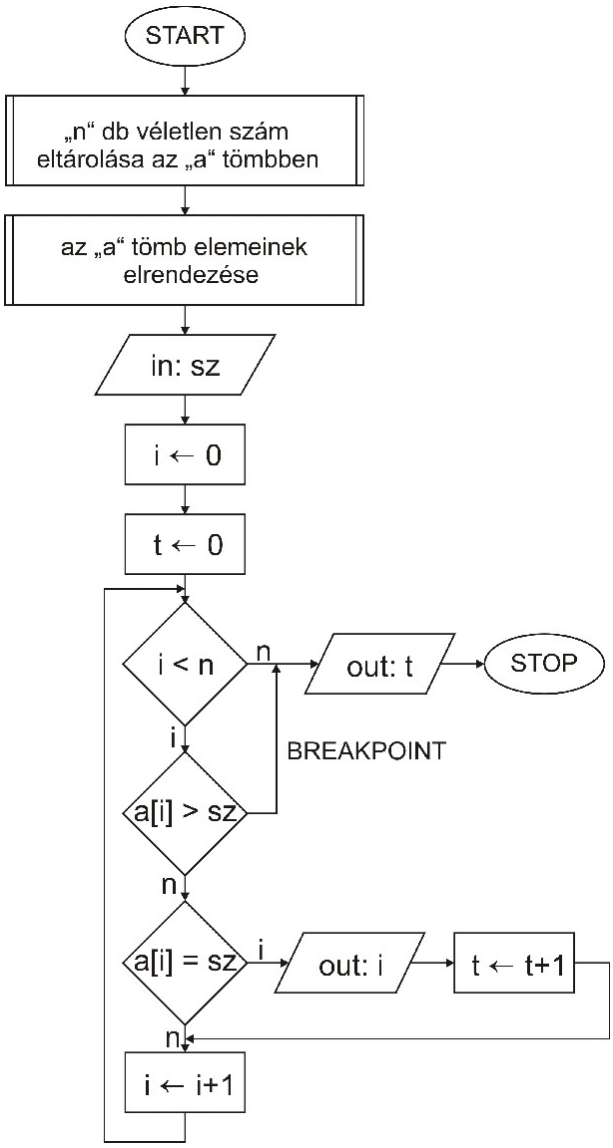


```
program LinearisKereses;
uses crt;
var i,n,sz,t:integer;
    a:array [1..50] of integer;

procedure tombfeltoltes;
begin
   i:=1;
   write('Hany darab szam legyen a tombben? ');
   readln(n);
   while (i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      end;

   i:=1;
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme: ',a[i]);
      i:=i+1;
      end;
   end;

procedure bekeres;
{Bekérjük a billentyűzetről a keresendő számot. }
begin
   write('A keresett szam: ');
   readln(sz);
   end;

BEGIN
   clrscr;
   tombfeltoltes;
   bekeres;
   t:=0;
{t = találatok száma, melyet kezdetben 0-ra állítunk }
   i:=1;
   while (i<=n) do begin
      if(a[i]=sz) then begin
{Végigmegyünk a tömb elemein és megvizsgáljuk, hogy a tömb aktuális
eleme megegyezik e a keresett számmal. Ha igen, akkor a találatok számát
növeljük, valamint kiírjuk a képernyőre, hogy melyik tömb elemnél van
egyezés. }
         t:=t+1;
         writeln('A talalat a tomb ',i,'. eleme');
         end;
      i:=i+1;
      end;
   writeln('A talalatok szama: ',t);
```

```
END.
```


Turbo Pascal 7.0
Hany darab szam legyen a tombben? 10
A tomb 1. eleme: 1
A tomb 2. eleme: 3
A tomb 3. eleme: 78
A tomb 4. eleme: 19
A tomb 5. eleme: 25
A tomb 6. eleme: 61
A tomb 7. eleme: 29
A tomb 8. eleme: 15
A tomb 9. eleme: 34
A tomb 10. eleme: 39
A keresett szam: 25
A talalat a tomb 5. eleme
A talalatok szama: 1

Tömb feltöltése véletlen számokkal, elrendezése, majd egy bekért szám keresése a tömbben szekvenciális kereséssel (SzekvencialisKereses.java)

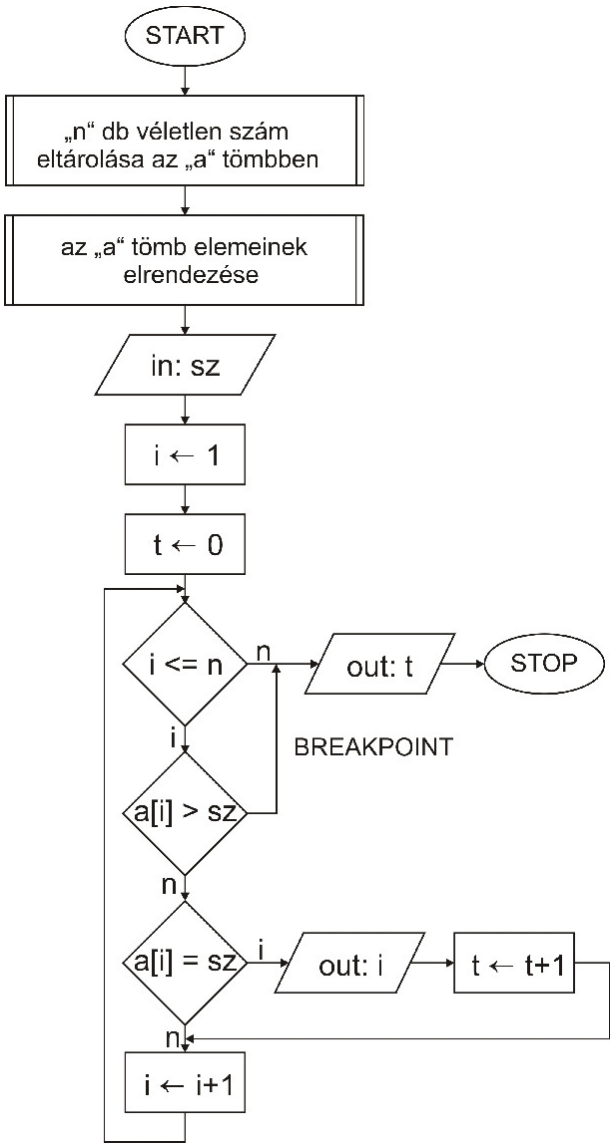



```
import java.io.*;
public class SzekvencialisKereses {
public static int in() throws Exception{
/* Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza */
   LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
   String s=x.readLine();
   int i=Integer.parseInt(s);
   return i;
   }
```

```
public static void main(String[] args) throws Exception{
   int i,j,n,sz,t,x;
   int[] a=new int[100];
   System.out.print("Hany szam legyen a tombben? ");
   n=in();
   i=0;
   while (i<n){
      a[i]=(int)(90*Math.random()+1); i++;
      }
   i=0;
   while(i<n) {
      System.out.println("A tomb "+i+". eleme: "+a[i]);
      i++;
      }
   i=0;
   while (i<n){
/* Legkisebb alapján történő rendezéssel elrendezzük a tömb elemeit. */
      j=i+1;
      while(j<n){
         if (a[j]<a[i]){
            x=a[j];
            a[j]=a[i];
            a[i]=x;
            }
         j=j+1;
         }
      i=i+1;
      }
   i=0;
   System.out.println("A tomb elemei elrendezve:");
   while (i<n){
      System.out.println(a[i]);
      i=i+1;
      }
   System.out.print("A keresett szam: ");
   sz=in();
/* Bekérjük a keresett számot. */
   i=0;
   t=0;
/* t = találatok száma, melyet kezdetben 0-ra állítunk */
   while (i<n) {
      if(a[i]>sz) break;
/* Mivel rendezett tömbről van szó, ezért megvizsgáljuk, hogy a tömb
aktuálisan vizsgált eleme nagyobb e, mint a keresendő szám. Ha igen, akkor
a további vizsgálat fölösleges, ezért break-el megszakítjuk a további
vizsgálódást. */
      if(a[i]==sz) {
```

```
/* Ha az előbbi feltétel nem teljesül, azaz az aktuálisan vizsgált tömb elem
kisebb, vagy egyenlő mint a keresett szám, akkor megvizsgáljuk, hogy van e
egyezés. Ha van, kiírjuk a találat indexét és a találatok számát növeljük
eggyel. */
          System.out.print("A keresett szam a sorozat
"+i+". eleme.");
          t=t+1;
          }
       i=i+1;
       }
   System.out.print("Talalatok szama: "+t);
   }
}
```

Tömb feltöltése véletlen számokkal, elrendezése, majd egy bekért szám keresése a tömbben szekvenciális kereséssel, eljárások használatával (SzekvencialisKereses.pas)




```
program SzekvencialisKereses;
uses crt;
var i,j,n,sz,x,t:integer;
    a:array [1..100] of integer;

procedure tombfeltoltes;
begin
   i:=1;
   write('Hany db szam legyen a tombben? ');
   readln(n);
```

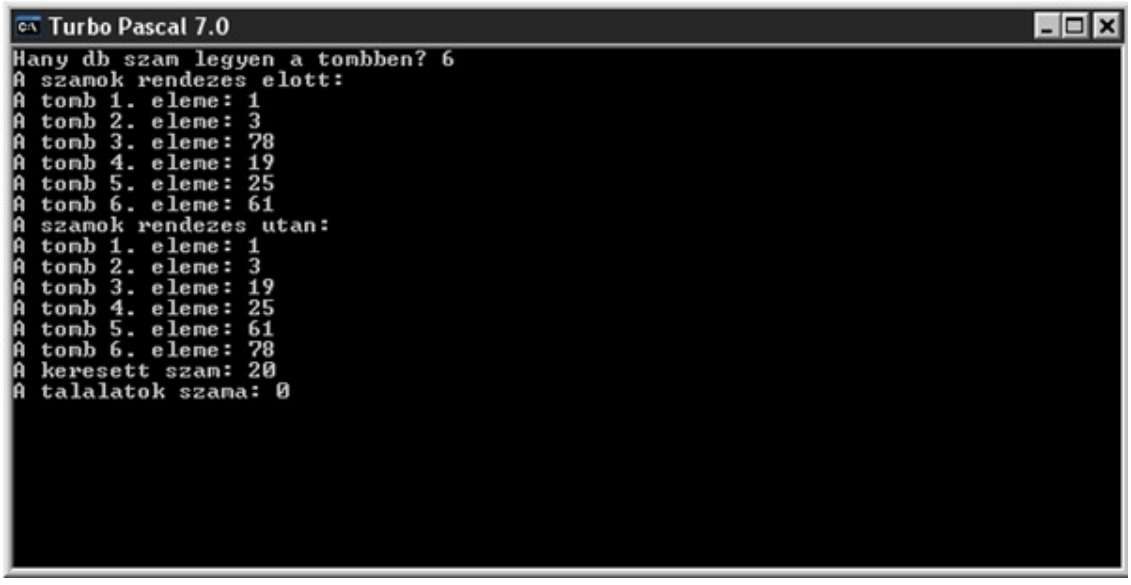
```
   while (i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      end;

   writeln('A szamok rendezes elott:');
   i:=1;
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme: ',a[i]);
      i:=i+1;
      end;
   end;

procedure rendezes;
{Legkisebb alapján történő rendezéssel elrendezzük a tömb elemeit. }
begin
   i:=1;
   while(i<n) do begin
      j:=i+1;
      while (j<=n) do begin
         if (a[j]<a[i]) then begin
            x:=a[j];
            a[j]:=a[i];
            a[i]:=x;
            end;
         j:=j+1;
         end;
      i:=i+1;
      end;
   writeln('A szamok rendezes utan:');
   i:=1;
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme: ',a[i]);
      i:=i+1;
      end;
   end;

procedure kereses;
begin
   write('A keresett szam: ');
   readln(sz);
{Bekérjük a keresett számot. }
   t:=0;
{t = találatok száma, melyet kezdetben 0-ra állítunk }
   i:=1;
   while (i<=n) do begin
      if (sz<a[i]) then break;
{Mivel rendezett tömbről van szó, ezért megvizsgáljuk, hogy a tömb
aktuálisan vizsgált eleme nagyobb e, mint a keresendő szám. Ha igen, akkor
```

a további vizsgálat fölösleges, ezért break-el megszakítjuk a további vizsgálódást. }

