EFOP-3.4.3-16-2016-00014

# Jegyzet

## Programozás-technikai praktikum

**Készítette: Árgilán Viktor Sándor**
**dr. Kelemen András**

**SZTE JGYPK Informatika Alkalmazásai Tanszék**

**2019. június 30.**

Jelen oktatási segédanyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával. Projekt azonosító: EFOP-3.4.3-16-2016-00014.

Alprojekt azonosító: AP1 alprojekt – Hallgatói diploma-szerzést segítő szolgáltatások
Altéma azonosító: AP1_JGYPK altéma Lemorzsolódást csökkentő program kifejlesztés a JGYPK-n az MTMI területeken

**Szegedi Tudományegyetem**
**Cím: 6720 Szeged, Dugonics tér 13.**
**www.u-szeged.hu**
**www.szechenyi2020.hu**

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

# Tartalom

## Tantárgy tanításának célja

A kurzus elsődleges célja a Flowgorithm szoftver segítségével egyfajta problémamegoldó gondolkodásmód kialakítása, valamint a programozáshoz szükséges legalapvetőbb tudás átadása a hallgatóknak. A programozási feladatok megoldását, folyamatábra segítségével lépésről lépésre mutatja be azért, hogy a hallgató később önállóan is képes legyen alapvető problémákat megoldani. A kurzus hallgatói megtanulhatják a problémamegoldó gondolkodás alapjait, valamint a programozáshoz szükséges legalapvetőbb elemeket (változók kifejezések, tömbök, stb). Mindemellett elsajátíthatják a programokban használatos különböző vezérlési szerkezetek (szekvenciális, szelekciós, iterációs) alapjait. Célunk továbbá példákon és elméleti ismeretanyagokon keresztül egyfajta algoritmikus gondolkodás megtanítása a hallgatóknak, amely a programozóvá válás első fontos és elengedhetetlen mérföldköve.

### A tantárgy tanulási eredményei

Azoknak az **előírt szakmai kompetenciáknak, kompetencia-elemeknek** *(tudás, képesség* stb., *KKK 7. pont*) a felsorolása, **amelyek kialakításához a tantárgy jellemzően, érdemben hozzájárul:**

| **Tudás** | **Képesség** | **Attitűd** | **Autonómia/felelősség** |
|---|---|---|---|
| Ismeri az algoritmus fogalmát.<br><br>Tisztában van a folyamatábra fogalmával. | Képes önállóan algoritmusok leírására, megadására. | Elkötelezett a minőségi munka iránt, jól átlátható és pontos folyamatábrát készít.<br><br>Megérti a további megadási módok jelentőségét. | Tisztában van az általa létrehozott algoritmusokkal szemben támasztott követelményekkel. |
| Ismeri a különböző vezérlési szerkezeteket. | Képes önállóan algoritmusok készítésére, a megismert vezérlési szerkezetek segítségével. | Megérti, hogy mikor melyik vezérlési szerkezetet célszerű használni. | Önállóan készít egyszerű algoritmusokat. |

| | | | |
|---|---|---|---|
| Ismeri a programozás építőelemeit és alap módszereit. | Képes önállóan kisebb programok algoritmusának megadására és matematikai problémák programozással történő megoldására. | Elkötelezett a minőségi munka iránt, jól dokumentált algoritmust ír és megérti, hogy a jó programozó egyik, ha nem a legfontosabb ismérve, hogy jól tud csapatban dolgozni. | Tudatában van, hogy az általa megadott algoritmusokkal szemben milyen követelményeket támaszt az esetleges megrendelő, valamint felelősséget vállal az általa létrehozott szellemi termékért. |
| Ismeri a Flowgorithm szoftver által kínált eszköztárat, valamint a környezet tulajdonságait. | Képes komplex programok alapjainak a megtervezésére és kivitelezésére, valamint hozzájuk tartozó hatékony programrészletek létrehozására. | Megérti, hogy a programozónak folyamatosan képeznie kell önmagát és tisztában kell lennie az éppen aktuális technológiákkal és trendekkel. | Önállóan készít komplex algoritmusokat, ellenőrzi és javítja hibáit, önállóan tesztel. |
| Ismeri a programozással kapcsolatos alapfeladatokat és a rájuk vonatkozó megoldási módszereket. | Képes egyszerű programok futtatható algoritmusának megadására. | Érdeklődik a valós élet problémái iránt, megoldásukra programokat készít. | |
| Ismeri a szoftverfejlesztéshez és teszteléshez szükséges alapvető módszereket, és képes ezek használatára. | Képes az ismert módszerekkel tesztelni az elkészült algoritmust. | Elkötelezett a precíz, hibátlan munkavégzés iránt. | Önállóan ellenőrzi és javítja munkáját, önállóan tesztel. |

| | | | |
|---|---|---|---|
| Tisztában van a Flowgoritm-ben írt programok működésével, valamint konvertálási lehetőségeivel | Képes kialakítani a programozáshoz szükséges hardver és szoftverkörnyezetet.<br><br>Képes több programozási nyelvre (JAVA-ra is) fordítani a munkáját a Flowgorithm segítségével. | Érdeklődik az új programozással kapcsolatos technológiák iránt, valamint törekszik ezek megismerésére és a tudásának a karbantartására. | Önállóan hoz létre kisebb alkalmazásokat, és határozza meg a szükséges technológiákat. |
| Tisztában van a változók, tömbök, kifejezések szerepével, valamint a különböző megoldási módszerek során a használatukkal és rendelkezik az értelmezésükhöz és használatukhoz szükséges képességekkel. | A követelményeknek és a megrendelő igényeinek megfelelő programot hoz létre. | Elkötelezett a minőségi munka és a best practice használata mellett annak érdekében, hogy hatékonyan működő algoritmust adjon meg. | Felelősséget vállal az általa létrehozott algoritmusért, valamint tisztában van az általa létrehozott programmal kapcsolatos felelősségével és annak következményeivel. |
| Ismeri az algoritmusokban előforduló függvények szerepét és tisztában van a szerepükkel. | Korszerű technológiákat használ a szoftvertermék létrehozásakor, valamint képes a tudásra építkezve új technológiák elsajátítására. | Nyitott a programozási nyelvek megismerésére és törekszik is a velük kapcsolatos tudás megszerzésére. | Betartja a struktúrált kóddal és annak megfelelő és jól átlátható kommentelésével kapcsolatos elvárásokat, és tisztában van a fontosságukkal. |

## Adatok ábrázolása

Programozói szemmel nézve a számítógépes programok adatokból és az adatokon különböző műveleteket végző utasításokból állnak. Az utasításoknak a számítógép számára egyértelmű módon kell leírniuk az adatokon végezendő műveleteket. Az ilyen utasítás sorozatokat az informatikában algoritmusoknak nevezzük. Példák algoritmusokra:

- Szavak abc sorrendbe valamint számok növekvő vagy csökkenő sorrendbe rendezése
- Szövegben különböző szavak keresése
- Egyenletek, egyenletrendszerek megoldása.
- Személyi jövedelemadó kiszámítása.

A Neumann elv szerint a mai számítógépek az adatokat és az utasításokat ugyanabban a formában tárolják a memóriában.

Az algoritmusok utasításainak függvényében az adatok értéke a program futása során változhat, ezért a különböző programozási nyelvek az adatatok tárolására bevezették a változó fogalmát. Egy változó a következő tulajdonságokkal rendelkezik:

- Minden változónak van címe, mely azt mutatja meg, hogy a változó hol van a memóriában.
- Minden változónak van adattípusa, azt mondja meg, hogy az adott változót hogyan kell a programozónak is és a számítógépnek is értelmeznie (szám, betű, szöveg, logikai érték, kép, stb). A programozási nyelvek egy részénél a programozónak már a változó létrehozásakor a meg kell adnia az adtatípust. Más nyelvek esetén futtató környezet ezt automatikusan határozza meg, a programozónak nem kell ezzel törődnie.
- Minden változónak van mérete, mely azt határozza meg, hogy az illető változó mekkora helyet foglal el a memóriában.
- Minden változónak van neve, melyet a program forráskódjában a változó azonosítására szolgál. A változó nevét a programozó a használt nyelv keretein belül szabadon meghatározhatja.
- Minden változónak van értéke, mely természetesen a program során változhat.

A változókat csoportosíthatjuk úgy is, hogy a változóban tárolt adat milyen információ tartalommal bír. Így egy változót egyszerű változónak nevezünk, ha értéke pl. egy szám, betű, logikai érték. Egy változót összetett változónak nevezünk, ha a változó értéke több egyszerű változó értékeiből áll. A változók létrehozása (deklarálása) a C/C++, C#, Java nyelveken a következő:

```
adatípus változónév;
```

A pontosvessző ezekben a nyelvekben az utasításelválasztó jel. Ezt forráskódban minden utasítás végén ki kell tenni. Értéket a változónak a következőképpen adhatunk:

```
változónév = érték;
```

Természetesen a deklarációt és az értékadást össze is fűzhetjük:

```
adattípus változónév = érték;
```

A fent említett programozási nyelvek az egyszerű változók kezelésére beépített adattípusokkal rendelkeznek.

**Egyszerű adattípusok:**

*Szám*

Általában külön típusként kezeljük az egész (integer) és a valós (double) számokat. Továbbá meg kell említenünk, hogy az egyes programozási nyelvekben lehetőség van további szám típusú változók használatára, melyek a következők lehetnek:

*Egész típusok:*

- 8 biten tárolt egész, bájt méretű egész (JAVA-ban: byte)
- 16 biten tárolt egész, rövid egész (JAVA-ban: short)
- 32 biten tárolt egész (JAVA-ban: int)
- 64 biten tárolt egész, hosszú egész (JAVA-ban: long)

Műveletek egész számokkal:

- értékadás;
- összehasonlítás (operátorok: <, >, <=, >=, =, <>)

- matematikai műveletek (+, -, *, maradékos osztás DIV:/ MOD:%; hatványozás ^)

*Valós szám típusok:*

- 32 biten tárolt egyszeres pontosságú lebegőpontos valós szám (JAVA-ban: float)
- 64 biten tárolt dupla pontosságú lebegőpontos valós szám (JAVA-ban: double)

Műveletek valós számokkal:

- értékadás;
- összehasonlítás (operátorok: <, >, <=, >=, =, <>)
- matematikai műveletek (+, -, *, /, hatványozás ^)

*Logikai*

Speciális változó, amely értéke lehet igaz (true) vagy hamis (false), (JAVA-ban: boolean).

Műveletek logikai kifejezésekkel:

- értékadás;
- tagadás NOT, és AND, vagy OR, kizáró vagy XOR

*Szöveg*

Minden adat, amit a billentyűzetről begépelünk eltárolható szövegként. A szöveg karakterekből épül fel, ezért több fejlesztői környezetben lehetőség van karakter típusú adatok használatára (JAVA-ban: char; 16 biten tárolt Unicode karakter), valamint a szöveg mint adattípus szerepelhet alap adattípusként, vagy lehet összetett típus (lásd később), mint például JAVA-ban a stringek karakterekből álló láncok, azaz karakterláncok (JAVA-ban: String, a nagy kezdőbetű nem véletlen!).

Műveletek karakterekkel:

- értékadás;
- kódtábla szerinti értékvizsgálat
- nincsenek karakter-specifikus műveletek

Műveletek szöveggel:

- értékadás;
- összefűzés
- a szöveget alkotó karakterekhez tartozó kódtábla szerinti értékvizsgálat
- egy adott indexű karakter kiválasztása

**Összetett adattípusok:**

A gyakorlati életben sok olyan probléma található, melyek megoldásához nem elegendő az egyszerű adattípusok használata. Lehetőségünk van úgynevezett összetett adatok, vagy más néven adatstruktúrák használatára, melyeket az egyszerű adattípusok felhasználásával alakítottak ki. Ezek a következők:

- tömbök
- rekordok
- láncolt listák
- kulcs – értékpár típusú tárolók (map)
- bináris fák
- verrmek (FILO First In Last Out)
- sorok (FIFO First In First Out)
- gráfok

Ebben a jegyzetben a felsorolt fenti összetett adattípusok közül csak a tömbökkel, rekordokkal és láncolt listákkal foglalkozunk.

*Tömbök*

Az azonos típusú adatokból álló rendezett adatsorokat tömböknek nevezzük. A tömbök gyakorlatilag mátrixok, melyeket lineáris algebrai struktúrák. Az adatokat sorokba és oszlopokba rendezzük, azaz egy mátrix (tömb) általánosságban a következő alakban adható meg:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{m \times n}$$

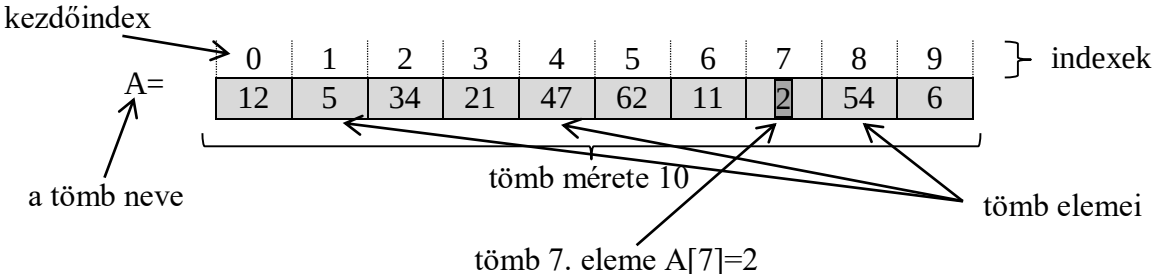
ahol m, n egész szám, és $a_{ij}$-t a mátrix általános elemének nevezzük, ahol $1 \leq i \leq m$ és $1 \leq j \leq n$

Léteznek speciális mátrixok, azaz olyan esetek, amikor vagy az oszlopok, vagy a sorok száma egy. Természetesen további speciális mátrixok is léteznek, úgy mint négyzetes mátrix, egységmátrix, nullmátrix, stb., de ezek tárgyalására jelen dokumentumban nem térünk ki.

$$B = \begin{pmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{m1} \end{pmatrix}_{m \times 1} \qquad C = (c_{11} \quad c_{12} \quad \cdots \quad c_{1n})_{1 \times n}$$

Azokat a mátrixokat, melyek csak egyetlen sorból, vagy egyetlen oszlopból állnak, vektoroknak nevezzük, külön beszélhetünk oszlopvektorról vagy sorvektorról. Programozás közben külön kezeljük a vektorokat és a több sorból és oszlopból álló mátrixokat, előbbieket egydimenziós tömbnek (JAVA-ban pl.: int[] anArray;), míg utóbbiakat kétdimenziós tömböknek (JAVA-ban pl.: int[][] anArray;) nevezzük. A tömb mérete a program minden egyes futtatásakor változik, létrehozásakor kerül megállapításra, mely a program végéig állandó.

Használatuk megértéséhez tekintsük a következő példát:



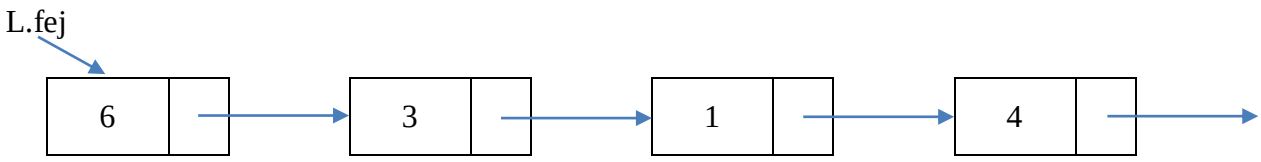

A tömb egy olyan változó, amely több azonos típusú adatot tartalmaz. A tömb (futásidei) hossza a létrehozásakor kerül megállapításra, és attól kezdve a tömb egy állandó méretű adatszerkezet. A tömbben tárolt elemeket a tömbben elfoglalt helyük (indexük) alapján érhetjük el.

*Rekordok*

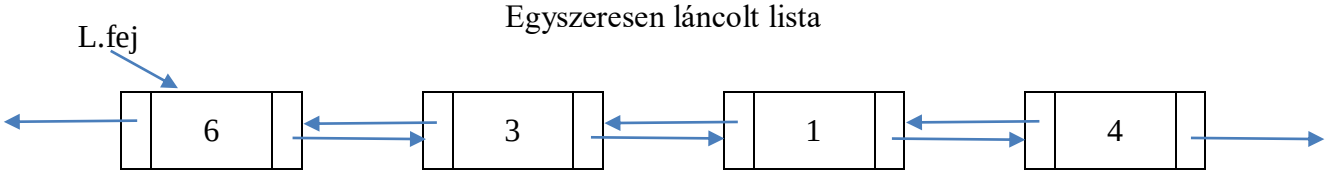
A tömbök használata megkönnyíti a programozók munkáját, ha nagy mennyiségű azonos adatok feldolgozásáról, tárolásáról van szó. A gyakorlati problémák megoldásához szükségünk lehet olyan adatokra is, melyek egy dologhoz tartoznak, de különböző tulajdonságait különböző alap-adattípusokban kell eltárolni. A probléma megoldásához vezették be az úgynevezett rekordstruktúrát, röviden rekordokat, melyek használata számos esetben nélkülözhetetlen (pl.: adatbázis-kezeléssel kapcsolatos feladatoknál).

*Láncolt listák*

A láncolt lista már dinamikus adatszerkezet, mely abban nyilvánul meg, hogy a lista elemeinek a száma a program futása során változhat. Futás időben tudjuk a listát bővíteni, tudunk a listából törölni. Ráadásul a tömbökkel ellentétben a listaelemeknek nem kell egymás mellett lenniük a memóriában. A listának egy eleme mindig két részből épül fel: adatrész, címzés rész. Az adatrész a listaelemben tárolt információ. Ez lehet elemi adattípus, de lehet rekord is. A címzés részben azok az információk vannak, amelyekből egyszeresen láncolt lista esetén hivatkozni lehet következő listaelemre Kétszeresen láncolt lista esetén pedig az előző elemre is.




Egyszeresen láncolt lista




Kétszeresen láncolt lista

A láncolt listában minimálisan az az alábbi műveleteket szokták megvalósítani:

- új listaelem hozzáadása
- meglévő listaelem módosítása
- listaelem törlése
- listaelem elérése

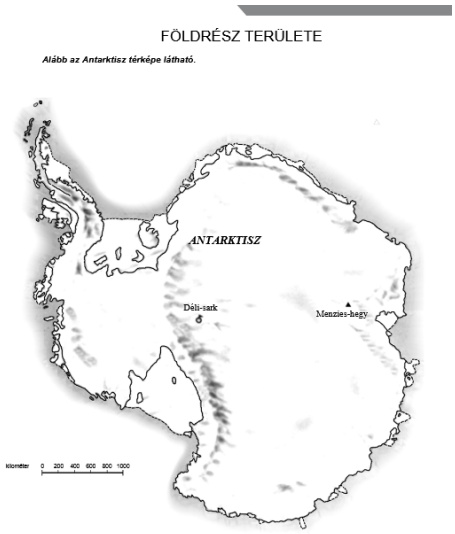
## Algoritmusok

Hétköznapi életünk során folyamatosan megoldandó problémákkal találkozunk, melyek megoldása nélkülözhetetlen az életben való boldoguláshoz. Természetesen a legtöbb feladatot ösztönösen oldjuk meg, a feladat megoldásához vezető lépések természetesek számunkra. Az összetettebb feladatok azonban komolyabb elemzést igényelnek. Fel kell térképeznünk a probléma megoldásához szükséges műveleteket, a hozzájuk tartozó feltételeket, majd meg kell határozni a megoldáshoz vezető lépéseket. A megoldás során általában törekszünk az egyszerűségre, valamint a probléma pontos megoldására. Tapasztalataink alapján egy feladat megoldása során több megoldás adható, ezért a megoldás meghatározása előtt szükséges az összes peremfeltétel megismerése, az esetlegesen felmerülő problémák azonosítása, azaz a tervezés fázisa. A tervezés során meghatározott megoldási módszereket elemezzük, majd megadjuk a feladat megoldását. Fontos, hogy a megoldási módszerek közül a legjobbat adjuk meg. A „legjobb” jelző meghatározása nem túl egyszerű feladat:

- a legpontosabb megoldást kapjuk
- a hibalehetőségek minimalizálása
- a megoldáshoz vezető lépések minimalizálása

Tekintsük a következő feladatot, mellyel a PISA2000 felmérésben találkozhattunk:

Becsüld meg az Antarktisz területét a térképen feltüntetett méretarány segítségével! Írd le a számításaidat és azt, hogyan végezted el a becslést! (Rajzolhatsz is a térképre, ha ez segít a becslésben.)



A feladat megoldása során számos problémával találkozhatunk:

nem a „megszokott” (háromszög, négyszögek, sokszögek, kör és részei) síkidomok területét kell meghatározni, illetve az alakzatot nem tudjuk az említett síkidomokra bontani

- nincs megadva területmérték, csak hosszmérték
- számolni kell a méretaránnyal is
- stb…

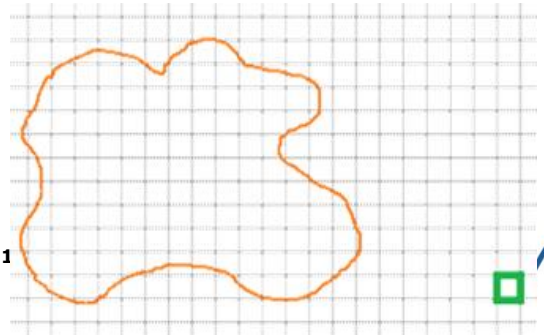
A feladatból kiindulva vizsgáljuk a következő feladatot:

Becsüljük meg a következő alakzat területét!

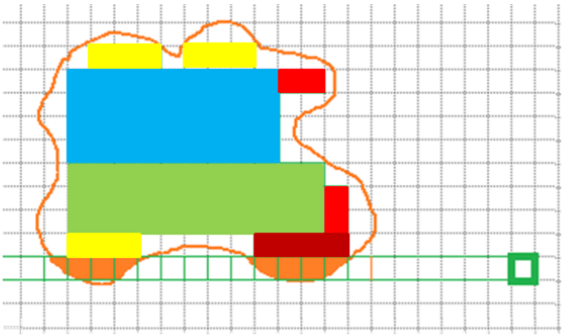
A fent említett problémák jelentős része itt szintén megtalálható, azaz a tervezés fázisban próbáljuk meg ezen problémákat kiküszöbölni, hogy az eredeti problémát is meg tudjuk oldani.

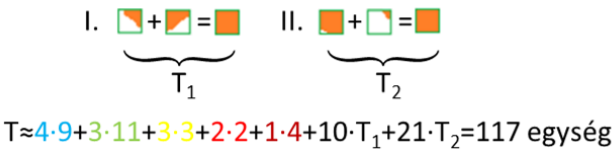
1. Adjunk meg egy területegységet!

Fektessünk egy területegységekből álló rácsot az alakzatra.



2. Próbáljuk meg olyan síkidomok területére bontani az alakzatot, melyeknek ismert a területszámítási metódusa.






$T \approx 4 \cdot 9 + 3 \cdot 11 + 3 \cdot 3 + 2 \cdot 2 + 1 \cdot 4 + 10 \cdot T_1 + 21 \cdot T_2 = 117$ egység

Jól látható, hogy a síkidom belső része téglalapokra bontható, a probléma a fektetett négyzetháló azon részével van, amely a határvonal közelében található. Ezen részek becsléséhez vezessünk be két területtípust T1-et és T2-őt:

T1 esetében közel azonos a felhasznált két négyzetben az alakzat által lefedett és az alakzathoz nem tartozó területek aránya, azaz két ilyen négyzet alkot egy területegységet.

T2 esetében a felhasznált két négyzet esetén az egyikben túlnyomóan az alakzat által lefedett rész található, míg a másikban az alakzathoz nem tartozó rész aránya a nagyobb, azaz két ilyen négyzet alkot egy területegységet.

Ily módon jól becsülhető a kérdéses terület, annak ellenére, hogy kezdetben, a feladat megismerése után komoly elemzésre volt szükség.

## Az algoritmus fogalma

Azt a folyamatot, mely egyszerű lépések meghatározott sorozatával általános megoldást ad egy adott problémára, algoritmusnak nevezünk.

Algoritmusok tulajdonságai:

- véges (mind időben, mind lépésszámban)

- jól specifikált
- végrehajtható
- általános megoldást ad egy problémára

Az algoritmusok megadásánál a feladatot elemi lépésekre bontjuk, mely lépések egyszerűen megoldhatók. Az egyes lépéseket meghatározott sorrendben szigorú szabályok szerint kell végrehajtani, melyről később a vezérlési szerkezetek részben tárgyaljuk.

Stratégiák:

- Bottom-up: kisebb algoritmusrészek összefűzése nagyobbakká
- Top-down: probléma formalizálása, majd fokozatosan kisebb és kisebb részekre bontása

Az algoritmusok általános tárgyalásánál láttuk, hogy a problémákat részproblémákra, majd elemi megoldható lépésekre kell felbontani. Az algoritmus tulajdonképpen ezen elemi lépések megfelelő sorrendben történő megoldása, azaz feladatunk a sorrend megadása. Minden feladatnak van kezdete és vége, így az őket megoldó algoritmusnak is van kezdete, kezdőpontja és vége, végpontja. Bármilyen algoritmust tervezünk, ezek a lépések, azaz a kezdőpont (START) és végbont (END) egységbe foglalják a tervezett algoritmust. Az elemi lépések a rendszer számára végrehajtható feladatok, de az algoritmus végrehajtása közben elképzelhető, hogy problémák merülnek fel, így az elemi lépés nem lesz megoldható. Például: az osztás művelete elemi művelet, azaz két számot el tud osztani egymással a rendszer, kivéve, ha az osztó zérus. Abban az esetben, ha két szám típusú változót kell egymással elosztani, semmi sem garantálja, hogy az osztó nem nulla, hacsak erre a lehetőségre nem térünk ki, azaz meg kell vizsgálnunk, hogy az osztás elvégezhető vagy sem, tehát feltételeket adunk meg. A feltételek kiértékelésénél az összes lehetséges megoldást figyelembe kell venni, és a korrekt megoldás érdekében új lépéseket beiktatni, vagy visszatérni az előző lépéshez. Az is előfordulhat, hogy bizonyos lépéseket többször kell elvégezni, ha ezeket egymás után kell megtenni, akkor ismétlésről beszélünk. Például: ha össze szeretnénk adni a természetes számokat 20-ig, akkor az ismert képlet felhasználásán kívül lehetőségünk van arra is, hogy egy összeg változó értékét mindig a következő természetes számmal növeljük, és ezen lépésünket ismételjük mindaddig,

míg az utolsó hozzáadandó szám a 20 lesz. Jól látható, hogy az algoritmus felépítésénél szükségünk lehet feltételek megadására, kiértékelésére, melyek alapján elágazások építhetők az algoritmusokba, valamint bizonyos részproblémák többszöri végrehajtására is szükség lehet, így úgynevezett vezérlési szerkezeteket tudunk megadni.

Az algoritmusoknál használt vezérlési szerkezetek Edsger Wybe Dijkstra holland matematikus, informatikus kutató nevéhez fűződnek aki 1960-ban a következőt publikálta:

Minden strukturált algoritmus meghatározható három alapelem segítségével:

- Szekvenciális: részproblémák egymás utáni megoldása
- Szelektív: feltételhez kötött megoldása a részproblémáknak (elágazások)
- Iteráció: részproblémák egy csoportjának többszöri megoldása (ismétlés)

Az algoritmusok meghatározásának megadásakor láthattuk, hogy egy adott problémára általános megoldást keresünk, azaz a feladatok megoldása közben nem konkrét adatokkal, hanem adatokat helyettesítő változókkal dolgozunk.

## Az algoritmus leírása

*Algoritmus megadása mondatokkal*

Az emberi kommunikációhoz legközelebb álló megadási mód, ha mondatokat alkotva, azokat meghatározott sorrendbe állítva (ezt általában úgy érjük el, hogy sorszámozott mondatokat használunk) adjuk meg az algoritmust. Ezt a fajta algoritmus megadási módot a sorszámozott utasítású programozási nyelvekhez, mint pl. a Fortran használták, illetve algoritmusok tanításának kezdetén, illetve az algoritmikus gondolkodás kialakításához vezető út elején szokták alkalmazni. A mondatokkal mindent le tudunk írni, hátránya azonban, hogy nem látható a program szerkezete, azaz kicsit átláthatatlan az algoritmus.

*Leírás mondatszerű elemekkel*

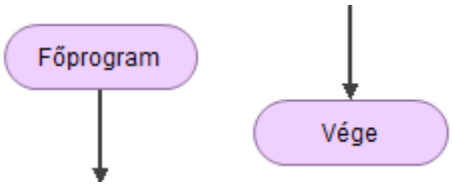
Az előző algoritmus megadási módszertől alapvetően abban különbözik, hogy nem használunk teljes mondatokat, hanem csak mondatszerű elemeket (szavakat, mondatrészeket, azaz hiányos mondatokat). A mondatszerű elemeket strukturáltan lehet leírni, így a program szerkezete jól

átlátható. Megjegyezzük azonban, hogy a későbbi fejezetekben ugyan használjuk ezt az algoritmus leírási módot, de nem a szokásos formában.

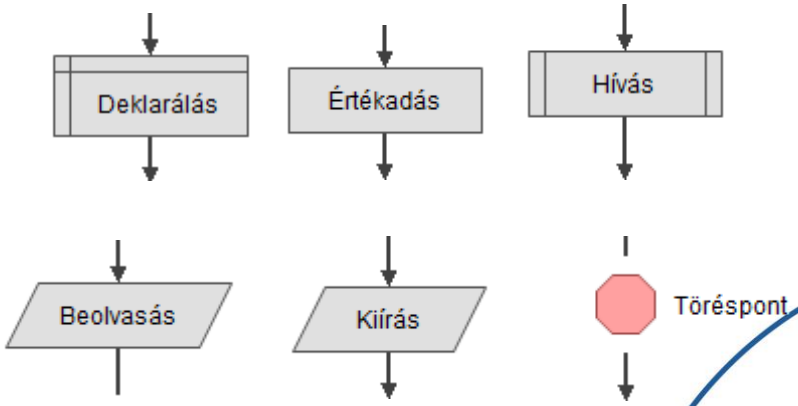
*Folyamatábra*

Az egyik legkorábbi és legtöbbször használt módszer az algoritmusok leírására, alapja, hogy a programot egy irányított gráffal szemléletesen adja meg. A gráf csomópontjai irányított élekkel vannak összekötve, minden él egyértelműen meghatározott kezdő és végponttal rendelkezik, továbbá a folyamatábrában egyetlen kezdő és egyetlen befejező él található. További tulajdonsága még a gráfnak, hogy bármely csomópontból el lehet jutni a befejező csomóponthoz. Az algoritmikus gondolkodás kialakításához egy Flowgorithm nevű szoftvert használunk, mely szintén az algoritmusok folyamatábrával történő megadását segíti, valamint futtató felülettel is rendelkezik, ahol a megadott algoritmus tesztelhető. Flowgorithm esetében a folyamatábra a következő csomópontokat tartalmazhatja:

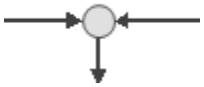
Kezdő és befejező csomópont (folyamatábráknál előfordulhat, hogy ezek a csomópontok nincsenek, az algoritmus egy bemenő éllel kezdődik, illetve egy kimenő éllel végződik)



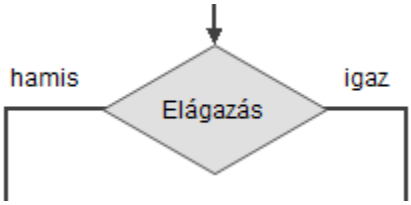

Utasítás csomópont (függvénycsomópont) végrehajtás közben a csomópontokba írt utasításokat kell elvégezni.



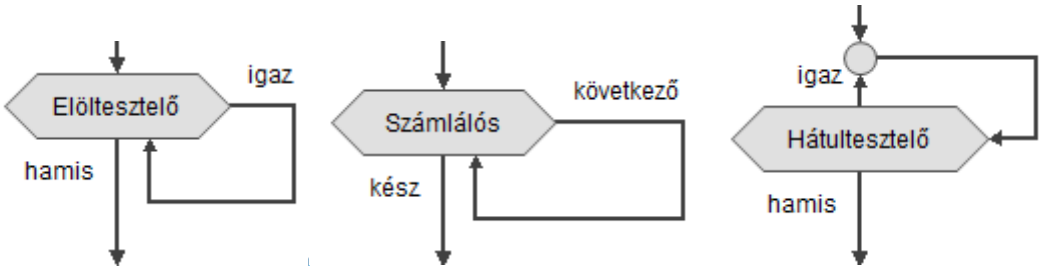
Gyűjtő csomópont (nem külön lépés, csak az egyes programrészek kimenő éleit fogja össze)



Döntés csomópont (A csomópontba mindig egy feltétel, azaz egy logikai kifejezés kerül, melyről egyértelműen eldönthető, hogy igaz vagy hamis. A feltételtől függően a megfelelő élen kell továbbhaladni.

