

# **Univerzális programozás**

---

**Írd meg a saját programozás tankönyvedet!**

Ed. BHAX, DEBRECEN,  
2019. február 19, v. 0.0.4

Copyright © 2019 Huri Patrik

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

<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>

**COLLABORATORS**

|               | <i>TITLE :</i>                      |                   |                  |
|---------------|-------------------------------------|-------------------|------------------|
|               | Univerzális programozás             |                   |                  |
| <i>ACTION</i> | <i>NAME</i>                         | <i>DATE</i>       | <i>SIGNATURE</i> |
| WRITTEN BY    | Bátfai, Norbert<br>ÁCs Huri, Patrik | 2019. november 4. |                  |

**REVISION HISTORY**

| 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  | 2019-03-05 | 2.1, 2.2, 2.4 és 2.7 Feladatok megoldása.  | huripatrik |
| 0.2.0  | 2019-03-06 | 2.3, 2.6, 2.5 és 2.8 Feladatok megoldása.  | huripatrik |
| 0.3.1  | 2019-03-13 | 3.4, 3.6, Feladatok megoldása.   | huripatrik |
| 0.3.2  | 2019-03-13 | 3.7, 3.2 Feladat kidolgozása, simítások, csokor befejezése   | huripatrik |

**REVISION HISTORY**

| NUMBER | DATE       | DESCRIPTION   | NAME       |
|--------|------------|---|------------|
| 0.4.1  | 2019-03-19 | Exor titkosító C, kisebb javítások.                           | huripatrik |
| 0.4.2  | 2019-03-20 | Caesar 4-es csokor befejezése.                                | huripatrik |
| 0.4.3  | 2019-03-22 | Belekezdtem az olvasónaplóba.                                 | huripatrik |
| 0.5.0  | 2019-04-05 | Mandelbrot feladatok megoldása.                               | huripatrik |
| 0.6.0  | 2019-04-09 | Welch csokor befejezése                                       | huripatrik |
| 0.7.0  | 2019-04-13 | A 7-es csokor hangyszimuláció feladat befejezése.             | huripatrik |
| 0.7.1  | 2019-04-20 | A 7-es csokor befejezése.                                     | huripatrik |
| 0.5.3  | 2019-04-25 | Mandelbrot feladat szerkeztése.                               | huripatrik |
| 0.9.0  | 2019-05-07 | 9-es csokorból az első feladat befejezése, a lips-es feladat. | huripatrik |
| 0.7.1  | 2019-05-08 | Olvasónapló javítása, pontosítása                             | huripatrik |
| 1.0.0  | 2019-05-09 | Tutor+Tutoriáltak.  | huripatrik |

# 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? . . . . .    | 6        |
| 2.3. Változók értékének felcserélése . . . . .               | 8        |
| 2.4. Labdapattogás . . . . .                                 | 9        |
| 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . . .    | 13       |
| 2.6. Helló, Google! . . . . .                                | 14       |
| 2.7. 100 éves a Brun téTEL . . . . .                         | 17       |
| 2.8. A Monty Hall probléma . . . . .                         | 20       |
| 3. Helló, Chomsky!   | 23       |
| 3.1. Decimálisból unárisba átváltó Turing gép . . . . .      | 23       |
| 3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen . . . . . | 25       |
| 3.3. Hivatalos nyelv . . . . .                               | 26       |
| 3.4. Saját lexikális elemző . . . . .                        | 27       |
| 3.5. l33t.l . . . . .  | 28       |
| 3.6. A források olvasása . . . . .                           | 31       |
| 3.7. Logikus . . . . .                                       | 33       |
| 3.8. Deklaráció . . . . .                                    | 34       |

|   |            |
|---|------------|
| <b>4. Helló, Caesar!</b>  | <b>37</b>  |
| 4.1. double *** háromszögmátrix . . . . .                                 | 37         |
| 4.2. C EXOR titkosító . . . . .   | 40         |
| 4.3. Java EXOR titkosító . . . . .  | 43         |
| 4.4. C EXOR törő . . . . .  | 45         |
| 4.5. Neurális OR, AND és EXOR kapu . . . . .                              | 51         |
| 4.6. Hiba-visszaterjesztéses perceptron . . . . .                         | 56         |
| <b>5. Helló, Mandelbrot!</b>  | <b>59</b>  |
| 5.1. A Mandelbrot halmaz . . . . .  | 59         |
| 5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal . . . . . | 62         |
| 5.3. Biomorfok . . . . .  | 66         |
| 5.4. A Mandelbrot halmaz CUDA megvalósítása . . . . .                     | 69         |
| 5.5. Mandelbrot nagyító és utazó C++ nyelven . . . . .                    | 72         |
| 5.6. Mandelbrot nagyító és utazó Java nyelven . . . . .                   | 79         |
| <b>6. Helló, Welch!</b>   | <b>84</b>  |
| 6.1. Első osztályom . . . . .   | 84         |
| 6.2. LZW . . . . .  | 87         |
| 6.3. Fabejárás . . . . .  | 92         |
| 6.4. Tag a gyökér . . . . .   | 93         |
| 6.5. Mutató a gyökér . . . . .  | 101        |
| 6.6. Mozgató szemantika . . . . .   | 102        |
| <b>7. Helló, Conway!</b>  | <b>104</b> |
| 7.1. Hangyszimulációk . . . . .   | 104        |
| 7.2. Java életjáték . . . . .   | 113        |
| 7.3. Qt C++ életjáték . . . . .   | 120        |
| 7.4. BrainB Benchmark . . . . .   | 129        |
| <b>8. Helló, Schwarzenegger!</b>  | <b>134</b> |
| 8.1. Szoftmax Py MNIST . . . . .  | 134        |
| 8.2. Mély MNIST . . . . .   | 134        |
| 8.3. Minecraft-MALMÖ . . . . .  | 134        |

---

|  |            |
|--|------------|
| <b>9. Helló, Chaitin!</b>  | <b>135</b> |
| 9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .                                     | 135        |
| 9.2. Gimp Scheme Script-fu: króm effekt . . . . .  | 136        |
| 9.3. Gimp Scheme Script-fu: név mandala . . . . .  | 136        |
| <b>10. Helló, Gutenberg!</b>   | <b>137</b> |
| 10.1. Juhász István: Magas szintű programozási nyelvek 1 . . . . .                           | 137        |
| 10.2. Kernigan-Ritchie: A C programozási nyelv . . . . .                                     | 138        |
| 10.3. Benedek-Levendovszky: Szoftverfejlesztés C++ nyelven . . . . .                         | 138        |
| <b>III. Második felvonás</b>   | <b>140</b> |
| <b>11. Helló, Berners-Lee!</b>   | <b>142</b> |
| 11.1. Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven . . . . .          | 142        |
| 11.2. Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II . . . . .    | 142        |
| 11.3. Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba . . . . . | 143        |
| <b>12. Helló, Arroway!</b>   | <b>145</b> |
| 12.1. OO szemlélet . . . . .   | 145        |
| 12.2. Homokozó . . . . .   | 147        |
| 12.3. Gagyi . . . . .  | 148        |
| 12.4. Yoda . . . . .   | 149        |
| 12.5. Kódolás from scratch . . . . .   | 150        |
| <b>13. Helló, Liskov!</b>  | <b>154</b> |
| 13.1. Liskov helyettesítés sértése . . . . .   | 154        |
| 13.2. Szülő-gyerek . . . . .   | 157        |
| 13.3. Anti OO . . . . .  | 159        |
| 13.4. deprecated - Hello, Android! . . . . .   | 164        |
| 13.5. Hello, Android! . . . . .  | 164        |
| 13.6. Hello, SMNIST for Humans! . . . . .  | 164        |
| 13.7. Ciklomatikus komplexitás . . . . .   | 164        |

---

|   |            |
|---|------------|
| <b>14. Helló, Mandelbrot!</b>                               | <b>166</b> |
| 14.1. Reverse engineering UML osztálydiagram                | 166        |
| 14.2. Forward engineering UML osztálydiagram                | 169        |
| 14.3. Egy esettan   | 172        |
| 14.4. BPMN  | 182        |
| 14.5. BPEL Helló, Világ! - egy visszhang folyamat           | 183        |
| 14.6. TeX UML   | 183        |
| <b>15. Helló, Chomsky!</b>                                  | <b>184</b> |
| 15.1. Encoding  | 184        |
| 15.2. OOCWC lexer   | 189        |
| 15.3. l334d1c4  | 189        |
| 15.4. Full screen   | 191        |
| 15.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció | 193        |
| 15.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció           | 209        |
| 15.7. Perceptron osztály                                    | 209        |
| <b>16. Helló, Stroustrup!</b>                               | <b>210</b> |
| 16.1. JDK osztályok   | 210        |
| 16.2. Másoló-mozgató szemantika                             | 213        |
| 16.3. Hibásan implementált RSA törése                       | 213        |
| 16.4. Változó argumentumszámú ctor                          | 218        |
| 16.5. Összefoglaló  | 220        |
| <b>17. Helló, Gödel!</b>                                    | <b>221</b> |
| 17.1. Gengszterek   | 221        |
| 17.2. C++11 Custom Allocator                                | 222        |
| 17.3. STL map érték szerinti rendezése                      | 222        |
| 17.4. Alternatív Tabella rendezése                          | 225        |
| 17.5. Prolog családfa                                       | 230        |
| 17.6. GIMP Scheme hack                                      | 230        |
| <b>18. Helló, !</b>   | <b>237</b> |
| 18.1. FUTURE tevékenység editor                             | 237        |
| 18.2. OOCWC Boost ASIO hálózatkezelése                      | 237        |
| 18.3. SamuCam   | 237        |
| 18.4. BrainB  | 238        |
| 18.5. OSM térképre rajzolása                                | 238        |

---

---

|   |            |
|---|------------|
| <b>19. Helló, Schwarzenegger!</b>                           | <b>239</b> |
| 19.1. Port scan . . . . .                                   | 239        |
| 19.2. AOP . . . . .   | 239        |
| 19.3. Android Játék . . . . .                               | 239        |
| 19.4. Junit teszt . . . . .                                 | 240        |
| 19.5. OSCI . . . . .  | 240        |
| <b>20. Helló, Calvin!</b>                                   | <b>241</b> |
| 20.1. MNIST . . . . .                                       | 241        |
| 20.2. Deep MNIST . . . . .                                  | 241        |
| 20.3. CIFAR-10 . . . . .                                    | 241        |
| 20.4. Android telefonra a TF objektum detektálója . . . . . | 242        |
| 20.5. SMNIST for Machines . . . . .                         | 242        |
| 20.6. Minecraft MALMO-s példa . . . . .                     | 242        |
| <b>IV. Irodalomjegyzék</b>                                  | <b>243</b> |
| 20.7. Általános . . . . .                                   | 244        |
| 20.8. C . . . . .   | 244        |
| 20.9. C++ . . . . .   | 244        |
| 20.10Lisp . . . . .   | 244        |

# Ábrák jegyzéke

|   |    |
|---|----|
| 3.1. Decimálisból unárisba átváltó Turing gép ábrája. | 24 |
|---|----|

# 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?

### 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 csemegeznék 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 videó:

Megoldás forrása:

[Vegtelen1](#)

[Vegtelen2](#)

[Vegtelen3](#)

Tanulságok, tapasztalatok, magyarázat...

A c programunkban, ha azt szeretnénk elérni, hogy 1 mag 100%-on fusson, akkor ahhoz használnunk kell egy for ciklust a main függvényen belül, ahol a for-nak, csak 1 szálat adunk oda, és azt az egy szálat fogja maximálisan kihasználni ezáltal, elérjük azt, hogy a programunkban egy mag 100%-on fusson. A mainbe elhelyezünk még egy return-t ami egy kifejezés, amely a visszaadandó étéket jelent.

Vegtelen ciklus 100% (egy mag)

```
int
main ()
{
    for (;;);

    return 0;
}
```

Ahhoz, hogy elérjük, hogy a programunk 0%-ban használja a CPU-t, be kell importálnunk az unistd.h-t, ugyanis ezt a függvénykönyvtárat kell használnunk a sleep parancshoz amit alkalmaznunk kell majd a cikluson belül ahhoz, hogy a CPU 0%-on fusson. Így az importálással kell kezdeni, majd az előző feladathoz hasonlóan a main-en belül egy for ciklus és itt kell használnunk sleep-et, ami azért kell, felfüggeszti az adott szál végrehajtását, így elérjük azt, hogy a programunkban a CPU 0% legyen.

```
Vegtelen ciklus 0%
#include <unistd.h>
int
main ()
{
    for (;;)
        sleep (1);

    return 0;
}
```

Az utolsó végtelen ciklusos feladatban, azt szeretnénk eléni, hogy az összes mag 100%-on fusson. Ehhez be kell importálnunk az omp.h függvénykövtárat, mivel ezzel tudjuk majd elérni a későbbiekben azt, hogy a programunkban több szál is 100%-on fusson. Ahhoz, hogy a programunk sikeres legye, és lefusson az omp paralellt kell használnunk(#pragma omp parallel). Az omp paralellnek az irányelv az, hogy utasítsa a fordítót arra, hogy párhuzamosítsa a kiválasztott kódblokkot. Ezt az omp.h-t kell használjuk ahhoz, hogy elérjük az összes mag(szál) 100%-on fusson. Számomra nem volt elérhető ez a könyvtár, ezért telepítettem kellett. Ehhez (MACOS-en), ha még nincs telepítve, vagy nem elérhető a megfelelő pack, akkor ezt a brew install libomp parancssal tudjuk telepíteni amit a terminálba kell beírnunk. (ide majd a terminálról kép.)

A paralellről tudni kell azt, hogy csak a szabad magokat “párhuzamosítja”, ez azt jelenti jelen esetben csak a szabad magok fognak 100%-on futni a programunkban.

Vegtelen ciklus 100%, az összes mag

```
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    {

        for (;;)
        {

        }
    }
    return 0;
}
```

## 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:

Megoldás forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a Lefagy függvényt, amely tetszőleges programról el tudja dönteni, hogy van-e benne végtelen ciklus:

```
Program T100
{
    boolean Lefagy(Program P)
    {
        if (P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

    main(Input Q)
    {
        Lefagy(Q)
    }
}
```

A program futtatása, például akár az előző v.c ilyen pszeudódjára:

```
T100(t.c.pseudo)
true
```

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra épőlő Lefagy2 már nem tartalmaz feltételezett, csak csak konkrét kódot:

```
Program T1000
{
    boolean Lefagy(Program P)
    {
        if (P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

    boolean Lefagy2(Program P)
    {
        if (Lefagy(P))
            return true;
        else
```

```
    for(;;);
}

main(Input Q)
{
    Lefagy2(Q)
}

}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true
- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

akkor most hogy fog működni? Sehogy, mert ilyen Lefagy függvényt, azaz a T100 program nem is létezik.

Tanulságok, tapasztalatok, magyarázat...

## 2.3. Változók értékének felcserélése

Írj olyan C programot, amely felcseréli két változó értékét, bármiféle logikai utasítás vagy kifejezés nasználata 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:

csere

csere1

csere2

Tanulságok, tapasztalatok, magyarázat...

A változócsere programnak a lényege, hogy felcseréljük két változó értékét. Ismét beimportáljuk az stdio.h könyvtárat, majd a mainben dolgozunk, ahogy eddig is ezt tettük. A main-en belül az int változóval, amit arra használunk, hogy egész típusú változót dekrrarálunk, a mi esetünkben számokat. Megadtuk, hogy az első szám(v1) értéke legyen 5, a második szám(v2) értéke pedig legyen 10. A printif parancssal kiiratjuk a 2 számot. Miután ezt megtettük dekraláljuk a csere segédváltozót, amellyel végrehajtjuk a "buborékos" cserét. Ezt követően már csak annyi dolgunk van, hogy kiiratjuk ismét a 2 számot, és megnézzük, hogy valóban sikerült-e a 2 változó felcserélése. Ha sikerrel jártunk akkor v1-nél 10 értéket kell kapunk, v2-nél pedig 5-ös értéket.

Két változó cseréje

```
#include<stdio.h>

int main()
{
    int v1 = 5, v2= 10;

    printf("v1=%d v2=%d\n",v1,v2);

    int csere;
    csere = v1;
    v1 = v2;
    v2 = csere;

    printf("v1=%d v2=%d\n",v1, v2);

    return 0;
}
```

## 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 videónkon.)

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

pattog

Tutor: Fenyvesi Mátyás

A fenti megoldás videóban szereplő program forráskódját használtam fel, ennek alapján írtam le tapasztalataimat. Ennél a programnál megpróbáljuk imitálni azt, hogy milyen az amikor a labda visszapattan a földről, úgy mint a való életben. Ehhez 3 függvénykönyvtárat kell beimportálnunk, az stdio.h-t a unistd.h-t (az usleep) miatt, illetve a curses.h-t a további utasításokhoz. A mainben kell egy ablak, amiben le fog futni a program illetve létrehozzuk az x, y változókat, ahol az x az oszlopot jelenti az y pedig a sort, nekik egy kezdeti értéket adunk ami a 0. Ezt követően derkaráljuk az xnov-ot és az ynov-ot ami a függőleges és vízszintes eltolás miatt lesz lényeges illetve dekraráljuk még az mx(oszlopok számát) és az my(sorok számát) is. Kell nekünk egy utasítás ami végtelen ezt a for ciklussal adjuk meg, ez azt jelenti, hogy a program nem fog leállni csak, ha mi magunk állítjuk majd le.

Labda pattogás `if`-el

```
#include <stdio.h>
#include <curses.h>
#include <unistd.h>
```

```
int
main ( void )
{
    WINDOW *ablak;
    ablak = initscr ();

    int x = 0; // oszlop
    int y = 0; // sor

    int xnov = 1;
    int ynov = 1;

    int mx; //oszlopok száma
    int my; //sorok száma

    for ( ; ; ) {
```

A getmaxyx-ra azért van szükségünk, hiszen ez az adott ablakban a kurzor pozícióját helyezi el két változóban x és y-ban. Az mvprintw-vel megjelenítük az ablakban és megadhatjuk milyen legyen a “minta” jelen esetben egy O betű. Az usleep-el megadhatjuk, hogy milyen gyorsan vagy lassan mozogjon a minta az ablakunkban, viszont fontos az, hogy ha érdemi eredményt szeretnénk látni akkor ezt az értéket minimum 50000-re állítsuk, így még szemmel egyszerűen követhető a folyamat.

```
Labda pattogás if-el
    getmaxyx ( ablak, my , mx );

    mvprintw ( y, x, "O" );

    refresh ();
    usleep ( 100000 );
```

Az if utasításokkal pedig azokat az eseteket vizsgáljuk amikor a mintánk eléri e az adott oldalakat (bal, jobb oldal, felül, alul), amikor eléri valamelyik oldalt, akkor “visszapattan”, ezért szorozzuk meg -1 -el. Így elérjük azt, hogy egy “labdapattogást” imitálunk. Érdekesség továbbá az, hogy a for cikluson belül a getmaxyx után megadhatjuk az mx my értékét amivel szabályozni tudjuk azt, hogy az ablakunkból mekkora helyen végződjön a labdapattogás.

```
Labda pattogás if-el
    x = x + xnov;
    y = y + ynov;

    if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
        xnov = xnov * -1;
    }
    if ( x<=0 ) { // elerte-e a bal oldalt?
        xnov = xnov * -1;
    }
```

```
    if ( y<=0 ) { // elerte-e a tetejet?
        ynov = ynov * -1;
    }
    if ( y>=my-1 ) { // elerte-e a aljat?
        ynov = ynov * -1;
    }

}

return 0;
}
```

Az if nélküli megoldáshoz, be kell importálnunk az stdlib.h könyvtárat ami a (void)system(" ") -hez kell majd. A static void-ban létrehozzuk az x, y változókat, ahol x sor y oszlopot jelöl. Deklarálunk i-t a for ciklushoz és ha az i kisebb mint az y akkor lefelé toljuk, illetve a másik for ciklusnál ha az i az kisebb min az x akkor pedig jobbra toljuk az alakzatunkat. A printf-en belül megadjuk milyen ikonja legyen a "labdánknak"

```
Labda pattogás if nélkül
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

static void gotoxy(int x, int y)
{
    int i;
    for(i=0; i<y; i++) printf("\n");
    for(i=0; i<x; i++) printf(" ");
    printf("o\n");
}
```

A main-en belül ismét deklarálunk az egyx és az egyy-nal megadjuk, hogy mennyit menjel le és oldalra a labdánk milyen mértékben változzon a labda pozíciója. Az i az egy ciklus változó lesz ismét amit majd megint a for ciklusnál használunk majd. Az x,y a labda kezdeti pozícióját adja, míg a ty a magasságát, a tx pedig a szélességét adja meg, ezzel megadtuk, hogy mekkora méretű legyen a pályánk. A for ciklussal meghatározzuk a pálya szélességét, illetve magasságát is, majd egy végtelen ciklust hozunk létre. Alább a 2 for ciklus és a 2 printf-el az ablakon szemléltetjük a pozícióját a labdánknak.

```
Labda pattogás if nélkül
int main(void)
{
    int egyx=1;
    int egyy=-1;
    int i;
    int x=10;
```

```
int y=20;
int ty[23];
int tx[80];

for(i=0; i<23; i++) {
    ty[i]=1;
}
ty[1]=-1;
ty[22]=-1;

for(i=0; i<79; i++) {
    tx[i]=1;
}
tx[1]=-1;
tx[79]=-1;

for(;;)
{
    for(i=0; i<36; i++)
        printf("_");

    printf("x=%2d", x);
    printf("y=%2d", y);

    for(i=0; i<=35; i++)
        printf("_");
}
```

A void szekcióban a gotoxy függvényt felhasználva, amit ugye a programunk elején megadtunk, itt mozgatjuk a pozíciót az  $x+=egyx$  illetve a  $y+=egyy$ -nal. Az  $egyx^*=tx[x]$ -el és az  $egyy^*=ty[y]$ -al pedig a sebességét változtatjuk úgy, hogy megnézzük, hogy hol helyezkedik el és ennek függvényében szorozzuk meg. A usleep-et felhasználva csak úgy mint if-el megadhatjuk, hogy milyem gyorsan illetve lassan mozogjon a labda ikonunk, érdemes a minimum értéket 50000-re állítani. A system adja át a parancsot/utasítást a shell-nek, a terminálnak, ezzel tudjuk végrehajtani a clear-t.