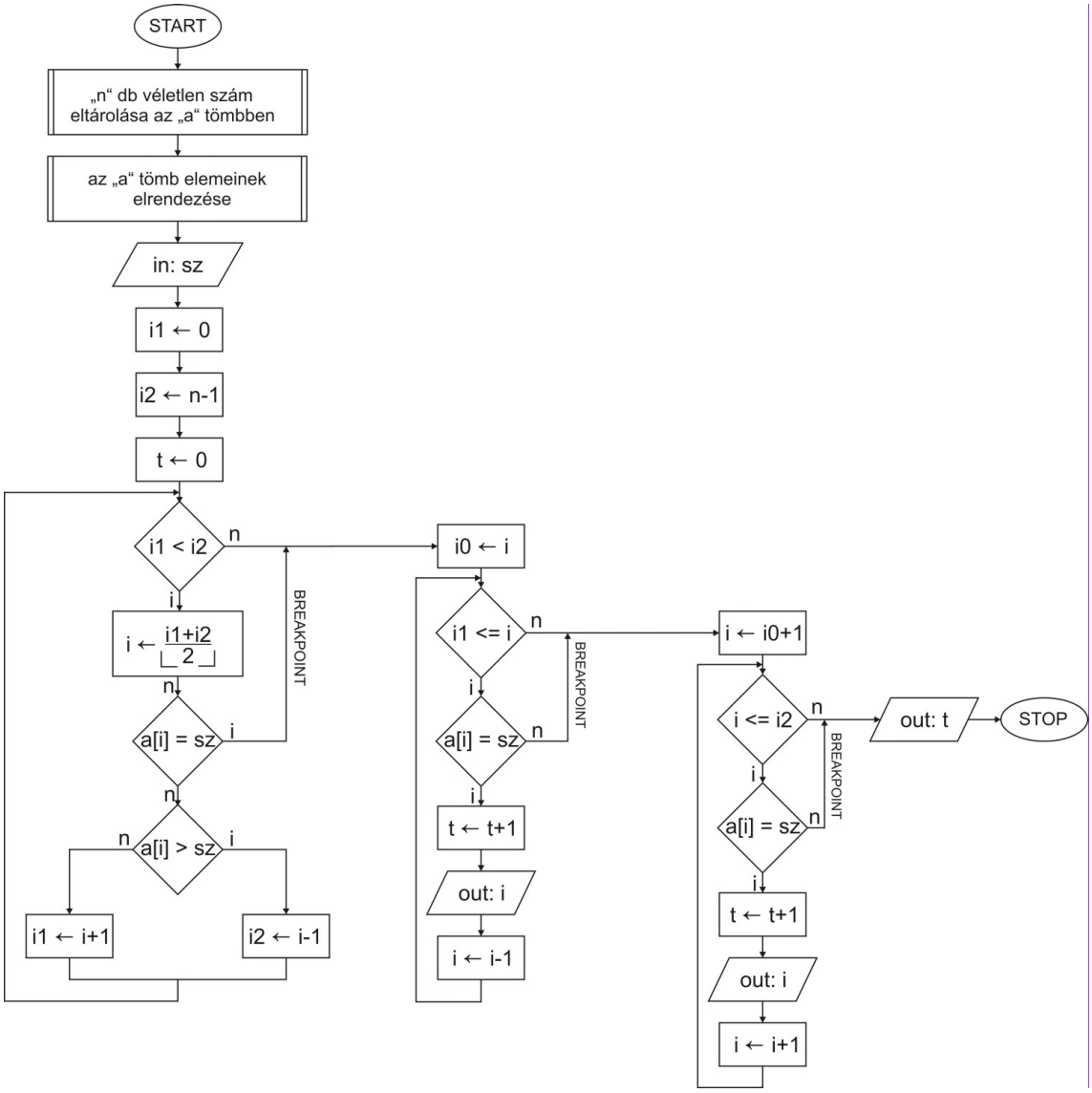
```
      if (sz=a[i]) then begin
```

{Ha az előbbi feltétel nem teljesül, azaz az aktuálisan vizsgált tömb elem kisebb, vagy egyenlő mint a keresett szám, akkor megvizsgáljuk, hogy van e egyezés. Ha van, kiírjuk a találat indexét és a találatok számát növeljük eggyel. }

```
         t:=t+1;
         writeln('A talalat a sorozat ',i,'. eleme');
         end;
      i:=i+1;
      end;
   writeln('A talalatok szama: ',t);
   end;

BEGIN
   clrscr;
   tombfeltoltes;
   rendezes;
   kereses;
   readkey;
END.
```



Tömb feltöltése véletlen számokkal, elrendezése, majd egy bekért szám keresése a tömbben bináris kereséssel, eljárások használatával (BinarisKereses.java)

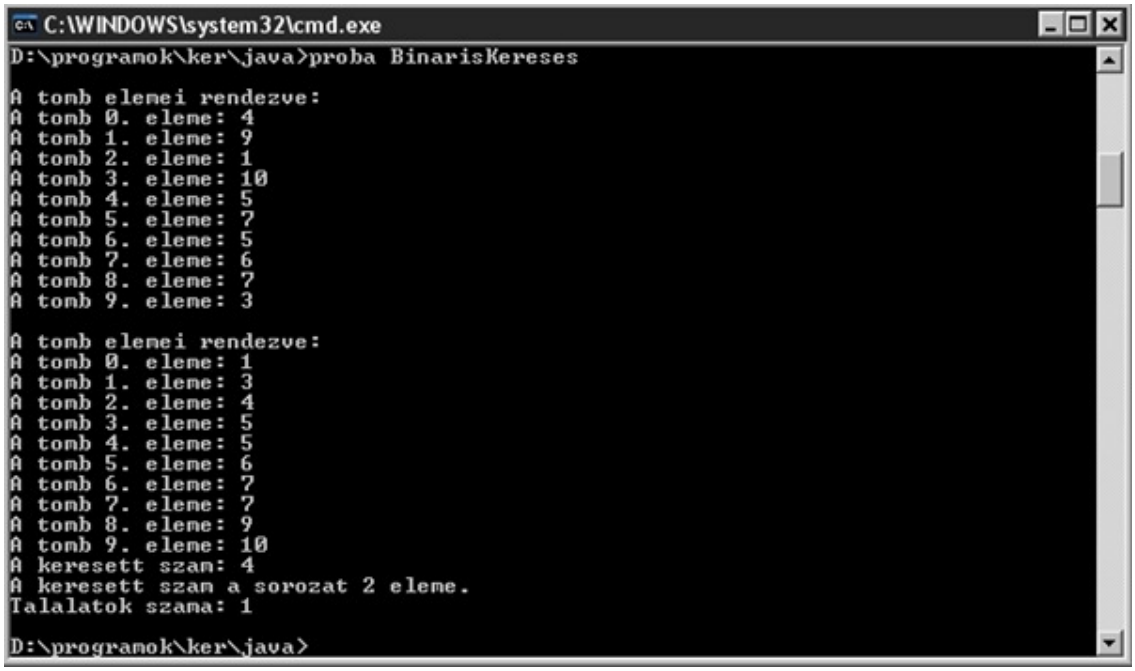



```
import java.io.*;
public class BinarisKereses{
   public static int n=10;
   public static int [] a=new int[n];

   public static int in() throws Exception {
/* Ez a beolvasást végrehajtó függvény, mely egész számot ad vissza. */
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      String s=x.readLine();
```

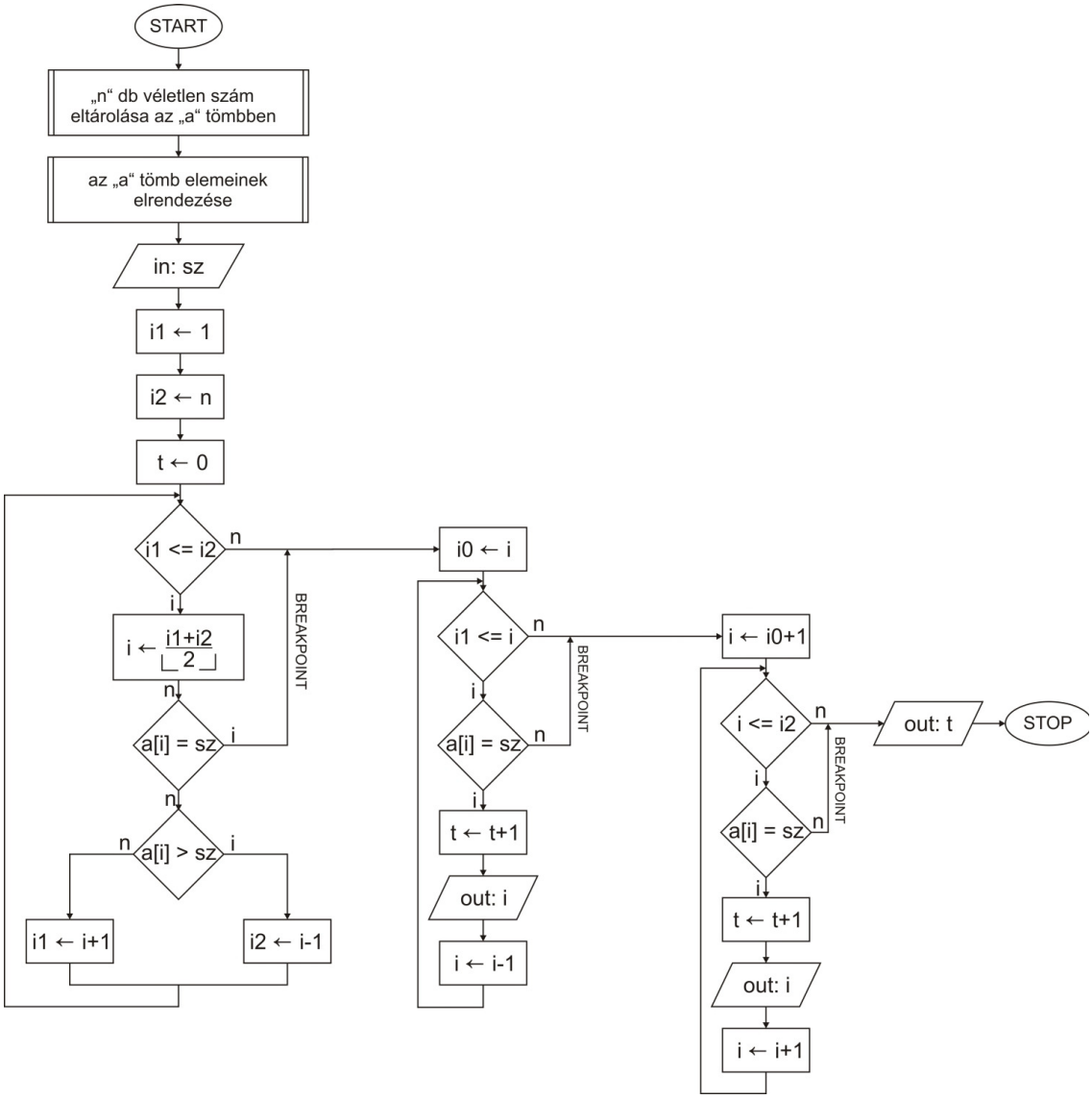
```
      int i=Integer.parseInt(s);
      return i;
      }

   public static void feltolt(){
      int i=0;
      while (i<n) {
         a[i]=(int)(Math.random()*10)+1;
         i=i+1;
         }
      }

   public static void rendez(){
/* Legkisebb alapján történő rendezéssel elrendezzük a tömb elemeit. */
      int i=0, x, j;
      while (i<n) {
         j=i+1;
         while (j<n) {
            if (a[i]>a[j]) {
               x=a[i];
               a[i]=a[j];
               a[j]=x;
               }
            j=j+1;
            }
         i=i+1;
         }
      }

   public static void kiir(){
      System.out.println(" ");
      System.out.println("A tomb elemei rendezve:");
      int i=0;
      while (i<n){
         System.out.println("A tomb "+i+". eleme:
"+a[i]);
         i=i+1;
         }
      }

   public static void keres() throws Exception{
      int i1,i2,i=0,talalat,szam,i0;
      i1=0;
/* i1-be eltároljuk a tömb első elemének indexét, azaz a 0-át. */
      i2=n-1;
/* i2-be eltároljuk a tömb utolsó elemének indexét, mivel Java-ban 0-val
kezdődik az indexelés, n-1-et rakjuk bele. */
      talalat=0;
      System.out.print("A keresett szam: ");
```

```
      szam=in();
/* Bekérjük a keresett számot. /*
      while (i1<i2) {
         i=(int)((i1+i2)/2);
/* i-be beletesszük a középső elem indexét, amelyet az i1 és i2 aktuális értékének átlagaként kapunk meg. Mivel nem egész számot kapunk, típuskényszerítéssel egész számmá alakítjuk az átlagot. */
         if (a[i]==szam) break;
/* Ha a tömb i-edik eleme megegyezik a számmal, azaz találatunk van break-el kilépünk a ciklusból. */
         if (a[i]<szam)
/* Ha a tömb i-edik eleme kisebb, mint a keresett szám, akkor már csak az i-edik elem után találhatjuk meg a számunkat, mivel a tömb elemeit elrendeztük. Ha az elem nagyobb a keresett számnál, akkor pedig már csak az i-edik elem előtti számok között keresünk tovább. */
            i1=i+1;
         else i2=i-1;
         }
      i0=i;
/* i0-ba eltároljuk az i aktuális értékét */
      while (i1<=i){
/* A ciklus segítségével megvizsgáljuk, hogy az i-edik tömbelem előtti számok között van e találat, ha igen, kiírjuk a találat indexét, és a találatok számát növeljük eggyel, majd az i értékét csökkentjük eggyel, így vizsgálva a további elemeket. Ha nincs találatunk break-kel kilépünk a ciklusból. */
         if (a[i]==szam) {
            System.out.println("A keresett szam a sorozat
"+i+" eleme.");
            talalat=talalat+1;
            i=i-1;
            }
         else break;
         }
      i=i0+1;
/* i-be beletesszük az i0-nál eggyel nagyobb értéket. Erre azért van szükség, mert miután az előző ciklus lefutott, már csak az i0-adik tömbelem után lehet találatunk. */
      while (i2>=i) {
/* A ciklus segítségével megvizsgáljuk, hogy az i-edik tömbelem utáni számok között van e találat, ha igen, kiírjuk a találat indexét, és a találatok számát növeljük eggyel, majd az i értékét is növeljük eggyel, így vizsgálva a további elemeket. Ha nincs találatunk break-kel kilépünk a ciklusból. */
         if (a[i]==szam) {
            System.out.println("A keresett szam a sorozat
"+i+" eleme.");
            talalat=talalat+1;
            i=i+1;
            }
         else break;
```

```
         }
      System.out.println("Talalatok szama: "+talalat);
      }

   public static void main (String [] args) throws
Exception{
      feltolt();
      kiir();
      rendez();
      kiir();
      keres();
      }
   }
```