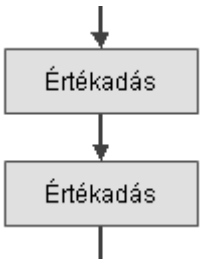



Ciklushoz tartozó csomópontok (a csomópontokba az adott ciklusra jellemző paraméterek találhatók)

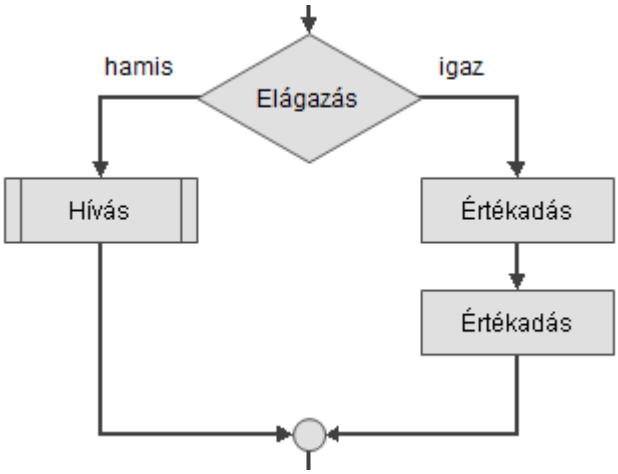



A folyamatábra nagyon jó szemléltető és leíró eszköz. A program struktúrája jól látható, valamint a végrehajtás is könnyen követhető, hiszen minden végrehajtás az input adatok függvényében egy bejárása a gráfnak. Számos előnye mellett a hátrányairól is szót kell ejteni. Sajnálatos módon a nagyobb programok leírása rendkívül körülményes, a hozzá tartozó gráf hatalmas, nehezen átlátható. Strukturált programoknál használatos vezérlési szerkezetek a következő módon jelennek meg a folyamatábrákban:

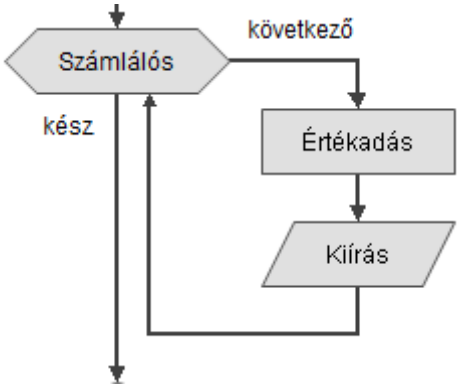
Szekvenciális vezérlés:




Szelekciós vezérlés:



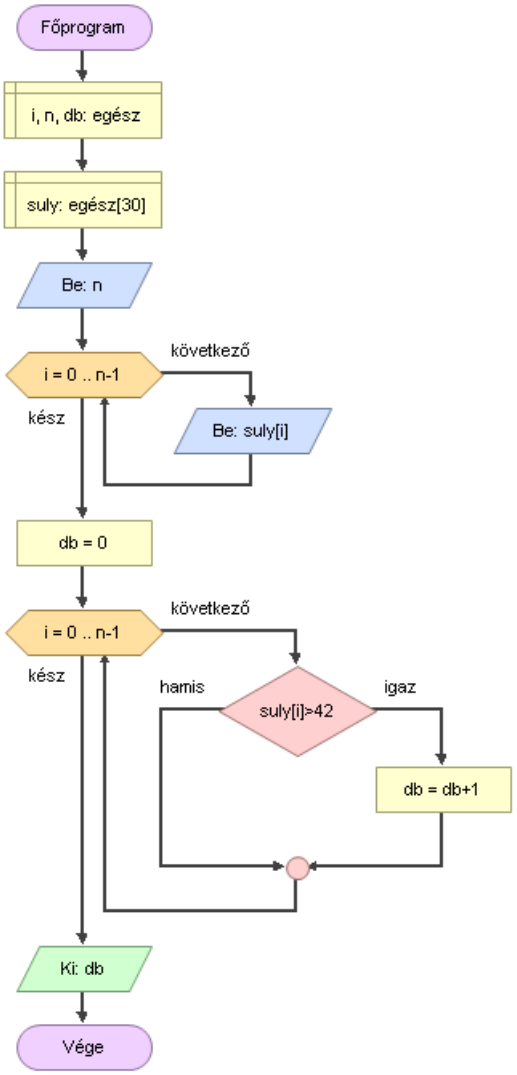

Iterációs vezérlés:



Tekintsük a következő egyszerű programot:

Egy osztály tanulói közül adjuk meg azon tanulók számát, akik testsúlya meghaladja a 42 kilogrammot!

A megadott folyamatábra Flowgorithm-ben készült, futtatható, a feladat megoldását szolgáltatja. Meg kell azonban jegyezni, hogy számos hiányosság fellelhető, amiket egy korrekt program algoritmusának írásakor természetesen meg kell adni.



*Pszeudokód*

Az algoritmusok megadásánál lehetőségünk van egy mesterséges formális nyelv használatára, mely nagyon hasonlít a programozási nyelvekre, de nem alkalmazza azok szintaktikai elemeit, így nem futtatható. A pszeudokód esetén az elemi lépésekhez szavak, jelek elnevezések vannak hozzárendelve, melyek segítségével az algoritmus könnyen leírható. A Flowgorithm-ben lehetőségünk van az általunk folyamatábrával megadott algoritmus pszeudokódjának generálására is. Az előző feladathoz tartozó Gaddis pszeudokód:

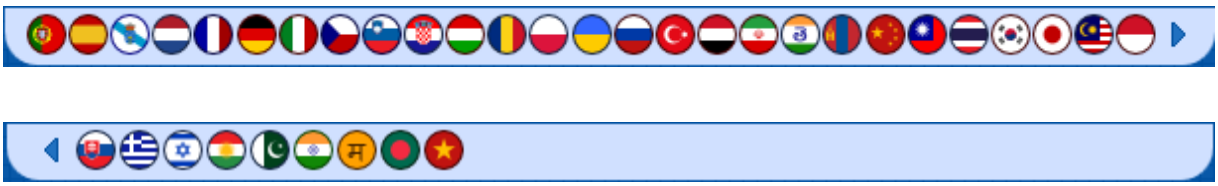
Az algoritmusok leírására számos további módszer ismert (Struktogram, Jackson-diagram, leírás fával, leírás programozási nyelven, stb.). Jelen dokumentum a programozástechnikai praktikum nevű tárgyhoz készült jegyzet, így csak a kurzuson használt leírási módokat ismertettük részletesen.

## Flowgoritm

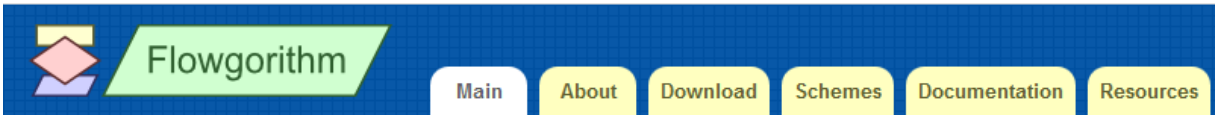
A kurzus tervezése közben több fejlesztői környezetet, egyszerű programozási nyelvet (Logo, Arduino, Scratch, Java, stb.) vizsgáltunk, de minden nyelv esetén az adott nyelv szintaktikája nagymértékben nehezítette az algoritmikus gondolkodás megértését, fejlesztését. Olyan környezetet kerestünk, melyben nem kell külön erőfeszítéseket tenni a szintaktikai elemek megismerésére, hanem elegendő csak kizárólag az algoritmusra koncentrálni. A kutatás közben találkoztunk a Sacramento Állami Egyetemen létrehozott Flowgorithm névre keresztelt freeware grafikus szerkesztő eszközzel, melyben kizárólag az algoritmus megadásával lehet programokat írni, tesztelni. Az algoritmus megadásához a folyamatábrát választották a fejlesztők, így hangsúly az algoritmuson van, a programozási nyelvek szintaxisa gyakorlatilag lényegtelen. Meg kell jegyeznünk azonban, hogy a létrehozott algoritmusok programozási nyelvekre konvertálhatók. A folyamatábra csomópontjaiban található alakzatok különböző színűek, mely színek a generált forráskód megfelelő soraiban is megjelennek. További segítség a hallgatóknak, hogy az angol nyelv mellett számos nyelven, többek közt magyarul is elérhető a felhasználói felület.

**Telepítés:**

A program hivatalos oldala a http://www.flowgorithm.org/ URL címen érhető el. Első lépésként a felkínált nyelvek közül válasszuk ki a magyar nyelvet.

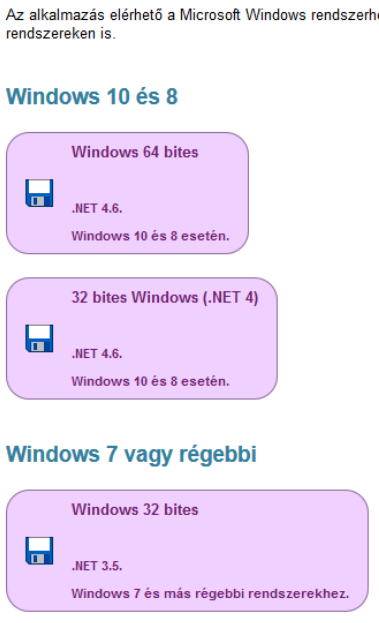


Az oldal menürendszere a google fordítónak köszönhetően a következő:

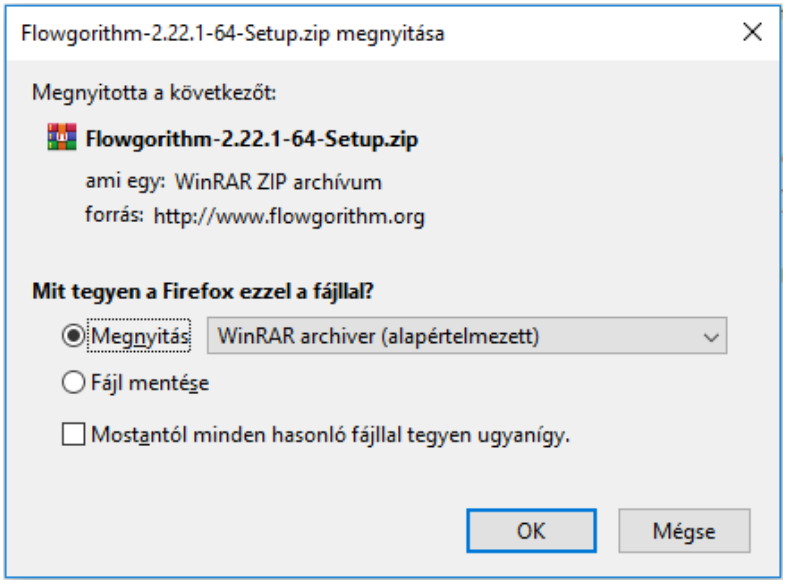




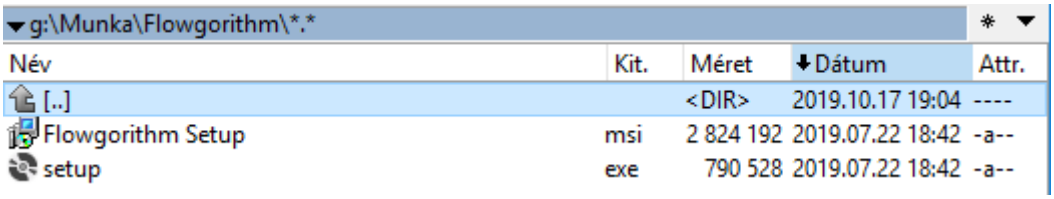
A telepítéshez a főmenüből ki kell választani a letöltés menüpontot, ahol jól látható, hogy jelenleg csak a Microsoft Windows operációs rendszerének különböző verziói a támogatott platformok, ígéret azonban van, hogy Linux és macOS rendszerekre is elérhető lesz a telepítő csomag.



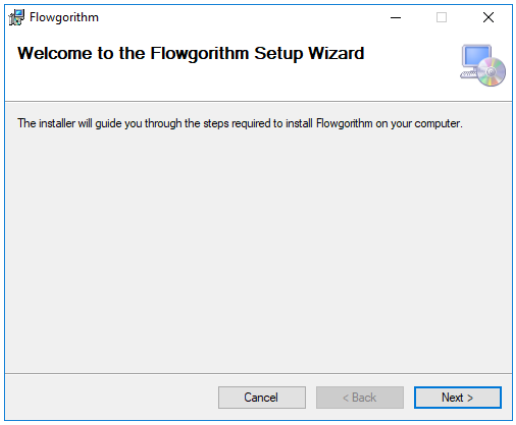
Fontos, hogy a számítógépünkön lévő operációs rendszernek megfelelő telepítő csomagot töltsük le, illetve, hogy a gépünkön telepítve legyen a megfelelő keretrendszer, azaz a DOTNET megfelelő változata. Miután kiválasztottuk a kívánt telepítő csomagot, a következő párbeszédpanellel találkozunk:

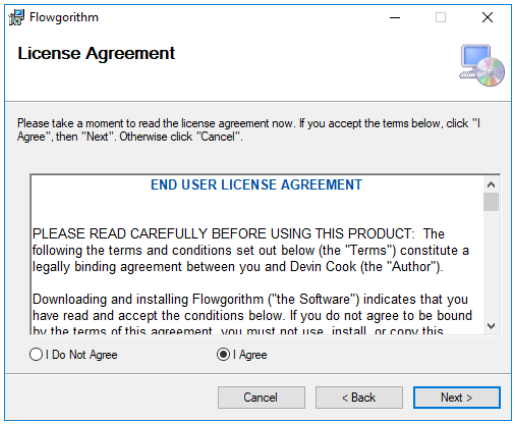


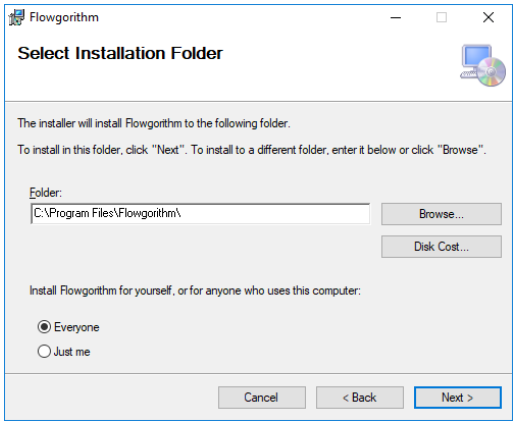

A letöltendő fájl egy tömörített állomány, mely tartalmazza a telepítéshez szükséges fájlokat. Két lehetőség közül választhatunk vagy megnyitjuk tömörített állományt a gépünkön lévő alapértelmezett program segítségével, vagy letöltjük és használhatjuk a Windows beépített megoldásait. A csomagban két fájl található:

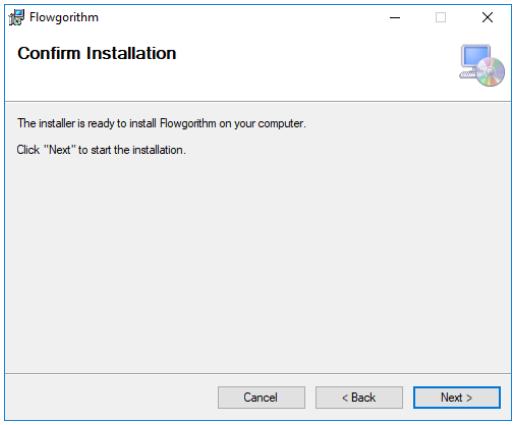


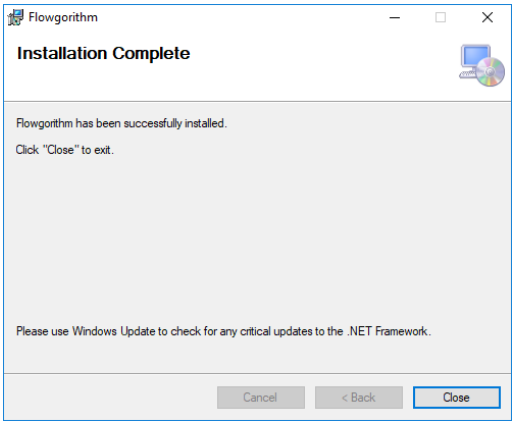

A setup.exe állománnyal indítható a telepítés, melyhez a megfelelő jogosultságok szükségesek (Windows rendszereknél általában rendszergazdai jogosultság). Az telepítés lépései a következők:


Flowgorithm
Welcome to the Flowgorithm Setup Wizard
The installer will guide you through the steps required to install Flowgorithm on your computer.
Cancel
< Back
Next >

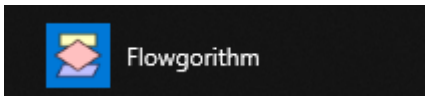

Flowgorithm
License Agreement
Please take a moment to read the license agreement now. If you accept the terms below, click "I Agree", then "Next". Otherwise click "Cancel".
END USER LICENSE AGREEMENT
PLEASE READ CAREFULLY BEFORE USING THIS PRODUCT: The following the terms and conditions set out below (the "Terms") constitute a legally binding agreement between you and Devin Cook (the "Author").
Downloading and installing Flowgorithm ("the Software") indicates that you have read and accept the conditions below. If you do not agree to be bound
I Do Not Agree
I Agree
Cancel
< Back
Next >

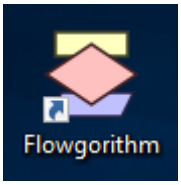

Flowgorithm
Select Installation Folder
The installer will install Flowgorithm to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files\Flowgorithm\
Browse...
Disk Cost...
Install Flowgorithm for yourself, or for anyone who uses this computer:
Everyone
Just me
Cancel
< Back
Next >


Flowgorithm
Confirm Installation
The installer is ready to install Flowgorithm on your computer.
Click "Next" to start the installation.
Cancel
< Back
Next >

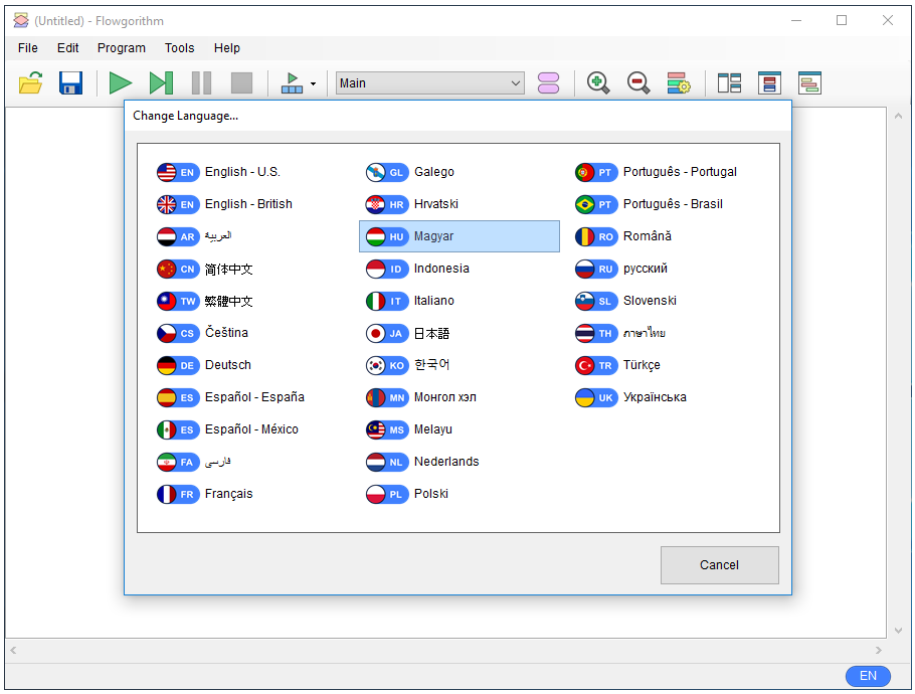

Flowgorithm
Installation Complete
Flowgorithm has been successfully installed.
Click "Close" to exit.
Please use Windows Update to check for any critical updates to the .NET Framework.
Cancel
< Back
Close

A telepítő varázsló utolsó lépése előtt az operációs rendszer megkérdezi, hogy engedélyezzük-e a szoftvernek, hogy módosításokat hajtson végre a számítógépen. Erre a kérdésre természetesen igen a válasz, annak ellenére, hogy a felugró ablakban a telepítendő szoftver gyártója ismeretlen. A sikeres telepítést követően megjelenik a startmenüben egy parancsikon, illetve az asztalon is elhelyezhető egy parancsikon, mely a Flowgorithm indító állományára mutat.

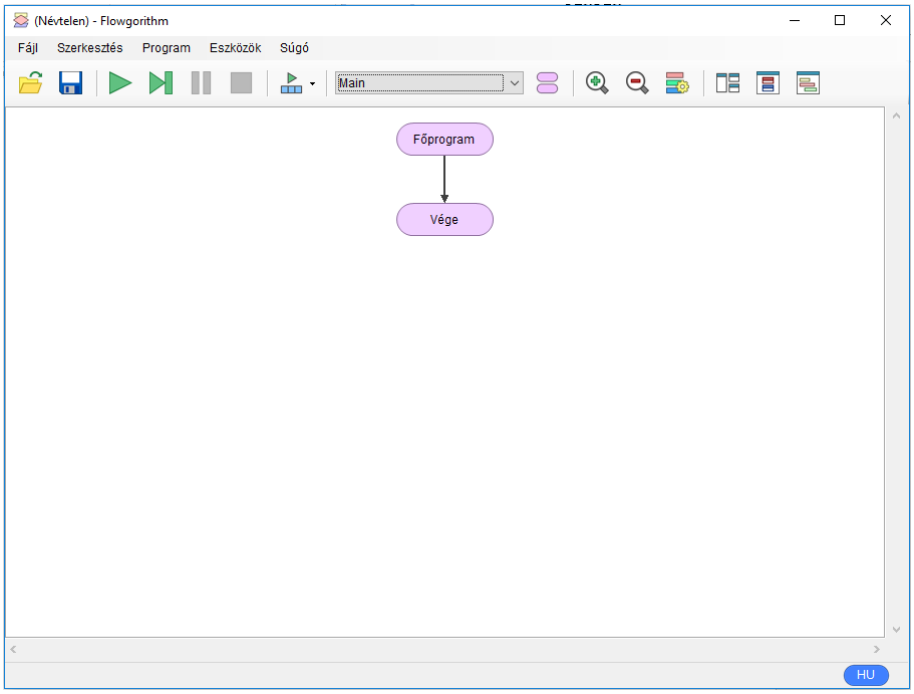



A program első indításakor automatikusan felugrik egy ablak, melyen kiválasztható a kívánt nyelv. Az aktuális verzióban 30 nyelv közül választhatunk, melyek közözz a magyar is megtalálható. Természetesen ez csak ajánlás, de mivel az oktatás magyar nyelven történik, ezért célszerű a magyar nyelvet választani a programban is.



## A programhoz tartozó felhasználói interfész:




A felületen található egy menüsor és egy eszköztár. Az eszköztár elemei a következők:


létező algoritmus megnyitása


aktuális algoritmus mentése


aktuális algoritmus futtatásának eszközei

aktuális függvény


függvény létrehozása


nagyítás, kicsinyítés

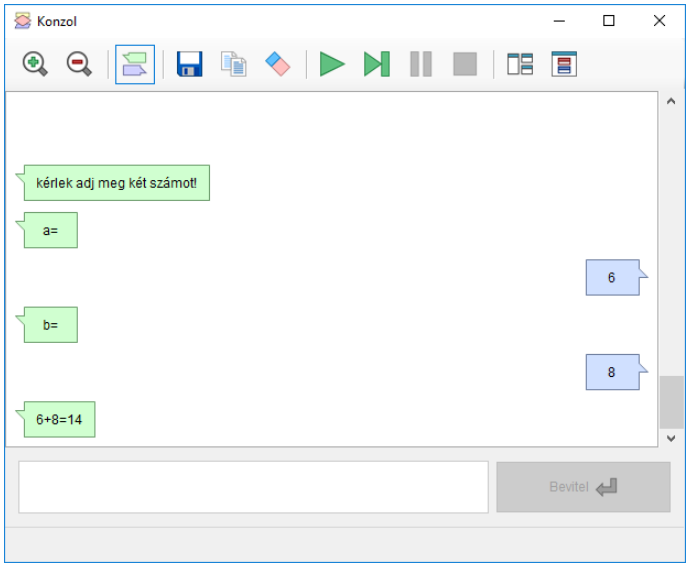

stílus módosítása

ablak elrendezése

változók figyelése

forráskód megtekintése

A program használatához szükség van még felhasználói felület további elemeinek az ismertetésére. Aktuális algoritmusunkat a  gomb lenyomásával tudjuk futtatni. A futtatás a Konzolon történik, mely felépítése alább látható.



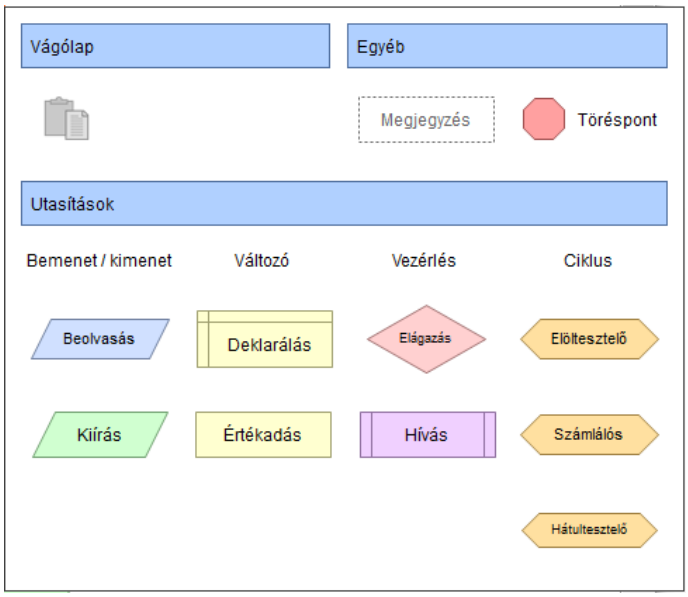

A fent említett eszközökön kívül két új eszköz található az ablakban:

váltás a buborékok, illetve a karakteres kijelző között

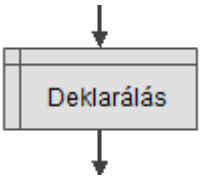
másolás

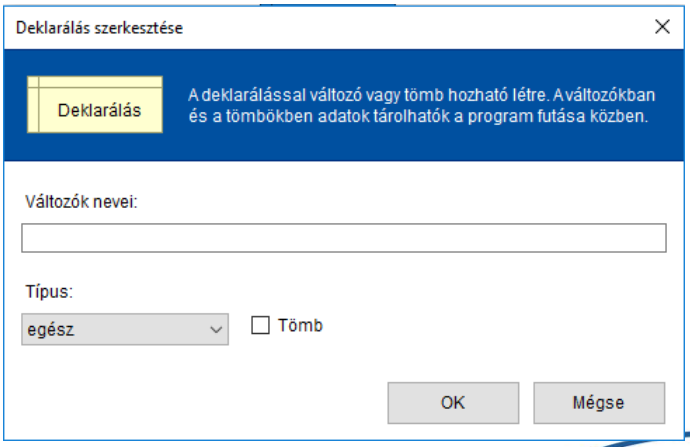
kijelző törlése

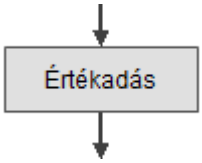
A folyamatábra elkészítése nagyon egyszerű. Az irányított gráf megfelelő élére kattintva a kattintás helyére tudunk újabb csomópontot beszúrni. Miután rákattintottunk az élre, megjelenik egy ablak, melyből a kívánt elem kiválasztható.

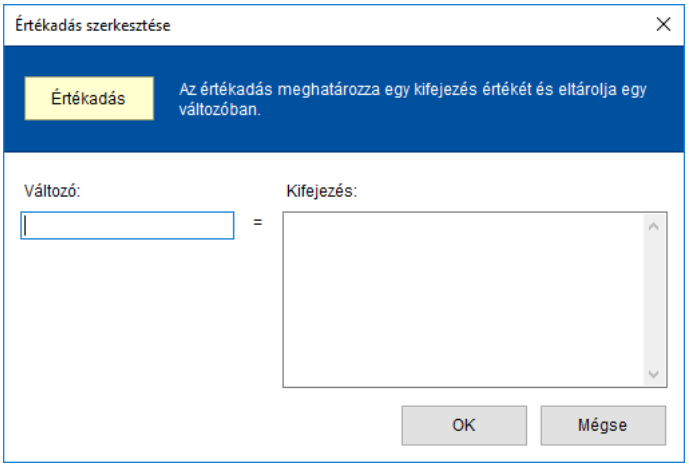


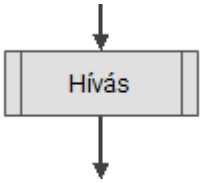

Miután kiválasztottuk a megfelelő csomópontot, a program a folyamatábrába elhelyezi az üres csomópontot, tartalommal való feltöltéshez duplán kattintunk a csomópontra, majd a megjelenő ablakokban a megfelelő szövegdobozok kitöltése után megjelenik a kívánt elem a folyamatábrában.

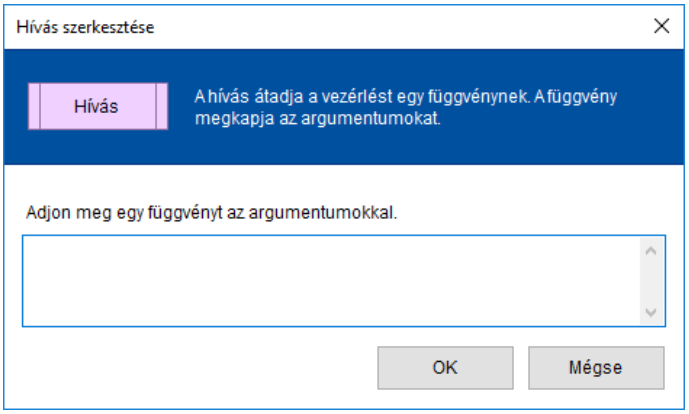


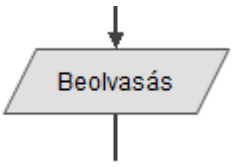



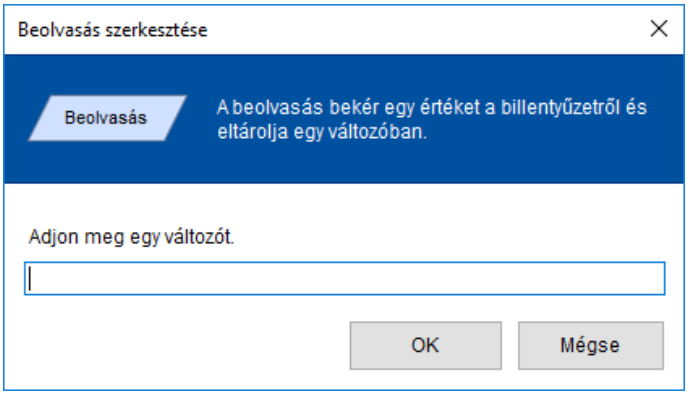

Értékadás

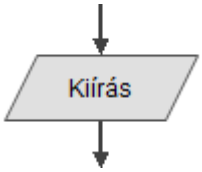

Értékadás szerkesztése
Értékadás
Az értékadás meghatározza egy kifejezés értékét és eltárolja egy változóban.
Változó:
=
Kifejezés:
OK
Mégse

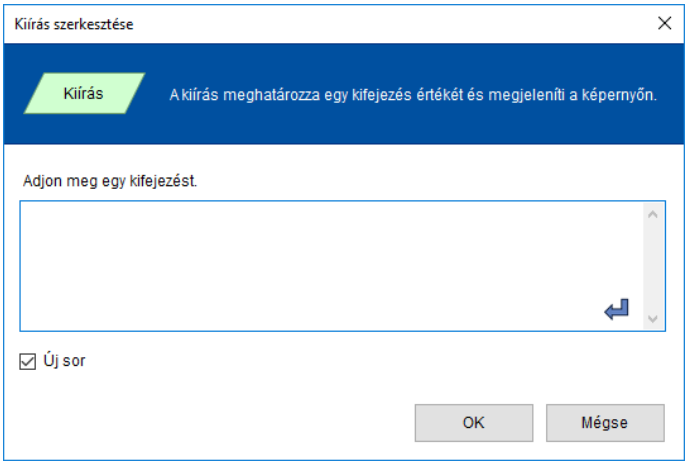

Hívás

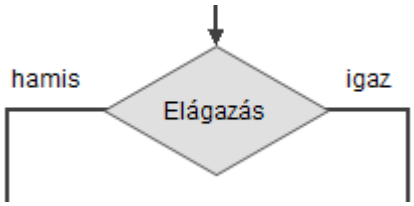

Hívás szerkesztése
Hívás
A hívás átadja a vezérlést egy függvénynek. A függvény megkapja az argumentumokat.
Adjon meg egy függvényt az argumentumokkal.
OK
Mégse

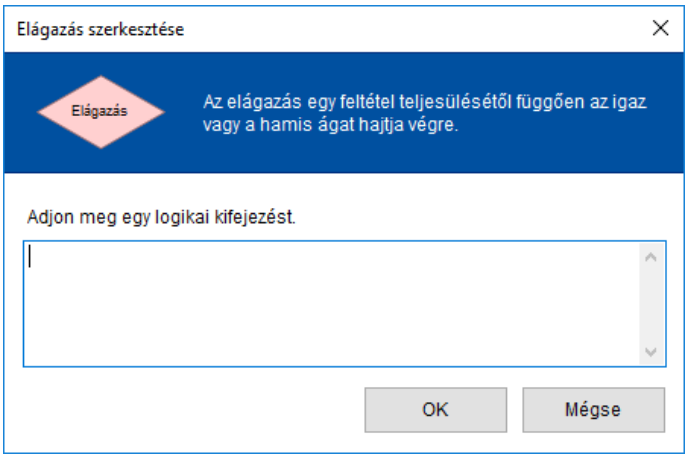

Beolvasás

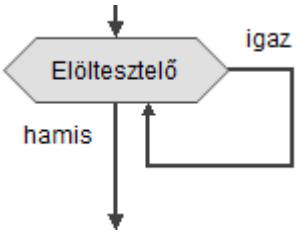

Beolvasás szerkesztése
Beolvasás
A beolvasás bekér egy értéket a billentyűzetről és eltárolja egy változóban.
Adjon meg egy változót.
OK
Mégse

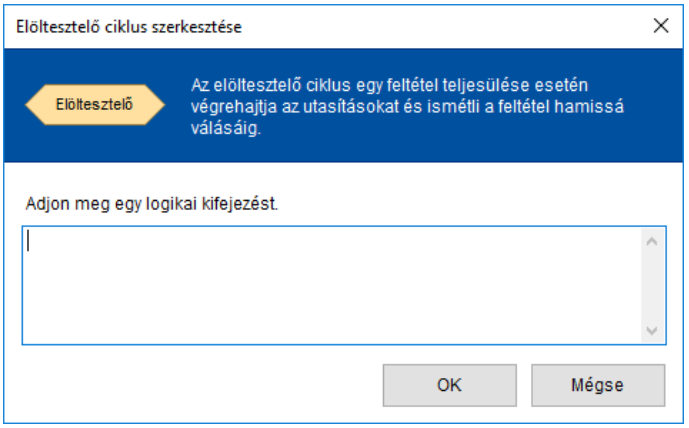

Kiírás

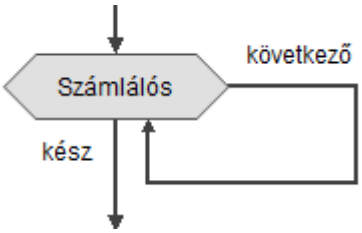

Kiírás szerkesztése
Kiírás
A kiírás meghatározza egy kifejezés értékét és megjeleníti a képernyőn.
Adjon meg egy kifejezést.
Új sor
OK
Mégse