```
Labda pattogás if nélkül
(void)gotoxy(x,y);
//printf("o\n"); Áthelyezve a gotoxy függvénye

egyx+=egyx;
egyy+=egyy;

egyx*=tx[x];
egyy*=ty[y];
```

```
    usleep (200000);
    (void) system("clear");
}

}
```

## 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:

Tanulságok, tapasztalatok, magyarázat...

**szohossz**

Ahhoz hogy megnézzük hány bites a szó a gépünkön, importálnunk kell ismét az stdio.h függvénykönyvtárat, majd a main-en belül létre kell hoznunk két változót. Az egyik a bittolást fogja vizsgálni, a másik pedig számolni fogja azt, hogy hányszor tolta el. A számláló értéké legyen 0(i), amivel pedig a "tolást" vizsgáljuk annak pedig az értéke legyen 1(j). A Linus Torvalds által felhasznál while ciklus fejet felhasználva, a while addig fut amíg nem lesz egyenlő 0-val. Ezt követően az j-t addig tolja minden 1-el arrébb ameddig el nem éri a 0-t, tehát ameddig nem tudja, ekkor az i értéket növeljük minden eggyel, ő számolja meg nekünk majd, hogy hány bit-es lesz. A cikluson kívül, egy printf-el kiiratjuk, hogy hány bit-es a szó a gépünkön, és odaadjuk neki az i-t, majd egy return-el végeünk. Így ha minden jól csináltunk, akkor megkapjuk a várva várt eredményt.

Szóhossz

```
#include <stdio.h>

int main()
{
    int i = 0;
    int j = 1;

    while(j!=0) {
        j <= 1;
        i++;
    }

    printf ("A szohossz ezen a gépen: %d bites\n", i);
    return 0;
}
```

## 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:

[pagerank](#)

Tanulságok, tapasztalatok, magyarázat...

A PageRank egy olyan algoritmus, amely hiperlinkekkel összekötött dokumentumhoz számokat rendel a hiperlink hálózatban betöltött szerep alapján. Ehhez a programhoz be kell importálnunk a már jól ismert stdio.h-t illetve a math.h függvénykönyvtárat az sqrt művelet miatt. A void kiirban deklalárunk egy i-t amit a ciklusban használunk fel, lényegében ezt kiiratja azt a tömböt amit odaadunk neki. A távolsághoz double változót használunk az egész és tört számok miatt, a double kétszeres pontosággal képes tárolni a valós számokat. A távolságos részben deklarálunk egy i-t amit majd a for ciklusnál fogunk használni, illetve double-e, egy összeget aminek értéke legyen 0. A ciklus után azt vizsgáljuk, hogy az összeget négyzetre emeled akkor sem lehet negatív értéke, ugyanis a gyök alatt negatív szám nem szerepelhet, ezzel ezt próbáljuk kizární.

PageRank

```
#include <stdio.h>
#include <math.h>

void
kiir (double tomb[], int db) {

    int i;

    for (i=0; i<db; ++i)
        printf("%f\n",tomb[i]); // kiirja azt a tömböt amit odaadsz neki.
    }

}

double
tavolsag (double PR[], double PRv[], int n) {

    int i;
    double osszeg=0;

    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]); // nem lehet negatív ←
    értéke, ez csak ennyire jó (ezt négyzetre emeled)

    return sqrt(osszeg);
}
```

A void pagerank részben 2 tömböt hozunk létre, az elsőbe megy az aktuális eredmény amit vizsgálunk. A második tömbbe pedig kerül majd az előző PR értéke, ezekkel az értékekkel szorozzuk meg azokat az adatokat amiket a mainben találunk. Ezt követően létrehozzuk a mátrix műveletet, ehhez deklarálnunk kell egy i-t és egy j-t, majd létrehozunk egy végtelen ciklust. Ezt követően jön a tényleges mátrix művelet, ahol az első ciklusban soronként megyünk végig a tömbön, ha az i kisebb mint 4 akkor belép a ciklusba és a PR[i], tehát az aktuális tömböt 0.0 értékkel látja el, majd. Következő ciklussal az oszloponként megy végig a tömbön és elvégzi a műveletet, majd ezt eltárolja a PR[i]-ben.

PageRank

```
void
    pagerank(double T[4][4]) {
double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény ez az ←
    aktuális
double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0}; //ezzel szorzok ez ←
    pedig az előző PR értéke.

int i, j;

for(;;) {

    // ide jön a mátrix művelet

    for (i=0; i<4; i++){ // Soronként megyünk végig a tömbön
        PR[i]=0.0;
        for (j=0; j<4; j++){ //Oszloponként megy végig a tömbön.
            PR[i] = PR[i] + T[i][j]*PRv[j];
        }
    }
}
```

A az if függvényben megadjuk, ha ez a bizonyos érték amit az előbb vizsgáltunk kisebb mint az általunk megadott nagyon kicsi érték, akkor szakítsa félbe a folyamatot. Mivel ezt a folyamatot szinte a végtelen-ségi tudná csinálni mivel egyre csak közelít a 0 értékhez, de soha nem éri azt el, azaz konvergens, ezáltal megszakításra kényszerítjük őt. Ha ezekkel az utasításokkal végeztünk akkor átpakoljuk az előző PRv[i] tömbből, az aktuálisba, így már más értékekkel fogunk dolgozni. Ezt követően pedig kiiratjuk a PR-t

PageRank

```
if (tavolsag(PR, PRv, 4) < 0.0000000001)
break;

// ide meg az átpakolás PR-ből PRv-be

for (i=0;i<4; i++){
    PRv[i]=PR[i];
}
}

kiir (PR, 4);
}
```

A mainban deklaráltunk 3 tömböt, az első az eredeti tömb, itt az eredeti mátrix értékeivel fut le a program. A második tömbben az egyik oldal semmire sem mutat, itt érjük el azt, hogy az érték nagyon közelít a 0-hoz, de igazából sosem fogja elérni azt, ezt úgy lehet elképzelni, hogy van egy lyukas hordód amibe folyamatosan engedsz vizet, mindenkor csak úgy lehetséges előre látogatni, hogy a hordóban van egy lyuk, amiből nem tudsz kiemelni a vizet, de sohasem ürül majd az ki teljesen. A harmadik tömb pedig azt deklaráljuk, hogy mindenkor az egyik oldal magára mutat. Itt az előző tömbbhöz hasonlóan csak itt fordítva, itt az értékünk közelít az 1-hez, de nem fogja azt elérni, így ez is konvergens. A program végén a 3 printf-el kiiratjuk a 3 különböző tömbök eredményeit.

## PageRank

```
int main (void) {
double L[4][4] = {
{0.0, 0.0, 1.0/3.0, 0.0},
{1.0, 1.0/2.0, 1.0/3.0, 1.0},
{0.0, 1.0/2.0, 0.0, 0.0},
{0.0, 0.0, 1.0/3.0, 0.0}
};

double L1[4][4] = {
{0.0, 0.0, 1.0/3.0, 0.0},
{1.0, 1.0/2.0, 1.0/3.0, 0.0},
{0.0, 1.0/2.0, 0.0, 0.0}, // Közelít a 0-hoz, de nem éri el . ←
    konvergens és nem mutat semmire
{0.0, 0.0, 1.0/3.0, 0.0}
};

double L2[4][4] = {
{0.0, 0.0, 1.0/3.0, 0.0},
{1.0, 1.0/2.0, 1.0/3.0, 0.0}, // Közelít az egyhez de nem éri el, ez ←
    is konvergens, önmagára mutat.
{0.0, 1.0/2.0, 0.0, 0.0},
{0.0, 0.0, 1.0/3.0, 1.0}
};

printf("\nAz eredeti mátrix értékeivel történő futás:\n");
pagerank(L);

printf("\nAmikor az egyik oldal semmire sem mutat:\n");
pagerank(L1);

printf("\nAmikor az egyik oldal csak magára mutat:\n");
pagerank(L2);

printf("\n");

return 0;
}
```

## 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 feladat teljesítéséhez, a megoldás videóban szereplő forráskódot felhasználva próbáltam ki a programot, és ez alapján értelmeztem a feladatot. Ahhoz, hogy ez a feladatot megoldjuk, és a programunk működjön, legyen eredménye, először is telepítenünk kell az R nyelvet. Az R egy programozási nyelv, ami arra szolgál, hogy különböző számításokat, egyéb matematikai feladatokhoz alkalmas ez a nyelv. Ebben az R programozási nyelvben, matematikusok adnak meg különböző csomagokat, amiket mi fogunk majd használni. Nekünk most a matlab csomagra lesz szükségünk, ezt kell feltelepítenünk.

A Brun téTEL azt mondja ki, miszerint az ikerprímek reciprokosszege egy véges vagy végtelen értékhez konvergál, ez annyit jelent, hogy a határ számot, eldönti, hogy az ikerprímszámok halmaza végtelen-e ezt  $B_2$  Brun konstansnak nevezik. Ikerprím: Az ikerprím egy olyan prímpár, melyek különbsége 2.

A programunkban a primes(első sor) kiirja (x)-ig a prímszámokat. Adjunk meg például 50-et, ekkor kiirja 50-ig a prímszámokat.

```
primes = primes(x)
primes(50)
```

Ebben esetben a következőt kapjuk:

```
primes(50)
[1] 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

A length(primes) megmondja nekünk, hogy milyen hosszú a vektor(hány elemet tartalmaz).

```
length(primes)
```

A kapott eredmény a length-re.

```
length(primes)
[1] 15
```

A diff sorban, elkezdjük az ikerprímek keresését. Itt a 2. elemtől indulva az utolsó -1 -ig vizsgálja a prímszámok különbségét.

```
diff = primes[2:length(primes)]-primes[1:length(primes)-1]
```

A kapott eredmény diff esetén:

```
diff
[1] 1 2 2 4 2 4 2 4 6 2 6 4 2 4
```

Az idx esetében, azt vizsgákljuk, hogy ez az előző érték (amit diff néven adtunk meg), mikor lesz 2(mert ugye ekkor beszélünk okerprímről). Ezzel megtudhatjuk azt, hogy hanyadik helyen fordulnak elő az ikerprímek.

```
idx = which(diff==2)
```

Kapott eredmény idx esetén:

```
idx  
[1] 2 3 5 7 10 13
```

A t1primes a párok első száma. A t2primes-nél hozzá kell adnunk 2-t, hogy megkapjuk az ikerprímet.

```
t1primes = primes[idx]
```

A t1primes esetén kapott eredmény:

```
t1primes  
[1] 3 5 11 17 29 41
```

```
t2primes = primes[idx]+2
```

Itt, ugye ebben az esetben hozzá adtunk 2-t az ikerprím miatt. Az így kapott eredmény:

```
t2primes  
[1] 5 7 13 19 31 43
```

Az rt1plust2-ben képezzük a reciprokot Majd ezeket az értékeket össze kell adnunk a sum függvényel, és megkapjuk a végeredményt.

```
rt1plust2 = 1/t1primes+1/t2primes
```

Az rt1plust2-ben kapott eredmény:

```
rt1plust2  
[1] 0.53333333 0.34285714 0.16783217 0.11145511 0.06674082 ←  
0.04764606
```

Ezeket az értékeket összeadva a végeredmény:

```
sum(rt1plust2)  
[1] 1.269865
```

Ha ezt az egészet betesszük az R nyelvben egy függvénybe, a function-t használva, itt jelenesetben az stp-vel hivatkozunk rá, akkor ha beírjuk, hogy stp(50), akkor ugyanazt az eredményt fogjuk kapni, amit az előbb levezettünk.

```
stp(50)
[1] 1.269865
```

Így könnyen megtudhatjuk, hogy 100-ig a prímszámoknak mennyi az ikerprímek reciprokösszege, de akár 1000-ig stb, és így tovább. Példa a Brun tételere 100-ig a prímszámok:

```
stp(100)
[1] 1.33099
```

A programunk utolsó 3 sorában pedig, grafikusan ábrázoljuk kapott eredményt, amivel egy rajzot kapunk.

```
x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

A program forráskódja:

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
library(matlab)

stp <- function(x) {

  primes = primes(x)
  diff = primes[2:length(primes)]-primes[1:length(primes)-1]
  idx = which(diff==2)
  t1primes = primes[idx]
  t2primes = primes[idx]+2
  rt1plust2 = 1/t1primes+1/t2primes
  return(sum(rt1plust2))
}

x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

## 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)

Tanulságok, tapasztalatok, magyarázat...

A feladat értelmezéséhez a megoldás forrásában szereplő forráskódöt használtam fel, ez alapján teszteltem le a programot. A Monty-Hall: Adott számunkra 3 ajtó, ami csukva van, nem tudjuk mi van mögötte. Legyen a főnyeremény egy Ferrari.

Ez a Ferrari a 3 ajtó valamelyike mögött van, nem tudjuk, hogy melyik mögött. A másik 2 ajtó mögött egy régi rossz trabant van. A cél az, hogy megnyerjük a Ferrarit. Erre a legnagyobb esélyünk akkor van ha követjük a Monty Hall problémát.

Ugyanis ha követjük a Monty-Hall-t, akkor választunk egy adott ajtót(Ekkor ugye ebben a szituációban csak 33,3% esélyünk van eltalálni a Ferrarit), a maradék kettő közül a műsorvezető kinyit egy ajtót ami mögött egy trabant van. Itt jön a lényeg, amikor is majd felteszi a műsorvezető azt a kérdést, hogy maradunk- e az általunk választott ajtónál, vagy változtatunk, és a másik ajtó mellett tesszük le a voksunkat(értelemszerűen a másik csukott ajtó mellett, nem annál amelyik mögött már a trabant van.). Egyfajta döntés elé állít minket a műsorvezető. Sok ember foglalkozott már ezzel a problémával, főként matematikusok, akik különböző programokkal, bebizonyították azt, hogy megéri változtatni a döntésünkön, ugyanis ebben az esetben már 66,7% esélyünk van arra, hogy eltaláljuk a megfelelő ajtót, így nagyobb az esélyünk arra, hogy megnyerjük a Ferrarit.

Ezzel a Monty-Hall problémával, egy film is foglalkozott, ahol a diáktanár szemszögéből ismertetik, lát-hatjuk a Monty-Hall-t. Több információt a filmről itt találsz:

A forráskódunkban az elején megadjuk a kísérletek számát. Jelen esteben ez most : 10000000.

```
kiserletek_szama=10000000
```

Megadunk 2 kísérlet vektort, majd ezeket feltöljük az 1,2,3 számmal, ez szimbolizálja majd nekünk a 3 ajtót, ahol az egyik mögött ott a Ferrari. A vektorban megtalálható, a játékos tippje, a másikban pedig a jó azaz a nyertes ajtó.

```
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
```

Megadunk még egy kísérlet számú méretű vektort, ahol azt tároljuk, hogy melyik ajtót nyitja ki a műsorvezető.

```
musorvezeto=vector(length = kiserletek_szama)
```

A for cílus az 1-től fog menni a kísérletek számáig, melyben egy feltétel szerepel, ha a játékos tippje azonos akkor a mibol vektorba eltaroljuk, hogy a műsorvezető miből válogat majd. Egyébként pedig tároljuk az 1,2,3 halmazból a játé kostippet tippet és a győztes ajtó műveletét.

```
for (i in 1:kiserletek_szama) {  
  if(kiserlet[i]==jatekos[i]) {  
    mibol=setdiff(c(1,2,3), kiserlet[i])  
  }else{  
    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))  
  }  
}
```

A ciklus vége előtt megadjuk, a műsörvezető i-edik elemét a mibol vektorbol választott random elemre.

```
musorvezeto[i] = mibol[sample(1:length(mibol),1)]
```

A which függvény eltárolja az indexeket, ez azt adja meg mikor lett volna az a lehetőság amikor nem változtatunk akkor megnyerjük a Ferrarit. Létrehoztunk egy kisérletszám méretű vektort, ahol a változtatás ( Ha változtatunk egy ajtót) fog menni.

```
nemvaltoztatesnyer= which(kiserlet==jatekos)  
valtoztat=vector(length = kiserletek_szama)
```

Ezt követően lefut még 1 ciklus 1-től a kiserletek számáig. Itt a műsörvezető i-edik eleme és a jatekos tippelt i-edik elemét az ajtoszámokat rakjuk el, és belehelyezzük a holvált vektorba.

```
for (i in 1:kiserletek_szama) {  
  holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
  valtoztat[i] = holvalt[sample(1:length(holvart),1)]  
}
```

A ciklus után a which függvényben azt vizsgáljuk, hogy hányszor nyertünk ha ajtót változtattunk.

```
valtoztatesnyer = which(kiserlet==valtoztat)
```

A végén pedig kiíratjuk az eredményeket.

```
sprintf("Kiserletek szama: %i", kiserletek_szama)  
length(nemvaltoztatesnyer)  
length(valtoztatesnyer)  
length(nemvaltoztatesnyer)/length(valtoztatesnyer)  
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

A program forráskódja:

```
kiserletek_szama=10000000  
kiserlet = sample(1:3, kiserletek_szama, replace=T)  
jatekos = sample(1:3, kiserletek_szama, replace=T)  
musorvezeto=vector(length = kiserletek_szama)  
  
for (i in 1:kiserletek_szama) {  
  if(kiserlet[i]==jatekos[i]) {  
    mibol=setdiff(c(1,2,3), kiserlet[i])  
  }  
  musorvezeto[i] = mibol[sample(1:length(mibol),1)]  
}  
  
nemvaltoztatesnyer= which(kiserlet==valtoztat)  
valtoztat=vector(length = kiserletek_szama)  
  
for (i in 1:kiserletek_szama) {  
  holvart = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
  valtoztat[i] = holvart[sample(1:length(holvart),1)]  
}  
  
valtoztatesnyer = which(kiserlet==valtoztat)  
  
sprintf("Kiserletek szama: %i", kiserletek_szama)  
length(nemvaltoztatesnyer)  
length(valtoztatesnyer)  
length(nemvaltoztatesnyer)/length(valtoztatesnyer)  
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

```
    }else{
        mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))
    }
    musorvezeto[i] = mibol[sample(1:length(mibol),1)]
}

nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {
    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i] -->
        )))
    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]
}

valtoztatesnyer = which(kiserlet==valtoztat)

sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

### 3. fejezet

## Helló, Chomsky!

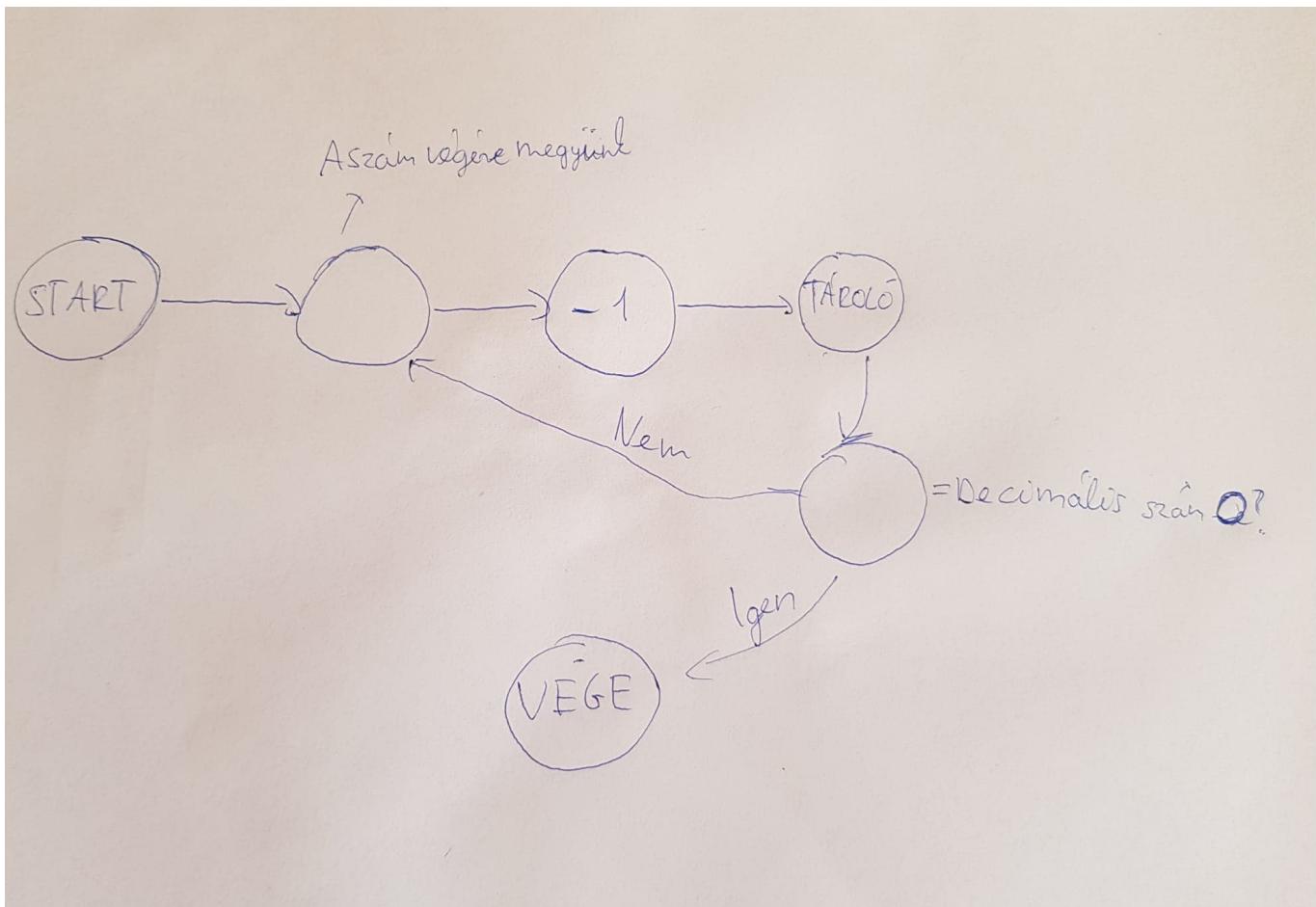
### 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó: -

Megoldás forrása: -

Tanulságok, tapasztalatok, magyarázat...



3.1. ábra. Decimálisból unárisba átváltó Turing gép ábrája..

A Turing-gép egy absztrakt automata, ami a valóságos digitális számítógépek egy nagyon leegyszerűsített modellje.

A Turing-gép egy univerzális algoritmikus modell, mely matematikai számítástudománynak olyan részterületei foglalkoznak, mint például a számításelmélet.

A gép feladata annyi, hogy Decimális azaz 10-es számrendszerből állítson elő egy unáris alakot azaz egyes számrendszer beli alakot hoz létre.

Működése: Bemenetként megadunk neki egy decimális számot, ezt karakterenként beolvasva végig megy rajta. Utolsó karakterből fog kivonni.

Ha kivont egyet, akkor ír egy 1-est a tárolóba.

Ha a végén 0-át talál akkor 9-el helyettesíti és vizsgálja tovább és ha 0-át talál vizsgál megint, addig ameddig nem 0-át talál, ha nem 0-át talál akkor ebből is kivon 1-et majd visszatér a számnak a végére.

Mindig levon egyet, és ellenőrzi, hogy a decimális szám elfogyott-e már.

Ha nem akkor újra levon 1-et és beírja a tárba.

Ha pedig elfogyott akkor, a gép elvégezte az átalakítást, így a kimenete az a decimális szám lesz aminek formája unáris lesz.

## 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: -

Tanulságok, tapasztalatok, magyarázat...

A formális nyelv egyik legmeghatározóbb fajtája a generatív nyelvtan. A generatív nyelvtan a valóságban egy algoritmust formalizál, ami a nyelv összes jelsorozatát generálja.

A generatív nyelvtanokat 4 csoportja van, ezeket Noam Chomsky úgy adta meg, hogy van a legtágabb ami a 0,1,2 és a legszűkebb ami a 3.

Általános a 0-s típus.

Környezetfüggő az 1-es típus.

Környezetfüggetlen 2-es típus.

Reguláris 3-mas típus.

A  $G = \{N, T, S, H\}$  egy rendezett négyes, amit generatív grammaticának (vagy generatív nyelvtannak) nevezünk, de akkor az N és T diszjunk halmaz. N-ben változókat tárolunk, T-ben pedig konstansokat.

Az S (nemterminális nyelv), ō lesz a nyelv kezdő eleme.

N elemeit nemterminális jeleknek nevezzük, ezeket általánosságban nagy betűkkel adjuk meg (S, A, B, C).