Tömb feltöltése véletlen számokkal, elrendezése, majd egy bekért szám keresése a tömbben bináris kereséssel, eljárások használatával (BinarisKereses.pas)




```
program BinarisKereses;
uses crt;
var i,i0,i1,i2,t,sz,x,j,n:integer;
    a:array [1..100] of integer;

procedure tombfeltoltes;
begin
   i:=1;
   write('Hany darab szam legyen a tombben? ');
   readln(n);
```

```
   while(i<=n) do begin
      a[i]:=trunc((10)*random+1);
      i:=i+1;
      end;
   end;

procedure rendezes;
{Legkisebb alapján történő rendezéssel elrendezzük a tömb elemeit. }
begin
   i:=1;
   while(i<n) do begin
      j:=i+1;
      while(j<=n) do begin
         if (a[j]<a[i]) then begin
            x:=a[j];
            a[j]:=a[i];
            a[i]:=x;
            end;
         j:=j+1;
         end;
      i:=i+1;
      end;

   writeln('A szamok rendezes utan:');
   i:=1;
   while(i<=n) do begin
      writeln('A tomb ',i,'. eleme: ',a[i]);
      i:=i+1;
      end;
   end;

procedure kereses;
begin
   t:=0;
   write('A keresett szam: ');
   readln(sz);
{Bekérjük a keresett számot. }
   i1:=1;
{i1-be eltároljuk a tömb első elemének indexét, azaz 1-et. }
   i2:=n;
{i2-be eltároljuk a tömb utolsó elemének indexét, azaz n-et. }
   while(i1<=i2) do begin
      i:=trunc((i1+i2)/2);
{i-be beletesszük a középső elem indexét, amelyet az i1 és i2 aktuális értékének átlagaként kapunk meg. Mivel nem egész számot kapunk, típuskényszerítéssel egész számmá alakítjuk az átlagot. }
      if (a[i]=sz) then break;
{Ha a tömb i-edik eleme megyegyezik a számmal, azaz találatunk van break-el kilépünk a ciklusból. }
```

```
      if (sz<a[i]) then begin
```
{Ha a tömb i-edik eleme kisebb, mint a keresett szám, akkor már csak az i-edik elem után találhatjuk meg a számunkat, mivel a tömb elemeit elrendeztük. Ha az elem nagyobb a keresett számnál, akkor pedig már csak az i-edik elem előtti számok között keresünk tovább. }
```
          i2:=i-1;
          end
     else
          i1:=i+1;
     end;

   i0:=i;
```
{i0-ba eltároljuk az i aktuális értékét }
```
   while (i1<=i) do begin
```
{A ciklus segítségével megvizsgáljuk, hogy az i-edik tömbelem előtti számok között van e találat, ha igen, kiírjuk a találat indexét, és a találatok számát növeljük eggyel, majd az i értékét csökkentjük eggyel, így vizsgálva a további elemeket. Ha nincs találatunk break-kel kilépünk a ciklusból. }
```
      if (a[i]=sz) then begin
         writeln('A talalat a sorozat ',i,'. eleme');
         t:=t+1;
         i:=i-1;
         end
      else break;
      end;

   i:=i0+1;
```
{i-be beletesszük az i0-nál eggyel nagyobb értéket. Erre azért van szükség, mert miután az előző ciklus lefutott, már csak az i0-adik tömbelem után lehet találatunk. }
```
   while (i<=i2) do begin
```
{A ciklus segítségével megvizsgáljuk, hogy az i-edik tömbelem utáni számok között van e találat, ha igen, kiírjuk a találat indexét, és a találatok számát növeljük eggyel, majd az i értékét is növeljük eggyel, így vizsgálva a további elemeket. Ha nincs találatunk break-kel kilépünk a ciklusból. }
```
      if (a[i]=sz) then begin
         writeln('A talalat a sorozat ',i,'. eleme');
         t:=t+1;
         i:=i+1;
         end
      else break;
      end;

   if (t=0) then writeln ('Nincs talalat.')
```
{Ha a találatok száma , akkor kiírjuk, hogy Nincs találat, egyébként pedig kiírjuk a találatok számát. }
```
   else writeln('A talalatok szama: ',t);
   end;
```

```
BEGIN
   clrscr;
   tombfeltoltes;
   rendezes;
   kereses;
END.
```

2.2.5 Fájlkezelés

Szövegfájl (proba.txt) tartalmának beolvasása, az "a" és "e" betűk számának kiírása (BeolvasoSzamolo.java)

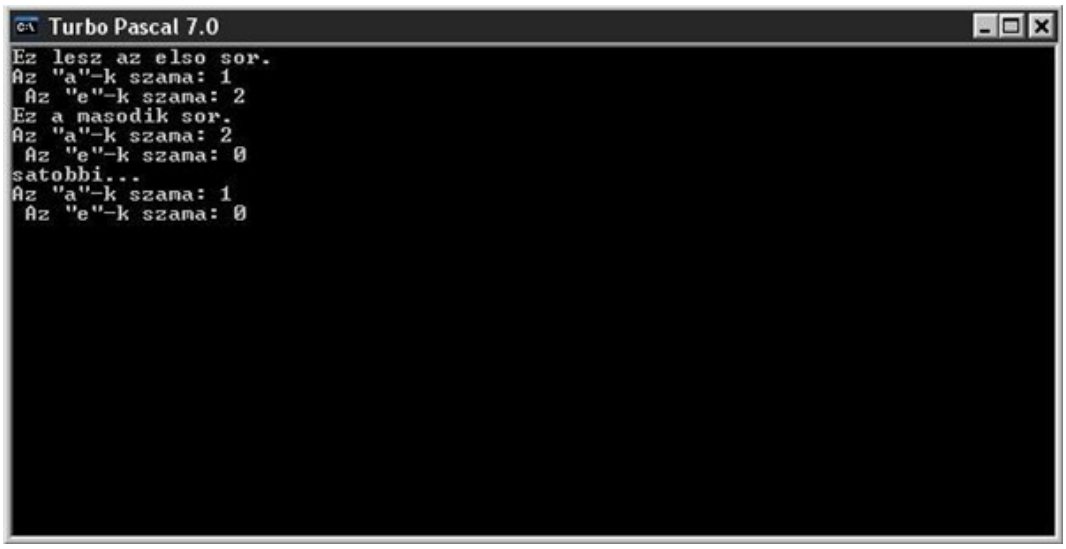
```
import java.io.*;
/* A java.io csomag importálása a be és kimeneti műveletekhez szükségesek. */
public class BeolvasoSzamolo{
   public static int szamola (String s){
/* A szamola függvény megszámolja az a betűk számát és egészként visszadja azt. */
      int i,ta=0;
/* ta = a betűre vonatkozó találatok száma */
      for (i=0; i<s.length();i++){
/* A ciklus addig megy, amíg a String végére nem érünk. A String osztály .length() függvénye számként visszaadja a String hosszát. */
         if(s.charAt(i)=='a'){
/* Megvizsgáljuk, hogy a string i-edik karaktere a betű e. Ha igen, növeljük az a betűre vonatkozó találatok számát. */
            ta++;
            }
         }
      return ta;
/* Visszaadjuk az a betűre vonatkozó találatszámot. */
      }

   public static int szamole (String s){
/* A szamole függvény megszámolja az e betűk számát és egészként visszaadja azt. */
      int i,te=0;
/* te = e betűre vonatkozó találatok száma */
      for (i=0; i<s.length();i++){
         if(s.charAt(i)=='e'){
/* Megvizsgáljuk, hogy a String i-edik karaktere e betű e. Ha igen, növeljük az e betűre vonatkozó találatok számát. */
            te++;
            }
         }
      return te;
/* Visszaadjuk az e betűre vonatkozó találatszámot */
      }
```

```
   public static void main (String[] args) throws
Exception{
/* Az io műveleteknél kötelező a kivételkezelés. */
       LineNumberReader f=new LineNumberReader( new
InputStreamReader(
                  new FileInputStream("proba.txt")));
/* Ahhoz, hogy sorokat olvassunk be, szükségünk van csatornák
megnyitására. Jelen esetben létrehozunk egy f nevű LineNumberReader
csatornát, beleágyazunk egy InputStreamReader-t, azaz bemeneti csatornát,
amelybe egy FileInputStream-et, azaz fájl bemeneti csatornát ágyazunk be,
amellyel a proba.txt-t érjük el. */
       String s;
       s=f.readLine();
/* Az s nevű String típusú változóba beolvassuk az f (proba.txt) egy sorát. */
       while(s!=null){
/* Amíg a beolvasott sor nem üres, addig él a ciklus. */
          System.out.println(s);
          System.out.println("Az a betuk szama:
"+szamola(s));
/* A kiírásba meghívjuk a szamola(s) függvényt, ami megszámolja a sorban
lévő a betűket és egész számként visszaadja az értéket. */
          System.out.println("Az e betuk szama:
"+szamole(s));
          s=f.readLine();
          }
       f.close();
/* Lezárjuk a csatornát. */
       }
   }
```

Szövegfájl (proba.txt) tartalmának beolvasása, az "a" és "e" betűk számának kiírása (BeolvasoSzamolo.pas)

```
program BeolvasoSzamolo;
uses crt;
var f:text;
    s:string;

function szamola(s:string):integer;
{A szamola függvény megszámolja az a betűk számát és egészként visszadja azt. }
var i,t:integer;
begin
   t:=0;
   for i:=1 to length(s) do
{A ciklus addig tart, amíg a stringnek nincs vége. A length(s) visszaadja a string hosszát. }
      if(s[i]='a') then inc(t);
{Ha a string alktuálisan vizsgált karaktere a betű, akkor növeljük a találat számát. }
         szamola:=t;
{Visszaadjuk a találatszámot. }
   end;

function szamole(s:string):integer;
{A szamole függvény megszámolja az e betűk számát és egészként visszadja azt. }
var i,t:integer;
begin
   t:=0;
   for i:=1 to length(s) do
      if(s[i]='e') then inc(t);
         szamole:=t;
{Visszaadjuk a találatszámot. }
   end;

BEGIN
   clrscr;
   assign(f,'proba.txt');
{Az f változóhoz hozzárendeljük a proba.txt-t. }
   reset(f);
{Állomány megnyitása olvasásra. }
   while(not eof(f)) do begin
{Addig él a ciklus, amíg a fájlnak nincs vége. }
      readln(f,s);
{Beolvasunk az f-ből egy sort s-be. }
      writeln(s);
```

```
      writeln('Az "a"-k szama: ',szamola(s));
```
{A kiírásba meghívjuk a szamola(s) függvényt, ami megszámolja a sorban lévő a betűket és egész számként visszaadja az értéket }
```
      writeln(' Az "e"-k szama: ',szamole(s));
      end;
   close(f);
```
{Bezárjuk az olvasásra megnyitott fájlt. }
```
   readkey;
END.
```



Szövegfájl (szoveg.txt) létrehozása és feltöltése 100 darab nullákat tartalmazó sorral (Ha a fájl létezik, akkor a tartalma kitörlődik) (Feltolto.java)

```
import java.io.*;
public class Feltolto{
   public static String szaz0(){
```
/ A függvény String típusú változót ad vissza. */*
```
      int i;
      StringBuffer sb=new StringBuffer();
```
/ Létrehozunk egy sb nevű StringBuffert. A StringBuffer egy változtatható hosszúságú String, melyet ezért vághatunk, bővíthetünk. */*
```
      for (i=0; i<100;i++){
```
/ A ciklus segítségével 100-szor kiegészítjük az sb-t egy darab 0-val. */*
```
         sb.append('0');
         }
      return sb.toString();
```

```
/* Mivel a függvény String típusú változót ad vissza, ezért a StringBuffer
osztály .toString() függvényével String típusúvá alakítjuk a StringBuffert. */
        }

   public static void main (String[] args) throws
Exception{
      PrintStream f=new PrintStream( new
FileOutputStream("szoveg.txt"));
/* A fájlba íráshoz PrintStream csatornát hozunk létre, melyet f-nek nevezünk
el. Beleágyazzuk a FileOutputStream-t, ami a fájlkimeneti csatorna és ezzel
a szoveg.txt-t azonosítjuk. */
      for (int i=0;i<100;i++){
         f.println(szaz0());
/* 100-szor meghívjuk a szaz0() függvényt, ami 100 db 0-t ad vissza és ezt
kiírjuk az f-be, azaz a fájlba. */
         }
      f.close();
/* Lezárjuk a csatornát. */
      }
   }
```

Szövegfájl (szoveg.txt) létrehozása és feltöltése 100 darab nullákat tartalmazó sorral (Ha a fájl létezik, akkor a tartalma megmarad) (FeltoltoV1.java)

```
import java.io.*;
public class FeltoltoV1{
   public static String szaz0(){
      int i;
      StringBuffer sb=new StringBuffer();
      for (i=0; i<100;i++){
         sb.append('0');
         }
      return sb.toString();
      }

   public static void main (String[] args) throws
Exception{
      PrintStream f=new PrintStream( new
FileOutputStream("szoveg.txt",true));
/* A true arra vonatkozik, hogy ha a fájl létezik, akkor a benne lévő tartalom
megmarad és a program csak hozzáír a fájl tartalmához.
Alapértelmezettként, ha nem írjuk ki a true-t, akkor a fájl tartalma elvész. */
      for (int i=0;i<100;i++){
         f.println(szaz0());
         }
      f.close();
```

```
      }
   }
```

Szövegfájl (szoveg.txt) létrehozása és feltöltése 100 darab nullákat tartalmazó sorral (Ha a fájl létezik, akkor a tartalma kitörlődik) (Feltolto.pas)

```
program Feltolto;
uses crt;
var f:text;
    i:integer;

function szaz0:string;
{ A függvény String típusú változót ad vissza. }
var i:integer;
    s:string;
begin
   s:='';
   for i:=1 to 100 do
      s:=s+'0';
{Mint látható, a string kiegészítése itt nem jelent problémát, csak hozzá kell adni az új karaktert. }
      szaz0:=s;
   end;

BEGIN
   assign(f,'szoveg.txt');
{Az f változóhoz hozzárendeljük a szoveg.txt-t. }
   rewrite(f);
{Állomány megnyitása írásra, a tartalom törlődésével. }
   for i:=1 to 100 do begin
      writeln(f,szaz0);
{100-szor meghívjuk a szaz0() függvényt, ami 100 db 0-t ad vissza és ezt kiírjuk az f-be, azaz a fájlba. }
      end;
   close(f);
{Bezárjuk az írásra megnyitott fájlt. }
END.
```

Szövegfájl (szoveg.txt) létrehozása és feltöltése 100 darab nullákat tartalmazó sorral (Ha a fájl létezik, akkor a tartalma megmarad) (FeltoltoV1.pas)

```
program FeltoltoV1;
uses crt;
var f:text;
    i:integer;

function szaz0:string;
var i:integer;
    s:string;
begin
   s:='';
   for i:=1 to 100 do
      s:=s+'0';
      szaz0:=s;
   end;

BEGIN
   assign(f,'szoveg.txt');
   append(f);
{Állomány megnyitása írásra, a tartalom megtartásával. }
   for i:=1 to 100 do begin
      writeln(f,szaz0);
      end;
   close(f);
END.
```

Szövegfájl (szoveg.txt) másolása egy új szövegfálba (ujszoveg.txt) (Masolo.java)

```
import java.io.*;
public class Masolo{
   public static void main (String[] args) throws
Exception{
      LineNumberReader f=new LineNumberReader( new
InputStreamReader(
                        new
FileInputStream("szoveg.txt")));
/* Létrehozunk egy beolvasást szolgáló f nevű csatornát, melyhez
hozzárendeljük a szoveg.txt. */
```

```
      PrintStream g=new PrintStream( new
FileOutputStream("ujszoveg.txt"));
/*  Létrehozunk egy fájlba írást szolgáló g nevű csatornát, melyhez
hozzárendeljük az ujszoveg.txt. */
      String s=f.readLine();
/* Beolvasunk egy sort az f-ként azonosított fájlból és eltároljuk a String
típusú s nevű változóba. */
      while(s!=null){
         g.println(s);
/* Amíg a ciklus tart, minden beolvasott sort kiírunk a g-ként azonosított fájl
egy sorába. */
         s=f.readLine();
/* Újra beolvasunk egy sort az f-ként azonosított fájlból. */
         }
      f.close();
/* Lezárjuk a megnyitott csatornákat. */
      g.close();
      }
   }
```

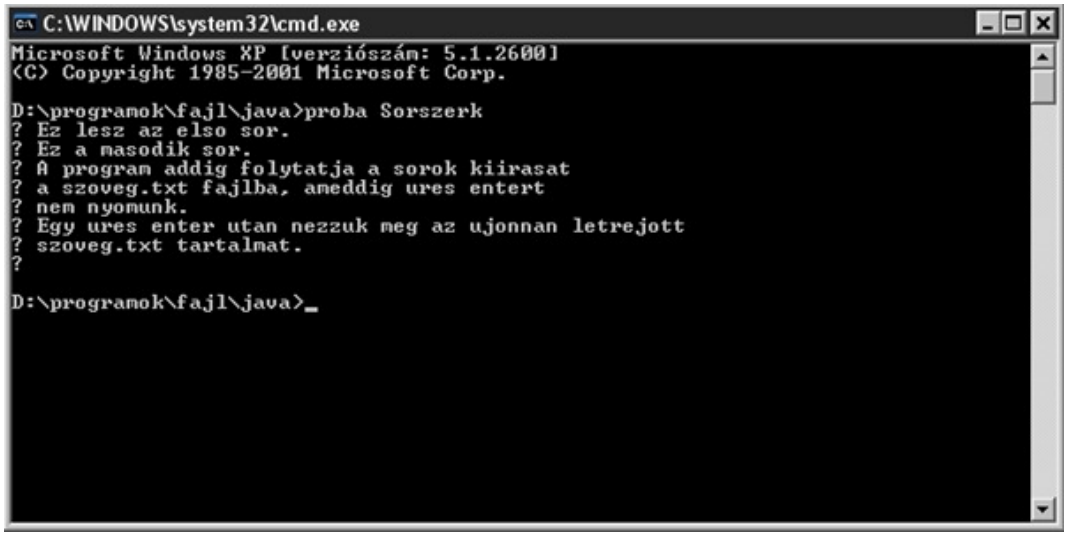
Szövegfájl (szoveg.txt) másolása egy új szövegfálba (ujszoveg.txt) (Masolo.pas)

```
program Masolo;
uses crt;
var f,g:text;
    s:string;
BEGIN
   writeln('A masolas kezdete...');
   assign(f,'szoveg.txt');
{Az f változóhoz hozzárendeljük a szoveg.txt-t. }
   assign(g,'ujszoveg.txt');
{A g változóhoz hozzárendeljük az ujszoveg.txt-t. }
   reset(f);
   rewrite(g);
   while (not eof(f)) do begin
      readln(f,s);
{Kiolvasunk egy sort az f-ként azonosított fájlból és eltároljuk az s nevű string
típusú változóba. }
      writeln(g,s);
{Kiírjuk az előbb beolvasott sort a g-ként azonosított fájl egy sorába. }
      end;
   close(f);
{Bezárjuk az olvasásra és írásra megnyitott fájlokat }
```

```
   close(g);
END.
```

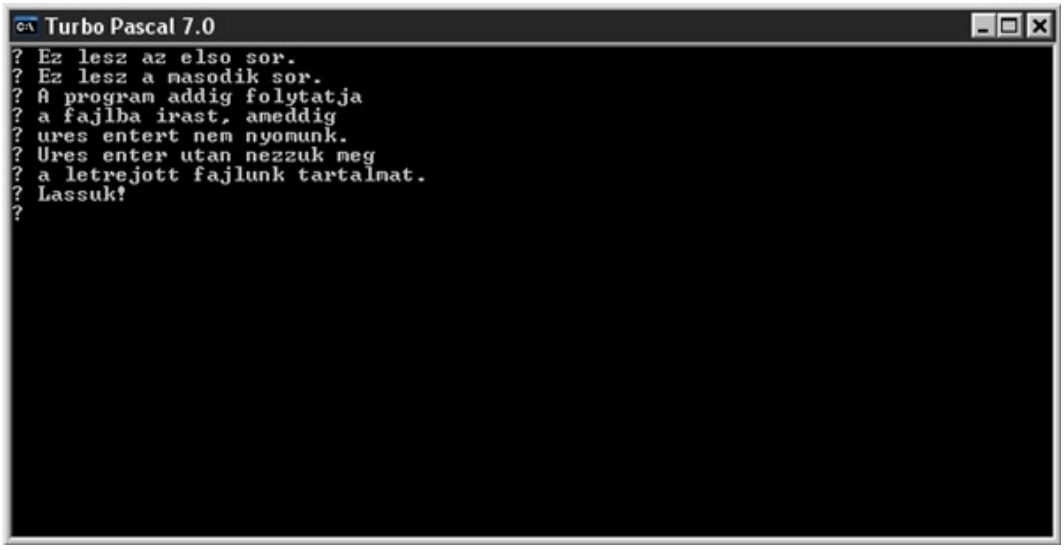
Billentyűzetről bevitt sorok írása fájlba (szoveg.txt) (Sorszerk.java)

```
import java.io.*;
public class Sorszerk{
   public static String in() throws Exception{
/* A String visszatérési értékű in nevű függvény a billentyűzetről beolvasott
egy sort adja vissza. */
      LineNumberReader s=new LineNumberReader(new
InputStreamReader(System.in));
      return s.readLine();
      }
   public static void main(String[] args) throws
Exception{
      PrintStream out=new PrintStream(new
FileOutputStream("szoveg.txt"));
      String s;
      do{
         System.out.print("? ");
/* A sor elejére kiírunk egy kérdőjelet. */
         s=in();
/* Meghívjuk az in() függvényt, mely visszatérési értékét eltároljuk az s nevű
String típusú változóba. */
         if(s.length()==0) break;
/* Ha a bevitt sor üres, azaz üres entert tartalmaz, akkor vége a ciklusnak. */
         out.println(s);
/* Kiírja a bevitt sort a fájlba. */
         }while(true);
/* A ciklus mindig igaz, végtelen ciklusnak számítana, ha nem lenne a
cikluson belül egy kilépési pont. */
      out.close();
      }
   }
```



Billentyűzetről bevitt sorok írása fájlba (szoveg.txt) (Sorszerk.pas)

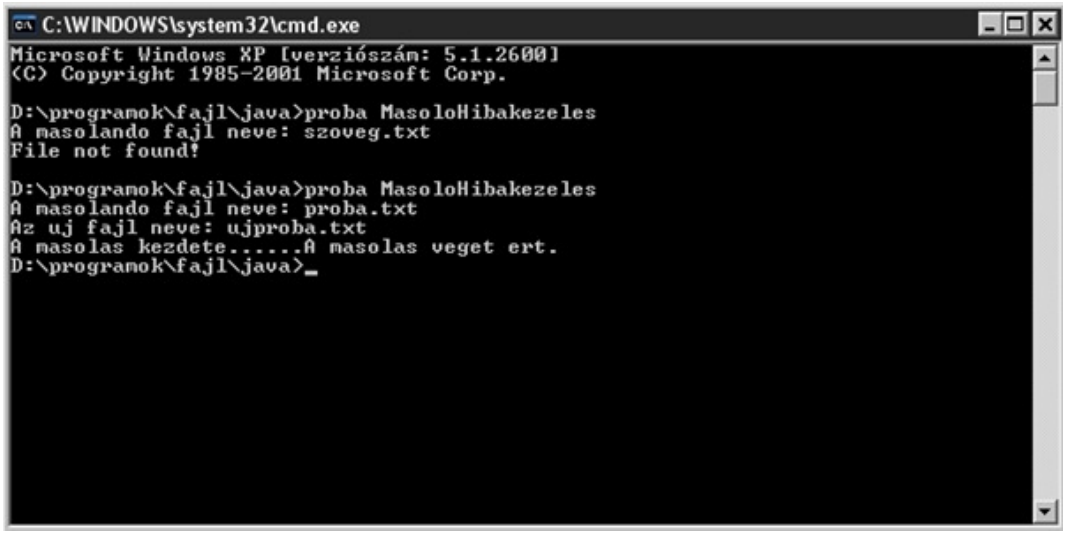
```
program Sorszerk;
uses crt;
var f:text;
    s:string;
BEGIN
   clrscr;
   assign (f,'szoveg.txt');
   rewrite (f);
   s:='*';
{Azért rakunk bele egy *-ot, hogy legyen értéke, ne legyen üres. }
   while (length(s)>0) do begin
{Addig tart a ciklus, amíg az s nevű string értéke nem nulla.}
   write('? ');
{A sor elejére kiírunk egy kérdőjelet. }
   readln(s);
{Beolvassuk billentyűzetről az adatot. }
   if(length(s)=0) then break;
{Ha a bevitt sor üres, azaz üres entert tartalmaz, akkor vége a ciklusnak. }
      writeln(f,s);
{Kiírja a bevitt sort a fájlba. }
      end;
   close(f);
END.
```



Szövegfájl másolása egy új szövegfájlba úgy, hogy mi adjuk meg a programban a fájlok neveit és lekezeljük a lehetséges hibát is (MasoloHibakezeles.java)

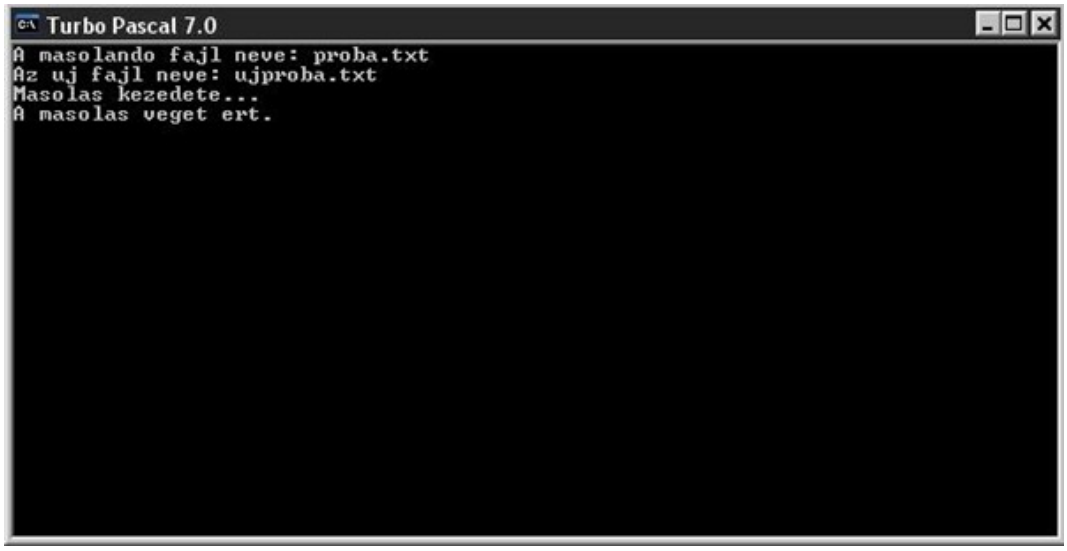
```
import java.io.*;

public class MasoloHibakezeles{
   public static String s;
   public static String in() throws Exception{
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      s=x.readLine();
      return s;
      }

   public static void main (String[] args) throws
Exception{
      String nev, ujnev;
      System.out.print("A masolando fajl neve: ");
      nev=in();
/* Bekérjük a másolandó fájl nevét. */
      File fajl=new File(nev);
/* Létrehozunk egy File típusú objektumot, mely a beolvasott fájlt fogja azonosítani. */
      if(!fajl.exists()){
/* Az .exists() függvény boolean, azaz igaz vagy hamis értéket ad vissza. Mivel ott van a tagadás, ezért ha nem létezik a fájl, akkor a program hibát ír ki és System.exit(1);-el kilép. */
         System.out.println("File not found!");
```

```
            System.exit(1);
            }

         System.out.print("Az uj fajl neve: ");
         ujnev=in();
/* Bekérjük az új fájl nevét, amibe másolni szeretnénk. */
         File ujfajl=new File(ujnev);

         System.out.print("A masolas kezdete...");
         System.out.print("...");

         LineNumberReader f=new LineNumberReader( new
InputStreamReader(
                              new FileInputStream(fajl)));
/* Figyeljük meg, hogy most már a csatornák nyitásakor változóként
hivatkozunk a fájlokra. */
         PrintStream g=new PrintStream( new
FileOutputStream(ujfajl));
         String s=f.readLine();
         while(s!=null){
            g.println(s);
            s=f.readLine();
            }
         System.out.print("A masolas veget ert.");
         f.close();
         g.close();
         }
   }
```



Szövegfájl másolása egy új szövegfájlba úgy, hogy mi adjuk meg a programban a fájlok neveit és lekezeljük a lehetséges hibát is (MasoloHibakezeles.pas)

```
program MasoloHibakezeles;
uses crt;
var f,g:text;
    s1,s2,s:string;
BEGIN
   clrscr;
   write('A masolando fajl neve: ');
   readln(s1);
{Bekérjük a másolandó fájl nevét. }
   {$I-} assign(f,s1); {$I+}
{f-hez hozzárendeljük a bekért fájlnevet, de mivel ez hibához vezethet akkor, ha a megadott fájl nem létezik, lekezeljük a lehetséges hibát, azaz {$I-} és {$I+} közé írjuk be azt az utasítást, ami hibához vezethet. }
   if (IOresult>0) then begin
{Az IOresult, vagyis InputOutputresult akkor ad vissza 0-nál nagyobb értéket, ha valamilyen hiba bekövetkezik, jelen esetben, ha a fájl nem található. }
      writeln('File not found!');
{Ha a hiba fellép, kiírjuk, hogy a fájl nem található és halt;-tal kilépünk a programból. }
      halt;
      end;
   write('Az uj fajl neve: ');
   readln(s2);
{Bekérjük az új fájl nevét, amibe másolni szeretnénk, ennek a fájlnak nem kell létezőnek lenni, mivel a program létrehozza nekünk. }
   assign(g,s2);
{g-hez hozzárendeljük az új fájlt, majd mint a korábbi másoló programnál, elvégezzük a másolást. }
   reset(f);
   rewrite(g);
   writeln('Masolas kezedete...');
   while (not eof(f)) do begin
      readln(f,s);
      writeln(g,s);
      end;
   writeln('A masolas veget ert.');
   close(f);
   close(g);
   readkey;
END.
```



Szövegfájlból (eredeti.txt) beolvasott sorok kiírása egy új fájlba (rendes.txt) rendezetten (RendezoMasolo.java)

```
import java.io.*;

public class RendezoMasolo{
   public static int n=100;
   public static String[] sz=new String[n];

   public static void rendez(){
      int i,j;
      String x;
      i=0;
      while (i+1<n){
/* Végigmegyünk az sz tömb elemein és elrendezzük azokat. */
         j=i+1;
         while(j<n){
            if (sz[j].compareTo(sz[i])<0){
/* String-eket a .compareTo() függvénnyel hasonlítunk össze, amely 3 különböző értéket adhat vissza. Pozitív értéket ad vissza, ha sz[i] nagyobb, mint sz[j], 0-t, ha egyenlők és negatívat, ha sz[j] nagyobb, mint sz[i]. */
               x=sz[j];
               sz[j]=sz[i];
               sz[i]=x;
               }
           j=j+1;
           }
        i=i+1;
        }
```

```
      }

   public static void main(String[] args) throws
Exception{
      LineNumberReader in=new LineNumberReader(new
InputStreamReader(
                           new
FileInputStream("eredeti.txt")));
      PrintStream out=new PrintStream(new
FileOutputStream("rendes.txt"));

      int i=-1;
      String s=in.readLine();
      while(s!=null){
         i++;
         sz[i]=s;
/* Az sz tömbbe tároljuk el a beolvasott sorokat. */
         s=in.readLine();
         }
      n=i;
      rendez();
/* Meghívjuk a rendezést. */
      for (i=0; i<n; i++){
/* Kiírjuk a fájlba a már rendezett sorokat. */
         out.println(sz[i]);
         }
      out.close();
      in.close();
      }
   }
```

Szövegfájlból (eredeti.txt) beolvasott sorok kiírása egy új fájlba (rendes.txt) rendezetten (RendezoMasolo.pas)

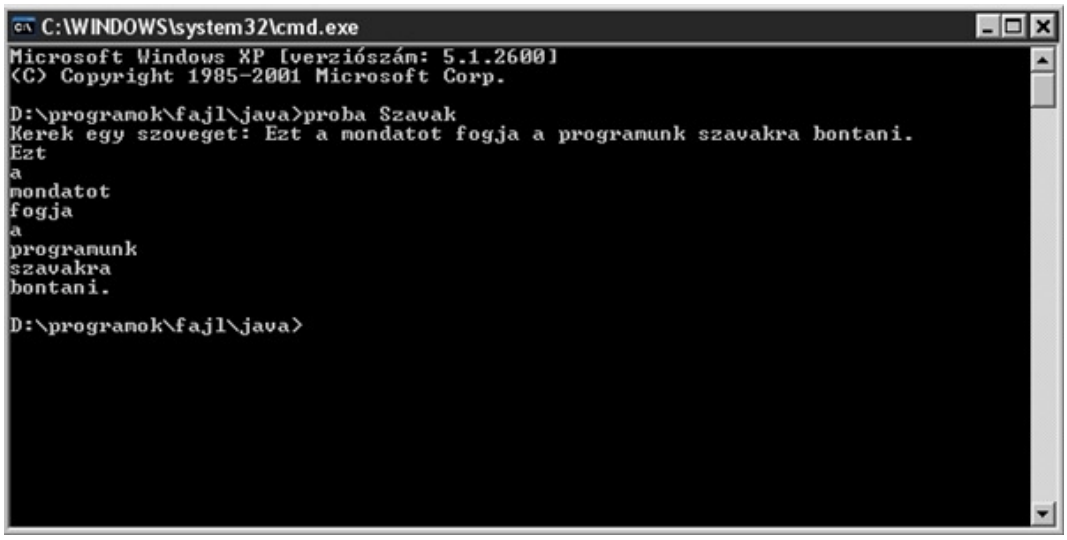
```
program RendezoMasolo;
uses crt;
var i,j,n:integer;
    f,g:text;
    s:string;
    sorok:array [1..100] of string;

procedure cserel(k,j:integer);
var x:string;
begin
   x:=sorok[k];
```

```
   sorok[k]:=sorok[j];
   sorok[j]:=x;
   end;

BEGIN
   clrscr;
   assign (f,'eredeti.txt');
   assign (g,'rendes.txt');
   reset(f);
   rewrite(g);
   i:=1;
   while (not eof(f)) do begin
      readln(f,s);
      sorok[i]:=s;
{A sorok tömbbe tároljuk el a beolvasott sorokat. }
      i:=i+1;
      end;
   n:=i-1;
{Az n változót a sorok száma szerint állítjuk be. Azért -1, mivel az i-hez az
előző ciklus végén hozzáadódik 1. }
   for i:=1 to n-1 do
      for j:=i+1 to n do
{Végigmegyünk a sorok tömb elemein és elrendezzük azokat. }
         if (sorok[i]>sorok[j]) then
            cserel(i,j);

   for i:=1 to n do
{Kiírjuk a fájlba a már rendezett sorokat. }
      writeln(g,sorok[i]);

   close(f);
   close(g);
END.
```

Billentyűzetről beolvasott sor szavakra bontása (Szavak.java)

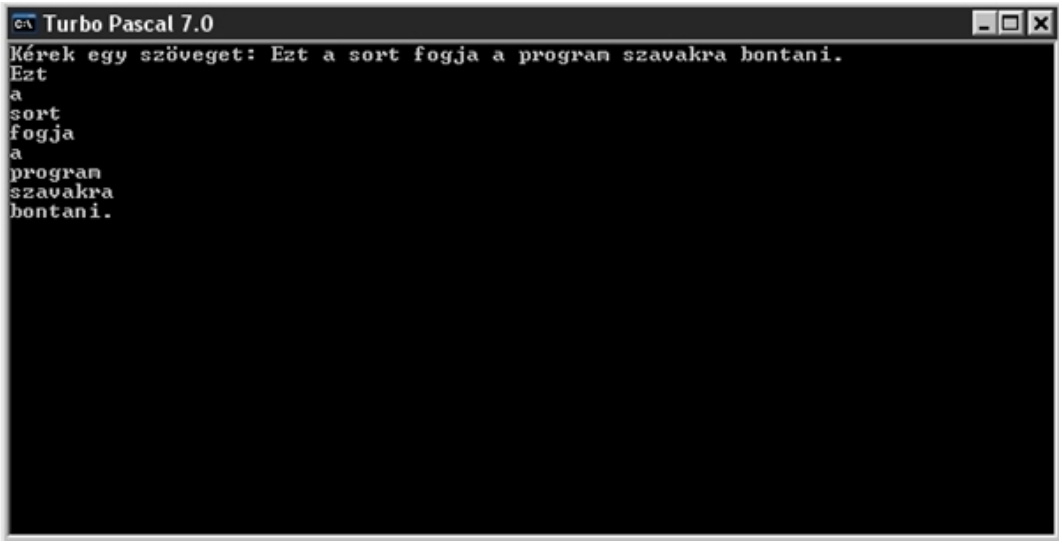
```
import java.io.*;

public class Szavak{
   public static String in() throws Exception{
      LineNumberReader x=new LineNumberReader(new
InputStreamReader(System.in));
      String s=x.readLine();
      return s;
      }
```

```
   public static void main(String[] args) throws
Exception{
       String s;
       int i1,i2;
       System.out.print("Kerek egy szoveget: ");
       s=in()+" *";
/* A bekért szöveget berakjuk egy String típusú s változóba úgy, hogy a
végére egy szóköz után *-ot rakunk. */
       i1=0;
/* Beállítjuk az első karakterpozícióra az i1-et. */
       i2=s.indexOf(' ');
/* Beállítjuk az s változóban található szöveg első szóközének pozícióját. Az
.indexOf() függvény -1-et ad vissza, ha nem talál szóközt. */
       while(i2>=0){
/* Amíg van szóköz, addig működik a ciklus. */
           if (i2>i1)
/* Hogyha talált egy szóközt és az nem egyezik meg azzal a pozícióval,
amihez képest vizsgálunk, akkor kiírja az s változóban az első szóközig tartó
karaktersorozatot, azaz egy szót. */
               System.out.println(s.substring(i1,i2));
           i1=i2+1;
/* Az i1-et beállítjuk a szóköz utáni első pozícióra. */
           i2=s.indexOf(' ',i1);
/* i2-be beállítjuk az i1-től számított első szóköz pozícióját. */
           }
       }
   }
```



Billentyűzetről beolvasott sor szavakra bontása (Szavak.pas)

```
program Szavak;
uses crt;
var s,sb:string;
    i2:integer;
BEGIN
   clrscr;
   write('Kerek egy szoveget: ');
   readln(s);
{Bekérünk egy szöveget. }
   sb:=s+' *';
{A bekért szöveget berakjuk a string típusú sb változóba úgy, hogy a végére
egy szóköz után *-ot rakunk. }
   i2:=pos(' ',sb);
{Az i2 változóba beállítjuk az sb változóban lévő első szóköz pozícióját. A
pos() függvény 0-t ad vissza, ha nem talál szóközt. }
   while (i2>0) do begin
{Amíg van szóköz, addig működik a ciklus. }
      if(i2>1) then begin
{Hogyha talált egy szóközt és ez nem az első helyen van, akkor kimásolja az
sb változóból az első pozíciótól a szóköz előtti karakterig tartó
karaktersorozatot, azaz egy szót és ezt kiírja a képernyőre. }
         writeln (copy(sb,1,i2-1));
         end;
      sb:=copy(sb,i2+1,255);
{Az sb-t úgy módosítjuk, hogy a már kiírt szót ne tartalmazza. }
      i2:=pos(' ',sb);
{i2-be újra beletesszük az sb-ben található első szóköz pozícióját. }
      end;
END.
```



Szövegfájlból (szoveg.txt) indexfájl (index.txt) létrehozása (Indexelo.java)

```
import java.io.*;

public class Indexelo{
   public static String[] sor=new String[100];
   public static String[] szo=new String[500];
   public static int N;

   public static void rendez(String[] sz, int N){
/* Elrendezzük a paraméterként megkapott tömb elemeit. */
      int i,j;
      String x;
      i=0;
      while (i+1<N){
         j=i+1;
         while(j<N){
            if (sz[j].compareTo(sz[i])<0){
               x=sz[j];
               sz[j]=sz[i];
               sz[i]=x;
               }
            j=j+1;
            }
         i=i+1;
         }
      }

   public static void main(String[] args) throws
Exception{
      LineNumberReader in=new LineNumberReader(new
InputStreamReader(
               new FileInputStream("szoveg.txt")));
      PrintStream out=new PrintStream(new
FileOutputStream("index.txt"));

      int i=-1;
      String s=in.readLine();
      while(s!=null){
/* A sor tömbbe eltároljuk a szoveg.txt sorait. */
         if (i==100) break;
         i++;
         sor[i]=s;
         s=in.readLine();
```