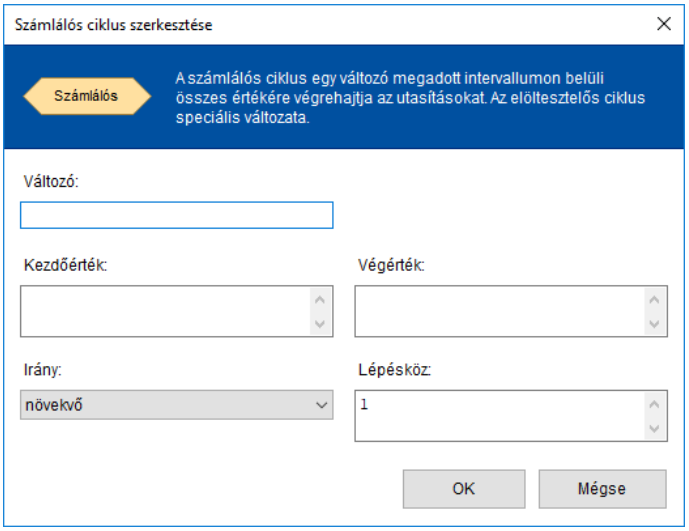

hamis
Elágazás
igaz

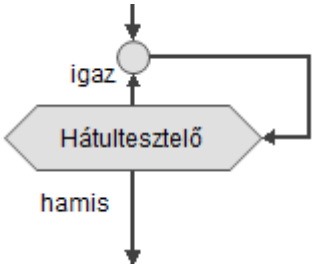

Elágazás szerkesztése
Elágazás
Az elágazás egy feltétel teljesülésétől függően az igaz vagy a hamis ágat hajtja végre.
Adjon meg egy logikai kifejezést.
OK
Mégse

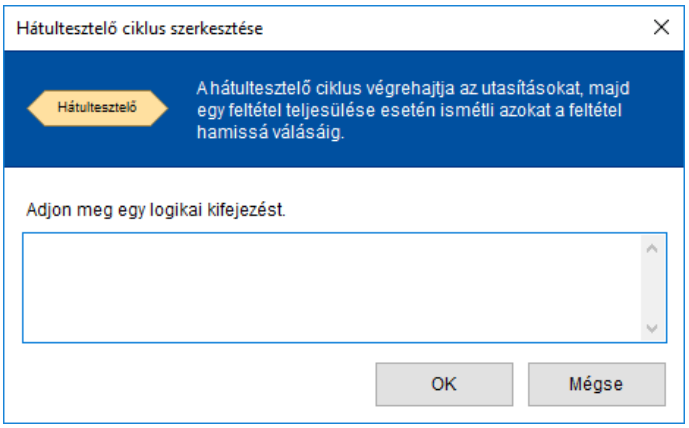

Elöltesztelő
igaz
hamis

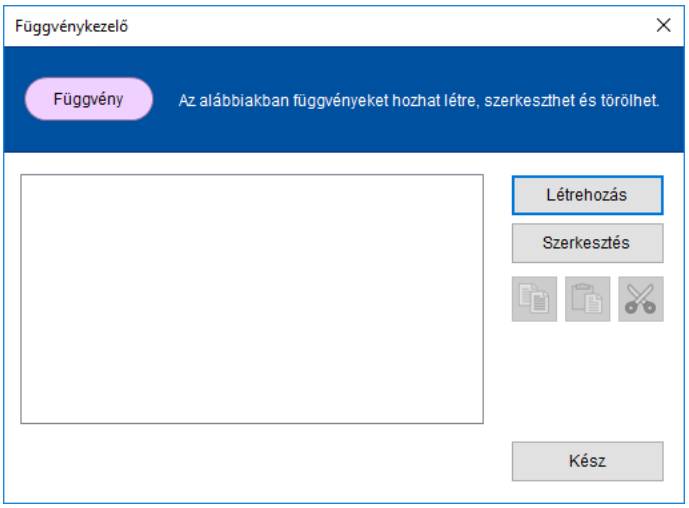

Elöltesztelő ciklus szerkesztése
Elöltesztelő
Az elöltesztelő ciklus egy feltétel teljesülése esetén végrehajtja az utasításokat és ismétli a feltétel hamissá válásáig.
Adjon meg egy logikai kifejezést.
OK
Mégse


következő
Számlálós
kész


Számlálós ciklus szerkesztése
Számlálós
A számlálós ciklus egy változó megadott intervallumon belüli összes értékére végrehajtja az utasításokat. Az elöltesztelős ciklus speciális változata.
Változó:
Kezdőérték:
Végérték:
Irány:
növekvő
Lépésköz:
1
OK
Mégse


igaz
Hátultesztelő
hamis


Hátultesztelő ciklus szerkesztése
Hátultesztelő
A hátultesztelő ciklus végrehajtja az utasításokat, majd egy feltétel teljesülése esetén ismétli azokat a feltétel hamissá válásáig.
Adjon meg egy logikai kifejezést.
OK
Mégse


Függvénykezelő
Függvény
Az alábbiakban függvényeket hozhat létre, szerkeszthet és törölhet.
Létrehozás
Szerkesztés
Kész

A különböző csomópontokban különböző előre definiált műveleteket, illetve operátorokat használhatunk. A flowgorithm dokumentációjában a következő táblázatokat találhatjuk:

Operátorok:

| Operator | C Family | BASIC Family |
|---|---|---|
| Negation | ! | not |
| Modulo | % | mod |
| Equality | == | = |
| Inequality | != | <> |
| Logical And | && | and |
| Logical Or | \|\| | or |

| Visual Basic Operator | Name |
|---|---|
| & | String Concatenation |
| ^ | Exponent |

| Level | Name | Operators | Notes |
|---|---|---|---|
| 8 | Unary | - ! not | In Visual Basic, "not" precedence level is far lower - above "and", but below all relational operators. |
| 7 | Exponent | ^ | The exponent operator does not exist in C# or Java. |
| 6 | Multiply | * / % mod | Division will always be high-precision (floating point) |
| 5 | Addition | + - | In Flowgorithm, "+" will only work with numbers. |
| 4 | Concatenate | & | C# and Java use the ambiguous "+" operator for addition and concatenation. |
| 3 | Relational | > >= < <= == = != <> | |
| 2 | Logical And | and && | |
| 1 | Logical Or | or \|\| | |

| Expression | Result | Notes |
|---|---|---|
| 1 + 3 ^ 2 | 10 | |
| 10 * 2 + 5 * 6 | 50 | 10 * 2 and 5 * 6 have higher precedence than addition. The addition is done last. |
| 7 * (4 - 1) | 21 | Parenthesis are used for subexpressions, which are evaluated as a whole. |
| 6 / 3 * 2 | 4 | In mathematics, multiplication and division have the same precedence levels. So, they are evaluated left-to-right. The "PEMDAS" acronym, used in high-school, is a tad misleading. |
| 10 mod 3 | 1 | Modulo math gives the remainder from division |
| 10 % 3 | 1 | Same expression, but using the C-Family operator |

Függvények:

### Mathematics

| Function | Description |
|---|---|
| `Abs(n)` | Absolute Value |
| `Arcsin(n)` | Trigonometric Arcsine |
| `Arccos(n)` | Trigonometric Arccos |
| `Arctan(n)` | Trigonometric Arctangent |
| `Cos(n)` | Trigonometric Cosine |
| `Int(n)` | Integral (whole value) of a real number |
| `Log(n)` | Natural Log |
| `Log10(n)` | Log Base 10 |
| `Sgn(n)` | Mathematical sign (-1 if ***n*** is negative, 0 if zero, 1 if positive) |
| `Sin(n)` | Trigonometric Sine |
| `Sqrt(n)` | Square Root |
| `Tan(n)` | Trigonometric Tangent |

### Strings

| Function | Description |
|---|---|
| `Len(s)` | Length of a string |
| `Char(s, i)` | Returns a character from the string ***s*** at index ***i***. Characters are indexed starting at 0. |

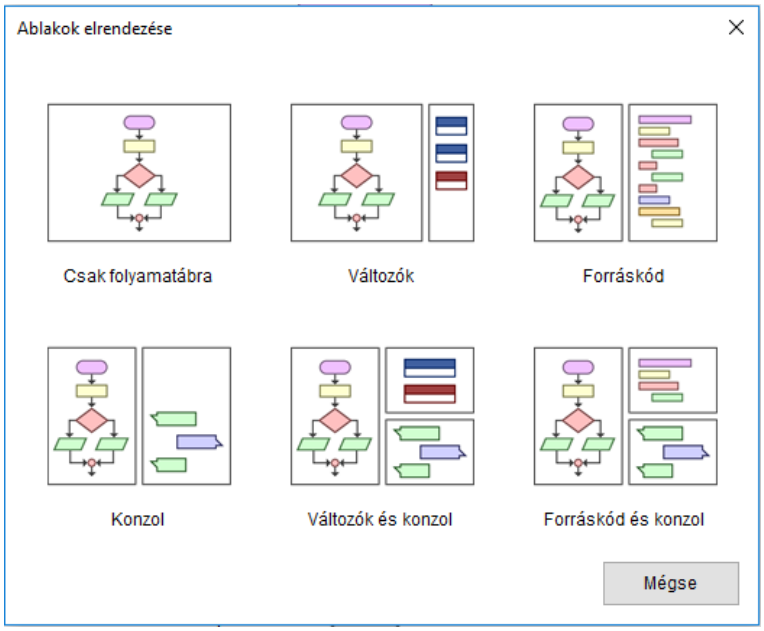
### Data Type Conversion

| Function | Description |
|---|---|
| `ToChar(n)` | Convert a character code ***n*** into a character. |
| `ToCode(c)` | Convert a character ***c*** into a character code (integer). |
| `ToFixed(r, i)` | Convert real number ***r*** to a string with ***i*** digits after the decimal point. This function is useful for currency. |
| `ToInteger(n)` | Convert a string to an integer |
| `ToReal(n)` | Convert a string to an real |
| `ToString(n)` | Convert a number to a string |

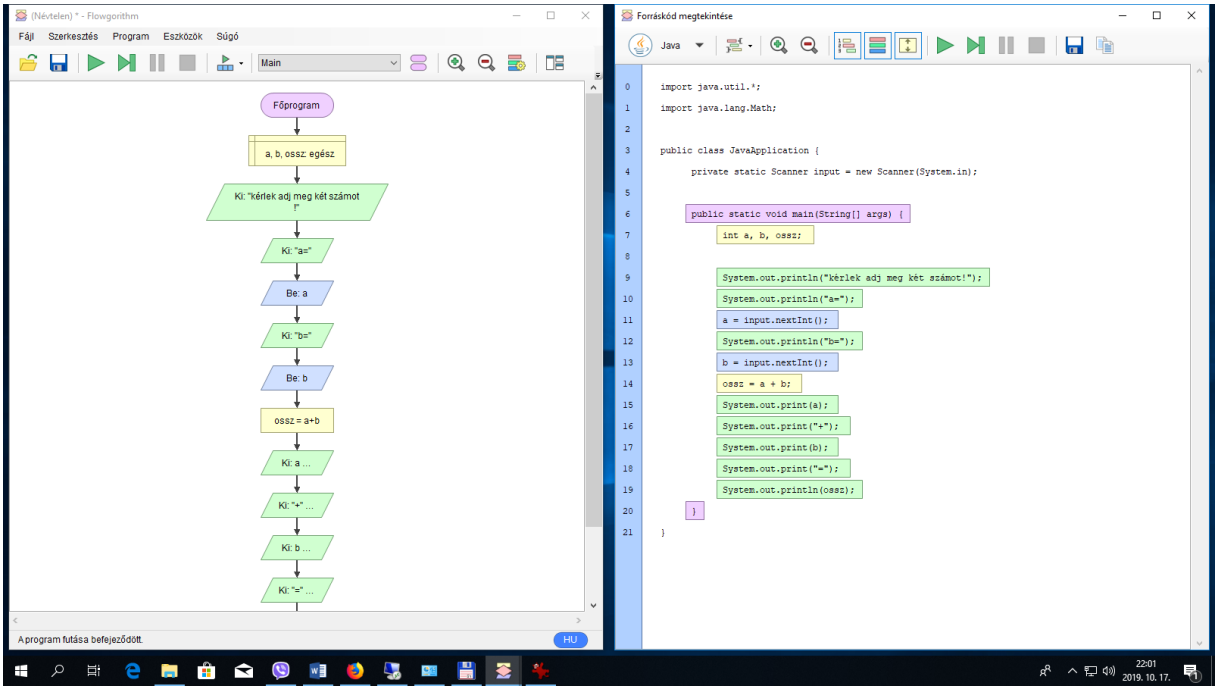
### Other

| Function | Description |
|---|---|
| `Random(n)` | A random number between 0 and (***n*** - 1) |
| `Size(a)` | The size (number of elements) in an array |

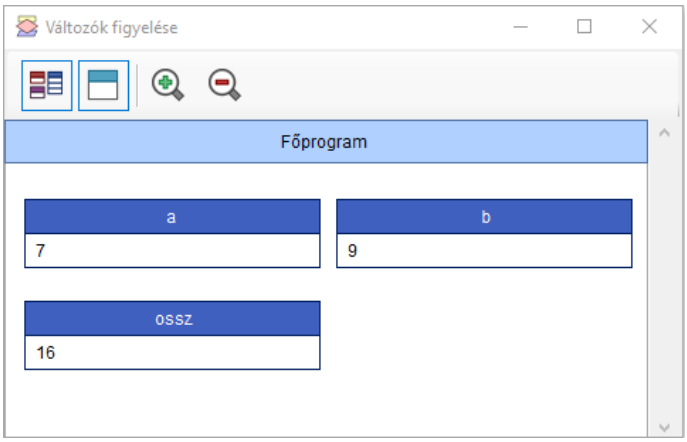
A feladatok algoritmusának elkészítésénél általában osztott ablakot használunk, több lehetőség közül választhatunk:



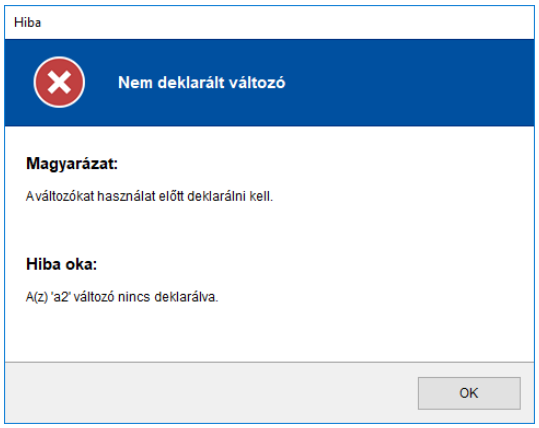

Az esetek döntő többségében a forráskód nézetet választva dolgozunk, ahol a folyamatábra mellett a választott programozási nyelvben írt kód látható. A választható programozási nyelvek listája, valamint az osztott ablak alább látható.

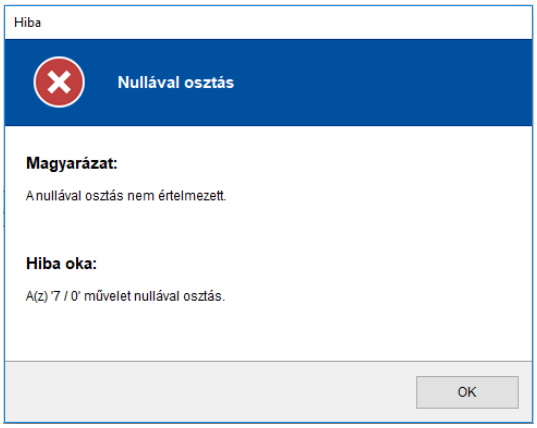



Lehetőségünk van továbbá a változók értékeinek nyomon követésére. Az algoritmusok tesztelésénél nélkülözhetetlen, hogy a pillanatnyi változó értékék tükrében elemezzük a megoldásunkat.




Az algoritmusok írása közben, legyen bármennyire körültekintő a tervezés, előfordulhatnak hibák, melyek lehetnek szintaktikai hibák, ismeretlen, nem deklarált változó használatából eredő hibák, hibás függvényhívások, hibás végre nem hajtható műveletek. A program természetesen minden hibáról tájékoztat, a megfelelő hibaüzenettel.




Összegezve a Flowgorithm program egy remek eszköz az algoritmikus gondolkodás fejlesztésére. Rengeteg előnye mellett a hátrányairól is ejtenénk pár szót. A nagyobb összetettebb programok már átláthatatlanok, folyamatábrájuk nagyméretű és összetett, teljes algoritmushoz tartozó folyamatábra sokszor meg sem jeleníthető egy képernyőnyi oldalon. Másik nagy hiányossága, hogy nem kezeli a kétdimenziós tömböket, melyek kezeléséhez tartozó folyamatábra ugyan elkészíthető, de nem futtatható, így nem tesztelhető.

## Feladatok megoldása Flowgorithm segítségével.

Lineáris egyenlet megoldása

**1. Feladat. Adjuk meg az $ax + b = cx + d$ alakú egyenletet algoritmikus megoldását!**

Mint minden feladat megoldását, ezt is elemzéssel kezdjük, majd következik a tervezés és megvalósítás fázisa.

Vizsgáljuk az $ax + b = cx + d$ egyenlet megoldását!

$$\begin{aligned} ax + b &= cx + d && / - cx \\ ax - cx + b &= d && / - b \\ ax - cx &= d - b \\ (a - c)x &= d - b && /:(a - c) \\ x &= \frac{d - b}{a - c} \end{aligned}$$

A megoldás közben végig változókkal (paraméterekkel) dolgoztunk, és az utolsó lépésben végrehajtottunk egy osztást. Tudjuk, hogy nullával való osztás nincs értelmezve, így szükséges az osztó vizsgálata, azaz $a - c \neq 0$ feltétel szükséges a művelet végrehajtásához.

Tekintsük az $a - c \neq 0$ feltételt:

$$a - c \neq 0$$
$$a \neq c$$

Vizsgáljuk azokat az eseteket, amikor $a = c$, azaz ha az ismeretlenek együtthatói egyenlők, akkor milyen esetek fordulhatnak elő (a vizsgálódást konkrét egyenleteken végezzük). Két eset lehetséges:

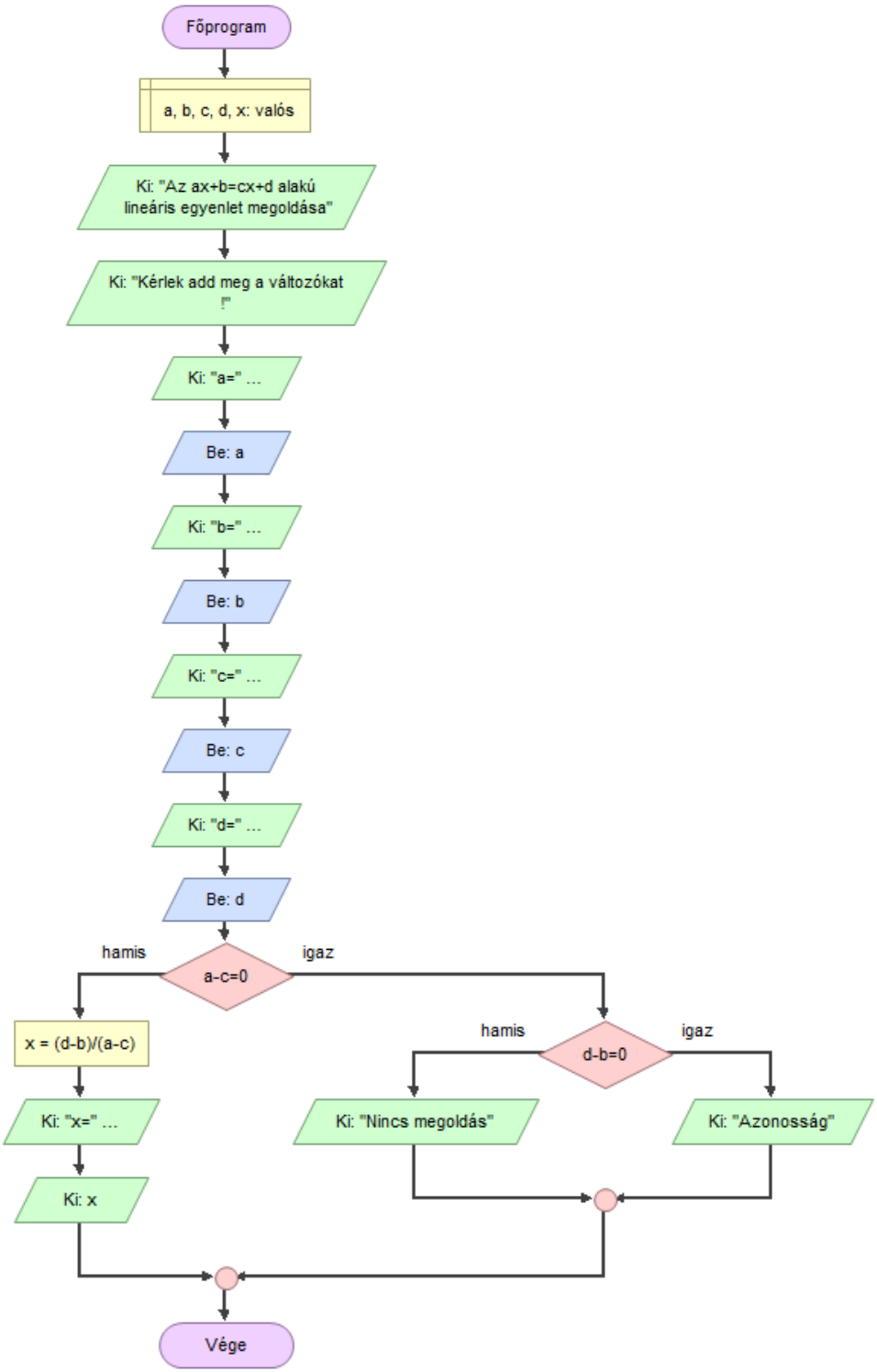
$$3x - 2 = 3x + 6$$
$$-2 = 6$$
Ellentmondás nincs megoldás

$$3x - 2 = 3x - 2$$
$$-2 = -2$$
Azonosság

A konkrét esetek vizsgálatából kiderül, hogy az ismeretlenek együtthatóinak vizsgálata mellett a konstansok vizsgálatára is szükség van. Az elemzés után tervezzük meg az algoritmust!

1. Az együtthatók és konstansok bekérése
2. Az $a = c$ feltétel vizsgálata:
    a. igaz ágban a $b = d$ feltétel vizsgálata:
        i. igaz ágban: Azonosság
        ii. hamis ágban: Ellentmondás
    b. hamis ágban: $x = (d - b)/(a - c)$
3. Az eredmény kiíratása

Az algoritmushoz tartozó folyamatábra Flowgorithm-ben elkészítve:


Főprogram
a, b, c, d, x: valós
Ki: "Az ax+b=cx+d alakú lineáris egyenlet megoldása"
Ki: "Kérlek add meg a változókat !"
Ki: "a=" ...
Be: a
Ki: "b=" ...
Be: b
Ki: "c=" ...
Be: c
Ki: "d=" ...
Be: d
hamis
a-c=0
igaz
x = (d-b)/(a-c)
Ki: "x=" ...
Ki: x
hamis
d-b=0
igaz
Ki: "Nincs megoldás"
Ki: "Azonosság"
Vége

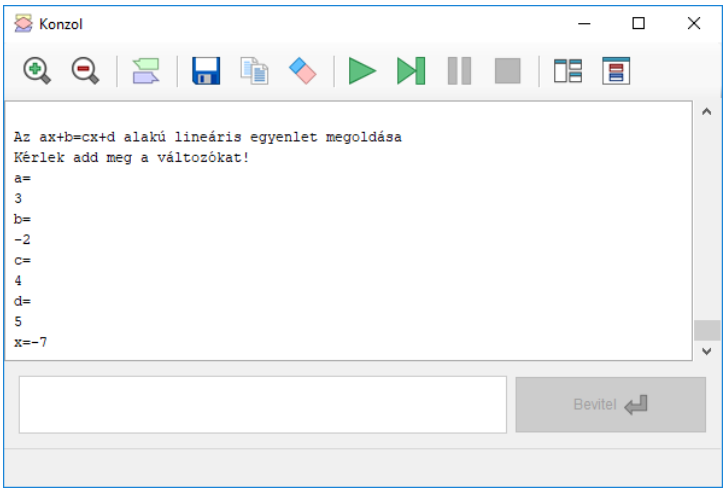
A program által generált JAVA forráskód:

```
import java.util.*;
import java.lang.Math;
import java.util.Scanner;

class JavaApplication {
      private static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
          double a, b, c, d, x;

          System.out.println("Az ax+b=cx+d alakú lineáris egyenlet megoldása");
          System.out.println("Kérlek add meg a változókat!");
          System.out.print("a=");
          a = input.nextDouble();
          System.out.print("b=");
          b = input.nextDouble();
          System.out.print("c=");
          c = input.nextDouble();
          System.out.print("d=");
          d = input.nextDouble();
          if (a - c == 0) {
              if (d - b == 0) {
                  System.out.println("Azonosság");
              } else {
                  System.out.println("Nincs megoldás");
              }
          } else {
              x = d - b / (a - c);
              System.out.print("x=");
              System.out.println(x);
          }
    }
}
```

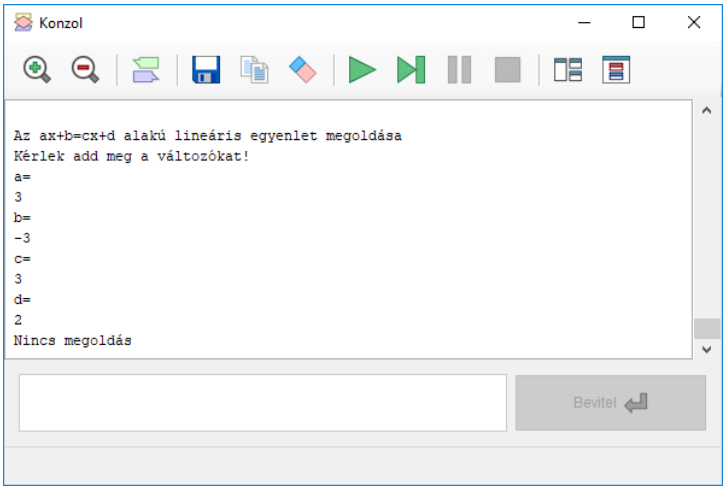
Jól látható, hogy az egymásnak megfelelő programrészletek azonos színnel vannak jelölve mind a folyamatábrában, mind a forráskódban. Minden esetben szükséges az elkészült algoritmus tesztelése, minden lehetséges input esetén, azaz jelen esetben a tesztelendő esetek a következők:

1. $a \neq c$ és $b \neq d$ $pl.: 3x - 2 = 4x + 5$
2. $a = c$ és $b \neq d$ $pl.: 3x - 3 = 3x + 2$
3. $a = c$ és $b = d$ $pl.: 3x + 1 = 3x + 1$

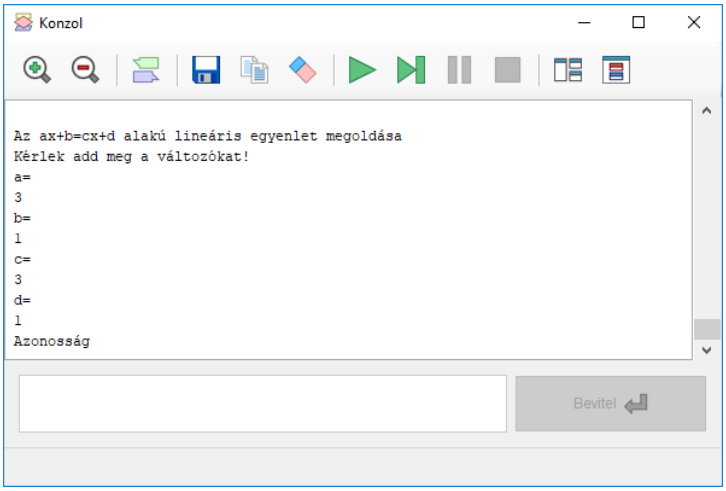
1.




2.



3.




## 2. Feladat. Adjuk meg az $ax^2 + bx + c = 0$ alakú egyenletet algoritmikus megoldását!

Vizsgáljuk az $ax^2 + bx + c = 0$ egyenlet megoldását!

Az világos, hogy az egyenlet csak akkor másodfokú, ha $a \neq 0$. Ellenkező esetben az egyenlet lineáris, melynek a megoldását az előző feladatnál részletesen tárgyaltuk.

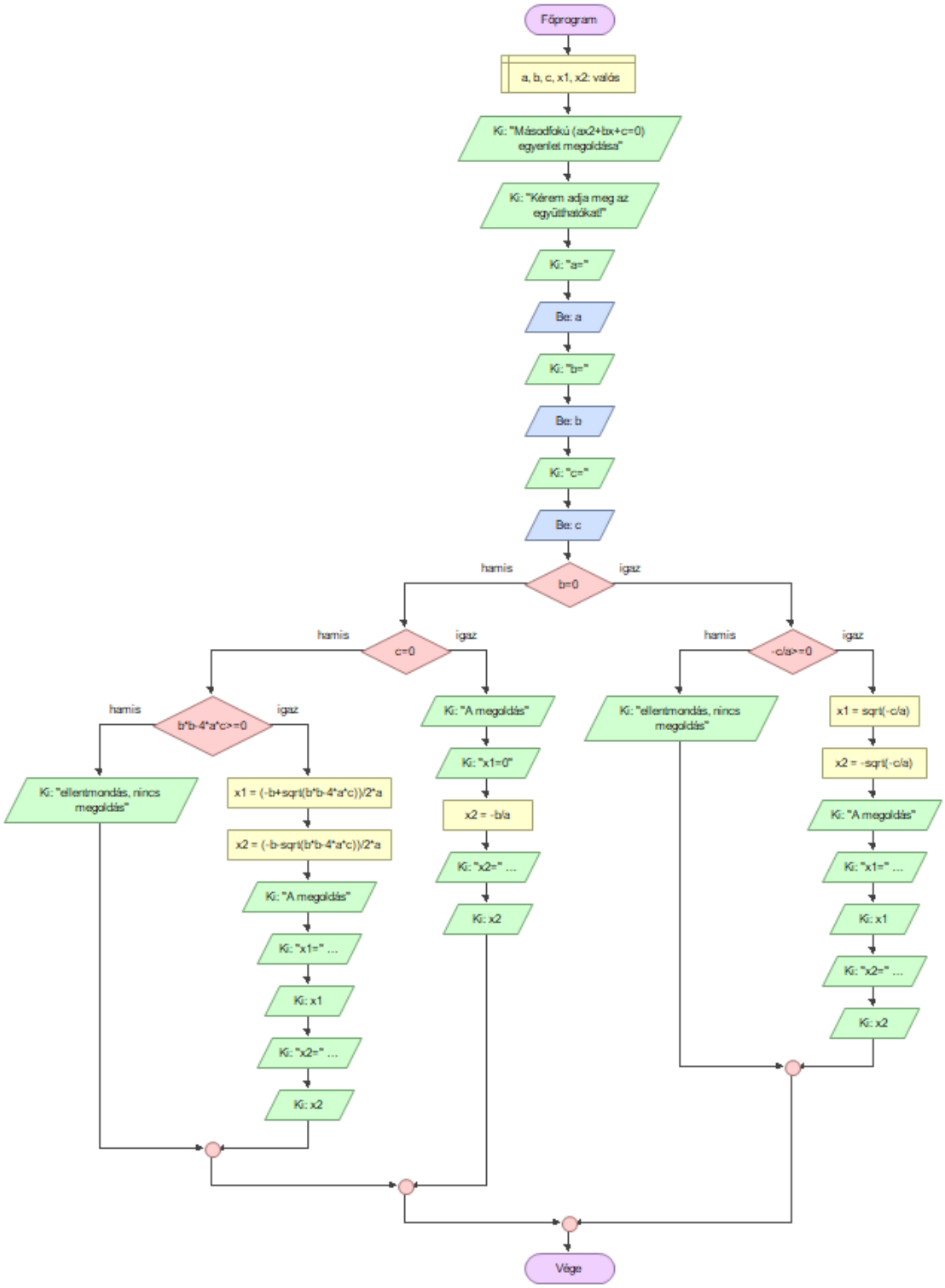
Másodfokú egyenlet

| Teljes | Hiányos | |
|---|---|---|
| Ha, $a \neq 0; b \neq 0; c \neq 0$, akkor | $a \neq 0; b = 0; c \neq 0$ | $a \neq 0; b \neq 0; c = 0$ |
| $ax^2 + bx + c = 0$ | $ax^2 + c = 0$ | $ax^2 + bx = 0$ |
| ha $b^2 - 4ac \geq 0$, akkor | Megoldás: | Megoldás: |
| Megoldó képlet: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ | $ax^2 + c = 0$<br>$ax^2 = -c$<br>$x^2 = \frac{-c}{a}$<br>ha $\frac{-c}{a} \geq 0 \Rightarrow x_{1;2} = \pm\sqrt{\frac{-c}{a}}$ | $ax^2 + bx = 0$<br>$x(ax + b) = 0$<br>$\Updownarrow$<br>$x_1 = 0$ v. $ax + b = 0$<br>$ax = -b$<br>$x_2 = \frac{-b}{a}$ |

A lehetséges esetek vizsgálata közben látható, hogy több feltétellel számolnunk kell, vizsgálnunk kell az együtthatókat, továbbá az egyes eseteknél további feltételekkel találkozhatunk, mint például a diszkrimináns vizsgálata. Az előző feladatnál alkalmazott módszer segítségével tervezzük meg az algoritmust!