T elemeit terminális jeleknek nevezzük, ōket kisbetűvel jelöljük általánosságban. (a, b, c).

H elemeit (p, q) képezi amik rendezett párok, egyfajta helyettesítési szabály, amit  $p \rightarrow q$  alakban írunk fel.

1. Környezetfüggő generatív grammatica.

S, X, Y "változók"

a, b, c "konstansok"

$S(\text{kezdőelem, szabályok}) \rightarrow abc, S \rightarrow aXbc, Xb \rightarrow bX, Xc \rightarrow Ybcc, bY \rightarrow Yb, \leftarrow aY \rightarrow aaX, aY \rightarrow aa$

S-ből indulunk ki

-----  
S (S  $\rightarrow$  aXbc)

aXbc (Xb  $\rightarrow$  bX)

abXc (Xc  $\rightarrow$  Ybcc)

abYbcc (bY  $\rightarrow$  Yb)

aYbbcc (aY - aa)

aabbcc

## 2. Környezetfüggő generatív grammatika.

A, B, C "változók"

a, b, c "konstansok"

A (kezdőelem, szabályok)  $\rightarrow$  aAB, A  $\rightarrow$  aC, CB  $\rightarrow$  bCc, cB  $\rightarrow$  Bc, C  $\rightarrow$  bc

S-ből indulunk ki

-----  
A (A  $\rightarrow$  aAB)

aAB (A  $\rightarrow$  aC)

aaCB (CB  $\rightarrow$  bCc)

aabCc (C  $\rightarrow$  bc)

aabbcc

## 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:

else

masodik

Tanulságok, tapasztalatok, magyarázat...

Tutorált: [Bacsik Mátyás](#)

A BNF Backus-Naur-féle forma, egy használt jelölésmód, egy programozási nyelv szintaxisának leírására, formális nyelveket is írhatunk vele. A legtöbb programozási nyelv elméleti leírása általában BNF-ben

vannak leírva. Ezt a jelölés rendszert John Backus hozta létre. Egy BNF leírás valójában leszármaztatási szabályok halmaza.

```
int main() {
    for(int i = 1; i>0; i++) {
}
```

A megadott programban megfigyelhetjük azt, hogy ha úgy akarjuk lefordítani a programot, hogy gcc -o hivatkozasnyelv hivatkozasnyelv.c -std=c89, ekkor egy hibába ütközünk mivel nem for működni a program, ugyanis felszólít minket arra, hogy a for ciklus fejrészében még nem lehetett deklarálni változót. Ha ugyanezt a programot azon lefuttatjuk úgy, hogy gcc -o hivatkozasnyelv hivatkozasnyelv.c -std=c99, akkor a program le fog fordulni, és futni is, ugyanis, ebben a verzióban már lehet deklarálni a for ciklus fejrészében a változót.

### 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: [https://youtu.be/9KnMqrkj\\_kU](https://youtu.be/9KnMqrkj_kU)

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat elemzésében,felfogásában sokat segített a megoldás videóban szereplő forrá, ennek segítségével sikerült lefuttatnom a programot. A programmal lexikális szabályokból elemző programkódot készíthetünk. Jelen esetben ez a program beolvashat egy forráskódöt, ahol felismeri a valós számokat a beolvasásnál, és szemlélteti ezt nekünk.

A lexikális elemző program 3 részből áll, ezek %% jellel vannak elválasztva, elkülönítve. Az első részben a definíciók vannak, ezek biztosan benne lesznek a C programunkban, így jelen esetben ide deklaráljuk a realnumbers változót.

```
% {
    #include <stdio.h>
    int realnumbers = 0;
}
digit [0-9]
```

A második részben a fordítási szabályokat adjuk meg, itt már használjuk az előző részben megadott definíciót, amiben ugye megadtuk, hogy számokat keresünk, tehát egy számjegyet várunk, amiből akármennyi lehet, akár 0 is, ezután pedig azt vizsgálja, hogy a pont után, van-e legalább 1 db számjegy, ez lehetséges e vagy sem, erre utal a kerdőjel a végén. Ha talál ilyet akkor növeljük a vszamok nevű változót, majd kiiratjuk stringként, és számként is.

```
%%
{digit}*(\.{digit}+)? {++realnumbers;
```

```
printf("[realnum=%s %f]", yytext, atof(yytext));}
```

Az utolsó részben pedig a yylex () függvényel hívjuk a lexikális elemzést és el is indítjuk vele, és amikor ennek a függvénynek vége van, akkor kiirjuk a vszámokat. Ezzel a programmal ki tudjuk “szűrni” a valós számokat a bemenetnél, ha csak betű karaktereket adunk meg, akkor azt is kapjuk vissza, ha viszont beleírunk a betűkbe karaktereket is akkor azt a programunk felismeri és szemlélteti, hogy valós számot talált a bemenetben.

```
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", ←
           realnumbers);
    return 0;
}
```

A lexikális elemző program az l forrásból készít egy c forráskódöt, amit már le tudunk fordítani, így amikor ezt lefordítottam, a kvetkező eredményt kaptam meg:

```
Administrator-MacBook-Pro:bin user$ flex realnumber.l
Administrator-MacBook-Pro:bin user$ gcc lex.yy.c -l
Administrator-MacBook-Pro:bin user$ ./a.out
Lemegyek a boltba 2 liter tejért és 10 dgk szalámiért
Lemegyek a boltba [realnum=2 2.000000] liter tejért és [ ←
realnum=10 10.000000] dgk szalámiért
```

## 3.5. I33t.I

Lexelj össze egy I33t ciphert!

Megoldás videó: [https://youtu.be/06C\\_PqDpD\\_k](https://youtu.be/06C_PqDpD_k)

Megoldás forrása:

I33t

Tanulságok, tapasztalatok, magyarázat...

A feladat sikeres futtatásához, felhasználtam a megoldás videóban szereplő forráskódöt, és ez alapján értelmeztem a feladatot. A I33t.I feladatban a leet egy internetes változat az angol ábécében. Azt érjük el vele, hogy különböző karaktereket/számokat tudunk benne formálni. Régebben használták titkosításra, ám manapság már ennek nem sok értelme van, hiszen nagyon könnyen fel lehet törni. A lényege a programnak, hogy a bemeneti részben amikor megadjuk a karaktereket illetve a számokat, azokat átalakítja, egy

hasonló kinézetű formára. Ez a program is (mivel ez is flex), az előzőhöz hasonlóan 3 részből épül fel. Itt is megadjuk a különböző definíciókat, szabályokat, és itt is meghívjuk a lexikális elemzőt.

Létrehozunk a programunkban egy struktúrát, amiből létrehozunk egy tömböt, amiket feltöltünk elemekkel. Ezt egy C típusú karakterláncokat használunk stringként. Ezekben az elemekben megadjuk, hogy a különböző karaktereket (jelen esetben az angol abécé betűit), milyen karakterrel helyettesítse. Mivel több helyettesítési karakter is megvan adva, így ezt egy random sorsolással, fogja eldönteni, hogy mikor melyik karaktert fogja használni a programunk a helyettesítéshez. Ebben a részben adtuk meg a definicót.

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <ctype.h>  
  
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))  
  
struct cipher {  
    char c;  
    char *leet[4];  
} l337d1c7 [] = {  
  
'a', {"4", "4", "@", "/-\\"},  
'b', {"b", "8", "|3", "|{}"},  
'c', {"c", "(", "<", "{}"},  
'd', {"d", "|)", "[", "|{}"},  
'e', {"3", "3", "3", "3"},  
'f', {"f", "|=", "ph", "|#"},  
'g', {"g", "6", "[", "[+}"},  
'h', {"h", "4", "|-", "[ - ]"},  
'i', {"1", "1", "|", "!"},  
'j', {"j", "7", "_|", "_/"},  
'k', {"k", "|<", "1<", "|{"},  
'l', {"l", "1", "|", "|_"},  
'm', {"m", "44", "(V)", "\\\|"},  
'n', {"n", "|\\|", "/\\/", "/V"},  
'o', {"0", "0", "()", "[]"},  
'p', {"p", "/o", "|D", "|o"},  
'q', {"q", "9", "O_", "(, )"},  
'r', {"r", "12", "12", "|2"},  
's', {"s", "5", "$", "$"},  
't', {"t", "7", "7", "'|'"},  
'u', {"u", "|_|", "(_)", "[_]"},  
'v', {"v", "\\\|", "\\/", "\\\|"},  
'w', {"w", "VV", "\\\|\\\|", "(/\\)"},  
'x', {"x", "%", ")("), (""},  
'y', {"y", "", "", ""},  
'z', {"z", "2", "7_", ">_"},  
  
'0', {"D", "0", "D", "0"},  
'1', {"I", "I", "L", "L"},
```

```
{'2', {"Z", "Z", "Z", "e"}},  
{'3', {"E", "E", "E", "E"}},  
{'4', {"h", "h", "A", "A"}},  
{'5', {"S", "S", "S", "S"}},  
{'6', {"b", "b", "G", "G"}},  
{'7', {"T", "T", "j", "j"}},  
{'8', {"X", "X", "X", "X"}},  
{'9', {"g", "g", "j", "j"}}

// https://simple.wikipedia.org/wiki/Leet
};

%
```

A következő részben meghatározzuk a szabályokat, ha .(pont) van, ez azt jelenti ha beütünk bármilyen betűt a bemenetnél akkor végrehajtja a szabályt, kivéve ha új sort ütünk be, ez akkor következik be, ha entert ütünk. Ebben a blokkban végigmegyünk a sorokon, amit a definícióban adtunk meg, és keressük a betűt, a .c részt amit már fentebb megadtunk, ha ez teljesül, akkor belép, és egy random számot generál, és vizsgáljuk, ha ez a szám 91-nél kisebb, akkor az első esetet fogja választani. A továbbiakban is ezt vizsgáljuk, csak más értékekkel, ha 95-től kisebb akkor a 2.-at, ha 98-tól kisebb akkor a 3.-at és ha 98 100 között van akkor a 4.-et fogja helyettesíteni, ha megtalálta akkor megszakítjuk a ciklust. Ha nem találta meg a betűt, akkor visszaadja nekünk kimenként azt a karaktert amit a bemenetnél megadtunk.

```
%%  
. {  
  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
    {  
  
        if(l337d1c7[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int) (100.0*rand() / (RAND_MAX+1.0));  
  
            if(r<91)  
                printf("%s", l337d1c7[i].leet[0]);  
            else if(r<95)  
                printf("%s", l337d1c7[i].leet[1]);  
            else if(r<98)  
                printf("%s", l337d1c7[i].leet[2]);  
            else  
                printf("%s", l337d1c7[i].leet[3]);  
  
            found = 1;  
            break;  
        }  
    }  
}
```

```
if(!found)
    printf("%c", *yytext);

}
```

Az utolsó részben inicializáljuk a random számlálót, valamit elindítjuk a lexikális elemzést( ekkor fog lefutni az amit a szabályban megadtunk.).

```
%%
int
main()
{
    srand(time(NULL)+getpid());
    yylex();
    return 0;
```

Itt is hasonlóképp mint az előző feladatban, a lexikális elemző az 1 forrásból készít egy c forráskódot, amit le tudunk majd futtatni, és fordítani is. Én lefordítottam a c forráskódot és a következő eredményt kaptam:

```
Administrator-MacBook-Pro:bin user$ flex 1337d1c7.1
Administrator-MacBook-Pro:bin user$ gcc lex.yy.c -lI
Administrator-MacBook-Pro:bin user$ ./a.out
Ma elmegyek edzeni a debreceni Hungarofit edzotermebe
m4 3lm3gy3k 3dz3n1 4 d3br3c3/V1 h[_]ng4r0f1t 3dz0t3rm3b3
```

Láthatjuk, hogy a programunk sikeresen lefutott, és helyettesítette az általunk megadott karaktereket.

### 3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelo)==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 jelkezelo 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:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

```
for(i=0; i<5; ++i)
    // Akkor fog lefutni a ciklus ha előtte deklaráljuk az i-t (ezt ←
     megtehetjük a for ciklusban is).
    // Az i változó 0-ról indul lépéseként megyünk, amíg az i ←
     értéke el nem éri a 4-t hisz eddig fut majd
    // a ciklusunk, ezt meg is adtuk úgy, hogy i < 5. Itt az ++i = ←
     1+i jelent.
```

```
for(i=0; i<5; i++)
    // Nagyon hasonlít a 2-es példára, itt is ha deklaráljuk az i-t ←
     akkor le fog futni
    // a ciklus(deklarálhatjuk a for ciklusban is). A különbség, ←
     hogy itt ++i helyet i++ van, ez i++ = i + 1, tehát ugyanazt ←
     az eredményt kapjuk majd mint az előző esetben.
```

```
for(i=0; i<5; tomb[i] = i++)
    // Ahhoz, hogy lefusson a ciklus deklaráltnál kell egy tömböt, ←
     illetve az i változót. A ciklusban látunk egy olyat, hogy ←
     a lépésköznél a tomb i-edik eleme az egyenlő i + 1-el.
    // Ez nem egyezik meg a ciklus formájával. A ciklus lefut, ←
     viszont a tömb értékeivel az utasítás részben kezdhetünk ←
     valamit.
```

```
for (i = 0; i < n && (*d++ = *s++); ++i)
    // A ciklusunk akkor fog lefutni, ha deklaráltuk a szükséges ←
    // változókat, az n-t, a *d-t illetve a *s-t.
    // Ha végeztünk a deklarárással, akkor láthatjuk, hogy a for ←
    // ciklusban két feltétel van amiket a &&(ÉS) jellel ←
    // kapcsolunk össze.
    // Addig minden le fog futni a ciklus, amíg az n értéke ←
    // nagyobb mint 0, hisz i < n, és az i értéke 0.
    // Amint látjuk a ciklusban 2 feltétel van mégis le lehet ←
    // futtatni.
```

```
printf("%d %d", f(a, ++a), f(++a, a));
// Ebben a feladatban a printf függvényt láthatjuk, amit ←
// kiiratáskor használunk.
// A %d, egy int típusú decimális egész számok kifejezésére ←
// használjuk. Megtalálható még a kiiratásban egy f függvény is ←
//
// illetve egy változó is. Deklaráljuk az a változót, valamit ←
// az f függvényt is a kiiratásnak megfelelően.
// Azonban ha futtatni szeretnénk, akkor azt kapjuk, hogy az f ←
// függvény implicit kijelentése érvénytelen a C99-ben.
```

```
printf("%d %d", f(a), a);
// Az elején deklaráljuk a szükséges változót és függvényt a ←
// helyes működéshez.
// Szintén a printf függvényt használjuk kiiratásra, ahol ismét ←
// használjuk a %d, ahol a d a decimális számok kifejezésére ←
// használjuk,
// valamint az f függvényt, aminek most csak egy paramétere van ←
//
// A printf függvény ha minden jó csináltunk kiirja az ←
// értéket helyesen.
```

```
printf ("%d %d", f(&a), a);
// A 8. feladatban találkozunk ismét egy printf függvénnyel, ←
// aminek működéséhez ismét deklarálnunk kell
// az a változót(int), illetve az f függvényt(int).
// Ebben az esetben, helyesen fut le a kiiratás.
```

## 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 (\forall y (y > x) \rightarrow (y \text{ prim}))) $
```

```
$ (\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)

Tanulságok, tapasztalatok, magyarázat...

1. minden számnál van nagyobb prímszám.
2. végtelen sok ikerprím számpáros van.
3. Van olyan szám amelynél minden prím szám kisebb.
4. Van olyan szám aminél bármely nagyobb szám nem prím.

## 3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referencia
- egészek tömbje
- egészek tömbjének referencia (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;
- int \*b = &a;

- `int &r = a;`
- `int c[5];`
- `int (&tr)[5] = c;`
- `int *d[5];`
- `int *h();`
- `int *(*l)();`
- `int (*v(int c))(int a, int b)`
- `int (*(*z)(int))(int, int);`

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
int a;
//Létrehozunk egy egész típusú változót
```

```
int *b=&a;
//Az a változó egészre mutató mutató.
```

```
int &r = a;
// A már deklarált változó (a), (&r) referenciája
```

```
int c[5];
// Az egészek 5 elemű tömbje.
```

```
int (&tr)[5] = c;
// Az előző feladatban már létrehoztuk a tömböt, ez ennek a ↪
referenciája.
```

```
int *d[5];
// Ebben a feladatban a d mutató egy tömb, ami az egészekre ↪
mutató mutató.
```

```
int *h();
// Ebben a részben a h egy függvény, ami egy egészre mutató ↪
mutatót ad majd nekünk vissza.
```

```
int *(*l) ();
// Az egészre mutató mutatót ad vissza és erre a függvényre ↪
// mutató mutató, .
```

```
int (*v (int c)) (int a, int b);
// Egészet ad vissza, és kap két egészet az a-t és a b-t, ↪
// függvényre mutató mutatót, visszaad egy egészet kapó ↪
// függvényt.
```

```
int (*(*z) (int)) (int, int);
// A (*(*z)), visszaad egy egészet (int), és kap két egészet ( ↪
// int, int), függvényre mutató mutatót visszaadó, egészet ↪
// kapó függvényre.
```

## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\*\* háromszögmátrix

Megoldás videó:

Megoldás forrása:[https://gitlab.com/nbatfai/bhax/blob/master/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgCaesar/tm.c](https://gitlab.com/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProgCaesar/tm.c)

**tm**

Tanulságok, tapasztalatok, magyarázat...

A feladat értelmezéshez, a megoldás forrásban mellékelt forráskódöt használtam fel, ez segített a feladat megértésében.

A háromszögmátrix egy olyan négyzetes mátrix melynek a főátlója alatti összes elem vagy a főátlója feletti összes elemnél csupa 0-át látunk.

A main-en belül az első sorban egy int változó típussal, megadjuk, hogy a háromszögünk hány sorból álljon, tehát megadjuk a sorok számát.

```
int nr = 4;
```

A második sorban deklaráltuk a tm nevű változót, ami egy mutató, és a tárban foglalja le biteket.

```
double **tm;
```

A printf-ben kiíratjuk az első bájtnak a címét

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

Az if-es utasításban, ismeretlen a malloc, a `malloc()` függvény arra szolgál, hogy megosztja a memória bátjait, és a kijelölt memóriára mutató mutatót, ha hibát talál akkor a NULL értéket fogja visszaadni, ami nem mutat sehol(=man 3 malloc). Tehát a malloc függvényben megadjuk, hogy hány bájtot foglaljon le, jelen esetben ez 32 bájt lesz, mert a `double*` az 8 bájt, de az nr értékével ezt megszoroztuk, és ugye ahogy azt már megadtuk még az első sorban, hogy az nr értéke 4, így  $4*8 = 32$ . Majd kiíratjuk a tm-et.

```
if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
```

```
        return -1;
    }

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

A for ciklusban  $i = 0$ -tól addig megyünk, ameddig az  $i$  kisebb mint az  $nr$  (ugye ez az érték 4), ebben az esetben fog megvalósulni az ami a törzsben van, az if-es rész, amiben ismét a malloc() függvényt alkalmazzuk, ahol a malloc visszakap egy pointert és ad egy méretet, és arra kényszerítjük, hogy egy double\*-ot adjon vissza. Itt is ha hibát talál akkor a NULL értéket fogja visszaadni, ami nem mutat sehol vá, majd kiíratjuk.

```
for (int i = 0; i < nr; ++i)
{
    if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL)
    {
        return -1;
    }

}

printf("%p\n", tm[0]);
```

A 2 for ciklus azt csinálja, hogy feltölti egész számokkal amíg ez lehetséges, tehát az első esetben ez 4 a második esetben pedig  $i+1$ -ig, majd kiíratja az elemeket a terminál ablakra. Az alábbi részekben adatszerkezetre hivatkozunk, ebből nézhetünk meg néhányat(4-et).

```
for (int i = 0; i < nr; ++i)
    for (int j = 0; j < i + 1; ++j)
        tm[i][j] = i * (i + 1) / 2 + j;

for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
}

for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
}
```

A programban megtalálható még a free függvény is, ami osztja a ptr által kijelölt memória elosztását. Ha a ptr, egy NULL mutató, akkor nem történik művelet( man 3 free). Ebben a függvényben felszabadítjuk a mutatott területet, illetve a foglalt memóriát is felszabadítjuk.

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

```
    free (tm[i]);  
  
    free (tm);
```

A fordítást végrehajtjuk a terminálba, és megkapjuk a 4 soros háromszögmátrixunkat, ami a következőképp néz ki:

```
./haromszog  
0x7ffeeeade8b80  
0x7fdf57500640  
0x7fdf57500660  
0.000000,  
1.000000, 2.000000,  
3.000000, 4.000000, 5.000000,  
6.000000, 7.000000, 8.000000, 9.000000,  
0.000000,  
1.000000, 2.000000,  
3.000000, 4.000000, 5.000000,  
6.000000, 7.000000, 8.000000, 9.000000,
```

A program forráskódja:

```
#include <stdio.h>  
#include <stdlib.h>  
  
int  
main ()  
{  
    int nr = 4;  
    double **tm;  
  
    printf("%p\n", &tm);  
  
    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;  
        }  
  
    }  
}
```

```
printf("%p\n", tm[0]);\n\n    for (int i = 0; i < nr; ++i)\n        for (int j = 0; j < i + 1; ++j)\n            tm[i][j] = i * (i + 1) / 2 + j;\n\n    for (int i = 0; i < nr; ++i)\n    {\n        for (int j = 0; j < i + 1; ++j)\n            printf ("%f, ", tm[i][j]);\n        printf ("\n");\n    }\n\n    for (int i = 0; i < nr; ++i)\n    {\n        for (int j = 0; j < i + 1; ++j)\n            printf ("%f, ", tm[i][j]);\n        printf ("\n");\n    }\n\n    for (int i = 0; i < nr; ++i)\n        free (tm[i]);\n\n    free (tm);\n\n    return 0;\n}
```

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

e

Tanulságok, tapasztalatok, magyarázat...

A program értelmezéséhez az udprog repositoryban talált e.c forráskódöt fogom felhasználni.

Az EXOR(XOR), egy egyszerű titkosítási eljárás. A szövegünköt egy kulcs segítségével tudjuk titkosítani, és ezzel a művelettel vissza is tudjuk fejteni, ha ismerjük a kulcsot.

Ennek az eljárásnak a lényege, hogy a titkosítani kívánt szöveg mellé megadunk egy kulcsot ami lehetőleg egy tetszőleges szöveg, melynek segítségével elvégezzük a titkosítást. Itt a két szöveget felhasználva, amiket párosával nézünk, és végrehajtjuk a bitenkénti kizáró vagy (xor) műveletet.

Ez azért jó, mert együk fel van egy szöveged, amit csak akkor tudsz elolvasni, ha tudod hozzá a megfelelő kulcsot. Amíg nem tudod hozzá a kulcsot, addig ehhez a szöveghez nem tudsz hozzáférni, ez a program lényege.

A programunk elején importálnunk kell a szükséges könyvtárakat. Az stdio.h könyvtár szerintem fölösleges, viszont a másik 2 függvénykönyvtár igazán fontos ahhoz, hogy a programunk megfelelően működjön.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

A unistd.h függvénykönyvtár a `read` és a `write` miatt kell, míg a string.h függvénykönyvtár a string függvények miatt fontos.

A `#define` részben megadjuk a kulcs max méretét, ami 100, illetve a buffer azaz a tároló méretét ami 256.

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

A main metódusban megadunk paramétereket, az argc-t illetve a `**argv`-t. Az `argc` az argumentumok számát, míg a `argv` az argumentumokat tárolja.