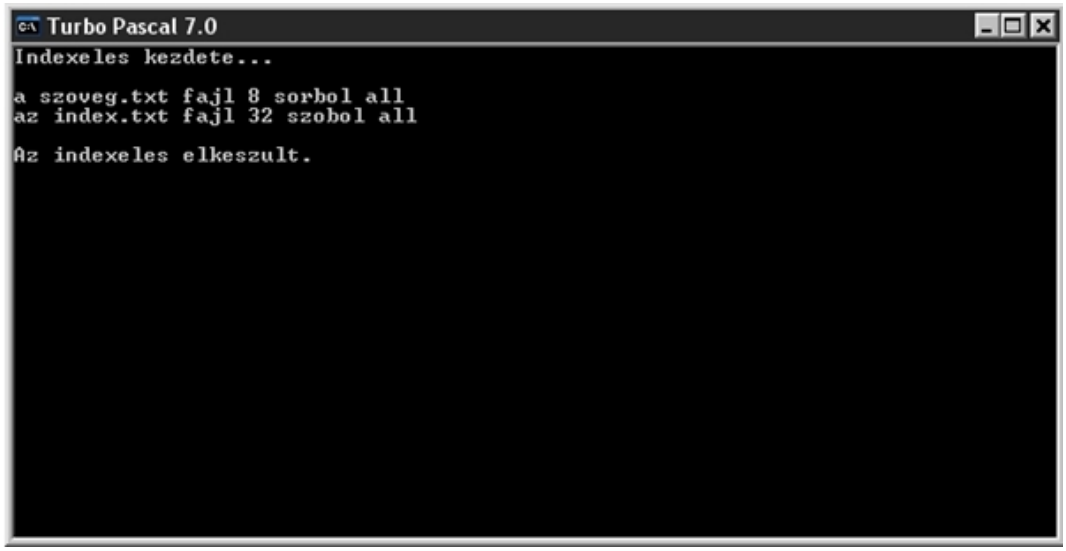
```
            }
         in.close();
/* Használat után bezárjuk az in csatornát. */
         N=i;
/* Beállítjuk az N változóba a sorok számát. */
         int j=-1;
         for (i=0; i<N; i++){
/* A sorok tömb elemeit szavakra bontjuk a korábban bemutatott program
elvének használatával. */
            int i1=0;
            int i2=sor[i].indexOf(' ');
            while(i2>=0){
               if((i2>i1) && (j<500)){
                  j++;
                  szo[j]=sor[i].substring(i1,i2);
                  }
               i1=i2+1;
               i2=sor[i].indexOf(' ',i1);
               }
            if(i1<sor[i].length())
szo[++j]=sor[i].substring(i1);
/* Ha a .substring() függvénynek 1 db bemenő paramétere van, akkor az
azután levő összes karaktert visszaadja. */
            }
         rendez(szo,j);
/* Elrendezzük a szo tömb elemeit, az ezt követő ciklussal beleírjuk az
elrendezett tömb elemeit az index.txt fájlba. */
         for(i=0;i<j;i++){
            out.println(szo[i]);
            }
         out.close();
         }
   }
```

Szövegfájlból (szoveg.txt) indexfájl (index.txt) létrehozása kicsit bonyolultabban, mint Java-ban, mivel itt figyelembe vesszük az írásjeleket és azok nem kerülnek be az indexfájlba (Indexelo.pas)

```
program Indexelo;
uses crt;
var i,i1,i2j,r,n:integer;
    f,g:text;
    s,s1:string;
    sorok:array [1..50] of string;
    szavak:array [1..100] of string;
```

```
procedure cserelgetes (i,j:integer);
var x:string;
begin
   x:=szavak[i];
   szavak[i]:=szavak[j];
   szavak[j]:=x;
   end;

procedure rendezes;
{Elrendezzük a szavak tömb elemeit. }
{a szavak n elemu tomb elrendezese}
var i,j:integer;
begin
   for i:=1 to r-1 do
      for j:=i+1 to r do
         if (szavak[j]<szavak[i]) then cserelgetes(j,i);
   end;

function postit(s:string):integer;
var i,p:integer;
begin
   p:=0;
   for i:=1 to length(s) do
{Végigmegy a string karakterein, megvizsgálja azokat és amennyiben talál egy írásjelet, akkor visszaadja annak a pozícióját. }
      case s[i] of
         'a'..'z','A'..'Z':continue;
          #128..#255:continue;
      else begin
         p:=i;
         break;
         end;
      end;
      postit:=p;
   end;

BEGIN
   clrscr;
   assign(f,'szoveg.txt');reset(f);
   assign(g,'index.txt');rewrite(g);
   i:=0;
   while (not eof(f)) do begin
{Eltároljuk a sorok tömbbe a szoveg.txt sorait. }
      if (i=51) then break;
{A fent létrehozott tömb 50 elemű, ezért megszakítjuk a ciklust, ha túlcsordulás történik. }
      readln(f,s);
      inc(i);
```

```
      sorok[i]:=s;
      end;
   close(f);
   writeln('a szoveg.txt fajl ',i,' sorbol all');
   r:=i; {sorok szama}
```
{Az r változóba beletesszük a sorok számát. }
```
   j:=0;
   for i:=1 to r do begin
```
{A sorok tömb elemeit szavakra bontjuk a korábban bemutatott program elvének használatával. }
```
      s:=sorok[i]+' xxx';
      i2:=postit(s);
      while (i2>0) do begin
         if ((i2>1) and (j<1000)) then begin
            inc(j);
            str(i,s1);
```
{Az i értékét string-gé konvertáljuk. }
```
            szavak[j]:=copy(s,1,i2-1)+' '+s1;
            end;
         s:=copy(s,i2+1,255);
         i2:=postit(s);
         end;
      end;
   writeln('az index.txt fajl ',j,' szobol all');
   r:=j;
   rendezes;
```
{Elrendezzük a szavak tömb elemeit, az ezt követő ciklussal beleírjuk az elrendezett tömb elemeit az index.txt fájlba. }
```
   for i:=1 to r do writeln(g,szavak[i]);
   close(g);
   readkey;
END.
```

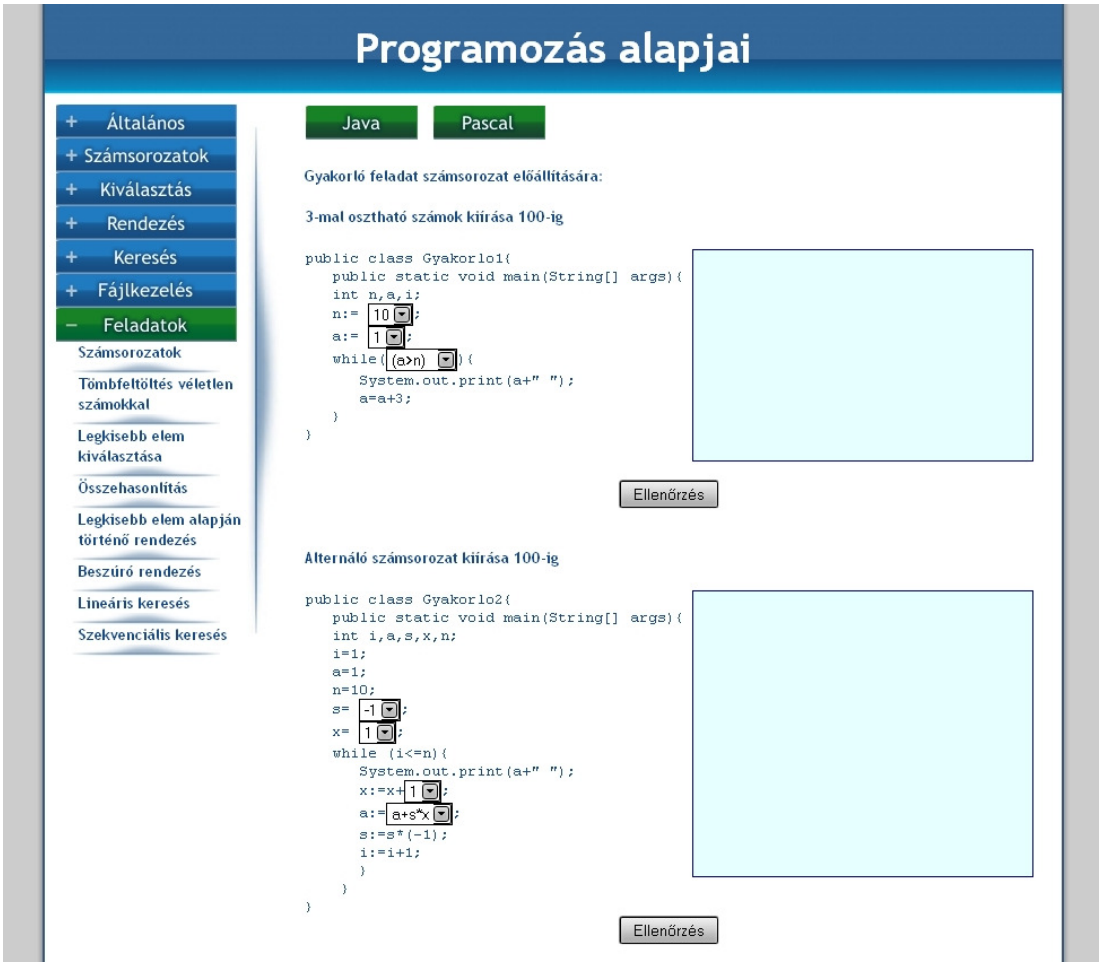


2.2.6 Feladatok

A programok gyakorlására külön részt állítottunk össze, melyek elkészítésénél űrlapot és JavaScript programozást használtunk fel. A felhasználó számára előre megírt programok állnak rendelkezésre, melybe űrlapelemek (egysoros szövegbeviteli mező vagy legördülő menü) segítségével adhatja meg a hiányzó adatokat. Az „Ellenőrzés” gomb lenyomásakor meghívódik a JavaScript függvény, mely a kapott eredményt a program mellett jobb oldalt található, világoskék hátterű **`div`** tartalmaként jeleníti meg. A div mérete tartalom mennyiségének függvényében változik. Természetesen a gyakorló feladatokat Java és Pascal nyelven is elkészítettük, mely ugyanúgy, mint a programok esetében, a fentebb lévő két gomb segítségével érhető el.

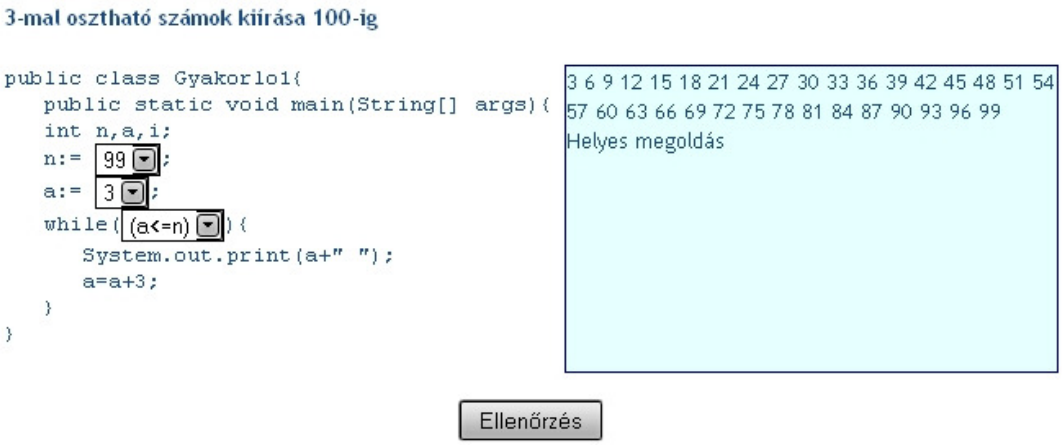
Mivel a JavaScript nyelv különbözik a Java-tól és a Pascal-tól is, a kódoláskor kiemelt figyelmet fordítottunk az ebből adódó nehézségek kiküszöbölésére, így a felhasználó számára úgy tűnhet, mintha az eredeti nyelven írt program futna a weblapon. A programokban előforduló kritikusabb hibalehetőségeket (rossz kezdőérték megadása, végtelen ciklus létrehozása) úgy kezeltük le, hogy a rossz kezdőérték, üresen hagyott mező vagy hibás feltételek megadásakor az eljárás hibát jelez a felhasználó számára. A JavaScript technológia nem ad lehetőséget fájlkezelésre, ezért ezt a programcsoportot mellőztük a feladatok menüpontból, azonban minden más típusra készítettünk gyakorló feladatot.

A következőkben a feladatokról készített képernyőképek láthatók.

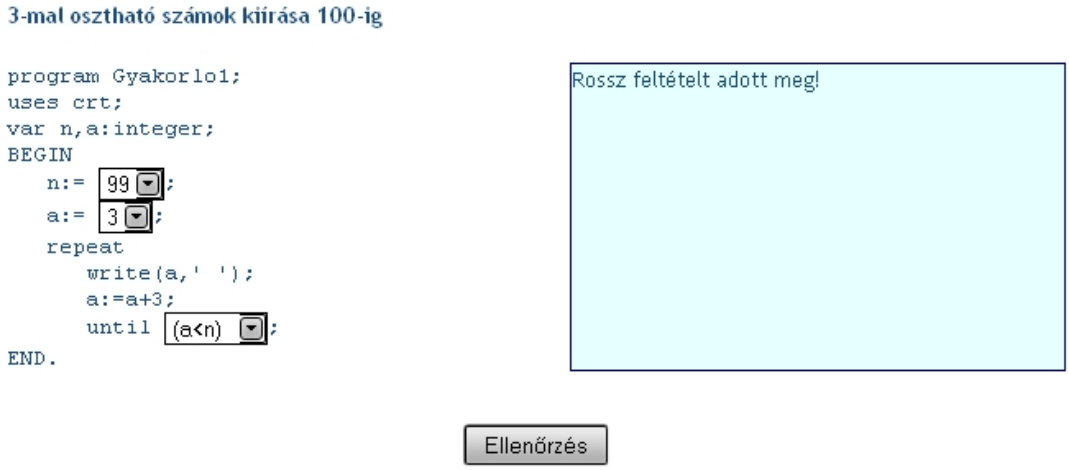


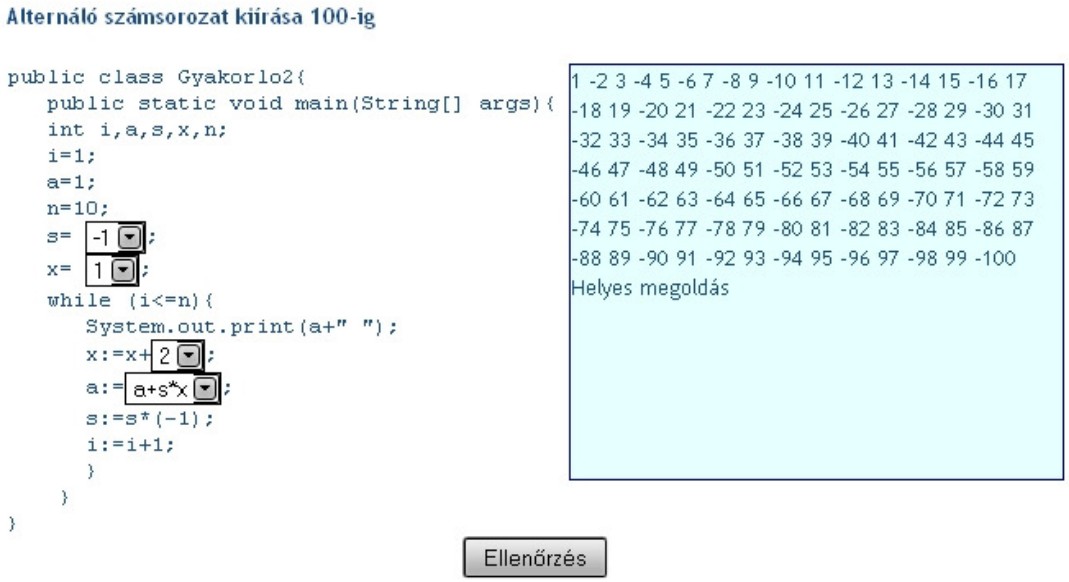
2.2.6.1. Gyakorló feladatok számsorozatok előállítására

Amennyiben a felhasználó helyesen tölti ki az űrlapot, akkor a program ezt jelzi számára



Ebben a példában azt láthatjuk, hogy rossz feltétel megadásakor a felhasználó tájékoztatást kap a hibájáról.





Alternáló számsorozat kiírása 100-ig

```
program Gyakorlo2;
uses crt;
var i,a,s,x,n:integer;
BEGIN
   i:= 1;
   a:= 1;
   s:= -1;
   x:= 1;
   while (i<=n) do begin
      write(a,' ');
      x:=x+2;
      a:= a+s*x;
      s:=s*(-1);
      i:=i+1;
      end;
END.
```

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

Ellenőrzés

Számok faktoriálisának kiírása (10 db szám)

```
public class Gyakorlo3{
   public static void main(String[] args){
   int i,a,n;
   i:= 1;
   a:= 1;
   n:= 10;
   while (i<=n){
      System.out.print(a+" ");
      a:= a*(i+1);
      i:=i+1;
      }
   }
}
```

```
1 2 6 24 120 720 5040 40320 362880 3628800
Helyes megoldás
```

Ellenőrzés

Számok faktoriálisának kiírása (10 db szám)

```
program Gyakorlo3;
uses crt;
var i,a,n:integer;
BEGIN
   i:= 1 ;
   a:= 1 ;
   n:= 10;
   while (i<=n) do begin
      write(a,' ');
      a:= a*(i+1) ;
      i:=i+ 1 ;
      end;
END.
```

Ellenőrzés

```
1 2 6 24 120 720 5040 40320 362880 3628800
Helyes megoldás
```

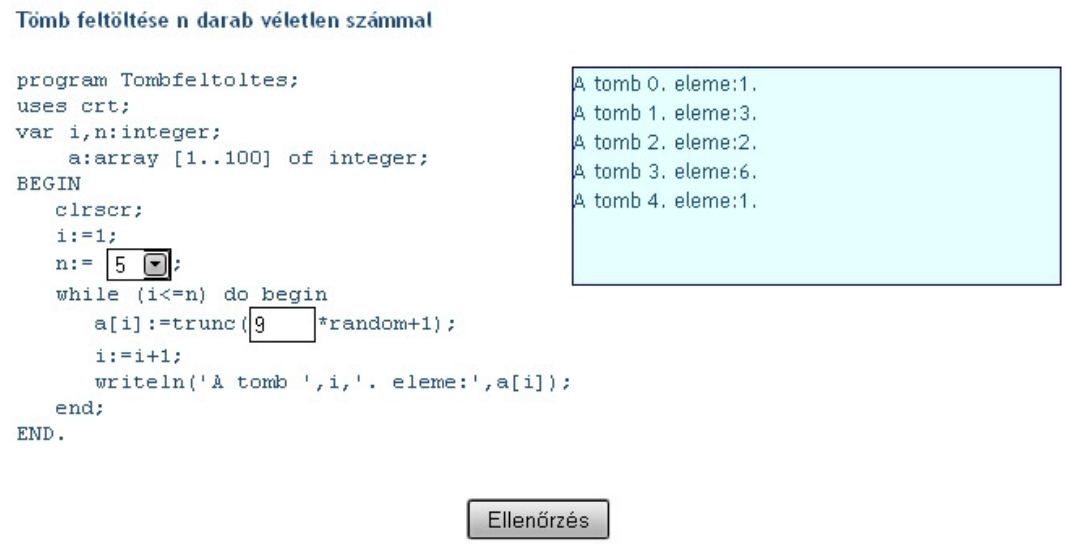
2.2.6.2. Gyakorló feladat tömbkezelésre

Tömb feltöltése n darab véletlen számmal

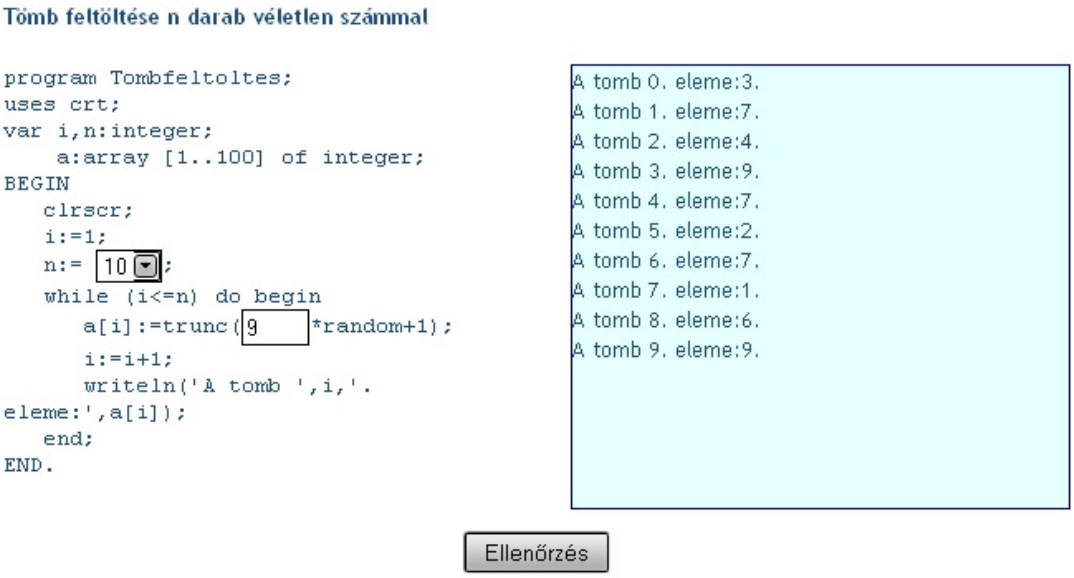
```
public class Tombfeltoltes;
public static void main String[] args){
   int i,n;
   int [] a= new int[100];
   i=1;
   n= 5 ;
   while (i<=n) {
      a[i]=(int)( 20  *Math.random()+1);
      i=i+1;
      System.out.println("A tomb "+i+". eleme:"+a[i]);
    }
}
```

Ellenőrzés

```
A tomb 0. eleme:7.
A tomb 1. eleme:14.
A tomb 2. eleme:13.
A tomb 3. eleme:18.
A tomb 4. eleme:11.
```



A fenti és a lenti példa összehasonlításakor láthatjuk az eredményt megjelenítő div méretének változását, a tartalomtól függően.



2.2.6.3. Gyakorló feladat kiválasztásra

Tömb feltöltése n darab véletlen számmal és legkisebb elem kiválasztása

```
program Minimum;
uses crt;
var i,n:integer;
a:array [1..100] of integer;
BEGIN
   clrscr;
   n:=5;
   i:=1;
   writeln('A tomb a rendezes elott');
   while (i<=n) do begin
      a[i]:=trunc(30*random+1);
      i:=i+1;
      writeln('A tomb ',i,'.
eleme:',a[i]);
   end;
   min:=a[1];
   i:=1;
   while (i<=n) do begin
      if (a[i]<min) then begin
      min:=a[i];
      end;
      i:=i+1;
   end;
   writeln('Legkisebb szam: ',min);
   end;
END.
```

Rossz kezdőérték!

Ellenőrzés

Tömb feltöltése n darab véletlen számmal és legkisebb elem kiválasztása

```
public class Kivalaszt{
   public static void main(String[] args){
   int n,i,min;
   int[] a=new int[100];
   n=10;
   i=0;
   System.out.println("A tomb a rendezes
elott");
   while (i<n){
     a[i]=(int)(50*Math.random()+1);
     i=i+1;
     System.out.println("A tomb "+i+".
eleme:"+a[i]);
     }
   min=a[0];
   i=1;
   while (i<n){
      if (a[i]<min) {
      min=a[i];
      }
      i=i+1;
   }
   System.out.println("Legkisebb szam: "+min);
   }
}
```

```
A tomb 0. eleme:27.
A tomb 1. eleme:2.
A tomb 2. eleme:39.
A tomb 3. eleme:58.
A tomb 4. eleme:28.
A tomb 5. eleme:3.
A tomb 6. eleme:89.
A tomb 7. eleme:50.
A tomb 8. eleme:38.
A tomb 9. eleme:54.

Legkisebb elem: 2
Helyes megoldás
```

Ellenőrzés

2.2.6.4. Gyakorló feladat összehasonlításra

Két véletlen szám létrehozása és összehasonlítása

```
program Hasonlit;
uses crt;
var a,b:integer;
BEGIN
   a:=trunc(30 *random+1);
   writeln('Elso szam:',a);
   b:=trunc( *random+1);
   writeln('Masodik szam:',b);
   if (a<b ) then begin
      writeln('Az elso szam (',a,')
nagyobb.');
      end
      else begin
      writeln('A masodik szam (',b,')
nagyobb.');
      end;
END.
```

Minden mezőt ki kell tölteni!

Ellenőrzés

Két véletlen szám létrehozása és összehasonlítása

```
public class Hasonlit{
   public static void main(String []args){
   int a,b;
   a=(int)(20 *Math.random()+1);
   System.out.println("Elso szam:"+a);
   b=(int)(20 *Math.random()+1);
   System.out.println("Masodik szam:"+b);
   if (a>b ) {
      System.out.println("Az elso szam
("+a+") nagyobb.");
      }
      else {
      System.out.println("A masodik szam
("+b+") nagyobb.");
      }
}
}
```

Elso szam: 18
Masodik szam: 8
Az elso szam (18) nagyobb.
Helyes megoldás!

Ellenőrzés

2.2.6.5. Gyakorló feladatok rendezésre

Tömb feltöltése n darab véletlen számmal és elrendezése beszúró rendezéssel

```
program Rend1;
uses crt;
var i,n,x,j:integer;
    a:array [1..100] of integer;
BEGIN
   clrscr;
   i:=1;
   n:=5;
   writeln('A tomb a rendezes elott');
   while (i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      writeln('A tomb ',i,'.
eleme:',a[i]);
   end;
   i:=2;
   while (i<=n)do begin
      j:=i;
      while (j>1) do begin
         if (a[j]<a[j-1]) then begin
             x:=a[j];
             a[j]:=a[j-1];
             a[j-1]:=x;
             end;
         else break;
         j:=j-1;
      end;
   i:=i+1;
   end;
   writeln('');
   writeln('Rendezés után');
   i:=1;
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme:',a[i]);
      i:=i+1;
      end;
END.
```

```
A tomb 1. eleme:57.
A tomb 2. eleme:25.
A tomb 3. eleme:19.
A tomb 4. eleme:23.
A tomb 5. eleme:9.

A tomb rendezes utan:

A tomb 1. eleme:9.
A tomb 2. eleme:19.
A tomb 3. eleme:23.
A tomb 4. eleme:25.
A tomb 5. eleme:57.

Helyes megoldás
```

Ellenőrzés

Tömb feltöltése n darab véletlen számmal és elrendezése legkisebb alapján történő rendezéssel

```
public class Rendezes{
   public static void main(String[] args){
   int n,i,j,x;
   int[] a=new int[10];
   n=10;
   i=0;
   System.out.println("A tomb a rendezes
elott");
   while (i<n){
     a[i]=(int)(90*Math.random()+1);
     i=i+1;
     System.out.println("A tomb "+i+".
eleme:"+a[i]);
     }
   i=0;
   while (i+1<n){
      j=i+1;
      while (j<n){
         if (a[j]<a[i]){
             x=a[j];
             a[j]=a[i];
             a[i]=x;
             }
         j=j+1;
      }
   i=i+1;
   }
   System.out.println("");
   System.out.println("Rendezes utan")
   i=0;
   while (i<n){
      System.out.println("A tomb "+i+". eleme:"+a[i]);
      i=i+1;
      }
}
}
```

```
A tomb 0. eleme:62.
A tomb 1. eleme:57.
A tomb 2. eleme:82.
A tomb 3. eleme:55.
A tomb 4. eleme:7.
A tomb 5. eleme:78.
A tomb 6. eleme:18.
A tomb 7. eleme:39.
A tomb 8. eleme:58.
A tomb 9. eleme:82.

A tomb rendezes utan:

A tomb 0. eleme:7.
A tomb 1. eleme:18.
A tomb 2. eleme:39.
A tomb 3. eleme:55.
A tomb 4. eleme:57.
A tomb 5. eleme:58.
A tomb 6. eleme:62.
A tomb 7. eleme:78.
A tomb 8. eleme:82.
A tomb 9. eleme:82.

Helyes megoldás
```

Ellenőrzés

Tömb feltöltése n darab véletlen számmal és elrendezése legkisebb alapján történő rendezéssel

```
program Rend1;
uses crt;
var i,n,x,j:integer;
    a:array [1..100] of integer;
BEGIN
   clrscr;
   i:=1;
   n:= 10 ;
   writeln('A tomb a rendezes elott');
   while (i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      writeln('A tomb ',i,'.
eleme:',a[i]);
   end;
   i:=1;
   while (i<n) do begin
      j:=i+1 ;
      while (j<=n) do begin
         if (a[j]<a[i]) then begin
             x:=a[j];
             a[j]:=a[i];
             a[i]:=x;
             end;
         j:=j+1 ;
      end;
   i:=i+1 ;
   end;
   writeln('');
   writeln('Rendezés után');
   i:=1;
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme:',a[i]);
      i:=i+1;
      end;
END.
```

```
A tomb 0. eleme:2.
A tomb 1. eleme:13.
A tomb 2. eleme:76.
A tomb 3. eleme:38.
A tomb 4. eleme:70.
A tomb 5. eleme:55.
A tomb 6. eleme:53.
A tomb 7. eleme:86.
A tomb 8. eleme:45.
A tomb 9. eleme:39.

A tomb rendezes utan:

A tomb 0. eleme:2.
A tomb 1. eleme:13.
A tomb 2. eleme:38.
A tomb 3. eleme:39.
A tomb 4. eleme:45.
A tomb 5. eleme:53.
A tomb 6. eleme:55.
A tomb 7. eleme:70.
A tomb 8. eleme:76.
A tomb 9. eleme:86.

Helyes megoldás
```

Ellenőrzés

Tömb feltöltése n darab véletlen számmal és elrendezése beszúró rendezéssel

```
public class Rendezes{
   public static void main(String[] args){
   int n,i,j,x;
   int[] a=new int[100];
   n=5;
   i=0;
   System.out.println("A tomb a rendezes
elott");
   while (i<n){
     a[i]=(int)(90*Math.random()+1);
     i=i+1;
     System.out.println("A tomb "+i+".
eleme:"+a[i]);
     }
      i=0;
      while (i<n){
      j=i;
      while (j>1){
         if (a[j]<a[j-1]){
             x=a[j];
             a[j]=a[j-1];
             a[j-1]=x;
             }
        else break;
        j=j-1;
      }
   i=i+1;
   }
   System.out.println("");
   System.out.println("Rendezes utan")
   i=0;
   while (i<n){
      System.out.println("A tomb "+i+". eleme:"+a[i]);
      i=i+1;
      }
}
```

Rossz kezdőérték!

Ellenőrzés

2.2.6.6. Gyakorló feladatok keresésre

Tömb feltöltése n darab véletlen számmal és meghatározott szám szekvenciális keresése a tömbben

```
program Kereses;
uses crt;
var i,n,x,j:integer;
    a:array [1..100] of integer;
BEGIN
   clrscr;
   i:=1;
   n:=5;
   writeln('A tomb elemei:');
   while (i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      writeln('A tomb ',i,'.
eleme:',a[i]);
   end;
   i:=1;
   while (i<n) do begin
      j:=i+1;
      while (j<=n) do begin
         if (a[j]<a[i])then begin
             x:=a[j];
             a[j]:=a[i];
             a[i]:=x;
             end;
         j:=j+1;
      end;
   i:=i+1;
   end;
   writeln('');
   writeln('Rendezés után');
   i:=1;
   while (i<=n) do begin
      writeln('A tomb ',i,'. eleme:',a[i]);
      i:=i+1;
      end;
   i:=1;
   t:=0;
   keresett:=11;
   writeln('Keresett szam:',keresett);
   while (i<=n) do begin
      if (a[i]>keresett) then break;
      if (a[i]=keresett) then begin
          t:=t+1;
          writeln('A tomb ',i,'. eleme es a keresett szam megegyezik');
          end;
      i:=i+1;
      end;
   writeln('A talalatok szama: ',t);
END.
```

```
A tomb 1. eleme:43.
A tomb 2. eleme:75.
A tomb 3. eleme:20.
A tomb 4. eleme:61.
A tomb 5. eleme:89.

Rendezes utan:

A tomb 1. eleme:20.
A tomb 2. eleme:43.
A tomb 3. eleme:61.
A tomb 4. eleme:75.
A tomb 5. eleme:89.

A keresett szam: 11
A talalatok szama: 0
Helyes megoldás
```

Tömb feltöltése n darab véletlen számmal és meghatározott szám lineáris keresése a tömbben

```
public class Kereses{
   public static void main(String[] args){
   int n,i,t,keresett;
   int[] a=new int[100];
   n=5;
   i=0;
   System.out.println("A tomb elemei:");
   while (i<n){
     a[i]=(int)(90*Math.random()+1);
     i=i+1;
     System.out.println("A tomb "+i+".
eleme:"+a[i]);
     }
   i=0;
   t=0;
   keresett=2;
   System.out.println("Keresett szam:"+keresett);
   while (i<n){
      if (a[i]=keresett){
          t++;
          System.out.println("A tomb "+i+". eleme es a keresett szam
megegyezik");
           }
      i=i+1;
      }
   System.out.println("A talalatok szama: "+t);
   }
}
```

```
A tomb 0. eleme:86.
A tomb 1. eleme:67.
A tomb 2. eleme:36.
A tomb 3. eleme:70.
A tomb 4. eleme:33.

A keresett szam: 2
A talalatok szama: 0
Helyes megoldás
```

Ellenőrzés

Tömb feltöltése n darab véletlen számmal és meghatározott szám lineáris keresése a tömbben

```
program Kereses;
uses crt;
var i,n,x,j:integer;
    a:array [1..100] of integer;
BEGIN
   clrscr;
   i:=1;
   n:=10;
   writeln('A tomb elemei:');
   while (i<=n) do begin
      a[i]:=trunc(90*random+1);
      i:=i+1;
      writeln('A tomb ',i,'.
eleme:',a[i]);
   end;
   i:=1;
   t:=0;
   keresett:=16;
   writeln('Keresett szam:',keresett);
   while (i<=n) do begin
      if (a[i]=keresett)the begin
          t:=t+1;
          writeln('A tomb ',i,'. eleme es
a keresett szam megegyezik');
           end;
      i:=i+1;
      end;
   writeln('A talalatok szama: ',t);
END.
```

```
A tomb 1. eleme:86.
A tomb 2. eleme:65.
A tomb 3. eleme:69.
A tomb 4. eleme:31.
A tomb 5. eleme:67.
A tomb 6. eleme:16.
A tomb 7. eleme:14.
A tomb 8. eleme:8.
A tomb 9. eleme:35.
A tomb 10. eleme:11.

A keresett szam: 16
A tomb 6. eleme es a keresett szam megegyezik.
A talalatok szama: 1
Helyes megoldás
```

Ellenőrzés

Tömb feltöltése n darab véletlen számmal és meghatározott szám szekvenciális keresése a tömbben

```
public class Kereses{
   public static void main(String[] args){
   int n,i,t,x,j,keresett;
   int[] a=new int[100];
   n=5;
   i=0;
   System.out.println("A tomb elemei:");
   while (i<n){
     a[i]=(int)(90*Math.random()+1);
     i=i+1;
     System.out.println("A tomb "+i+".
eleme:"+a[i]);
     }
   while (i+1<n){
      j=i+1;
      while (j<n){
         if (a[j]<a[i]){
             x=a[j];
             a[j]=a[i];
             a[i]=x;
             }
         j=j+1;
      }
   i=i+1;
   }
   System.out.println("");
   System.out.println("Rendezes utan")
   i=0;
   while (i<n){
      System.out.println("A tomb "+i+". eleme:"+a[i]);
      i=i+1;
   }
   i=0;
   t=1;
   keresett=22;
   System.out.println("Keresett szam:"+keresett);
   while (i<n){
      if (a[i]>sz) break;
      if (a[i]=keresett){
          t++;
          System.out.println("A tomb "+i+". eleme es a keresett szam
megegyezik");
           }
      i=i+1;
      }
   System.out.println("A talalatok szama: "+t);
   }
}
```

```
A tomb 0. eleme:62.
A tomb 1. eleme:20.
A tomb 2. eleme:3.
A tomb 3. eleme:26.
A tomb 4. eleme:29.

Rendezes utan:

A tomb 0. eleme:3.
A tomb 1. eleme:20.
A tomb 2. eleme:26.
A tomb 3. eleme:29.
A tomb 4. eleme:62.

A keresett szam: 22
A talalatok szama: 1
```

2.3. Felhasznált programok

Az oktatási segédanyag webes felületen történő megjelenítéséhez több programot használtunk fel.
A grafikai részt a CorelDRAW Graphics Suite X4 próbaverzióját, a Photoscape v3.3-mat és az IrfanView-t használtuk fel. A szerkesztésekhez használt Corel programcsomag segítségével egyszerűen elkészíthetőek voltak a grafikus felület részei (gombok, menürendszer és egyéb díszítő elemek), valamint a folyamatábrák. A Photoscape és Irfanview programokat leginkább a képek utólagos, esetleg csoportos átnevezésére, vágására, átméretezésére alkalmaztuk. Ezen programok lehetőségeit kihasználva jó minőségben, egyszerűen tudtuk végrehajtani a kívánt műveleteket.
A webes alkalmazás kódolását a Microsoft Expression Web 2.0 próbaverziójával végeztük. Az új fejlesztésű alkalmazás több programozási nyelvben is segítséget nyújtott (HTML, CSS, JavaScript). A grafikus és a forráskód szerkesztő rész együttes használata az oldalak összeállítását megkönnyítette.

3. Összegzés

A webes felületre elkészített oktatási segédanyag kialakításakor saját tapasztalatainkra alapozva igyekeztünk érthető magyarázatokat nyújtani a következő évfolyamok számára. Reméljük, hogy a jövőbeli felhasználóknak segítséget fog jelenteni munkánk a Java és Pascal programozási nyelvek sikeres elsajátításához.

Köszönetnyílvánítás

Szeretnénk köszönetet mondani témavezetőnknek, Dr. Boda Istvánnak a szakdoglozatunk elkészítésében nyújtott segítségéért, ötleteiért, javításaiért.

Irodalomjegyzék

Programozási nyelvek

[1.] http://hu.wikipedia.org/wiki/Programoz%C3%A1si_nyelv
[2.] http://zeus.nyf.hu/~etelka/prognyelvfejl.doc

Pascal

[1.] http://hu.wikipedia.org/wiki/Pascal_(programoz%C3%A1si_nyelv)
[2.] http://indy.poliod.hu/program/Pascal/Tankonyv/Tankonyv.htm
[3.] http://prog.hu/cikkek/330/Alapok.html
[4.] http://programozo.cellszaksuli.hu/download/programming/pascal/tetelsor.doc
[5.] http://www.kobakbt.hu/jegyzet/PascalPrg/pascal3.htm
[6.] http://www.prog.ide.sk/pas.php
[7.] http://www.sulinet.hu/inform/szakkor/jegyzet/pascal1/
[8.] http://zeus.nyf.hu/~akos/pascal/pasframe.htm

Java

[1.] http://hu.wikibooks.org/wiki/Java_Programoz%C3%A1s/A_nyelv_jellemz%C5%91i
[2.] http://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv)
[3.] http://www.cab.u-szeged.hu/WWW/java/kiss/article.html
[4.] http://www.cab.u-szeged.hu/WWW/java/kiss/javaprog.html
[5.] http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/ficsorl/Targyak/OOP/Segedletek/java1.pdf
[6.] http://www.jgypk.u-szeged.hu/~devosa/dll/stud/Java/docs/e-Book_Java_prog_1.3.pdf
[7.] http://www.jgypk.u-szeged.hu/~devosa/dll/stud/Java/docs/JAVA.pdf
[8.] Rogers Cadenhead: Tanuljuk meg a java programozási nyelvet 24 óra alatt, 2006

E-learning

[1.] http://edutech.elte.hu/multiped/szst_11/szst_11.pdf
[2.] http://en.wikipedia.org/wiki/E-learning
[3.] http://ip.gallup.hu/elearning/eu.htm
[4.] http://ip.gallup.hu/elearning/mo.htm
[5.] http://ip.gallup.hu/elearning/szabv.htm
[6.] http://mek.oszk.hu/06800/06829/06829.pdf
[7.] http://thaleia.kodolanyi.hu/~dancso/dancso/2004.11.08/2004.11.08_DancsoT.pdf

[8.] http://tmt.omikk.bme.hu/show_news.html?id=4255&issue_id=467
[9.] http://www.bgk.bmf.hu/mpt/az_e-learning_szerepe_a_felnottoktatasban_es-kepzesben.pdf
[10.] http://www.coedu.hu/domain9/files/modules/module15/2699844175F3CB3.pdf
[11.] http://www.edu-online.eu/hu/letoltes.php?fid=tartalomsor/271
[12.] http://www.eduweb.hu/pdf/e_learning_tan.pdf
[13.] http://www.efeb.hu/felnottkepzes-e-learning
[14.] http://www.elek.co.hu/dpi/moodledata/46/Az_e-learning.ppt
[15.] http://www.enc.hu/1enciklopedia/fogalmi/ped/elern.htm
[16.] http://www.hrportal.hu/index.phtml?page=article&id=61323
[17.] http://www.oki.hu/oldal.php?tipus=cikk&kod=akademia-2002-Hidvegi-elearning