1. Az együtthatók bekérése
2. Az $b = 0$ feltétel vizsgálata:
    a. igaz ágban a $-\frac{c}{a} \geq 0$ feltétel vizsgálata:
        i. igaz ágban: $x_{1;2} = \pm\sqrt{-\frac{c}{a}}$
        ii. hamis ágban: Ellentmondás, nincs megoldás
    b. hamis ágban: $c = 0$ feltétel vizsgálata:
        i. igaz ágban: $x_1 = 0; x_2 = -\frac{b}{a}$
        ii. hamis ágban a $b^2 - 4ac \geq 0$ feltétel vizsgálata:
            1. igaz ágban: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
            2. hamis ágban: Ellentmondás, nincs megoldás
3. Az eredmény kiíratása

Az algoritmushoz tartozó folyamatábra Flowgorithm-ben elkészítve, valamint a generált JAVA forráskód:


Főprogram
a, b, c, x1, x2: valós
Ki: "Másodfokú (ax2+bx+c=0) egyenlet megoldása"
Ki: "Kérem adja meg az együtthatókat!"
Ki: "a="
Be: a
Ki: "b="
Be: b
Ki: "c="
Be: c
b=0
hamis
igaz
c=0
-c/a>=0
b*b-4*a*c>=0
Ki: "A megoldás"
Ki: "x1=0"
x2 = -b/a
Ki: "x2=" ...
Ki: x2
Ki: "ellentmondás, nincs megoldás"
x1 = (-b+sqrt(b*b-4*a*c))/2*a
x2 = (-b-sqrt(b*b-4*a*c))/2*a
Ki: "A megoldás"
Ki: "x1=" ...
Ki: x1
Ki: "x2=" ...
Ki: x2
x1 = sqrt(-c/a)
x2 = -sqrt(-c/a)
Ki: "A megoldás"
Ki: "x1=" ...
Ki: x1
Ki: "x2=" ...
Ki: x2
Vége

A program által generált JAVA forráskód:

```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
    private static Scanner input = new Scanner(System.in);
    public static void main(String[] args) {
        double a, b, c, x1, x2;

        System.out.println("Másodfokú (ax2+bx+c=0) egyenlet megoldása");
        System.out.println("Kérem adja meg az együtthatókat!");
        System.out.println("a=");
        a = input.nextDouble();
        System.out.println("b=");
        b = input.nextDouble();
        System.out.println("c=");
        c = input.nextDouble();
        if (b == 0) {
            if (-c / a >= 0) {
                x1 = Math.sqrt(-c / a);
                x2 = -Math.sqrt(-c / a);
                System.out.println("A megoldás");
                System.out.print("x1=");
                System.out.println(x1);
                System.out.print("x2=");
                System.out.println(x2);
            } else {
                System.out.println("ellentmondás, nincs megoldás");
            }
        } else {
            if (c == 0) {
                System.out.println("A megoldás");
                System.out.println("x1=0");
                x2 = -b / a;
                System.out.print("x2=");
                System.out.println(x2);
            } else {
                if (b * b - 4 * a * c >= 0) {
                    x1 = (-b + Math.sqrt(b * b - 4 * a * c)) / 2 * a;
                    x2 = (-b - Math.sqrt(b * b - 4 * a * c)) / 2 * a;
                    System.out.println("A megoldás");
                    System.out.print("x1=");
                    System.out.println(x1);
                    System.out.print("x2=");
                    System.out.println(x2);
                } else {
                    System.out.println("ellentmondás, nincs megoldás");
                }
            }
        }
    }
```

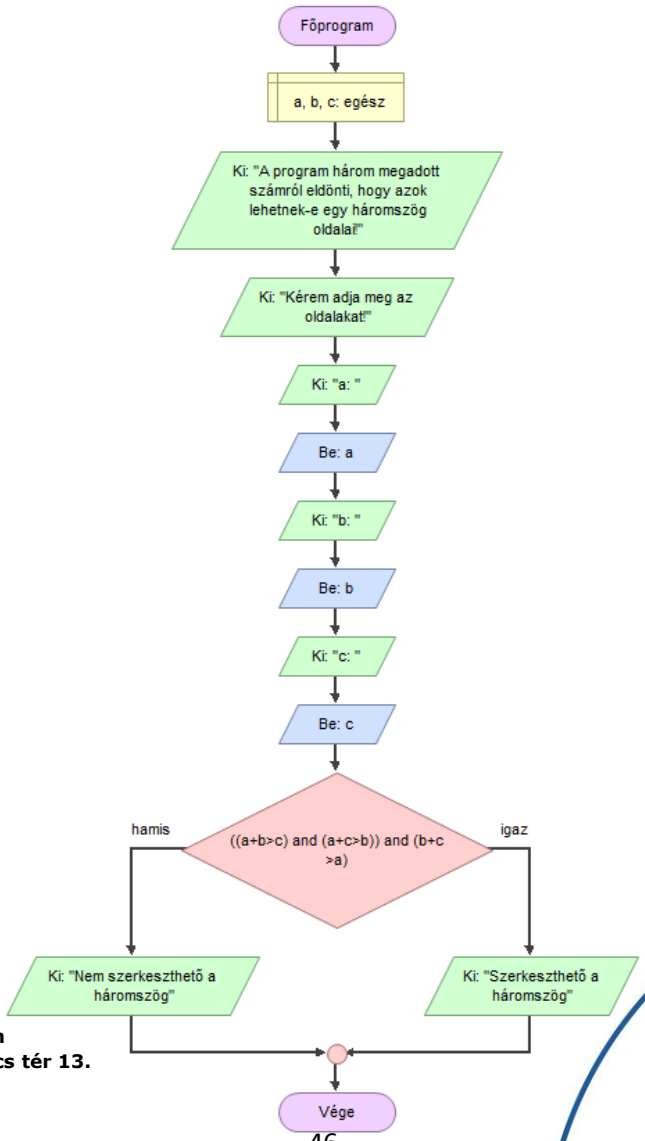
Az algoritmus tesztelését a hallgatókra bízzuk!

Az 1. számú mellékletben megtalálható az egyik hallgató munkája, ahol a fenti két feladatot ötvözte.

**3. Feladat. Adjon meg egy algoritmust, amely három egész számról eldönti, hogy ha az egész számok szakaszok hosszát jelölik, akkor a megadott adatokból szerkeszthető-e háromszög.**

A megoldás megadása előtt vizsgáljuk a feladatot. Háromszög akkor szerkeszthető, ha teljesül az úgynevezett háromszög-egyenlőtlenség, azaz ha a háromszög oldalait rendre *a,b,c*-vel jelöljük, akkor::

$$a + b > c$$
$$a + c > b$$
$$b + c > a$$



A program által generált JAVA forráskód:

```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
      private static Scanner input = new Scanner(System.in);

      public static void main(String[] args) {
            int a, b, c;

            System.out.println("A program három megadott számról eldönti, hogy azok lehetnek-e egy háromszög oldalai!");
            System.out.println("Kérem adja meg az oldalakat!");
            System.out.println("a: ");
            a = input.nextInt();
            System.out.println("b: ");
            b = input.nextInt();
            System.out.println("c: ");
            c = input.nextInt();
            if (a + b > c && a + c > b && b + c > a) {
                  System.out.println("Szerkeszthető a háromszög");
            } else {
                  System.out.println("Nem szerkeszthető a háromszög");
            }
      }
}
```

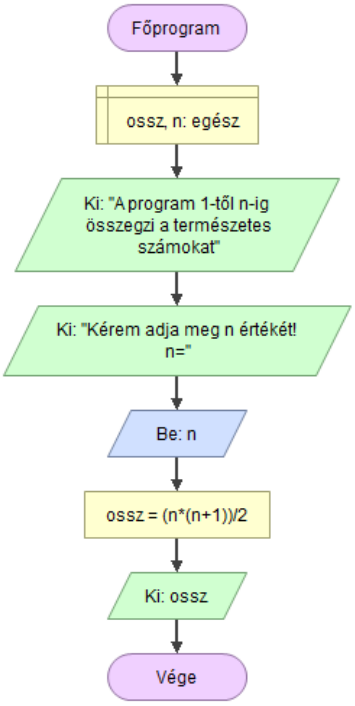
Az előző feladatokban a szekvenciális és szelekciós vezérlésre láttunk néhány példát, a következő feladatokkal az iterációs vezérléshez tartozó eszközöket ismertetjük. Elsőként egy egyszerű feladaton keresztül bemutatjuk a megszámlálós, elöltesztelő és hátultesztelő ciklusokat.

**4. Feladat. Adjunk meg egy algoritmust, amely összegzi n-ig (n is természetes szám) a természetes számokat!**

1. megoldás:

Az összeadandó számsorozat tekinthető számtani sorozatnak, melynek kezdő eleme: 1, utolsó eleme: *n*, elemek száma: *n*, differenciája 1. Alkalmazzuk rá a számtani sorozat összegképletét (Gauss):

$$ossz = \frac{n(n+1)}{2}$$

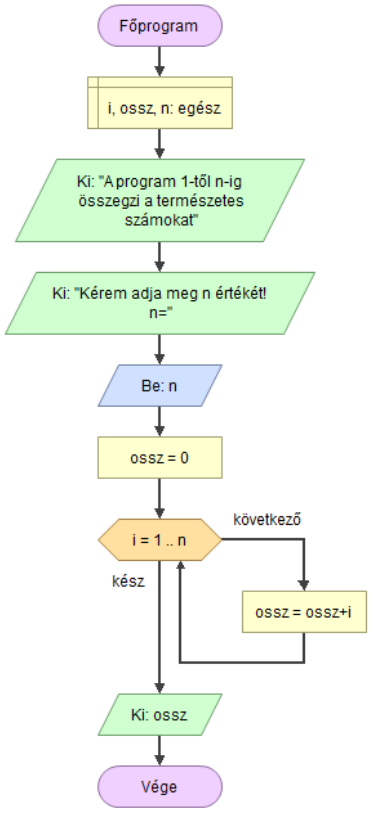



```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
      private static Scanner input = new Scanner(System.in);

      public static void main(String[] args) {
            int ossz, n;

            System.out.println("A program 1-től n-ig összegzi a természetes számokat");
            System.out.println("Kérem adja meg n értékét! n=");
            n = input.nextInt();
            ossz = (double) n * (n + 1) / 2;
            System.out.println(ossz);
      }
}
```

Előfordulhat azonban, hogy nem tudjuk, vagy nem jut eszünkbe az összegzési képlet, ilyenkor alkalmazhatjuk az iterációkat.

2. megoldás (for):

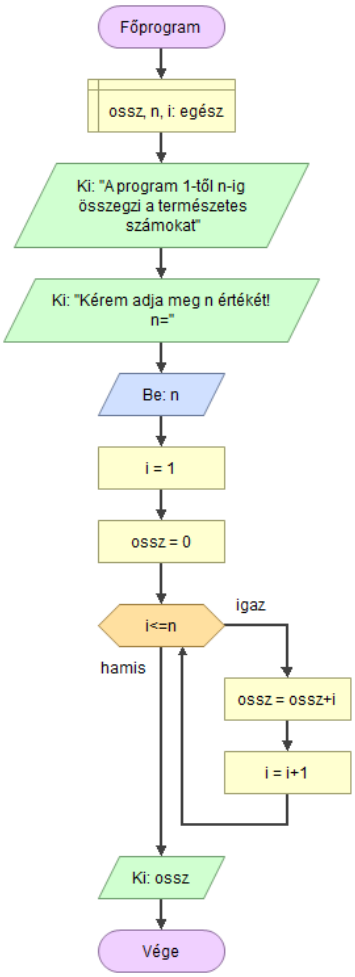
Lehetőségünk van számlálós ciklus alkalmazására, melyet akkor használunk, amikor pontosan tudjuk az ismétlések számát. A feladatot úgy is megoldhatjuk, hogy egy összeg változó kezdeti értékét nullára állítjuk, majd minden egyes számot sorra hozzáadunk *n*-ig.




```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
      private static Scanner input = new Scanner(System.in);

      public static void main(String[] args) {
            int ossz, n, i;

            System.out.println("A program 1-től n-ig összegzi a természetes számokat");
            System.out.println("Kérem adja meg n értékét! n=");
            n = input.nextInt();
            ossz = 0;
            for (i = 1; i <= n; i++) {
                  ossz = ossz + 1;
            }
            System.out.println(ossz);
      }
}
```

Alkalmazhatjuk a feltételhez kötött iterációs vezérléseket is, nevezetesen az elöltesztelő és hátultesztelő ciklusokat. A két lehetőség között az az alapvető különbség, hogy a hátultesztelő ciklus ciklusmagja a feltételtől függetlenül legalább egyszer végrehajtódik, míg elöltesztelő ciklusnál elképzelhető, hogy a ciklusmag egyetlen egyszer sem hajtódik végre. Mindkét esetben fontos, hogy a vizsgált feltételben szereplő változók értékét módosítsuk a ciklusmagon belül, ellenkező esetben a program lehet, hogy soha sem áll meg, úgynevezett végtelen ciklusba lép.

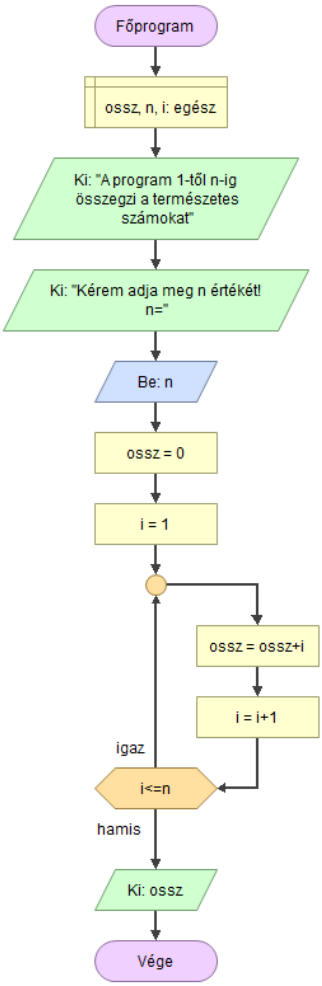
3. megoldás (while):




```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
      private static Scanner input = new Scanner(System.in);

      public static void main(String[] args) {
            int ossz, n, i;

            System.out.println("A program 1-től n-ig összegzi a természetes számokat");
            System.out.println("Kérem adja meg n értékét! n=");
            n = input.nextInt();
            i = 1;
            ossz = 0;
            while (i <= n) {
                  ossz = ossz + i;
                  i = i + 1;
            }
            System.out.println(ossz);
      }
}
```

A folyamatábráról leolvasható, hogy a ciklusváltozó, azaz az a változó, amely értékének függvényében végrehajtódik a ciklus vagy sem az az *i* változó. A 2. megoldásnál a ciklusváltozó kezdő értékét magában a ciklus fejrészében állítjuk be, továbbá az utoljára felvehető értéket, valamint a lépésközt is itt találjuk. Elöltesztelő és majd látni fogjuk hátultesztelő ciklus esetében is a cikluson kívül kell egy kezdőértéket beállítani a ciklusváltozónak, majd a ciklusmagon belül módosítani az értékét.

4. megoldás (do while):

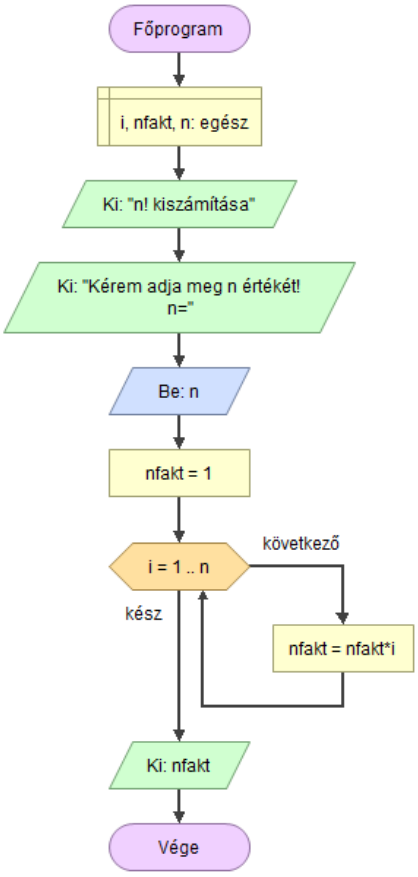



```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
      private static Scanner input = new Scanner(System.in);

      public static void main(String[] args) {
            int ossz, n, i;

            System.out.println("A program 1-től n-ig összegzi a természetes számokat");
            System.out.println("Kérem adja meg n értékét! n=");
            n = input.nextInt();
            ossz = 0;
            i = 1;
            do {
                  ossz = ossz + i;
                  i = i + 1;
            } while (i <= n);
            System.out.println(ossz);
      }
}
```

A feladatban összegeztük a ciklusváltozó értékeit egy változóban, melynek neve: ossz. A probléma vizsgálatánál tapasztaltuk, hogy ha a későbbiekben valamilyen összegzést kell végezni, mely során egy változó értékét módosítjuk, az adott változót mindig nulla értékkel kell inicializálnunk (ez mindhárom ciklusnál jól látható). Előfordulhat azonban, hogy nem összegeznünk kell, hanem egy szorzatot kell kiszámítani úgy, hogy a szorzat változó értékét többször változtatjuk. Az ilyen típusú értékadásokat, valamint a kezdeti érték meghatározását általában $n!$ kiszámításával szokták elmagyarázni, ahol $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1)n$. Matematikai ismereteinkre támaszkodva kijelenthetjük, hogy a kezdeti értéke a változónak nem lehet nulla, hiszen akkor a szorzat változó értékét bármilyen értékkel szorozva a szorzat továbbra is nulla marad. Tudjuk továbbá azt is, hogy ha egy számot eggyel szorzunk, az értéke

változatlan marad, továbbá figyelembe véve, hogy definíció szerint $0! = 1$, ha szorzatot tartalmazó változónak kezdeti értéket kell adni, akkor az mindig egy legyen.

**5. Feladat. Adjunk algoritmust n! kiszámítására!**

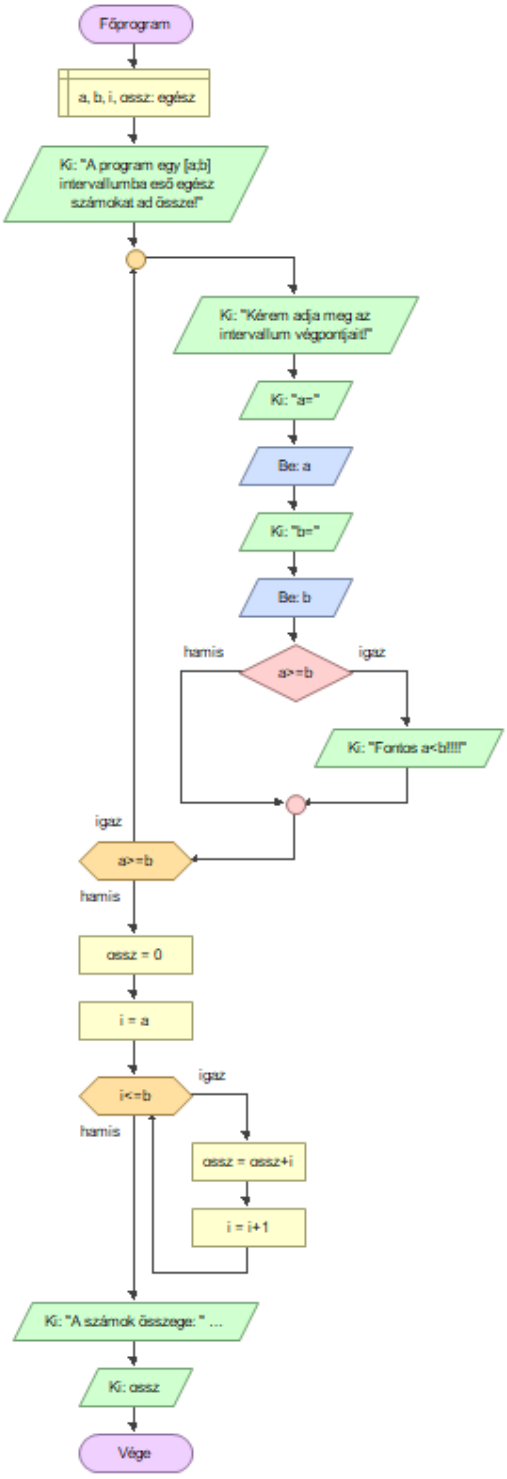



```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
      private static Scanner input = new Scanner(System.in);

      public static void main(String[] args) {
            int i, nfakt, n;

            System.out.println("n! kiszámítása");
            System.out.println("Kérem adja meg n értékét! n=");
            n = input.nextInt();
            nfakt = 1;
            for (i = 1; i <= n; i++) {
                  nfakt = nfakt * i;
            }
            System.out.println(nfakt);
      }
}
```

Az előzőekben tárgyalt vezérlési szerkezetek segítségével oldjuk meg a következő feladatot.

**6. Feladat. Kérjen be a felhasználótól két számot – a, b – melyek egy zárt intervallum kezdő és végpontjai. Adjon meg egy algoritmust, mely az intervallumba eső egész számokat összegzi!**

Vizsgáljuk az $a$, $b$ számokat! Mivel $a$ egy intervallum kezdő és $b$ az intervallum végpontjai, ezért szükséges, hogy $a < b$ feltétel teljesüljön. Mivel már elemeztük a hátultesztelő ciklusokat, ezért próbáljuk meg ennek az algoritmusnak az adatbekérő részében alkalmazni, és mindaddig újra és újra bekérni az $a$, $b$ számokat, míg az $a < b$ feltétel nem teljesül. Az összeg kiszámításához a 3. feladatban adott megoldást kell egy kicsit átgondolni.


Főprogram
a, b, i, assz: egész
Ki: "A program egy [a;b] intervallumba eső egész számokat ad összel"
Ki: "Kérem adja meg az intervallum végpontjait!"
Ki: "a="
Be: a
Ki: "b="
Be: b
hamis
a>=b
igaz
Ki: "Fontos a<b!!!!"
igaz
a>=b
hamis
assz = 0
i = a
i<=b
igaz
hamis
assz = assz+i
i = i+1
Ki: "A számok összege: " ...
Ki: assz
Vége

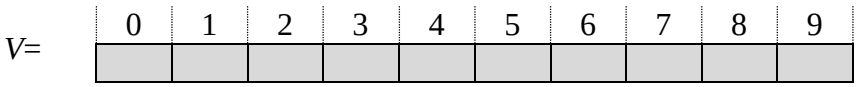
```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
     private static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
         int a, b, i, ossz;

         System.out.println("A program egy [a;b] intervallumba eső egész számokat ad össze!");
         do {
             System.out.println("Kérem adja meg az intervallum végpontjait!");
             System.out.println("a=");
             a = input.nextInt();
             System.out.println("b=");
             b = input.nextInt();
             if (a >= b) {
                 System.out.println("Fontos a<b!!!!");
             }
         } while (a >= b);
         ossz = 0;
         i = a;
         while (i <= b) {
             ossz = ossz + i;
             i = i + 1;
         }
         System.out.print("A számok összege: ");
         System.out.println(ossz);
    }
}
```

Hasonlóan a 3. feladat megoldásához ebben az esetben is szabadon választhatunk, hogy az összeg kiszámításához melyik ismétléses vezérlést választjuk. Általában elmondható, hogy vannak olyan feladatok, melyeket számlálós ciklussal szokás megoldani, és vannak olyanok, melyek csak feltételes ciklusokkal végezhetők el. A tömbök kezelése elképzelhetetlen (nagyon körülményes) az iteratív vezérlés nélkül. Ha visszagondolunk a tömbstruktúrára és jellemzőire, azonnal kiderül, hogy bármilyen tömbről legyen szó, bejárását mindig elemenként hajtjuk végre, azaz ugyanazt a műveletet ismételjük. Minden tömböket használó feladatban szükséges tömbök feltöltése, amit szintén elemenként kell megtenni.

**7. Feladat. Kérjük be egy n dimenziós vektor koordinátáit, majd írjuk ki a teljes vektort!**

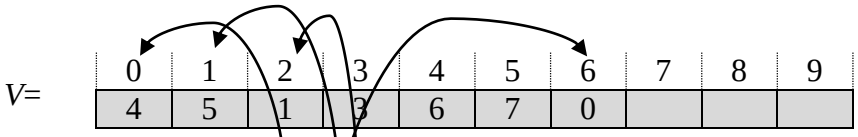
Vizsgáljuk az egydimenziós tömböket!

Deklarálunk egy *V* maximum 10 elemű egydimenziós tömböt. Ekkor a következő üres tömb jön létre:




tömb mérete 10

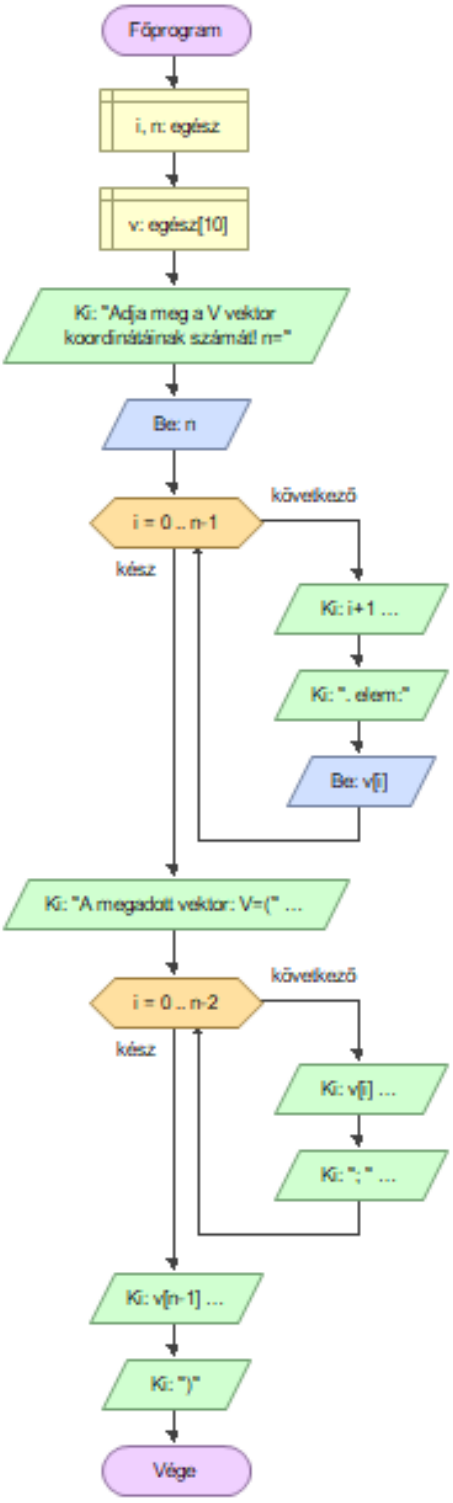
A programban meg kell adni a használni kívánt tömb méretét ($n \leq 10$), és el kell tárolni *n* db számot. A programtól illetve az input adatok feldolgozásától függően ezt egyszerre, vagy külön-külön koordinátánként kell megadni. Mindkét esetben a tömb feltöltése koordinátánként történik, azaz lépésről-lépésre ugyan azt a műveletet kell ismételni, csak a rögzítendő elem sorszáma változik.




1. lépés: első elem rögzítése
2. lépés: második elem rögzítése
3. lépés: harmadik elem rögzítése
⋮
*n*. lépés: *n*.(=7) elem rögzítése

Az elemzésből is jól látszik, hogy ugyanazt a műveletet kell végrehajtani, csak a rögzítendő elem indexe változik, azaz bármely ismétléses vezérlés használható a feltöltésre, az elemek indexe kapcsolatban kell hogy legyen a ciklusváltozóval (általában feltöltésnél for ciklust alkalmazunk és az i. elemmel foglalkozunk). Fontos megemlíteni, hogy a különböző programozási nyelvek különböző a tömbök elemeinek indexelése. A Flowgorithm rendszerben, mivel több programozási nyelvre képes átkonvertálni a folyamatábrával megadott algoritmust, ezért az egydimenziós tömb első elemének indexe 0, amire a kiíratásnál figyelnünk kell.

A tömb feltöltésének és kiíratásának algoritmusa:




```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
      private static Scanner input = new Scanner(System.in);

      public static void main(String[] args) {
            int i, n;
            int[] v = new int[10];

            System.out.println("Adja meg a V vektor koordinátáinak számát! n=");
            n = input.nextInt();
            for (i = 0; i <= n - 1; i++) {
                  System.out.print(i + 1);
                  System.out.println(". elem:");
                  v[i] = input.nextInt();
            }
            System.out.print("A megadott vektor: V=(");
            for (i = 0; i <= n - 2; i++) {
                  System.out.print(v[i]);
                  System.out.print("; ");
            }
            System.out.print(v[n - 1]);
            System.out.println(")");
      }
}
```

Az első for ciklus felel a tömb elemeinek feltöltéséért, a második for ciklus a kiíratásért. A második for esetében a vektorszerű kiíratás miatt kellett az utolsó előtti elemig futtatni a ciklusváltozót, míg az utolsó elemet a cikluson kívül írattuk ki.

```
Adja meg a V vektor koordinátáinak számát! n=
3
1. elem:
1
2. elem:
5
3. elem:
4
A megadott vektor: V=(1; 5; 4)
```

Az egydimenziós tömbök kezelésének gyakorlására rengeteg egyszerű feladatot lehet megadni:

- Adjuk meg egy tömb legnagyobb elemét!
- Adjuk meg egy tömb legkisebb elemét!
- Adjuk meg egy tömb legnagyobb elemének a helyét!
- Adjuk meg egy tömb legkisebb elemének a helyét!
- Számítsuk ki egy tömb elemeinek összegét!
- Számítsuk ki egy tömb elemeinek átlagát!
- Adjuk meg a páros indexű elemek összegét!
- Adjuk meg a páratlan indexű elemek összegét!
- ⋮

Az előző feladat és az említett feladatokat ötvözve megadunk egy algoritmust, mely alapján a többi egyszerű feladatra is könnyedén adható algoritmus.

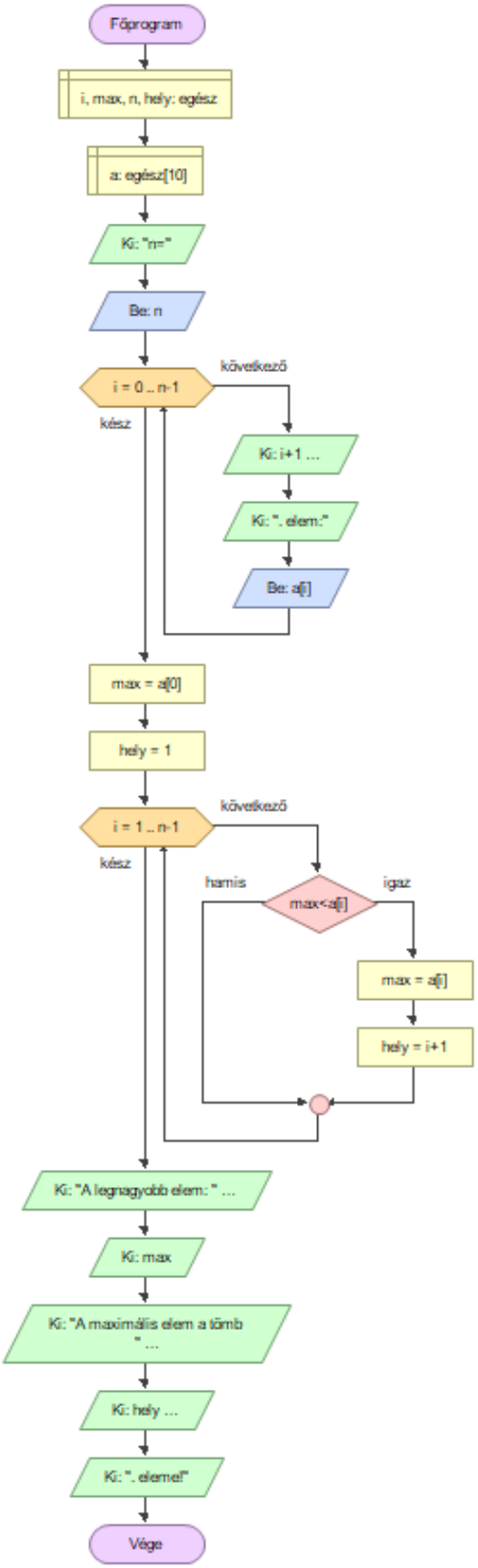
**8. Feladat. Kérjük be n egész számot, melyeket egy tömbben tárolunk. Határozzuk meg a megadott számok maximumát, illetve a maximális elem indexét a tömbben!**

Vizsgáljuk a feladatot!

Első esetben a maximális elem meghatározására koncentráljunk. Több lehetséges megoldás közül egy segédváltozót alkalmazó algoritmus megadása mellett döntöttünk. Deklaráljunk egy *max* nevű változót, melyben a maximális elemet szeretnénk eltárolni. A változó használatához több alapötlet is használható:

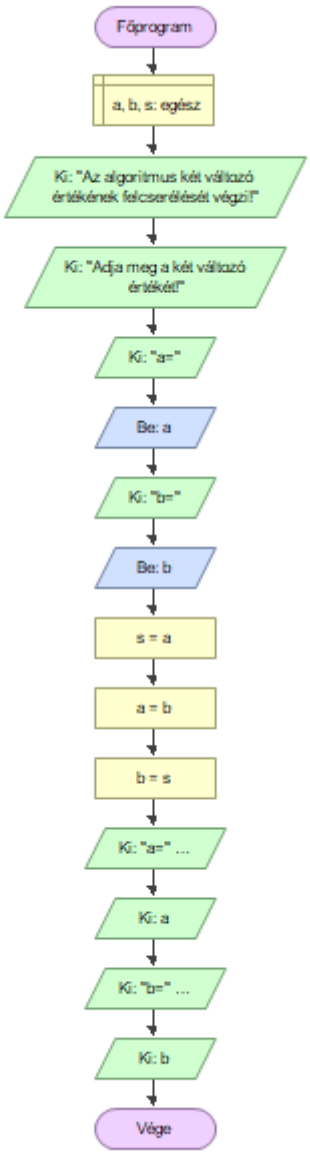
- a deklarált változóba tároljuk el a változó típusának megfelelő legkisebb értéket, majd ez után a tömb elemein végighaladva hasonlítsuk össze a változó értékét a tömb aktuális elemével.
- a deklarált változóba tároljuk el a tömb első indexű elemének értékét, majd ez után a tömb első elemét kivéve haladjunk végig a tömb elemein, és hasonlítsuk össze a változó értékét a tömb aktuális elemével.

Vizsgáljuk a maximális elem helyének meghatározását. Több lehetőség közül itt is az egyik legegyszerűbb megoldást választottuk, ami szintén egy segédváltozóval könnyedén megoldható. Felveszünk egy *hely* egész típusú változót, mely értékét akkor változtatjuk, ha a *max* változó értékét is változtatjuk. Gyakorlatilag ez azt jelenti, hogy amikor inicializáljuk a *max* változót a tömb első elemének értékével, akkor a hely változót is inicializáljuk, értéke 1 lesz. A cikluson belül, amikor a *max* értékét változtatjuk, akkor a hely változónak az aktuális ciklusváltozótól eggyel nagyobb értéket adjuk (az indexelés 0-tól indul, így az első elem a tömb 0. eleme).




```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
     private static Scanner input = new Scanner(System.in);

     public static void main(String[] args) {
          int i, max, n, hely;
          int[] a = new int[10];

          System.out.println("n=");
          n = input.nextInt();
          for (i = 0; i <= n - 1; i++) {
               System.out.print(i + 1);
               System.out.println(". elem:");
               a[i] = input.nextInt();
          }
          max = a[0];
          hely = 1;
          for (i = 1; i <= n - 1; i++) {
               if (max < a[i]) {
                    max = a[i];
                    hely = i + 1;
               }
          }
          System.out.print("A legnagyobb elem: ");
          System.out.println(max);
          System.out.print("A maximális elem a tömb ");
          System.out.print(hely);
          System.out.println(". eleme!");
     }
}
```

A maximális elem meghatározásához egy másik algoritmust is megadunk, mely a buborékrendezés alapja. Az algoritmus alapötlete, hogy az egymás melletti elemeket összehasonlítja, és ha a sorrend jó, azaz két elem közül a nagyobb indexű értéke is nagyobb vagy egyenlő ($a[i] \leq a[i+1]$), akkor minden rendben, ellenkező esetben a két elemet felcseréljük. Elemek felcseréléséről eddig még nem esett szó, holott ez is alap algoritmus. A csere megoldásához is segédváltozót használunk, melyben a két felcserélendő változó közül az egyik értékét tároljuk.

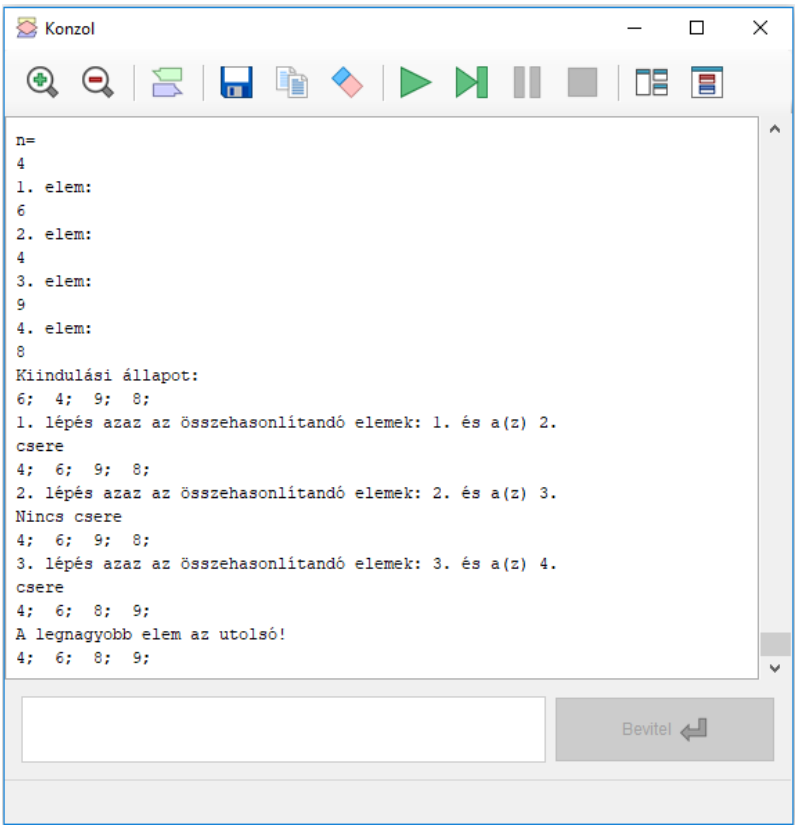



```
Függvény Főprogram
    Deklarálás: a, b, s: egész

    Ki: "Az algoritmus két változó értékének felcserélését végzi!"
    Ki: "Adja meg a két változó értékét!"
    Ki: "a="
    Be: a
    Ki: "b="
    Be: b
    Értékadás: s = a
    Értékadás: a = b
    Értékadás: b = s
    Ki: "a="
    Ki: a
    Ki: "b="
    Ki: b
Vége
```

```
Az algoritmus két változó értékének felcserélését végzi!
Adja meg a két változó értékét!
a=
6
b=
2
a=2
b=6
```

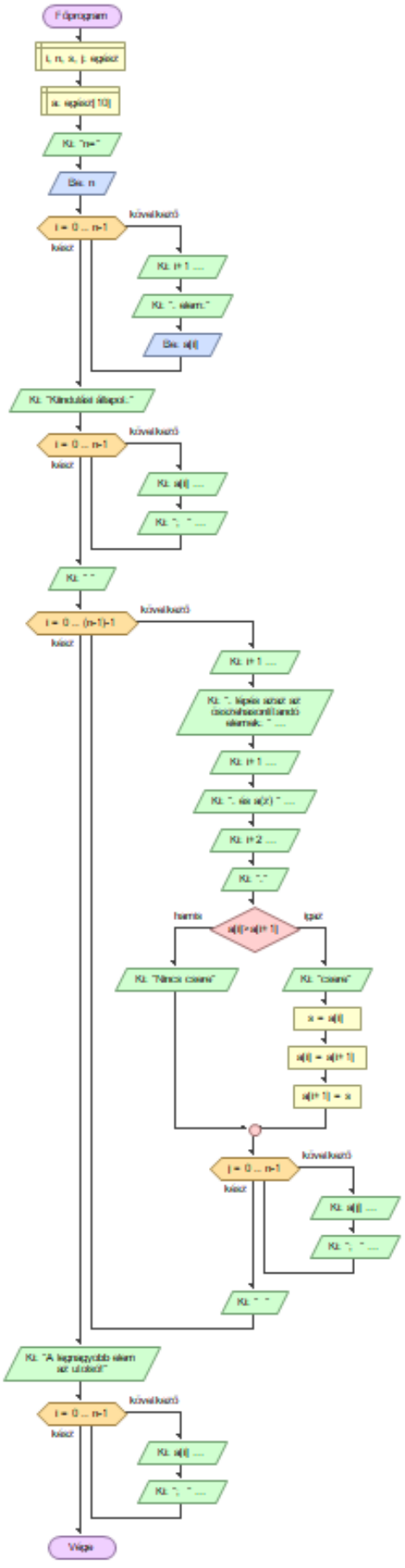
A fenti algoritmusnál nem JAVA kódot, hanem magyar pszeudokódot adtunk meg, melyet szintén a Flowgorithm program segítségével generáltunk. Visszatérve a maximális elem meghatározására, a csere algoritmus segítségével könnyedén elérhetjük, hogy a legnagyobb elem a tömb legnagyobb indexű eleme legyen (algoritmus a következő oldalon).

A program tesztelése a konzol segítségével:




Az algoritmus közvetlenül nem adja vissza a maximális elemet, de jól látható, hogy az algoritmus végén a tömb utolsó elemét kell csak visszaadni, hisz a legnagyobb elem, mindig az utolsó pozícióban található elem lesz. A legnagyobb elem helyének meghatározás a kezdeti tömbben egy kicsit elgondolkodtató, de megoldható, mely algoritmusát az olvasóra bízzuk.

A tömbök feltöltésére több lehetőség van, nem kell mindig a felhasználótól bekérni (a gyakorlati problémáknál a felhasználó adja meg), lehet például véletlen szám generátort használni. Többször előfordul, hogy a tárolt értékek és a tömb indexei között valamiféle összefüggés van (páratlan indexű helyen az érték nulla, páros indexű helyen az érték 1).

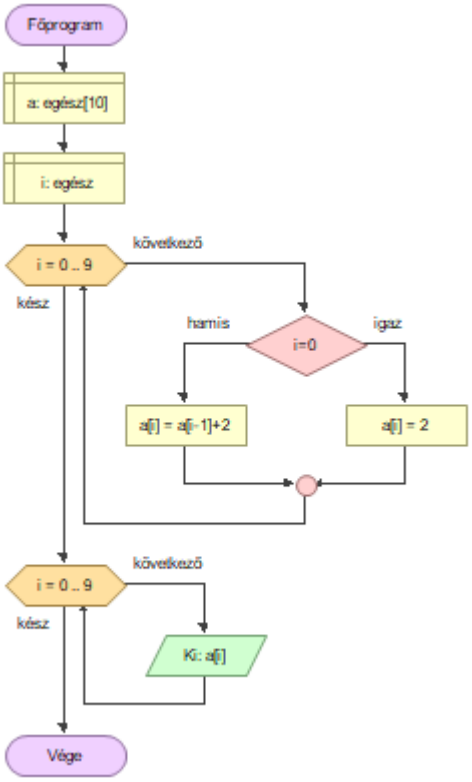



```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
    private static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int i, n, s, j;
        int[] a = new int[10];

        System.out.println("n=");
        n = input.nextInt();
        for (i = 0; i <= n - 1; i++) {
            System.out.print(i + 1);
            System.out.println(". elem:");
            a[i] = input.nextInt();
        }
        System.out.println("Kiindulási állapot:");
        for (i = 0; i <= n - 1; i++) {
            System.out.print(a[i]);
            System.out.print(";  ");
        }
        System.out.println(" ");
        for (i = 0; i <= n - 1 - 1; i++) {
            System.out.print(i + 1);
            System.out.print(". lépés azaz az összehasonlítandó elemek: ");
            System.out.print(i + 1);
            System.out.print(". és a(z) ");
            System.out.print(i + 2);
            System.out.println(".");
            if (a[i] > a[i + 1]) {
                System.out.println("csere");
                s = a[i];
                a[i] = a[i + 1];
                a[i + 1] = s;
            } else {
                System.out.println("Nincs csere");
            }
            for (j = 0; j <= n - 1; j++) {
                System.out.print(a[j]);
                System.out.print(";  ");
            }
            System.out.println("  ");
        }
        System.out.println("A legnagyobb elem az utolsó!");
        for (i = 0; i <= n - 1; i++) {
            System.out.print(a[i]);
            System.out.print(";  ");
        }
    }
}
```

**9. Feladat. Tároljuk el egydimenziós tömbben az első tíz pozitív páros számot!**

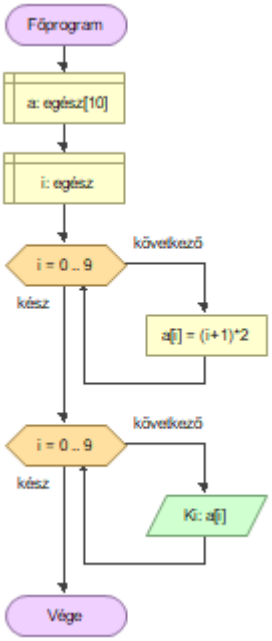
1. megoldás:

A páros számok számtani sorozatot alkotnak, melynek a differenciája kettő. Jelen esetben a sorozat első eleme is kettő. Ezt a tulajdonságot kihasználva egyszerűen az első elem megadásával, majd az $a[i] = a[i-1] + 2$ összefüggést felhasználva az algoritmus a következő:



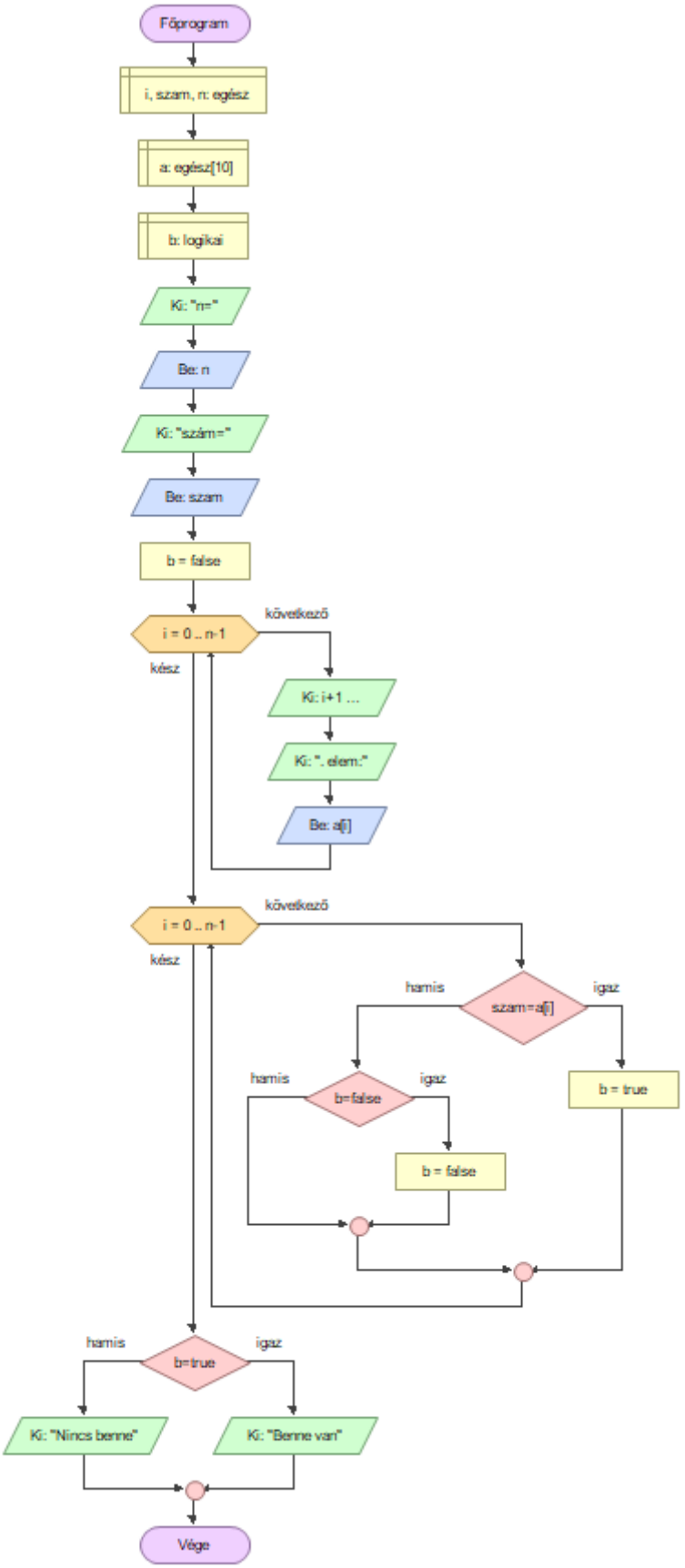
2. megoldás:

A páros számokat a matematikában általában $2n$ alakban adják meg, a keresett páros számok meghatározhatók az indexek függvényében, mégpedig a $a[i] = (i+1) \cdot 2$ hozzárendelési szabállyal.




A gyakorlati életben sokszor szembesülünk azzal a problémával, hogy azonos típusú adatok halmazában keresünk egy megadott értéket. Az azonos típusú adatokat tömbökben tároljuk, így a feladat tömbökben egy megadott érték keresése. Elemezzük egy kicsit, hogyan lehet meghatározni, hogy a keresett elem megtalálható-e a tömbben vagy sem.

Először próbáljunk megoldást adni a következő problémára: Adott 1000 darab természetes szám és egy keresendő érték, ami szintén természetes szám. Állapítsuk meg, hogy a keresett szám megtalálható-e az adott 1000 szám között. Rövid gondolkodás után arra a megállapításra jutunk, hogy a keresett értéket az összes, azaz mind az ezer számmal össze kell hasonlítani (a legrosszabb esetben, ha a keresett szám az ezredik szám, vagy a keresett szám nincs a halmazban). A megoldáshoz tartozó algoritmus:




```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
    private static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int i, szam, n;
        int[] a = new int[10];
        boolean b;

        System.out.println("n=");
        n = input.nextInt();
        System.out.println("szám=");
        szam = input.nextInt();
        b = false;
        for (i = 0; i <= n - 1; i++) {
            System.out.print(i + 1);
            System.out.println(". elem:");
            a[i] = input.nextInt();
        }
        for (i = 0; i <= n - 1; i++) {
            if (szam == a[i]) {
                b = true;
            } else {
                if (b == false) {
                    b = false;
                }
            }
        }
        if (b == true) {
            System.out.println("Benne van");
        } else {
            System.out.println("Nincs benne");
        }
    }
}
```

```
n=
3
szám=
4
1. elem:
5
2. elem:
6
3. elem:
4
Benne van
```

```
n=
3
szám=
4
1. elem:
1
2. elem:
2
3. elem:
3
Nincs benne
```

Másodszor tekintsük a következő problémát: Gondoltam egy maximum három jegyű természetes számra. Melyik ez a szám? Ebben az esetben is rövid gondolkodás után láthatjuk, hogy az előző módszer itt is alkalmazható, azaz maximum 1000 kérdésből kitaláljuk melyik számra gondoltunk, de kihasználhatjuk, hogy a 0-tól 999-ig a számok növekvő sorba vannak rendezve, azaz ezt kihasználva is kérdezhetünk. Felezzük meg a halmazt és válasszuk azt a részhalmazt, melyben a keresett elem található, majd felezzük meg ezt a halmazt is és válasszuk azt a halmazt melyben a keresett elem található, és így tovább. Bizonyítható, hogy 1000 szám esetében 10 összehasonlítás segítségével megadható a keresett elem.

Természetesen az ember ösztönösen érzi, hogy ha nem össze-vissza megadott értékek között keresgél, hanem egy rendezett halmazban, akkor sokkal könnyebben megtalálja a keresett értéket, elemet. A fentiek alapján feladatunk lehet egy tömb elemeinek valamilyen szempont szerinti rendezése. A 7. feladatban megadott algoritmus alapját képezi az úgynevezett buborék rendezésnek, mely alapötlete a következő:

Rendezzük a következő tömböt:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 2 | 9 | 6 | 12 | 4 | 3 | 2 | 1 | 10 |

1. lépés: hajtsuk végre a 7. feladatban megadott algoritmust a tejes tömbre, ekkor a legnagyobb elem kerül az utolsó helyre.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 6 | 9 | 4 | 3 | 2 | 1 | 10 | 12 |

2. lépés: hajtsuk végre a 7. feladatban megadott algoritmust az előző lépésben kapott tömb rendezetlen részére (ebben az esetben csak az utolsó elemet hagyjuk el, így egy 9 elemű tömbre alkalmazzuk az algoritmust), ekkor a vizsgált tömb legnagyobb eleme kerül az utolsó előtti helyre a teljes tömbben.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 6 | 8 | 4 | 3 | 2 | 1 | 9 | 10 | 12 |

3. lépés: hajtsuk végre a 7. feladatban megadott algoritmust az előző lépésben kapott tömb rendezetlen részére (ebben az esetben csak az utolsó két elemet hagyjuk el, így egy 8 elemű tömbre alkalmazzuk az algoritmust), ekkor a vizsgált tömb legnagyobb eleme kerül a megfelelő helyre a teljes tömbben.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 6 | 4 | 3 | 2 | 1 | 8 | 9 | 10 | 12 |

⋮

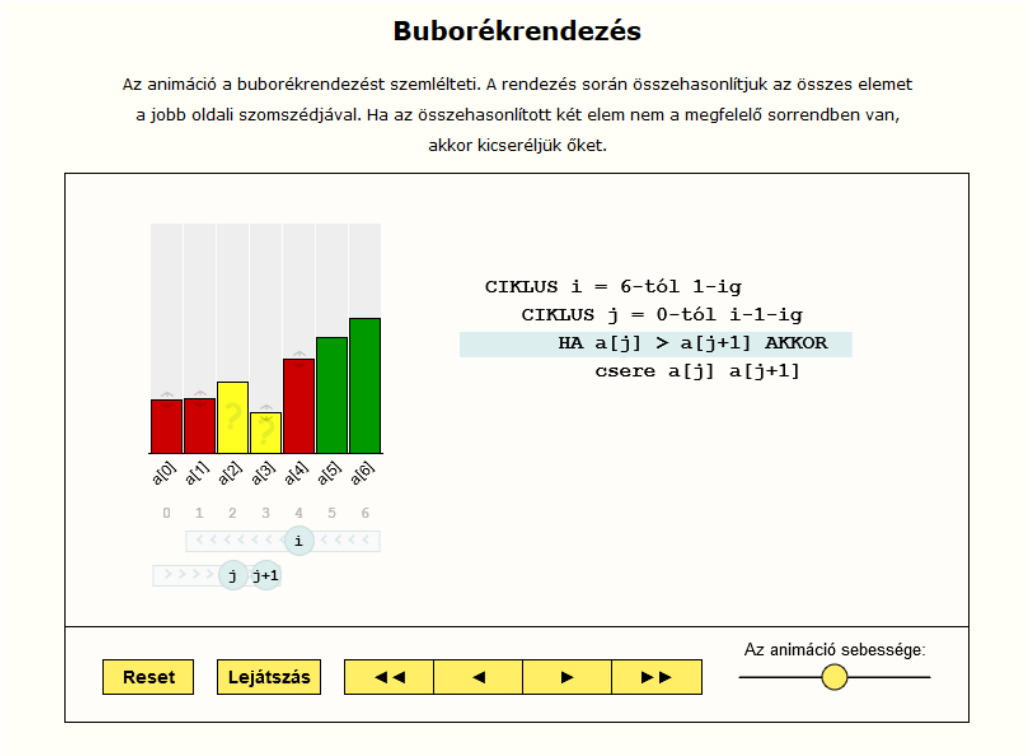
és így tovább mindaddig, míg a végén már csak két elemet kell összehasonlítani

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 2 | 3 | 4 | 6 | 8 | 9 | 10 | 12 |

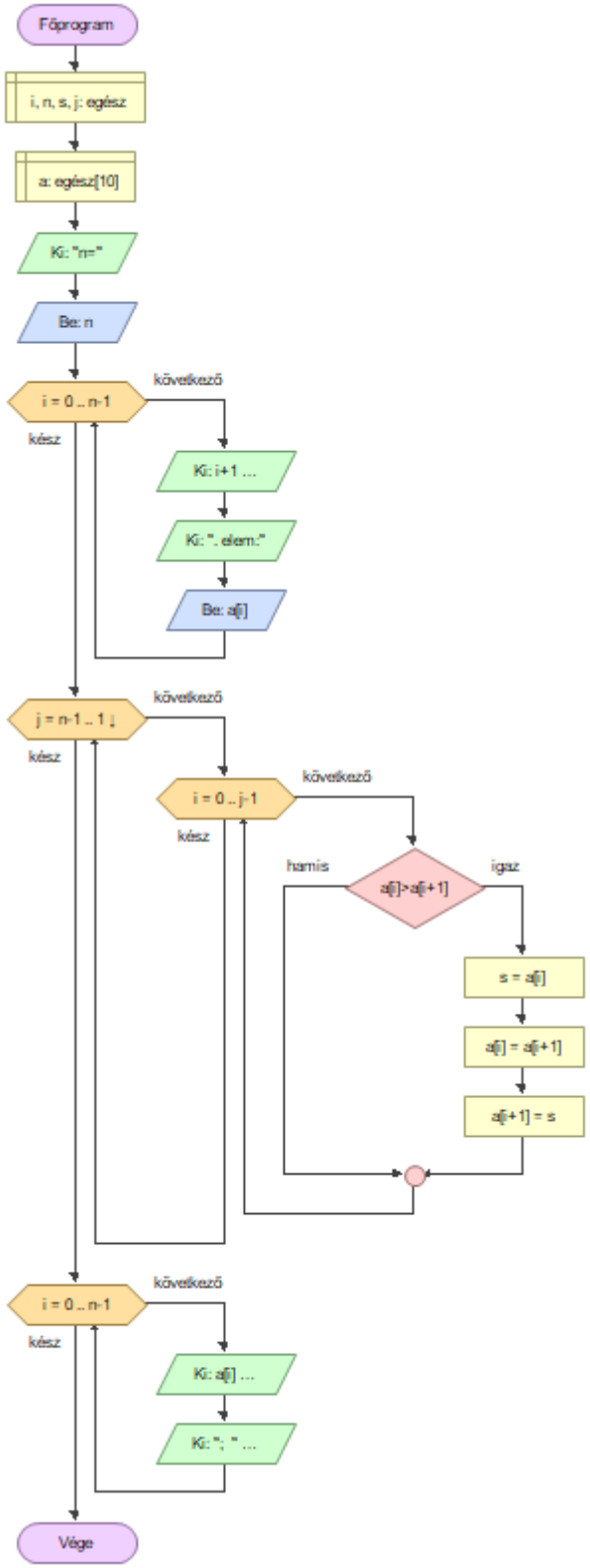
9. lépés (*n*-1. lépés, ahol *n* a tömb mérete): hajtsuk végre a 7. feladatban megadott algoritmust a tömb utolsó két rendezetlen elemére (ebben az esetben csak az utolsó két elemet hagyjuk el, így egy 8 elemű tömbre alkalmazzuk az algoritmust), ekkor a vizsgált tömb legnagyobb eleme kerül a megfelelő helyre a teljes tömbben.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 3 | 4 | 6 | 8 | 9 | 10 | 12 |

Jól látható, hogy a buborékrendezés hatására a tömb elemei növekvő (vagy csökkenő) sorba rendezhetők. Rengeteg segítség található az Interneten programozás tárgykörében, sőt egyre több algoritmikus gondolkodást fejlesztő oldal is elérhető. Az egyik algoritmusok vizualizálásával foglalkozó oldal az interaktív animációkat tartalmazó http://anim.ide.sk/bevezeto.php. Itt is megtalálható a buborékrendezés interaktív animációval valamint pszeudokóddal segítve a megértését az algoritmusnak.




## 10. Feladat. A buborékrendezés megvalósítása Flowgorithm segítségével:




```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
    private static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int i, n, s, j;
        int[] a = new int[10];

        System.out.println("n=");
        n = input.nextInt();
        for (i = 0; i <= n - 1; i++) {
            System.out.print(i + 1);
            System.out.println(". elem:");
            a[i] = input.nextInt();
        }
        for (j = n - 1; j >= 1; j--) {
            for (i = 0; i <= j - 1; i++) {
                if (a[i] > a[i + 1]) {
                    s = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = s;
                }
            }
        }
        for (i = 0; i <= n - 1; i++) {
            System.out.print(a[i]);
            System.out.print("; ");
        }
    }
}
```

```
n=
6
1. elem:
5
2. elem:
8
3. elem:
6
4. elem:
9
5. elem:
3
6. elem:
2
2;  3;  5;  6;  8;  9;
```

A fent leírtak alapján látható, hogy számos feladat adható, melyek megoldására egydimenziós tömbök használatával könnyedén adható jó algoritmus. A gyakorlati életben a problémák megoldásához sok esetben mátrixok feldolgozásán keresztül jutunk, így fontos a kétdimenziós mátrixok kezelésének programozás-technikai ismerete. Vizsgáljuk meg a kétdimenziós mátrixokat:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{m \times n}$$

Jól látható, hogy a kétdimenziós mátrixok elemzésénél két megközelítést használhatunk:

1. Az $A$ mátrix gyakorlatilag $m$ db sorvektorból épül fel. A feldolgozás során először meg kell mondani, hogy melyik vektorról van szó, azaz meg kell határozni, hogy a mátrix hányadik sorában lévő adatokkal szeretnénk dolgozni. Az elemek indexeléséhez két számot használunk, melyben az első szám határozza meg, hogy melyik sorban található az adott elem. Amennyiben soronként szeretnénk feldolgozni az adatokat, először a sor indexét kell meghatározni, majd a fent leírtak alapján bejárni a sorindex által meghatározott vektort.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{m \times n}$$

2. Egy másik lehetőség, ha nem a sorokat, hanem az oszlopokat vizsgáljuk. Az $A$ mátrix $n$ db oszlopvektorból épül fel. Az indexelésnél használt rendezett számpárból a második szám határozza meg, hogy melyik oszlopban található az adott elem. Az előző esethez hasonlóan itt is meg kell határozni, hogy melyik vektorral szeretnénk dolgozni, amely az oszlopindex megadásával történik.

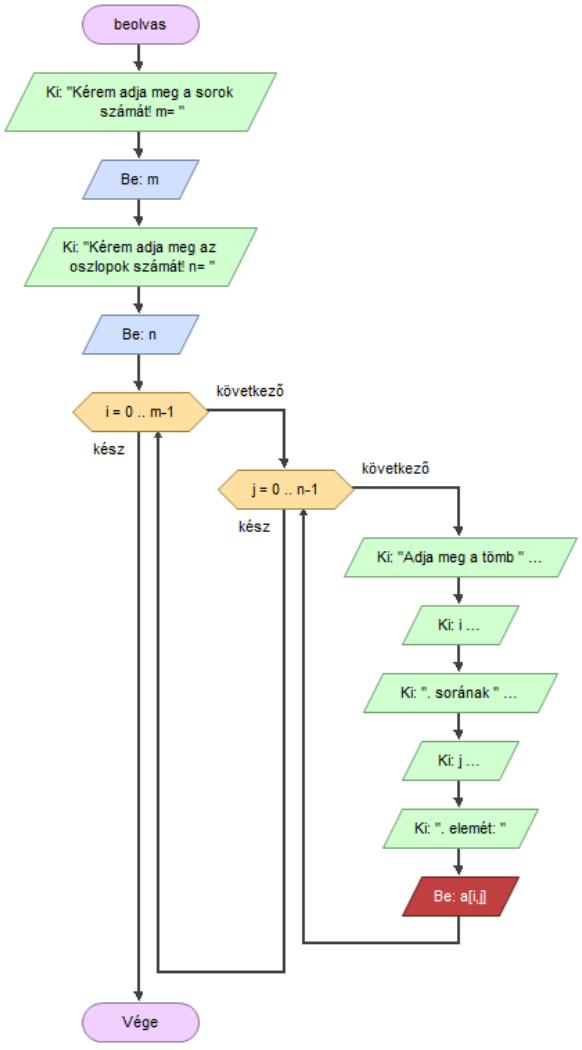
$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{m \times n}$$

Az előzőekben leírt feldolgozási módszerek közül azt választjuk, amely segítségével a feladat megoldása egyszerűbb (előfordulhat, hogy mindkét módszert használnunk kell, például mátrixok szorzása esetén, amely sor-oszlop kombinációval történik.).

A Flowgorithm szoftver sajnos nincs felkészítve többdimenziós tömbök kezelésére, így a deklarálásuk sem lehetséges, aminek a következménye, hogy a megadott folyamatábrák ugyan a kívánt algoritmust írják le, de sajnos futtatásukra nincs lehetőség. Figyelembe véve, hogy a hallgatók a kapcsolódó „Programozás alapjai” nevű kurzus keretében már elsajátították a JAVA nyelv alapjait, így a jegyzet második részében megadott futtatható forráskódok segítségével tudják az algoritmusokat futtatni, tesztelni. Minden feladat alapja a tömbök feltöltése, így első lépésként tekintsük ezt a feladatot, melyre a későbbiekben csak hivatkozni fogunk.

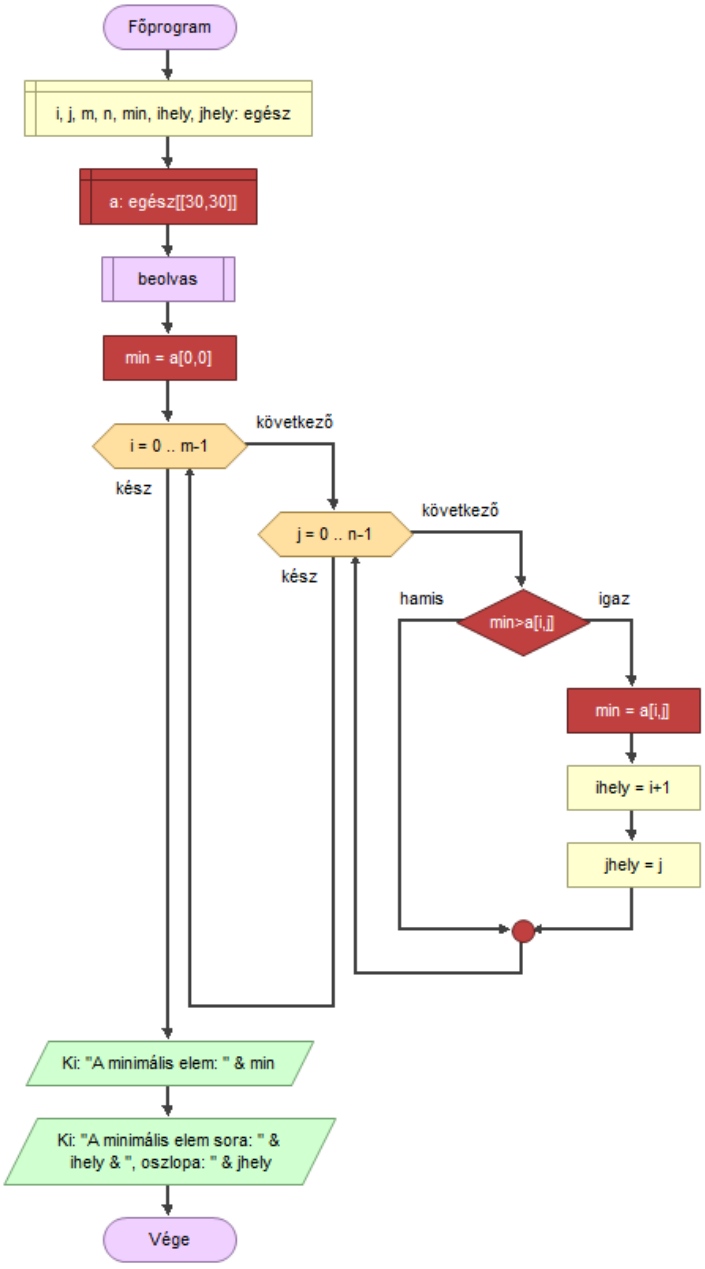
## 11. Feladat. Kétdimenziós tömb feltöltése

Vizsgáljuk a feladatot a feldolgozási esetek ismeretében. Mivel először vagy a sorok vagy az oszlopok indexét határozzuk meg, és utána dolgozzuk fel a hozzá tartozó vektort, ezért két egymásba ágyazott ciklusra van szükségünk. Az első módszert alkalmazva a külső ciklus meghatározza a sorindexet, a belső a sorban lévő elem helyét, azaz az oszlopindexet.

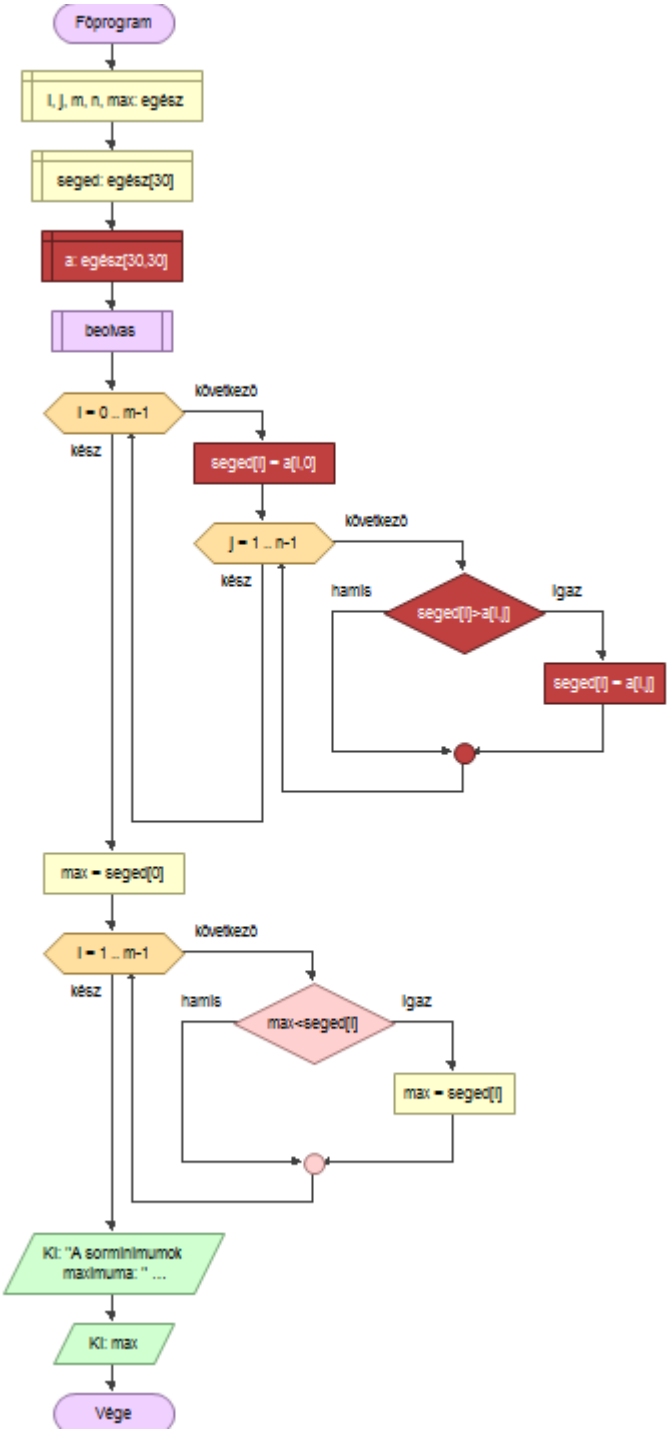


Kétdimenziós tömbökre is számos alapfeladat adható, ezek közül tekintsünk néhányat. Mivel az előzőekben már részleteztük a felhasználói felületet, és részeit, ezért a későbbiekben csak folyamatábrával írjuk le az algoritmusokat.

**12. Feladat. Határozza meg kétdimenziós tömb minimális elemét, valamint az elem helyét!**

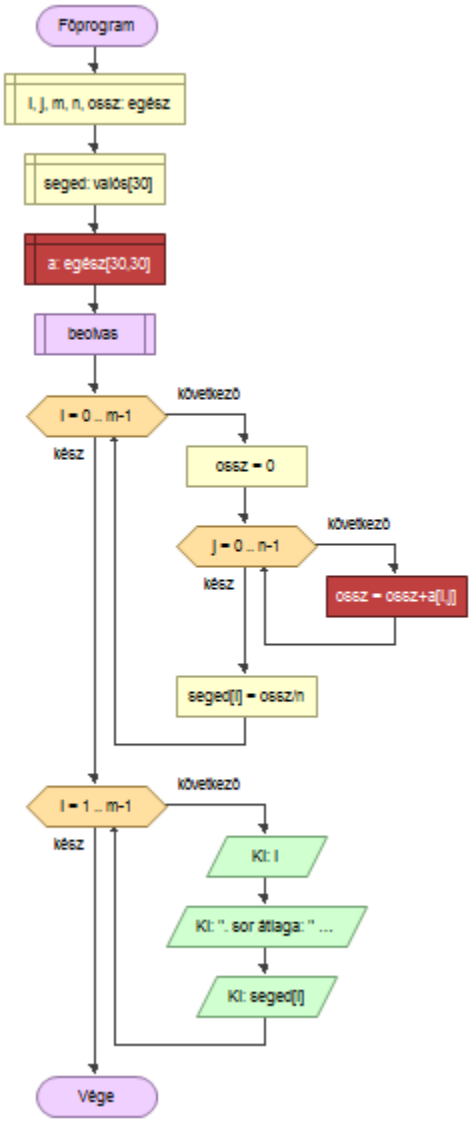


**13. Feladat. Határozza meg kétdimenziós tömb sorminimumainak maximumát!**




Az algoritmus alapötlete, hogy a sorok minimumait eltároljuk egy segéd tömbben, melynek a mérete megegyezik a sorok számával. Minden egyes sorra alkalmazzuk a korábban megadott maximum keresési algoritmust, majd a minimum értékét eltároljuk a segéd tömbben, melyre alkalmazzuk a maximum kereső algoritmust.

**14. Feladat. Határozza meg kétdimenziós tömb sorainak az átlagát!**

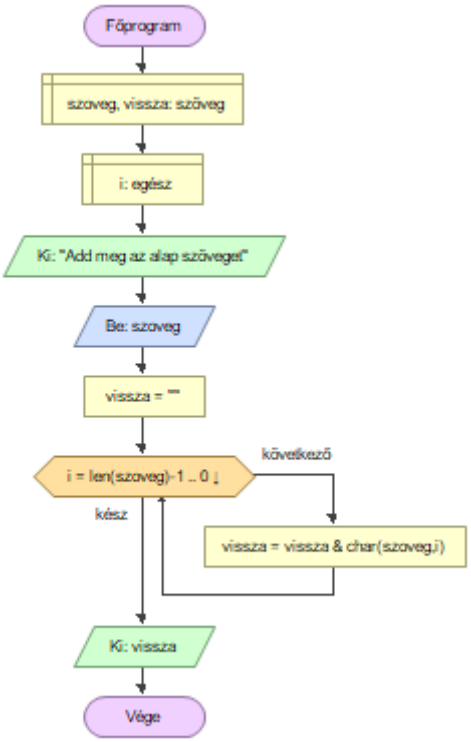



Az algoritmus megvalósításához ismernünk kell az átlagszámítás alapjait (mérésben szereplő értékek összegét elosztjuk a mérésben szereplő elemek számával). A megvalósítás során a segéd tömb *i.* elemébe eltároljuk az eredeti sor *i.* sorának átlagát.

Az összetett adatstruktúrákat elemezése közben felmerülhet az igény a sztringek vizsgálatára is. Ahogyan azt már az algoritmusok alapjairól szóló fejezetben leírtuk a stringek karakterekből álló láncok, azaz karakterláncok. Amennyiben ezt a megközelítést választjuk egy szöveg, azaz sztring tekinthető olyan egydimenziós tömbnek, melynek az elemei karakterek. Hasonlóan az egy dimenziós tömbökhöz, szöveg esetében is beszélhetünk a karekterek (elemek) indexéről, azaz megadhatjuk, hogy hányadik karakterrel akarunk dolgozni, illetve index alapján karaktereket tudunk kivágni sztringekből. A különböző programozási nyelvekben több

beépített stringkezelő függvény található, melyek közül a JAVA-ban használt függvényekről részletes leírást találunk a jegyzet második felében. Jelen fejezetben a szöveg hosszát, azaz a szöveget alkotó karakterek számát, valamint a szövegben előforduló karakterek közül adott indexű karaktert visszaadó függvényt használjuk. Meg kell említenünk, hogy vannak alapvető szövegkezelési műveletek, mint például az összefűzés, amit szintén használni fogunk. Első feladatunk egy egyszerű „substring" visszadása lesz, melyhez az említett műveleteket használjuk.

**15. Feladat. Írjon olyan algoritmust, mely egy megadott szöveg karaktereit fordított sorrendben adja vissza!**




```
Add meg az alap szöveget
indul a görög aludni
indula görög a ludni
```

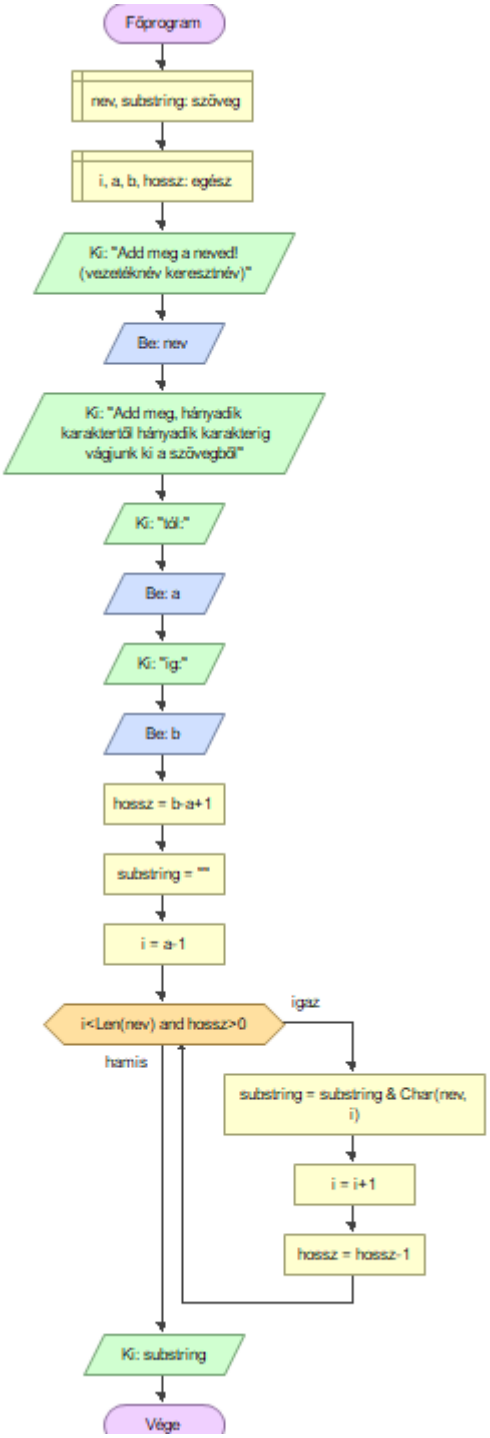
**16. Feladat. Írjon olyan algoritmust, mely egy megadott szöveg tetszőleges karakterindexétől tetszőleges karakterindexéig visszaadja a szöveg részletét!**

Vizsgáljuk a feladatot!

A felhasználótól három adatot kérünk:

- szöveg
- kezdőindex
- végindex

Jól látható, hogy, ha a végindex kisebb vagy egyenlő, mint a kezdőinex, akkor nincs értelmezhető intervallum, melyhez tartozó szövegrészletet meg lehetne adni. Továbbá azt is vizsgálnunk kell, hogy mi van abban az esetben, ha valamelyik index nagyobb, mint a szöveg hossza. Az imént említett feltételeket vizsgálva a következő algoritmus adható meg:

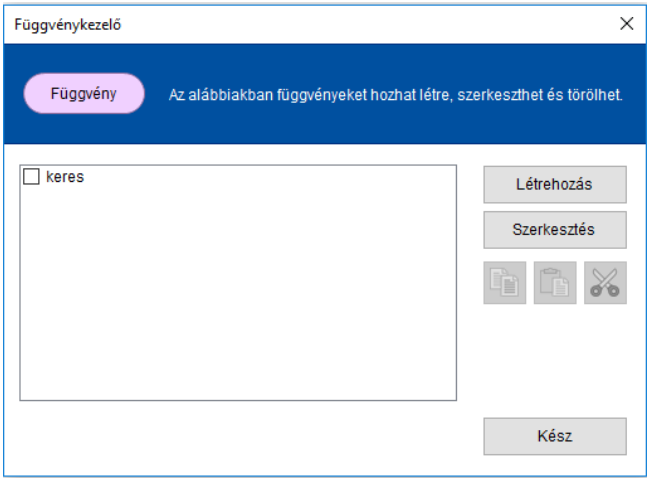


Előfordulhat, hogy ugyanazt a feladatot más adatokon többször végre kell hajtani egy algoritmusban, ilyenkor úgynevezett függvényeket (különálló programrészleteket) alkalmazunk, melyeket csak meghívunk a kívánt adatokkal, és azonnal a visszatérési értékét használhatjuk. Az adott algoritmust minimális módon átalakítva, készíthetünk függvényt is, melyeket eddig még nem használtunk. A függvények jellemzője, hogy vannak meghívási adataik, és visszatérési értékük. Természetesen vannak olyan függvények is, melyek egyik említett értékkel sem rendelkeznek.

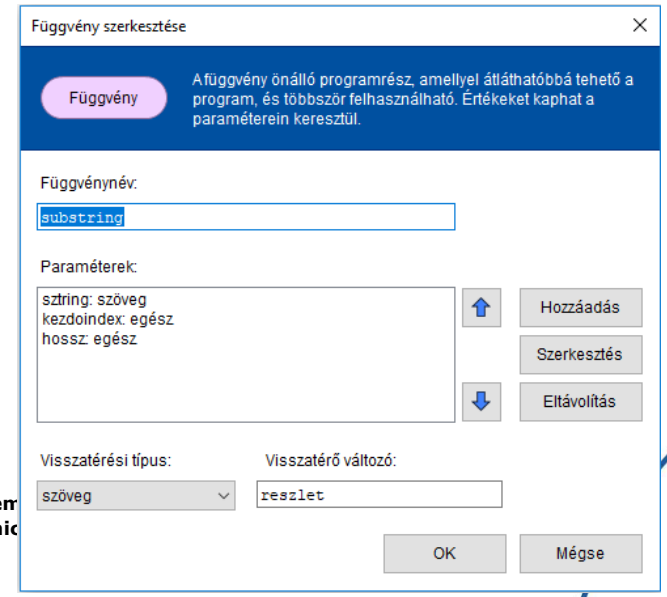
**17. Feladat. Készítsünk függvényt, mely egy adott szövegből megaadott karakterpozíciótól megadott számú karaktert metsz ki!**

Első lépésként nézzük meg, hogyan lehet függvényeket létrehozni a Flowgorithm segítségével:

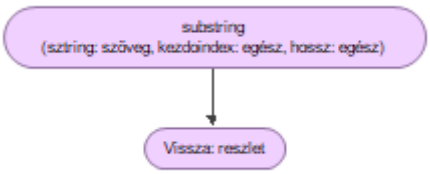
Nyissuk meg a függvénykezelőt a  gombra kattintva:



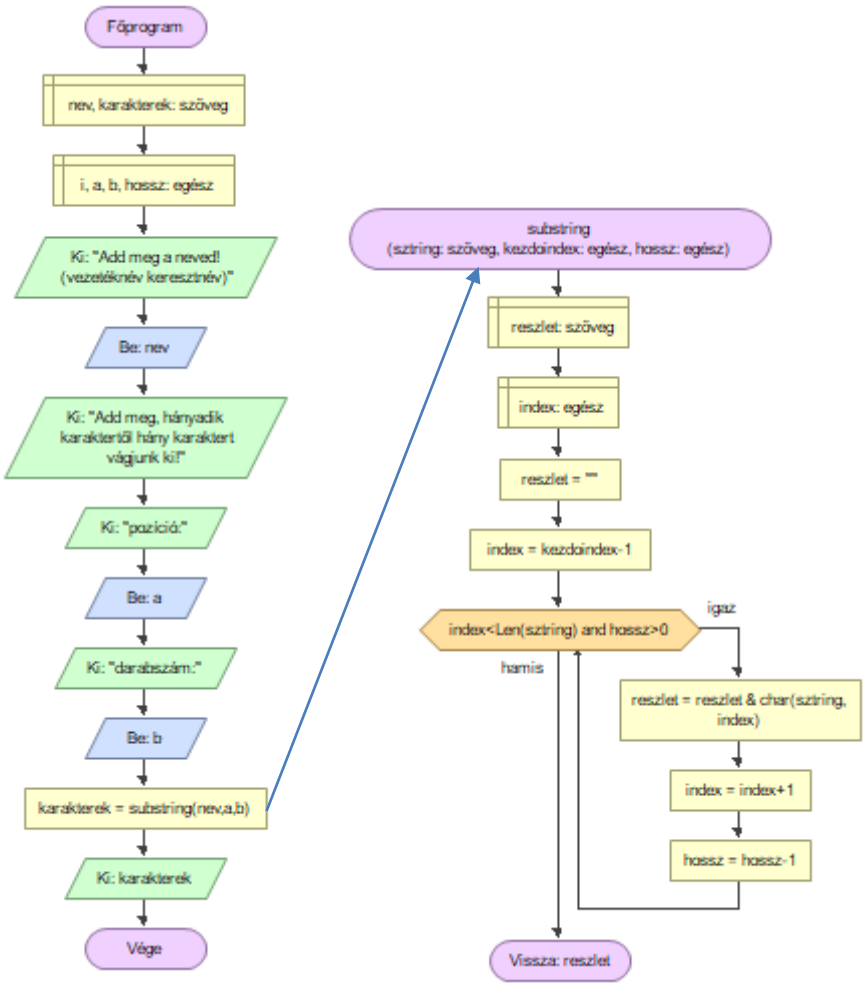

Majd válasszuk a létrehozás gombot, és adjuk meg a függvény nevét, valamint a meghívásánál használt adatokhoz tartozó változókat adattípusukkal együtt!



Miután minden adatot megadtunk, elkezdhetjük a függvény szerkesztését. A folyamatábra minimálisan megváltozik, a kezdő síkidomban a függvény neve valamint a meghívásához szükséges paraméterek találhatók.



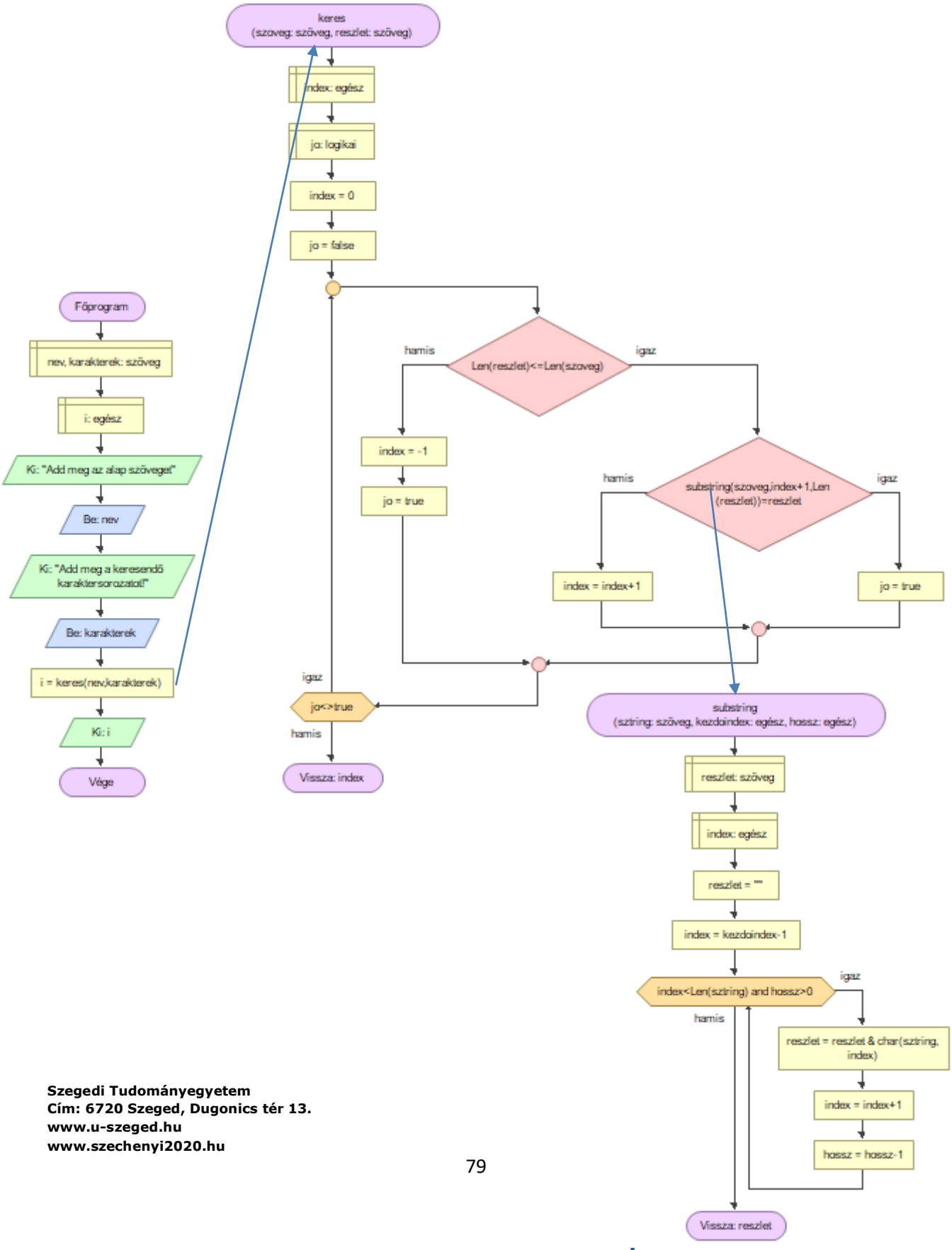

A függvény és az őt meghívó főprogram folyamatábrája Flowgorithm-ben:




```
Add meg a neved! (vezetéknév keresztnév)
Árgilán Viktor
Add meg, hányadik karaktertől hány karaktert vágjunk ki!
pozíció:
2
darabszám:
4
rgil
```

**18. Feladat. Készítsünk függvényt, mely egy karaktersorozat adott szövegbeli első előfordulásának kezdőindexét adja vissza!**

Az előzőekben elkészített függvényt felhasználva a folyamatábra:



# Algoritmusok megvalósítása JAVA nyelvben

A programozás 0. lépcsője a programozási nyelv és környezet kiválasztása és telepítése. A jegyzetben a tantervi előírásoknak megfelelően a Java programozási nyelve építünk. Ennek megfelelően a Java fejlesztői környezet két részből áll: a Java Development Kit-ből, és a kényelmes fejlesztést lehetővé tevő Eclipse integrált fejlesztői környezetből

## A JDK telepítése

A JDK az alábbi oldalról tölthető le:

https://www.oracle.com/technetwork/java/javase/overview/index.html

A letöltés és használat regisztrációhoz kötött, de nem kereskedelmi célokra ingyenes. Teljesen szabadon használható az OpenJDK, ,mely az alábbi oldalról tölthető le:

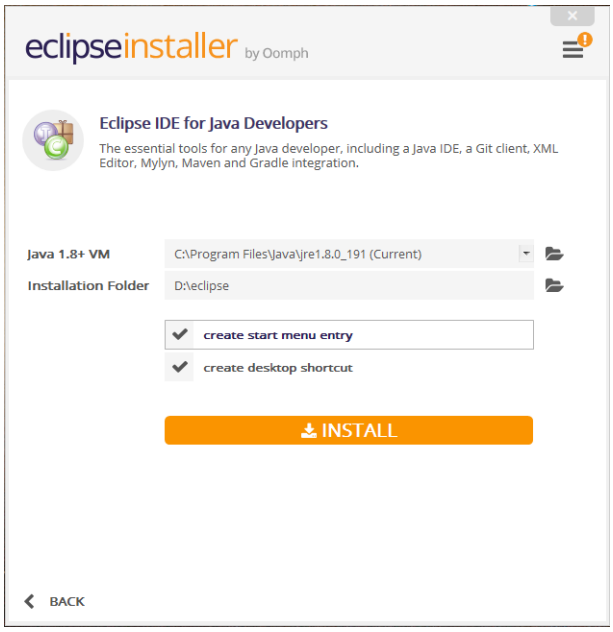
https://openjdk.java.net/

Bármelyik JDK-t választhatjuk. Az alapbeállításokat elfogadva, mind a két JDK automatikusan települ.

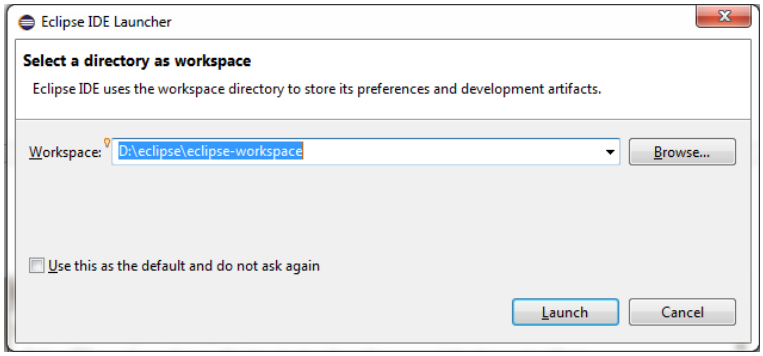
## Az Eclipse telepítése

Amennyiben az Eclipse-t szeretnénk használni, akkor a http://www.eclipse.org/downloads/ oldalról töltsük le az operációs rendszerünknek megfelelő legfrissebb stabil verzióját. Windows operációs rendszer esetén ez egy keret program, melyben kiválaszthatjuk, hogy milyen fejlesztői beállításokkal rendelkező verziót szeretnénk használni. A jegyzet használatához az „Eclipse IDE for Java Developers” verzió elegendő.


A következő lépésben meg kell adnunk, hogy az Eclipse melyik JVM-et használja a futáskor, valamint, hogy melyik könyvtárba települjön.

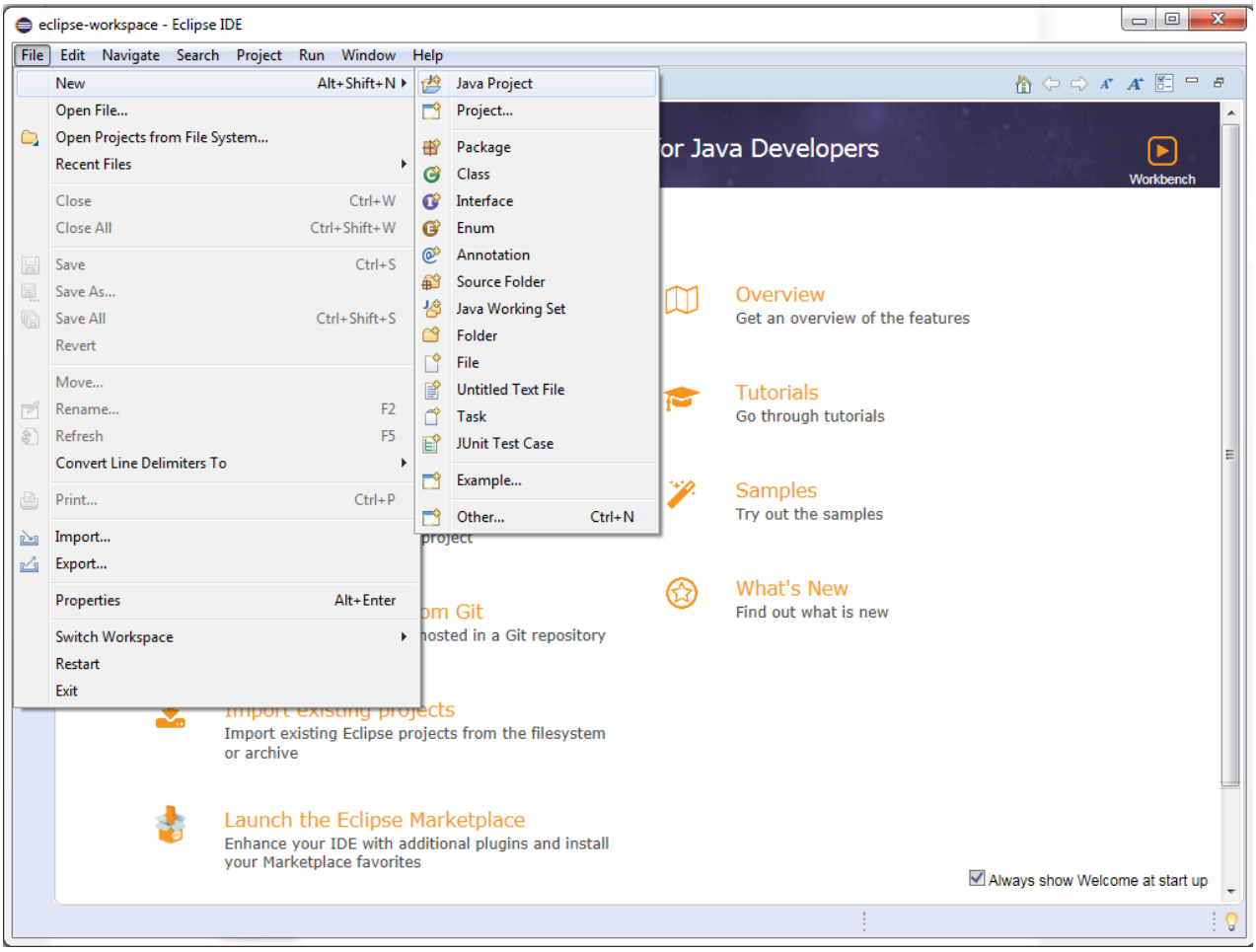



A telepítő a telepítési folyamat során rákérdez a licenszekre, melyeket el kell fogadnunk a sikeres telepítéshez. Az Eclipse az első indulásakor rákérdez a wokspace helyére. A terminológiában ez azt a könyvtárat jelenti, ahol az Eclipse a projekteket kezeli.



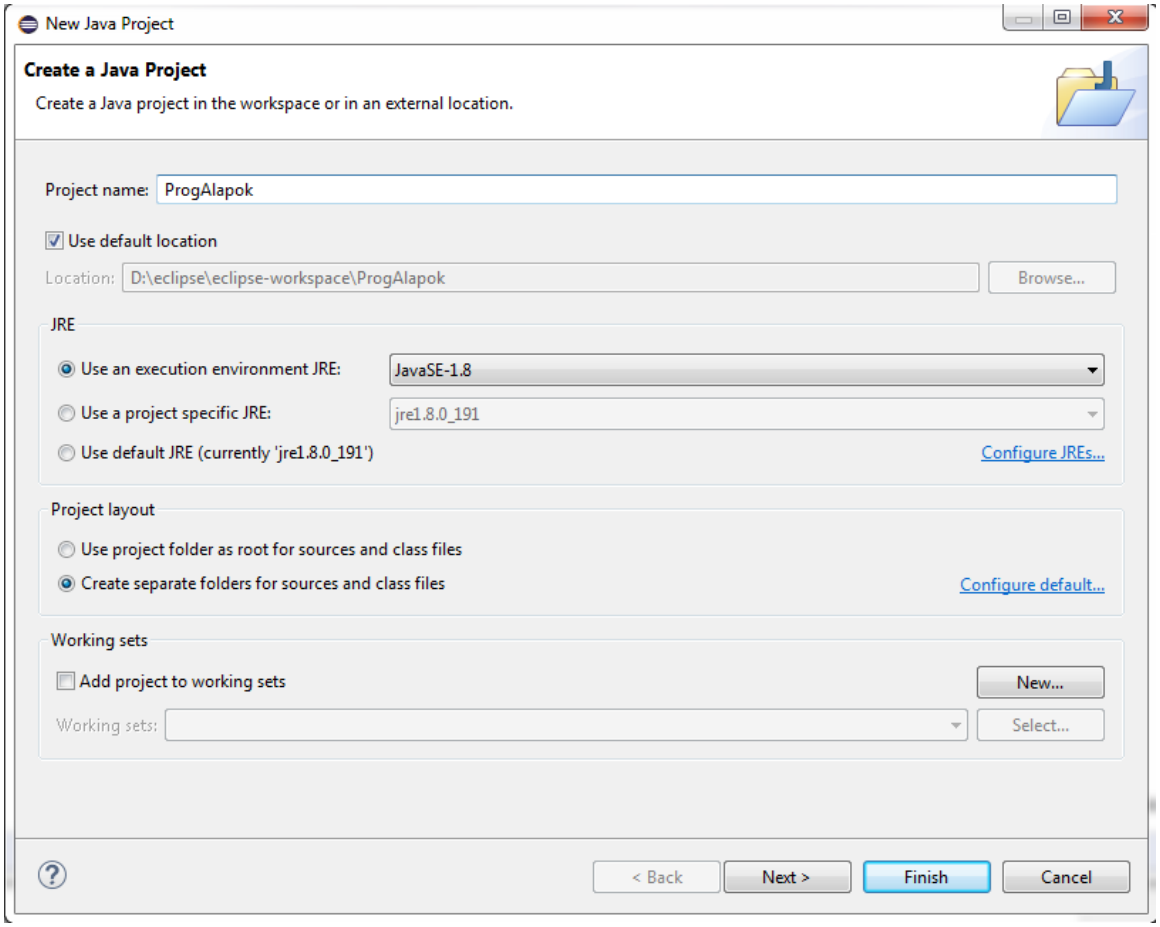

A jegyzetben szereplő forráskódok futtatásához a következő lépéseket kell végrehajtani:

1. Készítsünk egy új Java projektet (**File->New->Java** project menüpont).

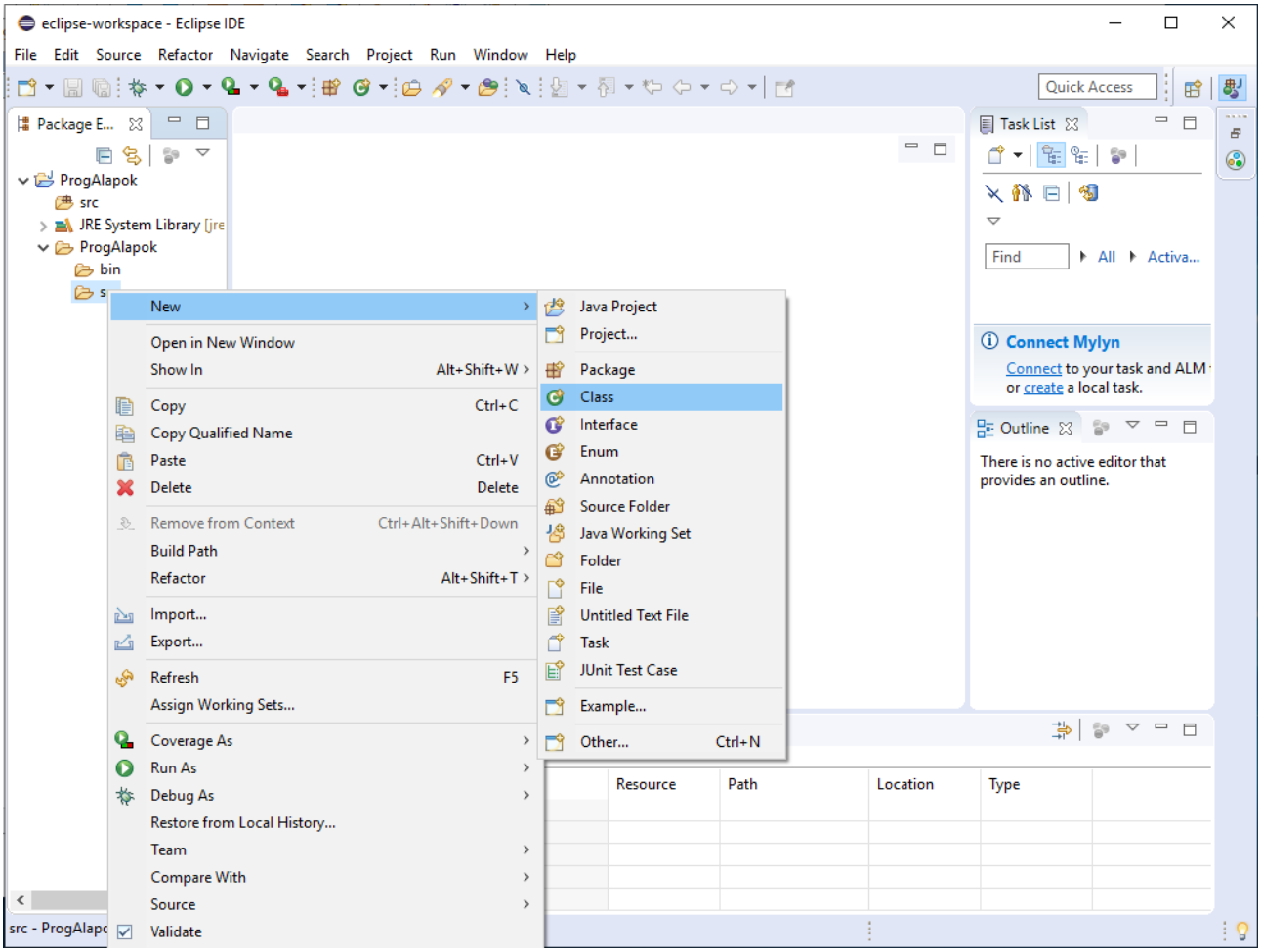



A felbukkanó **Create a Java Project** dialógus ablakban adjuk meg a projekt nevét. A projekt név megadásánál ne használjunk szóközt, valamint magyar ékezetes betűket. Továbbá a projektnév ne kezdődjön számmal. Célszerű a projekt nevet úgy választani, hogy az utalja a

a feladatra, amelyet a projektben megszeretnénk valósítani. A dialógus ablak többi beállításait hagyjuk az alapértelmezett értékeken. Kattintsunk a **Finish** gombra.