```
int
main (int argc, char **argv)
```

A `char` részben, ugye létrehozzuk a karakter típusú tömbjeinket kulcs ennek a mérete `MAX_KULCS`, és buffer, ennek a mérete `BUFFER_MERET`.

```
char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];
```

A következő részben deklarálunk két egész típusú változót melyeknek értéke 0. Az első a `kulcs_index`et jelöli, a második pedig a beolvasott bajtok számát.

```
int kulcs_index = 0;
int olvasott_bajtok = 0;
```

A harmadik deklarált int-nél megadjuk, a kulcs méretét, ehhez szükségesek a string.h függvénykönyvtárban szereplő függvények a `strlen`, a `strncpy`, és odaadjuk a `MAX_KULCS`-ot is hiszen addig foglalkozunk a jelszóval (addig a méretig), amit a `MAX_KULCS`-nak már megadtunk.

```
int kulcs_meret = strlen (argv[1]);
strncpy (kulcs, argv[1], MAX_KULCS);
```

A while ciklusunk addig fog lefutni, amíg van mit kiolvasnia. Itt a `read` által visszaadott értéket vizsgáljuk, akkor fog lefutni a ciklus, ha ez a visszaadott érték pozitív lesz.

```
while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET) > 0))
```

Ha ez a szám pozitív, akkor lefut a for ciklus, ami végigmegy az olvasott bajtokon( amit beolvasott) Amíg ez végigmegy egy exor műveletet hajt végre. Majd a `kulcs_index`-et növeljük 1-el és elosztjuk a `kulcs_meret`-el (maradékos osztással). Végigment a for ciklus az elemeken, ekkor kilép

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

        buffer[i] = buffer[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;

    }
```

Ekkor a `write` függvényel kiíratjuk a buffer tartalmát, aminek a mérete az `olvasott_bajtok` lesz. A program futtatása után a kimenet már titkosított lesz vagy fel lesz oldva a szöveg.

```
    write (1, buffer, olvasott_bajtok);
```

A program forráskódja:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#define MAX_KULCS 100
#define BUFFER_MERET 256

int
main (int argc, char **argv)
{

    char kulcs[MAX_KULCS];
    char buffer[BUFFER_MERET];

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

    int kulcs_meret = strlen (argv[1]);
    strncpy (kulcs, argv[1], MAX_KULCS);

    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);
```

```
    }  
}
```

### 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html> 1.12. példa `exor.java`

Tanulságok, tapasztalatok, magyarázat...

Észrevehető, hogy a forráskód nagy részben hasonlít a már elemzett C program forráskódjához.

```
public class ExorTitkosító {  
    public ExorTitkosító(String kulcsSzöveg,  
        java.io.InputStream bejövőCsatorna,  
        java.io.OutputStream kimenőCsatorna)
```

Az program megírása a public class osztályban történik, itt észrevesszük, hogy nincsenek függvénykönyvtárak.

Az ExorTitkosító, három paramétert kap: A kulcsSzöveg-et, bejövőCsatorna-t, kimentőCsatorna-t.

```
throws java.io.IOException {  
  
    byte [] kulcs = kulcsSzöveg.getBytes();  
    byte [] buffer = new byte[256];  
    int kulcsIndex = 0;  
    int olvasottBájtok = 0;
```

Az első sor azért fontos a programunkban, mert ő fog jelezni, ha valamilyen hibát észlel és ezt jelzi majd nekünk a kimeneten. A kulcsot egy byte nevű tömbben tároljuk el, ahol a karaktereket a kulcsSzövegből tudjuk meg és ezt a getBytes() függvénytel bontjuk.

```
byte [] buffer = new byte[256];
```

Ebben a byte tömbben találhatjuk továbbá a buffert is, melynek megadjuk a méretét is, jelen esetben ez az érték 256.

```
int kulcsIndex = 0;  
int olvasottBájtok = 0;
```

Szükségünk lesz 2 db Integer változóra is melyeket létre is hozunk, az egyik lesz a kulcsIndex, a másik pedig a beolvasott bájtok száma ( olvasottBájtok ).

```
while ((olvasottBájtok =
        bejövőCsatorna.read(buffer)) != -1) {
    for (int i=0; i<olvasottBájtok; ++i) {
        buffer[i] = (byte) (buffer[i] ^ kulcs[kulcsIndex]);
        kulcsIndex = (kulcsIndex+1) % kulcs.length;
    }
    kimenőCsatorna.write(buffer, 0, olvasottBájtok);
}
```

Két ciklus használunk a forráskódunkban a while ciklust és a for ciklust. A while ciklus addig fog futni, amíg tud mit kiolvasni a read függvényel.

```
for (int i=0; i<olvasottBájtok; ++i) {
    buffer[i] = (byte) (buffer[i] ^ kulcs[kulcsIndex]);
    kulcsIndex = (kulcsIndex+1) % kulcs.length;
}
kimenőCsatorna.write(buffer, 0, olvasottBájtok);
}
```

A for ciklusban azt vizsgáljuk, hogy a buffer i-edik eleme és annak értéke lesz e a kulcs tömb kulcs index-edik Exor művelet eredménye. Így lesz a szövegünk olvashatatlan. Ennek az eredménye byte típus. Majd növeljük a kulcs index értékét.

```
kimenőCsatorna.write(buffer, 0, olvasottBájtok);
}
```

A class végén pedig kiíratjuk az elemeket, a write függvény segítségével a kimenetre.

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

A main-ben láthatunk egy try catch-et, ahol a try részben példányosítás látható.

```
try {  
  
    new ExorTitkosító(args[0], System.in, System.out);  
  
}
```

A try-on belül található a new ExorTitkosító, tehát készítünk egy új ExorTitkosítót ( new ), ami az argumentumként kapott szöveget alakítja át.

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

A catch részben meghívjuk a printStackTrace() függvényt, ami azért jó, mert segít nekünk abban, ha valamilyen problémáttal talál, és meg is mondja nekünk, hogy az a bizonyos probélma hol is található.

## 4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

[t.c](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat értelmezéséhez az udprog repóban található t.c forráskódöt fogom felhasználni.

A program célja az, hogy a már titkosított szövegeket feltörje. A titkosítás feladattal a 4.2-es rész foglalkozik.

A program elején definiálunk: Elsőként megadjuk a titkos szöveg max hosszát, az olvass bufferrel, a buffer méretét ezzel olvasunk be, megadjuk továbbá amit ki kell találni kulcs méretét, valamint definiálva van egy \_GNU\_SOURCE változó, aminek nincsen értéke.

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256  
#define KULCS_MERET 8  
#define _GNU_SOURCE
```

Ezt követően beimportáljuk a program helyes működéséhez nélkülözhetetlen függvénykönyvtárakat. Ezek a következők: stdio.h, unistd.h, és string.h

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

Még a main() metódus előtt azonban kell nekünk 4 függvény, amikkel dolgozni fogunk. atlago\_szohossz () : Paraméterenként vár egy char\* típusú változót, és ennek a méretét egy egész típusú változóban(titkos\_meret). Visszaad egy nekünk egy valós számot.

A függvényen belül létrehozunk egy változót ami egész típusú a neve sz, melynek kezdőértéke 0. A for ciklusban végigmegyünk a stringen, és az utasításban azt mondjuk, ha találunk egy szöközt, akkor növeljük az sz változót egyel. Majd visszaadjuk a számok méretét melyet elosztunk a szóközök számával, ezáltal megkapjuk a szavak hosszát.

```
double
atlago_szohossz (const char *titkos, int titkos_meret)
{
    int sz = 0;
    for (int i = 0; i < titkos_meret; ++i)
        if (titkos[i] == ' ')
            ++sz;

    return (double) titkos_meret / sz;
}
```

A tiszta\_lehet () függvénynek a paramétere ugyanaz mint az atlago\_hossz függvénynek, a tiszta\_lehet függvény egy egész számot ad vissza.

Létrehozunk egy szohossz változót amely, eltárolja a már előbb kiszámolt szavak hosszát . Majd visszaadjuk a logikai művelet értékét. Az ÉS művelettes kötjük össze, és az figyeljük, hogy a kapott szóhossz egy intervallum közé esik e, (ez a programban 6 és 9 közé), illetve a strcasestr segítségével nézzük meg azt, hogy a gyakori szavak előfordulnak-e a programban ezek a (hogy, nem, az, ha) szavak. Ha a szóhossz jó hosszúságú, és a szövegben megtalálhatóak az előbb említett szavak, akkor a művelet igaz, és visszaad egy pozitív értéket.

```
int
tiszta_lehet (const char *titkos, int titkos_meret)
{
    // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    / potenciális töréseket

    double szohossz = atlago_szohossz (titkos, titkos_meret);

    return szohossz > 6.0 && szohossz < 9.0
        && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
        && strcasestr (titkos, "az") && strcasestr (titkos, "ha");

}
```

A harmadik függvény a `exor()` paraméterei: kulcs, kulcs mérete, egy szöveg, és annak mérete. Visszatérési értéke nincs.

Már az előző feladatban használt `exor` műveletet elvégzi a kulccsal, az adott szövegen. Létrehozunk egy változót `kulcs_index` néven ebben tároljuk az aktuális indexét a kulcsnak. A for ciklusban végigmegyünk a szövegen, és lefut az `exor` művelet. A kulcs indexét növeljük egyel, majd maradékosan elosztjuk a kulcs méretével.

```
void
exor (const char kulcs[], int kulcs_meret, char titkos[], int ←
      titkos_meret)
{
    int kulcs_index = 0;

    for (int i = 0; i < titkos_meret; ++i)
    {

        titkos[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}
```

A negyedik, egyben utolsó függvény a `exor_tores()` paramétere megegyezik az `exor` függvényével, visszatérési értéke egész szám. Lényege, hogy a már előbb megírt 3 függvényt 1-nek fogja használni. Hivatkozik az `exor` függvényre, majd erre a szövegre meghívjuk a `tiszta_lehet` függvényt. Az `exor_tores` arra jó, hogy előtöntse egy kulcsról, hogy feltör-e egy titkosított szöveget vagy nem.

```
int
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{
    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);
}
```

Miután megadtuk a 4 függvényt rátérhetünk a főprogramra, melyben létrehozunk 2 tömböt kulcs-ot melynek mérete a `KULCS_MERET`, illetve a titkos-at, melynek mérete a `MAX_TITKOS`.

```
char kulcs[KULCS_MERET];
char titkos[MAX_TITKOS];
```

Létrehozunk még egy mutatót ami `char` típusú, ez a titkosra fog mutatni, illetve deklarálunk egy egész típusú változót(`int`) melynek neve: `olvasott_bajtok`.

```
char *p = titkos;
int olvasott_bajtok;
```

A while ciklusban ami addig fog lefutni amíg van mit olvasni a bemenetről.

A ? operátor arra szolgál, hogy ha a ? előtti kifejezés igaz, akkor visszadja azt az értéket ami a ? után van, egyébként pedig azt adja vissza ami a : után van. Ezzel nézzük meg, hogy hány bajtot olvasunk be. A ciklusban a p mutatót annyival toljuk arrébb, ahány bajtot olvasunk.

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

A for ciklusban azt vizsgáljuk ha a p pointer nem érte el buffernek a végét akkor a maradék helyet nullázzuk a titkos bufferben.

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

A következő részben, ahol az egymásba ágyazott for ciklusok vannak, ott előállítjuk az összes lehetséges kulcsot ez jelen esetben 8. Az aktuális jelszót a kules tömbben tároljuk el.

Végezetül pedig meghívjuk az exor függvényt, a kulccsal ami itt található, a titkos szövegre, így visszakapjuk az eredeti titkos szöveget, így újrakezdődik a ciklus ami azt jelenti, hogy próbálkozhatunk új jelszóval. Ha végeztek a ciklusok akkor a program kilép a 0 visszatérési értékkel.

```
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;

    if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
        printf
("Kulcs: [%c%c%c%c%c%c%c] \nTiszta szoveg: [%s] \n",
ii, ji, ki, li, mi, ni, oi, pi, titkos);
```

```
// ujra EXOR-ozunk, így nem kell egy második buffer  
exor (kulcs, KULCS_MERET, titkos, p - titkos);  
}  
  
return 0;
```

### A program forráskódja:

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256  
#define KULCS_MERET 8  
#define _GNU_SOURCE  
  
#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
  
double  
atlagos_szohossz (const char *titkos, int titkos_meret)  
{  
    int sz = 0;  
    for (int i = 0; i < titkos_meret; ++i)  
        if (titkos[i] == ' ')  
            ++sz;  
  
    return (double) titkos_meret / sz;  
}  
  
int  
tiszta_lehet (const char *titkos, int titkos_meret)  
{  
    // a tiszta szöveg valszeg tartalmazza a gyakori magyar szavakat  
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a  
    // potenciális töréseket  
  
    double szohossz = atlagos_szohossz (titkos, titkos_meret);  
  
    return szohossz > 6.0 && szohossz < 9.0  
        && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")  
        && strcasestr (titkos, "az") && strcasestr (titkos, "ha");  
}  
  
void  
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)  
{  
    int kulcs_index = 0;
```

```
for (int i = 0; i < titkos_meret; ++i)
{
    titkos[i] = titkos[i] ^ kulcs[kulcs_index];
    kulcs_index = (kulcs_index + 1) % kulcs_meret;
}

int
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{
    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);
}

int
main (void)
{
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;

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

    // maradek hely nullazasa a titkos bufferben
    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\0';

    // osszes kulcs eloallitasa
    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;  
  
    if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))  
        printf  
        ("Kulcs: [%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",  
         ii, ji, ki, li, mi, ni, oi, pi, titkos);  
  
    // ujra EXOR-ozunk, igy nem kell egy masodik buffer  
    exor (kulcs, KULCS_MERET, titkos, p - titkos);  
}  
  
return 0;  
}
```

## 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\\_nn.c](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R_nn.c)

Tanulságok, tapasztalatok, magyarázat...

Az előbbiekben már jól ismert R nyelvet kell használnunk ebben a részben is, az R nyelvnek a matematikai tulajdonságait kamatoztatjuk. Ahhoz, hogy ezt a forráskódot futtatni tudjuk megfelelően, elengedhetetlen egy további csomag telepítése, az Rstudio.

```
library(neuralnet)
```

A célunk az lenne, hogy egy olyan hálót hozzunk létre, amely képes logikai műveletekre. Ahhoz, hogy ez megvalósuljon szükség van a neuralnet függvényre. Ez a függvénykönyvtár nagyon fontos, hogy meg tudjuk oldani a problémát.

```
a1      <- c(0,1,0,1)  
a2      <- c(0,0,1,1)  
OR      <- c(0,1,1,1)
```

Az alábbiakban létrehozzuk az a1, a2, OR változókat. Az OR változó azért különleges, mert a VAGY művelet eredményt adja vissza.

```
or.data <- data.frame(a1, a2, OR)
```

Továbbá, láthatjuk a `data.frame` függvényt, ami tartalmazza a változókat, így a háló megtanulja ezekből az adatokból a helyes végrehajtást.

```
compute(nn.orand, orand.data[,1:2])
```

A `compute` parancs számunkra egy ellenőrzés. Ez a parancs segít nekünk abban, hogy az eredményünk helyes-e vagy nem, tehát kikalkulálja az eredményt számunkra.

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR      <- c(0,1,1,1)
AND     <- c(0,0,0,1)
```

```
compute(nn.orand, orand.data[,1:2])
```

A futtatás után láthatjuk, hogy az eredményünk helyes. Megkapja ugyanúgy a változókat ahogy az előzőekben, csak itt már bővítiük egy ÉS ( AND ) művelettel, ez is hasonlóan működik mint az OR.

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR   <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=0, linear.output ←
                      =FALSE, stepmax = 1e+07, threshold = 0.000001)

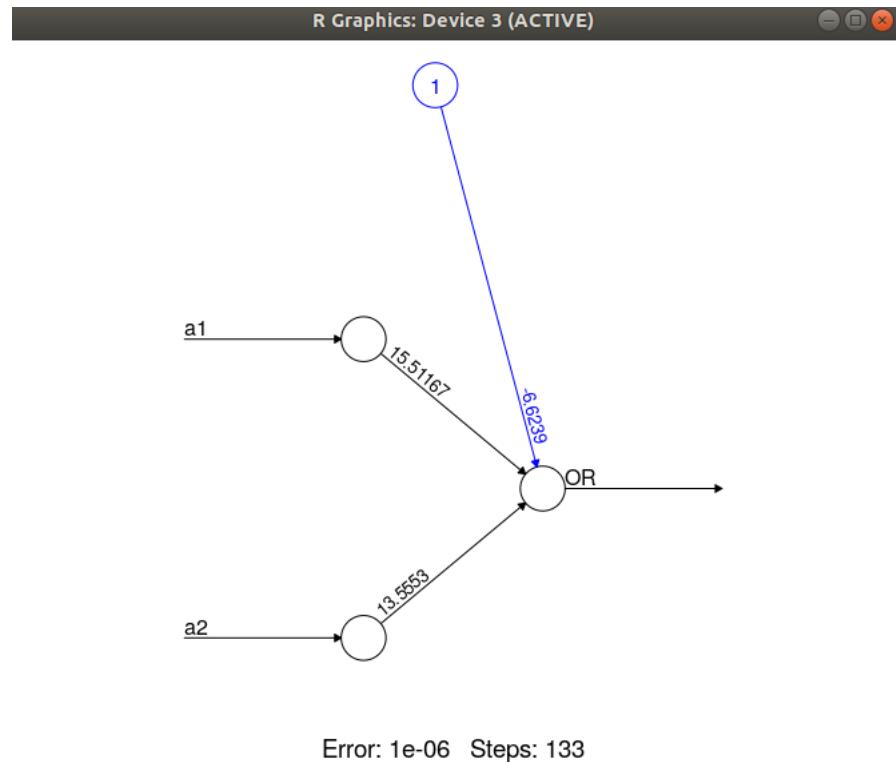
plot(nn.exor)

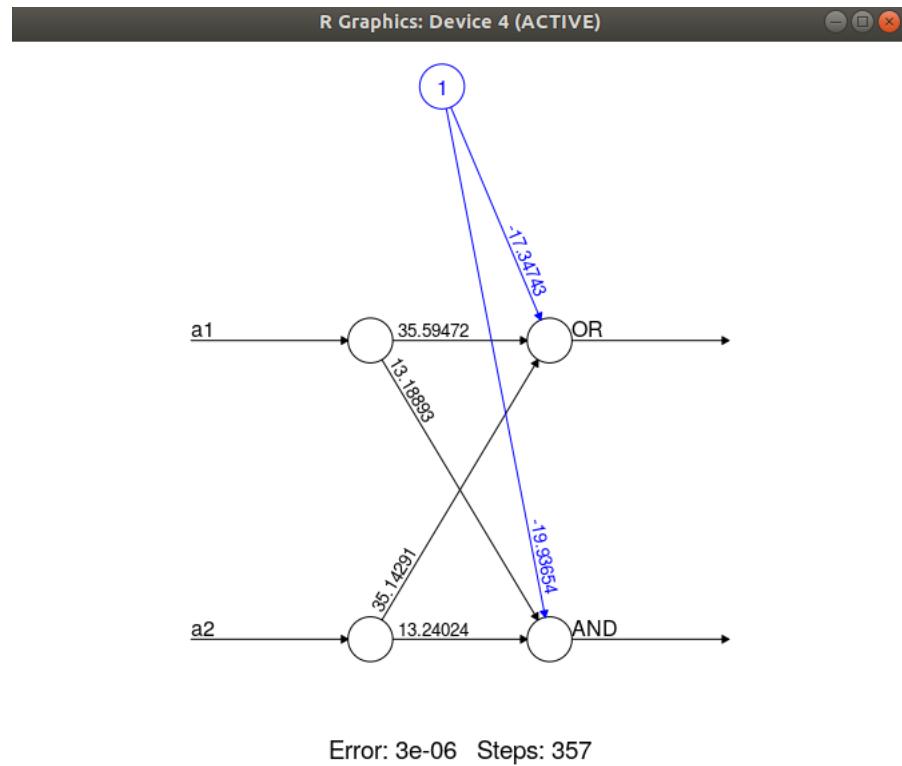
compute(nn.exor, exor.data[,1:2])
```

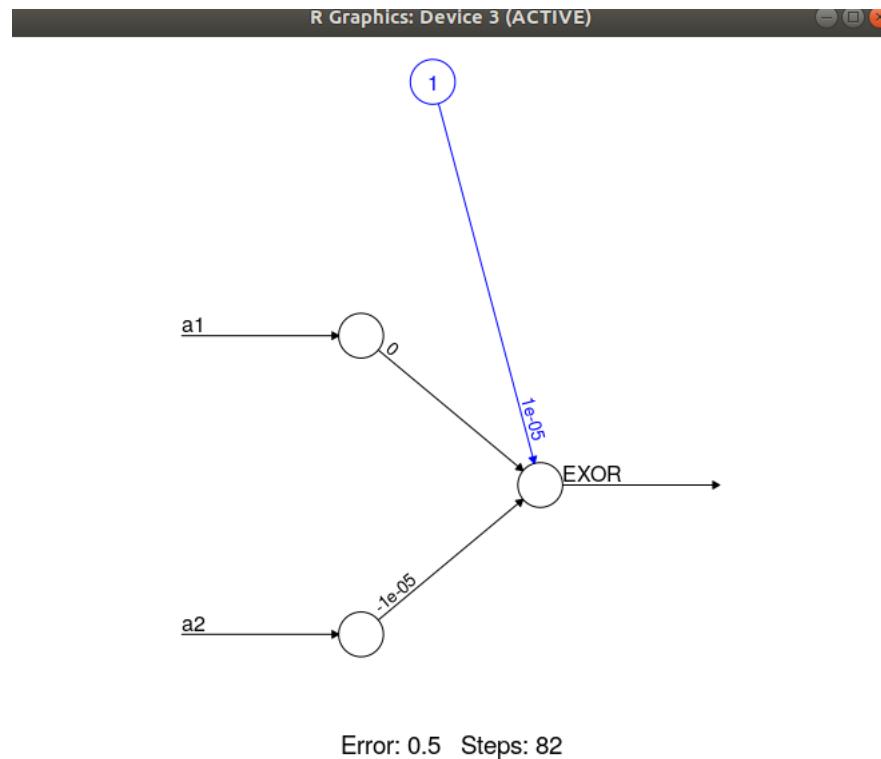
Amint láthatjuk itt is megkapja a változókat, csak itt most EXOR-al van kiegészítve nem OR-al vagy AND-el, hasonlóan működik mint az OR és az AND.

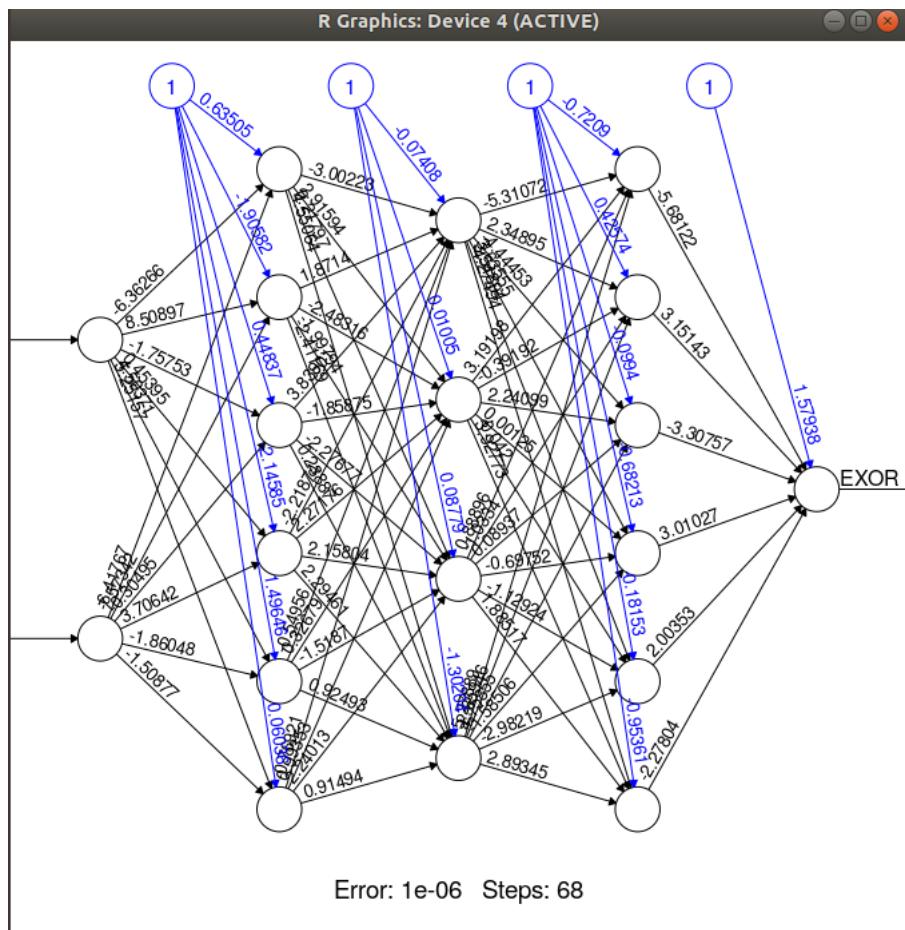
Azonban figyelnünk kell, hogy az EXOR művelet csak úgy adható meg, ha többrétegű neutronok vannak, csak így jöhet létre a „tanítás”.

A kapott eredmények képekben:









## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: <https://www.youtube.com/watch?v=XpBnR31BRJY>

Megoldás forrása:

[main.cpp](#)

[mlp.hpp](#)

[mandelpng.cpp](#)

Tanulságok, tapasztalatok, magyarázat...

A perceton a neuron mesterséges intelligenciában leggyakrabban használt változata, ami egy gép.

A gép feladata, hogy véges sok kísérletből megtanulja osztályozni a nullákból és egyesekből álló bemeneti mintázatokat. A bemeneteknek a súlyozott összegzését végzi.

Az első modell a Rosenblatt-tól eredt, Ő nevezte el a modellt Perceptronnak. A Perceptron 3 fő elemből áll: Az első elem a retina, ami a bemeneti jeleket fogadó cellákat tartalmazza, amelyek igen/nem válaszokat adnak.

A második elem azt asszociatív celláknak nevezték, amik csatlakoztak az első réteghez (retinacellákhoz), más asszociatív cellákhoz, és a harmadik réteghez, a döntési cellákhoz is. A cellák összegzik a hozzájuk érkező jeleket, impulzusokat.

A harmadik elem a döntési cellák, ami a perceptronoknak a kimenete. A bemenetek származhatnak egy, de akár több döntési celláktól, vagy akár az asszociatív celláktól.

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

A forráskódban látjuk, hogy 3 függvénykönnyvtárra lesz szükségünk, az iostream, az mlp és végül a png++/png könyvtárat.

Az utolsó 2 könyvtár lehet új számunkra, ez a két könyvtár azért fontos, mert enélkül a perceptron nem fog lefutni. A PNG-t ismerhetjük, ez a program egy PNG állományt próbál feldolgozni, ezért is fontos importálnunk őt.

```
int main(int argc, char **argv)
{
    png::image<png::rgb_pixel> png_image(argv[1]);

    int size = png_image.get_width() * png_image.get_height();
```

A main részben beolvasásra kerül a kép állomány ebben az argumentum lesz segítségünkre. A for ciklusban megadjuk a kép méretét, a get\_width és a get\_height-el.

```
for(int i=0; i<png_image.get_width(); ++i)
    for(int j=0; j<png_image.get_height(); ++j)
        image[i*png_image.get_width() + j] = png_image[i][j].red;
```

A forráskódunkban találkozunk a main-en belül még 2 egymásba ágyazott for ciklussal.

```
for(int i=0; i<png_image.get_width(); ++i)
```

Az egyik for ciklus az végigmegy a kép szélességén.

```
for(int j=0; j<png_image.get_height(); ++j)
```

A másik for ciklus pedig, végig fog menni a kép magasságán.

Amikor már lefutott a 2 for ciklusunk, nos ekkor az image fogja nekünk eltárolni a képünkben a vörös képpontokat.

```
double value = (*p)(image);  
  
cout<<value;  
  
delete p;  
delete [] image;
```

A value -val hívjuk meg az imaget. A value láthatjuk, hogy egy double típusú szám.

A program lényege miután kiíratjuk az outputon, akkor a már nem használt részeket törülni tudjuk, mivel az a cél hogy ami számunkra elérhető memória azt felszabadítsuk, így amit lefoglaltunk memóriát, azt újra tudjuk használni.

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: [https://progpater.blog.hu/2011/03/26/kepes\\_egypercesek  
mandelpng.c++](https://progpater.blog.hu/2011/03/26/kepes_egypercesek_mandelpng.c++)

A Mandelbrot , egy bizonyos Benoît Mandelbrot emertől származik, aki lengyel származású volt. Benoît főként a matematikában aratott maradandót, hozzá fűzhető a Mandelbrot-halmaz is.

A Mandelbrot-halmazt komplex számsíkon ábrázoljuk, ennek a jelentősége az, hogy nem tart a végtelenbe a sorozatunk, hanem a sorozat korlátos. Tehát ez a Mandelbrot-halmaz komplex számokkal dolgozik valamint még egy egyenlettel is. Ezt így jelöljük:  $x_1 := c$  és  $x_{n+1} := x_n * x_n + c$ .

Lássuk a forráskódot:

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

Először is be kell importálunk 2 függvénykönyvtárat, az iostreamet, és a png++/png-t. Mind 2 már ismerős az előző feladatokból. Emlkéztetésként a PNG könyvtár teszi lehetővé számunkra, hogy PNG állományokkal dolgozzunk a programunkban.

```
int main (int argc, char *argv[])
{
    if (argc != 2) {
        std::cout << "Használat: ./mandel fajlnev";
        return -1;
    }
}
```

Nézzük a main részét: Láthatjuk, hogy parancssori argumentummal dolgozunk, ezzel is adjuk meg azt, hogy miképpen mentjük majd el a képünket.

```
// számítás adatai
double a = -2.0, b = .7, c = -1.35, d = 1.35;
int szelesseg = 600, magassag = 600, iteraciosHatar = 1000;
```

```
// png-t készítünk a png++ csomaggal
png::image <png::rgb_pixel> kep (szelesseg, magassag);

// a számítás
double dx = (b-a)/szelesseg;
double dy = (d-c)/magassag;
double reC, imC, rez, imZ, ujrez, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
std::cout << "Szamitas";
// Végigzongorázzuk a szélesség x magasság rácsot:
for (int j=0; j<magassag; ++j) {
    //sor = j;
    for (int k=0; k<szelesseg; ++k) {
        // c = (reC, imC) a rács csomópontjainak
        // megfelelő komplex szám
        reC = a+k*dx;
        imC = d-j*dy;
        // z_0 = 0 = (rez, imZ)
        rez = 0;
        imZ = 0;
        iteracio = 0;
    }
}
```

Tovább haladva látjuk, hogy megadtuk a számítás adatait, létrehoztuk a szükséges változókat.

Ezt követően PNG-t készítünk, ezért volt szükséges még az elején beimportálni a png++/png függvénykönyvtárat.

```
// a számítás
double dx = (b-a)/szelesseg;
double dy = (d-c)/magassag;
double reC, imC, rez, imZ, ujrez, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
std::cout << "Szamitas";
```

Ezt követi a számítás, amihez a dx dy változókat használjuk. A komplex számok jelölésére bevezetjük a Z és C változókat is.

```
// Hány iterációt csináltunk?
int iteracio = 0;
std::cout << "Szamitas";
```

Iteráció számára is létrehozunk egy int típusú változót, melynek a 0 értéket adjuk.

```
// Végigzongorázzuk a szélesség x magasság rácsot:
```

```
for (int j=0; j<magassag; ++j) {
    //sor = j;
    for (int k=0; k<szelesség; ++k) {
        // c = (reC, imC) a rács csomópontjainak
        // megfelelő komplex szám
        reC = a+k*dx;
        imC = d-j*dy;
        // z_0 = 0 = (reZ, imZ)
        reZ = 0;
        imZ = 0;
        iteracio = 0;
    }
}
```

Az iterációt követően végigmegyünk a szélességen illetve a magasságon is, ehhez remek segítsűgünk a 2 for ciklus, amivel ezt meg is tesszük, majd vizsgáljuk C képzetes és valós változót, ugyanezt tesszük Z esetében is.

```
iteracio = 0;
}
```

Ezt követően láthatjuk az iteráció változó is megjelenik.

```
while (reZ*reZ + imZ*imZ < 4 && iteracio < iteracionsHatar) {
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ*reZ - imZ*imZ + reC;
    ujimZ = 2*reZ*imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;
```

```
++iteracio;
```

Ezt követően az iteráció számát növeljük.

```
kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,
                                    255-iteracio%256, 255- ←
                                    iteracio%256));
```

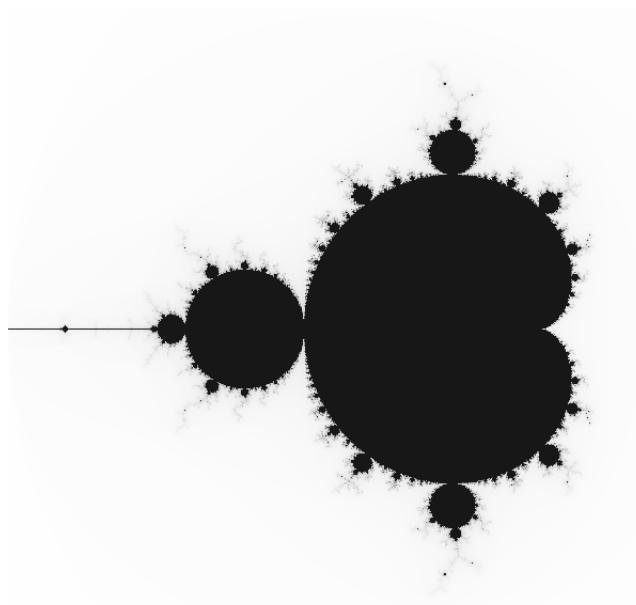
Tovább haladva barátkozhatunk a set\_pixel függvényel, ahol elérésünk lehet a koordinátán található helyekre, de az rgb\_pixel függvényel ezt meg tudjuk változtatni, és ezt meg is tesszük.

```
kep.write (argv[1]);
std::cout << argv[1] << " mentve" << std::endl;
```

A kep fajlba fogja kiírni a Mandelbrot halmazunkat a program.

Fordítunk, és futtatjuk a programunkat, majd ezt kapjuk:

```
Administrator-MacBook-Pro:~ user$ g++ /Users/user/Documents/Prog1-\ ↵
Programozás/Gyakorlás/5.Mandelbrot/5.1/mandelbroot.cpp -o mandel
Administrator-MacBook-Pro:~ user$ ./mandel mandel.png
```



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

Megoldás videó:

[3.1.2.cpp](#)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/Mandelbrot/3.1.2.cpp](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Mandelbrot/3.1.2.cpp)

Tutorált: [Fenyvesi Mátyás](#)

A program annyiban különbözik az előzőtől, hogy most a számoknak nem kell vizsgálni a képzetes és valós részét, hanem egyben megoldjuk.

A feladatunk lényegében ugyan az mint az előbb megoldott feladatban, tehát szintén egy Mandelbrot-halmazt kell készítenünk, azonban most ebben a forráskódban használunk egy complex osztályt.

Nézzük is a forráskódot:

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

A forráskód szerint beimportálunk 3 függvénykönyvtárat, az első 2 az az előző feladatból már ismerős, viszont a harmadik az a komplex függvény, ami a komplex számokban segít, ez megkönnyíti majd jelentősen a dolgunkat.

Lássuk a main részt.

```
int main ( int argc, char *argv[] )  
{  
  
    int szelesseg = 1920;  
    int magassag = 1080;  
    int iteraciosHatar = 255;
```

Megadjuk a leendő képünk magasságát, valamint szélességét is. Előbbinek az értéke 1080, utóbbinak pedig 1920. Szükségünk lesz még egy iterációshatár változóra is melynek értéke 255. Őket integer típussal adtuk meg.

```
double a = -1.9;  
double b = 0.7;  
double c = -1.3;  
double d = 1.3;
```

Double típussal pedig megadjuk a határértékeket, lényegében az, hogy hol fogunk dolgozni.

```
if ( argc == 9 )  
{  
    szelesseg = atoi ( argv[2] );  
    magassag = atoi ( argv[3] );  
    iteraciosHatar = atoi ( argv[4] );  
    a = atof ( argv[5] );  
    b = atof ( argv[6] );  
    c = atof ( argv[7] );  
    d = atof ( argv[8] );  
}
```

If-nél vizsgáljuk, hogy az argumentumok száma egyenlő e 9-el, ha ez teljesül akkor a már megadott szélességnek és magasságnak és így tovább egészen az utolsó double változóig ( amit még definiáltunk ), adunk nekik parancssori argumentumot.

Ehhez a művelethez 2 függvény szükséges az atoi( String -> Integer ), illetve az atof( String -> Double ).

```
    else
    {
        std::cout << "Hasznalat: ./mandelbrot_komplex fajlnev szelesseg ←
                     magassag n a b c d" << std::endl;

        return -1;
    }
```

Az else ágban azt vizsgáljuk, ha az eredményük nem 9, akkor, akkor csak kiíratjuk a “képünket”.

Azt vehetjük észre, hogy ezután a program -1-el tér vissza.

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

Az image függvényel létrehozzuk a képet, melynek mérete a szélesség\*magasság.

```
double dx = ( b - a ) / szelesseg;
double dy = ( d - c ) / magassag;
double reC, imC, reZ, imZ;
```

Megadjuk a dx( szélesség ) dy( magasság ) változóknak az értékeit, majd deklaráljuk a következőket: reC, imC, reZ, imZ.

```
int iteracio = 0;
```

Ha ezeket elvégztük akkor az iteráció változónak a 0 értéket adjuk meg.

```
std::cout << "Szamitas\n";

// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon

    for ( int k = 0; k < szelesseg; ++k )
    {
```

Ezután indítunk 2 for ciklust az első for ciklus a magasságon, míg a második for ciklus a szélességen megy végig.

```
// c = (reC, imC) a halo racspontjainak
```

```
// megfelelo komplex szam

reC = a + k * dx;
imC = d - j * dy;
std::complex<double> c ( reC, imC );

std::complex<double> z_n ( 0, 0 );
iteracio = 0;
```

A folytatásban, a reC-nek illetve az imC-nek beállítjuk a látható értéket, majd a complex segítségével megkapjuk a c változót ami már tartalmazza a szám képzetes és valós részét is. Az iterációnak ismét odaadjuk a 0 értéket.

```
while ( std::abs ( z_n ) < 4 && iteracio < iteracionsHatar )
{
    z_n = z_n * z_n + c;

    ++iteracio;
}
```

Ezt követően a while ciklusban ismét vizsgáljuk azt, amit már az előző feladatban, tehát kisebb e mint 4 a z\_n abszolultértéke, illetve az iteráció kisebb-e mint az iterációshatár. Ennek megfelőlen ismét változtatok a z\_n értékén, majd ezt követően az iterációt növeljük egyel.

```
kep.set_pixel ( k, j,
                 png::rgb_pixel ( iteracio%255, (iteracio*iteracio <%
) %255, 0 ) );
}
```

A set\_pixel függvény amilyen paramétert adtunk neki, olyan RGB színnel fogja elkészíteni a képünket.

```
int szazalek = ( double ) j / ( double ) magassag * 100.0;
    std::cout << "\r" << szazalek << "%" << std::flush;
}
    kep.write ( argv[1] );
    std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

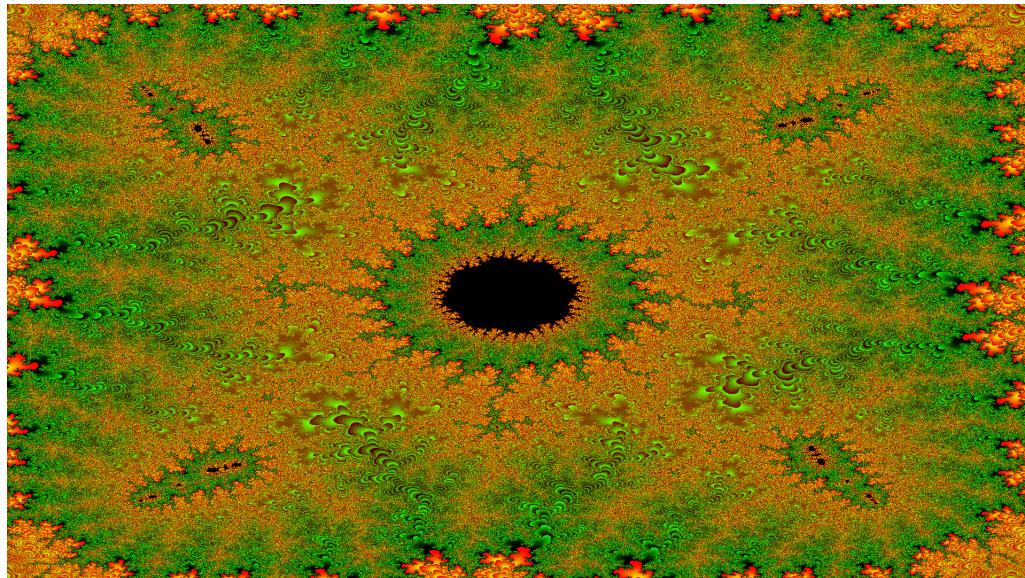
A szazalek változó ami integer típusú, arra szolgál, hogy megmondja hogyan is halad a képünk kirajzolata a képrenyőn.

A write függvényel kiíratjuk az elkészült képünket, majd az ezt követő sorban dokumentáljuk azt, és kiíratjuk, hogy a képünk mentve lett.

Most már mindenkel kész vagyunk fordíthatjuk, és futtathatjuk a programunkat.

```
Administrator-MacBook-Pro:~ user$ g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
```

```
Administrator-MacBook-Pro:~ user$ ./3.1.2 masodikmandel.png 1920 1080 ←
2040 -0.01947381057309366392260585598705802112818 ←
-0.0194738105725413418456426484226540196687 ←
0.7985057569338268601555341774655971676111 ←
0.798505756934379196110285192844457924366
```



### 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)

#### 3.1.3.cpp

Tanulságok, tapasztalatok, magyarázat...

A Biomorfok azok egy alakzatok, formák amik valamilyen biológiai alakra hasonlítanak. Egy ilyen alakzatot, formát azaz Biomorfot kell nekünk most készíteni Mandelbrolt-halmaz segítségével.

Nézzük a forráskódot:

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

Először is szükségünk lesz függvénykönyvtárakra, ezt az előző feladatomból a már jól megszokott iostream, png++/png és az előző feladatban megismert complex függvénykönyvtár kell nekünk ebben a feladatban.

```
int main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
```

```
double xmin = -1.9;
double xmax = 0.7;
double ymin = -1.3;
double ymax = 1.3;
```

Ugorhatunk is a main részre, ahol javarész szintén megegyzik az előző forráskódunkkal ez a rész, kivéve az utolsó 2 sort. Addig ugyanúgy megadjuk a szélességet magasságot stb stb, most a lényeg az utolsó 2 soron van.

```
double reC = .285, imC = 0;
double R = 10.0;
```

Az utolsó 2 sorban megadjuk az imC és a reC változókat, valamint az R változót is. Ebben különbözik eddig az előző programtól.

```
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] );
```

Nézzük az if-es részt, itt most azt vizsgáljuk, hogy az argumentumunk mikor lesz 12, ha ez teljesül, akkor végrehajtja az előző feladatban már megszokottakat, csak most ad parancssori argumentumot az ujjonan definiált reC, imC és R-nek is.

```
// j megy a sorokon
for ( int y = 0; y < magassag; ++y )
{
// k megy az oszlopokon

for ( int x = 0; x < szelesseg; ++x )

double rez = xmin + x * dx;
double imZ = ymax - y * dy;
std::complex<double> z_n ( rez, imZ );

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

```
    z_n = std::pow(z_n, 3) + cc;
    //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
    if(std::real(z_n) > R || std::imag(z_n) > R)
    {
        iteracio = i;
        break;
    }
    kep.set_pixel(x, y,
                  png::rgb_pixel((iteracio*20)%255, (iteracio*40)%255, (iteracio*60)%255));
}
```

```
// j megy a sorokon
for (int y = 0; y < magassag; ++y)
{
// k megy az oszlopokon

for (int x = 0; x < szelessseg; ++x)
```

Itt is a már jól ismert 2 for ciklussal végig megyünk a magasságon és a szélességen is, így megkapjuk a "rácsot".

A továbbiakban létrehozzuk a `z_n` változót, majd találkozunk ismét egy for ciklussal, ami addig fut, amíg kisebb mint az iterációshatár.

```
    kep.set_pixel(x, y,
                  png::rgb_pixel((iteracio*20)%255, (iteracio*40)%255, (iteracio*60)%255));
}

int szazalek = (double)y / (double)magassag * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
}

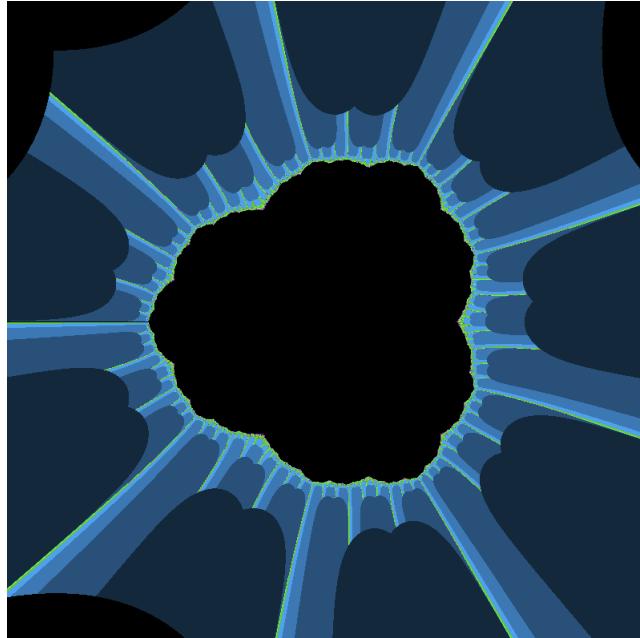
kep.write(argv[1]);
std::cout << "\r" << argv[1] << " mentve." << std::endl;
```

A `set_pixel` itt is elkészíti nekünk majd a képet, és színt adunk a képpontoknak, úgy ahogy azt már az előzőleg megtettük.

A programunk végéhez közeledve itt is megtalálhatjuk a százalek változót, ami a kirajzoltatásban segít, valamint találkozunk ismét a `write` függvényel, ahol ismét dokumentáljuk azt, hogy a képünk mentve lett.

Nézzük mi történik forrdításkor és futtatáskor:

```
Administrator-MacBook-Pro:~ user$ g++ 3.1.3.cpp -lpng -O3 -o ↵
biomorf
Administrator-MacBook-Pro:~ user$ ./biomorf biomorf.png 800 800 10 ↵
-2 2 -2 2 .285 0 10
Szamitas
biomorf.png mentve
```



## 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)

Ebben a feladatban a CUDA-ról lesz szó. Az NVIDIA-nak van CUDA kártyája, tehát a programunk csak abban az esetben fog működni, ha mi rendelkezünk egy ilyen NVIDIA-CUDA kártyával, más esetben a programunk nem fog működni. A lényege röviden az, hogy a videókártya segítségével tud párhuzamosan számolni.

Akinek van CUDA magos kártyája, annak a futtatáshoz még telepítenie kell az nvidia-cuda-toolkit-et.

Mivel a feladathoz a már említett kártya szükséges, és mivel én ilyen kártyával nem rendeklezem, ezért csak a forráskódot próbálom elemezni.

Nézzük a forráskódot:

```
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>

#include <sys/types.h>
#include <iostream>
```

A forráskód első néhány sorában be importáljuk a szükséges függvénykönyvtárakat, a legtöbb már ismerős lehet számunkra, azonban a sys/times.h függvénykönyvtárral még nem találkoztunk, ez új lehet számunkra.

```
#define MERET 600
#define ITER_HAT 32000
```

Ismét definiáljuk a Mandelbroot-halmazhoz hasonlóan a méretet, majd az iterációs határt is, ahogy azt láthatjuk.

```
__device__ int mandel (int k, int j)
```

A device azért fontos mert ezzel azt érjük el, hogy a fóggvényt a kártya vigye véghez.

```
{
    // Végigzongorázza a CUDA a szélesség x magasság rácson:
    // most eppen a j. sor k. oszlopban vagyunk

    // számítás adatai

    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ←
        ITER_HAT;

    // a számítás

    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;

    // Hány iterációt csináltunk?

    int iteracio = 0;
```

Ezt követően végig megyünk a magasság és a szélesség rácson, ezt követően láthatjuk a számítás adatait, majd magát a számolást is.

```
// Hány iterációt csináltunk?

int iteracio = 0;
```

Majd vizsgáljuk, hogy hány iterációt csináltunk, illetve látjuk a rács csomópontjainak a megfelelő komplex számot is.

```
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteracionsHatar ←
)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;

}
return iteracio;
}
```

A while ciklusban kismet azt vizsgáljuk, amit már megtettünk az előző feladatokban, tehát a z abszolultértéke kisebb-e mint 4 és azt, hogy az iteráció kisebb e mint az iterációshatár. Ezt követően megváltozik z értéke ahogyan eddig, majd a művelet végénaz iterációt növeljük.

Az iterációt fogjuk visszakapni.

```
__global__ void mandelkernel (int *kepadat)
```

A global-t a device-he hasonlóan a videókártya fogja megvalósítani.

Nézzük a main részt:

```
int main (int argc, char *argv[])
{
    for (int j = 0; j < MERET; ++j)
    {
        //sor = j;
        for (int k = 0; k < MERET; ++k)
    {
```

Látunk benne 2 egymásba ágyazott for ciklust, melyek a sorokon és az oszlopokon fognak véig menni.

```

    {
        kep.set_pixel (k, j,
            png::rgb_pixel (255 -
```

```
(255 * kepadat[j][k]) / ITER_HAT,  
255 -  
(255 * kepadat[j][k]) / ITER_HAT,  
255 -  
(255 * kepadat[j][k]) / ITER_HAT));  
}  
}  
kep.write(argv[1]);
```

Ezt követően a már ismert set\_pixel elkészíti a képet és ki is színezi nekünk a képpontokat.

```
kep.write(argv[1]);  
  
std::cout << argv[1] << " mentve" << std::endl;
```

Találkozunk még a már ismert write függvényel, ahol megint csak tudomásul vesszük, hogy a képünk mentve lett.

```
times(&tmsbuf2);  
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime  
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;  
  
delta = clock() - delta;  
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
```

A forráskód végén pedig, megtudhatjuk, hogy sikerült-e a sikeres futtatás, valamint azt, hogy ez mennyi időt használt fel, láthatjuk hogyan dolgozik a programunk.

## 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!

Megoldás forrása: [https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel\\_nagyito-frakszal.h](https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel_nagyito-frakszal.h)

[frakszal.h](https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel_nagyito-frakszal.h)

[frakszal.cpp](https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel_nagyito-frakszal.cpp)

[frakablak.h](https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel_nagyito-frakablak.h)

[frakablak.cpp](#)[main.cpp](#)

A program megfelelő működéséhez a QT GUI-t kell használnunk. Ha ez nem elérhető számunkra, akkor le kell töltenünk. A QT-ben grafikusan lehet létrehozni programokat, grafikusan jeleníti meg azt.

Ebben a feladatban több fájlt is találunk.

Nézzük ezeket a fájlokat:

[frakablak.h](#)

```
#ifndef FRAKABLAK_H
#define FRAKABLAK_H

#include <QMainWindow>
#include <QImage>
#include <QPainter>
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"

class FrakSzal;

class FrakAblak : public QMainWindow
{
    Q_OBJECT

public:
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
               double d = 1.35, int szelesseg = 600,
               int iteraciosHatar = 255, QWidget *parent = 0);
    ~FrakAblak();
    void vissza(int magassag, int * sor, int meret) ;
    void vissza(void) ;
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c ←
    // iterációt?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;

protected:
    void paintEvent(QPaintEvent*) ;
    void mousePressEvent(QMouseEvent*) ;
    void mouseMoveEvent(QMouseEvent*) ;
    void mouseReleaseEvent(QMouseEvent*) ;
    void keyPressEvent(QKeyEvent*) ;
```

```
private:  
    QImage* fraktal;  
    FrakSzal* mandelbrot;  
    bool szamitasFut;  
    // A nagyítandó kijelölt területet bal felső sarka.  
    int x, y;  
    // A nagyítandó kijelölt terület szélessége és magassága.  
    int mx, my;  
};  
  
#endif // FRAKABLAK_H
```

Ez egy header fájl. Láthatjuk hogy 3 különböző tag található benne: public, private és a protected is megtalálható. Itt a program feldolgozza az esetleges billentyűnyomást, és az egérmozgatást valamint az egérrel való kattintást is.

### frakablak.cpp

```
void FrakAblak::paintEvent (QPaintEvent*) {  
    QPainter qpainter(this);  
    qpainter.drawImage(0, 0, *fraktal);  
    if (!szamitasFut) {  
        qpainter.setPen(QPen(Qt::white, 1));  
        qpainter.drawRect(x, y, mx, my);  
  
    }  
    qpainter.end();  
}  
  
void FrakAblak::mousePressEvent (QMouseEvent* event) {  
  
    // A nagyítandó kijelölt területet bal felső sarka:  
    x = event->x();  
    y = event->y();  
    mx = 0;  
    my = 0;  
  
    update();  
}  
  
void FrakAblak::mouseMoveEvent (QMouseEvent* event) {  
  
    // A nagyítandó kijelölt terület szélessége és magassága:  
    mx = event->x() - x;  
    my = mx; // négyzet alakú  
  
    update();  
}  
  
void FrakAblak::mouseReleaseEvent (QMouseEvent* event) {
```

```
if (szamitasFut)
    return;

szamitasFut = true;

double dx = (b-a)/szelesseg;
double dy = (d-c)/magassag;

double a = this->a+x*dx;
double b = this->a+x*dx+mx*dx;
double c = this->d-y*dy-my*dy;
double d = this->d-y*dy;

this->a = a;
this->b = b;
this->c = c;
this->d = d;

delete mandelbrot;
mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
    iteraciosHatar, this);
mandelbrot->start();

update();
}

void FrakAblak::keyPressEvent(QKeyEvent *event)
{

    if (szamitasFut)
        return;

    if (event->key() == Qt::Key_N)
        iteraciosHatar *= 2;
    szamitasFut = true;

    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
        iteraciosHatar, this);
    mandelbrot->start();

}
```

Ebben a részben definiáljuk a már előbb bemutatott frekablak.h header fájlban megadottakat.

main.cpp

```
#include <QApplication>
#include "frakablak.h"
```

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    FrakAblak w1;
    w1.show();
    return a.exec();
}
```

A main fájlban beimportáltuk a QApplication függvénykönyvtárat, valamint beimportáljuk a már előbb megbeszélt frakablak.h header fájlt, így erre hivatkozunk a main programunkban. A továbbiakban a QApplication-nel példányosítunk ebben a részben.

### frakszal.h

```
#ifndef FRAKSZAL_H
#define FRAKSZAL_H

#include <QThread>
#include <math.h>
#include "frakablak.h"

class FrakAblak;

class FrakSzal : public QThread
{
    Q_OBJECT

public:
    FrakSzal(double a, double b, double c, double d,
              int szelesseg, int magassag, int iteraciosHatar, FrakAblak *frakAblak);
    ~FrakSzal();
    void run();

protected:
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a  $z_{n+1} = z_n \cdot z_n + c$  iterációt ?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;
    // Kinek számolok?
    FrakAblak* frakAblak;
    // Soronként küldöm is neki vissza a kiszámoltakat.
    int* egySor;

};
```

```
#endif // FRAKSZAL_H
```

Ez is egy header fájl hasonlóan a frakablak.h-hoz. Ebben a fájlban dekraláljuk az olyan változókat, amiket később a számolásnál illetve a razjzolásnál a hasznunkra lehet.

frakszal.cpp

```
#include "frakszal.h"

FrakSzal::FrakSzal(double a, double b, double c, double d,
                     int szelesseg, int magassag, int ←
                     iteraciosHatar, FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;

    egySor = new int[szelesseg];
}

FrakSzal::~FrakSzal()
{
    delete[] egySor;
}

void FrakSzal::run()
{
    // A [a,b]x[c,d] tartományon milyen sűrű a
    // megadott szélesség, magasság háló:
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double reC, imC, rez, imZ, ujrez, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság hálót:
    for(int j=0; j<magassag; ++j) {
        //sor = j;
        for(int k=0; k<szelesseg; ++k) {
            // c = (reC, imC) a háló rácspontjainak
            // megfelelő komplex szám
            reC = a+k*dx;
            imC = d-j*dy;
            // z_0 = 0 = (rez, imZ)
            rez = 0;
            imZ = 0;
```

```
iteracio = 0;
// z_{n+1} = z_n * z_n + c iterációk
// számítása, amíg |z_n| < 2 vagy még
// nem értük el a 255 iterációt, ha
// viszont elértek, akkor úgy vesszük,
// hogy a kiinduláci c komplex számra
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while(reZ*reZ + imZ*imZ < 4 && iteracio < ←
    iteracionsHatar) {
    // z_{n+1} = z_n * z_n + c

    ujreZ = reZ*reZ - imZ*imZ + reC;
    ujimZ = 2*reZ*imZ + imC;

    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;

}
// ha a < 4 feltétel nem teljesült és a
// iteráció < iteracionsHatar sérülésével lépett ki, ←
// azaz
// feltessük a c-ről, hogy itt a z_{n+1} = z_n * z_n ←
// + c
// sorozat konvergens, azaz iteráció = iteracionsHatar
// ekkor az iteráció %= 256 egyenlő 255, mert az ←
// esetleges
// nagyítások során az iteráció = valahány * 256 + ←
// 255

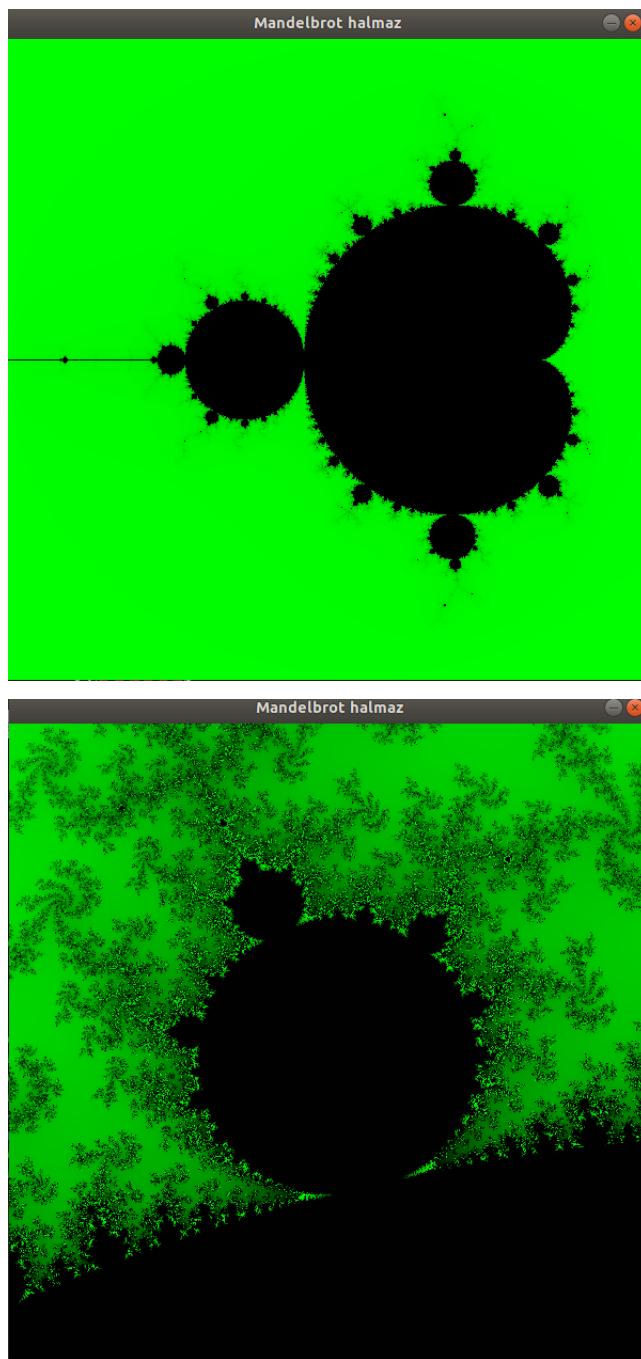
iteracio %= 256;

// a színezést viszont már majd a FrakAblak osztályban ←
// lesz
egySor[k] = iteracio;
}
// Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.
frakAblak->vissza(j, egySor, szelesség);
}
frakAblak->vissza();

}
```

Ebben a fájlban történik a Mandelbrot-halmaz rajzolása, ebben a részben a már ismerős dolgokat láthatjuk, ahogyan végig halad a szélességes és a magasságban is egyaránt.

Miután röviden sikerült rájönni melyik fájlban mi található, próbáljuk meg működésre bírni a QT-t, majd fordítás után ezt látjuk:



## 5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html>

[MandelbrotHalmaz.java](#)

[MandelbrotHalmazNagyító.java](#)

Ebben a részben a feladat, az előzőleg megoldott programunkat kell fejleszteni, Mandelbrot nagyító és utazó Java nyelven.

Lássuk a forrást:

```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {  
    /** A nagyítandó kijelölt területet bal felső sarka. */  
  
    private int x, y;  
    /** A nagyítandó kijelölt terület szélessége és magassága. */  
  
    private int mx, my;
```

Láthatjuk MandelbrotHalmazNagyító osztályt és ebben a sorban van az extends is. Ezen kívül még létrehozunk 2 privát változót is : int x, y és int mx, my.

```
public MandelbrotHalmazNagyító(double a, double b, double c, double d,  
                                 int szélesség, int iterációsHatár) {  
  
    super(a, b, c, d, szélesség, iterációsHatár);  
    setTitle("A Mandelbrot halmaz nagyításai");  
  
    // Egér kattintó események feldolgozása:  
    addMouseListener(new java.awt.event.MouseAdapter() {  
        // Egér kattintással jelöljük ki a nagyítandó területet  
        // bal felső sarkát:  
        public void mousePressed(java.awt.event.MouseEvent m) {  
            // A nagyítandó kijelölt területet bal felső sarka:  
            x = m.getX();  
            y = m.getY();  
            mx = 0;  
            my = 0;  
            repaint();  
        }  
  
        public void mouseReleased(java.awt.event.MouseEvent m) {  
            double dx = (MandelbrotHalmazNagyító.this.b  
                         - MandelbrotHalmazNagyító.this.a)  
                        /MandelbrotHalmazNagyító.this.szélesség;  
            double dy = (MandelbrotHalmazNagyító.this.d  
                         - MandelbrotHalmazNagyító.this.c)  
                        /MandelbrotHalmazNagyító.this.magasság;  
  
            // Az új Mandelbrot nagyító objektum elkészítése:  
            new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+  
                                         x*dx,  
                                         MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,  
                                         MandelbrotHalmazNagyító.this.d-y*dy-my*dy,  
                                         MandelbrotHalmazNagyító.this.d-y*dy,  
                                         600,  
                                         MandelbrotHalmazNagyító.this.iterációsHatár);  
        }  
    });
```

Ezt követően feldolgozzuk a kattintó eseményeket amiket az egérrel viszünk be, majd kijelöljük a bal felső sarkot.

A new MandelBrotHalmazNagyító résznél elkészítjük az új Mandelbrot nagyító objektumot, miután ezt létrehoztuk, feldolgozzuk a mozgás eseményeket is.

```
// Egér mozgás események feldolgozása:  
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {  
    // Vonszolással jelöljük ki a négyzetet:  
    public void mouseDragged(java.awt.event.MouseEvent m) {  
        // A nagyítandó kijelölt terület szélessége és magassága:  
        mx = m.getX() - x;  
        my = m.getY() - y;  
        repaint();  
    }  
});  
}
```

Ezután megadjuk azt amit ki szeretnénk nagyítani, és kijelöljük ezt a területet és megadjuk ennek a szélességet és magasságát is.

```
repaint();  
}
```

A repaint arra szolgál, hogy a nagyítás követően kirajzolja a halmazt.

```
public void pillanatfelvétel() {  
    // Az elmentendő kép elkészítése:  
  
    java.awt.image.BufferedImage mentKép =  
        new java.awt.image.BufferedImage(szélesség, magasság,  
            java.awt.image.BufferedImage.TYPE_INT_RGB);  
    java.awt.Graphics g = mentKép.getGraphics();  
    g.drawImage(kép, 0, 0, this);  
    g.setColor(java.awt.Color.BLUE);  
    g.drawString("a=" + a, 10, 15);  
    g.drawString("b=" + b, 10, 30);  
    g.drawString("c=" + c, 10, 45);  
    g.drawString("d=" + d, 10, 60);  
    g.drawString("n=" + iterációsHatár, 10, 75);  
    if(számításFut) {  
        g.setColor(java.awt.Color.RED);  
        g.drawLine(0, sor, getWidth(), sor);  
    }  
    g.setColor(java.awt.Color.GREEN);
```

```
g.drawRect(x, y, mx, my);
g.dispose();
// A pillanatfelvétel képfájl nevének képzése:
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyitas_");
sb.append(++pillanatfelvételszámláló);
sb.append("_");
// A fájl nevébe belevesszük, hogy melyik tartományban
// találtuk a halmazt:

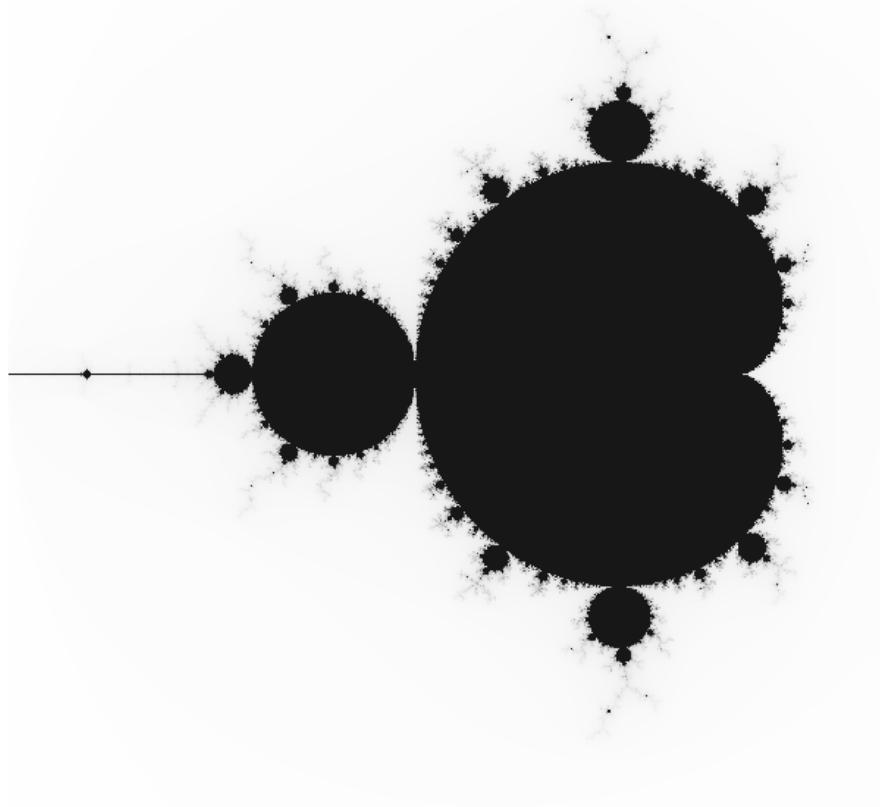
sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".");
sb.append(".png");
// png formátumú képet mentünk
```

A public void pillanatfelvétel részben létrehozzuk az üres képfájlt ami kell a program megfelelő működéséhez, valamint létrejön benne a képunkben a képpontok színezése is.

```
public static void main(String[] args) {
    // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
    // tartományában keressük egy 600x600-as hálóval és az
    // aktuális nagyítási pontossággal:
    new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
```

A program végéhez közeledve a main részhez érünk, ahol sor kerül a várva várt példányosításra, majd itt adjuk meg a szükséges paramétereket a program működéséhez.

Nézzük mit kapunk:



## 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:

Megoldás forrása:

[PolarGenerator.java](#)

Tanulságok, tapasztalatok, magyarázat... térd ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked!

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html>

Nézzük a Java-s programot:

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

Az algoritmus a PolgárGenerátor osztályban történik. Itt találunk egy boolean típusú változót is a nincsTároltat mely true-t ad vissza. Továbbá van egy double típusú változó is a tárolt.

```
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;

}
```

Találkozunk a do while ciklussal, ez akkor fut le ha nincsTárolt igaz. Ekkor az u1 és az u2 fog kapni egy véletlen számot, melyet a Math.random() függvény ad. A többiek az előző változókból kapják meg az értéküket.

Találunk továbbá egy r változót is, amit a Math.sqrt függvény segítségével számítunk ki.

```
else {
    nincsTárolt = !nincsTárolt;
    return tárolt;
}
}
```

Az else részben, ha a feltételünk nem valósul meg, akkor a már tárolt elemet adja majd vissza nekünk. A main részben gyakorlatilag semmi érdekes nem történik már, csak meghívjuk a függvényt és példányosítunk.

A forráskód C++-ban

Megoldás forrása: [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1\\_5.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf)

```
#ifndef POLARGEN_H
#define POLARGEN_H

#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;
```

```
class PolarGenerator
{
public:
    PolarGenerator ()
    {
        nincsTarolt = true;
        srand(time(NULL));
    }

    ~PolarGenerator()
    {

    }

    double kovetkezo ()
    {
        if (nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do
            {
                u1 = rand () / (RAND_MAX + 1.0);
                u2 = rand () / (RAND_MAX + 1.0);
                v1 = 2 * u1 - 1;
                v2 = 2 * u2 - 1;
                w = v1 * v1 + v2 * v2;
            }
            while (w > 1);

            double r = sqrt ((-2 * log (w)) / w);

            tarolt = r * v2;
            nincsTarolt = !nincsTarolt;

            return r * v1;
        }
        else
        {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }

private:
    bool nincsTarolt;
    double tarolt;
};
```

```
#endif

#include <iostream>

int main (int argc, char **argv)
{
    PolarGenerator polargen;

    for (int i = 0; i < 10; ++i)
        std::cout << polargen.kovetkezo () << "\n";

    return 0;
}
```

Amint látjuk a forráskódban a PolarGenerator részben létrehozzuk a konstruktort, és a destruktort is.

A kovetkezo függvény láthatjuk még, illetve még 2 változó a bool típusú nincsTarolt, valamint a double típusú tarolt változó is. Ez a 2 változó privateban van.

## 6.2. LZW

Valósítsd meg C-ben az LZW algoritmus fa-építését!

Megoldás videó:

Megoldás forrása:

[lzw.c](#)

Mi is az az LZW? Az LZW egy tömörítési eljárás. Ez az LZW tömörítés Lempel, Ziv valamint Welch nevéhez fűződik, így alakult ki az LZW.

A feladat lényege, hogy egy bináris fát kell felépíteni, ezzel az LZW eljárással. Ennek a fának a tulajdonsága az, hogy minden csomópontnak maximum 2 rákövetkezője van.

Két előfordulása van ha az értéke (bit) 0 vagy 1.

Nézzük a programot:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
```

Beimportáljuk a szükséges függvénykönyvtárakat, az stdio.h-t , az stdlib.h-t, unistd.h-t, és a math.h-t.

```
typedef struct binfa
{
    int ertek;
    struct binfa *bal nulla;
    struct binfa *jobb_egy;
```

```
    } BINFA, *BINFA_PTR;
```

Ezt követően láthatjuk magát a struktúrát, melynek 3 tagja van: az ertek, a bal\_nulla és a jobb\_egy mutató.

```
BINFA_PTR uj_elem ()
{
    BINFA_PTR p;

    if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
        perror ("memoria");
        exit (EXIT_FAILURE);
    }
    return p;
}
```

Ennél a résznél az uj\_elem lehet érdekes ez a függvény akkor fog nekünk szólni, ha kevés a memóriánk. Ezzel tudunk továbbá csomópontot kreálni is.

Át is térhetünk a main részére a programnak.

```
int main (int argc, char **argv)
{
    char b;

    BINFA_PTR gyoker = uj_elem ();
    gyoker->ertek = '/';
    gyoker->bal_nulla = gyoker->jobb_egy = NULL;
    BINFA_PTR fa = gyoker;
```

A main részben láthatjuk a gyokeret, ahol a bal\_nullanak, és a jobb\_egy-nek NULL értéket adunk.

Lássuk, hogyan épül fel a fánk.

```
while (read (0, (void *) &b, 1))
{
    if (b == '0')
    {
        if (fa->bal_nulla == NULL)
        {
            fa->bal_nulla = uj_elem ();
            fa->bal_nulla->ertek = 0;
            fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
            fa = gyoker;
```

```
        }
    else
    {
        fa = fa->bal_nulla;
    }
}
else
{
    if (fa->jobb_egy == NULL)
    {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
    }
    else
    {
        fa = fa->jobb_egy;
    }
}
}

printf ("\n");
kiir (gyoker);
```

Ehhez szükségesek az alábbi feltételvizsgálatok. A lényeg ha 0-át olvasunk be és a leendő csomópontnak nem áll rendelkezésre a 0-ás gyermek, ekkor meghívjuk az uj\_elem függvényt és megkapja a 0-ás értéket. Ezt követően vissza is ugorhatunk ismét a gyökérre.

Ha 1-est olvasunk be akkor ez is így működik, ahogy eljártunk a 0 esetén.

```
/* Átlagos ághossz kiszámítása */
atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
ratlag (gyoker);
// atlag = atlagosszeg / atlagdb;

atlag = ((double)atlagosszeg) / atlagdb;

atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
szorasosszeg = 0.0;

rszoras (gyoker);

double szoras = 0.0;

if (atlagdb - 1 > 0)
```

```
    szoras = sqrt( szorasosszeg / (atlagdb - 1));
else
    szoras = sqrt (szorasosszeg);

    printf ("atlag=%f\nszoras=%f\n", atlag, szoras);

    szabadit (gyoker);
}
```

Ezt követően már a számolásoké a főszerep.

Először is kiszámoljuk az átlagos ághosszat, melyet így tehetünk meg, majd ez kiíratásra is kerül.

Ezzel a számolással fejeztük be a main részét a programunkban.

```
int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

void ratlag (BINFA_PTR fa)
{
    if (fa != NULL)
    {
        ++melyseg;
        ratlag (fa->jobb_egy);
        ratlag (fa->bal_nulla);
        --melyseg;

        if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

double szorasosszeg = 0.0, atlag = 0.0;

void rszoras (BINFA_PTR fa)
{
    if (fa != NULL)
    {
        ++melyseg;
        rszoras (fa->jobb_egy);
        rszoras (fa->bal_nulla);
        --melyseg;

        if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
```

```
{  
    ++atlagdb;  
    szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));  
}  
}  
  
}  
int max_melyseg = 0;  
  
void kiir (BINFA_PTR elem)  
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        if (melyseg > max_melyseg)  
            max_melyseg = melyseg;  
        kiir (elem->jobb_egy);  
  
        for (int i = 0; i < melyseg; ++i)  
            printf ("---");  
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : ←  
               elem->ertek,  
               melyseg-1);  
        kiir (elem->bal nulla);  
        --melyseg;  
    }  
}  
  
void szabadit (BINFA_PTR elem)  
{  
    if (elem != NULL)  
    {  
        szabadit (elem->jobb_egy);  
        szabadit (elem->bal nulla);  
        free (elem);  
    }  
}
```

A program végéhez közeledve, már a binfáról kapunk egy képet, ahol megtudjuk a fa mélységét, szórását, és az átlagát is.

A szabadít függvény lényege, hogy memóriát szabadítson fel, azokból amiket már előzetesen lefoglaltunk. Nézzük, hogyan fordul le és fut la a programunk:

```
Administrator-MacBook-Pro:~ user$ cat kimenet.txt  
-----1(1)  
-----0(2)
```

```
---/ (0)
-----1 (3)
-----0 (4)
-----0 (5)
-----1 (2)
-----0 (3)
-----0 (1)
-----1 (4)
-----1 (3)
-----0 (2)
-----1 (4)
-----0 (5)
-----0 (3)
-----0 (4)
melyseg = 5
atlag = 3.83333
szoras = 1.16905
```

## 6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

[lzw.c](#)

A feladatban a fabejárásról lesz szó. A fáknak a bejárásánál 3 módszerről ejtünk igazán szót. Ez a 3 bejárási mód az inorder, preorder és a postorder. Nézzük mik is ezek a bejárások.

Inorder: Ebben az esetben valamelyik oldalát a fának feldolgozzuk, majd ezt követi a gyökér és vegül a másik oldal.

Preorder: Először feldolgozzuk a gyökérelemet, majd ezt követi a két oldal.

Postorder: Gyakorlatilag 1 lehetőség maradt már csak, ez úgy működik, hogy feldolgozza a 2 oldalt, és majd csak utána dolgozza fel a gyökérelemet.

Nézzük a Preordert:

```
void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c", elem->val);
        kiir (elem->left);
        kiir (elem->right);
    }
}
```

```
printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->←
        ertek,
        melyseg);
if (melyseg > max_melyseg)
    max_melyseg = melyseg;

kiir (elem->jobb_egy);
kiir (elem->bal nulla);
--melyseg;
}
}
```

Ez a bejárási módszernél, azt kell csinálni, hogy az elemek a kiíratását, fel kell hoznunk a program tetejére, oda amikor már megnöveltük a mélység változónkat. Ezáltal a programunkban elsőként a gyökérelem fog kerülni kiíratásra.

Nézzük a Postordert:

```
void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;

        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        kiir (elem->jobb_egy);
        kiir (elem->bal nulla);

        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->←
                ertek,
                melyseg);

        --melyseg;
    }
}
```

Ez a fabejárási módszer annyiban különbözik a preorder bejárástól, hogy itt már a kiíratások a végére kerülnek, oda ahol csökkentjük a mélység változót. Ezzel azt fogjuk elérni, hogy a gyökérelemet utoljára fogja kiírni.

## 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

[lzw.cpp](#)

Tutor: [Bacsik Mátyás](#)

A feladat megoldásához, a már előzőleg átírt binfát fogom használni.

Nézzük a forráskódot:

```
#include <iostream>
#include <cmath>
#include <fstream>
```

Beimportáljuk a szükséges függvénykönyvtárakat az iostream, math, és végül az fstreamet is.

```
class LZWBinFa
{
public:

    LZWBinFa () :fa (&gyoker)
    {
    }
    ~LZWBinFa ()
    {
        szabadit (gyoker.egyesGyermek ());
        szabadit (gyoker.nullasGyermek ());
    }
```

Ezt követően megalapítjuk az LZWBinfa osztályt. Ha ezekkel megvagyunk át is térhetünk a publikus részre, ahol látunk egy konstruktort, illetve egy destruktort is.

A konstruktur megkapja paraméterként a gyoker csomópontot. A destruktornak a feladata, hogy felszabadítja a fát.

Nézzük a fa felépítését:

```
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 ();
    }
}
}

void kiir (void)
{
    melyseg = 0;
    kiir (&gyoker, std::cout);
}

int getMelyseg (void);
double getAtlag (void);
double getSzoras (void);

friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)
{
    bf.kiir (os);
    return os;
}
void kiir (std::ostream & os)
{
    melyseg = 0;
    kiir (&gyoker, os);
}
```

A fa felépítése egy operátorral kezdődik, ahol megadunk változókat, de ezek már az egyes feladatok végre-hajtására szükségesek, tehát, ez most számunkra nem érdekes.

A private rész:

```
private:
    class Csomopont
    {
public:
    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
```

```
{  
};  
~Csomopont ()  
{  
};  
  
Csomopont *nullasGyermek () const  
{  
    return balNulla;  
}  
  
Csomopont *egyesGyermek () const  
{  
    return jobbEgy;  
}  
  
void ujNullasGyermek (Csomopont * gy)  
{  
    balNulla = gy;  
}  
  
void ujEgyesGyermek (Csomopont * gy)  
{  
    jobbEgy = gy;  
}  
  
char getBetu () const  
{  
    return betu;  
}  
  
private:  
  
    char betu;  
    Csomopont *balNulla;  
    Csomopont *jobbEgy;  
    Csomopont (const Csomopont &);  
    Csomopont & operator= (const Csomopont &);  
};
```

Tovább haladva elérkezünk a private részhez, ahol megadjuk a Csomópont osztályt. Ennek a lényege az, hogy ez beágyazásra került az LZWBinfा osztályba. Ezt követi a publikus rész, ahol megadjuk a konstruktort, a destruktort, valamint különböző változók jelennek még meg.

```
Csomopont *fa;  
int melyseg, atlagosszeg, atlagdb;  
double szorasosszeg;  
LZWBinfा (const LZWBinfा &);  
LZWBinfा & operator= (const LZWBinfा &);
```

```
void kiir (Csomopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->>nullasGyermek (), os);
        --melyseg;
    }
}
void szabadit (Csmopont * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek ());
        szabadit (elem->>nullasGyermek ());
        delete elem;
    }
}
```

Miután végeztünk a privát résszel is elérkeztünk a Csomópont kiíratása részhez, ahol egyéb algoritmusokat találunk.

Jöjjön a protected rész a programban:

```
protected:
    Csmopont gyoker;
    int maxMelyseg;
    double atlag, szoras;

    void rmelyseg (Csmopont * elem);
    void ratlag (Csmopont * elem);
    void rszoras (Csmopont * elem);

};
```

Ahogyan haladunk a forráskódunkban, elérjük a protected részt is a progarmunkban. Ebben a részben létrehozza a gyökérelemet a programunk, amit még az elején láthattunk.

Nézzük mi van még a forráskódban:

```
int
LZWBinFa::getMelyseg (void)
{
```

```
melyseg = maxMelyseg = 0;
rmelyseg (&gyoker);
return maxMelyseg - 1;
}

double
LZWBinFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}

double
LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (&gyoker);

    if (atlagdb - 1 > 0)
        szoras = std::sqrt (szorasosszeg / (atlagdb - 1));
    else
        szoras = std::sqrt (szorasosszeg);

    return szoras;
}

void
LZWBinFa::rmelyseg (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek ());
        rmelyseg (elem->>nullasGyermek ());
        --melyseg;
    }
}

void
LZWBinFa::ratlag (Csomopont * elem)
{
    if (elem != NULL)
    {
```

```
    ++melyseg;
    ratlag (elem->egyesGyermek ());
    ratlag (elem->>nullasGyermek ());
    --melyseg;
    if (elem->egyesGyermek () == NULL && elem->>nullasGyermek () == NULL -->
        )
    {
        ++atlagdb;
        atlagosszeg += melyseg;
    }
}

void
LZWBinFa::rszoras (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyesGyermek ());
        rszoras (elem->>nullasGyermek ());
        --melyseg;
        if (elem->egyesGyermek () == NULL && elem->>nullasGyermek () == NULL -->
            )
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}
```

Ezt követi a a feladatok megoldása, illetve látunk egy olyan metódust, ami kiírja majd az outputra azt, hogy hogyan kellene a programunkat használni.

Végül pedig nézzük a main részt:

```
int
main (int argc, char *argv[])
{
    if (argc != 4)
    {
        usage ();
        return -1;
    }
```

```
char *inFile = *++argv;

if (*((++argv) + 1) != 'o')
{
    usage ();
    return -2;
}

std::fstream beFile (inFile, std::ios_base::in);

if (!beFile)
{
    std::cout << inFile << " nem létezik..." << std::endl;
    usage ();
    return -3;
}

std::fstream kiFile (*++argv, std::ios_base::out);

unsigned char b;
LZWBinFa binFa;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
    if (b == 0x0a)
        break;

bool kommentben = false;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
{
    if (b == 0x3e)
    {
        kommentben = true;
        continue;
    }

    if (b == 0x0a)
    {
        kommentben = false;
        continue;
    }

    if (kommentben)
        continue;

    if (b == 0x4e)
        continue;
```

```
for (int i = 0; i < 8; ++i)
{
    if (b & 0x80)
        binFa << '1';
    else
        binFa << '0';
    b <<= 1;
}

kiFile << binFa;

kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;

kiFile.close ();
beFile.close ();

return 0;
}
```

Elérkeztünk a programunk main részéhez.

A main részben feldolgozásra kerül a szöveg, az érdekesség az, hogy bármilyen szöveget képes feldolgozni, tehát nem csak azt a szöveget amikor 0-át és 1-et ütünk le. Így el is értünk a forráskódunk végére, röviden végig mentünk a fán.

## 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:

[lzw.cpp](#)

Tutor: [Bacsik Mátyás](#)

Kezdésként a protected részben kell megkeresünk a Csomópontot.

```
Csomopont* gyoker;
```

A mutató a gyökér feladatban a protected részben kell dolgoznunk, ahol a megadott gyökeret hajlandóak leszünk átírni, ahhoz hogy a programunk működjön. Csomópont -> Csomopont\*-ra kell átírnunk a gyökeret.

Ezt követően nézzük meg a konstruktort:

```
LZWBinFa () :fa ()
{
```

```
gyoker=new Csomopont ('/');  
fa=gyoker;  
}
```

Ezután a konstruktőrben ejtünk változtatásokat, ahol a konstruktornak nem adjuk oda a gyökeret, sokkal inkább a gyökér mutatót adjuk oda neki, majd ezt fogjuk tárolni a binfában, úgy, hogy ő lesz az aktuális elem.

Miután végeztünk a konstruktőrral, már csak cserékre van szükségünk, ugyanis ahol eddig a gyökernek a referenciaját vártuk, mivel a gyökér már nem elem, sokkal inkább egy mutató, ezért elég már csak magát a gyökeret odaadnunk neki. Szóval ahol &gyoker-et látunk oda csak simán gyoker-et kell írni a továbbiakban.

Az átírást követően, a programunkban sok változást nem tapasztalunk, a program teljesen úgy fog működni, ahogyan azt eddig tette, csak egy kicsit feldolgoztuk a programot és megváltoztattunk egy két módszert.

## 6.6. Mozgató szemantika

Ír az előző programhoz mozgató konstruktort és értékadást, a mozgató konstruktőr legyen a mozgató értékadásra alapozva!

Megoldás videó:

Megoldás forrása:

[lzw.cpp](#)

Tutor: [Bacsik Mátyás](#)

A mozgató szemantika feladatnál még az osztály elején, szükséges megadnunk két függvényt.

Az első függvény a move () lesz, amikor binfával hívjuk meg, akkor először őt fogja használni.

Nézzük mit is csinál a move: A move ()-nak a szerepe, hogy amit kapott értéket paraméterként, azt előkészítse a mozgatásra, tehát őt egy jobbértek referenciajába fogja alakítani.

Nézzük a forrásban hogyan is néz ki:

```
LZWBinFa (LZWBinFa&& masik) {  
    gyoker=nullptr;  
    *this= std::move(masik);  
    std::cout<<"LZWBinFa mozgato ctor\n";  
}
```

Amint látható, az volt a feladatunk, hogy a binfának a gyökérmutatójának megadtuk a nullptr-t.

A második függvényt az operátor működésénél használjuk.

```
LZWBinFa& operator= (LZWBinFa&& masik) {  
    std::swap(gyoker,masik.gyoker);  
    //std::cout<<"LZWBinFa mozgato ertekeadas";  
    return *this;  
}
```

Az operátorban az történik hogy a swap függvényel megcseréljük a két fának a gyökérmutatóját. Ezzel a két függvényel elérünk azt, hogy most már megfelőlen tudjuk mozgatni az adott objektumainkat. Végül pedig ezt ki is iratjuk:

```
LZWBinFa binFa2 = std::move(binFa);  
  
kiFile << binFa2;  
  
kiFile << "depth = " << binFa2.getMelyseg () << std::endl;  
  
kiFile << "mean = " << binFa2.getAtlag () << std::endl;  
  
kiFile << "var = " << binFa2.getSzoras () << std::endl;
```

Végül a move () függvényel megfelően tudjuk mozgatni az elemeket, és kiíratjuk az új, elkészült fát. Fontos azonban az, hogy ha ezek után mi mégis szeretnénk látni az eredeti Binfa-t akkor erre már nincsen lehetőségünk, mivel őt már kinulláltuk.

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyszimulációk

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

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

Megoldás forrása:

[ant.h](#)

[antthread.h](#)

[antthread.cpp](#)

[antwin.h](#)

[antwin.cpp](#)

[main.cpp](#)

Tanulságok, tapasztalatok, magyarázat...

A 7-es csokornak ez első részében a feladatunk egy szimuláció elkészítése, melynek neve hangyszimuláció. A program lényegében szimulálni fogja a hangyákat.

A forrás a Bhax repóból származik, ezeket fogom most felhasználni a megoldásomhoz.

Mivel ebben a részben 6 fájlal is találkozunk, ezért ezeket a fájlokat egyesével fogjuk feldolgozni, értelmezni.

Ahhoz, hogy grafikusan is megjelenjen a program, telepítenünk kell a Qt csomagot, e nélkül ez nem fog működni.

Nézzük is az 1. fájlt:

[ant.h](#)

```
#ifndef ANT_H
#define ANT_H

class Ant
```

```
{  
  
public:  
    int x;  
    int y;  
    int dir;  
  
    Ant(int x, int y) : x(x), y(y) {  
  
        dir = qrand() % 8;  
  
    }  
  
};  
  
typedef std::vector<Ant> Ants;  
  
#endif
```

Amint láthatjuk ez egy header fájl, amiben látunk egy objektumot, ennek az objektumnak van pozíciója ( x, y ) illetve van egy iránya is.

A konstruktorban kapja meg az irányát, és választ egy irányt ennek segítségre szolgál a qrand() függvény, és ennek a % 8-at nézi tehát, a 8 maradékait.

A forráskód végén láthatunk egy typdef-et ez tulajtunképpen megadja a vektor típusát, aminek azt mondja, hogy Ants legyen.

antthread.h

```
#ifndef ANTTHREAD_H  
#define ANTTHREAD_H  
  
#include <QThread>  
#include "ant.h"  
  
class AntThread : public QThread  
{  
    Q_OBJECT  
  
public:  
    AntThread(Ants * ants, int ***grids, int width, int height,  
              int delay, int numAnts, int pheromone, int nbrPheromone,  
              int evaporation, int min, int max, int cellAntMax);  
  
    ~AntThread();  
  
    void run();  
    void finish()  
{
```

```
        running = false;
    }

    void pause()
    {
        paused = !paused;
    }

    bool isRunning()
    {
        return running;
    }

private:
    bool running {true};
    bool paused {false};
    Ants* ants;
    int** numAntsinCells;
    int min, max;
    int cellAntMax;
    int pheromone;
    int evaporation;
    int nbrPheromone;
    int ***grids;
    int width;
    int height;
    int gridIdx;
    int delay;

    void timeDevel();

    int newDir(int sor, int oszlop, int vsor, int voszlop);
    void detDirs(int irany, int& ifrom, int& ito, int& jfrom, int& jto );
    int moveAnts(int **grid, int row, int col, int& retrow, int& retcol, ←
                 int);
    double sumNbhs(int **grid, int row, int col, int);
    void setPheromone(int **grid, int row, int col);

signals:
    void step ( const int &);

};

#endif
```

El is érkeztünk a második fájlhoz, amint láthatjuk ez ismét egy header fájl.

Ebben a feladatban beimportáljuk a Qthread könyvtárat, valamint beimportáljuk még a már előzleg megis-

mert ant.h-t is. Tehát lényegében ez a rész használni fogja az ant.h-t amit már megbeszéltünk.

Létrehozásra kerül egy osztály, mégpedig az AntThread osztály, ez a QThread “gyermeke”, így eléri azt, hogy használatba tudja venni a függvényeket és a változókat.

A konstruktorban megkapja azokat az utasításokat, amik a hangya szimulációban való számításokban van szerepe.

Megaláljuk még a programban a destruktort is.

Találozunk a programban a run() függvényt is őt is megadjuk.

A finish()-nél észrevehetjük, hogy a runnig változónak a false értéket adja meg. A pause()-nél a pause változónak az ellentétjét adja meg. Az isRunning() pedig visszaadja nekünk a running-nek az állapotát.

Nézzük a privát részt:

Láthatjuk, hogy a running az true-ra van állítva, valamint alatta a paused viszont false-ra. A többi változót amit látunk, azok már szerepelnek a konstruktorban is, egyetelen egy kivételével, ez pedig a gridIdx változó.

A forráskód végéhez közeledve láthatjuk, hogy megadásra kerültek a timeDevel, newDir, detDirs, moveAnts, sumNbhs, és a setPheromone is.

A legvégére pedig megadjuk a step-et is.

antwin.h

```
#ifndef ANTWIN_H
#define ANTWIN_H

#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCloseEvent>
#include "antthread.h"
#include "ant.h"

class AntWin : public QMainWindow
{
    Q_OBJECT

public:
    AntWin(int width = 100, int height = 75,
           int delay = 120, int numAnts = 100,
           int pheromone = 10, int nbhPheromon = 3,
           int evaporation = 2, int cellDef = 1,
           int min = 2, int max = 50,
           int cellAntMax = 4, QWidget *parent = 0);

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {

        antThread->finish();
        antThread->wait();
        event->accept();
    }
}
```

```
}

void keyPressEvent ( QKeyEvent *event )
{

    if ( event->key() == Qt::Key_P ) {
        antThread->pause();
    } else if ( event->key() == Qt::Key_Q
                || event->key() == Qt::Key_Escape ) {
        close();
    }

}

virtual ~AntWin();
void paintEvent (QPaintEvent *);

private:

int ***grids;
int **grid;
int gridIdx;
int cellWidth;
int cellHeight;
int width;
int height;
int max;
int min;
Ants* ants;

public slots :
    void step ( const int &);

};

#endif
```

Elérkeztünk az utlosó header fájlunkhoz.

Ebben a fájlból elég sok minden importálunk be ezeket most fel is sorolom: Kezdésként a QMainWindow, majt őt követik az alábbiak, QPainter, QString, QCloseEvent. Az utolsó két importálásnál pedig hivatkozunk 2 header fájlra, amiket az előbb már elemeztünk, tehát a antthread.h-t és az ant.h-t.

Láthatjuk az AntWin osztályt, ami a QMainWindow-nak a “gyermeke”.

Ezt követően találkozunk a konstruktorral, majd a destruktorttal, és még a privát rész előtt megadjuk a paintEvent() függvényt is.

A privát részben találkozunk a már konstruktorban definiált változókkal.

A forrás végén pedig láthatjuk a slot-ot ami tartalmazza a step()-et.

## antwin.cpp

```
#include "antwin.h"
#include <QDebug>

AntWin::AntWin ( int width, int height, int delay, int numAnts,
                 int pheromone, int nbhPheromon, int evaporation, int ←
                 cellDef,
                 int min, int max, int cellAntMax, QWidget *parent ) : ←
                 QMainWindow ( parent )
{
    setWindowTitle ( "Ant Simulation" );

    this->width = width;
    this->height = height;
    this->max = max;
    this->min = min;

    cellWidth = 6;
    cellHeight = 6;

    setFixedSize ( QSize ( width*cellWidth, height*cellHeight ) );

    grids = new int**[2];
    grids[0] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[0][i] = new int [width];
    }
    grids[1] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[1][i] = new int [width];
    }

    gridIdx = 0;
    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j ) {
            grid[i][j] = cellDef;
        }

    ants = new Ants();

    antThread = new AntThread ( ants, grids, width, height, delay, numAnts, ←
                               pheromone,
                               nbhPheromon, evaporation, min, max, ←
                               cellAntMax);

    connect ( antThread, SIGNAL ( step ( int ) ),
              this, SLOT ( step ( int ) ) );
}
```

```
    antThread->start();  
  
}  
  
void AntWin::paintEvent ( QPaintEvent* )  
{  
    QPainter qpainter ( this );  
  
    grid = grids[gridIdx];  
  
    for ( int i=0; i<height; ++i ) {  
        for ( int j=0; j<width; ++j ) {  
  
            double rel = 255.0/max;  
  
            qpainter.fillRect ( j*cellWidth, i*cellHeight,  
                                cellWidth, cellHeight,  
                                QColor ( 255 - grid[i][j]*rel,  
                                          255,  
                                          255 - grid[i][j]*rel ) );  
  
            if ( grid[i][j] != min )  
            {  
                qpainter.setPen ( QPen (  
                    QColor ( 255 - grid[i][j]*rel,  
                            255 - grid[i][j]*rel, 255 ),  
                    1 )  
                );  
  
                qpainter.drawRect ( j*cellWidth, i*cellHeight,  
                                    cellWidth, cellHeight );  
            }  
  
            qpainter.setPen ( QPen (  
                QColor ( 0, 0, 0 ),  
                1 )  
            );  
  
            qpainter.drawRect ( j*cellWidth, i*cellHeight,  
                                cellWidth, cellHeight );  
        }  
    }  
  
for ( auto h: *ants) {  
    qpainter.setPen ( QPen ( Qt::black, 1 ) );
```

```
    qpainter.drawRect ( h.x*cellWidth+1, h.y*cellHeight+1,
                        cellWidth-2, cellHeight-2 );

}

qpainter.end();
}

AntWin::~AntWin()
{
    delete antThread;

    for ( int i=0; i<height; ++i ) {
        delete[] grids[0][i];
        delete[] grids[1][i];
    }

    delete[] grids[0];
    delete[] grids[1];
    delete[] grids;

    delete ants;
}

void AntWin::step ( const int &gridIdx )
{
    this->gridIdx = gridIdx;
    update();
}
```

Az antwin.cpp forrásban beimprotáljuk a QDebug-ot, valamint a már előbb elemzett header fájlt a antwin.h-t.

Ebben a fájlban rögtön egy konstruktor fogad minket, amit definiálunk, (szélesség és magasság) ez a cellának a mérete valamint definiáljuk a destruktort is. Itt történik meg az, hogy kirajzoljuk a hangyákat.

Nézzük a main azaz a főprogramot:

main.cpp

```
#include <QApplication>
#include <QDesktopWidget>
#include <QDebug>
#include <QDateTime>
#include <QCommandLineOption>
#include <QCommandLineParser>
```

```
#include "antwin.h"

/*
 *
 * ./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 - ←
 *   s 3 -c 22
 *
 */

int main ( int argc, char *argv[] )
{
    QApplication a ( argc, argv );

    QCommandLineOption szeles_opt ( { "w", "szelesseg" }, "Oszlopok (cellakban ←
        ) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( { "m", "magassag" }, "Sorok (cellakban) ←
        szama.", "magassag", "150" );
    QCommandLineOption hangyaszam_opt ( { "n", "hangyaszam" }, "Hangyak szama. ←
        ", "hangyaszam", "100" );
    QCommandLineOption sebesseg_opt ( { "t", "sebesseg" }, "2 lepes kozotti ←
        ido (millisec-ben).", "sebesseg", "100" );
    QCommandLineOption parolgas_opt ( { "p", "parolgas" }, "A parolgas erteke. ←
        ", "parolgas", "8" );
    QCommandLineOption feromon_opt ( { "f", "feromon" }, "A hagyott nyom ←
        erteke.", "feromon", "11" );
    QCommandLineOption szomszed_opt ( { "s", "szomszed" }, "A hagyott nyom ←
        erteke a szomszedokban.", "szomszed", "3" );
    QCommandLineOption alapertek_opt ( { "d", "alapertek" }, "Indulo ertek a ←
        cellakban.", "alapertek", "1" );
    QCommandLineOption maxcella_opt ( { "a", "maxcella" }, "Cella max erteke." ←
        , "maxcella", "50" );
    QCommandLineOption mincella_opt ( { "i", "mincella" }, "Cella min erteke." ←
        , "mincella", "2" );
    QCommandLineOption cellamerete_opt ( { "c", "cellameret" }, "Hany hangya ←
        fer egy cellaba.", "cellameret", "4" );
    QCommandLineParser parser;

    parser.addHelpOption();
    parser.addVersionOption();
    parser.addOption ( szeles_opt );
    parser.addOption ( magas_opt );
    parser.addOption ( hangyaszam_opt );
    parser.addOption ( sebesseg_opt );
    parser.addOption ( parolgas_opt );
    parser.addOption ( feromon_opt );
    parser.addOption ( szomszed_opt );
    parser.addOption ( alapertek_opt );
    parser.addOption ( maxcella_opt );
```

```
parser.addOption ( mincella_opt );
parser.addOption ( cellamerete_opt );

parser.process ( a );

QString szeles = parser.value ( szeles_opt );
QString magas = parser.value ( magas_opt );
QString n = parser.value ( hangyaszam_opt );
QString t = parser.value ( sebesseg_opt );
QString parolgas = parser.value ( parolgas_opt );
QString feromon = parser.value ( feromon_opt );
QString szomszed = parser.value ( szomszed_opt );
QString alapertek = parser.value ( alapertek_opt );
QString maxcella = parser.value ( maxcella_opt );
QString mincella = parser.value ( mincella_opt );
QString cellameret = parser.value ( cellamerete_opt );

qsrand ( QDateTime::currentMSecsSinceEpoch() );

AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), feromon ←
    .toInt(), szomszed.toInt(), parolgas.toInt(),
    alapertek.toInt(), mincella.toInt(), maxcella.toInt(),
    cellameret.toInt() );

w.show();

return a.exec();
}
```

A main programban ismét sok minden kell beimportálnunk, többek között a QApplication, QDesktopWidget, QDebug, QDateTime, QCommandLineOption, QCommandLineParser, valamint az antwin.h header fájlt.

A main részben példányosítás látható, QApplication néven, ez tartalmazza a konstruktort és annak paramétereit.

A process() függvény arra szolgál, hogy ő fogja feldolgozni az argumentumokat.

A forráskód végéhez közeledve láthatjuk a show() függvényt, aminek az a feladata, hogy megmutassa nekünk az ablakot, és így elénk tárul a hangyaszimuláció.

A legvégén pedig találkozunk az exec() függvényel, ezt adjuk vissza. A programból a q betű lenyomásával tudunk kilépni.

## 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: [Sejtautomata.java](#)

Tanulságok, tapasztalatok, magyarázat...

A forráskód elemzése előtt ejtsünk egy kis szót az életjátékról. A játék lényege az, hogy a játékos megad egy kezdőalakzatot, majd ezt követően annyi dolga van, hogy figyelje az eredményt. Ez a játék matematikailag a sejtautomaták közé tartozik.

A lépések eredményét a játékban a számítógép számolja ki. Érdekességekben a játéknak vannak matematikai és filozófia vonatkozásai. A játékot egyébként John Conway találta ki, az ő nevéhez fűződik a csokrunk is.

A játék lényege: Több kör van a játékban, minden körben változatos események vannak. A lehetőségek: A sejt él, ha pontosan 2 vagy pontosan 3 élő szomszédja van. A sejt elpusztul értelemszerűen, ha 2-nél kevesebb vagy 3-nál több élő szomszédja van. Új sejt akkor születik, ha pontosan 3 sejt található a környezetében.

Nézzük a forráskódot:

```
public class Sejtautomata extends java.awt.Frame implements Runnable {  
    /** Egy sejt lehet élő */  
    public static final boolean ÉLŐ = true;  
    /** vagy halott */  
    public static final boolean HALOTT = false;  
    /** Két rácsot használunk majd, az egyik a sejttér állapotát  
     * a t_n, a másik a t_n+1 időpillanatban jellemzi. */  
    protected boolean[][][] rácok = new boolean[2][][];  
    /** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen  
     * [2][][]-ból az első dimenziót használni, mert vagy az  
     * egyikre  
     * állítjuk, vagy a másikra. */  
    protected boolean[][] rács;  
    /** Megmutatja melyik rács az aktuális: [rácsIndex][][] */  
    protected int rácsIndex = 0;  
    /** Pixelben egy cella adatai. */  
    protected int cellaSzélesség = 20;  
    protected int cellaMagasság = 20;  
    /** A sejttér nagysága, azaz hányszor hány cella van? */  
    protected int szélesség = 20;  
    protected int magasság = 10;
```

Láthatjuk a Sejtautomata osztályt, ami a grafikus megjelenéshez használja a java.awt.Frame-t is. Ebben a részben láthatjuk a sejteknek az állapotait, vagyis azokna a változóit, illetve a rácsot alkotó változót, ami egy boolean, tehát egy logikai változó. A rács adatai után, pixelben megadjuk a cella adatait is változókban (cellaSzélesség, cellaMagasság).

```
addKeyListener(new java.awt.event.KeyAdapter() {  
    // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük  
    public void keyPressed(java.awt.event.KeyEvent e) {
```

```
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
            // Felezük a cella méreteit:
            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            // Duplázzuk a cella méreteit:
            cellaSzélesség *= 2;
            cellaMagasság *= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel = !pillanatfelvétel;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
            várakozás /= 2;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
            várakozás *= 2;
        repaint();
    }
});;

// Egér kattintó események feldolgozása:
addMouseListener(new java.awt.event.MouseAdapter() {
    // Egér kattintással jelöljük ki a nagyítandó területet
    // bal felső sarkát vagy ugyancsak egér kattintással
    // vizsgáljuk egy adott pont iterációt:
    public void mousePressed(java.awt.event.MouseEvent m) {
        // Az egérmutató pozíciója
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});;

// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});;
```

A programunkban fontos az, hogy figyeljük azt ami a billentyűzetről érkezik, mivel a gombokat csak lenyom-

jur mindig másféleképpen kell reagálni a programnak. Az egérnél is hasonló a helyzet, így érdemes őt is figyelemmel kísérni.

```
// Cellaméretek kezdetben
cellaSzélesség = 10;
cellaMagasság = 10;
// Pillanatfelvétel készítéséhez:
try {
    robot = new java.awt.Robot(
        java.awt.GraphicsEnvironment.
        getLocalGraphicsEnvironment().
        getDefaultScreenDevice());
} catch(java.awt.AWTException e) {
    e.printStackTrace();
}

// A program ablakának adatai:
setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
           magasság*cellaMagasság);
setVisible(true);
// A sejttér életrekeltése:
new Thread(this).start();
}
/** A sejttér kirajzolása. */
public void paint(java.awt.Graphics g) {
    // Az aktuális
    boolean [][] rács = rácsok[rácsIndex];
    // rácsot rajzoljuk ki:
    for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon
        for(int j=0; j<rács[0].length; ++j) { // s az oszlopok
            // Sejt cella kirajzolása
            if(rács[i][j] == ÉLŐ)
                g.setColor(java.awt.Color.BLACK);
            else
                g.setColor(java.awt.Color.WHITE);
            g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                       cellaSzélesség, cellaMagasság);
            // Rács kirajzolása
            g.setColor(java.awt.Color.LIGHT_GRAY);
            g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                       cellaSzélesség, cellaMagasság);
        }
    }
}
```

Ezt követően megadjuk a mérteteket a cellának, valamint megadjuk az albaknak is azt amik rá vonatkoznak,

gondolok itt például a méretre és a címére. A paint () függvényel meg rajzoljuk azt a teret ahol a sejtek lesznek ehhez kell a java.awt.Graphics G osztály. Ha ez megvan akkor elkezdődik a rajzolás, amikhez fekete és fehér színt fog használni a programunk.

```
// Készítünk pillanatfelvételt?  
if(pillanatfelvétel) {  
    // a biztonság kedvéért egy kép készítése után  
    // kikapcsoljuk a pillanatfelvételt, hogy a  
    // programmal ismerkedő Olvasó ne írja tele a  
    // fájlrendszerét a pillanatfelvétellekkel  
    pillanatfelvétel = false;  
    pillanatfelvétel(robot.createScreenCapture  
        (new java.awt.Rectangle  
            (getLocation().x, getLocation().y,  
            szélesség*cellaSzélesség,  
            magasság*cellaMagasság)));  
}  
}
```

Végül pedig megnézzük, hogy készített e pillanatfelvételt?

Nézzük a szomszédokat.

```
public int szomszédokSzáma(boolean [][] rács,  
    int sor, int oszlop, boolean állapot) {  
    int állapotúSzomszéd = 0;  
    // A nyolcszomszédok végigzongorázása:  
    for(int i=-1; i<2; ++i)  
        for(int j=-1; j<2; ++j)  
            // A vizsgált sejtet magát kihagyva:  
            if(!((i==0) && (j==0))) {  
                // A sejttéről szélénak szomszédai  
                // a szembe oldalakon ("periódikus határfeltétel")  
                int o = oszlop + j;  
                if(o < 0)  
                    o = szélesség-1;  
                else if(o >= szélesség)  
                    o = 0;  
  
                int s = sor + i;  
                if(s < 0)  
                    s = magasság-1;  
                else if(s >= magasság)  
                    s = 0;  
  
                if(rács[s][o] == állapot)  
                    ++állapotúSzomszéd;  
            }  
}
```

```
    return állapotúSzomszéd;
}
```

A nyolcszomszédokon végigmenve, ellenőrizzük azt, hogy hány szomszédja lesz a sejtünknek, ami azért fontos, hogy a sejtünknek mi lesz a sorsa élni fog-e vagy sem, megszületik-e vagy nem. Végezetül, pedig ez a függvény visszaadja a szomszédok számát.

```
public void sikló(boolean [][] rács, int x, int y) {

    rács[y+ 0][x+ 2] = ÉLŐ;
    rács[y+ 1][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 2] = ÉLŐ;
    rács[y+ 2][x+ 3] = ÉLŐ;

}

public void siklóKilövő(boolean [][] rács, int x, int y) {

    rács[y+ 6][x+ 0] = ÉLŐ;
    rács[y+ 6][x+ 1] = ÉLŐ;
    rács[y+ 7][x+ 0] = ÉLŐ;
    rács[y+ 7][x+ 1] = ÉLŐ;

    rács[y+ 3][x+ 13] = ÉLŐ;

    rács[y+ 4][x+ 12] = ÉLŐ;
    rács[y+ 4][x+ 14] = ÉLŐ;

    rács[y+ 5][x+ 11] = ÉLŐ;
    rács[y+ 5][x+ 15] = ÉLŐ;
    rács[y+ 5][x+ 16] = ÉLŐ;
    rács[y+ 5][x+ 25] = ÉLŐ;

    rács[y+ 6][x+ 11] = ÉLŐ;
    rács[y+ 6][x+ 15] = ÉLŐ;
    rács[y+ 6][x+ 16] = ÉLŐ;
    rács[y+ 6][x+ 22] = ÉLŐ;
    rács[y+ 6][x+ 23] = ÉLŐ;
    rács[y+ 6][x+ 24] = ÉLŐ;
    rács[y+ 6][x+ 25] = ÉLŐ;

    rács[y+ 7][x+ 11] = ÉLŐ;
    rács[y+ 7][x+ 15] = ÉLŐ;
    rács[y+ 7][x+ 16] = ÉLŐ;
    rács[y+ 7][x+ 21] = ÉLŐ;
    rács[y+ 7][x+ 22] = ÉLŐ;
    rács[y+ 7][x+ 23] = ÉLŐ;
    rács[y+ 7][x+ 24] = ÉLŐ;
```

```
rács[y+ 8][x+ 12] = ÉLŐ;
rács[y+ 8][x+ 14] = ÉLŐ;
rács[y+ 8][x+ 21] = ÉLŐ;
rács[y+ 8][x+ 24] = ÉLŐ;
rács[y+ 8][x+ 34] = ÉLŐ;
rács[y+ 8][x+ 35] = ÉLŐ;

rács[y+ 9][x+ 13] = ÉLŐ;
rács[y+ 9][x+ 21] = ÉLŐ;
rács[y+ 9][x+ 22] = ÉLŐ;
rács[y+ 9][x+ 23] = ÉLŐ;
rács[y+ 9][x+ 24] = ÉLŐ;
rács[y+ 9][x+ 34] = ÉLŐ;
rács[y+ 9][x+ 35] = ÉLŐ;

rács[y+ 10][x+ 22] = ÉLŐ;
rács[y+ 10][x+ 23] = ÉLŐ;
rács[y+ 10][x+ 24] = ÉLŐ;
rács[y+ 10][x+ 25] = ÉLŐ;

rács[y+ 11][x+ 25] = ÉLŐ;

}
```

Láthatunk 2 funcitont a sikló-t és a siklóKilövőt, őket betesszük a sejt térbe, ők minden esetben élők.

```
/** Pillanatfelvételek készítése. */
public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    // A pillanatfelvétel kép fájlneve
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételszámláló);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
    paint(g);
}
/**
 * Példányosít egy Conway-féle életjáték szabályos
```

```
* sejttér obektumot.  
*/
```

Forráskódunk végéhez közeledve szembesülünk a pillanatfelvétel () függvényel, ami abban az esetben, ha meghívjuk akkor láthatjuk, hogy egy képet hoz létre. Az itt elkészült képállományoknak a számát eltárolja egy változóban.

Nézzük a programunk legvégét, a maint.

```
public static void main(String[] args) {  
    // 100 oszlop, 75 sor mérettel:  
    new Sejtautomata(100, 75);  
}
```

A main részhez érkeztünk, ahol egy példányosítást láthatunk, amiben létrejön egy objektum melynek paraméterei 100 és 75. A 100 az oszlopot jelenti a 75 a sornak a méretét. Ezek a paraméterek az ablakunk méretét határozzák meg.

### 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

[sejtszal.h](#)

[sejtszal.cpp](#)

[sejtablak.h](#)

[sejtablak.cpp](#)

[main.cpp](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat megegyezik az előző feladattal amit nekünk kell tenni, az az, hogy a Java forráskódot “varázsoljuk” át C++-ra.

Ebben a feladatban 2 fájl forráskódját elemezzük, ezek a fájlok: sejtszal.h, setablak.h, sejtszal.cpp, sejtablak.cpp és végül a main.cpp

[sejtszal.h](#)

```
#ifndef SEJTSZAL_H  
#define SEJTSZAL_H  
  
#include <QThread>  
#include "sejtablak.h"  
  
class SejtAblak;
```

```
class SejtSzal : public QThread
{
    Q_OBJECT

public:
    SejtSzal(bool ***racsok, int szelesseg, int magassag,
              int varakozas, SejtAblak *sejtAblak);
    ~SejtSzal();
    void run();

protected:
    bool ***racsok;
    int szelesseg, magassag;
    // Megmutatja melyik rcs az aktulis: [rcsIndex] []
    int racsIndex;
    // A sejttr kt egymst kvet t_n s t_n+1 diszkrt idpillanata
    // kztti vals id.
    int varakozas;
    void idoFejlodes();
    int szomszedokSzama(bool **racs,
                         int sor, int oszlop, bool allapot);
    SejtAblak* sejtAblak;

};

#endif // SEJTSZAL_H
```

A sejtszal.h fájlunk egy header fájl, nézzük mit találhatunk benne.

Beimportáljuk a szükséges QThread osztályt, valamint hivatkozunk a sejtabalk.h fájusra.

Ezt követően létrehozásra kerül a Sejtblak osztály, valamint a SejtSzal osztály, ami a QTheard kiterjeszésével valósul meg.

A publikus részben láthatjuk a konstruktort, és a destruktort is, végül pedig található benne egy run() függvény is.

A protectedben találjuk a változókat, szelesseg, magassag, a racsIndex-et ezzel azt tudjuk meg, hogy melyik az aktuális rács, van még egy varakozas változónk is. Ugyanitt láthatunk még egy idoFejlodes() függvényt is, illetve a szomedokSzama() függvény is ebben a fájlban található. A végén a SejtAblak-ra mutató mutató van.

### sejtszal.cpp

```
#include "sejtszal.h"

SejtSzal::SejtSzal(bool ***racsok, int szelesseg, int magassag, int ←
                    varakozas, SejtAblak *sejtAblak)
{
    this->racsok = racsok;
    this->szelesseg = szelesseg;
```

```
this->magassag = magassag;
this->varakozas = varakozas;
this->sejtAblak = sejtAblak;

    racsIndex = 0;
}

/***
 * