

# **Rácz András BHAX könyve**

---

**Egy egyetemre járó programozást tanuló  
hallgató könyve.**

Ed. BHAX, DEBRECEN,  
2019. május 9, v. 1.0.1

Copyright © 2019 Rácz András

Copyright (C) 2019, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

Copyright (C) 2019, András Rácz, raczandras0204@gmail.com,

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License". The owner of the license is Dr. Norbert Bátfai, whose book sample inspired my book.

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

<http://gnu.hu/fdl.html>

| <b>COLLABORATORS</b> |  |  |  |
|----------------------|--|--|--|
|----------------------|--|--|--|

|               |                         |                    |                  |
|---------------|-------------------------|--------------------|------------------|
|               | <i>TITLE :</i>          |                    |                  |
|               | Rácz András BHAX könyve |                    |                  |
| <i>ACTION</i> | <i>NAME</i>             | <i>DATE</i>        | <i>SIGNATURE</i> |
| WRITTEN BY    | Rácz, András            | 2019. november 24. |                  |

| <b>REVISION HISTORY</b> |  |  |  |
|-------------------------|--|--|--|
|-------------------------|--|--|--|

| NUMBER | DATE       | DESCRIPTION  | NAME       |
|--------|------------|--|------------|
| 0.0.1  | 2019-02-12 | Az iniciális dokumentum szerkezetének kialakítása.   | nbatfai    |
| 0.0.2  | 2019-02-14 | Inciális feladatlisták összeállítása.  | nbatfai    |
| 0.0.3  | 2019-02-16 | Feladatlisták folytatása. Feltöltés a BHAX csatorna<br><a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> repójába. | nbatfai    |
| 0.0.4  | 2019-02-19 | Aktualizálás, javítások.   | nbatfai    |
| 0.1.0  | 2019-02-22 | Saját fork létrehozása, alapvető beállítások, draft kivétele, Turing fejezet szemrevételezése.   | raczandras |
| 0.1.1  | 2019-02-23 | Turing csokor elkezdése, a források elkészítése.   | raczandras |
| 0.2.0  | 2019-03-01 | A Turing csokor teljesen elkészült.  | raczandras |
| 0.2.1  | 2019-03-02 | A Chomsky csokor feladatainak tanulmányozása, a feladatok megoldásának elkezdése.  | raczandras |

**REVISION HISTORY**

| NUMBER | DATE       | DESCRIPTION   | NAME       |
|--------|------------|---|------------|
| 0.3.0  | 2019-03-8  | A Chomsky csokor feladatai elkészültek.   | raczandras |
| 0.3.1  | 2019-03-09 | A Caesar csokor feladatainak tanulmányozása.  | raczandras |
| 0.4.0  | 2019-03-13 | A Caesar csokor feladatai elkészültek.  | raczandras |
| 0.4.1  | 2019-03-15 | Elkezdődik a Mandelbrot forradalom.   | raczandras |
| 0.5.0  | 2019-03-22 | A Mandelbrot forradalom sikерrel zárul, a feladatok elkészültek.  | raczandras |
| 0.5.1  | 2019-03-23 | A Welch csokor elkezdése.   | raczandras |
| 0.6.0  | 2019-03-31 | A Welch csokor feladatai elkészültek.   | raczandras |
| 0.6.1  | 2019-04-04 | A Conway csokor feladatainak a tanulmányozása, munka elkezdése.   | raczandras |
| 0.6.9  | 2019-04-09 | A Conway csokor labor feladatai elkészültek.  | raczandras |
| 0.7.0  | 2019-04-19 | A Conway csokor előadás feladata elkészült.   | raczandras |
| 0.8.0  | 2019-04-21 | A Schwarzenegger csokor első feladata kész, a másik két feladat passzolva az SMNIST kutatásra hivatkozva. | raczandras |
| 0.8.1  | 2019-04-22 | A Gutenberg csokor hozzáadása, az olvasónapló elkezdése.  | raczandras |
| 0.8.2  | 2019-04-23 | A Chaitin csokor tanulmányozása.  | raczandras |
| 0.8.3  | 2019-04-29 | Kész az olvasónapló.  | raczandras |

**REVISION HISTORY**

| NUMBER | DATE       | DESCRIPTION                                   | NAME       |
|--------|------------|---|------------|
| 0.9.0  | 2019-04-30 | A Chaitin csokor elkészül.                    | raczandras |
| 0.9.1  | 2019-05-04 | Nyelvtani hibák javítása, a könyv áttekintése | raczandras |
| 1.0.0  | 2019-05-04 | A könyv elkészült                             | raczandras |
| 1.0.1  | 2019-05-09 | Utólagos módosítások                          | raczandras |

# Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it.”

—Gregory Chaitin, *META MATH! The Quest for Omega*, [[METAMATH](#)]

# Tartalomjegyzék

|  |          |
|--|----------|
| <b>I. Bevezetés</b>  | <b>1</b> |
| 1. Vízió   | 2        |
| 1.1. Mi a programozás? . . . . .                             | 2        |
| 1.2. Milyen doksikat olvassak el? . . . . .                  | 2        |
| 1.3. Milyen filmeket nézzek meg? . . . . .                   | 2        |
| <b>II. Tematikus feladatok</b>                               | <b>3</b> |
| 2. Helló, Turing!  | 5        |
| 2.1. Végtelen ciklus . . . . .                               | 5        |
| 2.2. Lefagyott, nem fagyott, akkor most mi van? . . . . .    | 5        |
| 2.3. Változók értékének felcserélése . . . . .               | 6        |
| 2.4. Labdapattogás . . . . .                                 | 7        |
| 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . . .    | 7        |
| 2.6. Helló, Google! . . . . .                                | 7        |
| 2.7. 100 éves a Brun téTEL . . . . .                         | 8        |
| 2.8. A Monty Hall probléma . . . . .                         | 8        |
| 3. Helló, Chomsky!   | 10       |
| 3.1. Decimálisból unárisba átváltó Turing gép . . . . .      | 10       |
| 3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen . . . . . | 11       |
| 3.3. Hivatkozási nyelv . . . . .                             | 11       |
| 3.4. Saját lexikális elemző . . . . .                        | 11       |
| 3.5. l33t.l . . . . .  | 12       |
| 3.6. A források olvasása . . . . .                           | 12       |
| 3.7. Logikus . . . . .                                       | 14       |
| 3.8. Deklaráció . . . . .                                    | 14       |

---

|   |           |
|---|-----------|
| <b>4. Helló, Caesar!</b>  | <b>17</b> |
| 4.1. double** háromszögmátrix . . . . .                                   | 17        |
| 4.2. C EXOR titkosító . . . . .   | 19        |
| 4.3. Java EXOR titkosító . . . . .  | 19        |
| 4.4. C EXOR törő . . . . .  | 20        |
| 4.5. Neurális OR, AND és EXOR kapu . . . . .                              | 22        |
| 4.6. Hiba-visszaterjesztéses perceptron . . . . .                         | 22        |
| <b>5. Helló, Mandelbrot!</b>  | <b>24</b> |
| 5.1. A Mandelbrot halmaz . . . . .  | 24        |
| 5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal . . . . . | 25        |
| 5.3. Biomorfok . . . . .  | 26        |
| 5.4. A Mandelbrot halmaz CUDA megvalósítása . . . . .                     | 27        |
| 5.5. Mandelbrot nagyító és utazó C++ nyelven . . . . .                    | 28        |
| 5.6. Mandelbrot nagyító és utazó Java nyelven . . . . .                   | 28        |
| <b>6. Helló, Welch!</b>   | <b>30</b> |
| 6.1. Első osztályom . . . . .   | 30        |
| 6.2. LZW . . . . .  | 30        |
| 6.3. Fabejárás . . . . .  | 31        |
| 6.4. Tag a gyökér . . . . .   | 32        |
| 6.5. Mutató a gyökér . . . . .  | 32        |
| 6.6. Mozgató szemantika . . . . .   | 33        |
| <b>7. Helló, Conway!</b>  | <b>35</b> |
| 7.1. Hangyszimulációk . . . . .   | 35        |
| 7.2. Java életjáték . . . . .   | 37        |
| 7.3. Qt C++ életjáték . . . . .   | 38        |
| 7.4. BrainB Benchmark . . . . .   | 39        |
| <b>8. Helló, Schwarzenegger!</b>  | <b>40</b> |
| 8.1. Szoftmax Py MNIST . . . . .  | 40        |
| 8.2. Mély MNIST . . . . .   | 41        |
| 8.3. Minecraft-MALMÖ . . . . .  | 42        |

---

|  |           |
|--|-----------|
| <b>9. Helló, Chaitin!</b>  | <b>44</b> |
| 9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .   | 44        |
| 9.2. Gimp Scheme Script-fu: króm effekt . . . . .  | 45        |
| 9.3. Gimp Scheme Script-fu: név mandala . . . . .  | 46        |
| <b>10. Helló, Gutenberg!</b>   | <b>48</b> |
| 10.1. Juhász István - Magas szintű programozási nyelvek 1 olvasónaplója . . . . .  | 48        |
| 10.2. Kerninghan és Richie olvasónaplója . . . . .   | 49        |
| 10.3. Benedek Zoltán, Levendovszky Tihamér - Szoftverfejlesztés C++ nyelven olvasónaplója . . . . .  | 50        |
| <b>III. Második felvonás</b>   | <b>52</b> |
| <b>11. Helló, Berners-Lee!</b>   | <b>54</b> |
| 11.1. Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II és Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven olvasónapló . . . . . | 54        |
| 11.2. Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba . . . . .   | 59        |
| <b>12. Helló, Arroway!</b>   | <b>61</b> |
| 12.1. OO szemlélet . . . . .   | 61        |
| 12.2. Homokozó . . . . .   | 62        |
| 12.3. Gagyi . . . . .  | 63        |
| 12.4. Yoda . . . . .   | 65        |
| 12.5. Kódolás from scratch . . . . .   | 66        |
| <b>13. Helló, Liskov!</b>  | <b>69</b> |
| 13.1. Liskov helyettesítés sértése . . . . .   | 69        |
| 13.2. Szülő-gyerek . . . . .   | 71        |
| 13.3. Anti OO . . . . .  | 73        |
| 13.4. Ciklomatikus komplexitás . . . . .   | 76        |
| <b>14. Helló, Mandelbrot!</b>  | <b>79</b> |
| 14.1. Reverse engineering UML osztálydiagram . . . . .   | 79        |
| 14.2. Forward engineering UML osztálydiagram . . . . .   | 80        |
| 14.3. Egy esettan . . . . .  | 82        |
| 14.4. BPMN . . . . .   | 87        |

---

|   |            |
|---|------------|
| <b>15. Helló, Chomsky!</b>  | <b>89</b>  |
| 15.1. Encoding . . . . .  | 89         |
| 15.2. 1334d1c4 . . . . .  | 90         |
| 15.3. Full screen . . . . .   | 92         |
| 15.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció . . . . . | 94         |
| <b>16. Helló, Stroustrup!</b>   | <b>97</b>  |
| 16.1. JDK osztályok . . . . .   | 97         |
| 16.2. Másoló-mozgató szemantika + Összefoglaló . . . . .              | 99         |
| 16.3. Hibásan implementált RSA törése . . . . .                       | 102        |
| 16.4. Összefoglaló . . . . .  | 107        |
| <b>17. Helló, Gödel!</b>  | <b>108</b> |
| 17.1. C++11 Custom Allocator . . . . .                                | 108        |
| 17.2. STL map érték szerinti rendezése . . . . .                      | 110        |
| 17.3. Alternatív Tabella rendezése . . . . .                          | 113        |
| 17.4. GIMP Scheme hack . . . . .                                      | 116        |
| <b>18. Helló, !</b>   | <b>121</b> |
| 18.1. FUTURE tevékenység editor . . . . .                             | 121        |
| 18.2. OOCWC Boost ASIO hálózatkezelése . . . . .                      | 126        |
| 18.3. SamuCam . . . . .   | 127        |
| 18.4. BrainB . . . . .  | 130        |
| <b>19. Helló, Lauda!</b>  | <b>134</b> |
| 19.1. Port scan . . . . .   | 134        |
| 19.2. AOP . . . . .   | 136        |
| 19.3. Android Játék . . . . .   | 139        |
| 19.4. Junit teszt . . . . .   | 144        |
| <b>20. Helló, Calvin!</b>   | <b>147</b> |
| 20.1. MNIST . . . . .   | 147        |
| 20.2. Deep MNIST . . . . .  | 150        |
| 20.3. Android telefonra a TF objektum detektálója . . . . .           | 153        |
| 20.4. SMNIST for Machines . . . . .                                   | 156        |

---

|                            |            |
|----------------------------|------------|
| <b>IV. Irodalomjegyzék</b> | <b>160</b> |
| 20.5. Általános . . . . .  | 161        |
| 20.6. C . . . . .          | 161        |
| 20.7. C++ . . . . .        | 161        |
| 20.8. Lisp . . . . .       | 161        |

## Táblázatok jegyzéke

13.1. Összehasonlítás . . . . . 75

# Előszó

Amikor programozónak terveztem állni, ellenezték a környezetemben, mondván, hogy kell szövegszerkesztő meg táblázatkezelő, de az már van... nem lesz programozói munka.

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz alkalmi igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Minden esetre a programozás a teoretikus kultúra csúcsa. A GNU mozgalomban látom annak garanciáját, hogy ebben a szellemi kalandban a gyerekeim is részt vehessenek majd. Ezért programozunk.

## Hogyan forgasd

A könyv célja egy stabil programozási szemlélet kialakítása az olvasóban. Módszere, hogy hetekre bontva ad egy tematikus feladatcsokrot. minden feladathoz megadja a megoldás forráskódját és forrásokat feldolgozó videókat. Az olvasó feladata, hogy ezek tanulmányozása után maga adja meg a feladat megoldásának lényegi magyarázatát, avagy írja meg a könyvet.

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyereknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyereknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

## Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogy lássuk más is) példával.

## Hogyan nyomjuk?

Rántsd le a <https://gitlab.com/nbatfai/bhax> git repót, vagy méginkább forkolj belőle magadnak egy sajátot a GitLabon, ha már saját könyvön dolgozol!

Ha megvannak a könyv DocBook XML forrásai, akkor az alább látható **make** parancs ellenőrzi, hogy „jól formázottak” és „érvényesek-e” ezek az XML források, majd elkészíti a dblatex programmal a könyved pdf változatát, íme:

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml ←
    --noout
output.xml validates
rm -f output.xml
dblatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
=====
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dblatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

Ha minden igaz, akkor most éppen ezt a legenerált **bhax-textbook-fdl.pdf** fájlt olvasod.



#### A DocBook XML 5.1 új neked?

Ez esetben forgasd a <https://tdg.docbook.org/tdg/5.1/> könyvet, a végén találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

## I. rész

### Bevezetés

# 1. fejezet

## Vízió

### 1.1. Mi a programozás?

Számomra a programozás az önkifejezés egy formája.

### 1.2. Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [KERNIGHANRITCHIE]
- [BMECPP]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

### 1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.

## **II. rész**

### **Tematikus feladatok**

**Bátf41 Haxor Stream**

A feladatokkal kapcsolatos élő adásokat sugároz a <https://www.twitch.tv/nbatfai> csatorna, melynek permanens archívuma a <https://www.youtube.com/c/nbatfai> csatornán található.

## 2. fejezet

# Helló, Turing!

### 2.1. Végtelen ciklus

Írj olyan C végtelen ciklusokat, amelyek 0 illetve 100 százalékban dolgoztatnak egy magot és egy olyat, amely 100 százalékban minden magot!

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Turing/ciklus.c>

Egy magot 100%-on dolgoztatni nem egy nagy kihívás, hiszen ha egy szimpla while ciklust megírunk, az alapvetően így működik. Egy magot 0%-on dolgoztatni sem egy egentregető kihívás, viszont itt már kell minimálisan gondolkodni. De hamar rájövünk, hogy a sleep(x) parancs kiadásával x másodpercig nem használja a processzort a program. Kicsit érdékes, hogy ha nincs parancs a cikluson belül, vagyis nincs mit tenni, akkor 100%-on dolgozik a processzor. Ez azért történik, mert az operációs rendszer azt hiszi, hogy van elvégzendő feladat, ezért a programnak adja közel az összes processzoridőt. Viszont az összes magot 100%-on dolgoztatni már feladta a leckét. Először megpróbáltam a thread parancssal kezdeni valamit, de az túl bonyolultnak tűnt egy ilyen feladathoz. Majd Besenci Renátó adott egy tippet, miszerint az OpenMp-t kellene tanulmányoznunk a feladat megoldásához. Innen pedig már pár fórumon és StackOverflow lapon keresztül egyenes út vezetett a győzelemhez.

A programot roppant egyszerű használni. Ha egy magot szeretnénk 100%-ban dolgoztatni, akkor semmit nem kell módosítani, szimplán csak le kell fordítani és futtatni.

Ha egy magot szeretnénk 100%-ban dolgoztatni, akkor vegyük ki a //t a

```
//sleep(1)
```

függvényhívásból.

Ha pedig az összes magot szeretnénk 100%-ban dolgoztatni, akkor ugyanúgy a //t kell kitörölni a következő helyről:

```
#pragma omp parallel while
```

### 2.2. Lefagyott, nem fagyott, akkor most mi van?

Mutasd meg, hogy nem lehet olyan programot írni, amely bármely más programról eldönti, hogy le fog-e fagyni vagy sem!

Megoldás videó:

Nem tudunk olyan programot írni, ami minden más programról eldönti, hogy van e benne végtelen ciklus. Mivel, ha tudnánk, akkor már valószínűleg lett volna olyan ember, aki ezt a programot megírja.

De tegyük fel, hogy megírjuk ezt a programot, aminek a neve legyen eldöntő. Annak a programnak a neve, amelyről el kell döntenи, hogy van e benne végtelen ciklus, legyen eldöntendő. Nyílván az eldöntő bemeneti argumentuma lesz az eldöntendő. Ahhoz, hogy eldöntő megállapítsa, hogy van e eldöntendőben végtelen ciklus, futtatnia kell az eldöntendő kérdéses részleteit. Ekkor ha az eldöntendő programban nincs végtelen ciklus, eldöntő hamissal tér vissza, ami azt jelenti, hogy nincs eldöntendőben végtelen ciklus.

Azonban ha az eldöntendő programban tényleg van egy végtelen ciklus, és azt eldöntő futtatja, hogy megbizonyosodjon róla, akkor eldöntő maga is egy végtelen ciklussá válik. Éppen ezért eldöntő sose fog igazzal visszatérni, mert minden ilyen esetben ő is le fog fagyni.

## 2.3. Változók értékének felcserélése

Írj olyan C programot, amely felcseréli két változó értékét, bármiféle logikai utasítás vagy kifejezés násználata nélkül!

Megoldás videó: [https://bhaxor.blog.hu/2018/08/28/10\\_begin\\_goto\\_20\\_avagy\\_elindulunk](https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk)

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Turing/csere.c>

Ez egy egyszerű matematikai/logikai feladat, amit ha egyszer megmutatnak az embernek, akkor örökké emlékezni fog rá. Olyan mint a biciklizés, nem lehet elfelejteni.

Még úgy is, hogy logikai utasítások, kifejezések nélkül kell megoldani ezt a problémát, rengeteg lehetőség közöl választhatunk. Én itt most kettőt fogok bemutatni.

Az egyik az, hogy összeggel és különbséggel cseréljük fel a két változót a következőképpen:

```
a = a+b;  
b = a-b;  
a = a-b;
```

Ha ezt végigvezetjük például az  $a=5$  és  $b=6$  értékekkel akkor az első lépés után  $a=11$  és  $b=6$ . A második lépés után  $a=11$  és  $b=6$  a harmadik lépés után pedig  $a=6$  és  $b=5$

Egy másik lehetőség pedig az, hogy szorzattal cseréljük meg a két változó értékét aminek az alapja hasonló az előző megoldáshoz egy kis módosítással:

```
a = a*b;  
b = a/b;  
a = a/b;
```

Ezeken kívül még vannak módszerek amik megfelelnek a feladat leírásának. Ezek a forrásban megtalálhatóak és a működésük alapja ugyan az mint az előző két megoldásnak.

## 2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írj egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videókon.)

Megoldás videó: <https://bhaxor.blog.hu/2018/08/28/labdapattogas>

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Turing/labda.cpp>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Egy egyszerű "grafikus" program, ami egy labdának álcázott o betűt mozgat a képernyőn egy egyszerű while ciklus segítségével. Habár maga a program nem hosszú, és nem is túl bonyolult, mégis elég nagy hatással van a kezdő mezei programozóra, hiszen nagyon sok programozást tanulónak (köztük nekem is) az egyik álma egy valódi grafikus felülettel működő program írása, és ez egy nagyon jó kezdet eme cél megvalósításához.

Maga a program két fő részből áll. Az egyik egy függvény, ami a labdát rajzolja ki a konzolra, A másik pedig maga a main.

A main-ben először létrehozunk egy maxX és egy maxY változót, amiket át is adunk a tx és a ty tömbök méretének.

Ezután két for ciklus végigmegy a két tömbön, a második, és az utolsó elemek értéke -1 lesz, a többi elemé pedig 1

végül pedig egy while ciklus és a függvény segítségével kiírja a konzolra a labdát.

## 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Turing/bogo2.cpp>

Alapvetően a BogoMips a processzorunk sebességének meghatározásához használatos mértékegység. Azt mondja meg, hogy a számítógép processzora mekkora szóhosszal dolgozik

Ezt a XOR ^ művelet segítségével számolja ki a program, ami a kizáró vagy művelete. Az int értékének 1-et adunk, és addig shifteljük balra, ameddig lehet, vagyis amíg az int értéke 0 nem lesz.

Közben egy másik változóval számoljuk, hogy hányszor shiftelt balra az int, ezzel meghatározva a szóhosszt. Az én esetben az eredmény 32 lett, ami azt jelenti, hogy az én processzorom szóhossza 32 bit, azaz 4 bájt.

## 2.6. Helló, Google!

Írj olyan C programot, amely egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Turing/pagerank.c>

Én a Bevezetés a Programozásba nevű tárgyon már átnézett [PageRank](#) programot vettetem alapnak, és azt alakítottam át c++-ból c-re. Ez azonban a vártnál több gondot okozott. De legalább mostmár tudom, hogy amennyiben c++ kódot akarok c-re átírni akkor nem érdemes az `abs()` függvényt használni, mert nem ugyanúgy működik a két nyelven belül ez a függvény. Ez fel is kellette, az érdeklődésemet, hogy miért nem? Kis utánajárás utána a [GeeksforGeeks](#) oldalon meg is találtam a választ, ami szerint C++ nyelven ennek a függvénynek a visszatérési típusa ugyan az, mint a bemeneti típus. Éppen ezért nyugodtan számolhattuk vele a double típusú távolságot. Ezzel szemben C nyelven a visszatérési típus minden esetben int lesz. Éppen ezért lett a végeredmény mind a négy lap esetén 0.25

Ezt a proglémát én egy egyszerű if-else szerkezzettel oldottam meg. Az eredeti c++ verzióban kiszámolta a függvény a távolságot, és annak az abszolút értékét adta vissza.

Ezzel szemben az én megoldásom megvizsgálja, hogy a távolság negatív-e. Ha nem, akkor szimplán visszadaja az értéket, ha viszont negatív, akkor az eredményt megszorozza -1 el ezáltal pozitív eredményt kapva. És ezt a pozitív értéket adja vissza a függvény.

## 2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

Megoldás videó: <https://youtu.be/xbYhp9G6VqQ>

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/Primek\\_R](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R)

A megoldás forrása Bátfa Norbert tulajdona.

Mint tudjuk, léteznek a prímszámok. Ezek olyan számok, amik csak 1-el és önmagukkal oszthatóak. Valamint léteznek az ikerprímek. Ezek pedig olyan prímszámpárok, amiknek a különbsége pontosan 2. Ha minden ikerprím reciprokának az összegének vesszük a sorozatát, akkor ez a sorozat egy számhoz konvergál. Ez a szám a Brun-konstans. Nem tudjuk azt, hogy az ikerprímek száma véges vagy végtelen e, de ez nem okoz gondot, hiszen elvileg ha végtelen se lépi túl az összegük a Brun-konstanst. Na most be kell vallanom, hogy számtalan olyan ember létezik a földön, aki nálam jobban ért a matematikához. Viszont nemek erről egy elég érdekes dolog jutott eszembe, ami nem más, mint Zeno paradoxona. E szerint x útat teszünk meg, hogy elérjük a célunkat. Ezek alapján megteszünk  $1/2x$  utat +  $1/4x$  utat +  $1/8x$  utat +  $1/16x$  utat... Ha ezekből képzünk egy sorozatot, az a sorozat 1-hez fog konvergálni. Éppen ezért soha nem érünk el oda, ahova megyünk. Maga a téTEL matematikailag helyes, azonban a való életben tudjuk, hogy ez nem így működik.

## 2.8. A Monty Hall probléMA

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/03/erdos\\_pal\\_mit\\_keresett\\_a\\_nagykonyvben\\_a\\_monty\\_hall-paradoxon\\_kapcsan](https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/MontyHall\\_R](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R)

A megoldás forrása Bátfai Norbert tulajdona.

A Monty Hall problémát még középiskolában ismertem meg, sok különböző változata van. Az általam ismert történetben Monty Hall egy műsorvezető volt, akinél a nyertes játékosok választhattak három darab ajtó közül. A háromból két ajtó mögött 1-1 darab kecske, míg a harmadik ajtó mögött egy sportautó volt. A játékos választott egy ajtót, majd Monty Hall, aki tudta, hogy melyik ajtó mögött van az autó, kinyitott egy másik ajtót, ami mögött egy kecske lapult. Ezek után a játékosnak lehetősége volt változtatni a döntésén, vagy maradhatott az eredetileg kiválasztott ajtónál. A kérdés az, hogy mely esetben van több esélye megnyerni az autót? A legtöbb ember azt mondaná, hogy 50-50% esélye van megnyerni az autót, hiszen vagy az egyik ajtó mögött van az autó, vagy a másik mögött. Ekkor persze hiába magyarázzuk, hogy 1/3 esélye van megnyerni az autót, ha nem vált, és 2/3 ha vált, a legtöbb embert elég nehéz meggyőzni erről. Ekkor kell kicsit átalakítani a kérdést. Ha van 1 millió ajtó, ebből kiválaszt a játékos 1-et, majd kinyitnak 999,998 ajtót, amik mögött kecske van, akkor melyik esetben van több esélye a játékosnak megnyerni az autót? Ilyenkor már a legtöbb ember eggyértelműnek tartja, hogy vált, de van olyan ismerősöm, aki még ekkor is azt mondta, hogy 50-50% esélye van megnyerni az autót, ha vált ha nem. Ez a program ennek a játéknak a nyerési eseteit szimulálja. Tízmillió esetből hányszor nyer az, aki minden vált, és az aki egyáltalán nem vált.

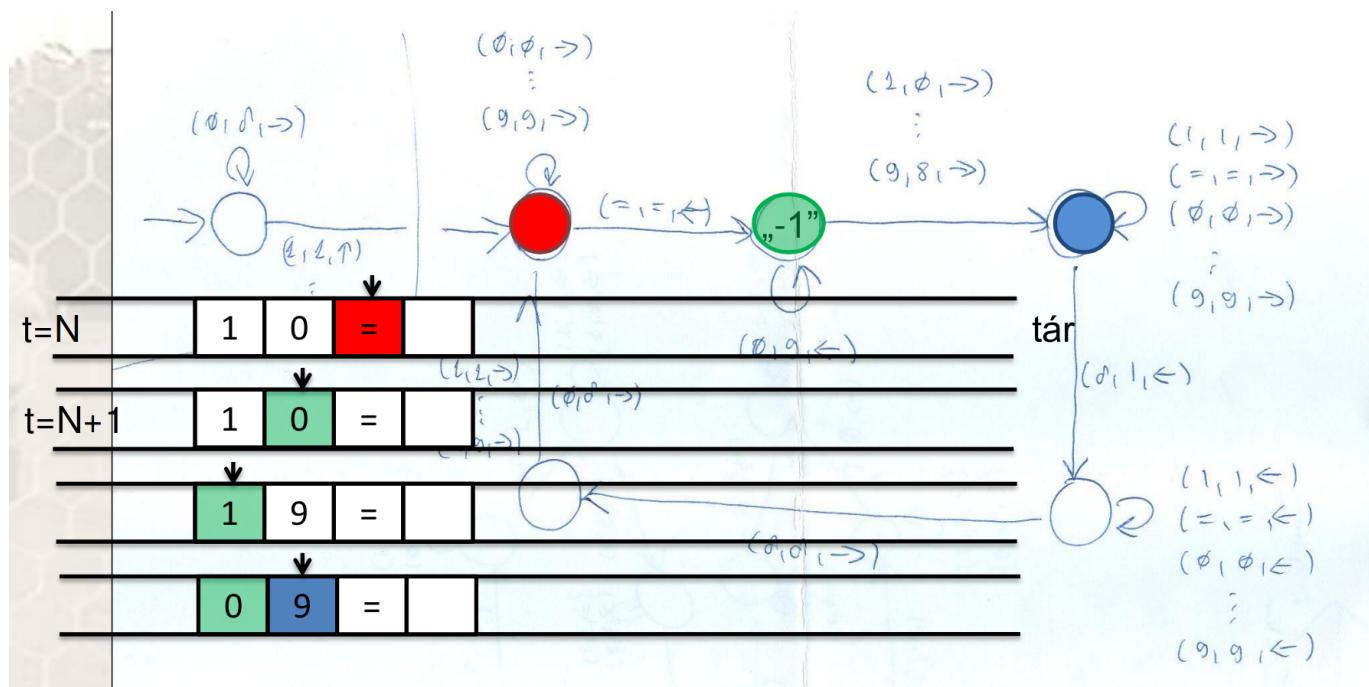
### 3. fejezet

## Helló, Chomsky!

### 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:



A kép nem az én tulajdonom, hanem a Magas szintű programozási nyelvek 1 nevű tárgyon kivetített előadás fóliáiról másoltam.

Maga az unáris számrendszer csupa 1-esekből vagy vonalakból áll. Ilyen például, amikor a kezünkön számolunk, vagy amikor az óvodákban a gyerekek a pálcikákkal rakják ki a dolgokat. Pontosan annyi 1-es vagy pálcika, vagy akármilyen jel kell, amennyi maga a szám értéke. Például ha az 50-es számot szeretnénk felírni unárisban, akkor 50 darab 1-est kellene leírni egymás után. Éppen ezért, ebben a számrendszerben csak a természetes számokat tudjuk ábrázolni.

Egy ilyen decimálisból unárisba átváltó Turing gépet mutat a fenti ábra is. Ez a gép a kapott szám utolsó számjegyéből von le egyeseket. Ha a számjegy 0 akkor 9db-ot von le, ha 5 akkor 4-et. Ezzel együtt a levont

egyeseket a tárba helyezi. Ezt minden számjeggyel megismétli.

## 3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

Mutass be legalább két környezetfüggő generatív grammatikát, amely ezt a nyelvet generálja!

Megoldás videó:

Megoldás forrása:

Az  $a^n b^n c^n$  nyelv nem környezetfüggetlen. Na de először értsük meg, hogy mit is jelent ez a nyelv.

Az  $a^n b^n c^n$  tulajdonképpen annyit jelent, hogy n darab a, majd n darab b, majd végül n darab c áll egymás után. Ezek a terminális szimbólumok. A szabály alapján a környezetfüggő nyelvenknél bal oldalt csak egy önmagában álló nem-terminális szimbólum állhat. Azonban nem létezik olyan képzési szabály ami alapján ez a szabály teljesíthető, éppen ezért ez a nyelv nem környezetfüggetlen.

## 3.3. Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiál BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Chomsky/nyelv.c>

A BNF (Backus-Naur-forma) használatával környezetfüggetlen nyelveket lehet leírni. Nagyon sok programozási nyelvek szintaxisai is BNF-ben vannak leirva.

A programozási nyelveknek is van nyelvtana, illetve nyelvtani szabályaik. Az egyik ilyen szabály C89-ben az, hogy a for ciklus fejrészében nem lehetett változót deklarálni, éppen ezért ha a a következőképpen szeretnénk lefordítani a fenti programot:

```
gcc -o nyelv nyelv.c -std=c89
```

Akkor a következő hibaüzenetet kapjuk:

```
nyelv.c:3:2: error: ‘for’ loop initial declarations are only allowed in C99 or C11 mode
```

Ez pontosan leírja nekünk, hogy a for ciklusban deklarálni csak c99 vagy c11 módban lehet.

Éppen ezért ha -std=c89 helyett nem írunk semmit, vagy -std=c99-et írunk, akkor a program gond nélkül lefordul.

## 3.4. Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetén megjelenő valós számokat! Nem elfogadható olyan megoldás, amely maga olvassa betűnként a bemenetet, a feladat lényege, hogy lexert használunk, azaz óriások vállán állunk és ne kispályázzunk!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Chomsky/lex.l>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

A lexer-rel szövegelemző programokat lehet generálni az általunk megadott szabályok alapján. A program különböző részeit % jelekkel kell elválasztani egymástól. Itt a numbers változóban fogjuk számolni a valós számok darabszámát. Majd megmondjuk, hogy a digit egy 0 és 9 között lévő számot jelöl. Ezek után jön az a kód részlet, ami megmondja a lexernek, hogy a valós számokat számolja meg. Végül pedig kiíratjuk a valós számok darabszámát. A futtatáshoz először is telepítenünk kell a lex-et majd a forrásban található programot kell megírni.

Majd azt a következőképp kell lefordítanunk:

```
lex -o lex.c lex.l
```

```
gcc -o lex lex.c -lfl
```

Ezzel magkapjuk a <https://github.com/raczandras/progbook/blob/master/src/Chomsky/lex.c> oldalon található programot, ami a feladat megoldása.

## 3.5. I33t.l

Lexelj össze egy l33t cipher-t!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Chomsky/leet.l>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

A leet vagy a saját formájában leírva 1337 egy, az internettel együtt elterjedt szleng nyelv, amiben a betűket különböző számokként, és egyéb ASCII karakterenként, a számokat pedig különböző betűkként ábrázoljuk.

Itt is érvényes az a szabály, hogy a program egyes részeit % jelekkel kell elválasztani egymástól. Itt a legelső részben a cipher struktúrában meg vannak adva a karakterek leet formái

A második részben történik az érdemi munka, először a szöveget kisbetűssé alakítja a program, majd pedig végigmegy a szövegen, és minden karaktert a neki megfelelő leet formájú karakterré alakítja át.

A harmadik és egyben utolsó részben található a main amiben a lex meghívása történik.

## 3.6. A források olvasása

Hogyan olvasod, hogyan értelmezed természetes nyelven az alábbi kódcsipeteket? Például

```
if(signal(SIGINT, jelkezelő)==SIG_IGN)
    signal(SIGINT, SIG_IGN);
```

Ha a SIGINT jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a jelkezelő függvény kezelje. (Miután a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)

**Bugok**

Vigyázz, sok csipet kerülendő, mert bugokat visz a kódba! Melyek ezek és miért? Ha nem megy ránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

```
if(signal(SIGINT, SIG_IGN) !=SIG_IGN)
    signal(SIGINT, jelkezelo);
```

ii.

```
for(i=0; i<5; ++i)
```

iii.

```
for(i=0; i<5; i++)
```

iv.

```
for(i=0; i<5; tomb[i] = i++)
```

v.

```
for(i=0; i<n && (*d++ = *s++) ; ++i)
```

vi.

```
printf("%d %d", f(a, ++a), f(++a, a));
```

vii.

```
printf("%d %d", f(a), a);
```

viii.

```
printf("%d %d", f(&a), a);
```

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Chomsky/signal.c>

Megoldás videó:

i: Ha kapunk egy INTERACT szignált, akkor a jelkezelő fügvénnyel eldöntjük, hogy mihez kezdjünk azzal a szignállal, mit reagáljon rá a program.

ii: Egy forciklus ami nullától négyig megy, és a ciklus törzsében lévő művelet elvégzése előtt nő az értéke eggyel.

iii: Szintén egy forciklus, ami szintén nullától négyig megy, viszont itt már a ciklus törzsében lévő műveletek elvégzése után növekszik az értéke.

iv: Egy for ciklus, ami berakja a tomb[i]-edik helyére az i értékénél eggyel nagyobb értéket, és közben i értékét is növeli.

v: Egy for ciklus, ami addig megy, amíg i kisebb mint n, illetve amíg a d és s pointerek értékei megyegyezők.

vi: Kiirunk két, az f nevű függvény által generált számot. az egyik szám az a majd a eggyel megnövelt értékének a feldolgozásából jön létre, mig a másik szám a+1 és a feldolgozásából. Fontos a sorrend.

vii: Szintén két számot irunk ki, az egyik szám az f nevű függvény által feldolgozott a nevű számból előállt érték, a másik pedig a értéke.

## 3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$ (\forall x \exists y ((x < y) \wedge (y \text{ prim})) ) $  
$ (\forall x \exists y ((x < y) \wedge (y \text{ prim})) \wedge (S y \text{ prim})) \leftrightarrow  
$ (\exists y \forall x (x \text{ prim}) \supset (x < y)) $  
$ (\exists y \forall x (y < x) \supset \neg (x \text{ prim})) $
```

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/MatLog\\_LaTeX](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX)

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, [https://youtu.be/AJSXOQFF\\_wk](https://youtu.be/AJSXOQFF_wk)

Első értelmezése: minden számra igaz, hogy létezik tőle nagyobb y prímszám.

Második értelmezése: minden számra igaz, hogy létezik egy olyan tőle nagyobb y prímszám, hogy  $y+2$  is prím.

Harmadik értelmezése: létezik olyan szám, amitől minden prímszám kisebb.

Negyedik értelmezése: létezik olyan szám, amitől egyik kisebb szám se prím.

## 3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciajára
- egészek tömbje
- egészek tömbjének referenciajára (nem az első elemé)
- egészre mutató mutatók tömbje
- egészre mutató mutatót visszaadó függvény
- egészre mutató mutatót visszaadó függvényre mutató mutató
- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény
- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

Mit vezetnek be a programba a következő nevek?

- `int a; //létrehoz egy egész típusú változót`
- `int *b = &a; //egy pointer, ami a memóriacímére hivatkozik`
- `int &r = a; //egy referencia a-ra`
- `int c[5]; //5 elemű tömb aminek c a neve`
- `int (&tr)[5] = c; //egy tr nevű referencia c-re`
- `int *d[5]; //egy 5 elemű pointerekből álló tömb`
- `int *h(); //Egy egészre mutató mutatót visszaadó függvény`
- `int *(*l)(); //Egy egészre mutató mutatóra mutató mutatót visszaadó ↵ függvény`
- `int (*v(int c))(int a, int b) //Függvénymutató, ami egy egészet ↵ visszaadó függvényre mutató mutatóval visszatérő függvény`
- `int (*(*z)(int))(int, int); //Függvénymutató, ami egy egészet visszaadó ↵ függvényre mutató mutatót visszaadó függvényre mutat`

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Chomsky/dekla.cpp>

Egész:

```
int a;
```

Egészre mutató mutató:

```
int *b = &a;
```

Egész referenciajá:

```
int &r = a;
```

Itt fontos megjegyezni, hogy c-ben nincs referencia, ezért ezt a kódcsipetet érdemes g++-al fordítani gcc helyett.

Egészek tömbje:

```
int c[5];
```

Egészek tömbjének referenciajá (nem az első elemé):

```
int (&tr) [5] = c;
```

Egészre mutató mutatók tömbje:

```
int *d[5];
```

Egészre mutató mutatót visszaadó függvény:

```
int *h();
```

Egészre mutató mutatót visszaadó függvényre mutató mutató:

```
int *(*l)();
```

Egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény:

```
int (*v (int c)) (int a, int b)
```

Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre:

```
int (*(*z) (int)) (int, int);
```

## 4. fejezet

# Helló, Caesar!

### 4.1. double\*\* háromszögmátrix

Írj egy olyan `malloc` és `free` párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

Tutoráltam: Duszka Ákos Attila

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Caesar/tm.c>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Először is egy alap fogalom. Az alsó háromszög mátrixnak ugyanannyi sora van, mint oszlopa. Ezen kívül még egy nagyon fontos tényezője az is, hogy a főátlója felett csak 0 szerepel.

Általában az ilyen mátrixokat, ha tömbökben tároljuk, akkor nincs értelme a nullákat is tárolni a többi, számunkra érdekes elemmel együtt, éppen ezért ezeket nem is tároljuk. Amikor egy ilyen tömböt vissza szeretnénk alakítani az eredeti alakjára, akkor sorfolytonosan írjuk fel az elemeit. ez minden össze annyit jelent, hogy a mátrix első sorába az első elemet írjuk fel, a második sorába a 2. és 3. elemet, és így tovább minden sorban egyelőre több elemet írunk fel mint az előző sorban.

Ebben a programban egy ilyen alsó háromszög mátrixot hozunk létre egy

```
double **
```

segítségével. Ez egy pointerre mutató pointer, ami tökéletes a többdimenziós tömbök használatához.

Ezek után a következő kis programrészlet:

```
if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
{
    return -1;
}

printf ("%p\n", tm);

for (int i = 0; i < nr; ++i)
{
```

```

if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL)
{
    return -1;
}

}

```

Ellenőrzi, hogy történt e valamilyen memóriahiba, (pl. nincs e tele a memória) és ha történt, akkor -1-el tér vissza.

Ellenkező esetben a program a

```
tm[i][j] = i * (i + 1) / 2 + j;
```

képletet használva feltölti, a tömböt. Ezután két egymásba ágyazott for ciklus segítségével kiírja azt.

Ezek után módosítunk a tömb egyes elemein, majd megint kiírjuk őket.

Legvégül pedig a

```

for (int i = 0; i < nr; ++i)
    free (tm[i]);

free (tm);

```

függvény használatával felszabadítjuk a tömbnek lefoglalt helyet.

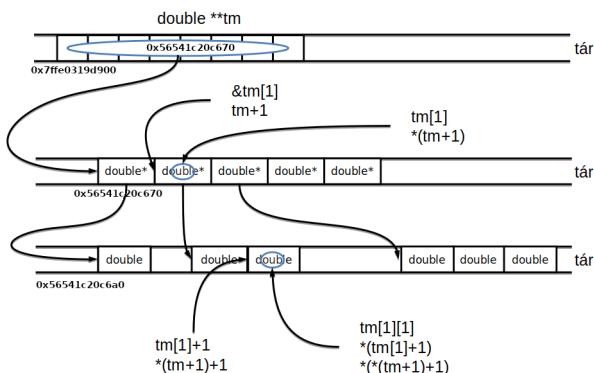
A program futtatásnál a következő memóriacímeket írta ki:

```

./tm
0x7ffe0319d900
0x56541c20c670
0x56541c20c6a0

```

Aminek a jelentése:



A képnek az alapját Bátfai Norbert Biztosította, én azt módosítottam.

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Caesar/exor.c>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Ez a fajta titkosítás a kizárt vagy műveleten alapul. A megadott kulcs, és a forrásfájl karaktereit kizártó vaggyal titkosítva egy szöveget úgy tudunk titkosítani, hogy egy olvashatatlan karaktermasszát kapunk végeredményül. Viszont aki ismeri a kulcsot az ugyan olyan egyszerűen vissza tudja alakítani a szöveget az eredeti alakjára úgy, hogy mégegyszer lefuttatja a programot, de a titkosított forrást adja meg titkosítandóként, ezzel visszakapva az eredeti szöveget. Így más nem tudja elolvasni a titkainkat, csak az, aki ismeri hozzá a kulcsot. (legalábbis egyenlőre. Két feladattal később már más lesz a helyzet.)

Először is a

```
#define MAX_KULCS 100
#define BUFFER_MERET 256
```

használatával megadjuk a maximális kulcs és buffer méretet. A main osztály első argumentuma a kulcs lesz, mig a második az maga a szöveg, amit titkosítani szeretnénk.

A következő ciklusok használatával:

```
while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
{
    for (int i = 0; i < olvasott_bajtok; ++i)
    {
        buffer[i] = buffer[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
    write (1, buffer, olvasott_bajtok);
}
```

program végigmegy a bemeneti adatok (titkosítandó fájl) karakterein, és mindegyiket titkosítja a kulcs használatával, és kiírja a végeredményt.

A program használata: ./exor kulcs <titkosítandó fájl> titkosított fájl

Erre egy példa: **./exor 12345678 <lista> titkoslista**

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Caesar/exort.java>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Itt az előző feladatban megírt EXOR titkosítót írjuk át java programozási nyelvre. Ehhez importálnunk kell az input/output streamet, ez ahhoz kell, hogy olvasni tudjuk a bemeneti fájlt, illetve, hogy írni tudjuk a kimeneti fájlt.

A main-ben megpróbáljuk a try-al beolvasni az args (argumentumok) tömbbe azt a fájlt, amit titkosítani szeretnénk, és ha ez nem sikerült, akkor "elkapjuk" a hibát a catch szerkezettel, és kiíratjuk, hogy mi a hiba:

```
[public static void main(String[] args) {
    try {
        new ExorTitkosító(args[0], System.in, System.out);
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
```

Ha viszont sikerült beolvasni a fájlt, akkor az ExorTitkosító nevű függvényt meghívva előállítjuk a titkosított szöveget. a System.in illetve System.out a bemenő és a kimenő fájlra utalnak.

Először is a függvény átadja a program a kulcs nevű tömbnek a bemenő szöveget, és létrehoz egy buffer nevű tömböt is 256-os mérettel. Erre az EXOR művelethez lesz szükség.

Végül a program egy while-ba épített for ciklus segítségével végigmegy a szövegen, és minden egyes karakternek meghatározza a titkosított verzióját, és kiírja azt a kimeneti fájlba.

## 4.4. C EXOR törő

Tutoráltam: [George Butcovan](#)

Írj egy olyan C programot, amely megtöri az első feladatban előállított titkos szövegeket!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Caesar/tores.c>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Önnel is előfordult már az, hogy elfelejtette egy EXOR-ral titkosított fájl kulcsát? Ön is akart már kutatnod mások fájljai között, de nem tudott, mert EXOR-ra voltak titkosítva a fájlok?

Ne szenvédjen tovább. Az EXOR törő biztos megoldást nyújt önnek! Csupán annyit kell tudnia, hogy hány karakterből áll a kulcs, és már is használhatja ezt a fenomenálist programot. A felhasználó ostobaságaiért és azok jogi következményeiért felelősséget nem vállalunk.

A működése roppant egyszerű. Mivel nem ismerjük a kulcsot, ezért a program az összes lehetséges kombinációt végigpróbálja. A következőkben bemutatott példában a kulcs 8 darab karakterből áll.

Legelőször a program a következő while ciklus:

```
while ((olvasott_bajtok =
        read (0, (void *) p,
              (p - titkos + OLVASAS_BUFFER <
               MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - ←
              p)))
p += olvasott_bajtok;
```

Használatával beolvassa a feltörni kívánt fájlt, majd a maradék helyet a bufferben egy for ciklust használva

```
for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
titkos[p - titkos + i] = '\0';
```

feltölti 0 értékkal.

Ezek után egy halom (ami jelen esetben 8) for ciklussal:

```
#pragma omp parallel for private(kulcs)
for (int ii = '0'; ii <= '9'; ++ii)
    for (int ji = '0'; ji <= '9'; ++ji)
        for (int ki = '0'; ki <= '9'; ++ki)
            for (int li = '0'; li <= '9'; ++li)
                for (int mi = '0'; mi <= '9'; ++mi)
                    for (int ni = '0'; ni <= '9'; ++ni)
                        for (int oi = '0'; oi <= '9'; ++oi)
                            for (int pi = '0'; pi <= '9'; ++pi)
                            {
                                kulcs[0] = ii;
                                kulcs[1] = ji;
                                kulcs[2] = ki;
                                kulcs[3] = li;
                                kulcs[4] = mi;
                                kulcs[5] = ni;
                                kulcs[6] = oi;
                                kulcs[7] = pi;

                                exor_tores (kulcs, KULCS_MERET, titkos, ←
                               p - titkos);
                            }
}
```

Megpróbálja a program előállítani az eredeti szöveget. Azonban több kombináció is ad eredményt, éppen ezért nekünk kell kitalálni, hogy a kapott eredmények közül melyik a helyes. Kis érdekesség, hogy ezek a for ciklusok az összes magot dolgoztatni fogják, ezzel jelentősen leccsökentve a töréshez szükséges időt.

Ha a kulcs nem 8 karakterből áll, akkor se essünk pánikba! Csupán néhány (Pontosan 3) szekcióban kell módosítani a program kódját. Ezek a következők:

Először is a program fejében a

```
[#define KULCS_MERET 8]
```

sorban a 8-at át kell írni arra a számra, amennyi karakterből áll a kulcs.

Majd a 70. és 71. sorokban lévő

```
[ printf("Kulcs: [%c%c%c%c%c%c%c] \nTiszta szoveg: [%s]\n",
kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],kulcs[6],kulcs[7], ←
buffer);
```

utasításokban annyi **%c** és **kulcs[n]** legyen, amennyi karakterből áll a kulcs.

Végül pedig az előzőekben már látott for ciklus halmon kell módosítanunk úgy, hogy pontosan annyi **for** ciklus, és pontosan annyi **kulcs[n] = xi;** legyen a programban, amennyi karakterből áll a kulcs.

Most hogy ezt mind tudjuk, a programot a következőképpen kell fordítani: **gcc tores.c -fopenmp -o tores -std=c99**

És futtatni: **./tores <titkosfájl**

## 4.5. Neurális OR, AND és EXOR kapu

R

Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/NN\\_R](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R)

A megoldás forrása Bátfai Norbert tulajdona.

A neurális hálózatokban például a machine learning esetében, A neuronok egy gráfban elhelyezkedve egymással kommunikálnak úgynevezett "Activation function", magyarul Aktivációs függvény segítségével.

Léteznek bemeneti, kimeneti, és rejtett neuronok is.

A bemeneti neuronok kapják meg a bemenetet. Itt több különböző fajta neuront is meg lehet különböztetni. Vannak egybemenetű és több bemenetű neuronok is. Ezeknek a neuronoknak nincs különösebb feldolgozó feladatuk, továbbítják a bemenetet a többi neuronnak.

A kimeneti neuronok amik a környezetnek adják tovább a kapott információt.

A rejtett neuronoknak pedig a bemenete és a kimenete is csakis más neuronokhoz kapcsolódik.

Ezek alapján egy neurális hálónak legalább két rétegből kell állnia. Egy bemenetiből, és egy kimenetiből. Felső határ, azaz hogy a bemeneti és a kimeneti neuronok között hány darab további réteg helyezkedik el, elviekben nincs.

Először minden neuron megkapja a saját bemeneteit, és minden neuron ebből a bemenetből előállít egy úgynevezett súlyozott összeget, és ezt az értéket vezeti végig az aktivációs függvényen. Egy példa lehet az, hogy ha a súlyozott összeg pozitív lesz, akkor az érték 1, míg ha a súlyozott összeg negatív, akkor az érték -1 lesz.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/nahshon/blob/master/ql.hpp#L64>

A megoldás forrása Bátfai Norbert tulajdona.

Perceptronról a Mesterséges Intelligenciák, és a neurális hálók témakörében lehet szó. Ellenőrzi a bemenetet, és egy feltétel alapján eldönti, hogy mi legyen a kimenet. Egy példa:

Van három bemeneti adatunk amikhez pozitív egész számokat várunk. Ha a három bemeneti számból kettő kisebb mint nulla, akkor a kimeneti adat -1 lesz, ha viszont a háromból legalább kettő pozitív szám, akkor a számok összege lesz a kimeneti adat.

Ekkor kimondhatjuk, hogy 1 a hibahatár, mert ekkor még megkapjuk az általunk kért dolgot, viszont ha már kettőt hibázunk akkor már -1 lesz a válasz.

Ezt a hibahatárt szokták finomhangolni. Nagyon magas hibahatárnál kezdenek, és egyre kisebbé teszik egészen addig amíg elfogadható a hibák mennyisége.

Persze a mi három bemeneti adatos példánknál nem sokat lehet finomhangolni, de ha több millió bemeneti adatról beszélünk, ott ez egy elég fontos dolog.

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Tutorálóm: [Duszka Ákos Attila](#)

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Mandelbrot/mandelbrot.cpp>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Mielőtt bármihöz hozzákezdenénk egy nagyon fontos információ. Ahhoz, hogy leforduljon a programunk, szükséges a png++. Ezt a legegyszerűbben a **sudo apt install png++** parancssal lehet megtenni. Most hogy ezt letudtuk, jöhet pár alapvető információ.

A Mandelbrot halmaz lényege (legalábbis számomra) az, hogy komplex számokkal, és egy egyenlettel dolgozik. Azok a számok amelyek kielégítik ezt az egyenletet egy nagyon szép képet alkotnak, ha levetítjük őket egy kétdimenziós sikra. Akit ez bővebben vagy részletesebben érdekel azoknak ajánlom a különböző weboldalakat, én nem fogom tovább boncolgatni, mert én magam sem értem.

A program legelején includeoljuk a png++-t, hiszen nagyrészt ezt fogja használni a program.

```
#include <png++-0.2.9/png.hpp>
```

Ezek után létrehozunk végeleges értékeket N-nek és M-nek, valamint megadjuk X és Y lehetséges minimum és maximum értékét is.

```
#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35
```

két egymásba ágyazott for ciklus használatával megadjuk a C, a Z, és a Zuj nevű Komplex számok valós és imaginárius értékét. Ezek korábban lettek létrehozva a mainen belül, és a Komplex nevű struktúrához tartoznak.

```
struct Komplex
{
    double re, im;
};

struct Komplex C, Z, Zuj;
```

Végül pedig a `GeneratePNG(tomb)` nevű függvény használatával a program generálja a PNG fájlt. pixelről pixelre.

A programot a következőképpen tudjuk fordítani: `g++ mandelbrot.cpp -lpng16 -o mandelbrot` Futtatni pedig a szokásos módon `./mandelbrot` parancsal tudjuk.

## 5.2. A Mandelbrot halmaz a `std::complex` osztályal

Tutorálóm: [Duszka Ákos Attila](#)

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Mandelbrot/mandelbrotkomplex.cpp>

A megoldás forrása nem az én tulajdonom. Az eredeti forrás megtalálható az [UDPROG](#) repóban.

Ebben a feladatban a végeredmény ugyan az kellene hogy legyen, mint az előző feladatban. Illetve azóta még a mandelbrot halmaz lényege sem változott, ezért azt nem írnám le újra.

A `png++` ebben az esetben is kelleni fog, így ha nincs leszedve, akkor pillants az előző feladat magyarázatára, ahol megtalálod a szükséges dolgokat ahoz, hogy le tudjon fordulni a program.

Ebben az esetben az `std::complex` osztályt fogjuk használni a program megvalósításához. Ez az osztály, ahogy a neve is utal rá, a komplex számok kezelése miatt jött létre.

A program által használt függvényei a következők:

A `real(C)` a komplex szám valós részét határozza meg.

A `imag(C)` a komplex szám képzetes részét határozza meg.

Legelőször a program

```
#include <png++-0.2.9/png.hpp>
#include <complex>
```

beincludeolja a `png++`-t és a komplex osztályt

Ezek után az előző feladathoz hasonlóan itt is megadjuk a végleges értékeket az N, M valamit X és Y maximum és minimum értékeinek.

Legnagyobb részben ennek a feladatnak a megoldása megegyezik az előző feladat megoldásával, ezért azt nem írnám le újra, inkább arra koncentrálnék, hogy miben más ez a forrás mint az előző.

Az érdemi különbség a két forrás között az az, hogy itt az `std::complex` osztályt használva, már nem kell létrehoznunk egy saját struktúrát a komplex számoknak.

E helyett szimplán létrehozzuk a double típusú komplex számokat a következőképpen:

```
std::complex<double> C, Z, Zu;j;
```

Illetve a for cikluson belül sem a struktúrán belüli elemek imaginárius és valós részére hivatkozunk, hanem a `real()` és `imag()` nevű függvényeket meghívva mondjuk meg a komplex szám részeinek értékét.

```
real(C) = MINX + j * dx;  
imag(C) = MAXY - i * dy;
```

A programot fordítani és futtatni ugyan úgy kell, mint az előző feladatot.

### 5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbgrZy76E>

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/Biomorf](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf)

A megoldás forrása [Bátfai Norbert](#) tulajdona.

A két előző feladathoz hasonlóan itt is szükségünk van a `png++` ra, ezért ha még nem szedted le akkor pillants rá a az első feladat magyarázatára, ahol részletesen le vannak írva az ehez szükséges parancsok.

Ez egy olyan mandelbrot program, ahol maga a user adja meg a határokat. Előnye hogy az eredetihez képest teljesen más képeket kapunk, hátránya viszon hogy ha a user nem tudja, hogy mit csinál akkor az egész kép egy nagy fekete semmi lesz.

Először is

```
#include <iostream>  
#include "png++/png.hpp"  
#include <complex>
```

includeoljuk az `iostream`et a `png++` és a komplex osztályt.

A `main` argumentumai a bemeneti adatok, amikből előállítjuk magát a képet.

Ellenőrzi a program, hogy megfelelő mennyiségű bemeneti értéket adott-e meg a felhasználó, és ha nem, akkor felvilágosítja, hogy hogyan kell használni a programot.

```
if ( argc == 12 )  
{  
    szelesseg = atoi ( argv[2] );  
    magassag = atoi ( argv[3] );  
    iteraciosHatar = atoi ( argv[4] );  
    xmin = atof ( argv[5] );  
    xmax = atof ( argv[6] );  
    ymin = atof ( argv[7] );  
    ymax = atof ( argv[8] );  
    reC = atof ( argv[9] );  
    imC = atof ( argv[10] );  
    R = atof ( argv[11] );  
  
}
```

```
else
{
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ←
                  d reC imC R" << std::endl;
    return -1;
}
```

Ha viszont megfelelő mennyiségű argumentumot adott meg a felhasználó, akkor létrehozza a képet aminek a szélessége és a magassága a felhasználó által megadot szélesség és magasság lesz.

```
png::image < png::rgb_pixel > kep ( szelesseg, magassag );
```

Ezek után a program két egymásba ágyazott for ciklus segítségével kiszámolja, és létrehozza a képet, és el is menti a felhasználó által megadott néven.

A fordítása az előző két programhoz hasonlóan működik, a futtatása azonban már így néz ki:

**./3.1.3 fajlnev szelesseg magassag n a b c d reC imC R** Erre egy példa:

**./3.1.3 biomorf.png 800 800 10 -2 2 -2 2 .285 0 10**

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/CUDA/mandelpngc\\_60x60\\_100](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/CUDA/mandelpngc_60x60_100)

A megoldás forrása Bátfai Norbert tulajdona.

Először is közérdekű közlemény, hogy ennek a programnak a sikeres fordításához szükségünk lesz egy CUDA magokat használó NVIDIA kártyára, illetve az nvidia-cuda-toolkit re amit a következő parancsal tudunk felenni:

**sudo apt install nvidia-cuda-toolkit**

Ez a program ugyanúgy a mandelbrot halmazt rajzolja ki, mint az előzőek, azonban itt egy nagyon fontos különbség az, hogy míg az előző feladatoknál a képet a CPU számolta ki és készítette el, addig itt, az NVIDIA kártyák CUDA magjait használjuk a kép kiszámításához.

Ez azért fontos, mert az előző feladatoknál egyetlen egy mag dolgozott és számolt ki minden, addig itt az én GTX 1050TI videókártyám esetében 768 darab cuda mag számolja és rajzolja ki a képet.

Ez nyílvánvalóan egy sokszor gyorsabb futási időt eredményez. Az én esetemben például amikor CPU-val futtattam a programot akkor a következő eredmények jöttek ki:

```
andras@andrasubuntu:~/cuda mandel$ ./mandelp t.png
2573
25.7395 sec
t.png mentve
```

Ez azt jelenti, hogy egy AMD FX8350 processzornál majdnem 26 másodpercbe került, hogy lefusson a program, és elkészüljön a kép.

Azonban ha már a fentebb említett GTX 1050TI kártyát használva futtatom a programot, akkor már egy kicsit hamarabb lefut a program.

```
andras@andrasubuntu:~/cuda mandel$ ./mandelcuda c.png
c.png mentve
4
0.047982 sec
```

Ezek alapján így már minden össze 0.05 másodpercebe került futtatni a programot ami egy jelentős csökkenés. Pontosabban körülbelül 514-szer gyorsabban futott le ezzel a módszerrel a programunk.

A programot a gcc helyett az nvcc nevű parancssal kell fordítani. Futtatni pedig a szokásos módon.

## 5.5. Mandelbrot nagyító és utazó C++ nyelven

Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteráció bejárta  $z_n$  komplex számokat!

**A feladat nem lett elkezdve időben ezért passzolásra került.**

Megoldás videó:

Megoldás forrása:

## 5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás Forrása: <https://github.com/raczandras/progbook/tree/master/src/Mandelbrot/MandelJava>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

A feladat az ezt megelőző (passzolt) feladat átírása Java nyelvre. A GUI megírásához szükség van egy keretrendszerre, ami jelen esetben az Abstract Window Toolkit lesz.

Először nézzük a *MandelbrothalmaZ.java* fájlt.

A main-ben a *MandelbrotHalmaz()* meghívásával létrehozunk egy új halmazt a megadott paraméterekkel. Ezek a paraméterek a tartományok koordinátái, a halmazt tartalmazó tömb szélessége, és a számítás pontossága.

Utána a felhasználó tevékenységeit figyeli a program, és megfelelően reagál rájuk, valamint a GUI ablak tulajdonságait adja meg, illetve kirajzolja magának a halmaznak a képét.

A következő fájlunk a *MandelbrotHalmazNagyító.java*

A nevéről adódóan ez végzi a halmazon a nagyítás folyamatát, illetve magának a halmaznak a kirajzolását is. Maga a *MandelborHalmazNagyító* osztály figyeli a felhasználó egér tevékenységeit, azzal kapcsolatban, hogy hol szeretné nagyítani a képet, illetve kirajzolja az új, nagyított képet. Ezen kívül ez végzi az elmentendő képek készítését, és elmentését is.

Végül pedig a *MandelbrotIterációk.java* fájl szerepe.

Ez a programrészlet a nagyított mandelbrot halmazok pontjait tartja nyílván. Ez egy számításra létrehozott osztály, ami a kiválasztott ponthoz tartó utat mutatja meg.

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

Valósítsd meg C++-ban és Java-ban az módosított polártranszformációs algoritmust! A matek háttér teljesen irreleváns, csak annyiban érdekes, hogy az algoritmus egy számítása során két normálist számol ki, az egyiket elspájzolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltéve kiszámolt szám.

Megoldás videó:

C++ forrás: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/polargen/>

java forrás: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsojava/PolarGen.java#l10>

Ehhez a programhoz java-ban szükségünk lesz az util.random, az io.\* illetve a lang.math java könyvtárára. Először is a bExist változót hamisra állítjuk a konstruktoron belül, majd pedig inicializálunk egy randomot, ennyi a konstruktor.

Ezek után a PolarGet függvény ami az érdemi munkát végzi. Először is ellenőrzi, hogy volt-e már generálás. Ha volt akkor azt adja vissza, de ha nem, akkor a matekos algoritmus segítségével legenerálja a két random normált és bExists-et átállítja az ellentetjére.

Érdekes, hogy a JDK-n belül is ez a megoldás van alkalmazva, ami annyit jelent, hogy azok akik a random java könyvtárat megírták, azok ugyan úgy gondolkoztak mint egy egyetemi hallgató.

### 6.2. LZW

Valósítsd meg C++-ban az LZW algoritmus fa-építését!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Welch/lzw.cpp>

A megoldás forrása nem az én tulajdonom. Az eredeti forrás megtalálható az **UDPROG** repóban.

Mindkét esetben a bináris fa felépítésének a lépései a következők:

Ha 1-est szeretnénk betenni a fába, akkor először megnézzük, hogy az aktuális csomópontnak van-e már ilyen eleme. Ha még nincs, akkor egyszerűen betesszük neki az 1-es gyermekének az 1-et. Azonban ha már van ilyen gyermek, akkor létre kell hozni egy új csomópontot és az ő gyermekének adjuk át az 1-et.

Ez hasonlóan működik akkor is ha nullást szeretnénk betenni, annyi különbéggel, hogy nem az 1-eseket vizsgáljuk, hanem a nullásokat. Ezt a lépést a programban a következő részlet oldja meg:

```
void operator<<(char b) {
    if (b == '0') {
        if (!fa->nullasGyermek()) {
            Csomopont *uj = new Csomopont('0');
            fa->ujNullasGyermek(uj);
            fa = &gyoker;
        } else {
            fa = fa->nullasGyermek();
        }
    } else {
        if (!fa->egyesGyermek()) {
            Csomopont *uj = new Csomopont('1');
            fa->ujEgyesGyermek(uj);
            fa = &gyoker;
        } else {
            fa = fa->egyesGyermek();
        }
    }
}
```

A megadott fájl tartalma alapján felépíti az LZWBinfa csomópontjait. Jelen esetben ezt a Bináris Fát in order bejárással dolgozzuk fel, ami annyit jelent, hogy először a fa bal oldalát dolgozzuk fel, majd a fának a gyökerét, és legvégül pedig a jobb oldalt. A következő feladatban ezen viszont már változtatunk.

Fordítása a szokásos módon történik a futtatása pedig a következőképpen:

**./lzw bemenet -o kimenet**

## 6.3. Fabejárás

Tutorálóm: [George Butcován](#)

Járd be az előző (inorder bejárású) fát pre- és posztorder is!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Welch/fabe.cpp>

A megoldás forrása nem az én tulajdonom. Az eredeti forrás megtalálható az [UDPROG](#) repóban.

Az előző feladatban tárgyalt fát In Order módszerrel járta be a program. Ez azt jelenti, hogy először a részfa bal oldalát dolgozzuk fel, majd a részfa gyökerét, és legvégül pedig a részfa jobb oldalát.

Erre ugyanúgy megmaradt a lehetőségünk, csupán a következőképp kell futtatni a programot:

**./lzw bemenet -o kimenet i**

Ezzel szemben itt két másik fajta bejárási módszerrel dolgozzuk fel a fát. Az egyik a Pre Order bejárási mód, a másik pedig a Post Order.

A Pre Order bejárási módnál először a részfa gyökerét dolgozzuk fel, másodjára a részfa bal oldalát, és utoljára pedig a részfa jobb oldalát. A pre order bejárási mód használatához a következőképpen kell futtatni a programot:

**./lzw bemenet -o kimenet r**

A Post Order bejárási módnál pedig legelőször a részfa bal oldalát dolgozza fel a program, majd a jobb oldalát, és legvégül pedig a részfa gyökerét. A Post Order bejáráshoz a következő parancs használatával kell futtatni a programot:

**./lzw bemenet -o kimenet r**

## 6.4. Tag a gyökér

Az LZW algoritmust ültess át egy C++ osztályba, legyen egy Tree és egy beágyazott Node osztálya. A gyökér csomópont legyen kompozícióban a fával!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Welch/tag.cpp>

A megoldás forrása nem az én tulajdonom. Az eredeti forrás megtalálható az **UDPROG** repóban.

Ez a program az eredeti Bevezetés a Programozásba tárgyon már tanult *z3a7.cpp* nevű program szerint működik, hiszem itt a csomópont már kompozícióban van a fával. Az egész az LZWBinfá osztállyal kezdődik, aminek van privát, és publikus része is. A publikus részen belül található a konstruktor, és a destrukturációja. Itt kerülnek vizsgálatra a bemenő elemek, és jönnek létre a 0-s illetve 1-es elemek is. Túlterhelődik az operátor, és megvizsgálja a program, hogy létezik-e már nullás gyermek. Ha nincs, akkor létrejön. Egyes gyermeknél ugyan ez a helyzet.

A kiír függvény pedig kiírja a csomópontokat.

Majd jön az LZWBinfá privát része. Itt található meg a Csomópont osztály amin belül a konstruktor megkapja a gyökeret. Még a Csomópont osztályon belül találhatóak azok a függvények, amivel le tudjuk kérdezni, hogy ki az aktuális csomópont nullás illetve egyes gyermeké, valamint az *ujNullasGyermek()* illetve *ujEgyesGyermek()* függvények, amik létrehozzák az új nullás és egyes gyermeket

## 6.5. Mutató a gyökér

Írd át az előző forrást, hogy a gyökér csomópont ne kompozícióban, csak aggregációban legyen a fával!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Welch/gyoker.cpp>

Ehhez a feladathoz az **UDPROG** repóban megtalálható BinFa programot vettem alapul.

Ebben a megoldásban az előző feladathoz képest kicsit másképp a megoldás. A következő dolgokat kell átírni a már meglévő programban:

Először is a 315. sorban a csomopont után tegyük egy \*-ot ezzel mutatóvá téve a gyökeret. Ha így megpróbáljuk lefordítani a programot akkor nagyon sok szintaktikai hibát fogunk kapni a fordítótól válaszként.

Nem kell pánikolni. Az a dolgunk, hogy ezeket a hibákat egyesével kijavítsuk. Az első két hiba kijavításához a következő részletet kell átírni.

A 92. és a 93. sorban a

```
szabadít (gyoker.egyesGyermek ());
szabadít (gyoker.nullasGyermek ());
```

utasítások helyett a

```
szabadít (gyoker->egyesGyermek ());
szabadít (gyoker->nullasGyermek ());
```

utasításokat kell használni.

Ez után már kettővel kevesebb hibát kapunk. Az összes többi hibát a referenciák okozzák. Ahoz hogy ezeket a hibákat megoldjuk a következő sorokban kell tevékenykednünk: 92, 132, 147, 170, 210, 336, 344, és 356. Azonban a hibát minden sorban ugyan azzal a módszerrel kell javítani, ami nem más mint hogy a

```
&gyoker
```

helyett azt kell írni hogy

```
gyoker
```

Vagyis kiszedjük a referenciákat, mivel alapból a memóriacímek lesznek átadva.

Ezek után a programunk ugyan lefordul, de amikor megprobáljuk futtatni, akkor szegmentálási hibát kapunk. Ennek a javításához a konstruktort kell átírni a következőképpen:

```
LZWBinFa () {
    gyoker = new Csomopont ();
    fa = gyoker;
}
```

Ezzel foglalunk helyet a memóriában a gyökérnek. Viszont amit lefoglalunk, azt fel is kell szabadítani, éppen ezért a destruktort is módosítani kell a következőképpen:

```
~LZWBinFa ()
{
    szabadít (gyoker->egyesGyermek ());
    szabadít (gyoker->nullasGyermek ());
    delete gyoker;
}
```

Mostmár fel is szabadul, amit lefoglaltunk.

## 6.6. Mozgató szemantika

Tutorálom: [Molnár Antal](#)

Írj az előző programhoz mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Welch/mozgato.cpp>

A megoldás forrásának az alapja megtalálható az **UDPROG** repóban. Én ezt módosítottam.

Maga az LZWBInFa osztály felépítése úgy néz ki, hogy az osztályon belül léteznek a beágyazott csomópont objektummok amik a fát alkotják. Ezek alapján a fa másolása nem más, mint ezeknek a csomópontoknak a másolása. Ehhez létre kell hoznunk a mozgató illetve mozgatás konstruktorokat.

```
LZWBInFa (LZWBInFa&& masik) {
    gyoker=nullptr;
    *this= std::move(masik);

}

LZWBInFa& operator= (LZWBInFa&& masik) {
    std::swap(gyoker,masik.gyoker);
    return *this;

}
```

Először a mozgató értékadásról (alsó) szólnék pár szót, ami csupán annyit jelent, hogy ha egyenlőségjel operátort használunk, akkor az `std::swap()` függvényel megcserélődik a két gyökér mutatója.

Másodsor pedig a mozgató konstruktur. Itt először is `nullptr` (`nullpointer`) értéket adunk abban a binfában lévő gyökérnek, amelyik fába akarjuk mozgatni a ("masik") fát. Majd a "masik" nevű fát átmozgatjuk az `std::move()` függvényvel, ami annyit jelent, hogy a gyökér mutató mostmár a paraméterként kapott "masik" fára mutat, ami azért történhetett meg, mert az `std::move()` függvény tulajdonképpen nem is mozgat semmit, hanem a paraméterül kapott értéket jobbérték referenciává alakítja.

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyszimulációk

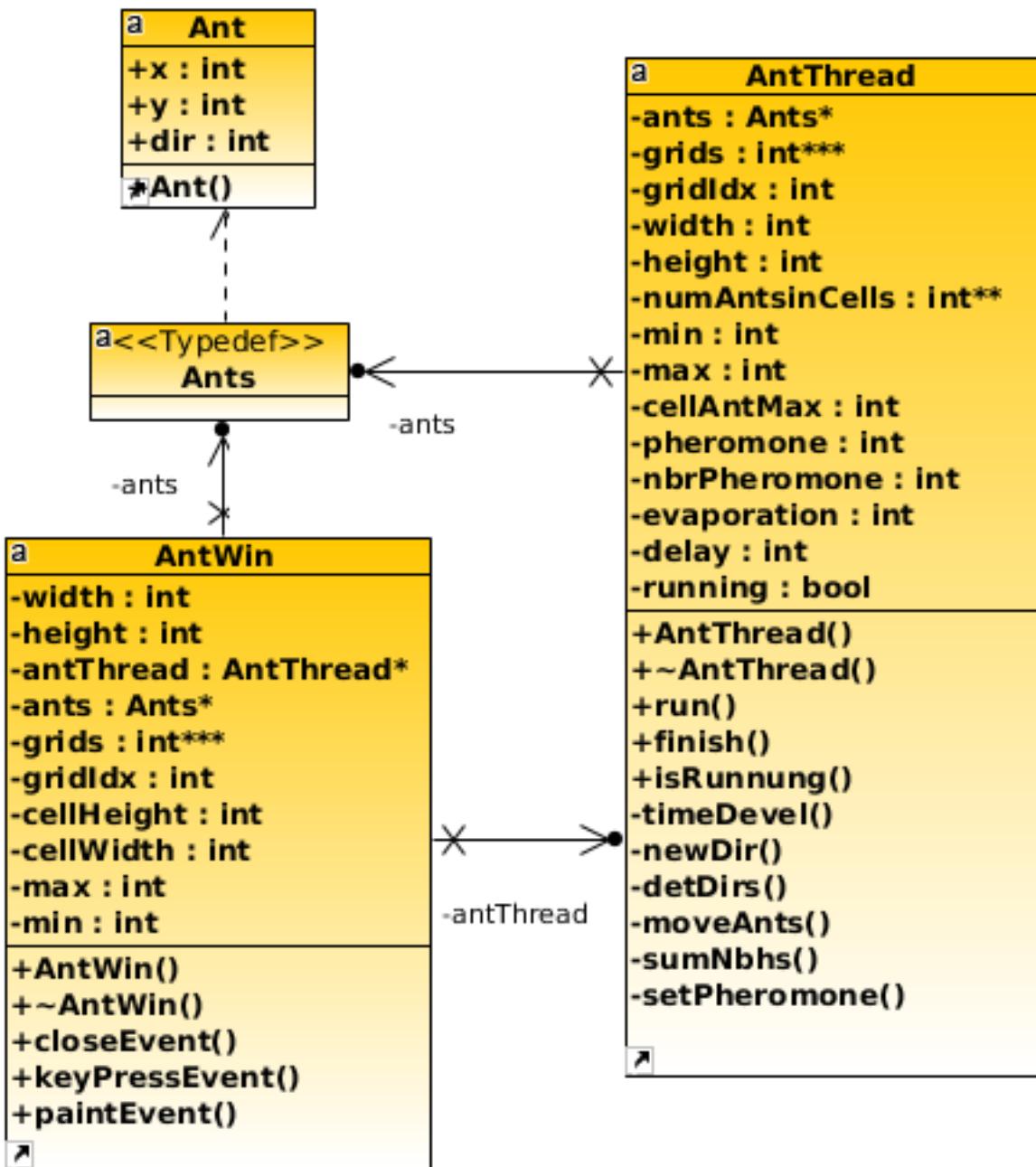
Írj Qt C++-ban egy hangyszimulációs programot, a forrásaidról utólag reverse engineering jelleggel készíts UML osztálydiagramot is!

Tutoráltam: Molnár Antal

Megoldás videó: <https://bhaxor.blog.hu/2018/10/10/myrmecologist>

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/Conway/Ant>

Az osztálydiagram:



A megoldás forrása, illetve az UML osztálydiagram [Bátfai Norbert](#) tulajdona.

Ebben a feladatban hangyákat kell szimulálni. Maga a megoldás azt a biológiai tényt alkalmazza, hogy a hangyák a való életben szagokkal, úgynevezett feromonokkal kommunikálnak egymással. Ha például egy hangya valamilyen érdekes dolgot talált, akkor ott hagyja a nyomát, illetve megjelöli az útvonalat. Az éppen arra járó többi hangya ezt megérzi, és a legfrissebb feromon nyomát követve ők is el fognak jutni a célba. Ezeket észben tartva készítették el ezt a hangyaszimulációt.

Az osztálydiagrammon belül négy egységet találhatunk, ezek a következők: Ant; Ants; AntWin; és AntThread

Ezek a programunk osztályai, ezeken az egységeken belül vannak megadva az adott osztály változói és függvényei. Ilyen például az AntWin egységen belül található *width* és *height* változók, amik a képernyő

hosszúságát, és szélességét adják meg. Vagy például a `closeEvent()` és a `keyPressEvent()` függvények, amik szintén az AntWin osztály részei. Ezek alapján meghatározhatjuk, hogy az AntWin osztály a szimuláció belül a világot kezeli.

Az AntThread osztály kezeli a hangyákat, illetve azok mozgását, illetve a virtuális feromonok terjedéséről is ez az osztály gondoskodik.

## 7.2. Java életjáték

Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/Conway/sejt.java>

A Megoldás forrása [Bátfai Norbert](#) tulajdona.

Arról, hogy mi az az életjáték, illetve, hogy mik a szabályai, a következő feladat leírásában részletesebben írok. Most legyen elég ennyi:

Ha egy négyzetnek pontosan három darab élő szomszédja van, akkor abban a négyzetben egy új sejt jön létre.

Ha egy már élő sejtnek pontosan kettő vagy három darab szomszédja van, akkor az a sejt továbbra is életben marad.

Ha viszont egy már meglévő élő sejtnek háromnál több élő szomszédja van (túlnépesedés), vagy kettőnél kevesebb, akkor az a sejt meghal. Ezt szimulálja az életjáték.

Ezek a szabályok az `időFejlődés()` függvényben vannak lefektetve.

```
if(rácsElőtte[i][j] == ÉLŐ) {  
    /* Élő élő marad, ha kettő vagy három élő  
     szomszedja van, különben halott lesz. */  
    if(élők==2 || élők==3)  
        rácsUtána[i][j] = ÉLŐ;  
    else  
        rácsUtána[i][j] = HALOTT;  
} else {  
    /* Halott halott marad, ha három élő  
     szomszedja van, különben élő lesz. */  
    if(élők==3)  
        rácsUtána[i][j] = ÉLŐ;  
    else  
        rácsUtána[i][j] = HALOTT;
```

Igaz ugyan, hogy az életjáték egy úgynevezett nullszemélyes játék, de ebben a példában a játékos mégis tudja irányítani kicsit a dolgokat. Ugyanis a program figyeli a billentyűzet bizonyos gombjait (`k`, `n`, `l`, `g`, `s`), illetve az egér mozgását, és kattintásait is. Ezt három fügvénnyel teszi. Az `addKeyListener(new java.awt.event.KeyAdapter())` fügvénnyel figyeli a billentyűzetet. Ezen a függvényen belül egy `if-else` szerkezet állapítja meg, hogy éppen melyik gombot nyomta le a felhasználó a

```
if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {}
```

feltétel a K betű lenyomását azonosítja, és csökkenti a sejtek méretét.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {}
```

feltétel az N betű lenyomását azonosítja, és növeli a sejtek méretét.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S) {}
```

feltétel az S betű lenyomását azonosítja, és készít egy képet a sejttérről.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
```

feltétel a G betű lenyomását azonosítja, gyorsítja a szimuláció sebességét.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
```

feltétel az L betű lenyomását azonosítja, és lassítja a szimuláció sebességét.

Az egér mozgását, illetve kattintásait pedig a addMouseListener(new java.awt.event.MouseAdapter() {} függvények figyelik. Az egér kattintásaival egy sejt állapotát tudjuk megváltoztatni. Az egér mozgatásával pedig az összes érintett sejt elő állapotba kerül.

### 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/Conway/Qt>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Az életjátékot John Conway találta ki, és nem teljesen hiteles rá a játék kifejezés, mert ez egy úgynevezett nullszemélyes játék. Magának a játékosnak annyi a dolga, hogy megadja a kezdőalakzatot, majd pedig megfigyelheti, hogy mi lesz az eredmény.

Alapja egy négyzetrácsos tér, amikben élhetnek sejtek, de minden egyes négyzetben csak egy darab sejt élhet. Magának a "játéknak" a szabályai a következők:

Ha egy sejtnak kettő vagy három élő szomszédos sejtje van, akkor a sejt meg fogja elni a következő generációt. Az összes többi esetben viszont kihal a sejt, akár azért mert túl sok, akár azért mert túl kevés szomszédja van.

Ahol azonban egy üres négyzetrácsnak pontosan három élő sejt a szomszédja, akkor ott új sejt jön létre.

Ez mellesleg két részre osztotta az embereket. Voltak akik minden napi rutinjukká tették azt, hogy az életjátékkal "játszanak", egyfajta függők lettek, és voltak azok, akik nem értették hogy mi a jó benne.

Maga a program ugyanúgy működik, mint a java verzió. Mind a két program két darab mátrixtal dolgozik, viszont itt a teljes kód megírása helyett a Q-t is segítságül hívjuk.

A programot a következőképpen tudjuk fordítani és futtatni: **qmake Sejtauto.pro make ./Sejtauto**

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/Conway/BrainB>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

A programhoz szükségünk lesz az OpenCV-re, aminek a feltelepítéséhez a lépéseket [ezen a linken](#) elérhető weblapon találjuk.

Ez egy miniatűr játék, ami a felhasználó szem-kéz koordinációjáról, illetve a megfigyelőképességéről gyűjt össze információkat.

Amikor elindítjuk a játékot akkor egy ablak fogad minket, és a lényege az, hogy a **Samu enthropy** nevű négyzeten belül lévő fekete pöttyön belül tartsuk az egerünk mutatóját.

A játék a teljesítményünk alapján lesz könnyebb, vagy nehezebb. Minél jobban játszunk, annál több Entropy lesz a képernyőn, ezáltal megnehezítve a Samu entropy követését. Viszont ha már nem tudjuk nyomon követni a Samu entropy-t akkor folyamatosan eltünteti a hozzáadott entropy-kat, ezáltal megkönnyítve a játékot.

Én közel két perc játék után a következő eredményeket produkáltam:

```
NEMESPOR BrainB Test 6.0.3
time      : 1164
bps       : 52830
noc       : 17
nop       : 0
lost      :
4700 9800
mean      : 7250
var       : 3606.24
found     : 11740 19150 12360 36950 49930 33470 38680 20860 4790 2230 16870 ←
       6500 25000 50340
mean      : 23490
var       : 16066.3
lost2found: 2230
mean      : 2230
var       : 0
found2lost:
mean      : 0
var       : 0
time      : 1:56
U R about 0.136108 Kilobytes
```

A programot futtatni az előző feladathoz hasonlóan szintén a **qmake** és **make** parancsokkal lehet fordítani.

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása: [https://progpater.blog.hu/2016/11/13/hello\\_samu\\_a\\_tensorflow-bol](https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol)

A megoldás forrása [Bátfai Norbert](#) tulajdona.

Ennél a feladatnál Tensorflow-t fogunk használni, ami egy nyílt forráskódú szoftver, amit főleg Machine Learning-nél (Gépi tanulás) használnak. A nagy cégek, pl Google is ezt használják. Jelentősége, hogy egyszerre több CPU-n és GPU-n is képes futni. Azonban ahoz, hogy ezt használhassuk, fel is kell telepítenünk.

Ez a program a TensorFlow Hello World-je. Két számot szoroz össze neurális hálókat használva.

```
#!/usr/bin/env python2
# TensorFlow Hello World 1!
# twicetwo.py
#
import tensorflow as tf

node1 = tf.constant(2)
node2 = tf.constant(2)

node_twicetwo = tf.math.multiply(node1, node2, name="twicetwo")

sess = tf.Session()
print sess.run(node_twicetwo)

writer = tf.summary.FileWriter("/tmp/twicetwo", sess.graph)
# nbatfai@robopsy:~/Robopsychology/repos/tf/tf/tensorboard$ python ←
    tensorboard.py --logdir=/tmp/twicetwo

tf.train.write_graph(sess.graph_def, "models/", "twicetwo.pb", as_text=←
    False)
```

A program importálja a tensorflow-t tf néven. Majd a node1-nek illetve a node2-nek értékül adja a 2 értéket a

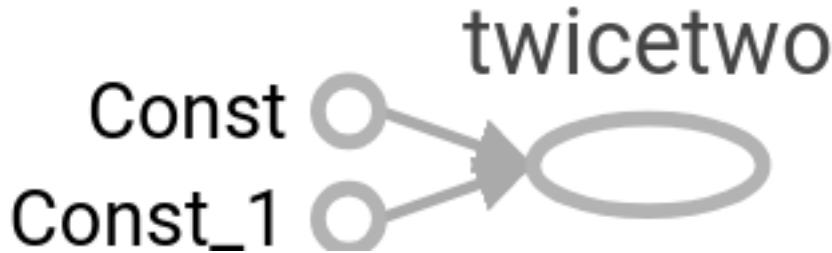
`tf.constant(2)` függvény segítségével.

Ezek után a `tf.math.multiply()` függvényteljesen kiszámolja a két szám szorzatát, és azt értékül adja a `node_twicetwo`-nak.

Végül kiírja a szorzatot a `sess = tf.Session()` értékadással és függvényel, illetve a `print sess.run(node_twicetwo)` parancssal.

Es a számitási gráfot a `writer = tf.summary.FileWriter("/tmp/twicetwo", sess.graph)` értékadással

illetve a `tf.train.write_graph(sess.graph_def, "models/", "twicetwo.pb", as_text=False)` függvényel.



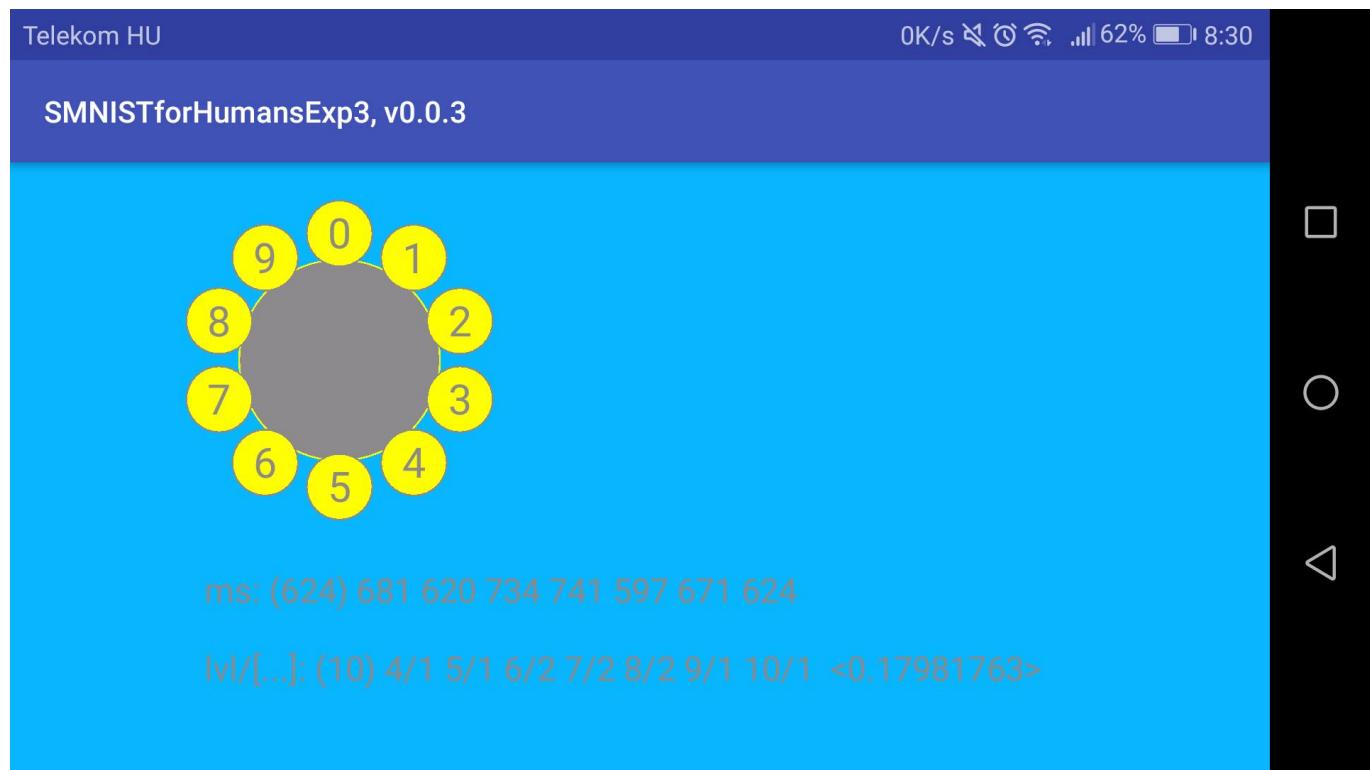
## 8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

**A feladatot az SMNIST-ben elért eredményem alapján passzoltam.**

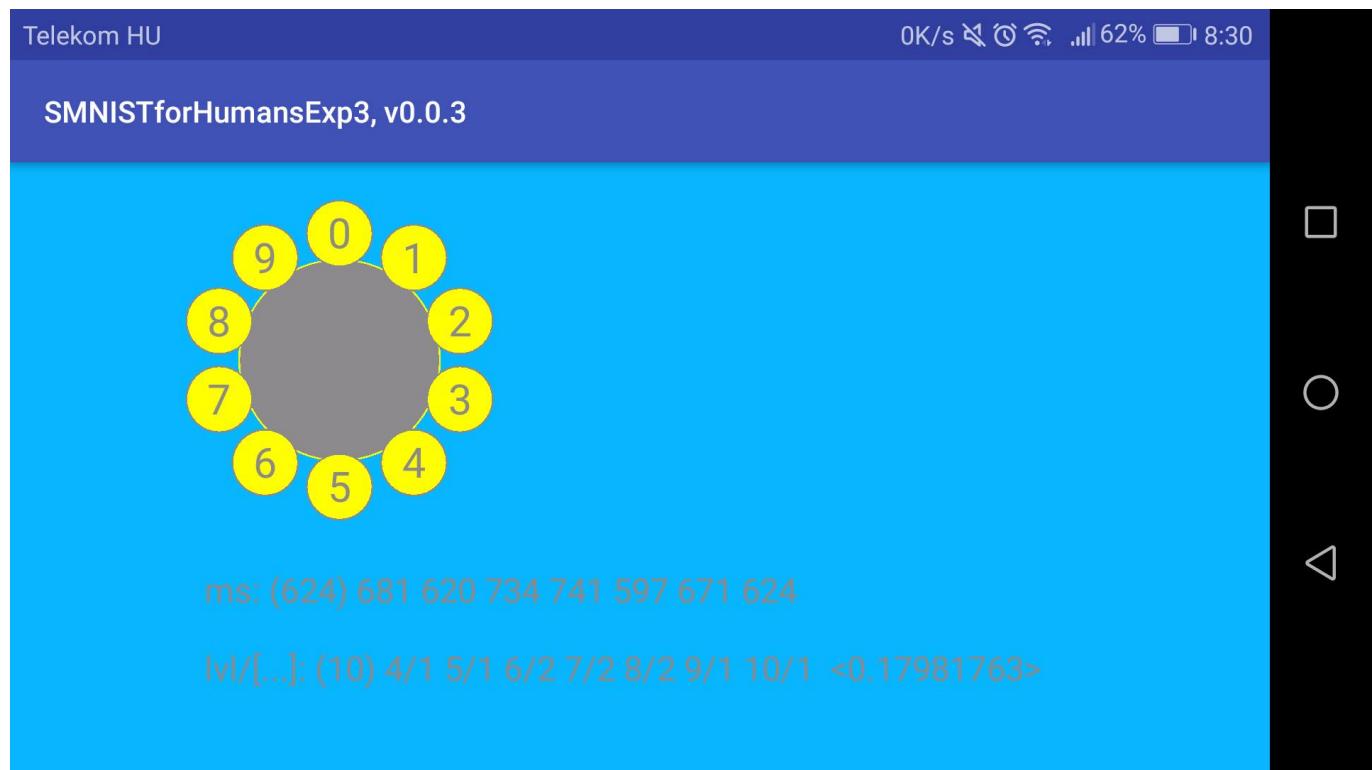


### 8.3. Minecraft-MALMÖ

Megoldás videó:

Megoldás forrása:

**A feladatot az SMNIST-ben elért eredményem alapján passzoltam.**



## 9. fejezet

# Helló, Chaitin!

### 9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás video:

Iteratív megoldás: <https://github.com/raczandras/progbook/blob/master/src/Chaitin/iter.lisp>

Rekurzív megoldás: <https://github.com/raczandras/progbook/blob/master/src/Chaitin/rek.lisp>

A forrást a [codeforsharing](#) inspirálta.

Ebben a feladatban a program faktoriálist számol iteratív illetve rekurzív módon. A lisp a második magas szintű programozási nyelv. Egyedül a fortran előzte meg. Először nézzük az iteratív módszert.

```
(defun fact (n)
  (setf f 1)
  (do ((i n (- i 1))) ((= i 1))
       (setf f (* f i)))
  )
fact (4))
```

A program első sorában definiáljuk magát a függvényt `fact` néven. Majd egy `f` nevű változót, aminek az értéket 1-re állítjuk. Majd jön egy ciklus, ami `i`-nek átadja a számot aminek a faktoriálisát ki kell számolni. A program `i`-ből folyamatosan kivon 1-et egészen addig, amíg `i` értéke 1 nem lesz. A ciklus törzsében pedig `f` értéke  $f \cdot i$  lesz. Végül a program meghívja magát a `fact ()` függvényt.

Ezzel szemben a rekurzív módszert valamennyivel könnyebb olvasni.

```
(defun fact (n)
  (if (= n 0) 1
      (* n (fact (- n 1))))
  )
fact (4))
```

Először ebben a példában is a `fact ()` függvény kerül definiálásra. Azonban ezek után itt egy `if` szerepel, ami azt ellenőrzi, hogy `n` egyenlő-e 0-val. Ha nem, akkor szimplán meghívja a függvény saját magát, azonban itt már `n-1` amivel számol, így számolva ki a faktoriális értékét.

## 9.2. Gimp Scheme Script-fu: króm effekt

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szöveget!

Megoldás videó: [https://youtu.be/OKdAkl\\_c7Sc](https://youtu.be/OKdAkl_c7Sc)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Chrome](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome)

A megoldás forrása Bátfai Norbert tulajdona

Ez egy program a gimp-hez, ami a megadott szöveghez króm effektet ad. Maga a gimp egy ingyenes képszerkesztő program.

Maga a forrás egy tömbbel indul, ami a króm effekt megvalósításához szükséges információkat tartalmazza.

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)
```

A következő függvény a betűk méretét határozza meg. A szükséges méretekek a a GIMP beépített függvényeivel határozza meg a következőképpen:

```
(set! text-width (car (gimp-text-get-extents-fontname text fontsize PIXELS ←
  font)))
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
```

Ez annyit jelent, hogy a `gimp-text-get-extents-fontname` első értékét (ami maga a méret) állítja be a `text-width` illetve a `text-height` változóknak a `set` utasítás használatával.

Majd a `script-fu-bhax-chrome-border` függvény hozza létre a tényleges króm effektes szöveget. Ezt egy új rétegen (layer) teszi. Ennek az új rétegnél a hátttere fekete, a rá kerülő szöveg pedig fehér színű lesz.

```
(gimp-image-insert-layer image layer 0 0)

  (gimp-image-select-rectangle image CHANNEL-OP-ADD 0 (/ text-height 2) ←
    width height)
  (gimp-context-set-foreground '(255 255 255))
  (gimp-drawable-edit-fill layer FILL-FOREGROUND )
```

Végül a program regisztrálásra kerül magába a gimp-be, hogy el tudjuk érni

### 9.3. Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/10/a\\_gimp\\_lisp\\_hackelete\\_a\\_scheme\\_programozasi\\_nyelv](https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelete_a_scheme_programozasi_nyelv)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Mandala](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala)

A megoldás forrása Bátfai Norbert tulajdona.

Az előző feladathoz hasonlóan itt is egy gimp kiegészítőről van szó. Itt azonban a bemenő szövegből egy név-mandala fog készülni. A mandala egy szimmetrikus kör alakú kép, ami a Hindu vallásban nagy szerepet játszik a Hindu istenek ábrázolásában.

Először a program meghatározza a szöveg hosszát, a gimp-text-get-extents-fontname függvény használatával. A kapott értéket a set! utasítással a text-width változó értékenek adja.

```
(set! text-width (car (gimp-text-get-extents-fontname text fontsize PIXELS ←
    font)))
```

Ebben a feladatban, ugyanúgy határozzuk meg a szöveg méretét, mint az előzőben:

```
(set! text-width (car (gimp-text-get-extents-fontname text fontsize PIXELS ←
    font)))
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font))))
```

A GIMP beépített gimp-text-get-extents-fontname függvényét, és a set! utasítást felhasználva a text-width és a text-height változók értékei lesznek a szükséges méretek.

Ezek után jön maga a mandala. Először létrejön egy réteg (layer)

```
(gimp-image-insert-layer image layer 0 0)

  (gimp-context-set-foreground '(0 255 0))
  (gimp-drawable-fill layer FILL-FOREGROUND)
  (gimp-image-undo-disable image)
```

Amit feltöltünk a felhasználó által megadott adatokkal. Ezek a a szöveg, a szöveg betűtípusa. Ezután a réteget tükrözi a program, ezzel elérve a szimmetriát, majd a program elforgatja a képet, és megismétli a tükrözést. Majd a réteget felnagyítja a kép teljes méretére:

```
(gimp-layer-resize-to-image-size textfs)
```

Ezután két körnek álcázott ellipszist illeszt a program a képre ezzel létrehozva a mandalát. Az egyik kör vastagsága 22, a másiké pedig 8 lesz.

```
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) (/ ←
    textfs-width 2)) 18)
    (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
        textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)
```

```
(gimp-context-set-brush-size 22)
(gimp-context-set-brush-size 8)
```

Végül pedig megjeleníti a képet:

```
(gimp-display-new image)
```

Ezek után ismét már csak a GIMP-be regisztrálás van hátra.

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Juhász István - Magas szintű programozási nyelvek 1 olvasónaplója

Ez a Rácz András nevű Debreceni Egyetemi hallgató által készített olvasónapló a Juhász István által írt Magas szintű Programozási Nyelvek 1 (pici-könyv) című könyvből.

A számítógépet programozó nyelveknek három szintje van. Ezek a gépi nyelv, az assembly nyelv és a magas szintű nyelv. Mi a magas szintű programozási nyelkekkel foglalkozunk, ami az emberek által a legjobban érthető. Az ilyen nyelven megírt programot nevezzük forrásprogramnak. Azonban a processzorok csak az adott gépi nyelven írt programokat tudják végrehajtani. Ezért forrásprogramot át kell írni gépi kódra. Ezt a munkát végzik el a fordítók.

Minden programnyelvnek saját szabványa, ez a hivatkozási nyelv. Pontosan meg vannak adva a nyelvtani szabályok, amiket be kell tartani, különben vagy szintaktikai, vagy szemantikai hibát fogunk kapni. A szintaktikai hiba az, amit a fordító észrevesz, és jelez nekünk, hogy gond van. Míg a szemantikai hiba esetén a fordító nem kapja el a hibát, de a program nem megfelelően fog működni.

Ezen kívül minden nyelvnek vannak Adattípusai is. Ezek lehetnek beépített, vagy a programozó által létrehozottak is. Ilyen típusok például az egész számok, a karakterek, a karakterláncok, a tömbök, a listák, a mutatók.

Léteznek nevesített konstansok is. Ezek is lehetnek beépítettek, vagy létrehozottak is. Ilyen konstans például a `pí`. Létrehozni pedig `c++` nyelvben a `#define` al míg `java`-ban a `final` utasítással tudunk.

A legalapvetőbb dolgok azonban a változók. Ezekben tároljuk a számunkra szükséges dolgokat. Egy változónak van típusa és értéke. A típusa lehet szám, karakter, karakterlánc, illetve logikai. Az értéke pedig a lehetséges típusok alapján lehet szám, karakter, karakterek sorozata, illetve igaz/hamis.

A programozási nyelvekben használunk még Kifejezéseket is. Ezek egyfajta műveletek. Egy kifejezésnek három része van. A művelet bal szélén valamilyen változó áll, aminek szeretnénk egy értéket adni, középen egy műveleti jel, és jobb oldalt pedig vagy egy másik változó, vagy egy konkrét érték, ami a bal oldalt álló változónak az új értéke lesz.

A gépi kódot a fordító az utasítások alapján generálja. Ezek az utasítások a következők lehetnek : Érték-adó utasítás; Üres utasítás; Ugró utasítás; Elágaztató utasítások; Ciklusszervező utasítások; Hívó utasítás; Vezérlésátadó utasítások; I/O utasítások; Egyéb utasítások.

A ciklusokat is nagyon sokszor használják a programozók. Ezek segítségével a megadott parancsokat egy-más után többször is elvégzi a program. A ciklusokhoz tartoznak a Vezérlő utasítások, amik a következők: A Continue parancs esetén a ciklus jelenlegi lépésében a hátra lévő utasításokat nem hajtja végre, hanem a következő cikluslépésre ugrik. A Break parancs esetén a ciklus megáll, és nem fut tovább. A return parancs esetén leáll a ciklus és visszaadja az eredményt.

Az alprogramok, vagy másnéven függvények olyan programrészletek, amiket megírva később meg lehet hívni őket, és a megadott értékekből előállítanak egy eredményt. Az alprogramoknak van neve és argumentumai. A nevével hívjuk meg őket, az argumentumok pedig azok az értékek amikből a végeredmény áll elő.

A programokban a blokkok olyan programrészletek amik programrészletekben helyezkednek el. Ilyen például az if elágazás után a potenciálisan végrehajtandó utasítások.

## 10.2. Kernighan és Richie olvasónaplója

Ez a Rácz András nevű Debreceni Egyetemi hallgató által készített olvasónapló a BRIAN W. KERNIGHAN – DENNIS M. RITCHIE által írt A C programozási nyelv című könyvből.

Először is A vezérlési Szerkezetek a ciklusok, az elágazások

Az elágazásokba beletartozik az if, if-else, else-if, else, és a switch feltételvizsgálatok. Ezekkel értékeket tudunk vizsgálni, és ezek végrehajtatni a megfelelő utasítást, utasításokat végrehajtani. Az if, if-else, else-if, else kifejezésekknél az if után vizsgáljuk meg az értéket, majd jönnek az utasítások, és végül opcionálisan else-if vagy else. Bármennyi else-if lehet egymás után, azonban else csak egy vagy nulla. Viszont célszerű else-t is használni, mert általában kevés esély van arra, hogy minden esetet lefedünk szimplán else-if használatával.

Ezzel szemben a switch esetében megadjuk az értéket, majd tetszőleges darabszámú case használatával megnézzük, hogy az e az érték, ami nekünk kell, és ha igen akkor az aktuális case utasításait hajtja végre. Célszerű megjegyezni, switch-case használatánál a program minden esetben végigellenőrzi az összes case-t, ezért ha nem szeretnénk, hogy az összeset ellenőrizze, ha már talált egy egyezést, akkor használjuk a break utasítást.

A Ciklusok esetében beszélhetünk for, while, és do while ciklusokról.

A for esetében a programozó adja meg, hogy hányszor fusson le a ciklus. A while és a do while esetében pedig addig fut a ciklus, amíg egy feltétel nem teljesül. Éppen ezért vigyázni kell, nehogy véletlenül egy végtelen ciklus alakuljon ki. Fontos különböző még a while és a do while ciklusok között, hogy míg a while ciklus először ellenőrzi, hogy teljesült-e a feltétel, majd pedig lefuttatja az utasításokat, addig a do while ciklus először lefuttatja az utasításokat, majd pedig ellenőrzi, hogy teljesült-e már a feltétel.

A C nyelv alapvető adattípusai az int, a float, a double, a char, és a bool.

Az intek (integerek) egész számok amik lehetnek pozitívak és negatívak is. Az int mérete 4bájt, azaz 32bit

A float és a double típusú változókban valós, úgynevezett lebegőpontos számokat lehet tárolni. Ilyen például a 0.5. A különböző a két változó között azonban az, hogy, hogy míg a float mérete csak 4bájt, addig a double mérete 8bájt.

A char (character) típusú változóban meglepő módon egy karaktert lehet eltárolni. A char mérete 1bájt.

Az alapvető adattípusokon túl a C nyelvnek vannak Állandói is. Ilyen például a #define, amivel meg tudunk adni meg nem változtatható értékeket. Ezekre később hivatkozni tudunk. De ilyen állandók még az escape sorozatok, amiket az adatok kiíratásánál tudunk alkalmazni. Ilyen például a \n amivel egy új sort kezdünk.

## 10.3. Benedek Zoltán, Levendovszky Tihamér - Szoftverfejlesztés C++ nyelven olvasónaplója

A C++ egy objektum orientált programozási nyelv, ami egyben alacsonyabb szintű elemeket is támogat.

A C++-ban ha egy függvényt paraméterek nélkül hívunk meg, akkor az egyenértékű egy void paraméterrel. Aminek pont az a jelentése, hogy a függvénynek nincs paramétere. További különbség, hogy míg a C nyelvben egy függvényt csak a neve alapján azonosítunk, addig C++-ban egy függvényt a neve és az argumentumai határoznak meg. Ezáltal C++-ban előfordulhat két ugyan olyan nevű függvény különböző argumentumokkal. További változás, hogy C++-ba be lettek vezetve a referenciák, valamint egy új típus is bevezetésre került, ami nem más mint a bool. A bool egy logikai változó ami lehet igaz vagy hamis értékű.

A C++ bevezette az osztályokat, amik az adatok, és metódusok együttese. Innen ered az objektum orientáltság, mivel az objektum a egy darab osztály egy darab előfordulása. A metódus pedig az osztálynak egy olyan eleme, egy olyan függvény, ami az osztályba tartozó adatokat manipulálja.

A konstruktörök és destruktörök előredefiniált függvénymezők, amelyek kulcsszerepet játszanak a C++-nyelvben. Alepető probléma a programozásban az inicializálás. Mielőtt egy adatstruktúrát elkezdenénk használni, meg kell bizonyosodnunk arról, hogy megfelelő méretű tárterületet biztosítsunk a számára, és legyen kezdeti értéke. Ezt a problémát orvosolják a konstruktörök.

A destruktörök pedig egy konstruktör által már létrehozott objektum törlésében segítenek. Törlik a tartalmát, és felszabadítják az objektum által elfoglalt helyet. Ha mi nem hozunk létre destruktort, akkor a C++ a saját alapértelmezett változatát fogja használni.

Létezik még másoló konstruktör is, ami egy már meglévő objektumból hoz létre egy újat. Lefoglal a memoriában egy részletet, és annak az értékét felülírja a már létező objektum értékeivel.

A C++ nyelven az osztályok adattagjai előtt szerepelhet a static szó. Ez azt jelenti, hogy ezeket a tagokat az osztály objektumai megosztva használják.

Gyakran kerülünk olyan helyzetbe, hogy egy adott típusnak úgy kellene viselkednie, mint egy másiknak. Ekkor kell típuskonverziót alkalmazni. Ezt meg lehet tenni implicit és explicit módon is.

Implicit konverziót általában haonló típusokon lehet elvégezni. Ilyen például ha egy integer változó értékét szeretnénk átadni egy long típusú változónak.

```
int x = 5;
long y = x;
```

Mind a ketten egész szám típusok, viszont a long nagyobb méretű, ezért a konverzió gond nélkül megtörténik.

Ez a módszer explicit konverzió esetén nem biztos, hogy működni fog, és még adatvesztéssel is járhat. Ilyen például ha egy integer változó értékét szeretnénk átadni egy byte értékű változónak. A byte mérete kisebb mint az it, ezért a változó előtt kell lennie egy zárójelnek benne a típussal.

```
int x = 300;
byte y = (byte)x;
```

Itt például az y értéke 44 lesz, mert a 300-at kilenc biten kell felírni, azonban a byte csak 8 bitet tárol, ezért az x-nek csak az első 8 bitjét fogja eltárolni.

C++ ban lehetőségünk van függvény sablonok és osztály sablonok létrehozására is, ezek a templatek. A template argumentumai eltérnek a hagyományos argumentumoktól. Egyrészt már a fordítás közben kiértékelődnek, ezért a futás közben már konstansok. Éppen e miatt az argumentumok típusok is lehetnek, nem csak értékek.

## **III. rész**

# **Második felvonás**

**Bátf41 Haxor Stream**

A feladatokkal kapcsolatos élő adásokat sugároz a <https://www.twitch.tv/nbatfai> csatorna, melynek permanens archívuma a <https://www.youtube.com/c/nbatfai> csatornán található.

## 11. fejezet

# Helló, Berners-Lee!

### 11.1. Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II és Benedek Zoltán, Levendovszky Tíhamér Szoftverfejlesztés C++ nyelven olvasónapló

1. hét: Az objektumorientált paradigmával kapcsolatos alapfogalmak. Osztály, objektum, példányosítás

Először értsük meg, hogy mi is az objektum, és hogy keletkezik, és mi játszik fontos szerepet a működésben. Az objektum a java programozási nyelv alapvető eleme, éppen ezért a Java egy objektum orientált programozási nyelv. Az objektum a valódi világ egy elemének a rá jellemző tulajdonságai és viselkedései által modellezett eleme. Az objektumokkal általában valamilyen feladatot szeretnénk megoldani. Egy objektum tulajdonságokból(változók), és viselkedésekkel(metódusok) áll. A változókkal írhatjuk le az adott objektum állapotát, minden egyednek saját készlete van a változókból, éppen ezért ezeket példányváltozóknak nevezzük. A metódus nagyrészt hasonlít egy függvényre. Azaz utasításokat hajt végre, kaphat paramétereket, és egy értékkal tér vissza. Az osztályok pedig az azonos típusú objektumok modelljét írják le. A program a működése során példányosítja az osztályokat, azaz konkrét objektumokat hoz létre, vagyis amikor egy objektumot létrehozunk, azt valójában egy osztályból hozzuk létre. Amikor egy új egyedet szeretnénk létrehozni, akkor azt a konstruktur fogja felépíteni. Az előzőekben említtettem, hogy a változókat a metódusok kezelik. Azonban alaphelyzetben ez nem igaz. Ha csak úgy megírunk egy osztályt, akkor annak a változóihoz kívülről is hozzá lehet férfi, a metódosuk figyelembe vétele nélkül. Ez pedig nem jó dolog. Ahhoz, hogy egy objektum biztonságos legyen, privátá kell tennünk a változóit, ezáltal csak az adott objektum férhet hozzá a saját változóihoz közvetlenül. minden más csak a metódusain keresztül férhet hozzájuk. Az, hogy a változók és metódusok egy helyen vannak tárolva az osztályokban (egységbe zárás), valamint az, hogy egy objektum változóihoz csak ellenőrzött körülmények között lehet hozzáérni (egységbe zárás) eggyüttesen az adatbsztraktciót, azaz az objektumorientált paradigmára egyik alapját alkotják.

C++-ban az osztályok példányokat tárolnak, például különböző bankszámlák. Ezen kívül megtalálhatóak a példányok tulajdonságai is, mintpéldául a számlán lévő egyenleg, illetve a számlán végzett műveletek (pént betétele, felvételle). Az ilyen egyedekeket nevezzük objektumoknak. Fontos, hogy egy objektum tulajdonságaihoz csak a műveletein keresztül lehessen hozzáérni. Ha például egy bankszámla egyenlegét csak úgy át lehetne írni, az nem lenne jó hatással a társadalomra. Éppen ezért az objektumok tulajdonságait és műveleteit egységbe kell zárni, illetve biztosítani kell, hogy az objektum tulajdonságaihoz a program többi része ne tudjon hozzáérni. Ezt hívjuk adatrejtésnek. Ha van egy madár, illetve egy papagáj osztályunk, akkor a két osztály között egyfajta kapcsolat van, mivel a papagáj maga is egy madár. Ezt nevezzük specializációknak. A

madár általánosabb fogalom mint a papagáj. Éppen ezért a speciálisabb osztály rendelkezik az általánosabb osztály tulajdonságaival és műveleteivel, másszóval örökli őket. Szóval egy papagájt bármikor kezelhetünk madárként. Ez a három fogalom (adatrejtés, specializáció, öröklődés) alkotja az objektumorientált programozás alapelveit. Azonban a C++-ban (sok OOP, köztük a JAVA nyelvvel ellentétben) megtalálható a típusmogatás is. Ez azt jelenti, hogy az osztályok ugyan úgy működhetnek, mint a beépített típusok.

2. hét: Öröklődés, osztályhierarchia. Polimorfizmus, metódustúlterhelés. Hatáskörkezelés. A bezárási eszközrendszer, láthatósági szintek. Absztrakt osztályok és interfések.

A legegyszerűbb példa az öröklődésre az az, amikor egy osztály egy már meglévő osztály kiterjesztéseként definiálunk. Ez lehet új műveletek, vagy új változók bevezetése is, maga az osztály pedig lehet public, illetve nem public is. Az eredeti osztályt szülőosztálynak, a kiterjesztettet pedig gyermek osztálynak nevezzük. A gyermekosztály megörökli a szülőosztály változót és metódusait, ha a láthatósági szintje az adott változónak/metódusnak lehetővé teszi azt. A láthatósági szint lehet public, ami azt jelenti, hogy az adott változót vagy metódust nem csak a gyermek osztályok, hanem bármely másik osztály objektumai is elérik. A láthatósági szint lehet protected is. Ebben az esetben már csak az adott osztályból kiterjesztett gyermekosztályok érik el őket. A harmadik lehetőség pedig a private, amikor pedig csak a szülőosztály objektumai tudják elérni az adott változót/metódust. Azonban a gyermek osztály nincs csupán ezekre korlátozva, vagyis a gyermekosztályoknak lehetnek saját változói, és metódusai, illetve fel is tudják írni a szülőtől örökölt metódusokat.

Mivel a gyermek osztály a szülő osztály minden változójával és metódusával rendelkezik, ezért használhatóak minden olyan esetben, amikor a szülő használható. Egy változó pedig nem csak a deklarált típusú, hanem egy leszármazott objektumra is hivatkozhat. Ezt polimorfizmusnak, azaz többlelképességnak nevezzük.

Ha egy kiterjesztett osztálybeli metódusnak ugyan az a szignatúrája, és visszatérési értéke, mint a szülőosztály metódusának, akkor a leszármazott osztály felülírja a szülőosztály metódusát. Ez lehetővé teszi, hogy egy osztály örökljön egy olyan szülőosztálytól, aminek hasonló a viselkedése, majd szükség esetén ezen változtasson. A felülíró metódus neve, paramétereinek a száma és típusa megegyezik a felülírt metódussal.

Alapértelmezetten egy újonnan létrehozott osztálynak az Object nevű osztály lesz az őse. Ez áll a Java osztályhierarchia csúcsán. Ebből kiindulva lehet ábrázolni az osztályok hierarchiáját egy fa adatszerkezetben.

Lehetőség van Absztrakt osztályokat is létrehozni az *abstract* módosítóval. Az ilyen osztályok tartalmazhatnak absztrakt, azaz törzs nélküli metódusokat, amiket szintén az *abstract* módosítóval kell jelölni. Az ilyen osztályok nem példányosíthatóak, mivel a példányokra nem lenne értelmezve minden metódus. Ennek ellenére van értelme absztrakt típusú változókat és paramétereket deklarálni, mivel az ilyen változók az adott absztrakt osztály bármely leszármazottjának példányára hivatkozhatnak.

C++ ban az öröklés során egy osztály specializált változatait hozzuk létre, amelyek öröklik a szülőosztály jellemzőit és viselkedését. Ezeket az osztályokat alosztályoknak nevezzük. Az alosztályok megváltoztathatók az öröklött tulajdonságokat, és új metódusokat is adhatunk hozzá (a Java nyelvhez hasonlóan). Az öröklődés fajtája lehet egyszeres öröklés, és többszörös öröklés is (az utóbbi a Java nyelvben csak az Absztrakt osztályok használatánál lehetséges). Az egyszeres öröklés esetén minden származtatott osztály pontosan egy közvetlen szülőosztály tagjait öröklíti, míg a többszörös öröklődés során a származtatott osztály több közvetlen szülőosztály tagjait öröklíti. Például létrehozhatunk egymástól független autó és hajó osztályokat, majd pedig ezekből örökléssel definiálhatunk egy kétéltű osztályt, ami egyaránt rendelkezik az autó és a hajó jellemzőivel és viselkedésével is. Ebben különbözik a C++ nagyon sok magasszintű programozási nyelvtől (Java, C# ...), mivel azok csak az egyszeres öröklést támogatják.

3. hét: Modellező eszközök és nyelvek. AZ UML és az UML osztálydiagramja.

Az UML, azaz Unified Modeling Language, vagy magyarul egységesített modellezőnyelv segítségével fejlesztési modelleket lehet szemléltetni. Egy integrált diagramkészletből áll, amelyet a szoftverfejlesztők számára fejlesztettek ki a programok megjelenítésére, felépítésére és dokumentálására. Az objektumorientált szoftverfejlesztési folyamat nagyon fontos részre. Többnyire grafikus jelöléseket használ a projektek tervezésére. Rengeteg diagram, azaz modell van hozzá. Az UML használható bármelyik ma ismert programozási nyelvvel, mivel azoktól független absztrakciós szinten fogalmazza meg a rendszer modelljét. Maga az UML egy grafikus modellező nyelv, azaz a diagramok téglalapokból, vonalakból, ikonokból, és szövegből állnak.

A Class diagram egy központi modellezési technika, amely szinte minden objektum-orientált módszert átfut. A rendszerben található osztályokat, interfészeket, egyéb típusokat, és a közöttük lévő kapcsolatot irja le. Rendkívül jól lehet osztálydiagrammal megmutatni egy program felépítését, valamint az osztályok között fennálló asszociációt, aggregációt, és kompozíciót. Ha egy modellben két osztálynak kommunikálnia kell egymással, akkor szükségünk van egy kapcsolatra a két osztály között. Ezt a kapcsolatot reprezentálja az asszociáció. Az asszociációt egy a két osztály között lévő vonal, valamit az azon lévő irányt mutató nyil(ak) jelöli(k). Ha a vonal minden oldalán van nyíl, akkor az asszociáció kétirányú. Az aggregáció és kompozíció az asszociáció részhalmazai, vagyis az asszociáció különleges esetei. Mind a két esetben egy osztály objektuma "birtokol" egy másik osztály másik objektumát, de van a kettő között egy kis különbség. Az aggregáció egy olyan kapcsolatot jelent, amiben a gyerek a szülőtől függetlenül létezhet. Például ha van tanóra, ami a szülőosztály, és tanuló, ami a gyerekosztály. Ha töröljük a tanórát, attól a tanulók még léteznek. Ezzel szemben a kompozíció esetében egy olyan kapcsolatról van szó, amiben a gyerek nem létezhet a szülő nélkül. Például ha van egy ház szülőosztályunk, és egy szoba gyerekosztályunk. A szoba nem létezhet a ház nélkül. Az aggregációt és a kompozíciót is vonal+rombusz kombinációval lehet jelölni, azonban az aggregációnál a rombusz üres, a kompozíciót pedig.

4. hét: Objektumorientált programozási nyelvek programnyelvi elemei: karakterkészlet, lexikális egységek, kifejezések, utasítások.

Java-ban a charset egy osztályt jelöl, ami tizenhat bites Unicode kód egységek és a bajt szekvenciák megnevezett leképezése. Ez az osztály meghatározza a dekóderek és kódolók létrehozásának, valamint a karakterkészlethez társított különféle nevek lekérésének módszereit. Ez az osztály immutable, azaz állandó. Ennek az osztálynak vannak statikus metódusai, amiknek a segítségével le lehet tesztelni, hogy egy adott karakterkészlet támogatott-e. Ezen kívül lehet ezeket a metódusokat karakterkészletek név szerinti keresésére is használni. A Java platform minden implementációjának támogatnia kell a következő karakterkészleteket: US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16. A JVM minden példányának van egy alapértelmezett karakterkészlete, ami a JVM bootolásakor kerül meghatározásra, és általában az operációs rendszer karakterkészletétől függ.

A kifejezés egy olyan változókból, operátorokból, és metódushívásokból álló programrészlet, amelynek egyetlen érték lesz az eredménye. A kifejezés által visszaadott érték típusa a kifejezésben használt elemektől függ. Például az int a = 0; kifejezés int értékkal tér vissza. De egy kifejezés más típusú értékkel is visszatérhet, például logikai, vagy string. A Java programozási nyelv lehetővé teszi összetett kifejezések létrehozását különféle kisebb kifejezésekkel, feltéve, hogy a kifejezés egyik részének a típusa megegyezik a kifejezés másik részének a típusával. De ilyen esetben figyelni kell a számolási sorrendre is intek esetében. Ha például csak szorzás szerepel az összetett kifejezésben, akkor teljesen mindegy a sorrend, de ha van összeadás és szorzás is, akkor előbb a szorzás lesz elvégezve. Ha azt szeretnénk, hogy előbb az összeadás hajtódjon végre, akkor zárójeleket kell használni. Az utasítások nagyjából megegyeznek a természetes nyelvek mondataival. Egy utasítás egy teljes végrehajtási egységet képez.

5. és 6. hét: Objektumorientált programozási nyelvek típusrendszer, Típusok tagjai: mezők, (nevesített)

konstansok, tulajdonságok, metódusok, események, operátorok, indexelők, konstruktörök, destruktörök, beágyazott típusok.

Java nyelvben négy fontos típus létezik. Ezek a primitív típusok, amelyek a primitív értékeknek megfelelő típusok, például int, short, long, byte, char, float, double, boolean. Null típusok, tömb típusok, valamint az osztály típusok.

A primitív típusok olyan primitív értékeket tárolnak, mint az egész számok, lebegőpontos számok, karakterek és logikai értékek. A négy egész szám típus abban különbözik egymástól, hogy milyen nagy az a szám, amit maximum el tudnak tárolni. Hasonlóképpen a lebegőpontos számok két típusa, között is az a különbség, hogy milyen nagy számot tudnak eltárolni. Egy kissemél típusú változó értékét gond nélkül bele lehet helyezni egy nagyobb típusú változóba, azonban ha egy nagyobb típusú szám értékét szeretnénk behelyezni egy kissemél típusú változóba, akkor át kell alakítanunk azt. A tömb olyan adatstruktúra, amely állandó hozzáférést tesz lehetővé a tömb elemeihez. A tömbök rögzített méretűek. Az osztály típusok pedig az objektumok típusai.

Java-ban egy mező egy osztályon belüli változó, amit a következő szintaxissal lehet létrehozni: *[elérés] [static] [final] típus név [= kezdőérték]*. A szögletes zárójel azt jelenti, hogy az adott rész opcionális. Először is egy mezőhöz deklarálhatunk hozzáférési módosítót. A hozzáférés-módosító meghatározza, hogy mely osztályok objektumai férhetnek hozzá a mezőhöz. Másodszor pedig, a mező típusát kell megadni. Ez lehet pl int, string, boolean stb. Harmadszor, a Java mező statikusnak nyilvánítható. A Java-ban a statikus mezők az osztályhoz tartoznak, nem pedig az osztály példányai. Így bármely osztály minden objektuma ugyanazt a statikus mezőváltozót fogja elérni. A nem statikus mező értéke az osztály minden objektumánál eltérő lehet. Negyedszer, a Java mező konstansnak is nyilvánítható. A konstans mező értékét nem lehet megváltoztatni. Ötödször, a Java mezőnek nevet kell adni. Végül pedig adhatunk a mezőnek kezdőértéket. A nevesített konstans egy azonosító, amely állandó értéket képvisel. A változó értéke a program végrehajtása során változhat, de a nevesített konstansok állandó adatokat képviselnek, amelyek soha nem változnak. Ilyen például pi. Két féle állandó létezik java-ban, final és static final. Ha csak simán final, akkor az értéknek csak az adott osztálypéldányban kell állandónak lennie, ha viszont static final, akkor az értéknek minden osztálypéldányban ugyan annak kell lennie.

A Properties osztály tulajdonságok halmazát reprezentálja. Ezek a tulajdonságok elmenthetőek streambe, vagy pedig betölthetőek egy streamből. A Properties a Hashtable osztály alosztálya. Egy értékklista fenn-tartására szolgál, amelyben a kulcs és az érték is egy string. A Properties osztály egyik hasznos képessége, hogy meghatározhat egy alapértelmezett tulajdonságot, amely visszaadódik, ha egy adott kulcsnak nincs érték társítva. A metódusok olyan utasítások összessége, amelyek egy konkrét feladatot hajtanak végre, és az eredményt visszaadják annak, ami meghívta a metódust. De az is lehet, hogy csak végrehajtanak egy feladatot, és nem térnek vissza semmivel. A metódusok lehetővé teszik a kód újra felhasználását a kód újraírása nélkül. Java-ban, a C++-al ellentétben minden metódusnak egy osztály részének kell lennie. A konstruktör objektumokat inicializál, amikor azok létrejönnek. Ugyan az a neve, mint az osztálynak, a szintaktikája pedig hasonló egy metódushoz. Jellemzően konstruktorkkal adjuk meg az objektumok kezdőértékeit. Java-ban minden osztály rendelkezik konstruktőrrel, attól függetlenül, hogy a programozó definiált-e egyet vagy nem, mert ha nem, akkor a Java automatikusan megad egy alapértelmezett konstruktort, amely az összes tagváltozó értékét null-ra inicializálja.

7. és 8. hét: Interfészök. Kollekciók. és Funkcionális nyelvi elemek. Lambda kifejezések.

Pont úgy mint az osztályoknak, az interfészöknek lehetnek metódusai és változói. De az interfészben deklarált metódusok alapértelmezés szerint absztraktak. Az interfész meghatározzák, hogy az osztálynak mit kell tennie, és nem pedig azt hogy hogyan. Az interfész meghatározza azokat a metódusokat, amelyeket az osztálynak végre kell hajtania. Interface deklarálásához bármilyen meglepő, az interface kulcsszót

kell használni. Ez a teljes absztrakció biztosítására szolgál. Ez azt jelenti, hogy az interfészben az összes metódus üres testtel kerül deklarálásra, hozzáérhetőségiuk public, és alapértelmezés szerint minden mező publikus, statikus és final. Az interfész megvalósító osztálynak végre kell hajtania az interfészben deklarált összes metódust. A Collection az egyedi objektumok csoportja, amely egyetlen egységként van ábrázolva. A java biztosít egy Collection Framework-ök, amely meghatároz több osztályt és interfész azért, hogy egy objektumcsoportot egyetlen egységként képviseljen. A Collection interfész és a Map interfész a két gyökér interfésze a Java Collection osztályának. A Lambda kifejezések, amelyek Java 8-tól érhetők el, alapvetően a funkcionális interfések példáit fejezik ki (egy absztrakt módszerrel rendelkező interfész funkcionális interfésznek hívunk. Példa erre a java.lang.Runnable). A lambda kifejezések megvalósítják az egyetlen elvont funkciót, éppen ezért megvalósítják a funkcionális interféseket. A lambda kifejezések lehetővé teszi a funkcionalitás metódus argumentumként kezelését, vagy a kód adatként kezelését. A lambda kifejezést úgy lehet átadni, mintha objektum lenne, és igény szerint végrehajtható. A lambda kifejezés teste tartalmazhat nulla, egy vagy több állítást is. Ha egynél több állítás van, akkor kapcsos zárójelbe kell őket zárni.

#### 9. és 11. héten: Adatfolyamok kezelése, streamek és állománykezelés. Szerizáláció.

A Java 8-ban bevezetett Stream API az objektumgyűjtemények feldolgozására szolgál. A stream nem adatszerkezet, hanem egy Collection-ból, tömbből vagy I/O csatornából származó adatot veszi be. A streamek nem változtatják meg az eredeti adatszerkezetet, csak az eredményt mondják meg a megvalósított metódusok alapján. Támogat különféle műveleteket az ilyen elemekre vonatkozó számítások elvégzéséhez. A stream műveletek közbenő vagy terminális műveletek. A közbenő műveletek steam-mel térnek vissza, ezért több is követheti egymást pontosvessző nélkül. A terminális műveletek vagy voidok, vagy nem stream adatot eredményeznek. A java.io csomag szinte minden olyan osztályt tartalmaz, amelyre esetleg szükség lehet a Java bemenet és kimenet (I/O) megvalósításához. Kétféle stream létezik, az InPutStream, ami egy forrásból származó adatok olvasására szolgál, és az OutPutStream, ami pedig az adatot egy célhelyre írja. A java byte streameket 8 bit méretű bajtok írására és olvasására lehet használni. Sok ilyen van, de a leggyakrabban használtak a FileInputStream és FileOutputStream. Az össze programozási nyelv támogatja azt a szabványos i/o-t ahol a felhasználó a konzolon keresztül tud adatot megkapni és megadni. Ilyen Java-ban a Standard Input, ami olvassa a felhasználó által megadott adatot, a Standard Output, ami a program által előállított adatot adja át a felhasználónak, és a Standard Error, ami a program által előállított hibák adatait közli a felhasználónak. A Java biztosít egy olyan mechanizmust, amelyet objektum-szerizálációnak nevezünk, ahol egy objektum egy olyan bajtsorozatként ábrázolható, amely tartalmazza az objektum adatait, valamint az objektum típusára és az objektumban tárolt adatok típusára vonatkozó információkat.

#### 10. héten és 12. héten: Kivételkezelés és Reflexió. A fordítást és a kódgenerálást támogató nyelvi elemek.

A Kivétel egy nem kívánt vagy váratlan esemény, amely egy program végrehajtásakor fordul elő, azaz futási időben, és megzavarja a program utasításainak normál folyamatát. Egy hiba olyan komoly problémát jelent, amelyet egy ésszerű alkalmazásnak nem szabad catchelnie. A kivétel ezzel szemben olyan feltételeket jelent, amelyeket egy ésszerű alkalmazás megpróbálhat catchelni. Az összes kivétel és hiba típus a Throwable osztály alosztálya, amely a hierarchia alaposztálya. Az egyik ágnak az Exception a feje. Ez az osztály kivételes körülmenye között használatos, amelyeket a felhasználói programoknak el kellene kapniuk, például NullPointerException. Másik ág, azaz a Hiba, JVM használatával jelzi azokat a hibákat, amelyek a JRE-vel kapcsolatosak, pl: StackOverflowError. A Java kivételkezelés öt kulcsszón keresztül történik: try, catch, throw, throws, és finally. Azok a kód részletek amelyek exception-t okozhatnak, egy try blokkba kerülnek. Ha exception történik a try blokkban, akkor azt eldobja (throw), amit a catch blokk elkap, és kezeli. Az a kód pedig, amelyet mindenkor végre kell hajtani exception esetén is, a final-be kerül.

A reflexió egy olyan API, amelyet a metódusok, osztályok, interfések viselkedésének megvizsgálására

vagy módosítására használnak futási időben. A reflexióhoz szükséges osztályokat a `java.lang.reflect` csomag tartalmazza. Reflexión keresztül metódusokat lehet futtatni, függetlenül a metódusok hozzáférési szintjétől. A reflexió segítségével információkat szerezhetünk az Osztályról, a `getClass()` metódussal, ez a metódus annak az osztálynak a nevét adja vissza, amelyhez az adott objektum tartozik. Információt szerezhetünk a konstruktorról, a `getConstructors()` metódussal. Ezzel annak az osztálynak a nyilvános konstruktoraikat kapjuk meg, amelyhez az adott objektum tartozik. Valamint lekérhetjük még a metódusokat is, a `getMethods()` függvénytel, amely annak az osztálynak a publikus metódusait adja vissza, amelyhez az adott objektum tartozik. Az annotációk kiegészítő információkkal szolgálnak a programról. Az annotációk @ karakterrel kezdődnek, és nem változtatják meg a program működését, de megváltoztathatják azt, hogy hogyan kezeli a fordító a programot. Három féle annotáció létezik. Az első a Jelölő Annotációk, amelyek deklarációt jelölnek, a második az eggyértékű Annotációk, a harmadik pedig a Teljes Annotációk.

13. és 14. het: Multiparadigmás nyelvek és Programozás multiparadigmás nyelveken.

A paradigműt úgy is nevezhetnénk, mint egy módszer valamilyen probléma megoldására vagy valamilyen feladat elvégzésére. A programozási paradigma pedig a probléma megoldásának megközelítése valamilyen programozási nyelv használatával. Sok féle paradigma létezik a különböző igények kielégítésére. Én most az Imperatív programozási paradigmáról fogok írni, amely az egyik legrégebbi programozási paradigma. Szoros kapcsolatban áll a gép architektúrájával. Úgy működik, hogy a program állapotát hozzárendelési utasításokkal változtatja meg. Az állapot megváltoztatásával lépésről lépésre hajtja végre a feladatot. A fő hangsúly a cél elérésének módja. Az előnyei, hogy nagyon egyszerű implementálni, valamint ciklusokat, változókat, stb. tartalmaz. Hátránya, hogy összetett problémákat nem lehet vele megoldani, valamint a párhuzamos programozás nem lehetséges vele. Az imperatív programozást három kategóriába lehet sorolni: eljárási, OOP és párhuzamos. Ezek a következők: A Procedurális programozási paradigma az eljárásokat hangsúlyozza. Az Objektumorientált programozás alapján készült program a kommunikációra szánt osztályok és objektumok gyűjteményeként van írva. A legkisebb és alapvető entitás az objektum, és mindenféle számítást csak az objektumokon végeznek. Nagyobb hangsúly van az adatokon az eljárás helyett. Szinte minden valós problémát képes kezelni, amelyek manapság forgatókönyvbe kerülnek. A Párhuzamos feldolgozási megközelítés jelentése a program utasításainak feldolgozása több processzor közötti elosztással. A párhuzamos feldolgozási rendszer számos processzort tartalmaz, azzal a céllal, hogy a feladatokat rövidebb idő alatt végezze el azok megosztásával.

## 11.2. Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobil-programozásba

A Python programozási nyelvet Guido van Rossum alkotta meg 1990-ben. Maga a python egy magas szintű, dinamikus, objektumorientált, és platformfüggetlen programozási nyelv. Leginkább egyszerű alkalmazások készítésére használatos. Viszonylag könnyen meg lehet tanulni a használatát, ezért hamar el lehet vele érni látványos eredményeket. Különlegessége más nyelvekkel szemben (pl. C, C++, Java), hogy nincs szükség a programkód fordítására. Elegendő egy forrás fájlt megírni, és az automatikusan fut is. A python programok általában sokkal rövidebbek, mint ugyanazon programok C++ vagy Java nyelven. Ennek több oka is van. Egyszer az adattípusai lehetővé teszik, hogy összetett kifejezéseket írunk le rövid állításokban. Másrészt nincs szükség a változók definiálására. És végül, a (szarkazmus) legkedveltebb ok, hogy a python nyelv nem használ se zárójeleket, se pontossesszóket. Ezek helyett a kód csoportosítása új sorral és tabulátorral történik. Pythonban egy programblokk végét egy kisebb behúzású sor jelzi, az utasítások pedig a sorok végéig tartanak. Éppen ezért nincs szükség pontossesszóra. Ha viszont egy utasítás nem fér el egy sorban, akkor az adott sor végére egy \ jelet kell tenni, a megjegyzések pedig kettőskereszt jellel tujuk jelezni.

A python nyelvben a változók az objektumokra mutató referenciák. Egy változó hozzárendelését a del kulcsszóval tudjuk törölni, ha pedig egy objektumra már egy változó se mutat, akkor a garbage collector fogja törölni az adott objektumot. Érdekesség ezzel kapcsolatban, hogy a változóknak nem kell konkrét típusat adnunk, mivel kitalálja, hogy mire gondolunk. Az adattípusok a következők lehetnek: számok, sztringek, ennesek, listák, és szótárak. A számok lehetnek egészek, komplexek, és lebegőpontosak is, a sztringeket pedig idézőjelek, illetve aposztrófok közé írva lehet megadnunk.

Maguk a változók lehetnek globálisak vagy lokálisak. Alapvetően a lokális az alapértelmezett, ezért ha azt szeretnénk, hogy egy változó globális legyen, akkor azt a változót a függvény elején kell felvenni, illetve elérni a global kulcsszót. A különböző típusok közötti konverziók támogatottak, ha van értelmük. Például int, long, float, illetve complex típusok közötti konverzió. De sztringekből is képezhetünk számot. Ehhez csak a használt számrendszer kell megadni, pl: int. Ezeknek a változóknak a kiiratását a print függvényel lehet megoldani. Ha több változó értékét szeretnénk kiiratni, akkor vesszővel kell elválasztani őket egymástól. Ezeken kívül a python nyelvben ugyanúgy elérhetők az elágazások, illetve a ciklusok is, mint más magasszintű programozási nyelvekben. A for, illetve a while ciklus is elérhető, azokon pedig a break, illetve a continue utasítások is használhatóak. Léteznek címkek, amiket a label kulcsszóval kell elhelyezni a kódban, majd pedig a kód más részeiről a goto utasítás használatával a labelhez ugorhatunk.

Python nyelven a függvényeket a *def* kulcsszóval lehet definiálni. A függvényekre úgy is lehet tekinteni, mint értékekre, mivel továbbadhatóak más függvényeknek, és objektumkonstruktornak is. Ettől függetlenül a függvényeknek vannak paraméterei, amelyeknek adhatunk alapértelmezett értéket is. A legtöbb paraméter érték szerint adódik át, ezalól kivételek a mutable típusok, amelyeknek a függvényben történő megvalósítása hatással van az eredeti objektumra is. A függvény hívásánál a paraméterek úgy követik egymást, mint a függvény definíciójában. Emellett van lehetőség közvetlenül az egyes konkrét argumentumoknak értéket adni a függvény hívásakor, ha a zárójelben elé írjuk a változó nevét és egy egyenlőségejét. A függvényeknek egy visszatérési értékük van.

A Python nyelvben -más nyelvekhez hasonlóan- létrehozhatunk osztályokat, és ezekből példányosíthatunk objektumokat. Az osztályok tartalmazhatnak metódusokat, amiket akár örökölhetnek is más osztályokból. Az osztály metódusait ugyanúgy lehet definiálni, mint a globális függvényeket, azonban az első paraméterük a *self* kell hogy legyen, amelynek az értéke minden az az objektumpéldány lesz, amelyen a metódust meghívták. Ezen kívül az osztályoknak lehet egy speciális, konstruktor tulajdonságú metódusa, az *\_\_init\_\_*.

Léteznek különböző modulok, amelyeknek a célja a fejlesztés megkönnyebbítése. Ilyen például az *appuifw*, ami a felhasználói felület kialakítását, kezelését segíti. A *messaging* modul az SMS és MMS üzenetek kezelését segíti. A *sysinfo* a mobilkészülékekkel kapcsolatos információk lekérdezésére használható. A *camera* modullal lehet elvégezni minden, a készülék kamerájával kapcsolatos műveletet. Az *audio* modul pedig a hangfelvételek készítéséért és lejátszásáért felelős.

Más nyelvhez hasonlóan a Python nyelvben is van lehetőség a kivételkezelésre a *try*, *except* és opcionálisan egy *else* utasítással. A try kulcsszó után szerepel az a kódblokk, amelyben a kivétel előállhat. Ha bekövetkezik a hiba, akkor az except részre ugrik a program, és az ott lévő utasításokat hajtja végre.

## 12. fejezet

# Helló, Arroway!

### 12.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! C++ ban

Megoldás forrása: [Java](#) [C++](#)

Ebben a feladatban a prog1-en már tárgyalt Polárgenerátor megírása volt a feladat. A lényege ennek a programnak az, hogy legelőször generál két értéket. Az egyik értéket eltárolja, a másikat pedig visszaadja. Majd amikor következőnek megint generálna, akkor először megnézi, hogy van-e már tárolt érték. Ha van akkor azt a tárolt értéket adja vissza, ha viszont nincs, akkor generál két értéket, amiből az egyiket eltárolja, a másikat pedig visszaadja. Azt, hogy van-e tárolt érték, egy boolean változóban tartja nyílván.

A program a PolárGenerátor osztálytalálkozásban kezdődik:

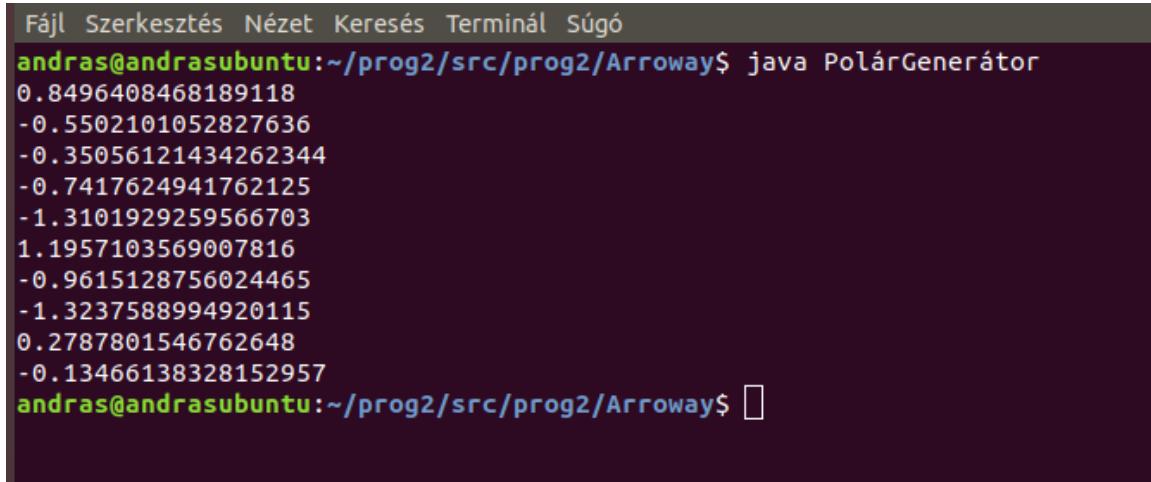
```
public class PolárGenerátor {  
    boolean nincsTárolt = true;  
    double tárolt;  
    public PolárGenerátor() {  
        nincsTárolt = true;  
    }
```

Itt kezdőértéknek meg van adva, hogy nincs tárolt érték, valamint egy double változó a majdani tárolt értéknek. Ezek után következik a következő nevű függvény, ami az érdemi munkát végzi. Ha már van tárolt értékünk, akkor a nincsTárolt változó értékét az ellenkezőjére változtatja, és visszaadja a tárolt értéket. Ha viszont nincs, akkor egy do while ciklusban először is az u1 és u2 változókhöz két véletlenszerű értéket rendel, majd pedig a Vx változók értékét úgy határozza meg, hogy Ux-et megszorozza kettővel, és a kapott eredményből kivon 1-öt. Ezek után a w változónak az értéke v1 négyzetének, és v2 négyzetének az összege. Ez a ciklus addig fog futni, amíg w értéke nagyobb mint 1. Ha véget ért a ciklus akkor az újonnan deklarált r változó kezdőértékét úgy határozza meg, hogy -2-vel megszorozza w logaritmusát, majd azt elosztja w-vel, és a kapott értéknek a négyzetgyöke lesz az eredmény. Ezek után a nincsTárolt változó értékét az ellenkezőjére állítja. Az eltárolt érték a r és v2 szorzata lesz, a visszaadott érték pedig r és v1 szorzata.

```
public double következő() {
    if(nincsTárolt) {
        double u1, u2, v1, v2, w;
        do{
            u1 = Math.random();
            u2 = Math.random();
            v1 = 2* u1 -1;
            v2 = 2* u2 -1;
            w = v1 * v1 + v2 * v2;
        } while ( w > 1);
        double r = Math.sqrt((-2 * Math.log(w)) / w);
        tárolt = r * v2;
        nincsTárolt = !nincsTárolt;
        return r * v1;
    } else {
        nincsTárolt = !nincsTárolt;
        return tárolt;
    }
}
```

Végül pedig a main, ami létrehoz egy PolárGenerátor objektumok, és egy for ciklussal 10 alkalommal futtatja a függvényt, és az eredmény.

```
public static void main(String args[]){
    PolárGenerátor g = new PolárGenerátor();
    for ( int i = 0; i< 10; i++) {
        System.out.println(g.következő());
    }
}
```



A terminal window showing the execution of a Java program. The command `java PolárGenerátor` is run, followed by ten lines of floating-point numbers representing the generated points. The session ends with the user's name and the command `exit`.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ java PolárGenerátor
0.8496408468189118
-0.5502101052827636
-0.35056121434262344
-0.7417624941762125
-1.3101929259566703
1.1957103569007816
-0.9615128756024465
-1.3237588994920115
0.2787801546762648
-0.13466138328152957
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ exit
```

## 12.2. Homokozó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden más módon működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer,

referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Serverbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Arroway/LzwBinFa.java>

Tanulságok, tapasztalatok, magyarázat...

## 12.3. Gagyi

Az ismert formális

```
while (x <= t && x >= t && t != x);
```

tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására, hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás videó:

Megoldás forrása:<https://github.com/raczandras/progbook/blob/master/src/prog2/Arroway/Gagyi.java>

Ebben a Feladatban létrehoztunk két Integer objektumok x és t néven. Az egyik alkalommal minden Integer értékét 127-re, a másik alkalommal pedig 128-ra állítjuk. Majd következik egy while ciklus ami addig fut, amíg x kisebb vagy egyenlő t-vel és x nagyobb vagy egyenlő t-vel és t nem egyenlő x-el.

```
import java.util.Scanner;
public class Gagyi{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Álljon elő a program? igen ha igen, bármi más ha ←
            nem");
        Integer t;
        Integer x;
        if(sc.nextLine().equals("igen")){
            x = 127;
            t = 127;
        }
        else{
            t = 128;
            x = 128;
        }
        while (x <= t && x >= t && t != x);
    }
}
```

Ebben az az érdekes, hogy amikor x és t értéke 127, akkor leáll a ciklus, míg amikor 128, akkor pedig egy végtelen ciklust kapunk:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ java Gagyí
Álljon e le a program? igen ha igen, bármi más ha nem
igen
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ java Gagyí
Álljon e le a program? igen ha igen, bármi más ha nem
nem
□
```

A kérdés pedig az, hogy ez miért történik? Erre a választ a JDK Integer.java forrásában kaphatunk.

```
778     /*
779
780     private static class IntegerCache {
781         static final int low = -128;
782         static final int high;
783         static final Integer cache[];
784
785         static {
786             // high value may be configured by property
787             int h = 127;
788             String integerCacheHighPropValue =
789                 sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
790             if (integerCacheHighPropValue != null) {
791                 try {
792                     int i = parseInt(integerCacheHighPropValue);
793                     i = Math.max(i, 127);
794                     // Maximum array size is Integer.MAX_VALUE
795                     h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
796                 } catch( NumberFormatException nfe ) {
797                     // If the property cannot be parsed into an int, ignore it.
798                 }
799             }
800             high = h;
801
802             cache = new Integer[(high - low) + 1];
803             int j = low;
804             for(int k = 0; k < cache.length; k++)
805                 cache[k] = new Integer(j++);
806
807             // range [-128, 127] must be interned (JLS7 5.1.7)
808             assert IntegerCache.high >= 127;
809         }
810
811         private IntegerCache() {}
812     }
813
814     /**
815      * Returns an {@code Integer} instance representing the specified
816      * {@code int} value. If a new {@code Integer} instance is not
817      * required, this method should generally be used in preference to
818      * the constructor {@code (Integer(int))}, as this method is likely
819      * to yield significantly better space and time performance by
820      * caching frequently requested values.
821      *
822      * This method will always cache values in the range -128 to 127,
823      * inclusive, and may cache other values outside of this range.
824      *
825      * @param i an {@code int} value.
826      * @return an {@code Integer} instance representing {@code i}.
827      * @since 1.5
828      */
829     public static Integer valueOf(int i) {
830         if (i >= IntegerCache.low && i <= IntegerCache.high)
831             return IntegerCache.cache[i + (-IntegerCache.low)];
832         return new Integer(i);
833     }
834
835     /**
836      * The value of the {@code Integer}.
837      *
```

Vagyis a -128 és az alapértelmezetten 127 (de ez konfigurálható) közötti értékekre egy már létező pool-ból fogjuk megkapni a nekünk kellő objektumot. Ami azt jelenti, hogy az x és a t ugyan azt az objektumot fogja viszakapni, vagyis ugyan arra a memóriacímre fognak mutatni. Éppen ezért le fog állni a while ciklus az x!=t feltétel miatt. Ezzel szemben, ha az érték 128, akkor nem az előre elkészített poolból fogják megkapni az értéküknek megfelelő objektumot, hanem a *return new Integer(i);*-vel fognak értéket kapni.

Ez azt jelenti, hogy x-nek és t-nek két különböző című objektuma lesz. Ebben az esetben pedig már az `x!=t` feltétel is igaz lesz, aminek az eredménye pedig egy végtelen ciklus.

## 12.4. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-re leáll, ha nem követjük a Yoda conditions-t!

[https://en.wikipedia.org/wiki/Yoda\\_conditions](https://en.wikipedia.org/wiki/Yoda_conditions)

Megoldás videó:

Megoldás forrása: [Forrás](#)

A sokak által, köztük általam is tanult összehasonlítási módszer szerint az egyenlőségjel bal oldalára kell kerülnie a változónak minden esetben. Azonban ezzel van egy probléma. Ha annak a bizonyos változónak null az értéke, akkor a programunk le fog állni egy java.lang.NullPointerException-nel. Erre ad megoldást a Yoda conditions, aminek az a lényege, hogy az összehasonlítás bal oldalára írjuk az értéket, a jobb oldalára pedig a változót.

```
import java.util.Scanner;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        String hason = null;
        String legyen;
        System.out.println("Kapunk-e NullPointerException-t? I/N");
        for(;;) {
            legyen = sc.nextLine();

            if( legyen.equalsIgnoreCase("I") ) {
                if( hason.equals("abrákádábra") ) {
                    break;
                }
            }
            else if( legyen.equalsIgnoreCase("N") ) {
                if(!"abrákádábra".equals(hason) ) {
                    System.out.println("Nem Kaptunk.");
                    break;
                }
            }
            else{
                System.out.println("Nem Tudom értelmezni amit írtál. ↵
                    próbáld újra.");
            }
        }
    }
}
```

Ebben a példában a hason egy String aminek null az értéke. A felső elágazás során a program le fog állni a fent említett NullPointerException hibával, mivel a stringet egy null pointerhez hasonlítanánk, ami nem lehetséges. Ezzel szemben az alsó esetben szimplán csak egy hamis értéket fogunk kapni eredményként. És a végeredmény:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ java Yoda
Kapjunk-e NullPointerException-t? I/N
N
Nem Kaptunk.
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ java Yoda
Kapjunk-e NullPointerException-t? I/N
I
Exception in thread "main" java.lang.NullPointerException
        at Yoda.main(Yoda.java:15)
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ 
```

## 12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhabailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: [https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi\\_jegyei](https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei) (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Arroway/BBP.java>

Ebben a feladatban a BBP algoritmust kellett megírni. Maga a program a main-el kezdődik:

```
public static void main(String[] args) {
    int k = 6;
    System.out.println(magic(k));
}
```

Amiben a k változó értéke azt mondja meg, hogy pi-nek 10 a hanyadikon számjegyétől kezdve írjuk ki a következő pár számjegyed hexadecimális alakban. Jelen esetben k értéke 6, ami az jelenti, hogy pi-nek az első  $10^6$  számjegye utáni pár számjegyét fogjuk megkapni. Majd kiíratjuk a magic(k) nevű függvény eredményét.

Ezután következik a magic függvény, ami egy értékadással kezdődik:

```
double s1 = solve(Math.pow(10, k), 1);
double s4 = solve(Math.pow(10, k), 4);
double s5 = solve(Math.pow(10, k), 5);
double s6 = solve(Math.pow(10, k), 6);
```

Az s1, s4, s5, és s6 változóknak úgy adunk értéket, hogy meghívjuk a solve() függvényt, ami pedig majd a mod() függvényt fogja használni. A solve() függvény egyik paramétere az  $10^k$  lesz, a másik pedig egy szám. Ha megvan az értékadás, akkor ugyan ezeket a változókat ráeresztjük a cut() függvényre:

```
s1 = cut(s1);
s4 = cut(s4);
```

```
s5 = cut(s5);
s6 = cut(s6);
```

A `cut()` függvénynek annyi a feladata, hogy visszaadja a paraméterként kapott double változó nem egész részét. Ezt úgy csinálja, hogy ha az értéke negatív, akkor hozzáadja saját magához saját maga egész részét, ha viszont pozitív, akkor pedig kivonja.

```
public static double cut(double db) {
    if(db < 0) {
        return db - (int)db+1;
    }
    else {
        return db - (int)db;
    }
}
```

Ezek után létrehozzuk, és értéket adunk pi-nek, majd ennek az értéknek kiszámoljuk a nem egész részét, illetve létrehozzuk a hexadecimális jeleket, és a végeredményt is. Egy while ciklusban addig számoljuk pi értékét, amíg a nem egész részének az értéke nem egyenlő nullával. Ha nem egyenlő akkor pi értékét megszorozzuk 16-tal. Majd, ha az egész része pi-nek nagyobb vagy egyenlő mint 10, akkor a végeredményt koncatenáljuk az értéknek megfelelő Hexadecimális jellel. Egyébként pedig Szimplán csak a Stringgé alakított számjegyeket koncatenáljuk a végeredménnyel, majd pedig elvesszük pi-ból az egész részét, és kezdődik előről a ciklus. Legvégül pedig visszaadjuk a végeredményt.

```
double pi = 4*s1 -2*s4 - s5 -s6;
pi = cut(pi);
String[] hexa = {"A", "B", "C", "D", "E", "F"};
String result = "";
while(cut(pi) != 0) {
    pi = pi*16;
    if((int)pi >= 10) {
        result = result.concat(hexa[(int)pi - 10]);
    }
    else {
        result = result.concat(Integer.toString((int)pi));
    }
    pi = cut(pi);
}
return result;
```

A `solve()` függvény egy összeget számol. Egy for ciklus addig megy, amíg az első kapott paraméter, azaz d értéke nagyobb, vagy egyenlő i-vel. A cikluson belül minden egyes lépésnél hozzáadja az összeg értékéhez a `mod()` függvény által kiszámolt értéket.

```
public static double solve(double d, double num) {
    double sum = 0.0;
    for(int i = 0; i <= d; i++) {
        sum += mod(16, (d-i), 8*i+num) / (8*i + num);
    }
    return sum ;
}
```

Végül pedig a mod() függvény. Létrehoz két double változót t és r néven, és minden a kettőnek az 1 kezdőértéket adja. Majd egy while ciklus addig megy amíg t kisebb vagy egyenlő mint n. N az a második paramétere a függvénynek. A cikluson belül minden egyes iterációban t értékét megszorozza kettővel. Ezek után jön még egy while ciklus, ami break utasítással fog leállni. Végül pedig a függvény visszaadja r értékét.

```
public static double mod(double b, double n, double k) {  
    double t = 1;  
    double r = 1;  
    while(t <= n) {  
        t = t * 2;  
    }  
  
    while(true) {  
        if(n >= t) {  
            r = (b * r) % k;  
            n= n - t;  
        }  
        t = t / 2;  
        if(t >= 1) {  
            r = (r*r) % k;  
        }  
        else {  
            break;  
        }  
    }  
    return r;  
}
```

Az eredmény pedig:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ java BBP  
6C65E5308  
andras@andrasubuntu:~/prog2/src/prog2/Arroway$ █
```

## 13. fejezet

# Helló, Liskov!

### 13.1. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megséríti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_1.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf) (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Megoldás videó:

Megoldás forrása: [Java](#) [C++](#)

A liskov elv azt jelenti, hogy ha S altípusa T-nek, akkor minden olyan helyen ahol T-t felhasználjuk S-t is minden gond nélkül behelyettesíthetjük anélkül, hogy a programrész tulajdonságai megváltoznának. Vagyis ha S osztály T osztály leszármazottja, akkor S szabadon behelyettesíthető minden olyan helyre (paraméter, változó, stb...), ahol T típust várunk. Ezt kellett megsérteni c++ ban és java-ban is. Ehhez én egy madárpingvin szülőosztály-gyerekosztály kombinációt használtam.

C++:

```
#include <iostream>
using namespace std;

class Madar {
public:
    void repul() {
        cout << "Repül";
    }
};

class Sas : public Madar
{};

class Pingvin : public Madar
{}
```

És java:

```
static class Madar{  
    public void repul(){  
        System.out.println("Repülök");  
    }  
}  
  
static class Sas extends Madar{  
}  
  
static class Pingvin extends Madar{  
}
```

A programok minden esetben úgy kezdődnek, hogy létrehozzuk a szülőosztályt, ami a madár. Ennek az osztálynak van egy olyan metódusa, hogy `repul()` ami jelen esetben csak annyit csinál, hogy kiirja a konzolra azt, hogy Repül vagy Repülök. Majd jön két újabb osztály, amiket a Madárból származtatunk, vagyis ők is meg fogják kapni a `repul()` metódust. Az egyik osztály a Sas, ami tud repülni és még madár is, szóval itt nincs probléma. Azonban a másik osztály a Pingvin, ami igaz, hogy madár, de repülni nem tud.

C++:

```
int main ( int argc, char **argv )  
{  
    Madar madar;  
    madar.repul();  
    cout << " a madár\n";  
  
    Sas sas;  
    sas.repul();  
    cout << " a sas\n";  
  
    Pingvin pingvin;  
    pingvin.repul();  
    cout << " a pingvin. De a pingvin nem tud repülni, ezért sérült a ←  
        Liskov elv.\n";  
}
```

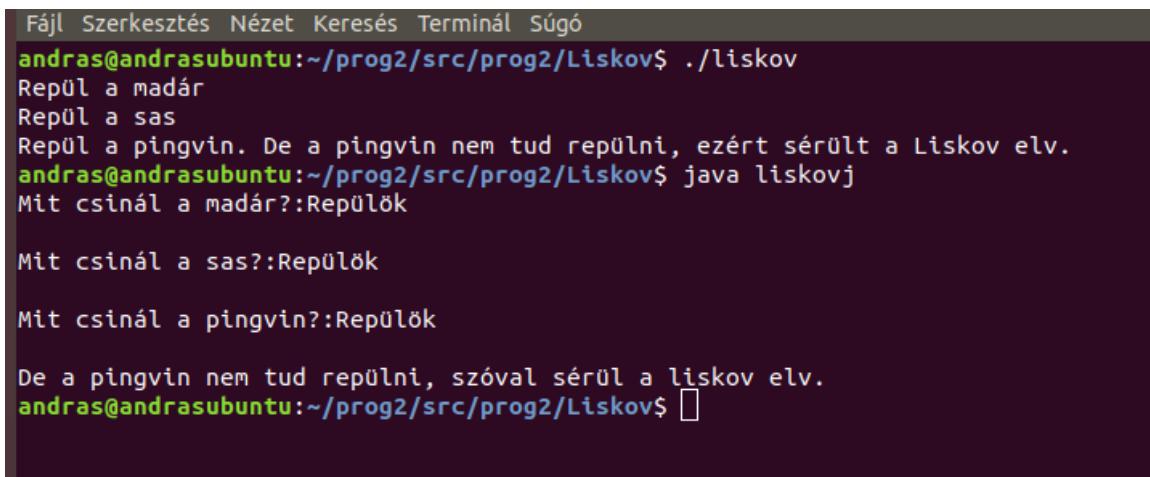
és Java:

```
public static void main(String args[]){  
    Madar madár = new Madar();  
    Sas sas = new Sas();  
    Pingvin pingvin = new Pingvin();  
  
    System.out.print("Mit csinál a madár? :");  
    madár.repul();
```

```
System.out.print("\nMit csinál a sas?:" );
sas.repul();

System.out.print("\nMit csinál a pingvin?:" );
pingvin.repul();
System.out.println("\nDe a pingvin nem tud repülni, szóval sérül a liskov ←
elv.");
}
```

Ezek után a main-ben mind a két esetben példányosítunk, azaz létrehozunk egy madarat, egy sast és egy pingvint is. Majd minden objektummal meghívjuk a repül függvényt. Az első kettővel nincs, és nem is lenne gond, mivel alapvetően tudnak repülni, viszont a pingvin, mint tudjuk nem tud repülni. Azonban ez a pingvin ahelyett, hogy hibát dobna a program, boldogan repked a virtuális térben, ami nekünk nem jó.



A terminal window showing the execution of a Java program named Liskov. The user runs the command ./liskov. The program outputs several lines of text, each starting with 'Mit csinál a ...?:' followed by the object's name (madár, sas, pingvin) and its corresponding behavior ('Repül' or 'De a pingvin nem tud repülni, szóval sérül a liskov elv.').

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ ./liskov
Repül a madár
Repül a sas
Repül a pingvin. De a pingvin nem tud repülni, ezért sérült a Liskov elv.
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ java liskovj
Mit csinál a madár?:Repülök

Mit csinál a sas?:Repülök

Mit csinál a pingvin?:Repülök

De a pingvin nem tud repülni, szóval sérül a liskov elv.
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ 
```

Erre egy megoldás a jobb OO tervezés. Vagyis ha például ha a Madár osztályunk megmadarna, de lenne két származtatott osztálya. Az egyik osztályba kerülnének a repülni tudó madarak, a másikba pedig azok a madarak, amik nem tudnak repülni. És ezekből az osztályokból származtatthatnánk tovább a sast, ami egy repülni tudó madár, illetve a pingvint is, ami pedig nem tud repülni.

## 13.2. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek! [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_1.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf) (98. fólia)

Megoldás videó:

Megoldás forrása: [Java](#) [C++](#)

Ebben a feladatban be kellett bizonyítani, hogy ha a gyerekosztályban létrehozunk egy metódust, akkor ha a gyerekosztályt szülőosztályként szeretnénk használni, akkor a gyerekosztály saját metódusait nem fogjuk tudni használni. Én az előző feladathoz hasonlóan maradtam a Madár-Sas példánál. A Madár a szülő, a Sas a gyerekosztály.

C++ kód:

```
class Madar{  
public:  
};  
  
class Sas : public Madar{  
public:  
    void repul() {  
        std::cout << "Repül";  
    }  
};
```

És java:

```
static class Madar{  
protected int szarnyhossz;  
public void setSzarnyhossz(int szarnyhossz) {  
    this.szarnyhossz = szarnyhossz;  
}  
  
}  
  
static class Sas extends Madar{  
    public int getSzarnyhossz() {  
        return szarnyhossz;  
    }  
}
```

A C++ esetében a gyerekosztálynak van egy `repul()` metódusa, ami szimplán csak kiirja a konzolra, hogy "Repül". Ezzel szemben a java példa egy kicsit bonyolultabb, mivel itt a szülő osztálynak, azaz a madárnak van egy szárnyhossz tulajdonsága, illetve egy `setSzarnyhossz()` metódusa, amivel a szárnyhossz tulajdonságot lehet beállítani. A gyermek osztály természetesen megörökli ezt a tulajdonságot, illetve metódust, szóval neki is szabadon lehet állítani a szárnyhossz tulajdonságát. De ezek mellett van egy `getSzarnyhossz()` metódusa is, ami visszaadja a sas objektum szárnyhosszát. Ez eddig teljesen normális, az érdekesség akkor kezdődik, amikor a main-be érünk.

C++:

```
int main() {  
  
    Madar* sas = new Sas();  
    Sas* sas2 = new Sas();  
  
    sas->repul();  
    sas2->repul();  
}
```

És java:

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

```
Madar sas = new Madar();
sas.setSzarnyhossz(80);

Sas sas2 = new Sas();
sas2.setSzarnyhossz(50);

System.out.println(sas2.getSzarnyhossz() + " " + sas.getSzarnyhossz()) ←
;
}
```

Mind a két esetben létrehozunk egy Sas típust Sas típusként, amivel nincs is gond, de létrehozunk egy Madár típust is Sas típusként. Ezek után a C++ kódban mind a két objektumok megpróbáljuk repteteni a repul() metódust használva. A Sas típusnak ezzel nem is lenne gondja, azonban a Madárnak igen.

A java kód itt is egy kicsit más képp működik. Itt a két típus létrehozása után minden a kettőnek beállítjuk a szárnyhosszát a setSzarnyhossz() segítségével, amivel nincs is gond, mivel ez eredetileg a Madár osztály metódusa, amit a Sas megörökölt. Azonban ezek után a getSzarnyhossz() metódus segítségével megpróbáljuk kiiratni minden a két objektum szárnyhosszát, ami csak az egyik esetben sikerülne. A végeredmény pedig:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ g++ -o SzuloGy SzuloGy.cpp
SzuloGy.cpp: In function 'int main()':
SzuloGy.cpp:22:7: error: 'class Madar' has no member named 'repul'
    sas->repul();
          ^
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ javac SzuloGyerek.java
SzuloGyerek.java:24: error: cannot find symbol
    System.out.println(sas2.getSzarnyhossz() + " " + sas.getSzarnyhossz());
                           ^
      symbol:   method getSzarnyhossz()
      location: variable sas of type Madar
1 error
andras@andrasubuntu:~/prog2/src/prog2/Liskov$
```

### 13.3. Anti OO

A BBP algoritmussal a Pi hexadecimális kifejtésének a 0. pozíciótól számított  $10^6$ ,  $10^7$ ,  $10^8$  darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/hu/tartalomat-javat/apas03.html#id561066>

Megoldás videó:

Megoldás forrása: [Java](#) [C++](#) [C#](#) [C](#)

Ehhez a feladathoz a Bárfai Norbert által [biztosított](#) forráskódokat használtam. A programokat egy 4.5 Ghz-re overclockolt AMD-FX8350 és 8GB ramot tartalmazó gépen futtattam. Lássuk is az eredményeket: A java program  $10^6$  számjegyet 0.906 másodperc alatt,  $10^7$  számjegyet 11.132 másodperc alatt,  $10^8$  számjegyet pedig 135.7 másodperc alatt számolta ki.

```
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ javac AntiJava.java
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ java AntiJava
6
0.906
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ javac AntiJava.java
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ java AntiJava
7
11.132
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ javac AntiJava.java
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ java AntiJava
12
135.7
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ 
```

A C++ program  $10^6$  számjegyet 2.37792 másodperc alatt,  $10^7$  számjegyet 26.5749 másodperc alatt,  $10^8$  számjegyet pedig 296.292 másodperc alatt számolta ki.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ g++ -o AntiCpp AntiCpp.cpp
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ ./AntiCpp
6
2.37792
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ g++ -o AntiCpp AntiCpp.cpp
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ ./AntiCpp
7
26.5749
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ g++ -o AntiCpp AntiCpp.cpp
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ ./AntiCpp
12
296.292
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ 
```

A C program  $10^6$  számjegyet 1.243691 másodperc alatt,  $10^7$  számjegyet 15.384056 másodperc alatt,  $10^8$  számjegyet pedig 186.029669 másodperc alatt számolta ki.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ gcc -o AntiC AntiC.c -lm
AntiC.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main ()
 ^~~~~
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ ./AntiC
6
1.243691
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ gcc -o AntiC AntiC.c -lm
AntiC.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main ()
 ^~~~~
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ ./AntiC
7
15.384056
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ gcc -o AntiC AntiC.c -lm
AntiC.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main ()
 ^~~~~
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ ./AntiC
12
186.029669
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ □
```

És végül a C# program  $10^6$  számjegyet 0.950131 másodperc alatt,  $10^7$  számjegyet 11.643125 másodperc alatt,  $10^8$  számjegyet pedig 140.350363 másodperc alatt számolta ki.

```
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ mono AntiCs.exe
6
0,950131
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ mcs AntiCs.cs
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ mono AntiCs.exe
7
11,643125
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ mcs AntiCs.cs
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ mono AntiCs.exe
12
140,350363
andras@andrasubuntu:~/prog2/src/prog2/Liskov$ □
```

Az egész összesítve egy táblázatba:

|        | <b>Java</b> | <b>C++</b> | <b>C#</b>  | <b>C</b>   |
|--------|-------------|------------|------------|------------|
| $10^6$ | 0.906       | 2.37792    | 0.950131   | 1.243691   |
| $10^7$ | 11.132      | 26.5749    | 11.643125  | 15.384056  |
| $10^8$ | 135.7       | 296.292    | 140.350363 | 186.029669 |

### 13.1. táblázat. Összehasonlítás

Ebből az látszik, hogy a sort a Java és a C# fej fej mellett haladva vezeti,  $10^6$  számjegynél a C# nyer, ám a másik két esetben pedig a Java. A dobogó harmadik helyét a C nyelv foglalja el, leghátul pedig a C++ kullog.

## 13.4. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_2.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf) (77-79 fóliát)!

Megoldás videó:

Megoldás forrása:

A ciklomatikus komplexitás egy forráskód összetettségét jelenti, amit gráfelmélettellel kell kiszámolni. A képlet hozzá  $M=E-N+2P$  ahol E a gráf elemeinek a száma, N a gráfban lévő csúcsok száma, és P pedig az összefüggő komponensek száma. Én ezt a feladatot a <http://www.lizard.ws/> oldal segítségével oldottam meg. Szimplán csak be kell illeszteni a forráskódot, ki kell választani, hogy milyen nyelven van írva a program, és ki is számolja nekünk. Én az első csokorban átnézett BBP és LZWBInFa programok java változatait számoltattam ki. Az eredmények:

Code analyzed successfully.

| Function Name                        | NLOC | Complexity | Token # | Parameter # |
|--------------------------------------|------|------------|---------|-------------|
| LzwBinFa::LzwBinFa                   | 3    | 1          | 9       |             |
| LzwBinFa::egyBitFeldolg              | 22   | 4          | 104     |             |
| LzwBinFa::kiir                       | 4    | 1          | 26      |             |
| LzwBinFa::kiir                       | 4    | 1          | 22      |             |
| LzwBinFa::Csomopont::Csomopont       | 5    | 1          | 21      |             |
| LzwBinFa::Csomopont::nullasGyermek   | 3    | 1          | 8       |             |
| LzwBinFa::Csomopont::egyesGyermek    | 3    | 1          | 8       |             |
| LzwBinFa::Csomopont::ujNullasGyermek | 3    | 1          | 11      |             |
| LzwBinFa::Csomopont::ujEgyesGyermek  | 3    | 1          | 11      |             |
| LzwBinFa::Csomopont::getBetu         | 3    | 1          | 8       |             |
| LzwBinFa::kiir                       | 15   | 3          | 107     |             |
| LzwBinFa::getMelyseg                 | 5    | 1          | 21      |             |
| LzwBinFa::getAtlag                   | 6    | 1          | 32      |             |
| LzwBinFa::getSzoras                  | 13   | 2          | 68      |             |
| LzwBinFa::rmelyseg                   | 11   | 3          | 51      |             |
| LzwBinFa::ratlag                     | 12   | 4          | 66      |             |
| LzwBinFa::rszoras                    | 12   | 4          | 78      |             |
| LzwBinFa::usage                      | 3    | 1          | 14      |             |
| LzwBinFa::main                       | 60   | 14         | 401     |             |

Itt viszont csak a különböző programrészek komplexitását kapjuk meg, nem pedig az egész programét. Ha összeadjuk a programrészek komplexitását akkor megkapjuk hogy a teljes program ciklomatikus komplexitása 45. De most nézzük a BBP algoritmust, az kicsit egyszerűbb:

| Code analyzed successfully. |       |             |         |             |
|-----------------------------|-------|-------------|---------|-------------|
| File Type                   | .java | Token Count | 468     | NLOC        |
| Function Name               | NLOC  | Complexity  | Token # | Parameter # |
| BBP::main                   | 4     | 1           | 26      |             |
| BBP::magic                  | 25    | 3           | 224     |             |
| BBP::cut                    | 8     | 2           | 36      |             |
| BBP::solve                  | 7     | 2           | 63      |             |
| BBP::mod                    | 21    | 5           | 99      |             |

Itt is ugyan az érvényes, mint az előző programnál. Azaz a teljes program ciklomatikus komplexitásának a meghatározásához össze kell adnunk az egyes programrészek ciklomatikus komplexitását. Ez által azt kapjuk, hogy a BBP algoritmus java változatának ciklomatikus komplexitása 13.

## 14. fejezet

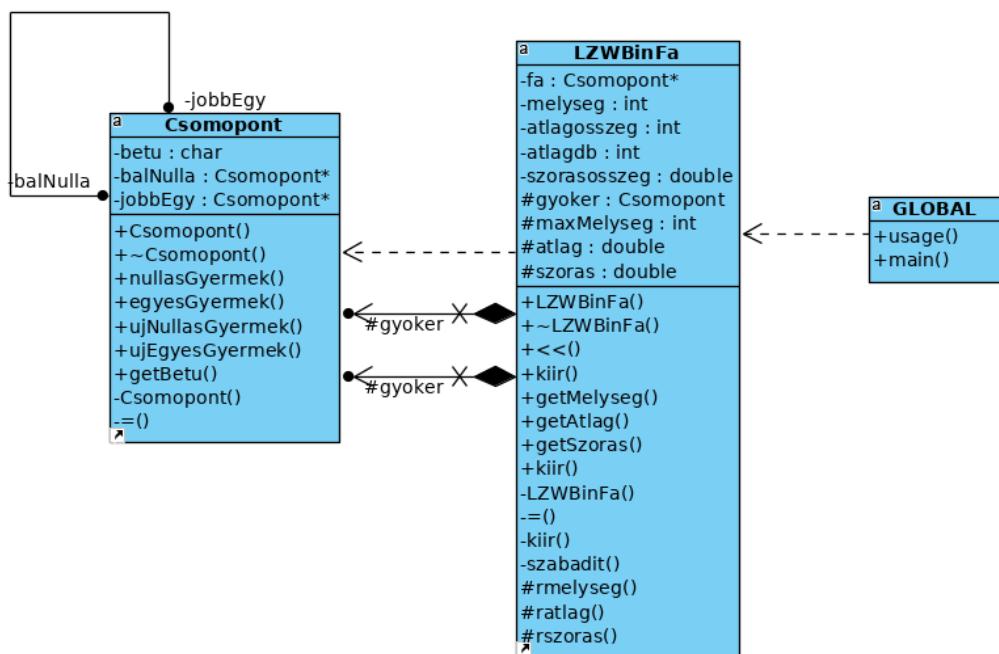
# Helló, Mandelbrot!

### 14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: [https://youtu.be/Td\\_nlERIEOs](https://youtu.be/Td_nlERIEOs). <https://arato.inf.unideb.hu/batfai.norbert/UD> (28-32 fólia)

Megoldás videó:

Megoldás forrása:



Ebben a feladatban a BevProgon és Prog1-en már tárgyalta [z3a7.cpp](#) program forrásából kellett UML diagramot létrehozni. Én ehhez a [Visual Paradigm](#) 30 napos ingyenes próbaverzióját használtam. A diagramot roppant egyszerű legenerálni. A visual paradigm feltelepítése és elindítása után oda kell navigálnunk, hogy Tools > Code > Instant Reverse..., majd pedig ki kell választani, hogy milyen nyelvű forrásból szeretnénk

diagramot generálni, és meg kell adni a fájl helyét. Ezek után már csak ki kell választani azokat az osztályokat, amelyeket meg szeretnénk mutatni a diagrammon, és kész is. Az általam generált diagramot a fentebb található képen lehetett látni.

Ezen kívül rá kell mutatni az aggregáció és kompozíció kapcsolatára. Azonban, mielőtt ezt megtehetnénk, először az asszociáció fogalmát kell tisztázni.

Ha egy modellben két osztálynak kommunikálnia kell egymással, akkor szükségünk van egy kapcsolatra a két osztály között. Ezt a kapcsolatot reprezentálja az asszociáció. Az asszociációt egy a két osztály között lévő vonal, valamit az azon lévő irányt mutató nyil(ak) jelöli(k). Ha a vonal minden oldalán van nyíl, akkor az asszociáció kétirányú.

Az aggregáció és kompozíció az asszociáció részhalmazai, vagyis az asszociáció különleges esetei. Mind a két esetben egy osztály objektuma "birtokol" egy másik osztály másik objektumát, de van a kettő között egy kis különbség.

Az aggregáció egy olyan kapcsolatot jelent, amiben a gyerek a szülőtől függetlenül létezhet. Például ha van tanóra, ami a szülőosztály, és tanuló, ami a gyerekosztály. Ha töröljük a tanórát, attól a tanulók még léteznek.

Ezzel szemben a kompozíció esetében egy olyan kapcsolatról van szó, amiben a gyerek nem létezhet a szülő nélkül. Például ha van egy ház szülőosztályunk, és egy szoba gyerekosztályunk. A szoba nem létezhet a ház nélkül.

Az aggregációt és a kompozíciót is vonal+rombusz kombinációval lehet jelölni, azonban az aggregációnál a rombusz üres, a kompozíciójánál pedig nem.

Ezek alapján meg tudjuk mondani, hogy a fentebb látható ábrán a

Csomopont gyoker;

elem a kompozíció.

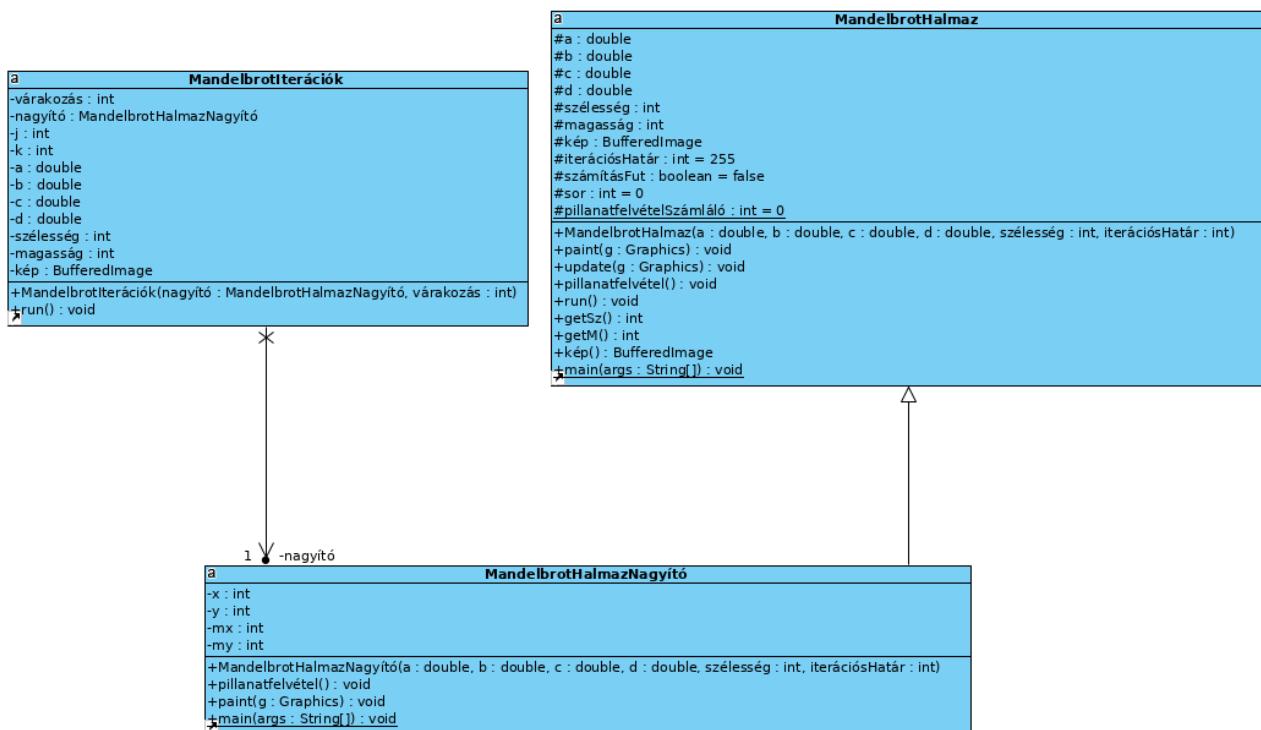
## 14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Megoldás videó:

Megoldás forrása:<https://github.com/raczandras/progbook/tree/master/src/prog2/Mandelbrot>

Ebben a feladatban UML-ben kellett osztályokat megtervezni, majd pedig a diagramból forrást generálni. Én ehhez a feladathoz, ha már Mandelbrot a csokor neve, a MandelbrotHalmazt próbáltam meg lemodellezni a visual paradigm nevű szoftver segítségével. Ehhez a következő diagrammot sikerült összeállítani:



Ahogy láthatjuk a **MandelbrotIterációk** forrásfájl nagyító objektuma asszociációban áll a **MandelbrotHalmazNagyító** osztállyal, ami pedig kompozícióban áll a **MandelbrotHalmaz** osztállyal. Ezek után a kód generálása már gyerekjáték. Annyit kell tenni, hogy rákattintunk a Tools > generate java code... opcióra, majd pedig a kapott menüben megadjuk azt, hogy hova szeretnénk generálni a forrást, és a visual paradigm automatikusan legenerálja nekünk. Ha a fent megjelölt forrást összehasonlítjuk az eredetivel, akkor elmondhatjuk, hogy a generált forrás nagyon hasonlít az eredetre, hiszen a források szerkezete megyegyezik.

```

public void update(java.awt.Graphics g) {
    // TODO - implement MandelbrotHalmaz.update
    throw new UnsupportedOperationException();
}

public void pillanatfelvétel() {
    // TODO - implement MandelbrotHalmaz.pillanatfelvétel
    throw new UnsupportedOperationException();
}

public void run() {
    // TODO - implement MandelbrotHalmaz.run
    throw new UnsupportedOperationException();
}

public int getSz() {
    // TODO - implement MandelbrotHalmaz.getSz
    throw new UnsupportedOperationException();
}
  
```

```
public int getM() {
    // TODO - implement MandelbrotHalmaz.getM
    throw new UnsupportedOperationException();
}

public java.awt.image.BufferedImage kép() {
    // TODO - implement MandelbrotHalmaz.kép
    throw new UnsupportedOperationException();
}

/**
 *
 * @param args
 */
public static void main(String[] args) {
    // TODO - implement MandelbrotHalmaz.main
    throw new UnsupportedOperationException();
}
```

Ez a generált forrásnak egy része. És ha ezt összehasonlítjuk az eredeti forrássnak ugyan ezen részével, akkor észrevehetjük, hogy a két forrás szerkezete megegyezik, viszont nyílvánvalóan, ha kódot generálunk, akkor csak a függvények létrehozása történik meg, a törlök viszont üres marad. Azonban, ha már valakinek összeállt a fejében egy ötlet arról, hogy hogyan fog kinézni a forrásának a felépítése, akkor annak az embernek nagyon jól tud jönni, ha azt a vázat létre tudja hozni egy UML diagramban, és abból generálni tud forrást.

### 14.3. Egy esettan

A BME-s C++ tankönyv 14. fejezetét (427-444 elmélet, 445-469 az esettan) dolgozzuk fel!

Megoldás video:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/Mandelbrot/esettan>

Ebben a feladatban a Szoftverfejlesztés C++ nyelven című könyvben szereplő részletet kellett feldolgozni. Az elméleti része főképp az UML nyelvet mutatja be. Szó esik az osztálydiagrammokról, azon belül arról, hogy az osztályokat téglalapokban lehet ábrázolni, amik három részre vannak osztva. Ezen kívül szó esik még az osztály nevéről, ami a téglalap felső részében található meg, az osztály attribútumairól, amik a téglalap középső részében helyezkednek el, illetve az osztály műveleteiről is, amelyek pedig a téglalap alsó részében találhatóak.

Majd pedig szó esik a láthatóságról, ami ugyebár lehet public, protected, és private. Ezeken kívül szó esik még a kapcsolatokról is, amit a könyv egy üres háromszöggel jelöl, valamit szó esik az asszociációkról, amelyet pedig egy nyíllal jelöl a könyv. Említést kap a kompozíció aminek a jelölése egy teli rombusz, illetve az aggregáció is, aminek pedig egy üres rombusz. Valamint ezek jelentésére is kitér. Ahogy azt már egy korábbi feladat leírásában említettem:

Az aggregáció egy olyan kapcsolatot jelent, amiben a gyerek a szülőtől függetlenül létezhet. Például ha van tanóra, ami a szülőosztály, és tanuló, ami a gyerekosztály. Ha töröljük a tanórát, attól a tanulók még léteznek. Ezzel szemben a kompozíció esetében egy olyan kapcsolatról van szó, amiben a gyerek nem

létezhet a szülő nélkül. Például ha van egy ház szülőosztályunk, és egy szoba gyerekosztályunk. A szoba nem létezhet a ház nélkül.

Végül pedig a kódgenerálásról és a kód visszafejtésről. Ezen belül arról, hogy pontosan mik ezek, ugye forward és reverse engineering, azaz a kész kódból UML diagram generálása, valamit kész UML diagramból kód generálása, amikről az első két feladat szólt. Éppen ezért lehet hogy nem utolsóként kellett volna megcsinálni ezt a feladatot, hanem elsőként. Mivel nagy segítség lehetett volna a többi feladat megoldásánál, ha már ismerem ezeket a dolgokat.

Ezek után következett maga az esettanulmány, ami egy program elkészítéséből állt. Maga a feladat egy számítógép kereskedéssel volt kapcsolatos. Eléggé összetett programról van szó, ami támogatja a termékek állományból való betöltését, képernyőre történő listázását, állományba való kiírását, és az árképzés rugalmas alakítását. És ha ez még nem lenne elég, még a lehetséges jövőbeli befektetésekre is gondolni kell, azaz a teljesen új termékcsaládok értékesítésének bevezetésére is lehetőséget kell biztosítani. Magához a feladathoz volt megadva forrás, azonban még kell mellé írni saját magunktól is. Na de nézzük, hogy mi mit is jelent a programunk kódja

Az első a product osztály, ami a programunk szülőosztálya, magyarra lefordítva termék. Ebből az osztályból lesz származtatva a többi három osztály, a display, azaz kijelző, a harddisk, azaz merevlemez, és a compositeproduct, ami pedig az összetett termék, például egy kész számítógép.

A product osztálynak három tagváltozója van. Ezek a name, ami a termék neve InitialPrice, azaz a termék eredeti ára, illetve a dateOfAcquisition, ami pedig a temék beszerzési ideje. Ezek a tagváltozók protectedek, éppen ezért szükség van getter függvényekre is hozzájuk. Valamint van két még másik getter függvényünk is, a getAge, ami a beszerzési, és a jelenlegi idő felhasználásával kiszámolja, hogy milyen idős az adott termék. Valamint a getCurrentPrice, ami a termék jelenlegi árát adná vissza, azonban jelenleg ennek a függvénynek a visszatérési értéke az eredeti ár. Maga az osztály:

```
time_t Product::getDateOfAcquisition() const {
    return dateOfAcquisition;
}

int Product::getInitialPrice() const {
    return initialPrice;
}

std::string Product::getName() const {
    return name;
}

Product::Product() {}

Product::Product(std::string name, int initialPrice, time_t ←
    dateOfAcquisition): name(name), initialPrice(initialPrice),
    dateOfAcquisition(dateOfAcquisition) {
}

int Product::getAge() const{
    time_t currentTime;
    time(&currentTime);
    double timeDiffInSec = difftime(currentTime, dateOfAcquisition);
    return (int)(timeDiffInSec/(3600*24));
}
```

```
}
```

```
int Product::getCurrentPrice() const {
    return initialPrice;
}
```

Az inputstream, illetve az outputstream operátorok segítségével a következő függvények fogják elvégezni a termékek beolvasását, illetve kiíratását: `print()`, amely megadja a termék típusát, és nevét, a `printParams()`, amely megadja a termék paramétereit, azaz a eredeti árat, a beszerzési időt, a termék korát, és a termék jelenlegi árát, a `writeParamsToStream()`, amely megadja a termék nevét, eredeti árát, és a beszerzési idejét stringgé alakítva, és a `loadParamsFromStream()`, amely pedig beolvassa a termékeket. Maga az osztály:

```
void Product::print(std::ostream &os) const {
    os << "Type: " << getType() << ", ";
    os << "Name: " << getName();
    printParams(os);
}

void Product::printParams(std::ostream &os) const {
    char strDateOfAcquisition[9];
    strftime(strDateOfAcquisition, 9, "%Y%m%d",
             gmtime(&dateOfAcquisition));

    os << ", " << "Initial price: " << initialPrice
    << ", " << "Date of acquisition: " << strDateOfAcquisition
    << ", " << "Age: " << getAge()
    << ", " << "Current price: " << getCurrentPrice();
}

void Product::writeParamsToStream(std::ostream &os) const {
    char strDateOfAcquisition[9];
    tm* t = localtime(&dateOfAcquisition);
    strftime(strDateOfAcquisition, 9, "%Y%m%d", t);
    os << " " << name << " " << initialPrice << " " << strDateOfAcquisition;
}

void Product::loadParamsFromStream(std::istream &is) {
    is >> name;
    is >> initialPrice;

    char buff[9];
    is.width(9);
    is >> buff;
    if (strlen(buff) != 8)
        throw range_error("Invalid time format");

    char workBuff[5];
    tm t;
    int year;
```

```
strncpy(workBuff, buff, 4); workBuff[4] = '\0';
year = atoi(workBuff); t.tm_year = year - 1900;
strncpy(workBuff, &buff[4], 2); workBuff[2] = '\0';
t.tm_mon = atoi(workBuff) - 1;
strncpy(workBuff, &buff[6], 2); workBuff[2] = '\0';
t.tm_mday = atoi(workBuff);
t.tm_hour = t.tm_min = t.tm_sec = 0;
t.tm_isdst = -1;

dateOfAcquisition = mktime(&t);
}

std::istream& operator>>(istream& is, Product& product) {
    product.loadParamsFromStream(is);
    return is;
}

std::ostream& operator<<(ostream& os, Product& product) {
    os << product.getCharCode();
    product.writeParamsToStream(os);
    return os;
}
```

A harddisk, azaz merevlemez osztálynak a tagváltozói szintén a name, initialPrice, dateOfAcquisition, amelyek megtalálhatóak a product osztályban is, azonban, itt még van egy speedRPM változó is, amely megmondja, hogy hány RPM-es a merevlemez. Ezen kívül itt az is függvényekhez hozzá lett fűzve, hogy az RPM-et is ki kell írni, illetve be kell olvasni. Valamint a getCurrentPrice() függvény itt már úgy működik, hogy ha a termék fiatalabb 30 napnál, akkor az eredeti árat, ha 30 és 90 nap közötti, akkor az eredeti ár 80 százalékát, ha pedig idősebb mint 90 nap, akkor pedig az eredeti ár 80 százalékát adja vissza jelenlegi árként. Illetve található egy getSpeedRPM() függvény is, ami visszaadja a merevlemez RPM értékét. Maga az osztály:

```
int HardDisk::getCurrentPrice() const{
    int ageInDays = getAge();
    if(ageInDays < 30)
        return initialPrice;
    else if (ageInDays >= 30 && ageInDays < 90)
        return (int)(0.9 * initialPrice);
    else
        return (int)(0.8 * initialPrice);
}

HardDisk::HardDisk() {};

HardDisk::HardDisk(std::string name, int initialPrice, time_t ←
    dateOfAcquisition, int speedRPM):
    Product(name, initialPrice, dateOfAcquisition), speedRPM(speedRPM) {}

int HardDisk::getSpeedRPM() const {
    return speedRPM;
```

```
}
```

```
void HardDisk::printParams(std::ostream& os) const {
    Product::printParams(os);
    os << ", " << "SpeedRPM: " << speedRPM;
}
```

```
void HardDisk::writeParamsToStream(std::ostream &os) const {
    Product::writeParamsToStream(os);
    os << ' ' << speedRPM;
}
```

```
void HardDisk::loadParamsFromStream(std::istream &is) {
    Product::loadParamsFromStream(is);
    is >> speedRPM;
}
```

A Display osztály tagváltozói a name azaz név, az initialPrice, azaz kezdő ár, a dateOfAcquisition, azaz beszerzési idő, az inchWidth, azaz a szélesség col-ban, illetve az inchHeight azaz a magasság colban. A két új tagváltozó kap gettereket, illetve az input output részben is beolvastatjuk, illetve kiírattatjuk ezeknek a változóknak az értékeit a programmal. A getCurrentPrice() pedig ugyan úgy működik itt is, mint a merevlemez esetében, azaz ha a termék fiatalabb 30 napnál, akkor az eredeti árat, ha 30 és 90 nap közötti, akkor az eredeti ár 80 százalékát, ha pedig idősebb mint 90 nap, akkor pedig az eredeti ár 80 százalékát adja vissza jelenlegi árként. Maga az osztály:

```
void Display::printParams(std::ostream& os) const {
    Product::printParams(os);
    os << ", " << "InchWidth: " << inchWidth;
    os << ", " << "InchHeight: " << inchHeight;
}

void Display::writeParamsToStream(std::ostream &os) const {
    Product::writeParamsToStream(os);
    os << ' ' << inchWidth << ' ' << inchHeight;
}

void Display::loadParamsFromStream(std::istream &is) {
    Product::loadParamsFromStream(is);
    is >> inchWidth >> inchHeight;
}

Display::Display() {}

Display::Display(std::string name, int initialPrice, time_t ←
dateOfAcquisition, int inchWidth, int inchHeight):
    Product(name, initialPrice, dateOfAcquisition), inchWidth(inchWidth), ←
    inchHeight(inchHeight) {}

int Display::getCurrentPrice() const {
    int ageInDays = getAge();
```

```
if(ageInDays < 30)
    return initialPrice;
else if (ageInDays >= 30 && ageInDays < 90)
    return (int)(0.9 * initialPrice);
else
    return (int)(0.8 * initialPrice);
}

int Display::getInchWidth() const {
    return inchWidth;
}

int Display::getInchHeight() const {
    return inchHeight;
}
```

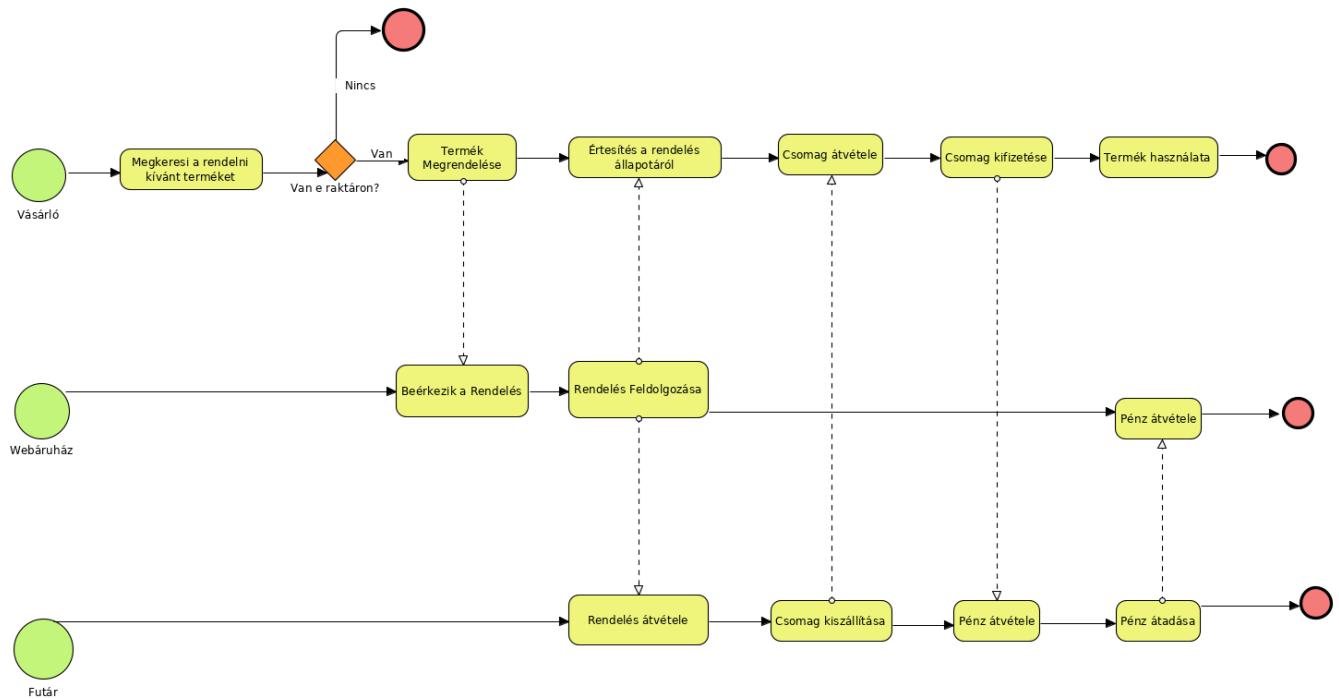
## 14.4. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog34-47.pdf> (34-47 fólia)

Megoldás video:

Megoldás forrása:

Ebben a feladatban BPMN-ben, azaz Business Process Model and Notation használatával kellett modellezni valamit. Maga a BPMN egy folyamatábra, egy grafikai reprezentációja az üzleti folyamatoknak. Az UML-hez hasonlóan szintén egy modellező eszköz. Az én példám egy minden napí esetet ír le, egy csomag megrendelését egy webáruházból, vagyis, hogy mi történik a között, hogy a vásárló megrendeli, és megkapja a csomagot. Ennek a feladatnak a megoldásához a visual paradigm nevű szoftvert választottam. A folyamatábra a következőképpen néz ki:



Látható a folyamatábrán, hogy három különböző entitás dolgozik a csomagért. Az egyik a vásárló, aki a csomagot rendeli, a második a webáruház, ami a csomagot eladja, és a harmadik pedig egy futárszolgálat, ami pedig házhoz viszi a csomagot. Maga a folyamat rendkívül egyszerű. Először is a vásárló meglátogatja a webáruházat, itt kezdődik a folyamatábra. Majd megkeresi a csomagot, amit rendelni szeretne. Ezek után jön egy elágazás, mégpedig hogy van-e a keresett termék raktáron. Amennyiben nincs, úgy itt véget is ér a folyamat. Azonban ha van, akkor megrendeli. Itt történik egy interakció a vásárló és a webáruház között. A webáruház megkapja a rendelést, azt feldolgozza és előkészíti a szállításra. Ezek után két interakció is történik. Egyrészt a webáruház átadja a futárszolgálatnak a csomagot, másrészt pedig szól a vásárlónak, hogy át lett adva a csomagja a futárnak. Majd a futárszolgálat kiszállítja a csomagot a vevőnek, amit az átvesz és kifizet. Ezek után a vevő már csak használja a terméket, ezzel az Ő folyamata véget ér. A futár pedig a kapott pénzt átadja a webáruháznak, és ezzel mindenbőlük folyamata véget ér. Ez nyilván egy nagyon egyszerű példa, amit lehetett volna sokkal bonyolultabb is, de a BPMN működésének a bemutatására tökéletes.

# 15. fejezet

## Helló, Chomsky!

### 15.1. Encoding

Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Megoldás videó:

Megoldás forrása:

Ebben a feladatban a Bátfa Norbert által megadott [MandelbrothalmazNagyító.java](#) forrást kellett futtatni. Ezzel csupán annyi a gond, hogy a forrás tele van ékezes betűkkel. Éppen ezért, amikor megpróbáljuk lefordítani, akkor a képen látható hibákat kapjuk:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
andras@andrasubuntu:~/prog2/src/prog2/Chomsky$ javac MandelbrotHalmazNagyító.java  
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xED) for encoding UTF-8  
 * MandelbrotHalmazNagyító.java  
          ^  
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xF3) for encoding UTF-8  
 * MandelbrotHalmazNagyító.java  
          ^  
MandelbrotHalmazNagyító.java:4: error: unmappable character (0xED) for encoding UTF-8  
 * DIGIT 2005, Javat tanítok  
          ^  
MandelbrotHalmazNagyító.java:5: error: unmappable character (0xE1) for encoding UTF-8  
 * Bátfa Norbert, nbatfai@inf.unideb.hu  
          ^  
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xED) for encoding UTF-8  
 * A Mandelbrot halmazt nagyítás kirajzolás osztály.  
          ^  
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding UTF-8  
 * A Mandelbrot halmazt nagyítás kirajzolás osztály.  
          ^  
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xE9) for encoding UTF-8  
 * A Mandelbrot halmazt nagyítás kirajzolás osztály.  
          ^  
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding UTF-8  
 * A Mandelbrot halmazt nagyítás kirajzolás osztály.  
          ^  
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xE1) for encoding UTF-8  
 * A Mandelbrot halmazt nagyítás kirajzolás osztály.  
          ^  
MandelbrotHalmazNagyító.java:11: error: unmappable character (0xE1) for encoding UTF-8  
 * @author Bátfa Norbert, nbatfai@inf.unideb.hu  
          ^  
MandelbrotHalmazNagyító.java:14: error: unmappable character (0xED) for encoding UTF-8  
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {  
          ^  
MandelbrotHalmazNagyító.java:14: error: unmappable character (0xF3) for encoding UTF-8  
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {  
          ^
```

Mint nagyon sokszor, a fordító most is a barátunk: "Unmappable character for encoding UTF-8". Vagyis a forrás kódolásával van a gond. Vagyis ezek a karakterek nem találhatóak meg az UTF-8 kódolásban. Ez azt jelenti, hogy egy másik kódolásra kell átállítani a forrást, amit a -encoding kapcsolóval lehet állítani. Mostmár csak arra kellett rájönni, hogy mire kellene átállítani a kódolást. Ehhez megkerestem a [Java által támogatott karakterkódolásokat](#). Itt amire felkaptam a fejem, az a windows-1250 kódolás, aminek a leírása az, hogy a Windows Kelet Európai karakterkódolása, és arra gondolva, hogy vagy jó vagy nem, kipróbáltam hogy működik-e, és műköött. Az eredmény pedig:



## 15.2. I334d1c4

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet> (Ha ez első részben nem tette meg, akkor írasd ki és magyarázd meg a használt struktúratömb memória foglalását!)

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Chomsky/Leet.java>

Ebben a feladatban a magasszintű programozási nyelvek 1 tárgyon már tárgyal leet cipheret kellett megírni. A különbség csak annyi, hogy míg legutóbb ezt lexer segítségével kellett megcsinálni, addig most már az egészet saját magunktól kell megírni. Az én megoldásom a Leet osztállyal kezdődik:

```
public class Leet {  
  
    public static void main(String[] args) throws Exception{  
        if(args.length != 2){  
            System.out.println("usage: inputfile outputfile");  
            System.exit(-1);  
        }  
    }  
}
```

```
}

java.io.FileReader file = new java.io.FileReader(args[0]);
java.io.FileWriter fw = new java.io.FileWriter(args[1]);

LeetCipher lc = new LeetCipher();
int k = 0;

while ((k=file.read()) != -1) {
    fw.write(lc.chiper((int)Character.toUpperCase((char)k)));
}

file.close();
fw.close();
}
}
```

Az egész program úgy kezdődik, hogy megvizsgáljuk az argumentumok számát. Amennyiben az argumentumok száma nem kettő, úgy felvilágosítjuk a felhasználót arról, hogy hogyan kell használni a programot. Azonban ha az argumentumok száma kettő, akkor létrehozunk egy FileReadert és egy FileWritert amik a fájlok beolvasását és kiírását fogják elvégezni. Majd pedig egy while ciklusban addig olvassuk a fájlt, amíg az véget nem ér, és ki írjuk minden egyes karakternek a leet formáját. Ezek után már csak bezártuk a fájl beolvasót és kiírót. Ezek után következik a LeetCipher osztály:

```
class LeetCipher {
private String[] leetchars = new String[]{
    "4", "8", "<", "[", "3", "|=", "6", "|-", "1", "_|", "|<", "|", "|V|", "\\", "O",
    "|>", "0.", "|2", "5", "7", "|_|", "\\", "\\", "X/", "}{", "/",
};

private String[] leetnums = new String[]{
    "O", "I", "Z", "E", "A", "S", "G", "T", "B", "g"
};

public String chiper(int ch) {
    if (ch >= 65 && ch <= 90) {
        return leetchars[ch - 65];
    }

    else if (ch >= 48 && ch <= 57) {
        return leetnums[ch - 48];
    }
    else {
        return String.valueOf((char)ch);
    }
}
}
```

Ebben az osztályban először egy String tömböt találunk, ami a karakterek leet alakjait tárolja, majd pedig egy másik String tömb, ami pedig a számoknak megfelelő leet jeleket tárolja. Ezek után a `cipher()`

metódus, ami az átalakítást végzi. A metódus eldönti, hogy a soron következő karakter szám vagy betű e, és az annak megfelelő leet jelet adja vissza. A program működés közben pedig így néz ki:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Chomsky$ cat teszt
valami szoveg amivel le lehet tesztelni azt hogy mennyire mukodik a sajat leet cipherem
amit prog2 orara kellett megirni mivel ez volt az egyik feladat
csak nem szerettem volna Lorem Ipsum szoveget betenni
andras@andrasubuntu:~/prog2/src/prog2/Chomsky$ java Leet teszt tesztout
andras@andrasubuntu:~/prog2/src/prog2/Chomsky$ cat tesztout
\|4|4|V|1 520\|36 4|V|1\|3| |3 |3|-|37 735273|\|1 427 |-|06`/ |V|3|\|\\`/1|23 |V||_||<0[]1|< 4 54_|47 |337 <1|>|-|3|23|V|
4|V|17 |>|206Z 0|24|24 |<3||377 |V|361|2|\|1 |V|1\|3| 32 \|0|7 42 36`/1|< |=3|4[|47
<54|< |\|3|V| 523|23773|V| \|0|\|4 |0|23|V| 1>5|_|V| 520\|3637 8373|\|\\|1
andras@andrasubuntu:~/prog2/src/prog2/Chomsky$ 
```

## 15.3. Full screen

Készítsünk egy teljes képernyős Java programot! Tipp: [https://www.tankonyvtar.hu/en/tartalom/tkt/javatanitok-javat/ch03.html#labirintus\\_jatek](https://www.tankonyvtar.hu/en/tartalom/tkt/javatanitok-javat/ch03.html#labirintus_jatek)

Megoldás videó:

Tutorált: Huri Patrik

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Chomsky/Client.java>

Ebben a feladatban egy teljes képernyős java programot kellett írni, amihez én egy korábbi projektemet használtam fel. Mivel maga a program elég hosszú, ezért csak azokra a részekre koncentrálnék, ami a teljes képernyő kialakulásában szerepet játszik. Ez azt jelenti, hogy az `ejj jatekos()` függvényre fogok koncentrálni, mivel a program többjátékos része nem használ grafikus felületet. De az egyjátékos részben már rögtön az első sor is fontos, hiszen létrehozunk egy `GraphicsDrive`-ot, ami segíteni fog a teljes képernyőre váltásban. Ezek után pedig létrehozunk egy `KeyListener`-t, ami az ESC gomb lenyomására fog fülelni, hiszen amennyiben a felhasználó lenyomja azt a billentyűt, akkor a program kilép.

```
GraphicsDevice gd = GraphicsEnvironment.getLocalGraphicsEnvironment().  
    getDefaultScreenDevice();  
  
KeyListener listener = new KeyListener() {  
  
    @Override  
    public void keyPressed(KeyEvent event) {  
  
        if(event.getKeyCode() == KeyEvent.VK_ESCAPE)  
            System.exit(0);  
    }  
  
    @Override  
    public void keyReleased(KeyEvent event) {}  
  
    @Override  
    public void keyTyped(KeyEvent event) {}  
};
```

Ezek után létrehozunk egy JFrame-t, ami tulajdonképpen az alkalmazás ablaka, és hozzá is adjuk a frame-hez a KeyListeneret, majd pedig létrehozunk pár gombot, illetve szövegmezőt is, és azokhoz is hozzárendeljük a KeyListeneret. Erre azért van szükség, mivel csak akkor fog működni a KeyListener-ünk, ha hozzá van rendelve ahhoz az elemhez, ami éppen fókuszban van, és nem elég csak a Frame-hez hozzárendelni. Vagyis ha a program épp egy gombra kattintásra vár, de ahhoz a gombhoz nincs hozzárendelve a KeyListener, akkor nem fog bezárulni az alkalmazás, hiába nyomogatjuk az ESC gombot (igen ezt tapasztalatból mondom sajnos).

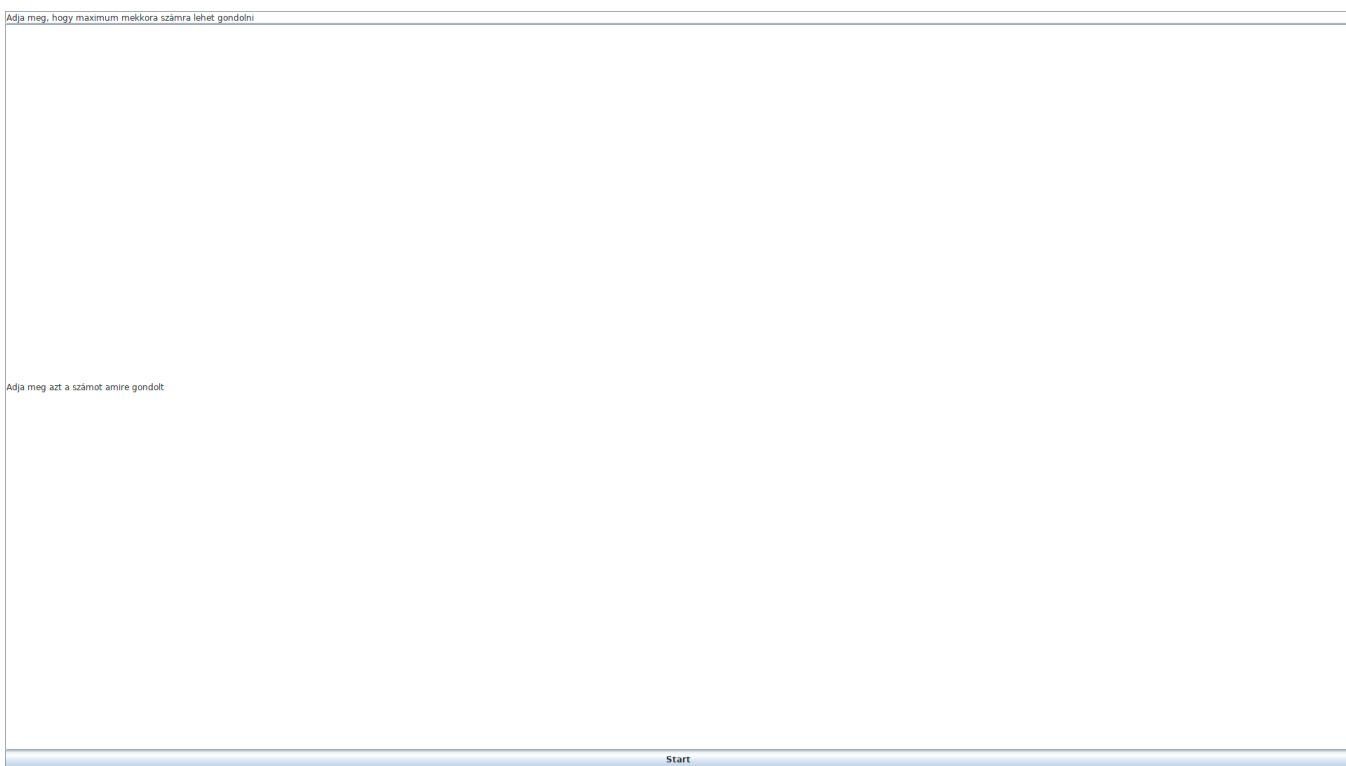
```
JFrame options = new JFrame("");
options.setTitle("Egy játékos mód");
options.addKeyListener(listener);
options.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JTextField maxszam = new JTextField("Adja meg, hogy maximum mekkora ↪
számra lehet gondolni");
JTextField gondolt = new JTextField("Adja meg azt a számot amire ↪
gondolt");
JButton start = new JButton("Start");
JButton ujra = new JButton("Újra");
options.getContentPane().add(BorderLayout.NORTH,maxszam);
options.getContentPane().add(BorderLayout.CENTER,gondolt);
options.getContentPane().add(BorderLayout.SOUTH,start);
maxszam.addKeyListener(listener);
gondolt.addKeyListener(listener);
```

Majd pedig a teljes képernyőre váltás, ami úgy történik, hogy megnézzük, hogy támogatja e a számítógépünk a teljes képernyőt a gd.isFullScreenSupported() függvényel. Ha igen, akkor undercorated-re állítjuk a frame-t, ami szükséges az igazi teljes képernyőhöz, majd pedig a setFullScreenWindow() függvényel teljes képernyőre állítjuk a frame-t.

```
if (gd.isFullScreenSupported()) {
    options.setUndecorated(true);
    gd.setFullScreenWindow(options);
}
else{
    System.err.println("Nem jó");
    options.setSize(600, 200);
    options.setVisible(true);
}
```

Az eredmény pedig egy teljes képernyős alkalmazás:



## 15.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis\_prel\_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Chomsky/para6.cpp>

Ebben a feladatban a Bátfai Norbert által megadott programon kellett apró változtatásokat elvégezni. Ehhez először fel kellett telepíteni a boost-ot, amit a következő parancssal lehet megtenni: `sudo apt-get install libboost-all-dev`, illetve e mellett még az OpenGL-re is szükség van, amit pedig a következő parancs kiadásával lehet megtenni: `sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev` majd pedig a forráskódban a kommentekben megadott módon lehet fordítani és futtatni.

Az egyik dolog, amin én módosítottam, az a színvilág, amihez a `glColor3f()` függvényt kellett használni. Ennek a függvénynek három paramétere van, az első a piros, a második a zöld, a harmadik pedig a kék szín intenzitását állítja be. Ez azt jelenti, hogy a 0.1 0.0 0.0 értékek egy sötét piros színt adnának, a 0.8 0.0 0.0 értékek pedig egy intenzív világos piros színt eredményeznek. Én egy minimalista stíussal dolgoztam, ezért a világos és sötét szürke színekkel játszadoztam, ahogy az az alábbi képen is látszani fog.

A másik dolog, amin változtattam, az az irányítás. Eddig a kockákat a billentyűzeten található nyilakkal, illetve a page up és page down gombokkal lehetett forgatni. Azonban, mivel nagyítani pedig a + és - gombokkal lehet, ezért nekem a forgatás a W,A,S,D,Q,E billentyűkkel jobban kézreáll. Ehhez töröltem az `skeyboard()` függvényt, és átírtam a `keyboard()` függvényt, ami mostmár a következőképpen néz ki:

```
void keyboard ( unsigned char key, int x, int y )  
{
```

```
if ( key == '0' ) {
    index=0;
} else if ( key == '1' ) {
    index=1;
} else if ( key == '2' ) {
    index=2;
} else if ( key == '3' ) {
    index=3;
} else if ( key == '4' ) {
    index=4;
} else if ( key == '5' ) {
    index=5;
} else if ( key == '6' ) {
    index=6;
} else if ( key == 'f' ) {
    transp = !transp;
} else if ( key == '-' ) {
    ++fovy;

    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective ( fovy, ( float ) w/ ( float ) h, .1f, ←
                    1000.0f );
    glMatrixMode ( GL_MODELVIEW );

} else if ( key == '+' ) {
    --fovy;

    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective ( fovy, ( float ) w/ ( float ) h, .1f, ←
                    1000.0f );
    glMatrixMode ( GL_MODELVIEW );

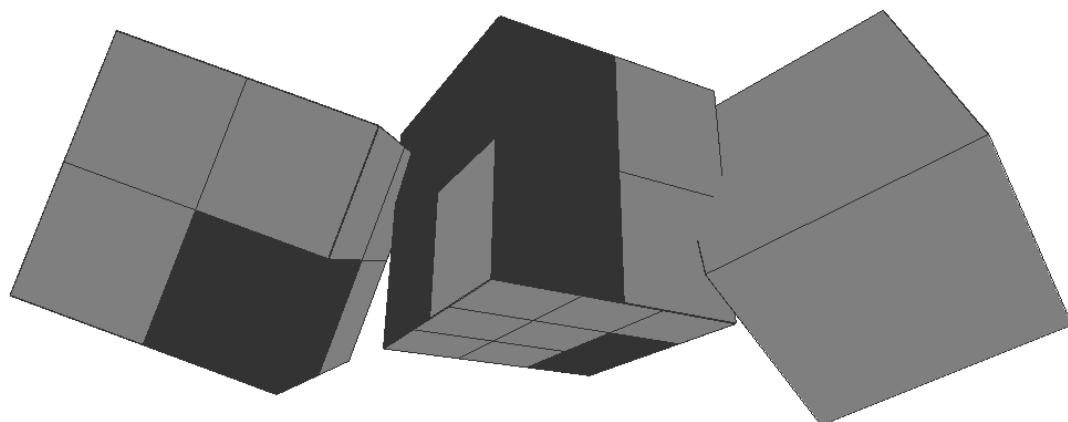
}

else if ( key == 'w' ) {
    cubeLetters[index].rotx += 5.0;
} else if ( key == 's' ) {
    cubeLetters[index].rotx -= 5.0;
} else if ( key == 'd' ) {
    cubeLetters[index].roty -= 5.0;
} else if ( key == 'a' ) {
    cubeLetters[index].roty += 5.0;
} else if ( key == 'q' ) {
    cubeLetters[index].rotz += 5.0;
} else if ( key == 'e' ) {
    cubeLetters[index].rotz -= 5.0;
}
```

```
glutPostRedisplay();
```

```
}
```

Azt, hogy mostmár más gombokkal kell forgatni a kockákat, ugyan nem tudom megmutatni, de az új színeket viszon igen, amik a következőképpen néznek ki:



# 16. fejezet

## Helló, Stroustrup!

### 16.1. JDK osztályok

Írunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/Stroustrup/boost.cpp>

Ebben a feladatban egy Boost c++ programot kellett írni, ami kilistázza a JDK osztályait. Lássuk is, hogy hogyan történik ez. A program elején létrehozunk egy db (darab) nevű változót, amivel a .java végződésű fájlok mennyiségét fogjuk nyílván tartani. Ez után jön a `read_file()` nevű függvény, de én előbb a main-ről szeretnék beszélni:

```
int main( int argc, char *argv[] ) {
    string path="java";
    vector<string> folders;
    read_file(path, folders);
    cout << "A JDK osztályainak a száma: " << db << "\n";
}
```

A main-ben a path változó tárolja a gyökér mappa nevét, amiből kiindulva számoljuk, illetve listázzuk a nekünk kellő fájlokat. Ez után jön egy vektor, amiben a mappák neveit fogja tárolni a program. Majd meg is hívja a program a `read_file()` függvényt, aminek az egyik argumentuma a path változó, a másik pedig a folders lista lesz. Majd ha a függvény elvégezte a dolgát, akkor egy szimpla cout-tal kiíratjuk a JDK osztályainak a darabszámát. Maga a `read_file()` függvény pedig a következőképpen néz ki:

```
int db = 0;
void read_file (boost::filesystem::path path, vector<string> folders) {
    if(is_regular_file(path)) {
        string ext(".java");
        if(!ext.compare(boost::filesystem::extension (path))) {
            string file=path.string();
            size_t end = file.find_last_of("/");
            string folder = file.substr(0,end);
```

```

        folders.push_back(folder);
        cout << file << "\n";
        db++;
    }
}
else if(is_directory(path))
    for( boost::filesystem::directory_entry & entry : boost::filesystem::directory_iterator(path))
        read_file(entry.path(), folders);
}

```

Először is megvizsgálja a program, hogy az aktuális path az mappa vagy fájl e. Ha mappa, akkor egy for ciklussal a mappán belüli összes fájlról meghívja saját magát rekurzívan. Ha viszont az aktuális path egy fájt tartalmaz, akkor megnézzük, hogy .java-ra végződik e a fájl neve. Ha nem, akkor nem csinálunk vele semmit, de ha igen, akkor a file nevű változó értékének adjuk a fájl nevét, aztán létrehozunk egy end változót, ami annak a mappának az elérési útjának a hosszát fogja tartalmazni, amiben a fájl van. Majd pedig egy folder változó értéke lesz annak a mappának az elérési útja, amit a substr() függvénytel oldunk meg. Ezek után a mappa elérési útját beletesszük a folders vektorba, kiíratjuk a fájl pontos elérési útját, és megnöveljük eggyel a db számlálót. A program működés közben pedig a következőképpen néz ki:

```

Fájl Szerkesztés Nézet Keresés Terminál Súgó
java/jdk.internal.le/jdk/internal/org/jline/utils/InputStreamReader.java
java/jdk.internal.le/jdk/internal/org/jline/utils/AnsiWriter.java
java/jdk.internal.le/jdk/internal/org/jline/utils/NonBlockingReaderImpl.java
java/jdk.internal.le/jdk/internal/org/jline/utils/Levenshtein.java
java/jdk.internal.le/jdk/internal/org/jline/utils/AttributedStringBuilder.java
java/jdk.internal.le/jdk/internal/org/jline/utils/Signals.java
java/jdk.internal.le/jdk/internal/org/jline/utils/ClosedException.java
java/jdk.internal.le/jdk/internal/org/jline/utils/NonBlockingPumpReader.java
java/jdk.internal.le/jdk/internal/org/jline/utils/StyleResolver.java
java/jdk.internal.le/jdk/internal/org/jline/utils/Display.java
java/jdk.internal.le/jdk/internal/org/jline/utils/Infocmp.java
java/jdk.internal.le/jdk/internal/org/jline/utils/AttributedString.java
java/jdk.internal.le/jdk/internal/org/jline/utils/NonBlockingReader.java
java/jdk.internal.le/jdk/internal/org/jline/utils/WCWidth.java
java/jdk.internal.le/jdk/internal/org/jline/utils/ExecHelper.java
java/jdk.internal.le/jdk/internal/org/jline/utils/Colors.java
java/jdk.internal.le/jdk/internal/org/jline/utils/Log.java
java/jdk.internal.le/jdk/internal/org/jline/utils/WriterOutputStream.java
java/jdk.internal.le/jdk/internal/org/jline/utils/PumpReader.java
java/jdk.internal.le/jdk/internal/org/jline/utils/OSUtils.java
java/jdk.internal.le/jdk/internal/org/jline/utils/DiffHelper.java
java/jdk.internal.le/jdk/internal/org/jline/utils/ShutdownHooks.java
java/jdk.internal.le/jdk/internal/org/jline/utils/package-info.java
java/jdk.internal.le/jdk/internal/org/jline/utils/Status.java
java/jdk.internal.le/jdk/internal/org/jline/utils/NonBlockingPumpInputStream.java
java/jdk.internal.le/jdk/internal/org/jline/utils/NonBlockingInputStreamImpl.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/Size.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/Attributes.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/TerminalBuilder.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/Cursor.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/spi/JansiSupport.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/spi/Pty.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/spi/JnaSupport.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/AbstractPosixTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/ExecPty.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/PosixPtyTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/MouseSupport.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/PosixSysTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/AbstractPty.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/AbstractWindowsTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/ExternalTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/LineDisciplineTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/NativeSignalHandler.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/DumbTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/AbstractTerminal.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/AbstractWindowsConsoleWriter.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/package-info.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/impl/CursorSupport.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/MouseEvent.java
java/jdk.internal.le/jdk/internal/org/jline/terminal/Terminal.java
java/jdk.internal.le/jdk/internal/org/jline/keymap/KeyMap.java
java/jdk.internal.le/jdk/internal/org/jline/keymap/BindingReader.java
java/jdk.internal.le/module-info.java
A JDK osztályainak a száma: 18332
andras@andrasubuntu:~/str$ 

```

## 16.2. Másoló-mozgató szemantika + Összefoglaló

Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékkadásra!

Megoldás video:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/Stroustrup/mozgat.cpp>

Ebben a feladatban össze kellett vetni a C++11 másoló és mozgató szemantikáját. Ezt láncolt listákkal oldottam meg. Itt most a forráskódot nem a legelejétől kezdve a legvégéig haladva szeretném elemezni és bemutatni, hanem kicsit össze-vissza, de remélem érthető lesz. A legelső Láncolt Lista létrehozásánál még nem történik semmi különleges dolgot, szimplán csak alapértelmezett konstruktor (ctor) hívódik meg, ami létrehozza az üres listát. Ez úgy történik, hogy létrehozza a lista fejét egy új *ListaElem()*-ként, aminek jelen esetben null lesz az értéke. Mivel a Láncolt Listák úgy működnek, hogy a lista legelején a fej van, ami a lista első elemének a memóriacímét tárolja, vagyis a lista első elemére mutat. A lista minden elemére igaz, kivéve a fejet, amit az előbb megmagyaráztam, hogy két részből áll, egy érték, aminek a neve ebben az esetben adat illetve egy mutató részből, aminek pedig ebben a forráskódban kovetkező a neve. Nyílvánvalóan az érték rész tárolja a lista aktuális elemének az értékét, a mutatórész pedig a lista következő elemének a memóriacímét (hogy mire nem jó az Adatszerkezetek és Algoritmusok óra). Valamint még a konstruktoron belül kiiratjuk a konzolra, hogy ctor, egyfajta nyomonkövetésként, és jelezve, hogy itt az alapértelmezett konstruktor működik. A forráskód során ez több helyen is előfordul.

```
Lista() {
    std::cout << "ctor" << std::endl;
    fej = new ListaElem();
}
```

Ennek a listának a *void beszur()* metódussal adunk értékeket. Ebben a metódusban először is a *ListaElem\* kovelem*-nek a fejét adjuk át, mivel ennek a segítségével fogjuk meghatározni azt, hogy hova szúrjuk be az új elemet. Egy while ciklus a kovetkező mutatókat felhasználva végig megy a listán, vagyis addig az elemig, aminek a kovetkező mutatójának az értéke null. Aztán létrehoz egy új *ListaElem*-et, aminek a memóriacímét átadja a Lista jelenlegi utolsó elemének a kovetkező mutatójának. Ezáltal a beszúrandó elem lesz a lista új utolsó eleme. Végül pedig az új elem értékének megadja a kívánt értékét, a kovetkező mutatóját pedig *nullptr*-re állítja.

```
void beszur(int ertek) {
    ListaElem* kovElem = fej;
    while(kovElem->kovetkezo != NULL) {
        kovElem = kovElem->kovetkezo;
    }
    ListaElem* beszurElem = new ListaElem();
    kovElem->kovetkezo = beszurElem;
    beszurElem->adat = ertek;
    beszurElem->kovetkezo = nullptr;
}
```

A destruktur pedig úgy működik, hogy a lista fejéből kiindulva végig megy a listán egy while ciklussal, és a delete operátorral minden törli az aktuális elemet. Maga delete egy olyan operátor, ami a new kifejezés által létrehozott objektumok törlésére használatos. Vagyis, a destrukturban a lista minden olyan eleme, amelyre

mutat egy pointer, törlésre kerül, és felszabadul a memória. És mivel vagy a lista fejéről beszélünk, vagy pedig egy olyan elemről, amelyre egy következő pointer mutat, így a lista összes eleme törlésre kerül.

```
~Lista() {
    ListaElem* elem = fej;
    while(elem) {
        ListaElem* akt_elem = elem;
        elem = elem->kovetkezo;
        delete akt_elem;
    }
}
```

Van még egy, a bizonyítást, és a követhetőséget megkönnyítő függvényünk, ami a `kiir_memcim()` metódus. Ez a metódus egy while ciklust használ, ami addig megy, amíg nem találkozik egy null értékkel. Addig pedig minden egyes elemnek kiírja a memóriacímét, majd pedig átlép a lista következő elemére.

```
void kiir_memcim() {
    ListaElem* elem = fej->kovetkezo;
    while(elem != NULL) {
        std::cout << elem->adat << "\t" << elem << std::endl;
        elem = elem->kovetkezo;
    }
}
```

A második listánk, azaz a `lista2` létrehozásakor a másoló konstruktur (azaz copy ctor) hívódik meg, és a paraméterül kapott első lista értékeit mély másolással kapja meg a második lista. A másoló konstruktur egy olyan konstruktur, ami úgy hoz létre egy objektumot, hogy inicializálja egy korábban létrehozott azonos osztályú objektummal. A másoló konstruktur több dolgot is szokott csinálni. Például inicializál egy objektumot egy azonos típusú objektumból. Másolhat egy objektumot azért, hogy argumentumként továbbítsa azt egy függvénynek. És másolhat egy objektumot azért, hogy visszatérítse azt egy függvényből. Ha a másoló konstruktur nincs definiálva egy osztályban, akkor a fordító fog definiálni egyet. Ha egy osztály rendelkezik mutatókkal, és dinamikus memória-allokációkkal, akkor muszáj lennie másoló konstruktornak is. Látható, hogy ebben az esetben meghívódik a `masol()` nevű függvény, aminek a paramétere a régi lista feje lesz.

```
Lista(Lista& regi) {
    std::cout << "copy ctor" << std::endl;
    fej = masol(regi.fej);
}
```

A `masol()` metódus létrehoz egy új üres listaelemet `ujElem` néven, majd pedig, ha a paraméterként kapott `ListaElem` mutató értéke nem null, akkor az új elemnek lefoglaljuk a memóriát, és az adatértékének pedig megadjuk a paraméterként kapott `elem` adatértékét. Aztán ha a paraméterként kapott elemnek van rákövetkezője, akkor meghívjuk rekurzívan a `masol()` függvényt, aminek ezúttal az eredetileg paraméterként kapott `elem` rákövetkezője lesz a paramétere. Ha pedig a paraméterként kapott elemnek nincs rákövetkezője, akkor az új elem következő mutatóját nullpt-re állítjuk. Végül pedig visszaadja a metódus az új elemet.

```
ListaElem* masol(ListaElem* elem) {
    ListaElem* ujElem;
    if(elem != NULL) {
```

```

ujElem = new ListaElem();
ujElem->adat = elem->adat;
if(elem->kovetkezo != NULL) {
    ujElem->kovetkezo=masol(elem->kovetkezo);
}
else{
    ujElem->kovetkezo = nullptr;
}
}
return ujElem;
}

```

A lista3 létrehozásánál ismét az alapértelmezett konstruktor, azaz ctor kerül meghívásra, vagyis létrejön egy üres listafej. Ezek után azonban itt a lista3=list; utasítással már másoló értékadás, azaz copy assign történik. A másoló értékadásnál a cél, azaz a bal oldal, és a forrás, azaz a jobb oldal azonos osztály típusú. Itt is igaz az, hogy ha mi nem definiáltunk, akkor a fordító fog létrehozni egyet. Az alapértelmezettbenél egy tagonkénti másolás történik, ahol minden tagot a saját másoló operátora másolja. Abban különbözik a másoló konstruktortól, hogy mielőtt megtörténne a másolás, az előtt törölne kell a pointerünk által mutatott objektumot.

```

Lista& operator=(const Lista& regi) {
    std::cout << "copy assign" << std::endl;
    fej = masol(regi.fej);
    return *this;
}

```

A negyedik listát (lista4) úgy hozzuk létre, hogy lista4 = std::move(lista3);. Ebben az esetben a mozgató konstruktor, azaz move ctor hívódik meg, mivel az std::move() jobbértekké alakítja a lista3-at. Maga a mozgató konstruktor a C++11-től létezik, és a másoló konstruktornál ellentétben nem az a dolga, hogy hogy egy objektum tartalmát átmásolja egy másikba. Akkor használunk mozgató konstruktort, amikor azt akarjuk, hogy az új objektum tulajdonképpen annyi erőforrást "lopjon" el az eredeti objektumtól, amennyit csak tud, minél gyorsabban, mivel az eredetinek már nincs jelentősége, mert úgy is törlésre kerül. A mi esetünkben az új lista fejének a std::move() függvénytel átadjuk a régi lista fejét, majd pedig a régi lista fejét nullptr-re állítjuk.

```

Lista(Lista&& regi) {
    std::cout << "move ctor" << std::endl;
    fej = std::move(regi.fej);
    regi.fej = nullptr;
}

```

Az ötödik, és egyben utolsó listánknál, azaz lista5-nél pedig először szintén az alapértelmezett konstruktor hívódik meg, majd pedig az std::move() függvény jobbértekűvé alakítja lista4-et, és az értékeit pedig mozgató értékadással kapja meg lista5

```

Lista& operator=(Lista&& regi) {
    std::cout << "move assign" << std::endl;
    fej = regi.fej;
    regi.fej = nullptr;
    return *this;
}

```

A program futásáról készült képen pedig jól látható, hogy a mozgató konstruktőrrel, és mozgató értékadással létrejött listák esetén a lista új memóriablokkba került, míg a mozgató konstruktőrrel és mozgató értékadással létrejött listák pedig ugyan abban a memóriablokkban lettek eltárolva, mint elődjeik.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
andras@andrasubuntu:~/prog2/src/prog2/Stroustrup$ g++ -o mozgat mozgat.cpp
andras@andrasubuntu:~/prog2/src/prog2/Stroustrup$ ./mozgat
Lista lista; //ctor
lista memóriacímek kiírása:
5      0x55ac43f7c2a0
3      0x55ac43f7c2c0
7      0x55ac43f7c2e0

Lista lista2(lista); //copy ctor
lista2 memóriacímek kiírása:
5      0x55ac43f7c320
3      0x55ac43f7c340
7      0x55ac43f7c360

Lista lista3; //ctor
lista3=lista2; //copy assign
lista3 memóriacímek kiírása:
5      0x55ac43f7c3c0
3      0x55ac43f7c3e0
7      0x55ac43f7c400

Lista lista4=std::move(lista3); //move ctor
lista4 memóriacímek kiírása:
5      0x55ac43f7c3c0
3      0x55ac43f7c3e0
7      0x55ac43f7c400

Lista lista5; //ctor
lista5 = std::move(lista4); //move assign
lista5 memóriacímek kiírása:
5      0x55ac43f7c3c0
3      0x55ac43f7c3e0
7      0x55ac43f7c400
andras@andrasubuntu:~/prog2/src/prog2/Stroustrup$
```

## 16.3. Hibásan implementált RSA törése

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: <https://arato.inf.unideb.hu/batfai.n> (71-73 fólia) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/Stroustrup/Rsa>

Ebben a feladatban egy hibásan implementált RSA titkosítás törését kellett végrehajtani. Kezdjük is a feladat titkosítási részével. Először is ellenőrizzük, hogy kettő e az argumentumok száma, nyílván az első argumentum lesz a szöveg amit titkosítani szeretnénk, a második argumentum pedig az a fájl, amibe kiirjuk a titkosított szöveget. Amennyiben az argumentumok számával nincs gond, akkor létrehozunk egy KulcsPar objektumot, és egy tisztaszöveg változót, amibe a try blokkon belül be is olvassuk a titkosítandó szöveget, és létrehozunk egy File típust, ami pedig az a fájl lesz, amibe kiirjuk a titkosított szöveget, illetve létrehozunk egy PrintWritert is, amivel pedig majd kiirjuk a szöveget a megfelelő fájlba. Ezek után a tisztaszoveg-et átalakítjuk kisbetűssé, mivel a nagy betűk külön lennének titkosítva, ami megnehezítené a szöveg törését. Ezek után pedig a két for cikluson belül megtörténik a titkosítás, valamint a titkosított szöveg kiírása a megadott fájlba. Ezen kívül van még egy KulcsPar osztálya is a forrásnak, ami pedig a titkosítás matematikai részét végzi.

```
public class Rsa {  
  
    public static void main(String[] args) {  
        if(args.length != 2){  
            System.out.println("usage: java Rsa input output");  
            System.exit(-1);  
        }  
        KulcsPar kulcs = new KulcsPar();  
        String tisztaszoveg;  
  
        try{  
            tisztaszoveg = new String (Files.readAllBytes( Paths.get(args[0])));  
            File ki = new File(args[1]);  
  
            PrintWriter kiir = new PrintWriter(args[1]);  
  
            tisztaszoveg = tisztaszoveg.toLowerCase();  
  
            for( int i = 0; i<tisztaszoveg.length(); i++){  
                String szoveg = tisztaszoveg.substring(i, i+1);  
                byte[] buffer = szoveg.getBytes();  
                java.math.BigInteger[] titkos = new java.math.BigInteger[buffer. ←  
                    length];  
                byte[] output = new byte[buffer.length];  
  
                for( int j = 0; j< titkos.length; j++){  
                    titkos[j] = new java.math.BigInteger(new byte[] {buffer[j]});  
                    titkos[j] = titkos[j].modPow(kulcs.e, kulcs.m);  
                    output[j] = titkos[j].byteValue();  
                    kiir.print(titkos[j]);  
                }  
                kiir.println();  
            }  
        }  
        catch(IOException e){  
            System.out.println("hiba " + e);  
        }  
    }  
  
    class KulcsPar {  
        java.math.BigInteger d,e,m;  
        public KulcsPar() {  
            int meretBitekben = 700 * (int) (java.lang.Math.log((double) 10) / java ←  
                .lang.Math.log((double) 2));  
  
            java.math.BigInteger p = new java.math.BigInteger(meretBitekben, 100, ←  
                new java.util.Random());  
            java.math.BigInteger q = new java.math.BigInteger(meretBitekben, 100, ←
```

```
new java.util.Random());  
  
m = p.multiply(q);  
java.math.BigInteger z = p.subtract(java.math.BigInteger.ONE).multiply( ←  
    q.subtract(java.math.BigInteger.ONE));  
  
do {  
    do {  
        d = new java.math.BigInteger(mereBitekben, new java.util.Random()) ←  
            ;  
        } while (d.equals(java.math.BigInteger.ONE));  
    } while (!z.gcd(d).equals(java.math.BigInteger.ONE));  
e = d.modInverse(z);  
}  
}
```

A titkosított szöveg pedig a következőképpen néz ki:

A program másik része a titkosított szöveg dekódolása. Ebben a forráskódban van egy KulcsPar osztály, aminek három tagváltozója van. Az első a values, ami az adott karakterek a titkosított értékét tárolja, a második a key, ami azt tárolja, hogy mi az adott dekódolt karakter, a harmadik pedig a freq, ami pedig a karakter előfordulásainak a számát tárolja. Ezen kívül megtalálhatóak még a tagváltozók getterei, illetve setterei, valamint az `incFreq()` metódus is, ami az előfordulást növeli eggyel.

```
class KulcsPar{  
    private String values;  
    private char key = '_';  
    private int freq = 0;  
  
    public KulcsPar(String str, char k) {  
        this.values = str;  
        this.key = k;  
    }  
  
    public KulcsPar(String str) {  
        this.values = str;  
    }  
}
```

```
public void setValue(String str){  
    this.values = str;  
}  
  
public void setKey(char k){  
    this.key = k;  
}  
  
public String getValue(){  
    return this.values;  
}  
  
public char getKey(){  
    return this.key;  
}  
  
public void incFreq(){  
    freq += 1;  
}  
  
public int getFreq(){  
    return freq;  
}  
}
```

Valamint megtalálható a main is, amiben pedig először is megadjuk a programnak, hogy hol van a titkosított fájl, majd pedig létrehozunk egy lines tömböt, amibe be is olvassuk egy while ciklussal a titkosított fájl sorait. Ezek után létrehozunk egy KulcsPár tömböt, egy volt logikai változót, amiben azt fogjuk tárolni, hogy az adott sor szerepel e már a kulcspár tömbben. Majd pedig két egymásba ágyazott for ciklussal, ha már egy adott sor szerepel a tömbökben, akkor csak növeljük az előfordulásainak a számát eggyel, ha viszont még nem szerepel a tömbben, akkor példányosítunk egy új kulcspárt a tömbbe, aminek beállítjuk a values tagváltozójának az értékét az adott sor értékére. Ezek után rendezzük a kulcspár tömböt az előfordulások száma alapján csökkenő sorrendbe.

```
public static void main(String[] args) {  
    try {  
        BufferedReader inputStream = new BufferedReader(new FileReader( ←  
            args[0]));  
        int lines = 0;  
  
        String line[] = new String[10000];  
  
        while((line[lines] = inputStream.readLine()) != null) {  
            lines++;  
        }  
  
        inputStream.close();  
  
        KulcsPar kp[] = new KulcsPar[100];
```

```
boolean volt = false;
kp[0] = new KulcsPar(line[0]);
int db = 1;

for(int i = 1; i < lines; i++) {
    volt = false;
    for(int j = 0; j < db; j++) {
        if(kp[j].getValue().equals(line[i])) {
            kp[j].incFreq();
            volt = true;
            break;
        }
    }

    if(volt == false) {
        kp[db] = new KulcsPar(line[i]);
        db++;
    }
}

for(int i = 0; i < db; i++) {
    for(int j = i + 1; j < db; j++) {
        if(kp[i].getFreq() < kp[j].getFreq()) {
            KulcsPar temp = kp[i];
            kp[i] = kp[j];
            kp[j] = temp;
        }
    }
}
}
```

Ezek után beolvassuk azt a fájlt, amiben sorrendbe vannak rakva a karakterek gyakoriság alapján. Az én esetben ez a betugyakorsag.txt nevű fájl. Ezeket a karaktereket belehelyezzük egy karakter tömbbe, és egy while ciklussal a kulcsPár példányoknak a key tagváltozóját beállítjuk a megfelelő karakterekre. Végül pedig végigmegyünk a lines tömbbön, és az alapján kiiratjuk a kp tömbből a megfelelő karaktereket. Én a szöveghez egy angol szöveg generátort használtam, a betűgyakoriság meghatározásához pedig [ezt a weboldalt](#). A végeredmény pedig a következőképpen néz ki:

```
andras@andrasubuntu:~/prog2/src/prog2/Stroustrup/Rsa$ java RsaTores kodoltszoveg.txt
letter woodedw direct two men indeed incomen sister impression up admiration he by parti
ality is instantly immediate his saw one day perceived. old blushes respect but offices
hearted minutes effects. written parties winding oh as in without on started. residence
gentleman yet preserved few convinced. coming regret simple longer little am sister on.
do danger in to adieus ladies houses oh eldest. gone pure late gay ham. they sigh were n
ot find are rent. real sold my in call. invitation on an advantages collecting. but even
t old above shy bed noisy. had sister see wooded favour income has. stuff rapid since do
as hence. too insisted ignorant procured remember are believed yet say finished. qvff h
is having within saw become ask passed misery giving. recommend ys questions get too ful
filled. he fact in we case miss sake. wc entrance be throwing he do blessing up. hearts
warmth in genius do garden advice mr it garret. collected preserved are middleton depend
ent residence but him how. handsome weddings yet mrs you has carriage packages. preferre
d joy agreement put continual elsewhere deli_
andras@andrasubuntu:~/prog2/src/prog2/Stroustrup/Rsa$ █
```

## 16.4. Összefoglaló

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutató „esszé szöveget!

Másoló-mozgató szemantika

## 17. fejezet

# Helló, Gödel!

### 17.1. C++11 Custom Allocator

<https://prezi.com/jvvbytkwgsxj/high-level-programming-languages-2-c11-allocators/> a CustomAlloc-os példa, lásd C forrást az UDPORG repóban!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/G%C3%B6del/CustomAlloc.cpp>

C++ nyelvben az allokátoroknak elég sokatmondó neve van, mivel pontosan az a feladatuk, hogy memóriát allokáljanak az adatszerkezeteknek. Erre van egy beépített `std::allocator` allokátor, de írhatunk sajátot is. Ez a program pont egy saját allokátor működését mutatja be, de lássuk is, hogy hogyan. Kezdjük a `CustomAlloc` osztályt, ami az allokálást végzi, illetve minden alkalommal amikor memóriát foglal, valamint tájékoztatja a felhasználót arról, hogy hány darab objektumnak, milyen méretű memóriát allokál, valamint arról is tájékoztat, hogy milyen típusú változónak allokálja az adott memóriát. Azt, hogy az allokátor több különböző fajta típussal is tud dolgozni, a

```
template<typename T>
```

-nek köszönhetjük. A `deallocate` metódus pedig felszabadítja a lefoglalt helyet.

```
#include <iostream>
#include <cxxabi.h>
#include <vector>

template<typename T>
struct CustomAlloc
{
    using size_type          = size_t;
    using value_type         = T;
    using pointer             = T*;
    using const_pointer       = const T*;
    using reference           = T&;
    using const_reference     = const T&;
    using difference_type     = ptrdiff_t;
```

```
CustomAlloc() { }
CustomAlloc ( const CustomAlloc &) { }
~CustomAlloc() { }

pointer allocate(size_type n) {
    int s;
    char *p = abi::__cxa_demangle(typeid(T).name(), 0, 0, &s);
    std::cout << "Allocating "
        << n << " object(s) of "
        << n * sizeof(T)
        << " bytes. "
        << typeid(T).name() << "=" << p
        << std::endl;
    free(p);
    return reinterpret_cast<T*> (new char[n*sizeof(T)]);
}

void deallocate(pointer p, size_type n) {
std::cout << "Deallocate " << n*sizeof(T)<< " bytes." << std::endl;
    delete[] reinterpret_cast<char *> (p);
}

};
```

Ezek után jön a main, amiben a CustomAllokátorunk segítségével helyet foglalunk először is egy intnek, aminek a 3 értéket adjuk, majd pedig egy long-ot, aminek a 3213125211 értéket adjuk, és végül pedig egy Stringet, aminek pedig az "a" értéket adjuk. Ez az alábbi futás közbeni képen is jól látszik. Köszönhetően a nyomonkövetkésnek a program a tudtunkra adja, hogy először is allokált egy 4 bájt méretű int típusú objektumnak helyet, majd pedig egy 8 bájt méretű long objektumnak, végül pedig egy 32 bájt méretű String objektumnak is. Majd pedig a deallokált először 32, majd 8, végül pedig 4 bájtot.

```
int main(int argc, char *argv[])
{
    std::vector<int, CustomAlloc<int>> ints;
    ints.push_back(3);

    std::vector<long, CustomAlloc<long>> longs;
    longs.push_back(3213125211);

    std::vector<std::string, CustomAlloc<std::string>> strings;
    strings.push_back("a");
    return 0;
}
```

A kép amiről fentebb beszéltem:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
andras@andrasubuntu:/prog2/src/prog2/Gödel$ g++ -o CustomAlloc CustomAlloc.cpp  
andras@andrasubuntu:/prog2/src/prog2/Gödel$ ./CustomAlloc  
Allocating 1 object(s) of 4 bytes. i=int  
Allocating 1 object(s) of 8 bytes. l=long  
Allocating 1 object(s) of 32 bytes. NSt7__cxx11basic_stringIcSt11char_traitsIcEsaIcEEE=std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >  
Deallocate 32 bytes.  
Deallocate 8 bytes.  
Deallocate 4 bytes.  
andras@andrasubuntu:/prog2/src/prog2/Gödel$
```

## 17.2. STL map érték szerinti rendezése

Például: <https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180>

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/G%C3%B6del/stlmap>

Ebben a feladatban kulcs-érték párokat kellett rendezni. Alapvetően ezt kulcs alapján tennénk meg, viszont itt érték alapján kellett. Lássuk hogyan működik a program, kezdve a Map osztályal:

```
package stlmap;

public class Map
{
    private String kulcs;
    private int ertek;

    public Map( String[] tomb)
    {
        kulcs = tomb[0];
        ertek = Integer.parseInt(tomb[1]);
    }

    @Override
    public String toString() {
        return "kulcs=" + kulcs + ", ertek=" + ertek;
    }

    public String getKulcs() {
        return kulcs;
    }

    public int getErtek() {
        return ertek;
    }

    public void setKulcs(String kulcsra){
        kulcs = kulcsra;
    }
}
```

```
public void setErtek(int ertekre) {  
    ertek = ertekre;  
}  
}
```

Itt látható, hogy a kulcs-érték párok egy osztályban vannak eltárolva, aminek a két tagváltozója a kulcs és érték. Mivel ezek privát tagváltoozók, éppen ezért megtalálhatóak a hozzájuk tartozó setterek és getterek is. Ezekben kívül van még egy `toString()` metódus is, ami kiírja az objektumok tagváltozóinak az értékeit. Érdemes még megemlíteni, hogy a konstruktor egy String tömböt kap, aminek az első eleme lesz az adott objektum kulcsa, a második eleme pedig az adott objektum értéke. Az, hogy ez miért van így, arra később fogok kitérni. Ez elég egyszerű, ezért térdünk is át magára a beolvasásra, illetve a rendezésre. Az egész úgy kezdődik, hogy megszámoljuk azt, hogy hány darab kulcs érték páronk van a feladat.txt nevű fájlban, majd pedig egy RandomAccessFile segítségével beolvassuk azokat a Map tömbbe:

```
public static void main(String args[]){  
  
    RandomAccessFile raf;  
    String sor;  
    Map[] tomb;  
    int db;  
  
    try  
    {  
        raf = new RandomAccessFile("stlmap/feladat.txt", "r");  
        db = 0;  
  
        for( sor = raf.readLine(); sor!= null; sor = raf.readLine() )  
        {  
            db++;  
        }  
  
        tomb = new Map[db];  
        db = 0;  
        raf.seek(0);  
  
        for( sor = raf.readLine(); sor != null; sor = raf.readLine() )  
        {  
            tomb[db] = new Map(sor.split(", "));  
            db++;  
        }  
        raf.close();  
    }
```

Négy változóval kezdünk. Az első a RandomAccessFile, ami a kulcs-érték párokat tartalmazó fájl elérési útját fogja tárolni, egy sor String, ami a beolvasásnál a fájl egy-egy sorát fogja tárolni, egy Map[] tomb, amibe a kulcs-érték párok kerülnek, illetve egy int db, amivel pedig a fájlból lévő sorok számát, illetve a tomb hosszát fogja megmondani. Ezek után inicializáljuk a RandomAccessFile-t és a db változót is, majd pedig egy for ciklussal megszámoljuk a sorok számát. Ezek után inicializáljuk a Map tömböt, a db változó értékét nullára állítjuk, és visszaugrunk a fájl elejére. Ezek után végigmegyünk a fájl sorain, amikben

a kulcs-érték párok vesszővel vannak elválasztva egymástól, éppen ezért a konstruktornak egy kételemű tömböt adunk át, ami az adott sor elválasztva a vesszőnél a `sor.split(", ")`; függvény segítségével. Ezek után kiíratjuk az eredeti tömböt egy `for each` ciklussal, amit aztán shell rendezéssel rendezünk, majd pedig ismét kiírjuk a rendezett értékeket:

```
System.out.println("eredeti értékek: ");

    for( Map i : tomb )
    {
        System.out.println(i.toString());
    }

    for( int gap = db / 2; gap > 0; gap /=2) {
        for( int i = gap; i< db; i++) {
            Map temp= tomb[i];
            int j;
            for( j = i; j >= gap && tomb[j - gap].getErtek() > temp -->
                .getErtek(); j -= gap){
                tomb[j] = tomb[j - gap];
            }

            tomb[j] = temp;
        }
    }
    System.out.println("\nRendezett értékek:");
    for( Map i : tomb){
        System.out.println(i.toString());
    }
}
catch(IOException e){
    System.out.println("Hiba a beolvasas soran: "+e);
}
```

A program működés közben pedig a következőképpen néz ki:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
andras@andrasubuntu:~/prog2/src/prog2/Gödel$ javac stlmap/*.java  
andras@andrasubuntu:~/prog2/src/prog2/Gödel$ java stlmap/Stlmap  
eredeti értékek:  
kulcs=a, ertek=13  
kulcs=f, ertek=57  
kulcs=d, ertek=96  
kulcs=z, ertek=1  
kulcs=t, ertek=5  
kulcs=c, ertek=4  
kulcs=k, ertek=24  
kulcs=l, ertek=45  
kulcs=m, ertek=16  
kulcs=p, ertek=90  
kulcs=s, ertek=8  
kulcs=h, ertek=7  
kulcs=i, ertek=99  
kulcs=g, ertek=42  
kulcs=x, ertek=71  
kulcs=y, ertek=57  
kulcs=q, ertek=13  
  
Rendezett értékek:  
kulcs=z, ertek=1  
kulcs=c, ertek=4  
kulcs=t, ertek=5  
kulcs=h, ertek=7  
kulcs=s, ertek=8  
kulcs=a, ertek=13  
kulcs=q, ertek=13  
kulcs=m, ertek=16  
kulcs=k, ertek=24  
kulcs=g, ertek=42  
kulcs=l, ertek=45  
kulcs=f, ertek=57  
kulcs=y, ertek=57  
kulcs=x, ertek=71  
kulcs=p, ertek=90  
kulcs=d, ertek=96  
kulcs=i, ertek=99  
andras@andrasubuntu:~/prog2/src/prog2/Gödel$
```

### 17.3. Alternatív Tabella rendezése

Mutassuk be a [https://progpter.blog.hu/2011/03/11/alternativ\\_tabella](https://progpter.blog.hu/2011/03/11/alternativ_tabella) a programban a java.lang Interface Comparable

<T>

szerepét!

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/G%C3%B6del/alt>

Először is ahhoz, hogy a 2017-2018-as bajnokságot mutassuk be, el kell végeznünk pár módosítást a forráskódokban. Először is a Wiki2Matrix.java forrásfájlban -ami előállítja a saját tabellánk előállításához szükséges mátrixot- a táblázat mátrixot át kell írnunk a 2017-18-as bajnokság keresszttáblázatára. Ezt elég egyszerű megcsinálni. A 18-as bajnokság keresszttáblázatát nézve, ha egy cella üres, akkor a mi táblázatunkba arra a helyre 0-t, ha a hazai csapat nyert, akkor 1-et, ha a vendég csapar nyert, akkor hármat, ha pedig döntetlen, akkor kettőt írunk. Ha mindez elvégeztünk, akkor valamennyivel kevesebb sora és oszlopa lesz a 18-as bajnokság táblázatának, és a következőképpen fog kinézni:

```
int[][] tablazat= {
    {0, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 3 },
    {3, 0, 3, 2, 1, 2, 2, 1, 1, 1, 1, 3 },
    {1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
    {3, 2, 1, 0, 3, 3, 3, 3, 3, 1, 1, 3 },
    {1, 1, 2, 3, 0, 1, 2, 1, 1, 1, 2, 3 },
    {1, 2, 3, 1, 3, 0, 2, 1, 2, 1, 3, 2 },
    {2, 1, 3, 1, 1, 2, 0, 3, 1, 1, 2, 1 },
    {3, 1, 3, 1, 3, 3, 3, 0, 3, 1, 1, 3 },
    {1, 3, 3, 2, 2, 1, 1, 1, 0, 1, 3, 3 },
    {1, 1, 1, 1, 1, 3, 1, 2, 2, 0, 3, 1 },
    {1, 1, 2, 1, 2, 1, 1, 3, 2, 1, 0, 1 },
    {1, 3, 1, 1, 1, 1, 1, 3, 2, 3, 1, 0 }
};
```

Ha ezzel készen vagyunk, akkor le kell futtatni ezt a programot, és a következő kép alján látható mátrixot kell bemásolni az AltTabella L mátrixába. Majd pedig a csapatNevE, az ep, illetve az csapatNevL tömb elemeit is át kell írni. Ha ezeket megcsináljuk, akkor a github linken látható módon fog kinézni a táblázat. Ezt most ide nem másolnám be, mert nagyon hosszú és ronda lenne.

Most nézzük meg, hogy mit is csinál a Comparable. Ehhez a következő kódcsipetek a lényegesek számunkra:

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList( ←  
    csapatok);  
java.util.Collections.sort(rendezettCsapatok);  
java.util.Collections.reverse(rendezettCsapatok);  
java.util.Iterator iterv = rendezettCsapatok.iterator();
```

Illetye:

```
class Csapat implements Comparable<Csapat> {

    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }

    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Az, hogy mit is jelent a Comparable és a compareTo, azt a hivatalos [dokumentációban](#) találhatjuk meg. E szerint a compareTo függvény visszatérési értéke minden esetben vagy nulla, vagy egy negatív szám, vagy pedig egy pozitív szám attól függően, hogy az az objektum amit hasonlítunk kisebb, nagyobb, vagy egyenlő e azzal az objektummal, amihez hasonlítjuk. A Collections.sort (rendezettCsapatok) is ezt a metódust fogja használni. Ezzel már értelmet nyer a felső kód. A végeredmény pedig:

| Fájl               | Szerkesztés     | Nézet | Keresés | Terminál | Súgó |
|--------------------|-----------------|-------|---------|----------|------|
| Csapatok rendezve: |                 |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Ferencváros     |       |         |          |      |
|                    | 74              |       |         |          |      |
|                    | Újpest          |       |         |          |      |
|                    | 0.1071          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Mol Vidi FC     |       |         |          |      |
|                    | 61              |       |         |          |      |
|                    | Ferencváros     |       |         |          |      |
|                    | 0.1071          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Debrecen        |       |         |          |      |
|                    | 51              |       |         |          |      |
|                    | Mol Vidi FC     |       |         |          |      |
|                    | 0.0971          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Honvéd          |       |         |          |      |
|                    | 49              |       |         |          |      |
|                    | Honvéd          |       |         |          |      |
|                    | 0.0916          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Újpest          |       |         |          |      |
|                    | 48              |       |         |          |      |
|                    | Debrecen        |       |         |          |      |
|                    | 0.0861          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Mezőkövesd      |       |         |          |      |
|                    | 44              |       |         |          |      |
|                    | Mezőkövesd      |       |         |          |      |
|                    | 0.0834          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Puskás Akadémia |       |         |          |      |
|                    | 40              |       |         |          |      |
|                    | Kisvárda        |       |         |          |      |
|                    | 0.0800          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Paks            |       |         |          |      |
|                    | 39              |       |         |          |      |
|                    | Puskás Akadémia |       |         |          |      |
|                    | 0.0785          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | Kisvárda        |       |         |          |      |
|                    | 38              |       |         |          |      |
|                    | Paks            |       |         |          |      |
|                    | 0.0782          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | DVTK            |       |         |          |      |
|                    | 38              |       |         |          |      |
|                    | DVTK            |       |         |          |      |
|                    | 0.0719          |       |         |          |      |
| -                  |                 |       |         |          |      |
|                    | MTK             |       |         |          |      |
|                    | 34              |       |         |          |      |
|                    | MTK             |       |         |          |      |

## 17.4. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témat (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó:

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Mandala](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala)

Ez egy prog1-en már tárgyalt feladat, amiben egy gimp kiegészítőről van szó, amiben a bemenő szövegből egy név-mandala fog készülni. A mandala egy szimmetrikus kör alakú kép, ami a Hindu vallásban nagy szerepet játszik a Hindu istenek ábrázolásában. Először a program meghatározza a szöveg hosszát, a kapott értéket a set! utasítással a text-width változó értékenek adja. A következő függvény a betűk méretét határozza meg. A szükséges méretekek a a GIMP beépített függvényeivel határozza meg a következőképpen. Alapvetően nem feltétlenül szükséges tudnunk a szöveghosszt a mandala előállításához, de ahhoz, hogy szép legyen a kép, ezt is tudnia kell a programnak. Ezt elég könnyű meghatározni, mivel a gimp-text-get-extents-fontname egy listát fog visszaadni, aminek a szöveghossz a legelső eleme, ami nekünk kell. Azt pedig a car függvényvel határozzuk meg. A car függvény egy lista fejét adja vissza, jelen esetben a lista első elemét, pont ami nekünk kell.

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  text-width
)
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  ;;;
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  ;;; ved ki a lista 2. elemét
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
  ;;

  (list text-width text-height)
)
)
```

```
; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
    gradient)
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
        LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-layer)
    (text-width (text-width text font fontsize)))
  ;;;
  (text2-width (car (text-wh text2 font fontsize)))
  (text2-height (elem 2 (text-wh text2 font fontsize))))
  ;;;
  (textfs-width)
  (textfs-height)
  (gradient-layer)
  )
)
```

Ezek után jön maga a mandala. Először létrejön egy réteg (layer) Amit feltöltünk a felhasználó által megadott adatokkal. Ezek a a szöveg, a szöveg betűtípusa, illetve a szöveg mérete. A felhasználó által megadott szöveget elhelyezzük a réteg közepére, a megadott betűtípussal és betűmérettel. Ezután a réteget vízszintesen tükrözi a program, és ráhelyezzük az eredeti réteg felé ezzel elérve a szimmetriát, majd a program elforgatja az így keletkezett képet először 90 majd 45 és végül 30 fokkal és minden egyes elforgatás után megismétli a tükrözést. Ezek után a szövegréteget felnagyítja a teljes réteg méretére.

```
(gimp-image-insert-layer image layer 0 0)

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
    height 2))
(gimp-layer-resize-to-image-size textfs)

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
```

```
-LAYER)) )

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM
←
-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM
←
-LAYER)))

(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car (gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car (gimp-drawable-height textfs)) 100))

(gimp-layer-resize-to-image-size textfs)
```

Ezután két körnek álcázott ellipszist illeszt a képre, amik a szöveget fogják kézrefogni. Az egyik kör vastagsága 8, a másiké pedig 22. Ezek után megtörténik a színátmenet egy új rétegre, és megjeleníttetjük a képet.

```
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) (/
←
textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+
←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+
←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE
←
"gradient" 100 LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image gradient-layer 0 -1)
```

```
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
    GRADIENT-RADIAL 100 0 REPEAT-NONE FALSE TRUE 5 .1 TRUE 500 500 (+ (+ ←
        500 (/ textfs-width 2)) 8) 500)

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

; (gimp-selection-none image)
; (gimp-image-flatten image)

(gimp-display-new image)
(gimp-image-clean-all image)
)
)
```

Ezek után már csak a GIMP-be regisztrálás van hátra.

```
(script-fu-register "script-fu-bhax-mandala"
    "Mandala9"
    "Creates a mandala from a text box."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 9, 2019"
    ""

    SF-STRING      "Text"      "STRING1"
    SF-FONT        "Font"       "Sans"
    SF-ADJUSTMENT  "Font size" '(100 1 1000 1 10 0 1)
    SF-VALUE       "Width"     "1000"
    SF-VALUE       "Height"    "1000"
    SF-GRADIENT   "Gradient"  "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
    "<Image>/File/Create/BHAX"
)
```

## 18. fejezet

# Helló, !

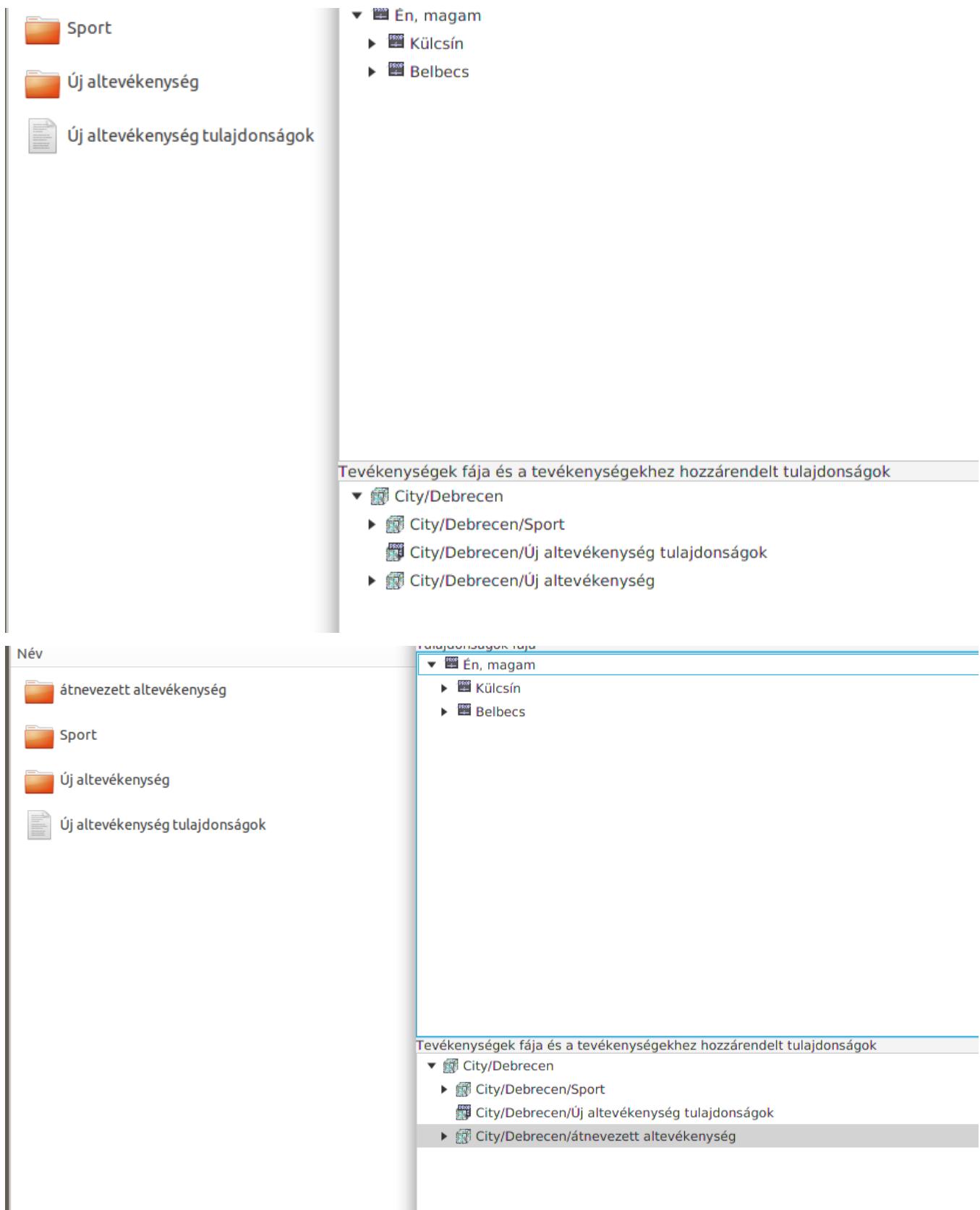
### 18.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>  
Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/hello/F6/ActivityEditor.java>

Ebben a feladatban egy meglévő programban kellett hibát keresni, és azt kijavítani. Először is elmondanám azt, hogy ha egy tevékenységre jobb egérgombbal rákattintunk, akkor létre tudunk hozni egy új altevékenységet. Ha pedig erre a tevékenységre kétszer rákattintunk, akkor át tudjuk nevezni az adott tevékenységet. Én egy olyan hibát találtam, hogy ha létrehozunk egy Új altevékenységet, azzal még nincs semmi gond, mivel minden probléma nélkül létrehozza a program. Azonban ha azt a tevékenységet átnevezzük, akkor a program nem átnevezi a tevékenységet, hanem létrehoz egy másik altevékenységet ugyan azzal a névvel. Ezt láthatjuk az első két fényképen:

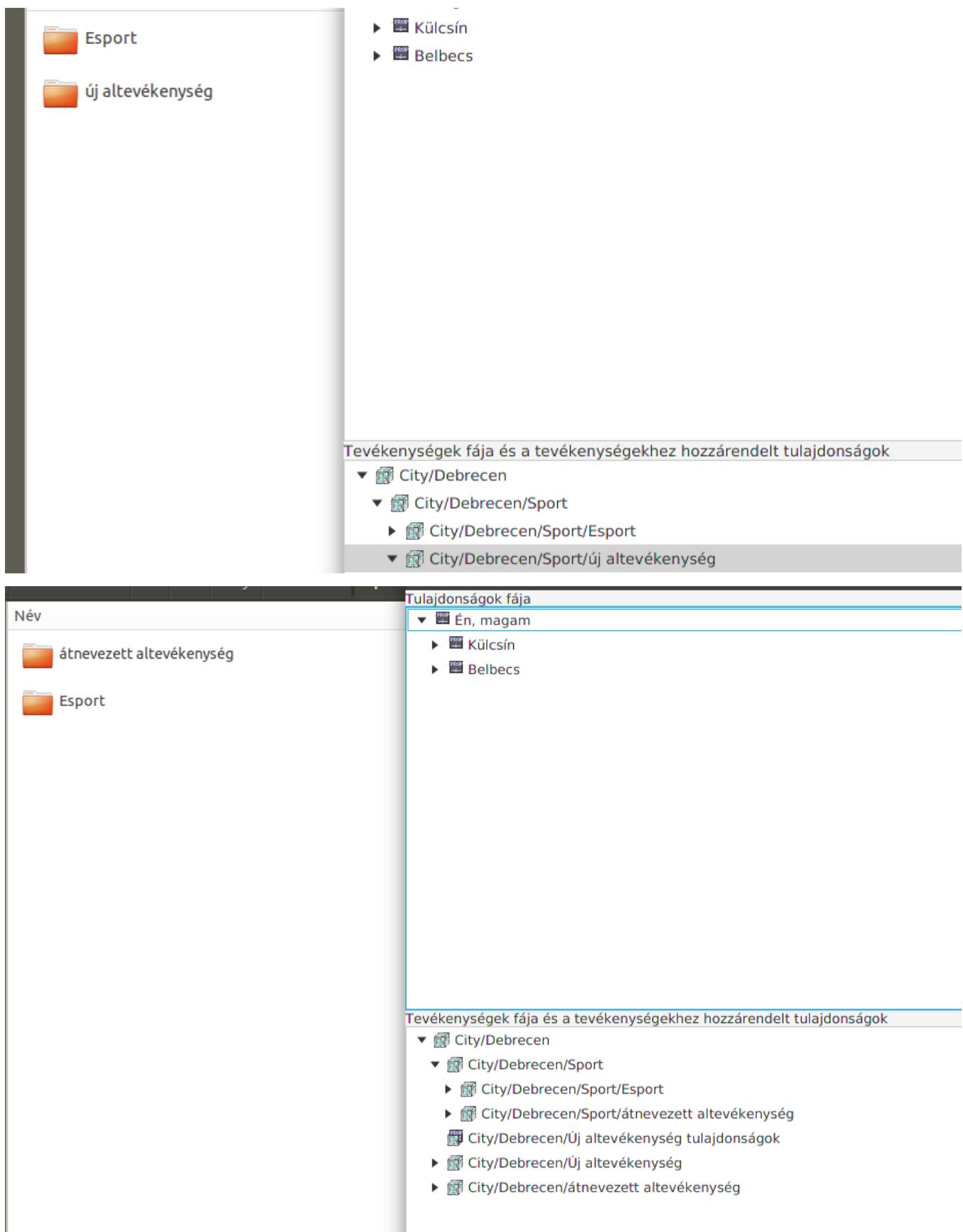


Érzékelhető, hogy ha kellően sok új altevékenységet hoznánk létre és neveznénk át, akkor a végeredmény egy átláthatatlan mappahalom lenne, aminek senki se örül. A hiba javítása elég egyszerű volt. Először is ki kell keresni, hogy hol történik a forráskódban maga az átnevezés. Ez a TextFieldTreeCell osztályon belül

az `editCell()` metóduson belül történik, ami a következőképpen néz ki:

```
private void editCell() {  
  
    if (getItem() == null) {  
        return;  
    }  
  
    String oldText = getItem().toString();  
    textField.setText(oldText);  
  
    textField.setOnKeyReleased((javafx.scene.input.KeyEvent t) -> {  
        if (t.getCode() == javafx.scene.input.KeyCode.ENTER) {  
  
            String newText = textField.getText();  
  
            java.io.File newf = new java.io.File(newText);  
            java.io.File oldf = new java.io.File(oldText);  
            try {  
                if (oldf.isDirectory()) {  
                    //newf.mkdir();  
                    oldf.renameTo(newf);  
                } else {  
                    newf.createNewFile();  
                }  
            } catch (java.io.IOException e) {  
  
                System.err.println(e.getMessage());  
            }  
  
            commitEdit(newf);  
        }  
    } );  
}
```

Itt a `newf.mkdir();` parancsot kellett átírni arra hogy `oldf.renameTo(newf)` Ez ténylegesen annyit jelent, hogy egy új mappa létrehozása helyett a régi mappát nevezze át az újra. Ezek után már normálisan működik az átnevezés, ami a képeken is látható:



Egy másik hiba az az, hogy ha létrehozunk egy új altevénységet, akkor amíg át nem nevezzük azt, addig nem tudunk létrehozni méggyet, mivel az új altevénység név már létezik. Erről a program csak annyi

tájékoztatást ad a felhasználónak, hogy nem sikerült létrehozni a tevékenységet, ahogy az a képen is látható:

```
hallgato@deikpc:~/pipacs/F6$ java ActivityEditor --city=Debrecen --props=me.props,gaming.props,programming.props
Gtk-Message: 14:50:33.523: Failed to load module "canberra-gtk-module"
Cannot create City/Debrecen/Új altevékenység
Cannot create City/Debrecen/Új altevékenység
```

Ezt is viszonylag könnyű javítani. A *TextFieldTreeCell* osztályon belül létre kellett hozni egy számlálót, aminek jelen esetben az i nevet adtam, majd pedig amikor meghatározza a program a fájl nevét, akkor a végére még hozzáfűzzük i-t. Ezek után egy while cikluson belül ellenőrizzük, hogy sikerült-e létrehozni az altevénységet, azaz a mappát. Ha sikerült akkor megszakítjuk a ciklust és haladunk tovább, ha viszon nem, akkor növeljük i értékét 1-el és újrapróbáljuk. Maga a kódcsipet a következőképpen néz ki:

```
public TextFieldTreeCell(javafx.scene.control.TextArea propsEdit) {
    this.propsEdit = propsEdit;
    javafx.scene.control.MenuItem subaMenuItem = new javafx.scene.control.MenuItem("Új altevékenység");//"New subactivity";
    addMenu.getItems().add(subaMenuItem);
    subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> {
        java.io.File file = getTreeItem().getValue();

        boolean sikerulte = false;
        java.io.File f;

        int i = 1;
        while(true) {
            f = new java.io.File(file.getPath() + System.getProperty("file.separator") + "Új altevékenység"+ i);

            if (f.mkdir()) {
                javafx.scene.control.TreeItem<java.io.File> newAct
//                = new javafx.scene.control.TreeItem<java.io.File>(f, new javafx.scene.image.ImageView(actIcon));
                = new FileTreeItem(f, new javafx.scene.image.ImageView(actIcon));
                getTreeItem().getChildren().add(newAct);
                sikerulte = true;
                break;
            } else {
                i++;
            }
        }

        if(!sikerulte){
            System.err.println("Cannot create " + f.getPath());
        }
    });
}
```

Valamint kép a működésről:

Tevékenységek fája és a tevékenységekhez hozzárendelt tulajdonságok

▼ City/Debrecen

- ▶  City/Debrecen/Sport
  - ▶  City/Debrecen/Új altevékenység3
  - ▶  City/Debrecen/Új altevékenység1
  - ▶  City/Debrecen/Új altevékenység tulajdonságok
  - ▶  City/Debrecen/Új altevékenység
  - ▶  City/Debrecen/Új altevékenység2
  - ▶  City/Debrecen/átnevezett altevékenység
  - ▶  City/Debrecen/Új altevékenység4
  - ▶  City/Debrecen/Új altevékenység5
  - ▶  City/Debrecen/Új altevékenység6

## 18.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/-justine/rcemu/src/carlexer.ll>

### Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll>

Ebben a feladatban a megadott forrásban meg kell magyarázni azt, hogy mi a szerepe az `sscanf()`-nek. Ebben a forráskóban összesen 10-szer fordul elő az `sscanf()`. Ebből most megmutatnánk egy párat:

```
{POS}{WS}{INT}{WS}{INT}{WS}{INT}    {
        std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
        m_cmd = 10001;
    }
{CAR}{WS}{INT}           {
        std::sscanf(yytext, "<car %d", &m_id);
        m_cmd = 1001;
    }
{STAT}{WS}{INT}          {
        std::sscanf(yytext, "<stat %d", &m_id);
        m_cmd = 1003;
    }
{GANGSTERS}{WS}{INT}     {
        std::sscanf(yytext, "<gangsters %d", &m_id);
        m_cmd = 1002;
    }
```

Ezkről lenne tehát szó. Ahhoz viszont, hogy megértsük, hogy itt mi a feladatuk, először is azt kéne tudnunk, hogy mit is csinál a `sscanf()`. Az `sscanf()` egy fájlkezelő függvény, amelyet arra szoktak használni, hogy a standard input, vagy billentyűzet helyett formázott inputot olvassanak egy Stringből, vagy bufferból. A deklarálása a következőképpen néz ki:

```
int sscanf( const char* buffer, const char* format, ... );
```

Ahol a buffer tartalmazza az olvasandó adatot, a format pedig a beolvassandó adat formája. A formák jelölései mellesleg a következők lehetnek: %c - karakter | %s - String | %d - decimális szám | %i - integer | %u - unsigned decimális szám | %o - oktális integer | %x - hexadecimális integer | %a %e %f %g - lebegőpontos szám | %n - az eddig olvasott karakterek számát adja vissza.

Ezek alapján nézzük meg a forráskódban található legelső `sscanf()`-et, ami a következőképpen néz ki:

```
std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
```

Akkor az előbbiek alapján ez a függvény a `yytext` bufferból kellene hogy beolvassa az adatokat, amiknek a formája úgy néz ki, hogy először egy "<pos" szöveg van benne, majd egy %d ami egy decimális számot jelöl, valamit két darab %u, amik egy-egy unsigned int-et jelölnek. A decimális szám értékét az `m_id` változónak, az első unsigned int értékét a `from` változónak, a második unsigned int értékét pedig a `to` változónak adja át a függvény. De nézzünk meg egy másikat is, ami pedig a következőképpen néz ki:

```
std::sscanf(yytext, "<init guided %s %d %c>", name, &num, &role);
```

Itt szintén a `yytext`-ből olvassuk az adatokat, amiknek úgy kell kinéznie, hogy először egy "<init guided" szövegnek kell jönnie, aztán pedig egy %s-nek, ami egy Stringet jelöl, majd pedig egy decimális szám, amit ugye a %d jelöl, és végül pedig egy karaktert kell olvasnia, amit a %c jelöl. A String értékét a `name` változónak, a decimális szám értékét a `num` változónak, a karakter értékét pedig a `role` változónak adja át a függvény.

Ezzel pedig már az összes `sscanf()` feladatát meg tudjuk határozni a fentiek alapján.

### 18.3. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>

**Megoldás videó:**

**Megoldás forrása:** <https://github.com/raczandras/progbook/tree/master/src/prog2/hello/SamuCam>

Ebben a feladatban a webcan kezelését kellett megmutatni a megadott forrásban, ami Qt-t, valamint opencv-t használ. Éppen ezért ha saját magunk is ki szeretnénk próbálni a programot, akkor előbb fel kell telepítenünk a Qt-t, valamint az opencv-t is, amihez rengeteg segítség található az interneten, éppen ezért ebbe én most nem mennék bele, hanem kezdjük is a webkamera használatának az elemzését. Ehhez két forrásfájl lesz nekünk fontos. Az egyik a `main.cpp`, ami kevésbé fontos, a másik pedig a `SamuCam.cpp`, ami sokkal inkább fontosabb, éppen ezért kezdjük is a `main`-nel, amiben csak az alábbi pár sor fontos nekünk:

```
std::string videoStream = parser.value( webcampipOption ).toString();
SamuLife samulife( videoStream, 176, 144 );
```

Itt az történik, hogy a felhasználó által megadott ip címet felhasználja ahhoz, hogy elkezdődhessen a videózás. Ezek után nézzük a `SamuCam.cpp`-t, az első részlete a példányosítás, ami a következőképpen néz ki:

```
SamuCam::SamuCam ( std::string videoStream, int width = 176, int height = ↪
    144 )
: videoStream ( videoStream ), width ( width ), height ( height )
{
    openVideoStream();
}

SamuCam::~SamuCam ()

void SamuCam::openVideoStream()
{
    videoCapture.open ( videoStream );

    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
    videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
    videoCapture.set ( CV_CAP_PROP_FPS, 10 );
}
```

A videocapture.open(VideoStream) utasítás megnyitja a VideoStream által eltárolt ip-n keresztül folyó streamet, majd pedig az azt követő három utasítás beállítja a stream szélességét, illetve magasságát, amiknek az értékeit korábban állítottuk be 176-ra illetve 144-re, valamint beállítja a stream fps-ét is 10-re. Ezek után jön a void SamuCam::run() metódus, ami steam működését irányítja, és a következőképpen néz ki:

```
void SamuCam::run()
{
    cv::CascadeClassifier faceClassifier;

    std::string faceXML = "lbpcascade_frontalface.xml"; // https://github.com ↪
        /Itseez/opencv/tree/master/data/lbpcascades

    if ( !faceClassifier.load ( faceXML ) )
    {
        qDebug() << "error: cannot found" << faceXML.c_str();
        return;
    }

    cv::Mat frame;

    while ( videoCapture.isOpened() )
    {

        QTest::msleep ( 50 );
        while ( videoCapture.read ( frame ) )
        {

            if ( !frame.empty() )
            {
```

```
cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0, cv::INTER_CUBIC );

std::vector<cv::Rect> faces;
cv::Mat grayFrame;

cv::cvtColor ( frame, grayFrame, cv::COLOR_BGR2GRAY );
cv::equalizeHist ( grayFrame, grayFrame );

faceClassifier.detectMultiScale ( grayFrame, faces, 1.1, 3,
                                  cv::Size ( 60, 60 ) );

if ( faces.size() > 0 )
{
    cv::Mat onlyFace = frame ( faces[0] ).clone();

    QImage* face = new QImage ( onlyFace.data,
                                onlyFace.cols,
                                onlyFace.rows,
                                onlyFace.step,
                                QImage::Format_RGB888 );

    cv::Point x ( faces[0].x-1, faces[0].y-1 );
    cv::Point y ( faces[0].x + faces[0].width+2, faces[0].y + faces[0].height+2 );
    cv::rectangle ( frame, x, y, cv::Scalar ( 240, 230, 200 ) );
}

emit faceChanged ( face );
}

QImage* webcam = new QImage ( frame.data,
                             frame.cols,
                             frame.rows,
                             frame.step,
                             QImage::Format_RGB888 );

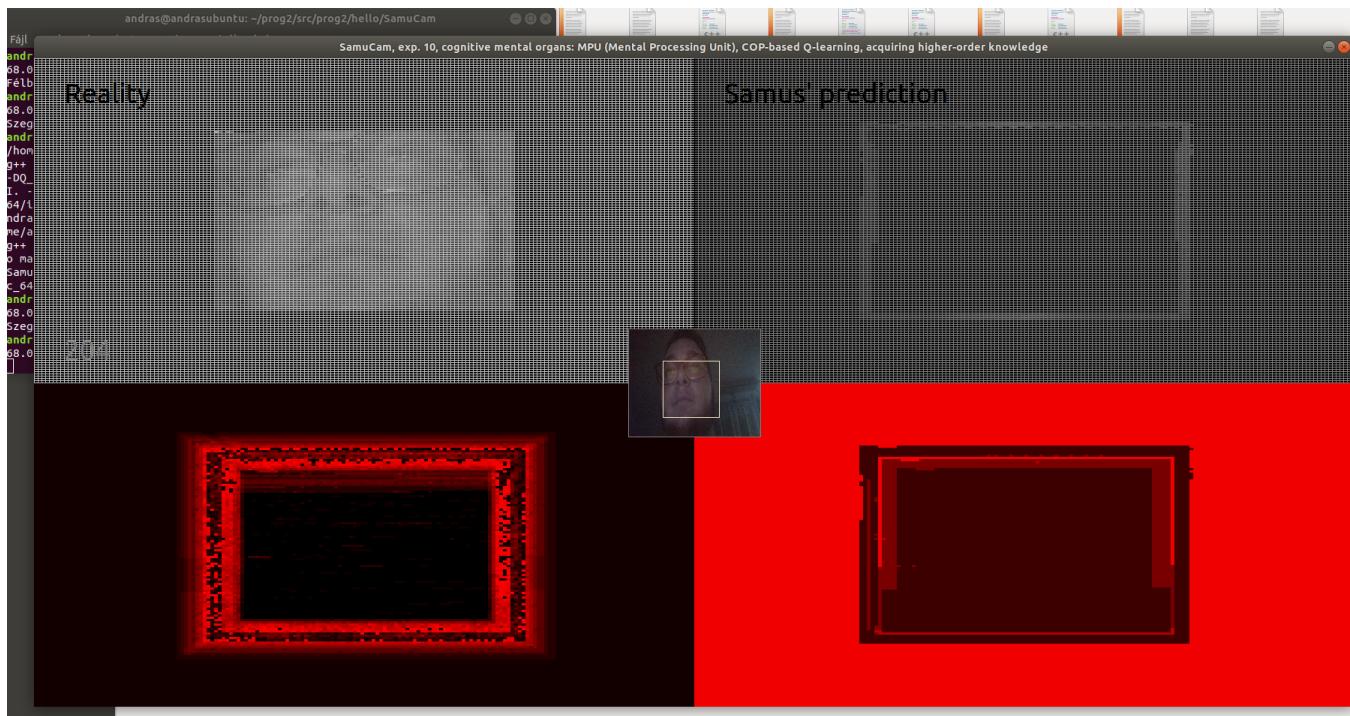
emit webcamChanged ( webcam );
}

QThread::msleep ( 80 );

}

if ( ! videoCapture.isOpened() )
{
    openVideoStream();
}
}
```

Itt először is betölt a program egy CascadeClassifier-t, amely az arcról készült képeket fogja elemezni. Majd pedig jön két while ciklus, ami a frameket fogja olvasni. A külső while ciklus addig fog menni, amíg megy a stream, a belső pedig addig, amíg jönnek a framek. Ezek után ha az adott fram nem üres, akkor az adott képkockát átméretezzük 176x144-es méretre. minden egyes képet eltárol a program a Mat tömbbe, valamint szürkés árnyalatúra állítja az összes eltárolt képkockát. Ezek után a `detectMultiScale()` fogja megkeresni az arcokat a képkockákon. minden egyes megtalált arc egy téglalapként kerül eltárolásra egy listában. Majd pedig végül az arcokból egy QImage fog készülni. Maga a program pedig működés közben a következőképpen néz ki:



## 18.4. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esport-talent-search>

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/hello/BrainB>

Ebben a feladatban a Qt slot-signal mechanizmust kellett bemutatni a megadott forráskódban. Ahhoz, hogy ezt be tudjuk mutatni, először azt kellene megtudni, hogy mi is az. Erre a Qt dokumentációja nagyon jó választ ad, mi szerint a slot-signal mechanizmus az objektumok közötti kommunikációra szolgálnak. Ez a Qt egyik központi jellemzője. Egy adott esemény bekövetkezésekor egy jel (signal) bocsátjódik ki. A slot pedig egy funkció, amely egy jelre adott válaszként hívódik meg. Egy jelhez több slot is tartozhat, és egy slot több jelre is lehet válasz. Ezt a `connect()` függvénytelhet létrehozni aminek a szintaktikája a következőképpen néz ki: `connect(obj1, signal, obj2, slot)`. Ezek alapján már tudjuk, hogy ilyen `connect()` függvényeket kell keresnünk a forrásokban. Valamennyi keresés után a `BrainBWin.cpp` fájlban, azon belül pedig a konstruktőrben fogunk találni két ilyet, amelyek a következőképpen néznek ki:

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ),
```

```
    this, SLOT ( updateHeroes ( QImage, int, int ) ) );  
  
connect ( brainBThread, SIGNAL ( endAndStats ( int ) ),  
          this, SLOT ( endAndStats ( int ) ) );
```

Ahhoz, hogy ezeket megmagyarázzam, néhány szó a játékról. Az a lényege, hogy négyzetek vannak egy képernyőn, bennük pedig körök. A játék lényege pedig, hogy a bal egérgombot nyomva tartva a Samu Entropy-n tartsuk az egerünket. Minél jobbak vagyunk, annál több entropy, azaz hős lesz a képernyőn. Egy kép a játékról:

És akkor így már elmondható, hogy mi történik a két előfordulásnál. Az elsőnél, minden esetben, amikor a hősök, azaz Entropy-k helyzete, pozíciója megváltozik, azaz lefut a heroesChanged() metódus, akkor a BrainBThread fog kibocsátani egy jelet, amit a BrainBWin fog megkapni, és ő pedig frissíteni fogja a hősök helyzetét, azaz le fog futni nála az updateHeroes() metódus. A második előfordulásnál pedig, amikor véger ér a játék valami oknál fogva, azaz lefut az endAndStats() metódus, akkor a brainBThread kibocsát egy jelet, amit a BrainBWin fog észlelni, és nála is le fog futni a saját endAndStats metódusa. Még érdekes lehet a signal elküldése, amiket a következő függvényekben találhatunk meg:

```
void draw () {  
  
    cv::Mat src ( h+3*heroRectSize, w+3*heroRectSize, CV_8UC3, cBg );  
  
    for ( Hero & hero : heroes ) {  
  
        cv::Point x ( hero.x-heroRectSize+dispShift, hero.y-  
                      heroRectSize+dispShift );  
        cv::Point y ( hero.x+heroRectSize+dispShift, hero.y+  
                      heroRectSize+dispShift );  
    }  
}
```

```
cv::rectangle ( src, x, y, cBorderAndText );

cv::putText ( src, hero.name, x, cv::FONT_HERSHEY_SIMPLEX, .35, ←
    cBorderAndText, 1 );

cv::Point xc ( hero.x+dispShift, hero.y+dispShift );

cv::circle ( src, xc, 11, cCenter, CV_FILLED, 8, 0 );

cv::Mat box = src ( cv::Rect ( x, y ) );

cv::Mat cbox ( 2*heroRectSize, 2*heroRectSize, CV_8UC3, cBoxes ←
    );
box = cbox*.3 + box*.7;
}

cv::Mat comp;

cv::Point focusx ( heroes[0].x- ( 3*heroRectSize ) /2+dispShift, ←
    heroes[0].y- ( 3*heroRectSize ) /2+dispShift );
cv::Point focusy ( heroes[0].x+ ( 3*heroRectSize ) /2+dispShift, ←
    heroes[0].y+ ( 3*heroRectSize ) /2+dispShift );
cv::Mat focus = src ( cv::Rect ( focusx, focusy ) );

cv::compare ( prev, focus, comp, cv::CMP_NE );

cv::Mat aRgb;
cv::extractChannel ( comp, aRgb, 0 );

bps = cv::countNonZero ( aRgb ) * 10;

//qDebug() << bps << " bits/sec";

prev = focus;

QImage dest ( src.data, src.cols, src.rows, src.step, QImage::←
    Format_RGB888 );
dest=dest.rgbSwapped();
dest.bits();

emit heroesChanged ( dest, heroes[0].x, heroes[0].y );
}
```

Valamint:

```
void BrainBThread::run()
{
    while ( time < endTime ) {

        QThread::msleep ( delay );
    }
}
```

```
if ( !paused ) {  
  
    ++time;  
  
    devel();  
}  
draw();  
}  
emit endAndStats ( endTime );  
}
```

Mind a két esetben az emit utasítás fog lefutni, ami kibocsátja a jelet, amit a BrainBWin fog megkapni.

## 19. fejezet

# Helló, Lauda!

### 19.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! <https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/ch01.html#id527287>

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Lauda/KapuSzkenner.java>

Ebben a feladatban a kivételkezelésre kellett rámutatni a megadott forrásban. Nézzük is meg magát a forráskódot, illetve azt, hogy milyen eredményt ad a program futtatás közben:

```
public class KapuSzkenner {  
    public static void main(String[] args) {  
  
        for(int i=0; i<1026; ++i)  
  
            try {  
                java.net.Socket socket = new java.net.Socket(args[0], i);  
  
                System.out.println(i + " figyeli");  
  
                socket.close();  
  
            } catch (Exception e) {  
  
                System.out.println(i + " nem figyeli");  
            }  
    }  
}
```

És a kép a futtatásról:

```
1000 nem figyeli
1001 nem figyeli
1002 nem figyeli
1003 nem figyeli
1004 nem figyeli
1005 nem figyeli
1006 nem figyeli
1007 nem figyeli
1008 nem figyeli
1009 nem figyeli
1010 nem figyeli
1011 nem figyeli
1012 nem figyeli
1013 nem figyeli
1014 nem figyeli
1015 nem figyeli
1016 nem figyeli
1017 nem figyeli
1018 nem figyeli
1019 nem figyeli
1020 nem figyeli
1021 nem figyeli
1022 nem figyeli
1023 nem figyeli
1024 nem figyeli
1025 nem figyeli
andras@andrasubuntu:~/prog2/src/prog2/Lauda$ java KapuSzkenner localhost | grep -v "nem"
631 figyeli
andras@andrasubuntu:~/prog2/src/prog2/Lauda$
```

Ebből annyit tudunk meg, hogy van egy for ciklus, ami 0-tól 1025-ig fut. Azon belül van egy try-catch szerkezet. A try-ban létrehozunk egy socket objektumot aminek ip címnek a legelső argumentum értékét, portnak pedig i aktuális értékét adjuk. Ilyenkor a program megpróbál a program egy TCP kapcsolatot létrehozni. Ha ez az utasítás nem dob exception-t, akkor kiiratjuk, hogy egy szerver folyamat figyeli a portot, vagy röviden: figyeli, és bezárja a program a socketet. Azonban, ha itt exception-t dob a program, akkor kiirja, hogy nem figyeli. Ezután egy kicsit módosítottam a programot annak érdekében, hogy megtudjam azt, hogy milyen exceptiont dob pontosan a try. A következő eredményt kaptam:

Mint azt láthatjuk, ha nem sikerül kialakítani a kapcsolatot, akkor egy ConnectException-t kapunk. Az

oracle dokumentációja tisztán és érthetően megmagyarázza, hogy a ConnectException jelzi azt, hogy hiba történt egy socket egy távoli címhez és porthoz történő csatlakoztatásakor. Általában a kapcsolatot távolról tagadják meg, pl: egy folyamat sem figyeli a cím adott portját. Vagyis, ha sikerül kapcsolatot létrehoznia a programnak, akkor tudja, hogy azt a portot figyelik, ha pedig nem, akkor pedig tudja azt, hogy nem figyelik.

## 19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/Lauda/aop>

Ebben a feladatban egy átszövő vonatkozást kellett írni az LZWBinFa.java átiratába. Mivel az eredeti forráshoz semmit nem kellett hozzáírni, ezért arról nem is beszélnék, hanem tértünk is át rögtön az átszövő vonatkozásra. Ezt a feladatot AspectJ-vel kellett megvalósítani, ami lehetőséget ad arra, hogy anélkül módosítsuk egy program forráskódját, hogy ténylegesen módosítanánk azt a forráskódot. Itt is pontosan ugyan ez történt. Meg kellett adni kódcsipeteket egy külön fájlban, amiknek megmondhattuk, hogy egy adott metódus előtt vagy után fussanak le. Az AspectJ telepítését linuxon a következő parancs kiadásával lehet megtenni: **apt install aspectj**. Ha ez sikerült, akkor fordítani pedig nem a **javac**, hanem az **ajc** parancs kiadásával kell. Ha jól emlékszem BevProg védési feladat volt, hogy in és post order módon is irassuk ki a fát. Itt most ugyan ezt valósítottam meg aspectj segítségével. Nézzük is a kód első részletét:

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public aspect BinFa{
    int melyseg = 0;

    public pointcut callkiir(LzwBinFa fa, LzwBinFa.Csomopont n, PrintWriter os) :call(void LzwBinFa.kiir(LzwBinFa.Csomopont, PrintWriter)) && args(n,os) && target(fa) && within(LzwBinFa);

    after(LzwBinFa fa, LzwBinFa.Csomopont n, PrintWriter os) :callkiir(fa,n,os) {
    }

    public pointcut hivas(LzwBinFa.Csomopont n, PrintWriter os) : call(void LzwBinFa.kiir(LzwBinFa.Csomopont, PrintWriter)) && args(n,os);

    after(LzwBinFa.Csomopont n, PrintWriter os) : hivas(n, os) {

        try{
            kiirPre(n, new PrintWriter("preorder.txt"));
        }
        catch(FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

```
}

melyseg = 0;

try{
    kiirPost(n,new PrintWriter("postorder.txt"));
}
catch(FileNotFoundException e){
    System.out.println(e);
}
}
```

Először is importáljuk a PrintWriter-t és a FileNotFoundException-t, hiszen ezeket használni fogjuk. Majd pedig megmondjuk a programnak, hogy az után, hogy ha meghívódik az eredeti programban az LzwBinFa.kiirPost metódus, akkor annak a metódusnak az argumentumait elkérjük, és mután lefutott, próbálja meg lefuttatni a kiirPre() valamint a kiirPost() függvényeket. Nyílvánvalóan itt kaphatunk FileNotFoundException-t, éppen ezért ezt try-catch blokkba írjuk, és kezeljük.

```
public void kiirPost(LzwBinFa.Csomopont elem, java.io.PrintWriter os) {

    if(elem != null) {
        ++melyseg;

        kiirPost(elem.nullasGyermek(), os);
        kiirPost(elem.egyesGyermek(),os);

        for(int i = 0; i < melyseg; i++){
            os.print("---");
        }
        os.print(elem.getBetu());
        os.print("(");
        os.print(melyseg -1);
        os.println(")");

        --melyseg;
    }
}

public void kiirPre(LzwBinFa.Csomopont elem, java.io.PrintWriter os) {

    if(elem != null) {
        ++melyseg;

        for(int i = 0; i < melyseg; i++){
            os.print("---");
        }
        os.print(elem.getBetu());
        os.print("(");
    }
}
```

```

os.print(melyseg -1);
os.println(")");

kiirPre(elem.nullasGyermek(), os);
kiirPre(elem.egyesGyermek(), os);
--melyseg;
}
}

```

A maradék része a kódnak pedig a két függvény. Az eredeti programban a fát in order módon járta be a program, azaz először az adott elem bal oldali gyerekét dolgozta fel, aztán az adott elemet, és végül pedig az adott elem bal oldali gyerekét. Ezzel szemben itt az első függvény a Post order bejárás, ahol jól láthatóan először az adott elem bal oldali gyerekét dolgozza fel a függvény, aztán az adott elem jobb oldali gyerekét, és végül pedig magát az elemet. A második függvény pedig a pre order bejárás, ahol először az adott elemet dolgozza fel a függvény, majd pedig az adott elem bal oldali gyerekét, és végül pedig az adott elem jobb oldali gyerekét. Ez a két függvény vég számon tartja és kiírja a fa mélységét is. Végül pedig kép az eredményről:

The screenshot shows three terminal windows side-by-side, each displaying the output of a different program:

- inorder.txt**: Shows the in-order traversal of a binary tree. The output consists of pairs of values separated by a dash, with the first value being the depth and the second being the node value. The traversal starts at depth 0 with node 0(1), goes down to depth 1 with nodes 0(2) and 1(6), and so on, reaching depth 7 with nodes 1(2) and 0(4).
- postorder.txt**: Shows the post-order traversal of the same binary tree. The output follows a similar pattern but with different node values due to the traversal order.
- ki.txt**: Shows the final output of the program, which includes the average depth (átlag), standard deviation (szórás), and the maximum depth (mélység) of the tree.

At the bottom of the terminal windows, the status bar displays "Tabulátorszélesség: 8" and "13. sc".

## 19.3. Android Játék

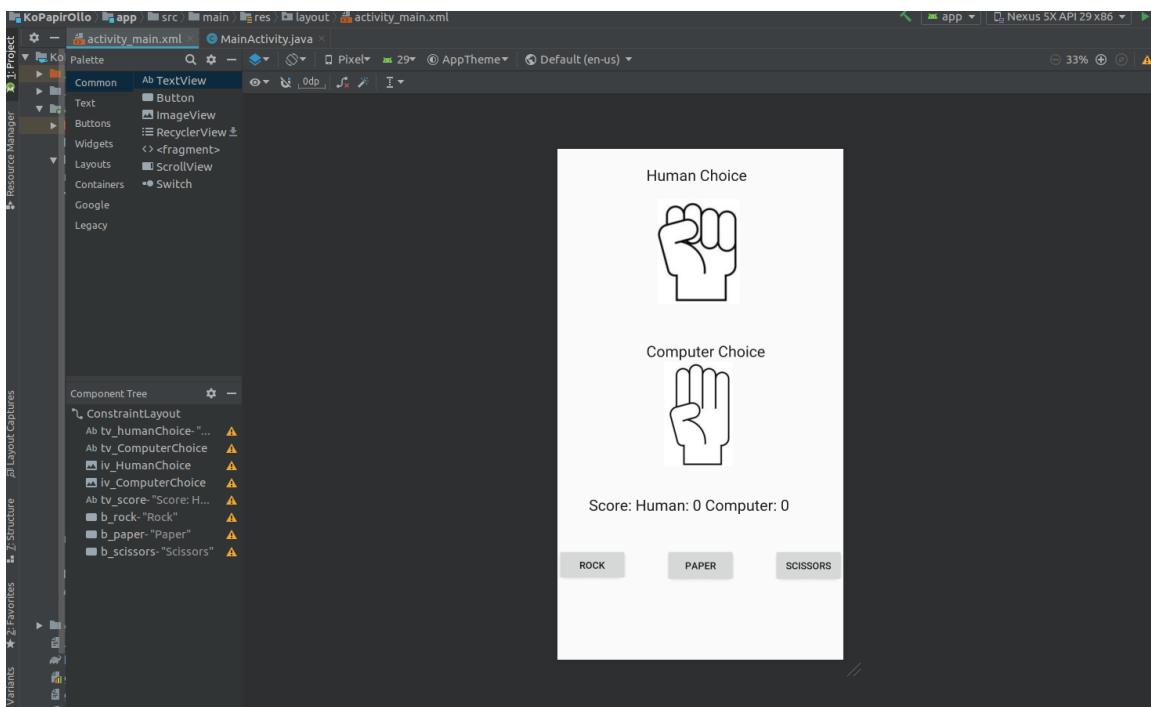
Írunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Hello, Android!” feladatára!

Megoldás videó:

Projekt forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/Lauda/KoPapirOllo>

Main forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Lauda/KoPapirOllo/app/src/main/java/com/example/kopapiollo/MainActivity.java>

Ebben a feladatban egy egyszerű android játékot kellett készíteni. Én egy kő papír olló játéket készítettem számítógép ellen az Android Studio segítségével.



Először is kerestem három szabadon felhasználható képet, amiket fel tudnék használni a játékhoz. Ezeket belehelyeztem a drawable mappába. A GUI létrehozása egyszerű, mivel szimplán csak be kell húzgálni a kívánt elemeket, illetve nevet és értéket adni azoknak. A fentebbi képen jól látható, hogy két képet helyeztem fel. Az egyik a számítógép legutóbbi választását mutatja, a másik pedig a játékosét. Találunk még három szövegablakot is. A legfelső megmagyarázza, hogy az első kép a játékos választását, a második szövegablak pedig azt, hogy a második kép pedig a számítógép választását mutatja. Ez a két szöveg nem fog változni a játék alatt. A harmadik szövegrész pedig a játék jelenlegi állását mutatja. Végül pedig található három gomb, amikkel a játékos a választását tudja jelezni a programnak. Most pedig nézzük a forrást.

```
public class MainActivity extends AppCompatActivity {
    Button b_rock, b_scissors, b_paper;
    TextView tv_score;
    ImageView iv_ComputerChoice, iv_HumanChoice;

    int HumanScore, ComputerScore = 0;

    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b_paper = (Button) findViewById(R.id.b_paper);
    b_scissors = (Button) findViewById(R.id.b_scissors);
    b_rock = (Button) findViewById(R.id.b_rock);

    iv_ComputerChoice = (ImageView) findViewById(R.id.iv_ComputerChoice);
    iv_HumanChoice = (ImageView) findViewById(R.id.iv_HumanChoice);

    tv_score = (TextView) findViewById(R.id.tv_score);
```

Először is létrehozzuk a gombokat, a képeket, illetve egy szövegrészt is, valamint összekapcsoljuk őket a GUI-n lévő elemekkel. Megfigyelhető, hogy a két nem változó szövegrészhet nem csatolunk semmit. Ez pontosan azért van, mert semmit nem szeretnénk csinálni velük, éppen ezért nincs is rájuk szükség. Ezen kívül még létrejön két változó, amik a játékos és a gép eredményeit fogják számoltartani. Ezek után jönnek a gombokra kattintást figyelő listenerek:

```
@Override
    public void onClick(View view) {
        iv_HumanChoice.setImageResource(R.drawable.papir);
        String message = play_turn("paper");
        Toast.makeText(MainActivity.this, message, Toast.LENGTH_SHORT).show();

        tv_score.setText("Score: Human: " + Integer.toString(HumanScore) + " Computer: " + Integer.toString(ComputerScore));
    }
}

b_rock.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        iv_HumanChoice.setImageResource(R.drawable.ko);
        String message = play_turn("rock");
        Toast.makeText(MainActivity.this, message, Toast.LENGTH_SHORT).show();
        tv_score.setText("Score: Human: " + Integer.toString(HumanScore) + " Computer: " + Integer.toString(ComputerScore));
    }
});

b_scissors.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        iv_HumanChoice.setImageResource(R.drawable.ollo);
```

```
        String message = play_turn("scissors");
        Toast.makeText(MainActivity.this, message, Toast.LENGTH_SHORT).show();
        tv_score.setText("Score: Human: " + Integer.toString(HumanScore) +
                         " Computer: " + Integer.toString(ComputerScore));
    }
});
```

Mind a három esetben a játékos választását jelző képet átállítjuk a megfelelő képre, majd pedig létrehozunk egy stringet, aminek az értéke a `play_turn()` függvény által kapott eredmény lesz. Ennek a függvénynek az attribútuma a játékos választása. Majd pedig megjelenítünk egy kis üzenetet, ami a legalsó képen látható, illetve beállítjuk az eredményt mutató szövegrész értékét a megfelelő értékre. Végül pedig nézzük a `play_turn()` függvényt:

```
public String play_turn( String player_choice) {
    String computer_choice = "";
    Random r = new Random();

    int computer_choice_number = r.nextInt(3)+1;

    if(computer_choice_number == 1){
        computer_choice = "rock";
        iv_ComputerChoice.setImageResource(R.drawable.ko);
    }

    if(computer_choice_number == 2){
        computer_choice = "scissors";
        iv_ComputerChoice.setImageResource(R.drawable.ollo);
    }

    if(computer_choice_number == 3){
        computer_choice = "paper";
        iv_ComputerChoice.setImageResource(R.drawable.papir);
    }

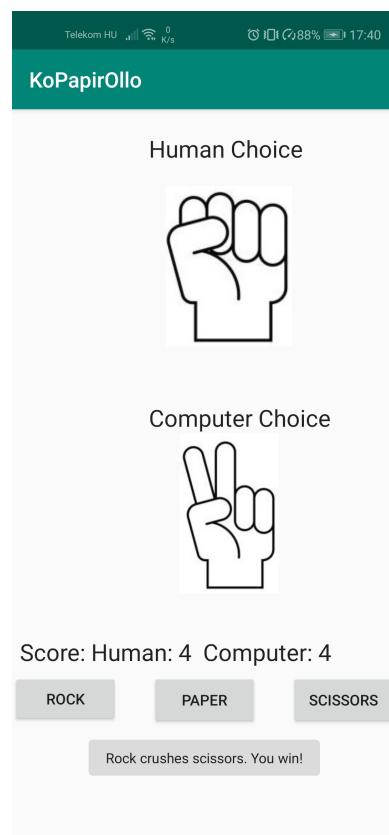
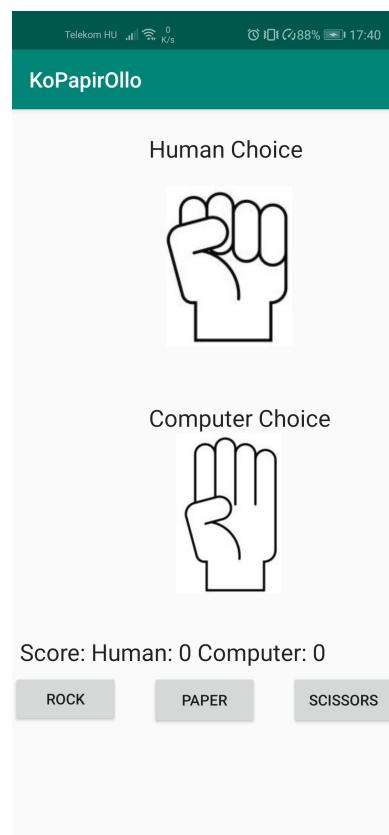
    if(computer_choice.equals(player_choice)) {
        return "Draw. Nobody won.";
    }

    else if(computer_choice.equals("rock") && player_choice.equals("scissors")) {
        ComputerScore++;
        return "Rock crushes scissors. Computer wins";
    }

    else if(computer_choice.equals("rock") && player_choice.equals("paper")) {
        HumanScore++;
        return "Paper covers rock. You win";
    }
}
```

```
else if(computer_choice.equals("scissors") && player_choice.equals ("rock")) {
    HumanScore++;
    return "Rock crushes scissors. You win!";
}
else if(computer_choice.equals("scissors") && player_choice.equals ("paper")) {
    ComputerScore++;
    return "Scissors cuts paper. Computer wins!";
}
else if(computer_choice.equals("paper") && player_choice.equals("scissors")) {
    HumanScore++;
    return "Scissors cuts paper. You win!";
}
else if(computer_choice.equals("paper") && player_choice.equals("rock")) {
    ComputerScore++;
    return "Paper Covers rock. Computer wins!";
}
else return "Not Sure";
}
```

Itt először is generálunk egy random számot 1 és 3 között, amivel meghatározzuk a gép választását. Majd pedig a sok if-else-if szerkezettel eldönti a program, hogy ki nyert. Ha a gép és a játékos is ugyan azt választotta, akkor döntetlen. Amennyiben a gép nyer, úgy megnöveljük a ComputerScore értékét, és azt adja vissza a függvény, hogy a gép nyert. Míg ha a játékos nyer, akkor a HumanScore változó értéke lesz 1-el megnövelve, és azzal tér vissza a függvény, hogy a játékos nyert. És kezdődik az egész előről. Néhány kép a játék működéséről:



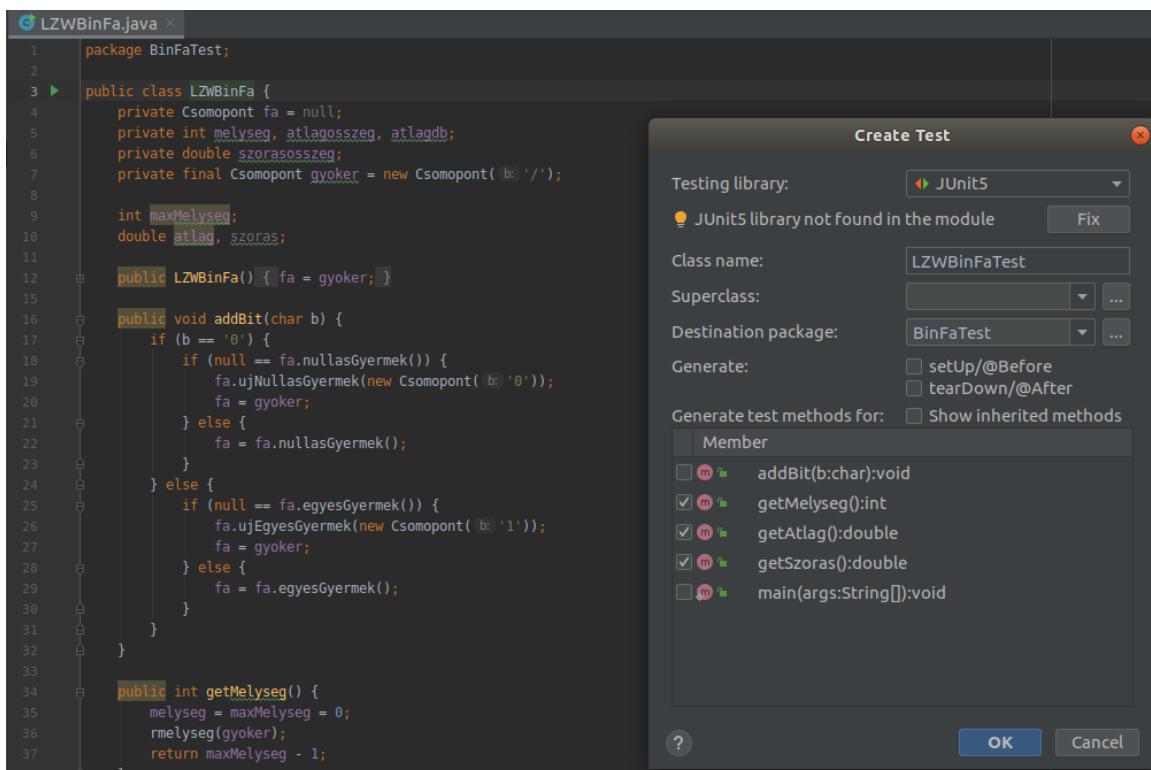
## 19.4. Junit teszt

A [https://progpater.blog.hu/2011/03/05/labormeres\\_oththon\\_avagy\\_hogyan\\_dolgozok\\_fel\\_egy\\_pedat\\_poszt\\_kézzel\\_számított\\_mélységét és szórását dolgozd\\_be egy Junit tesztbe \(sztenderd védési feladat volt korábban\).](https://progpater.blog.hu/2011/03/05/labormeres_oththon_avagy_hogyan_dolgozok_fel_egy_pedat_poszt_kézzel_számított_mélységét és szórását dolgozd_be egy Junit tesztbe (sztenderd védési feladat volt korábban).)

Megoldás video:

Megoldás forrása: <https://github.com/raczandras/progbook/tree/master/src/prog2/Lauda/junit>

Ebben a feladatban egy a Binfához megadott bemenethez kellett Junit tesztet írni, és remélhetőleg az eredmény megegyezik a papíron kiszámolt eredménnyel. A megadott bemenet az volt hogy 01111001001001000111. Erre kézzel kiszámolva 4-es mélység, 2.75-ös átlag, és 0.9574-es értékű szórás jött ki, ahogy az a feladat leírásában lévő linket megnyitva is látszik. Szóval ha minden jól megy, akkor a JUnit tesztünk is ugyan ezeket az eredményeket fogja kiszámolni. Én ennek a feladatnak a megoldásához IntelliJ Idea-t használtam. Mivel az eredeti LZWBInFa kódján megint nem kellett változtatni, ezért azt most sem taglalnám, hanem térdjük rá egyből a tesztre. Szerencsére IntelliJ-ben lehet tesztet generálni. Ezt azonban ugyan úgy kell elképzelni, mint amikor UML-ből generálunk kódot. Azaz csak egy sablont fogunk kapni. Egyszerűen csak a tesztelni kívánt osztály nevére jobb egérrel kattintunk, és aztán pedig a generate test-re:



Itt kijelölhetjük, hogy mely metódusokat szeretnénk tesztelni. Majd pedig ha leukézzük, akkor az előbb említett sablont kapjuk ami a következőképpen néz ki:

```
LZWBinFa.java x LZWBinFaTest.java x
1 package BinFaTest;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 >> class LZWBinFaTest {
6
7     @org.junit.jupiter.api.Test
8     void getMelyseg() {
9         }
10
11     @org.junit.jupiter.api.Test
12     void getAtlag() {
13         }
14
15     @org.junit.jupiter.api.Test
16     void getSzoras() {
17         }
18 }
```

Ezt ha egy kicsit kibővítjük akkor a következő kódot fogjuk kapni eredményül:

```
package junit;
import static org.junit.jupiter.api.Assertions.*;

class LZWBinFaTest {

    private LZWBinFa binfa = new LZWBinFa();
    private static final String testStr = "01111001001001000111";

    @org.junit.jupiter.api.Test
    void getMelyseg() {
        for (char i : testStr.toCharArray())
            binfa.addBit(i);
        double melyseg = binfa.getMelyseg();
        System.out.println("getMelyseg() teszt -- expecting: 4, got: " + ←
                           melyseg);
        assertEquals(4, melyseg);
    }

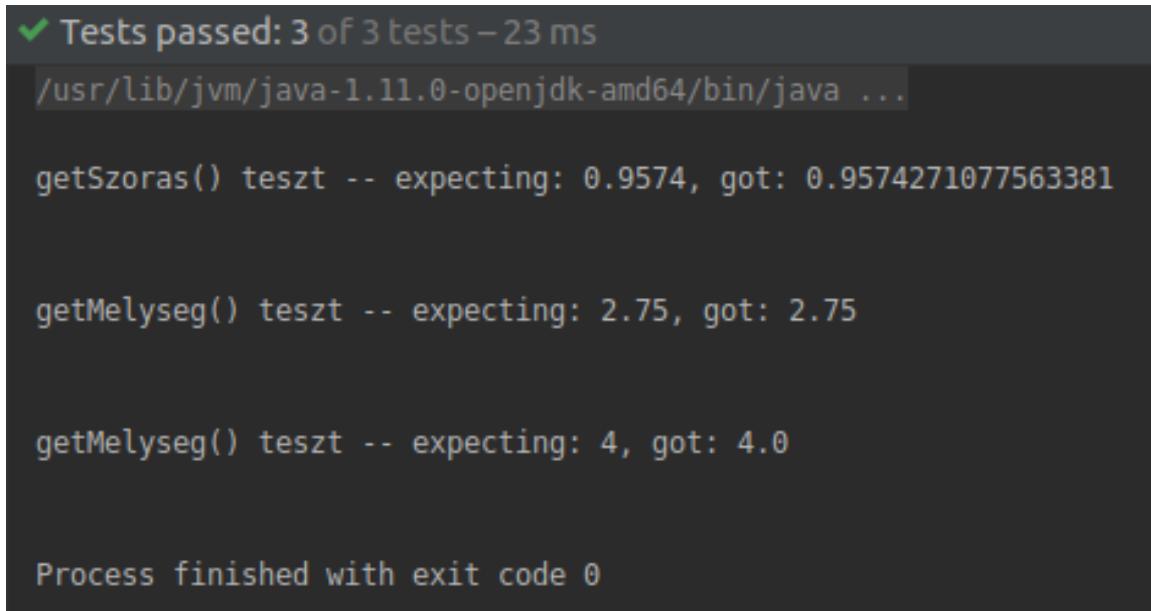
    @org.junit.jupiter.api.Test
    void getAtlag() {
        for (char i : testStr.toCharArray())
            binfa.addBit(i);
        double atlag = binfa.getAtlag();
```

```
        System.out.println("getMelyseg() teszt -- expecting: 2.75, got: " + ←
            atlag);
        assertEquals(2.75, atlag);
    }

    @org.junit.jupiter.api.Test
    void getSzoras() {
        for (char i : testStr.toCharArray())
            binfa.addBit(i);
        double szoras = binfa.getSzoras();
        System.out.println("getSzoras() teszt -- expecting: 0.9574, got: " ←
            + szoras);
        assertEquals(0.9574, szoras, 0.001);
    }
}
```

Menjünk sorba. Először is létrehozunk egy LZWBinFa objektumot, illetve létrehozzuk a stringet, amivel teszteljük. Ezek után jönnek a metódusok. Először is a void getMelyseg() metódus, amiben egy for-each ciklussal belepakoljuk a stringünk karaktereit a binfába, majd pedig meghatározzuk a keletkezett fa mélységét, kiirjuk azt a felhasználó számára, és az assertEquals() függvényel összehasonlítjuk a kapott eredményt a várt eredménnyel. Pontosan ugyan ez történik a további két metódusban is. Belepakoljuk a string karaktereit a fába, meghatározzuk az átlagot/szórást, és összehasonlítjuk a kapott eredményt a várt eredménnyel.

Végül pedig kép az eredményről:



```
✓ Tests passed: 3 of 3 tests – 23 ms
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java ...
getSzoras() teszt -- expecting: 0.9574, got: 0.9574271077563381

getMelyseg() teszt -- expecting: 2.75, got: 2.75

getMelyseg() teszt -- expecting: 4, got: 4.0

Process finished with exit code 0
```

## 20. fejezet

# Helló, Calvin!

### 20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, [https://progpater.blog.hu/2016/11/13/hello\\_sbol](https://progpater.blog.hu/2016/11/13/hello_sbol) Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Megoldás videó:

Megoldás forrása: [https://github.com/raczandras/progbook/tree/master/src/prog2/Calvin/Mnist/mnist\\_soft.py](https://github.com/raczandras/progbook/tree/master/src/prog2/Calvin/Mnist/mnist_soft.py)

Ebben a feladatban a Tensorflow segítségével kellett megtanítani egy mesterséges intelligenciának azt, hogy felismerjen egy kézzel írott, 28x28 pixeles képen lévő számot. Ehhez az MNIST könyvtárat használja tanulásként. A Bátfai Norbert által megadott forrást kicsit át kellett alakítani, mert azóta, hogy az elkészült, rengeteget változott a tensorflow. Legtöbbször függvényhívásokat kellett átírni, vagyis inkább hozzáadni a függvényhíváshoz azt, hogy *compat.v1.* és már működtek is. Azonban volt pár sor, amit másképp kellett átírni. Ezek a következők: Először is a 46. sorban a függvényhívást a következőképpen kellett átírni:

```
img = tf.image.decode_png(file, channels=1)
```

Ez annyit jelent, hogy a saját képünknek, amit fel kellene, hogy ismerjen a program, a grayscale-es változata lesz felhasználva. Arra, hogy ez miért kell, két okunk van. Az első az az, hogy az mnist képei is grayscale képek, a másik (a fontosabb) ok pedig az, hogy ha ezt a változtatást nem tesszük meg, akkor hibát fogunk kapni. Éppen ezért ez egy erősen ajánlott változtatás. A következő változtatást pedig a 72. sorban kellett elvégezni, ahol is a paramétereket kellett nevesíteni, amit a következőképpen lehet megoldani:

```
cross_entropy = tf.reduce_mean(tf.nn. ←  
    softmax_cross_entropy_with_logits(logits=y, labels=y_))
```

Valamint az utolsó kis kódcsipet amit nem át, hanem hozzáírtam a forráshoz, az a következő sor:

```
tf.io.write_graph(sess.graph, "models/", "mnist.pbtxt")
```

Ez annyit jelent, hogy a models mappába hozzon létre egy mnist.pbtxt nevű fájlt, ami a programnak a gráfja. Ezt később Tensorboard-al meg fogjuk tudni nyitni és elemezni. Ezek után nézzük magát a forráskódot:

```
tf.compat.v1.disable_eager_execution()  
x = tf.compat.v1.placeholder(tf.float32, [None, 784])  
W = tf.Variable(tf.zeros([784, 10]))
```

```
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
y_ = tf.compat.v1.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits( ←
    logits=y, labels=y_))
train_step = tf.compat.v1.train.GradientDescentOptimizer(0.5).minimize( ←
    cross_entropy)

tf.compat.v1.initialize_all_variables().run()
print("-- A halozat tanitasa")
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
        print(i/10, "%")
print("-----")

# Test trained model
print("-- A halozat tesztelese")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("-- Pontossag: ", sess.run(accuracy, feed_dict={x: mnist.test. ←
    images,
    y_: mnist.test.labels}))
print("-----")
```

Először is eltároljuk magát a képet, ami ugye 28x28, azaz összesen 784 pixel. Majd pedig elkezdjük a for ciklussal tanítani a programunkat. Ezer képpel tanítjuk, és minden századik képnél kiirjuk azt, hogy hol járunk. Ezek után ellenőrizzük a pontosságát, és azt is a felhasználó tudomására hozzuk. Majd pedig megpróbálunk felismertetni először egy mnist képet, majd pedig a saját magunk által rajzolt nyolcast.

```
img = mnist.test.images[42]
image = img

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm ←
    .binary)
matplotlib.pyplot.savefig("4.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

print("-- A saját kezi 8-asom felismerese, mutatom a szamot, a ←
    tovabbolteshez csukd be az ablakat")

img = readimg()
image = img.eval()
```

```
image = image.reshape(28*28)

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

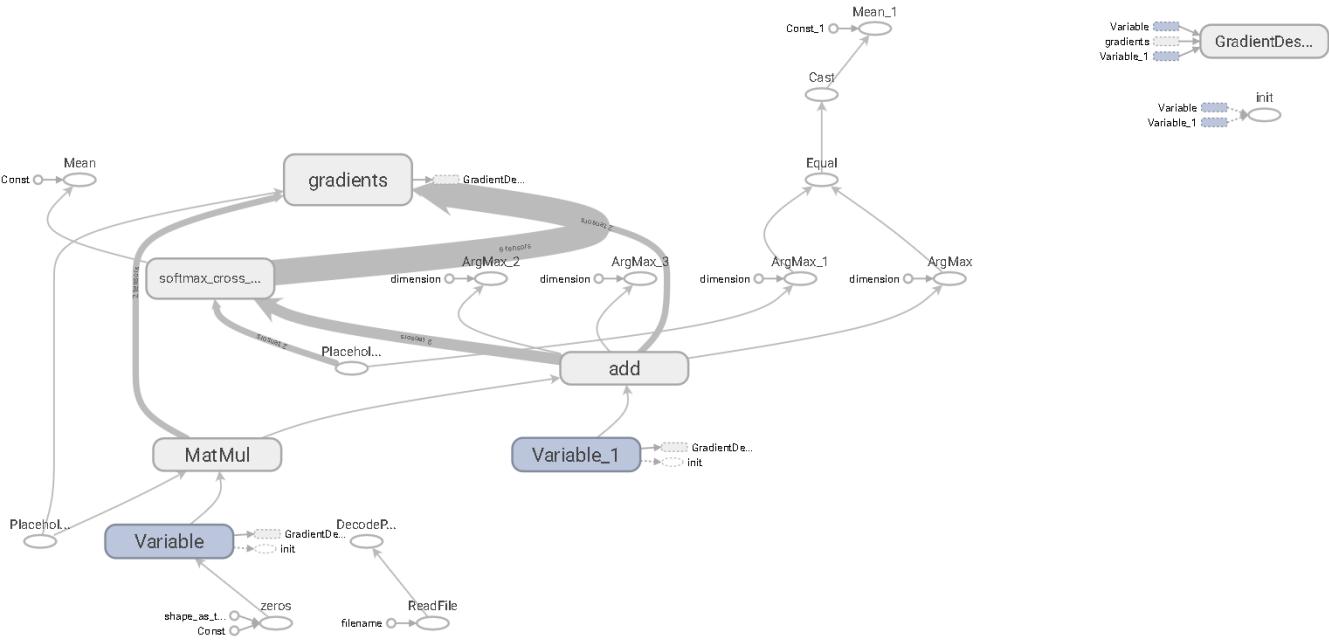
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

tf.io.write_graph(sess.graph, "models/", "mnist.pbtxt")
```

Mind a két kép esetében elmentjük az adott képet, aztán pedig kirajzoltatjuk a `matplotlib.pyplot.show()` függvénytellyel. Ezek után pedig meghatározza a program, hogy ő minek gondolja az adott számot, és végül pedig kiiratjuk a konzolra az eredményt. Az eredmény pedig:

```
Konkol 1/A mnist_softmax.py/A
warnings.warn("An interactive session is already active. This can
-- A halozat tanitasa
0.0 %
10.0 %
20.0 %
30.0 %
40.0 %
50.0 %
60.0 %
70.0 %
80.0 %
90.0 %
-----
-- A halozat tesztelese
-- Pontossag: 0.9162
-----
-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a tovabblepeshoz csukd be az ablakat
0
5
10
15
20
25
0 5 10 15 20 25
-- Ezt a halozat ennek ismeri fel: 4
-----
-- A sajat kezi 8-asom felismerese, mutatom a szamot, a tovabblepeshoz csukd be az ablakat
0
5
10
15
20
25
0 5 10 15 20 25
-- Ezt a halozat ennek ismeri fel: 8
```

Ezek után már csak a gráf megjelenítése van hátra Tensorboard-ban:



## 20.2. Deep MNIST

Mint az előző, de a mély változattal.

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Calvin/Deepmnist/deepmnist.py>

Ebben a feladatban az előző Tensorflow-os feladatot kellett ismét megoldani, azonban ezúttal már deep mnist-es változattal. Az alap forráskód megtalálható a Tensorflow github repójában, én is azt vettettem alapul. Viszont pár dolgot át kellett irni, azonban mindenhol ugyan azt. A következő sorokban: 76, 101, 111, 117, 121, 124, 135, 144, 145, 147, 148, és 179. Ezekben a sorokban a `tf` és a függvény neve közé azt kellett irni, hogy `.compat.v1` és már működött is az alap program. Azonban az alap program nem tartalmazza a saját magunk által rajzolt kép beolvasását és felismerését. De szerencsénkre ezt egy az egyben kimásolhatjuk az előző forrásból:

```
img = readimg()
image = img.eval()
image = image.reshape(28*28)
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
```

Erről már tudjuk, hogy azt jelenti, hogy beolvassuk a saját képünket, átalakítjuk, és elmentjük 8.png néven. Ezek után pedig megjelenítjük az elmentett képet, és kiíratjuk azt, hogy mit gondol a képről a program. Most pedig nézzük a forráskód többi részét.

```
with tf.name_scope('reshape'):
    tf.compat.v1.disable_eager_execution()
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
```

```
x = tf.compat.v1.placeholder(tf.float32, [None, 784])

y_ = tf.compat.v1.placeholder(tf.float32, [None, 10])

y_conv, keep_prob = deepnn(x)

with tf.name_scope('loss'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                             logits=y_conv)
cross_entropy = tf.reduce_mean(cross_entropy)

with tf.name_scope('adam_optimizer'):
    train_step = tf.compat.v1.train.AdamOptimizer(1e-4).minimize(←
        cross_entropy)

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)

graph_location = tempfile.mkdtemp()
print('Saving graph to: %s' % graph_location)
train_writer = tf.compat.v1.summary.FileWriter(graph_location)
train_writer.add_graph(tf.compat.v1.get_default_graph())
```

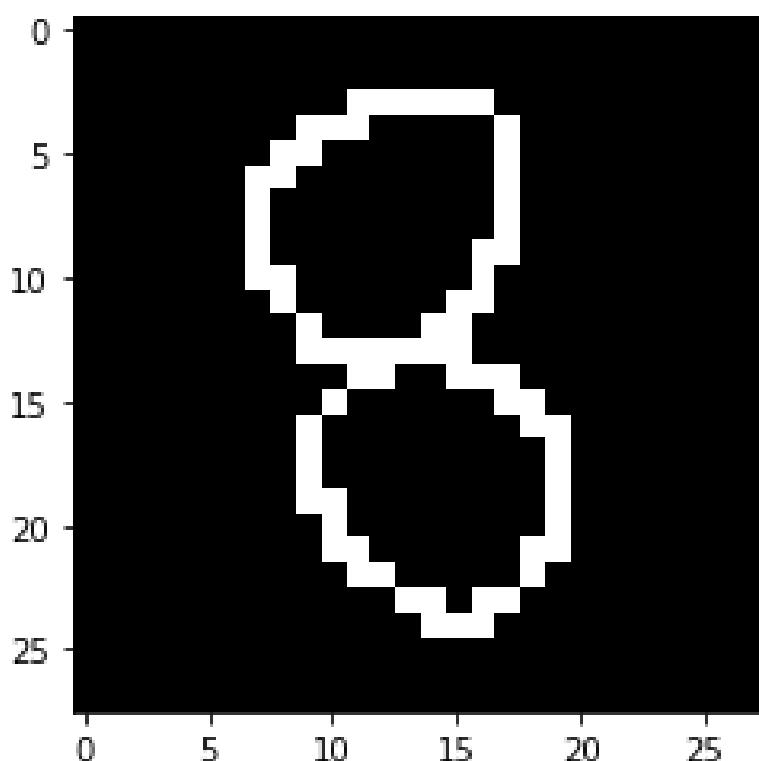
Először is létrehozza modellt a program, és mivel csak a grayscale-es képekre vagyunk kíváncsiak, éppen ezért átalakítjuk őket úgy hogy számunkra megfeleljenek.

```
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0})
            print('step %d, training accuracy %g' % (i, train_accuracy))
            train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

    print('test accuracy %g' % accuracy.eval(feed_dict={
        x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

Ezek után jön a tanítás része a dolognak. Húszezer képpel tanítjuk a programot, és 100 képenként kiíratjuk azt, hogy éppen hol járunk, illetve azt is, hogy mekkora pontosságal ismeri fel a képeket. Itt egy elég kis számmal fog kezdetűdni, de ahogy egyre több képet dolgoz fel, úgy fogja egyre jobban felismerni a képeket. Amint kész a tanítás, megmutatjuk neki a saját képünket, és reménykedünk abban, hogy felismeri. Szerencsére az én esetben igen lett a válasz. Azt még érdemes hozzátenni, hogy a tanítás sokkal több időt vesz igénybe, mint az előző feladatnál. Nekem körülbelül egy órába telt a folyamat. És végül pedig egy kép az eredményről:

```
step 18000, training accuracy 0.95
step 18100, training accuracy 0.98
step 18200, training accuracy 0.96
step 18300, training accuracy 0.95
step 18400, training accuracy 0.97
step 18500, training accuracy 0.96
step 18600, training accuracy 0.98
step 18700, training accuracy 0.98
step 18800, training accuracy 0.97
step 18900, training accuracy 0.97
step 19000, training accuracy 0.96
step 19100, training accuracy 0.98
step 19200, training accuracy 0.95
step 19300, training accuracy 0.97
step 19400, training accuracy 0.97
step 19500, training accuracy 0.96
step 19600, training accuracy 0.95
step 19700, training accuracy 0.98
step 19800, training accuracy 0.96
step 19900, training accuracy 0.95
test accuracy 0.95
```



-- Ezt a halozat ennek ismeri fel: 8

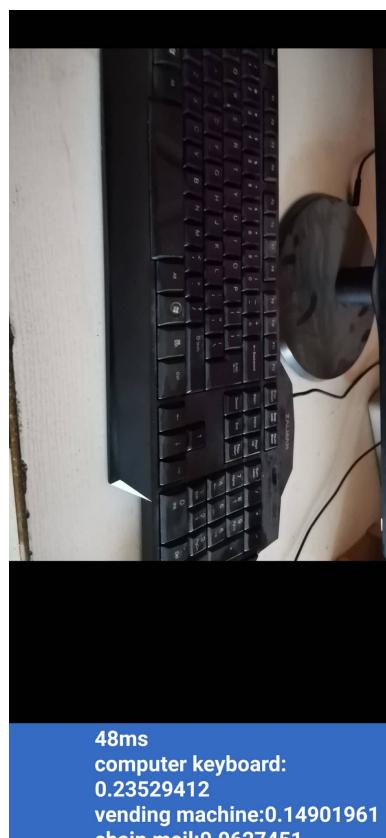
## 20.3. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

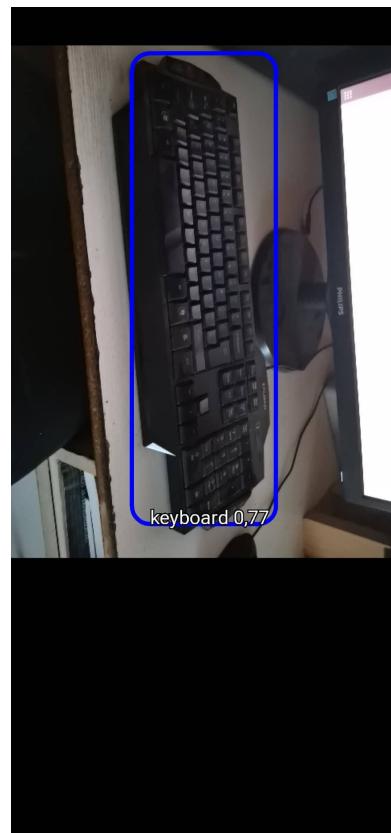
Megoldás videó:

Megoldás forrása:

Ebben a feladatban ki kellett próbálni a tensorflow object detect, azaz objektum detektáló programját. Ennek a forrását [ezen a linken](#) lehet elérni. Azt a repót le kell szedni, majd pedig android studio-ban buildelni a forrást, amiből kapunk egy apk fájlt. Ezzel fel lehet telepíteni androidos okostelefonokra a programot. Azonban ha ehhez lusták vagyunk, akkor le lehet tölteni egy demo apk-t, és azt feltelepíteni. Én mind a két megoldást kipróbáltam. A különbség az az, hogy az általunk buildelt forrással minden a négy program fel lesz rakva a telefonunkra (classify, detect, stylize, speech), a demo apk viszont csak a classify-t fogja feltelepíteni nekünk. Na de nézzük az eredményeket. Én először informatikai dolgokat próbáltam felismertetni a programmal, amik változóan sikeresek lettek.



Láthatjuk, hogy még egy egérrel teljes mértékben felismer, és biztosan állítja róla a program hogy az egy egér, addig ha egy billentyűzetről van szó, akkor abban már csak 23 százalékon biztos. Ez a két kép a Classify programmal készült. Ha ugyan arról a billentyűzetről a detect programmal készítünk képet, akkor teljesen más eredményt kapunk.



A detect program ugyan arra a billentyűzetre már 77 százalékos biztosággal állította azt, hogy ez egy billentyűzet. Azaz valószínűleg teljesen más módon működnek a programok, és ezt nem csak a billentyűzet esete bizonyítja. Kipróbáltam mind a két programot egy ventillátoron, és az eredmény pont fordított lett. Az első esetben...



...a classify program 99 százalékosan azt mondta, hogy ez egy ventillátor, ami igaz is. Azonban, amikor ugyan ezt a ventillátort megpróbáltam felismertetni a detect programmal, a következő eredményt kaptam:



Az eredmény ebben az esetben az lett, hogy 68% esély van arra, hogy ez egy wc, ami elég távol áll a valóságtól. Ezek után megpróbáltam egy igen specifikus dolgot felismertetni mind a két programmal, elég kevés sikkerrel:



Egy toyota logóra az egyik program azt mondta, hogy ez egy loupe, ami tulajdonképpen egy nagyító, a másik program pedig azt mondta, hogy ez egy olló. Összességében egy egy elégé érdekes kísérlet volt, és egy belélátás abba, hogy hol jár jelenleg ez a technológia.

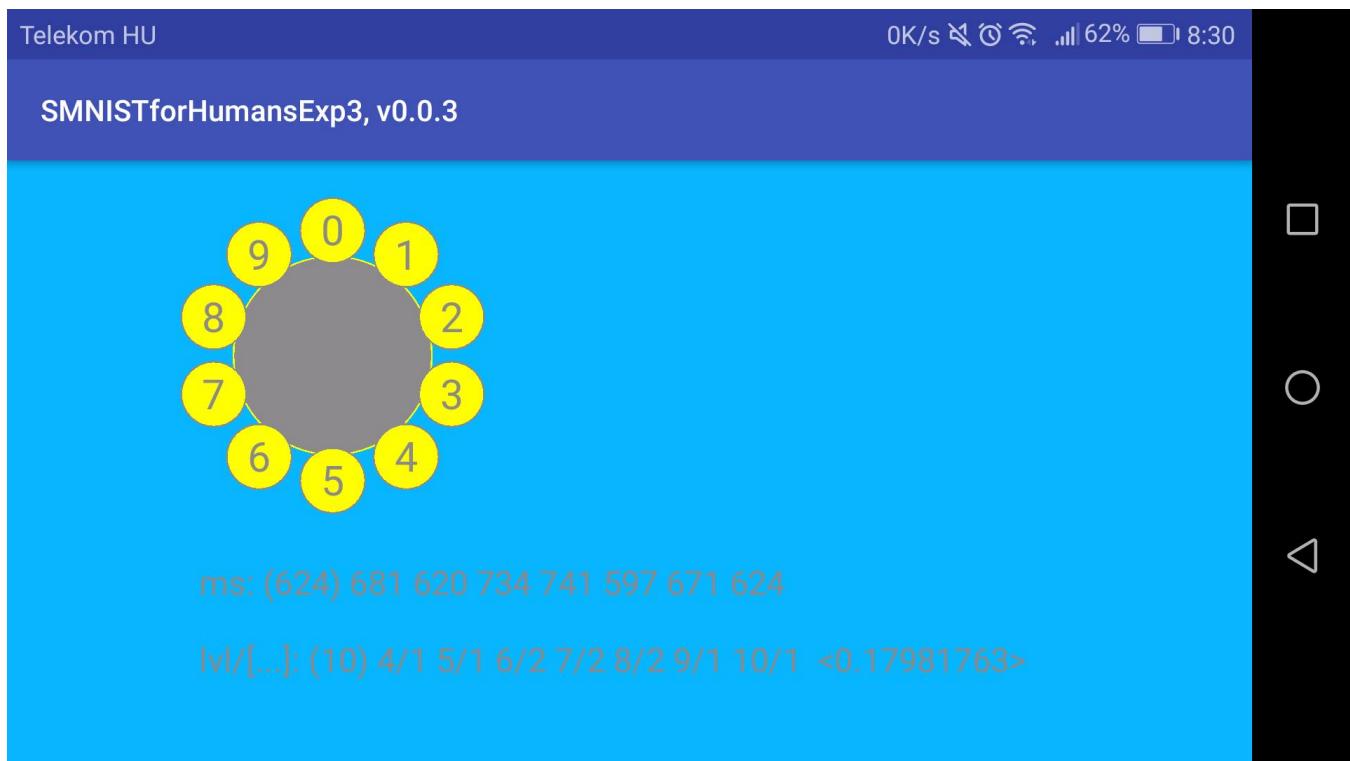
## 20.4. SMNIST for Machines

Készíts saját modellt, vagy használj meglévőt, lásd: <https://arxiv.org/abs/1906.12213>

Megoldás videó:

Megoldás forrása: <https://github.com/raczandras/progbook/blob/master/src/prog2/Calvin/Smnist/smnist.py>

Ebben a feladatban az SMNIST for humans program továbbfejlesztett változatát kellett kipróbálni. Az SMNIST for humans még prog1-es téma volt, és az a lényege, hogy egy programban található egy kör, és a körben pedig 0 és 9 darab közötti véletlenszerűen változó számú pontok találhatóak. minden egyes váltásnál fel kell ismernie a felhasználónak azt, hogy hány darab négyzet van a körben, és az ennek megfelelő számjegyre bökni. Egy kép erről a programról:



Egy átlag ember körülbelül a hatodik szintig képes eljutni. Azonban egy gép valószínűleg sokkal ügyesebb egy embernél. Éppen ezért jött létre az SMNIST for Machines. Ami nagyon hasonló a for humans verzióhoz, azonban ez kifejezetten számítógépek számára jött létre. A mi feladatunk az volt, hogy próbálunk ki egy ilyen SMNIST for Machines modellt. Én Kovács Ferencz modelljét választottam ehhez a feladathoz. A futtatáshoz szükségünk lesz a mxnet-re, amit egy egyszerű parancs kiadásával fel lehet telepíteni: **pip install mxnet**. Valamint szükségünk lesz még pár fájlra is, de ezeknek a beszerzési lehetőségét a forrás tágycalása közben fogom elmondani, amit nézzük is meg.

```
import numpy as np
import gzip
import os, sys
import mxnet as mx
import logging
logging.getLogger().setLevel(logging.INFO)

# Fix the seed
mx.random.seed(43)

# Set the compute context
ctx = mx.cpu()

# Load the data
os.listdir()
os.chdir('D:\smnist')

with gzip.open('t10k-images-idx3-ubyte.gz', 'r') as f:
    test_data = np.frombuffer(f.read(), np.uint8, offset=16)

with gzip.open('t10k-labels-idx1-ubyte.gz', 'r') as f:
```

```
    test_labels = np.frombuffer(f.read(), np.uint8, offset = 8)

with gzip.open('train-images-idx3-ubyte.gz', 'r') as f:
    train_data = np.frombuffer(f.read(), np.uint8, offset=16)

with gzip.open('train-labels-idx1-ubyte.gz', 'r') as f:
    train_labels = np.frombuffer(f.read(), np.uint8, offset = 8)
```

Először is beimportálunk pár dolgot, amire szükség lesz. Az első a numpy, amire a tesztképek átalakításához lesz szükség, a második a gzip, amivel a fájlokat fogjuk megnyitni, a harmadik az os, amivel abba a mappába fog navigálni a program, amelybe a szükséges fájlok találhatóak. Negyedik az mxnet, aminek a tanítás lesz a fő feladata, és végül pedig a logging, amivel logolni fog a program. Majd pedig rögtön látunk egy setLevel() függvényt, amivel azt lehet beállítani, hogy milyen szintig ignorálja a program a logging üzeneteket. Ezek után beállítjuk, hogy a cpu fogja elvégezni a munkát, és megnyitjuk a szükséges fájlokat. Ezeket a fájlokat [Erről](#) a linkről lehet beszerezni, ás ezek tartalmazzák a tanításhoz szükséges képeket.

```
#Reshape and prepare data
train_data = np.reshape(train_data, (-1,1,28,28))
test_data = np.reshape(test_data, (-1,1,28,28))
batch_size = 100
train_iter = mx.io.NDArrayIter(train_data, train_labels, batch_size, ←
    shuffle=True)
val_iter = mx.io.NDArrayIter(test_data, test_labels, batch_size)

data = mx.sym.var('data')
# first conv layer
conv1 = mx.sym.Convolution(data=data, kernel=(5,5), num_filter=32)
tanh1 = mx.sym.Activation(data=conv1, act_type="relu")
pool1 = mx.sym.Pooling(data=tanh1, pool_type="max", kernel=(2,2), stride ←
    =(2,2))

#second conv layer
conv2 = mx.sym.Convolution(data=pool1, kernel=(5,5), num_filter=64)
tanh2 = mx.sym.Activation(data=conv2, act_type="relu")
pool2 = mx.sym.Pooling(data=tanh2, pool_type="max", kernel=(2,2), stride ←
    =(2,2))

# first fullc layer
flatten = mx.sym.flatten(data=pool2)
fc1 = mx.symbol.FullyConnected(data=flatten, num_hidden=1024)
tanh3 = mx.sym.Activation(data=fc1, act_type="relu")
dropout = mx.sym.Dropout(data=tanh3, p=0.5)

# second fullc
fc2 = mx.sym.FullyConnected(data=dropout, num_hidden=10)

# softmax loss
lenet = mx.sym.SoftmaxOutput(data=fc2, name='softmax')
```

```

lenet_model = mx.mod.Module(symbol=lenet, context=ctx)

lenet_model.fit(train_iter,
                optimizer='sgd',
                optimizer_params={'learning_rate':0.001},
                eval_metric='acc',
                batch_end_callback = mx.callback.Speedometer(batch_size, ←
                                                100),
                num_epoch=50)

test_iter = mx.io.NDArrayIter(test_data, test_labels, batch_size)
# predict accuracy for lenet
acc = mx.metric.Accuracy()
lenet_model.score(test_iter, acc)
print(acc)

```

Ezek után átalakítjuk a képeket 28x28-as grayscale-es képekre, és létrehozzuk a szükséges rétegeket. Majd pedig megkezdődik a gép tanítása. minden századik kép után kiíratjuk azt, hogy hol járunk, hogy milyen gyorsan dolgozik a gép, és a gép pontosságát, illetve minden ötszázadik kép után pedig kiiratunk egy összesített pontosságot. Végül pedig egy kép a futtatásról:

```

INFO:root:Epoch[0] Batch [0-100] Speed: 573.24 samples/sec accuracy=0.124059
INFO:root:Epoch[0] Batch [100-200] Speed: 574.36 samples/sec accuracy=0.239300
INFO:root:Epoch[0] Batch [200-300] Speed: 577.09 samples/sec accuracy=0.413800
INFO:root:Epoch[0] Batch [300-400] Speed: 566.84 samples/sec accuracy=0.671100
INFO:root:Epoch[0] Batch [400-500] Speed: 557.60 samples/sec accuracy=0.814800
INFO:root:Epoch[0] Train-accuracy=0.519700
INFO:root:Epoch[0] Time cost=105.073
INFO:root:Epoch[1] Batch [0-100] Speed: 577.10 samples/sec accuracy=0.884752
INFO:root:Epoch[1] Batch [100-200] Speed: 577.59 samples/sec accuracy=0.897600
INFO:root:Epoch[1] Batch [200-300] Speed: 580.75 samples/sec accuracy=0.910000
INFO:root:Epoch[1] Batch [300-400] Speed: 581.30 samples/sec accuracy=0.916500
INFO:root:Epoch[1] Batch [400-500] Speed: 581.80 samples/sec accuracy=0.926300
INFO:root:Epoch[1] Train-accuracy=0.910533
INFO:root:Epoch[1] Time cost=103.533
INFO:root:Epoch[2] Batch [0-100] Speed: 555.08 samples/sec accuracy=0.934554
INFO:root:Epoch[2] Batch [100-200] Speed: 580.75 samples/sec accuracy=0.943100
INFO:root:Epoch[2] Batch [200-300] Speed: 580.72 samples/sec accuracy=0.941900
INFO:root:Epoch[2] Batch [300-400] Speed: 580.27 samples/sec accuracy=0.944100
INFO:root:Epoch[2] Batch [400-500] Speed: 579.73 samples/sec accuracy=0.950800
INFO:root:Epoch[2] Train-accuracy=0.944150
INFO:root:Epoch[2] Time cost=104.219
INFO:root:Epoch[3] Batch [0-100] Speed: 578.14 samples/sec accuracy=0.953960
INFO:root:Epoch[3] Batch [100-200] Speed: 578.95 samples/sec accuracy=0.954900
INFO:root:Epoch[3] Batch [200-300] Speed: 579.37 samples/sec accuracy=0.955600
INFO:root:Epoch[3] Batch [300-400] Speed: 580.06 samples/sec accuracy=0.960700
INFO:root:Epoch[3] Batch [400-500] Speed: 577.39 samples/sec accuracy=0.957500
INFO:root:Epoch[3] Train-accuracy=0.957017
INFO:root:Epoch[3] Time cost=103.608
INFO:root:Epoch[4] Batch [0-100] Speed: 577.58 samples/sec accuracy=0.966931
INFO:root:Epoch[4] Batch [100-200] Speed: 578.08 samples/sec accuracy=0.960900
INFO:root:Epoch[4] Batch [200-300] Speed: 578.24 samples/sec accuracy=0.963600
INFO:root:Epoch[4] Batch [300-400] Speed: 579.58 samples/sec accuracy=0.963900
INFO:root:Epoch[4] Batch [400-500] Speed: 577.38 samples/sec accuracy=0.965600
INFO:root:Epoch[4] Train-accuracy=0.964600
INFO:root:Epoch[4] Time cost=103.840
INFO:root:Epoch[5] Batch [0-100] Speed: 576.99 samples/sec accuracy=0.965743
INFO:root:Epoch[5] Batch [100-200] Speed: 580.76 samples/sec accuracy=0.967900
INFO:root:Epoch[5] Batch [200-300] Speed: 579.64 samples/sec accuracy=0.968100
INFO:root:Epoch[5] Batch [300-400] Speed: 580.74 samples/sec accuracy=0.969200
INFO:root:Epoch[5] Batch [400-500] Speed: 579.33 samples/sec accuracy=0.970500
INFO:root:Epoch[5] Train-accuracy=0.968583
INFO:root:Epoch[5] Time cost=103.534
INFO:root:Epoch[6] Batch [0-100] Speed: 580.14 samples/sec accuracy=0.972871
INFO:root:Epoch[6] Batch [100-200] Speed: 578.85 samples/sec accuracy=0.969500
INFO:root:Epoch[6] Batch [200-300] Speed: 572.85 samples/sec accuracy=0.970300
INFO:root:Epoch[6] Batch [300-400] Speed: 541.86 samples/sec accuracy=0.972900
INFO:root:Epoch[6] Batch [400-500] Speed: 577.08 samples/sec accuracy=0.973800
INFO:root:Epoch[6] Train-accuracy=0.971800
INFO:root:Epoch[6] Time cost=105.175

```

## **IV. rész**

### **Irodalomjegyzék**

## 20.5. Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

## 20.6. C

[KERNIGHANRITCHIE] Kernighan, Brian W. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

## 20.7. C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

## 20.8. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, [http://arxiv.org/PS\\_cache/math/pdf/0404/0404335v7.pdf](http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf) , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPORG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEAHCackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésekért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPORG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségen született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.