Adjunk egy új **Java classt**- a projekthez. A projekt **src** bejegyzésén jobb kattitntás-> New ->Class. A felbukkanó **New Java Class** dialógus ablakban adjuk meg a projekt nevét. A többi beállítást hagyjuk alapértelmezett esetben, majd kattintsunk a **Finish** gombra.


eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package E...
ProgAlapok
src
JRE System Library [jre
ProgAlapok
bin
Task List
Find All Activa...
Connect Mylyn
Connect to your task and ALM or create a local task.
Outline
There is no active editor that provides an outline.
New
Open in New Window
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Coverage As
Run As
Debug As
Restore from Local History...
Team
Compare With
Source
Validate
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
Task
JUnit Test Case
Example...
Other... Ctrl+N
Resource
Path
Location
Type
src - ProgAlapc

## A feladatok megoldáshoz szükséges Java alapismeretek

### Változók értékeinek kiírása a konzolra

```
public class MaxValtozok {

    public static void main(String[] args)
    {

        byte largestByte = Byte.MAX_VALUE;
        short largestShort = Short.MAX_VALUE;
        int largestInteger = Integer.MAX_VALUE;
        long largestLong = Long.MAX_VALUE;
        float largestFloat = Float.MAX_VALUE;
        double largestDouble = Double.MAX_VALUE;

        System.out.println("A legnagyobb byte érték: " + largestByte);
        System.out.println("A legnagyobb short érték: " + largestShort);
        System.out.println("A legnagyobb integer érték: " + largestInteger);
        System.out.println("A legnagyobb long érték: " + largestLong);
        System.out.println("A legnagyobb float érték: " + largestFloat);
        System.out.println("A legnagyobb double érték: " + largestDouble);

    }

}
```

A program futásának eredménye:

A legnagyobb byte érték: 127

A legnagyobb short érték: 32767

A legnagyobb integer érték: 2147483647

A legnagyobb long érték: 9223372036854775807

A legnagyobb float érték: 3.4028235E38

A legnagyobb double érték: 1.7976931348623157E308

**Formázott adatkiírás**

```
public class formatoutput
{
	public static void main(String args[])
	{
	 int x = 100;
	 System.out.printf("Printing simple integer: x = %d\n", x);

	 // this will print it upto 2 decimal places
	 System.out.printf("Formatted with precison: PI = %.2f\n", Math.PI);

	 float n = 5.2f;

	 // automatically appends zero to the rightmost part of decimal
	 System.out.printf("Formatted to specific width: n = %.4f\n", n);

	 n = 2324435.3f;

	 // here number is formatted from right margin and occupies a
	 // width of 20 characters
	 System.out.printf("Formatted to right margin: n = %20.4f\n", n);
	}

}
```

A program futásának eredménye:

Printing simple integer: x = 100
Formatted with precison: PI = 3,14
Formatted to specific width: n = 5,2000
Formatted to right margin: n = 2324435,2500

## Adatbeolvasás a konzolról

```
import java.io.BufferedReader;
import java.io.InputStreamReader;


public class adatbeolvasas
{
       public static void main(String[] args)
       {
         try
       {
             //Peldanyositas
         BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

         //Szoveg bekeres
         System.out.print("Szöveg = ");
         String szoveg=br.readLine();

         //Egeszszam bekeres
         System.out.print("Egész szám= ");
         int N=Integer.parseInt(br.readLine());

         //Valosszam bekeres
         System.out.print("Egész szám= ");
         double a=Double.parseDouble(br.readLine());

       }
       catch(Exception ex)
       {
           ex.printStackTrace();
       }
        }
}
```

### Véletlen számok

Programozás során gyakran előfordul, hogy valamilyen értéket véletlenszerűen kell megadni, vagy egy tömböt vagy listát véletlen számokkal fel kell tölteni. Javaban véletlen számokat kétféle módon generálhatunk:

1. Használhatjuk a `Math.random()` metódust.
   A `Math.random()` metódus a [0-1) intervallumban generál egy valós (lebegőpontos) számot. Ekkor ha egy adott intervallumon belül kell generálni egy véletlen számot, azt a következő formulával tehetjük:

```
(int)(Math.random()*(felső-alsó+1))+alsó;
```

2. Használhatjuk a java.util.Random osztályt.

```
import java.util.Random;

public class veletlen
{
      public static void main(String[] args)
    {
            Random r=new Random();

            //egész véletlen szám [0-2^32) intervallumból
            int a=r.nextInt();

            //egész véletlen szám [0-100) intervallumból
            int b=r.nextInt(100);

            //valós véletlen szám [0-1) intervallumból
            double v=r.nextFloat();
    }
}
```

## Szöveges (szekvenciális) file olvasása

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;


public class FileOlvaso {

    public static void main(String[] args) {
        // TODO code application logic here

       try
       {
         File file = new File("test.txt");
         BufferedReader br = new BufferedReader(new FileReader(file));
         String st;
         while ((st = br.readLine()) != null) {
                System.out.println(st);
        }
       }
       catch(Exception ex)
       {
         ex.printStackTrace();
       }
    }

}
```

**Szekvenciális file írása:**

```
import java.io.FileWriter;

try
 {
  FileWriter  fw = new FileWriter("filenev.txt");
  fw.write("Itt több soros string is állhat.");
  fw.flush();
  fw.close();
 }
catch(Exception ex)
{
ex.printStackTrace();
}



import java.util.List;
import java.util.ArrayList;
```

**Láncolt lista használata**

```
public class testList
{

	public static void main(String[] args)
	{
	  //Példányositás a listaelem adatrésze egész szám
	  List<Integer> szamok= new ArrayList<Integer>();

	  //Feltöltjük a listát páros számokkal
	  for(int i=0; i<20; i+=2 )
	  {
		szamok.add(i);
	  }

	  //A harmadik elemet adtrészét 5-re változtatjuk
	  szamok.set(2, 5);

	  //Töröljük a lista 2. elemét
	  szamok.remove(1);

	  //a lista tartalma
	  for(int i=0; i<szamok.size(); i++ )
	  {
	      System.out.println(szamok.get(i));
	  }

	}
}
```

# Feladatok:

## 1. Feladat. ASCII kódtábla

**Készítsen egy olyan programot, mely az ASCII kódtáblát írja ki a konzolra.**

Megoldás:

```
public class ASCII
{
	public static void main(String args[])
	{
	 for(int i=0; i<=255; i++)
	 {
		System.out.printf("%c	0x%X	%3d\n", i,i,i);
	 }
	}
}
```

## 2. Feladat. Elsőfokú egyenlet megoldása

**Készítsen egy olyan programot, amely a valós számok halmazán kiszámolja az ax + b = cx + d elsőfokú egyenlet megoldását. A bemenő adatokat (a,b,c,d) a felhasználótól kérje be!**

```
import java.io.*;
public class elsofoku
{
        public static void main(String[] args) throws IOException
        {
          // TODO code application logic here
           float x;
          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
           System.out.print("a = "); float a=Float.parseFloat(br.readLine());
           System.out.print("b = "); float b=Float.parseFloat(br.readLine());
           System.out.print("c = "); float c=Float.parseFloat(br.readLine());
          System.out.print("d = "); float d=Float.parseFloat(br.readLine());
          if(a==c)
          {
              System.out.println("Nincs megoldas");
          }
           else
           {
              x = (d-b)/(a-c);
              System.out.println("x = "+ x);
           }

        }
    }
```

### 3. Feladat. Háromszög számok

**Készítsen egy olyan programot, amely az a,b,c számhármasról eldönti, hogy lehetnek-e egy háromszög oldalai!**

```
import java.io.*;
public class haromszog
{
        public static void main(String[] args) throws IOException
        {
          // TODO code application logic here
           BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
           System.out.print("a = "); float a=Float.parseFloat(br.readLine());
           System.out.print("b = "); float b=Float.parseFloat(br.readLine());
           System.out.print("c = "); float c=Float.parseFloat(br.readLine());

           if( (a+b)>c && (b+c)>a && (a+c)>b)
             System.out.println("Háromszög számok");
           else
             System.out.println("nem háromszög számok");

         }
}
```

## 4. Feladat. Másodfokú egyenlet

**Készítsen egy olyan programot, amely a másodfokú egyenlet megoldó képlete alapján a valós számkörben megadja egy $ax^2+bx+c$ alakú másodfokú egyenlet gyökeit! A bemenő adatokat (a,b,c) a felhasználótól kérje be!**

```
import java.io.*;
public class masodfoku
{
        public static void main(String[] args) throws IOException
        {
          // TODO code application logic here
           float x1;
           float x2;
          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
           System.out.print("a = "); float a=Float.parseFloat(br.readLine());
           System.out.print("b = "); float b=Float.parseFloat(br.readLine());
           System.out.print("c = "); float c=Float.parseFloat(br.readLine());
           if( a==0)
           {
            x1=(b==0 ? c : c/b);
            System.out.println("x1 = "+x1);
             return;
           }
           float d=b*b-4*a*c;
           if(d>=0)
           {
              d=Math.sqrt(d);
              x1=(-b+d)/(2*a);
              x2=(-b-d)/(2*a);
              System.out.println("x1 = "+ x1);
              System.out.println("x2 = "+ x2);
          }
          else
          {
              System.out.println("Nincs megoldas");
          }
```

```
      }
    }
```

## 5. Feladat. Faktoriális

**Egy N természetes szám faktoriálisát N! jelöli és az 1*2*3*...*N szorzatot értjük alatta. Készítsen programot N! kiszámítására. Vigyázat, N! értéke nagyon gyorsan nő az N függvényében!**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class faktorialis
{
  public static void main(String[] args)
  {
    try
    {
      System.out.print("N =");
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      int N=Integer.parseInt(br.readLine());
      int fakt=1;

      for(int i=2; i<=N; i++)
      {
            fakt*=i;
      }

      System.out.print("N! ="+fakt);
    }
    catch(Exception ex)
    {
      ex.printStackTrace();
    }

   }

 }
```

## 6. Feladat. Lotto számok

**Készítsen olyan programot, amely az ötös, hatos, és hetes lotto játékokhoz generál lotto számokat.**

Megoldás:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Random;

class lottoGen
{
    public int szamok[];

    public void generalo(int which)
    {
        szamok =new int[which];
        Random r=new Random();
        boolean vanmar;
        for(int i=0; i<which;)
        {
            switch(which)
            {
                case 5:szamok[i]=r.nextInt(90)+1;break;
                case 6:szamok[i]=r.nextInt(45)+1;break;
                case 7:szamok[i]=r.nextInt(35)+1;break;
            }

            vanmar=false;
            for(int j=0; j<i; j++)
            {
                if( szamok[i]==szamok[j])
                    vanmar=true;
            }
            if(vanmar==false)
                i++;
        }
    }
    public void print()
    {
       for(int i=0; i<szamok.length; i++)
           System.out.println(szamok[i]);
    }
}


public class Lotto
{

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

```
            // TODO code application logic here
            lottoGen generalo= new lottoGen();

        BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Melyik lotto:");
        try
        {
            String s=br.readLine();
            s=s.trim();
            generalo.generalo(Integer.parseInt(s));
            generalo.print();
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }
    }

}
```

## 7. Feladat. Maximum és minimum érték keresés

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 100 elemű egész típusú egydimenziós tömböt, majd megkeresi és kiírja a konzolra a tömbben levő legnagyobb és legkisebb számot!**

```
import java.util.Random;
public class tomb
{
        public static void main(String[] args)
        {
          int tomb[];
          int maximum;
          int minimum;

          tomb= new int[100];
          Random rnd=new Random();
          for( int i=0; i<100; i++)
          {
            tomb[i]=rnd.nextInt(1000);
          }
          maximum=minimum=tomb[0];
          for(int j=1; j<100; j++)
          {
              if( tomb[j]<minimum)
                  minimum=tomb[j];
              if(tomb[j]>maximum)
                  maximum=tomb[j];
          }
          System.out.println("Minimum ="+ minimum);
          System.out.println("Maximum ="+ maximum);
        }
}
```

## 8. Feladat. Maximum és minimum index keresés

**Készítsen egy olyan programot, amely a felhasználó által megadott adatokkal tölt fel egy 10 elemű egész típusú egydimenziós tömböt, majd megkeresi és kiírja a konzolra a tömbben levő legnagyobb és legkisebb elem indexét!**

```
public class indexex
{
	public static void main(String[] args)
	{
		int tomb[];
		int maximum;
		int minimum;
		int maximum_index;
		int minimum_index;

		tomb= new int[10];
		tomb[0]=20;
		tomb[1]=60;
		tomb[2]=96;
		tomb[3]=0;
		tomb[4]=-10;
		tomb[5]=22;
		tomb[6]=38;
		tomb[7]=12;
		tomb[8]=19;
		tomb[9]=65;

		maximum=minimum=tomb[0];
		maximum_index=minimum_index=0;
		for(int j=1; j<10; j++)
		{
			if( tomb[j]<minimum)
			{
				minimum=tomb[j];
				minimum_index=j;
			}
			if(tomb[j]>maximum)
			{
				maximum=tomb[j];
				maximum_index=j;
			}
		}
		System.out.println("Minimum index="+ minimum_index);
		System.out.println("Maximum indeex="+ maximum_index);
    }
}
```

## 9. Feladat. Átlag számítás

**Készítsen egy olyan programot, amely a felhasználó által megadott adatokkal tölt fel egy 10 elemű egész típusú egydimenziós tömböt, majd kiszámítja a megadott számok átlagát!**

```
public class atlag
{
	public static void main(String[] args)
	  {
		int tomb[];
		float atl;

		tomb= new int[10];
		tomb[0]=20;
		tomb[1]=60;
		tomb[2]=96;
		tomb[3]=0;
		tomb[4]=-10;
		tomb[5]=22;
		tomb[6]=38;
		tomb[7]=12;
		tomb[8]=19;
		tomb[9]=65;

		atl=0;
		for(int j=1; j<tomb.length; j++)
		{
			atl+=tomb[j];
		}

		atl/=tomb.length;
		System.out.println("Átlag ="+ atl);

	  }
}
```

## 10. Feladat. Páros és páratlan indexű elemek

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 20 elemű egész típusú egydimenziós tömböt, kiszámítja a páros indexű elemek összegét valamint a páratlan indexű elemek szorzatát!**

```
import java.util.Random;

public class osszegszorzat
{

	public static void main(String[] args)
	{
		int tomb[];
		int osszeg;
		double szorzat;

		tomb= new int[20];
		Random rnd=new Random();
		osszeg=0;
		szorzat=1;
		for( int i=0; i<tomb.length; i++)
		{
			tomb[i]=rnd.nextInt(1000);

			System.out.println("tomb["+i+"] ="+tomb[i]);

			if(i%2==1)
				szorzat*=tomb[i];
			else
				osszeg+=tomb[i];
		}

		System.out.println("Összeg ="+ osszeg);
		System.out.println("Szorzat ="+ szorzat);
	}

}
```

## 11. Feladat. Tömben levő páratlan számok átlaga

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 20 elemű egész típusú egydimenziós tömböt, majd megkeresi és kiszámítja a tömbben tárolt páratlan számok átlagát!**

```
import java.util.Random;

public class paratlanszamokatlaga
{
        public static void main(String[] args)
         {
                int tomb[];
                double atlag;
                int db;

                tomb= new int[20];
                Random rnd=new Random();

                atlag=0;
                db=0;
                for( int i=0; i<tomb.length; i++)
                {
                        tomb[i]=rnd.nextInt(1000);

                        System.out.println("tomb["+i+"] ="+tomb[i]);

                        if(tomb[i]%2==1)
                        {
                                atlag+=tomb[i];
                                db++;
                        }
                }
                atlag/=db;
                System.out.println("Átlag ="+ atlag);
                System.out.println("Páratlan számok száma ="+db);
         }

}
```

## 12. Feladat. Szorzatösszeg

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 20 elemű egész típusú egydimenziós tömböt, majd kiszámítja a tömb első és második felébe eső számok szorzatösszegét!**

```
import java.util.Random;
public class feladat_11
{
	public static void main(String[] args)
	 {
		int tomb[];

		double osszeg=0;

		tomb= new int[20];
		Random rnd=new Random();

		for( int i=0; i<tomb.length; i++)
		{
			tomb[i]=rnd.nextInt(1000);
			System.out.println("tomb["+i+"] ="+tomb[i]);
		}

		for( int i=0; i<tomb.length; i++)
		{
			osszeg+=((double)tomb[i]*(double)tomb[tomb.length-1-i] );

		}

		System.out.println("Szorzatösszeg ="+osszeg);
	}
}
```

## 13. Feladat. Tömb feltöltése páros számokkal

**Készítsen egy olyan programot, amely az első 20 nemnegatív páros számmal tölt fel egy 20 elemű egész típusú egydimenziós tömböt!**

```
public class feladat_12
{

        public static void main(String[] args)
         {
                int tomb[];

                tomb= new int[20];
                int j=0;
                for(int i=0; i<40; i++)
                {
                        if( i%2==0)
                        {
                                tomb[j++]=i;
                        }
                }
                for(int i=0; i<20; i++)
                {
                        System.out.println("tomb["+i+"] ="+tomb[i]);
                }
         }
}
```

## 14. Feladat. Függvényértékek adott intervallumon

**Készítsen egy olyan programot, amely az $f(x) = -\frac{3}{2}x + 3$ hozzárendelési szabállyal megadott függvény $[4; 23]$ intervallumba eső egész számoknál felvett függvényértékekkel tölt fel egy 20 elemű valós típusú egydimenziós tömböt!**

```
public class feladat_13
{
	public static void main(String[] args)
	 {
		double tomb[];

		tomb= new double[20];
		int i=0;
		for(int x=4; x<=23; x++)
		{
			tomb[i++]=-1.5*x+3;
		}

		for(i=0; i<20; i++)
		{
			System.out.println("tomb["+i+"] ="+tomb[i]);
		}

	 }
}
```

## 15. Feladat. Maximum és minimum keresés két dimenziós tömben

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 100x20-as méretű egész típusú kétdimenziós tömböt, majd megkeresi és kiírja a konzolra a tömbben levő legnagyobb és legkisebb számot!**

```
import java.util.Random;
public class tomb2d
{
	public static void main(String[] args)
	{
		int tomb[][];
		int maximum;
		int minimum;

		tomb= new int[100][20];
		Random rnd=new Random();

		//Tömb feltöltése véletlen számokkal
		for( int sor=0; sor<100; sor++)
		{
			for(int oszlop=0; oszlop<20; oszlop++ )
			{
				tomb[sor][oszlop]=rnd.nextInt(1000);
			}
		}

		//Maximum és minimum keresés
		maximum=minimum=tomb[0][0];
		for( int sor=0; sor<100; sor++)
		{
			for(int oszlop=0; oszlop<20; oszlop++ )
			{
				if(tomb[sor][oszlop]>maximum)
					maximum=tomb[sor][oszlop];

				if(tomb[sor][oszlop]<minimum)
					maximum=tomb[sor][oszlop];
			}
		}
		System.out.println("Minimum ="+ minimum);
		System.out.println("Maximum ="+ maximum);

	}
}
```

## 16. Feladat. Sorminimumok átlaga kétdimenziós tömbben

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 100x20-as méretű egész típusú kétdimenziós tömböt, majd megkeresi és kiírja a konzolra a tömbben levő sorminimumok maximumát!**

```
import java.util.Random;
public class sorminumummaximuma
{
		public static void main(String[] args)
		{
			int tomb[][];
			int maximum;
			int sorminimum;

			tomb= new int[100][20];
			Random rnd=new Random();



			//Tömb feltöltése véletlen számokkal
			for( int sor=0; sor<100; sor++)
			{
				for(int oszlop=0; oszlop<20; oszlop++ )
				{
					tomb[sor][oszlop]=rnd.nextInt(1000);
				}
			}


			maximum=0;
			for( int sor=0; sor<100; sor++)
			{
				sorminimum=tomb[sor][0];
				for(int oszlop=1; oszlop<20; oszlop++ )
				{
					if(tomb[sor][oszlop]<sorminimum)
					{
						sorminimum=tomb[sor][oszlop];
					}
				}
				if(sorminimum>maximum)
					maximum=sorminimum;

			}
			System.out.println("Sorminumum maximuma ="+ maximum);

		}
 }
```

## 17. Feladat. Sorátlagok maximuma kétdimenziós tömbben

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 100x20-as méretű egész típusú kétdimenziós tömböt, majd megkeresi és kiírja a konzolra a tömbben levő sorátlagok maximumát!**

```
import java.util.Random;
public class sormatlagokmaxumuma
{
		public static void main(String[] args)
		{
			int tomb[][];
			float maximum;
			float atlag;

			tomb= new int[100][20];
			Random rnd=new Random();

			//Tömb feltöltése véletlen számokkal
			for( int sor=0; sor<100; sor++)
			{
				for(int oszlop=0; oszlop<20; oszlop++ )
				{
					tomb[sor][oszlop]=rnd.nextInt(1000);
				}
			}


			maximum=0;
			for( int sor=0; sor<100; sor++)
			{
				atlag=tomb[sor][0];
				for(int oszlop=1; oszlop<20; oszlop++ )
				{
					atlag+=tomb[sor][oszlop];
				}
				atlag/=20;
				if(atlag>maximum)
					maximum=atlag;

			}
			System.out.println("Sor atlag maximuma ="+ maximum);

		}
 }
```

## 18. Feladat. Sormaximumok minimuma kétdimenziós tömben

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 100x20-as méretű egész típusú kétdimenziós tömböt, majd megkeresi és kiírja a konzolra a tömbben levő sormaximumok minimumát!**

```
import java.util.Random;
public class sorminumummaximuma
{
		public static void main(String[] args)
		{
			int tomb[][];
			int minumum;
			int sormaximum;

			tomb= new int[100][20];
			Random rnd=new Random();



			//Tömb feltöltése véletlen számokkal
			for( int sor=0; sor<100; sor++)
			{
				for(int oszlop=0; oszlop<20; oszlop++ )
				{
					tomb[sor][oszlop]=rnd.nextInt(1000);
				}
			}


			maximum=0;
			for( int sor=0; sor<100; sor++)
			{
				sormaximum=tomb[sor][0];
				for(int oszlop=1; oszlop<20; oszlop++ )
				{
					if(tomb[sor][oszlop]>sormaximum)
					{
						sormaximum=tomb[sor][oszlop];
					}
				}
				if(sormaximummum<minimum)
					minimum=sormaximum;

			}
			System.out.println("Sormaximumok minimuma ="+ minimum);

		}
 }
```

## 19. Feladat. Buborékrendezés

**Készítsen egy olyan programot, amely 0 és 1000 közé eső véletlen számokkal tölt fel egy 100 elemű egész típusú egydimenziós tömböt, majd buborékrendezéssel növekvő sorrendbe helyezi a tömb elemeit!**

```
import java.util.Random;
public class bubblesort
{
		public static void bubble( int a[], int size )
		{
			int i;
			int j;
			int tmp;
			for (i=size-1; i>0; --i)
			{
				for (j=0; j < i; ++j)
				{
					if (a[j]>a[j+1])
					{
						// csere
						tmp=a[j];
						a[j]=a[j+1];
						a[j+1]=tmp;
					}
				}
			}
		}

		public static void main(String[] args)
		{
			int tomb[];


			tomb= new int[100];
			Random rnd=new Random();

			//Tömb feltöltése véletlen számokkal
			for( int i=0; i<100; i++)
			{
				tomb[i]=rnd.nextInt(1000);
			}

			bubble(tomb,100);

			for( int i=0; i<100; i++)
			{
				System.out.println(tomb[i]);
			}

		}
	}
```

## 20. Feladat. Minimum kiválasztásos rendezés

**Készítsen egy olyan programot, amely -1000 és 1000 közé eső véletlen számokkal tölt fel egy 100 elemű egész típusú egydimenziós tömböt, majd minimum kiválasztásos rendezéssel növekvő sorrendbe helyezi a tömb elemeit!**

```
import java.util.Random;

public class Rendezes
{
	public static void main(String[] args)
	{
		int tomb []  =new int[100];
	    Random r=new Random();
	    for (int i = 0; i < tomb.length; i++)
	    {
	      tomb[i]=r.nextInt(2000)-1000;

	      System.out.println(tomb[i]);
	    }

	    System.out.println("----------------------------------------");

	    // rendezes
	    for (int i = 0; i < tomb.length; i++)
	    {
	      int min = tomb[i];
	      int ind = i;
	      for (int k = i; k < tomb.length; k++)
	      {
	            if (tomb[k] < min)
	            {
	                  min = tomb[k];
	                  ind = k;
	            }
	      }

	      int v = tomb[i];
	      tomb[i] = min;
	      tomb[ind] = v;
	    }

	    //rendezett tomb kiirasa
	    for (int i = 0; i < tomb.length; i++)
	    {
	      System.out.println(tomb[i]);
	    }
	}
}
```

## 21. Feladat. Legközelebbi szám keresése egydimenziós tömben

**A [1,100] intervallumba eső véletlen számokkal töltsünk fel egy 20 elemű tömböt. Határozzuk meg azt a tömbelemet, amely a legközelebb van a generált számok számtani közepéhez.**

```
import java.util.Random;

public class Legkozelebb
{

	public static void main(String[] args)
	{
		int tomb []  =new int[20];
		Random r=new Random();


		float szk = 0;
		for (int i = 0; i < tomb.length; i++)
		{
		   tomb[i]=r.nextInt(100)+1;
		   szk += tomb[i];
		   System.out.println(tomb[i]);
		}

		szk /= tomb.length;


		float d = Math.abs(szk - tomb[0]);
		int index = 0;
		for (int i = 0; i < tomb.length; i++)
		{
		   float d2 = Math.abs(szk - tomb[i]);
		   if(d2<d)
		   {
			d = d2;
			index = i;
		   }

		}

		System.out.println("Szamtani kozep = "+szk);
		System.out.println("Hozza legkozelebb = " + tomb[index]);
	}
}
```

## 22. Feladat. Magánhangyók száma szövegben

**Készítsen egy olyan programot, amely bekér a felhasználótól egy szöveget, és megszámolja, hogy hány magánhangzó van benne.**

```
import java.io.*;

public class Maganhangzo
{

        public static void main(String[] args) throws Exception
        {
            // TODO code application logic here
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Mondat:");
            String szoveg=br.readLine();

            String magan="aáeéiíoöőuúüűó";
            int szam[]= new int[magan.length()];

            for(int i=0; i<szam.length; i++)
                   szam[i]=0;

            for(int i=0; i<szoveg.length(); i++)
            {
               for(int j=0; j<magan.length();j++)
               {
                  if(szoveg.charAt(i)==magan.charAt(j))
                            szam[j]++;
               }
            }

            for(int i=0; i<magan.length(); i++)
            {
                   System.out.println(magan.charAt(i)+": "+szam[i]);
            }
         }
}
```

## 23. Feladat. Betűcsere szövegben

**Kérjünk be a felhasználótól egy szöveget, és a szövegben levő összes 'a' betűt cseréljük le 'e' betűre.**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class betucsere
{
  public static void main(String[] args)
  {
    try
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

         String mondat=br.readLine();
         mondat.replace('a', 'e');
         mondat.replace('A', 'e');
         System.out.println(mondat);
    }
    catch(Exception ex)
    {
         ex.printStackTrace();
     }

  }

}
```

## 24. Feladat. Palindrom ellenőrző

**A palindrom olyan szó vagy szókapcsolat, amely visszafelé olvasva is ugyanaz. Készítsen egy olyan programot, amely bekér a felhasználótól egy mondatot, és eldönti, hogy palindrom-e!**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Palindrom
{

    public static void main(String[] args)
    {
       try
         {
       BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
       System.out.print("Mondat:");
        String szoveg=br.readLine();

        szoveg=szoveg.toLowerCase();
        System.out.println(szoveg);
        szoveg=szoveg.replace(" ","");
        System.out.println(szoveg);

        String sz="";
        for( int i=szoveg.length()-1; i>=0; i--)
        {
            sz+=szoveg.charAt(i);
        }

        System.out.println(sz);

        if(szoveg.compareTo(sz)==0)
          System.out.println("Palindrom");
        else
         System.out.println("Nem palindrom");

        int db=0;
        for(int i=0; i<szoveg.length(); i++)
        {
            if(szoveg.charAt(i)=='e')
                db++;
        }
         System.out.println("e betu: "+db);
         }
         catch(Exception ex)
         {
                ex.printStackTrace();
         }
    }

}
```

## 25. Feladat. Szöveg szavakra bontása

**Készítsen egy olyan programot, amely bekér a felhasználótól egy mondatot, és kiírja konzolra azokat a szavakat, melyeknek a második betűje 'e'!**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class szavakszama
{
	public static void main(String[] args)
	{
	 try
	 {
	  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
	  String mondat=br.readLine();

	  String szavak[]=mondat.split(" ");

	  for(int i=0; i<szavak.length; i++)
	  {
	   if(szavak[i].length()>1 && szavak[i].charAt(1)=='e')
	   {
	    System.out.println(szavak[i]);
	   }
	  }
	 }
	 catch(Exception ex)
	 {
	    ex.printStackTrace();
	  }
	}
}
```

## 26. Feladat. Szöveg szavainak tárolása láncolt listában

**Készítsen egy olyan programot, amely bekér a felhasználótól egy mondatot, és a mondat szavait fordított sorrendben nagybetűs alakban egy láncolt listában helyezi el!**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.List;
import java.util.ArrayList;

public class szolista
{
       public static void main(String[] args)
       {
        try
        {
          List<String> szavakLista = new ArrayList<String>();
          BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
          String mondat=br.readLine();

          String szavak[]=mondat.split(" ");

          for(int i=szavak.length-1; i>=0; i--)
          {
              szavakLista.add(szavak[i].toUpperCase());
          }

          for(int i=0; i<szavakLista.size(); i++)
          {
              System.out.println(szavakLista.get(i));
          }
         }
         catch(Exception ex)
         {
              ex.printStackTrace();
         }
       }
}
```

# Összetett feladatok

## 1. Feladat. Rómaiból arab szám

**Írjon olyan programot, amely római számot konvertál arab számra! A program kérje be a római számot, s az eredményt írja ki a konzolra!**

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;


public class Romai
{

        public static void main(String[] args)
        {
           BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
           try
             {
                    int arab=0;
                    int szam=0;
                    System.out.print("Romai szam: ");
                    String romai=br.readLine();
                    romai.toUpperCase();
                    for(int i=romai.length()-1; i>=0; i--)
                    {
                        switch(romai.charAt(i))
                        {
                            case 'I':arab=1; break;
                            case 'V':arab=5;
                                     if(i>0)
                                     {
                                       if(romai.charAt(i-1)=='I')
                                        {
                                            arab=4;
                                            i--;
                                        }
                                     }
                                     break;
                            case 'X':arab=10;
                                     if(i>0)
                                     {
                                        if(romai.charAt(i-1)=='I')
                                        {
                                            arab=9;
                                            i--;
                                        }
                                     }
                                     break;
                            case 'L':arab=50;
                                     if(i>0)
                                     {
```

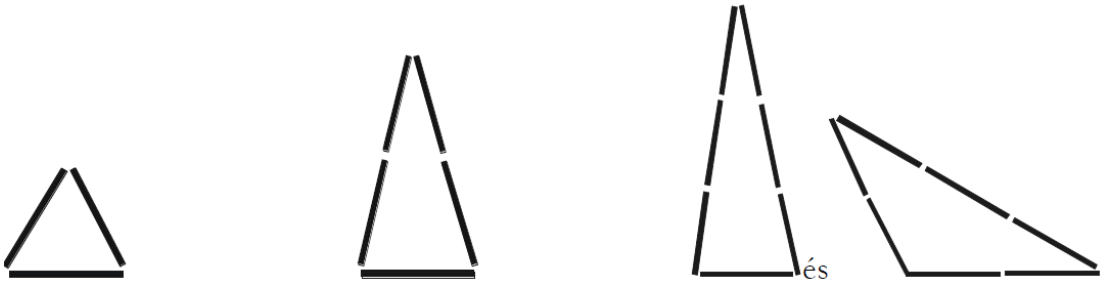
```
                                if(romai.charAt(i-1)=='X')
                                {
                                    arab=40;
                                    i--;
                                }
                            }
                            break;
                        case 'C':arab=100;
                            if(i>0)
                            {
                                if(romai.charAt(i-1)=='X')
                                {
                                    arab=90;
                                    i--;
                                }
                            }
                            break;
                         case 'D':arab=500;
                            if(i>0)
                            {
                                if(romai.charAt(i-1)=='C')
                                {
                                    arab=400;
                                    i--;
                                }
                            }
                            break;

                        case 'M':arab=1000;
                            if(i>0)
                            {
                                if(romai.charAt(i-1)=='C')
                                {
                                    arab=900;
                                    i--;
                                }
                            }
                            break;

                    }
                    szam+=arab;

                }
                System.out.println(szam);
            }
            catch(IOException ex)
            {
                ex.printStackTrace();
            }
        }
    }
```

## 2. Feladat. Gyufaszálak

**Írjon programot, amely megadja, hogy N gyufaszálból hány különböző háromszöget lehet összerakni!**

*Példák:*

N=3-ra 1-féle N=5-re 1-féle N=7-re 2-féle




A program kérje be gyufaszálak számát (3≤N≤100).

Megoldásként annyi sort kell írni, ahányféle különböző háromszög rakható ki az N gyufaszálból! Mindegyik sorban három egész szám legyen szóközzel elválasztva, a háromszög egyes oldalainak hossza gyufaszálak darabszámában megadva! Ha a megadott gyufaszálakból nem rakható ki háromszög, akkor nem szabad kiírni semmit! (forrás: mester.inf.elte.hu)

Példa

Gyufaszálak száma: 7

Háromszögek:

1 3 3

2 2 3

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class gyufa
{
  public static void main(String[] args)
  {
    try
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
         System.out.print("N = ");
         int N=Integer.parseInt(br.readLine());

         int a;
         int b;
         int c;

          for(a=1; a<N/2; a++)
          {
              for(b=a; b<N; b++)
              {
                 c=N-(a+b);
                 if( (a+b)>c && (a+c)>b && (b+c)>a && (c>=b) )
                 {
                     System.out.println(a+" "+b+" "+c);
                 }

              }
           }
  }
  catch(Exception ex)
  {
    ex.printStackTrace();
  }

        }
}
```

### 3. Feladat. Erősen páratlan számok

**„Erősen páratlan” számoknak nevezzük azokat a páratlan számokat, amelyek vagy egyjegyűek, vagy pedig a számjegyeik összege páratlan szám. Írjon programot, amely beolvas egy természetes számot, majd kiírja, hogy az erősen páratlan-e vagy sem! Ha nem erősen páratlan, akkor írja ki a számjegyeinek összegét is.**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class erosenparatlan
{
  public static void main(String[] args)
  {
    try
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
         System.out.print("N = ");
         String szam=br.readLine();
         int N=Integer.parseInt(szam);
         if(N<=9)
         {
              System.out.println("EROSEN PARATLAN");
         }
         else
         {

              int osszeg=0;
              for(int i=0;i<szam.length(); i++)
              {
                    osszeg+=szam.charAt(i)-'0';
              }
              if(osszeg%2!=0)
              {
                    System.out.println("EROSEN PARATLAN");
              }
              else
              {
                    System.out.println("NEM EROSEN PARATLAN");
                    System.out.println(osszeg);
              }
         }
          }
          catch(Exception ex)
          {
                ex.printStackTrace();
          }
       }
}
```

## 4. Feladat. Húsvét számítás

**Írjon programot, mely kiszámítja, hogy mely dátumokra esik a húsvét! A számításokhoz a Gauss módszert használja!**

Az év sorszámát jelöljük *Y*-nal, az azt követő *mod* jelölés az osztás maradékát jelenti az utána álló egész számmal való osztáskor (például 13 mod 5 = 3) Először számoljuk ki *a*, *b* és *c* egész számokat:

$a = Y \bmod 19$

$b = Y \bmod 4$

$c = Y \bmod 7$

Majd ezekből a következő értékeket:

$d = (19a + M) \bmod 30$

$e = (2b + 4c + 6d + N) \bmod 7$

A Julianus-naptár szerint (melyet a keleti egyházak használnak) $M = 15$ és $N = 6$, a Gergely-naptár szerint (melyet a nyugati egyházak használnak) *M* és *N* a következő táblázatból kapható:

| Évek | M | N |
|---|---|---|
| 1583–1699 | 22 | 2 |
| 1700–1799 | 23 | 3 |
| 1800–1899 | 23 | 4 |
| 1900–2099 | 24 | 5 |
| 2100–2199 | 24 | 6 |
| 2200–2299 | 25 | 0 |

Ha $d + e < 10$ akkor márciusnak $(d + e + 22)$-edik napja húsvét, különben április $(d + e - 9)$-edik napja.

A következő kivételeket kell figyelembe venni:

- Ha április 26-át kapunk, akkor a húsvét április 19-én van.
- Ha április 25-ét kapunk, $d = 28$, $e = 6$, és $a > 10$ értékekkel, akkor húsvét április 18-ára fog esni.

Megoldás:

```
import java.io.*;

public class Husvet
{

   public static void main(String[] args) throws IOException
   {
      int ev;
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Ev = "); ev=Integer.parseInt(br.readLine());
      System.out.println(husvet(ev));

    }

    public static String husvet(int ev)
    {
            int a;
            int b;
            int c;
            int d;
            int e;

            int M=24;
            int N=5;

            a=ev%19;
            b=ev%4;
            c=ev%7;


            if(ev>=1583 && ev<=1699)
            {
               M=22;
               N=2;
            }
            if(ev>=1700&&ev<=1799)
            {
               M=23;
                 N=3;
            }
            if(ev>=1800&&ev<=1899)
            {
               M=23;
                 N=4;
            }
```

```
            if(ev>=1900&&ev<=2099)
            {
                M=24;
                N=5;
            }
            if(ev>=2100 && ev<=2199)
            {
                M=24;
                N=6;
            }
            if(ev>=2200 && ev<=2299)
            {
                M=25;
                N=0;
            }
            d=(19*a+M)%30;
            e=(2*b+4*c+6*d+N)%7;
            String ret="";
            if((d+e)<10)
            {
               ret=Integer.toString(ev)+". marcius "+Integer.toString(d+e+22);
            }
            else
            {
               int v=d+e-9;
                ret=Integer.toString(ev)+". aprilis "+Integer.toString(v);
                if(v==26)
                {
                     ret=Integer.toString(ev)+". aprilis 19.";
                }
                if(v==25&&d==28&&e==6&&a>10)
                {
                       ret=Integer.toString(ev)+". aprilis 18.";
                }
            }
          return ret;
        }

}
```

## 5. Feladat. Szókitaláló játék

**Készítsen egy szókitaláló programot! A programban egy 30 elemű string tömböt töltsön fel szavakkal. A játék indulásakor a program véletlenszerűen válasszon egy szót a tömbből, majd kérje be a játékos tippjét. A tippet betűnként hasonlítsa össze a kitalálandó szóval. Amennyiben az adott betűpozíción egyezés van, akkor ott írja ki a betűt. Minden más esetben pedig a '.' jelet. A játékos számára pontosan annyi tipp lehetőséget biztosítson, amennyi betűből áll a kitalálandó szó. Ha a játékos a lehetőségek számán belül kitalálta, akkor írja ki, hogy „NYERT". Ellenkező esetben pedig „NEM NYERT".**

```
import java.io.*;
import java.util.Random;

class Jatek
{
	private String szotar[];
	public Jatek()
	{
		szotar= new String[30];
		szotar[0]="alma";
		szotar[1]="korte";
		szotar[2]="barack";
		szotar[3]="mama";
		szotar[4]="asztal";
		szotar[5]="szekreny";
		szotar[6]="android";
		szotar[7]="windows";
		szotar[8]="linux";
		szotar[9]="szilva";
		szotar[10]="kelkaposzta";
		szotar[11]="ajto";
		szotar[12]="ablak";
		szotar[13]="napilap";
		szotar[14]="tenisz";
		szotar[15]="teve";
		szotar[16]="oroszlan";
		szotar[17]="pap";
		szotar[18]="iskola";
		szotar[19]="agilis";
		szotar[20]="java";
		szotar[21]="informatika";
		szotar[22]="telefon";
		szotar[23]="fazek";
		szotar[24]="labas";
		szotar[25]="villany";
		szotar[26]="miniszter";
		szotar[27]="kakao";
		szotar[28]="cukor";
		szotar[29]="lotto";

	}
	public void Run()throws IOException
	{
```

```
        BufferedReader br= new BufferedReader(new InputStreamReader(System.in));

        boolean flag=true;
        while(flag)
        {
            Random rnd= new Random();
            int index= rnd.nextInt(30);
            int len=szotar[index].length();
            System.out.println("A kitalálandó szó hossza:"+Integer.toString(len));
            boolean jatek=true;
            while(jatek)
            {
                System.out.print("Tipp :");String tipp=br.readLine();
                if(tipp.length()!=len)
                {
                   System.out.println("Nem passzol a szó hossza!");
                   continue;
                }
                String valasz="";
                int db=0;
                for(int i=0; i<len; i++)
                {
                    char B=szotar[index].charAt(i);
                    if( B==tipp.charAt(i))
                    {
                        valasz+=B;
                    }
                    else
                    {
                        valasz+=".";
                        db++;
                    }
                }
                System.out.println(valasz);
                if(db==0)
                {
                    System.out.println("Talált");
                    jatek=false;
                }
            }
            System.out.print("Kilépés (igen/nem)");String s=br.readLine();
            if(s.compareToIgnoreCase("igen")==0)
                flag=false;

        }
    }
}


public class Szokitalalo
{
 /**
  * @param args the command line arguments
  */
```

```
    public static void main(String[] args) throws IOException
    {
        // TODO code application logic here
        Jatek jatek= new Jatek();
        jatek.Run();
    }
}
```

## 6. Feladat. 2019 május-júniusi emelt szintű írásbeli érettségi 4. feladat

**Zárófeladatként oldjuk meg a 2019 május-júniusi emelt szintű írásbeli érettségi 4. feladatát! Idemásoljuk a feladat pontos szövegét A feladat szövege és a megoldás ellenőrzéséhez szükséges adat fájl megtalálható az alábbi linken:**

https://www.oktatas.hu/kozneveles/erettsegi/feladatsorok/emelt_szint_2019tavasz/emelt_7nap

A feladat szövege:

Egy cég 10 olyan autóval rendelkezik, amelyet a dolgozók igénybe vehetnek az üzleti ügyeik intézésére. Az autókat akár többnapos útra is elvihetik, illetve egy autót egy nap több dolgozó is elvihet. A rendszer az autók parkolóból való ki- és behajtását rögzíti. A parkoló a hónap minden napján 7-23 óra között van nyitva, csak ebben az időszakban lehet elvinni és visszahozni az autókat. Az autót mindig annak a dolgozónak kell visszahoznia, amelyik elvitte. Egyszerre csak egy autó lehet minden dolgozónál.

Az `autok.txt` fájl egy hónap (30 nap) adatait rögzíti. Egy sorban szóközökkel elválasztva 6 adat található az alábbi sorrendben.

| | | |
|---|---|---|
| nap | egész szám (1-30) | a hónap adott napja |
| óra:perc | szöveg (óó:pp formátumban) | ki- vagy a behajtás időpontja |
| rendszám | 6 karakteres szöveg (CEG300-CEG309) | az autó rendszáma |
| személy azonosítója | gész szám (500-600) | az autót igénybe vevő dolgozó azonosítója |
| km számláló | egész szám | a km számláló állása |
| ki/be hajtás | egész szám (0 vagy 1) | a parkolóból kihajtáskor 0, a behajtáskor 1 |

A sorok száma legfeljebb 500. Az adatok a napok szerint, azon belül óra és perc szerint rendezettek. Továbbá tudjuk, hogy a hónap első napján a cég mind a tíz autója a parkolóban volt.

```
...
5   07:30  CEG300  590   30580  0
5   14:16  CEG300  590   30656  1
5   17:00  CEG300  534   30656  0
5   19:03  CEG300  534   30784  1
...
15  09:53  CEG308  543   35048  0
17  11:16  CEG308  543   35746  1
```

A példában látható, hogy a CEG300 rendszámú autót az 5. napon kétszer is elvitték. Először 7:30-kor vitték el és 14:16-kor hozta vissza az 590-es dolgozó. A kivitelkor a kilométerszámláló állása 30 580 km volt, amikor visszahozta 30 656 km volt. Másodszor 17:00-kor vitte el az 534-es dolgozó az autót és 19:03-kor hozta vissza. A CEG308 rendszámú autót pedig a 15. napon vitte el az 543-as dolgozó és a 17. napon hozta vissza. Készítsen programot, amely az `autok.txt` állomány adatait felhasználva az alábbi kérdésekre válaszol! A program forráskódját mentse `cegesauto` néven! (A program megírásakor a felhasználó által megadott adatok helyességét, érvényességét nem kell ellenőriznie, feltételezheti, hogy a rendelkezésre álló adatok a leírtaknak megfelelnek.) A képernyőre írást igénylő részfeladatok eredményének megjelenítése előtt írja a képernyőre a feladat sorszámát (például: `3. feladat`)! Ha a felhasználótól kér be adatot, jelenítse meg a képernyőn, hogy milyen értéket vár! Az ékezetmentes kiírás is elfogadott. Az eredmény megjelenítését és a felhasználóval való kommunikációt a feladatot követő minta alapján valósítsa meg!

1. Olvassa be és tárolja el az `autok.txt` fájl tartalmát!
2. Adja meg, hogy melyik autót vitték el utoljára a parkolóból! Az eredményt a mintának megfelelően írja a képernyőre!
3. Kérjen be egy napot és írja ki a képernyőre a minta szerint, hogy mely autókat vitték ki és hozták vissza az adott napon!
4. Adja meg, hogy hány autó nem volt bent a hónap végén a parkolóban!
5. Készítsen statisztikát, és írja ki a képernyőre mind a 10 autó esetén az ebben a hónapban megtett távolságot kilométerben! A hónap végén még kint lévő autók esetén az utolsó rögzített kilométerállással számoljon! A kiírásban az autók sorrendje tetszőleges lehet.

6. Határozza meg, melyik személy volt az, aki az autó egy elvitele alatt a leghosszabb távolságot tette meg! A személy azonosítóját és a megtett kilométert a minta szerint írja a képernyőre! (Több legnagyobb érték esetén bármelyiket kiírhatja.)

7. Az autók esetén egy havi menetlevelet kell készíteni! Kérjen be a felhasználótól egy rendszámot! Készítsen egy `X_menetlevel.txt` állományt, amelybe elkészíti az adott rendszámú autó menetlevelét! (Az X helyére az autó rendszáma kerüljön!) A fájlba soronként tabulátorral elválasztva a személy azonosítóját, a kivitel időpontját (nap. óra:perc), a kilométerszámláló állását, a visszahozatal időpontját (nap. óra:perc), és a kilométerszámláló állását írja a minta szerint! (A tabulátor karakter ASCII-kódja: 9.)

**Minta a szöveges kimenetek kialakításához:**

```
2. feladat
30. nap rendszám: CEG300
3. feladat
Nap: 4
Forgalom a(z) 4. napon:
12:50 CEG303 561 ki
19:17 CEG308 552 be
4. feladat
A hónap végén 4 autót nem hoztak vissza.
5. feladat
CEG300 6751 km
CEG301 5441 km
CEG302 5101 km
CEG303 7465 km
CEG304 6564 km
CEG305 5232 km
CEG306 7165 km
CEG307 6489 km
CEG308 6745 km
CEG309 1252 km
6. feladat
Leghosszabb út: 1551 km, személy: 506
7. feladat
Rendszám: CEG304
Menetlevél kész.
```

A `CEG304_menetlevel.txt` **fájl tartalma**:

```
...
588     21. 16:58       13452 km       23. 20:28       14335 km
512     24. 16:58       14335 km       26. 22:21       15041 km
504     27. 13:47       15041 km
```

Megoldás:

Összetett feladatoknál a megfelelően választott adatstruktúra a megoldás szempontjából döntő jelentőséggel bír. A feladat jellegéhez illeszkedő adatábrázolás a feladat megoldását egyszerűbb algoritmusokkal is megvalósíthatóvá teszik, mely értelemszerűen kevesebb hibázási lehetőséget és olvashatóbb, karbantarthatóbb forráskódot eredményez.

Ennél a feladatnál a kiinduló adatok az `autok.txt` fájlban egy 6 oszlopos táblázatban vannak. Mivel a feladat kiírása szerint a fájlban egy sorban egy autó mozgása van megadva, ezért az egy sorban lévő adatokat célszerű egy adatként kezelni, melynek hat darab részadata (attribútuma) van. Ez azt jelenti, hogy a feladatban egy új adattípust kell bevezetnünk. Jelöljük AUTO-val. Új adattípust a Java-ban egy új osztály bevezetésével lehet definiálni.

```
class AUTO
{
	public int nap;			// hónap napja
	public String idopont;		// ki vagy behajtás időpontja
	public String rendszam;		// kocsi rendszáma
	public int azonosito;		// sofőr
	public int km;			// km állás
	public int beki;		// be vagy kihajtás
}
```

Az AUTO adattípus felhasználásával a fájl tartalmát most már egy egydimenziós tömbben, vagy pedig láncolt listában tárolhatjuk. Mivel a feladat kiírása megadta, hogy maximálisan hány sor lehet az adat fájlban, ezért tömbben való tárolás is megfelelő. Programozói szemmel nézve a láncolt listával történő tárolás elegánsabb. A feladat megoldása során mi itt a láncolt listával történő adattárolást fogjuk használni. Magát a listát a publikus osztályunkban globális változóként hozzuk létre.

```
protected static List<AUTO> autok;
```

Most már jöhet az első részfeladat az `autok.txt` fájl beolvasása a memóriába. Ehhez egy statikus metódust írunk, mely a BufferedReader-t és a FileReader-t fogja használni.

```
public static void beolvas(String fajlnev)
{
try
	{
		// Példányositjuk a láncolt listát
		autok= new ArrayList<AUTO>();

		// Inicializáljuk a fájl olvasást
		File file = new File(fajlnev);
		 BufferedReader br = new BufferedReader(new FileReader(file));
		 String st;

		 // Soronként olvassuk a fájlt és feldolgozzuk
		  while ((st = br.readLine()) != null)
		  {
		       String s[]=st.split(" ");
		       AUTO a= new AUTO();
		       a.nap= Integer.parseInt(s[0]);
		       a.idopont=s[1];
		       a.rendszam=s[2];
		       a.azonosito=Integer.parseInt(s[3]);
		       a.km=Integer.parseInt(s[4]);
		       a.beki=Integer.parseInt(s[5]);
		       autok.add(a);
		  }
		}
		catch(Exception ex)
		{
		    ex.printStackTrace();
		}
	}
```

A második részfeladat szövegezése szerint arra vagyunk kíváncsiak, hogy melyik autót vitték el utoljára. Mivel a fájlban és így a listában is időrendben vannak a sorok, ezért a keresést a listában hátulról kezdjük és azt kell nézni, hogy a *beki* változó értéke hol lesz 1.

```
public static void feladat_2()
{
int index=autok.size()-1;

	while (autok.get(index).beki==1 && index > 0)
		index--;
	AUTO a=autok.get(index);

	System.out.println("2. feladat");
	System.out.println(a.nap +" nap rendszam "+a.rendszam);
}
```

A harmadik részfeladatban először bekérjük a felhasználótól, hogy melyik napra kíváncsi, majd a listán végig haladva kiírjuk azoknak az autónak az adatait, melyeknek a nap adata megegyezik az adott nappal.

```
public static void feladat_3()
{
try
        {
          System.out.println("3. feladat");
          System.out.print("Nap :");

          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
          int nap=Integer.parseInt(br.readLine());
          System.out.println("Forgalom a(z) "+nap+". napon");
          for(int i=0; i<autok.size(); i++)
          {
               if(autok.get(i).nap==nap)
               {
                 AUTO a=autok.get(i);
                 String s=a.beki==1 ? "be" : "ki";
                 System.out.println(a.idopont+" "+a.rendszam+" "+a.azonosito+" "+s);

               }
          }

        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }

}
```

A negyedik részfeladat megoldásánál abból indulunk ki, hogy maximum 10 db gépjármű lehet kint a forgalomban. Felveszünk egy 10 elemű tömböt, amely a bent vagy kint állapotát fogja tárolni. Ezt a tömböt a rendszámok utolsó számjegyét indexként használva feltöltjük az autók beki állapotával. A kint levő autók száma a tömbben levő 0 értékek száma lesz.

```
public static void feladat_4()
{

System.out.println("4. feladat");

        int db = 0;

        // egy nap egyszerre max 10 auto lehet kint
        int kint[] = new int [10];
```

```
        for (int i=0; i<10; i++)
                kint[i] = 0;

for(int i=0; i<autok.size(); i++)
        {
               AUTO a=autok.get(i);
              int index=Integer.parseInt(a.rendszam.substring(3)) %10;
              kint[index]=a.beki;
        }

        for (int i= 0; i<10; i++)
        {
            if (kint[i]==0)
                db++;
        }


System.out.println("A hónap végén "+ db+" autót nem hoztak vissza.");

}
```

Az ötödik részfeladat megoldásánál mind a 10 autóra két tömbben párosítjuk a ki és behajtásokat, majd kiírjuk konyolra ezek különbségeit.

```
public static void feladat_5()
{
        System.out.println("5. feladat");

        int autoki[] = new int[10];
         int autobe[] = new int[10];
         int szumkm;

         for (int i = 0; i < 10; i++)
               autoki[i] = 0;

         for(int i=0; i<autok.size(); i++)
         {
               AUTO a=autok.get(i);
               int index=Integer.parseInt(a.rendszam.substring(3)) %10;
               if (a.beki==0 && autoki[index] == 0)
                  autoki[index]=a.km;
               if (a.beki==1)
                 autobe[index] = a.km;
          }

          for (int i = 0; i <10; i++)
          {
              szumkm = autobe[i] - autoki[i];
              System.out.println("CEG30"+i+ " "+szumkm+"km");
          }

    }
```

A hatodik részfeladatnál először a rendszámok szerint rendezzük az *autok* listát. A rendezés miatt az azonos rendszámú ki-be hajtások egymás mellé kerülnek. Az autó által megtett távolság ezáltal két szomszédos azonos rendszámú megtett útjainak a különbsége. Ezen különbségek maximumához tartozó személy tette meg a legnagyobb távolságot.

```
public static void feladat_6()
{
System.out.println("6. feladat");
	String eredmeny = "";

	Collections.sort(autok, new Comparator<AUTO>()
	{
		@Override
		public int compare(AUTO a1, AUTO a2)
		{
			return a1.rendszam.compareTo(a2.rendszam);
		}
	});


	 int kmmax = 0;
	 int szemely = 0;
	 for (int i=1; i<autok.size(); i++)
	 {

if (autok.get(i).rendszam.compareTo(autok.get(i-1).rendszam) == 0 &&
autok.get(i).beki ==1)
		{
		    if (kmmax<autok.get(i).km - autok.get(i-1).km)
		    {
		        kmmax = autok.get(i).km - autok.get(i-1).km;
		        szemely = autok.get(i).azonosito;
		    }
		}
	  }
	  System.out.println("Leghosszabb út: " + kmmax + " km, személy: " +
szemely);


	}
```

A hetedik részfeladat megoldása:

```
public static void feladat_7()
	{
	 try
	 {
	  System.out.println("7. feladat");
	  System.out.print("Rendszám :");
```

```
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String rsz=br.readLine();

         FileWriter  fw = new FileWriter(rsz + "_menetlevel.txt");

         String kiir = "";
         for (int i=0; i<autok.size(); i++)
         {
           AUTO a=autok.get(i);
           if (a.rendszam.compareTo(rsz) == 0 && a.beki==0)
           {
               kiir = a.azonosito+"\t"+a.nap+". "+a.idopont+"\t"+a.km+" km";
              fw.write(kiir);
              kiir = "";
           }
           if (a.rendszam.compareTo(rsz) ==0 && a.beki==1)
           {
             kiir = "\t" + a.nap + ". " + a.idopont + "\t" +a.km + " km\n" ;
             fw.write(kiir);
             kiir = "";
           }
         }
         fw.flush();
         fw.close();

        }
       catch(Exception ex)
       {
          ex.printStackTrace();
       }

     }
```

Az érettségi feladat teljes megoldása:

```
package cegesauto;

import java.util.List;
import java.util.ArrayList;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.File;
import java.io.InputStreamReader;

class AUTO
{
       public int nap;                  // hónap napja
       public String idopont;           // ki vagy behajtás időpontja
       public String rendszam;          // kocsi rendszáma
       public int azonosito;            // soför
       public int km;                   // km állás
```

```
        public int beki;                // be vagy kihajtás
}

public class cegesuato
{
        protected static List<AUTO> autok;

        public static void beolvas(String fajlnev)
        {
              try
            {
                    // Példányositjuk a láncolt listát
                    autok= new ArrayList<AUTO>();

                    // Inicializáljuk a fájl olvasást
                    File file = new File(fajlnev);
                BufferedReader br = new BufferedReader(new FileReader(file));
                String st;

                // Soronként olvassuk a fájlt, képezzük a lista elemet, és
hozzáadjuk a listához
                while ((st = br.readLine()) != null)
                {
                        String s[]=st.split(" ");
                        AUTO a= new AUTO();
                        a.nap= Integer.parseInt(s[0]);
                        a.idopont=s[1];
                        a.rendszam=s[2];
                        a.azonosito=Integer.parseInt(s[3]);
                        a.km=Integer.parseInt(s[4]);
                        a.beki=Integer.parseInt(s[5]);
                        autok.add(a);
                }
            }
            catch(Exception ex)
            {
                ex.printStackTrace();
            }

        }

        public static void feladat_2()
    {
                int index=autok.size()-1;

           while (autok.get(index).beki==1 && index > 0)
                index--;
           AUTO a=autok.get(index);

           System.out.println("2. feladat");
           System.out.println(a.nap +" nap rendszam "+a.rendszam);
    }

        public static void feladat_3()
```

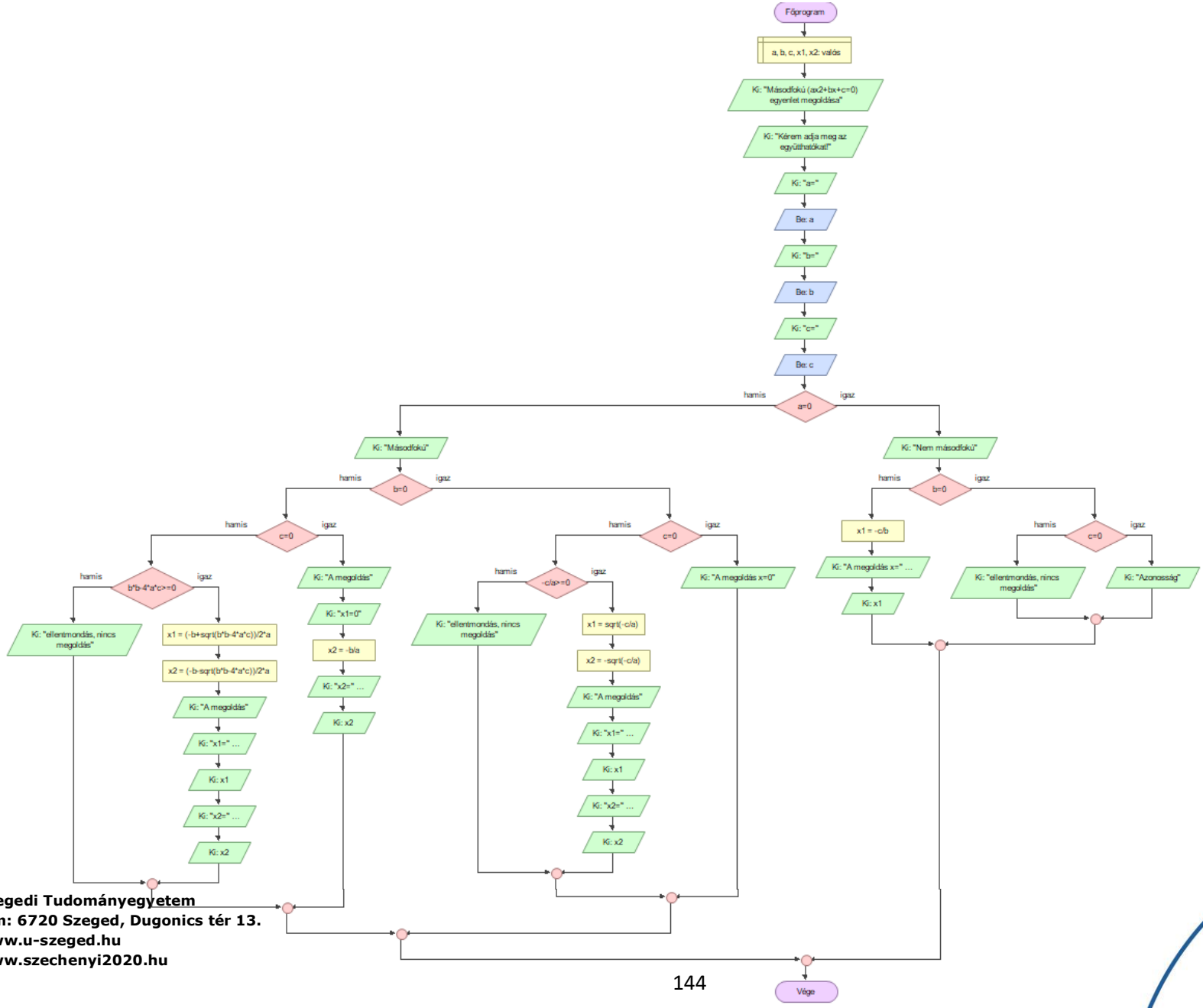
```
	{
		try
		{
			System.out.println("3. feladat");
			System.out.print("Nap :");

			BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
			int nap=Integer.parseInt(br.readLine());
			System.out.println("Forgalom a(z) "+nap+". napon");
			for(int i=0; i<autok.size(); i++)
			{
				if(autok.get(i).nap==nap)
				{
					AUTO a=autok.get(i);
					String s=a.beki==1 ? "be" : "ki";
					System.out.println(a.idopont+" "+a.rendszam+"
"+a.azonosito+" "+s);

				}
			}

		}
		catch(Exception ex)
	{
	  ex.printStackTrace();
	}


	}


public static void feladat_4()
	{

		System.out.println("4. feladat");

		int db = 0;

		// egy nap egyszerre max 10 auto lehet kint
		int kint[] = new int [10];

		for (int i = 0; i < 10; i++)
			kint[i] = 0;
		for(int i=0; i<autok.size(); i++)
		{
			AUTO a=autok.get(i);
			int index=Integer.parseInt(a.rendszam.substring(3)) %10;
			kint[index]=a.beki;
		}

		for (int i=0; i<10; i++)
		{
			if(kint[i]==0)
```

```
                db++;
            }


            System.out.println("A hónap végén "+ db+" autót nem hoztak vissza.");

        }

        public static void feladat_5()
        {
            System.out.println("5. feladat");

            int autoki[] = new int[10];
            int autobe[] = new int[10];
            int szumkm;

            for (int i = 0; i < 10; i++)
                autoki[i] = 0;

            for(int i=0; i<autok.size(); i++)
            {
                AUTO a=autok.get(i);
                int index=Integer.parseInt(a.rendszam.substring(3)) %10;
                if (a.beki==0 && autoki[index] == 0)
                    autoki[index]=a.km;
                if (a.beki==1)
                    autobe[index] = a.km;
            }

            for (int i= 0; i<10; i++)
            {
                szumkm = autobe[i] - autoki[i];
                System.out.println("CEG30"+i+ " "+szumkm+"km");
            }

        }

    public static void feladat_6()
        {
          System.out.println("6. feladat");
          String eredmeny = "";

           Collections.sort(autok, new Comparator<AUTO>()
           {
               @Override
               public int compare(AUTO a1, AUTO a2)
               {
                   return a1.rendszam.compareTo(a2.rendszam);
               }
           });


           int kmmax = 0;
           int szemely = 0;
```

```
            for (int i=1; i<autok.size(); i++)
            {

if (autok.get(i).rendszam.compareTo(autok.get(i-1).rendszam) == 0 &&
autok.get(i).beki ==1)
              {
                  if (kmmax<autok.get(i).km - autok.get(i-1).km)
                  {
                      kmmax = autok.get(i).km - autok.get(i-1).km;
                      szemely = autok.get(i).azonosito;
                  }
              }
           }
           System.out.println("Leghosszabb út: " + kmmax + " km, személy: " +
szemely);

       }

       public static void feladat_7()
       {
        try
        {
         System.out.println("7. feladat");
         System.out.print("Rendszám :");

         BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
         String rsz=br.readLine();

          FileWriter  fw = new FileWriter(rsz + "_menetlevel.txt");

          String kiir = "";
          for (int i=0; i<autok.size(); i++)
          {
            AUTO a=autok.get(i);
            if (a.rendszam.compareTo(rsz) == 0 && a.beki==0)
            {
                kiir = a.azonosito+"\t"+a.nap+". "+a.idopont+"\t"+a.km+" km";
               fw.write(kiir);
               kiir = "";
            }
            if (a.rendszam.compareTo(rsz) ==0 && a.beki==1)
            {
              kiir = "\t" + a.nap + ". " + a.idopont + "\t" +a.km + " km\n" ;
              fw.write(kiir);
              kiir = "";
            }
         }
         fw.flush();
         fw.close();

        }
       catch(Exception ex)
       {
          ex.printStackTrace();
```

```
        }

    }


    public static void main(String[] args)
    {
        beolvas("autok.txt");
        feladat_2();
        feladat_3();
        feladat_4();
        feladat_5();
        feladat_6();
        feladat_7();

    }

}
```

1. számú melléklet



## Felhasznált irodalom:


1. http://progalap.elte.hu/downloads/seged/eTananyag/lecke4_lap1.html
2. http://java.progtanulo.hu/java-programozas/3-valtozok/31-adattipusok
3. http://www.flowgorithm.org/documentation/index.htm
4. Szabó Zsanett: Algoritmizálás tanítása Flowgorithm-mel, In: Szlávi, Péter; Zsakó, László (szerk.) INFODIDACT 2017, Budapest, Magyarország: Webdidaktika Alapítvány, (2017) Paper: 18 , 12 p.
5. http://java.progtanulo.hu/java-programozas/1-elso-lepesek
6. http://nagygusztav.hu/java-programozas
7. http://java2.uw.hu/
8. https://ak-akademia.hu/java-programozas-kezdoknek/
9. http://www.java2s.com/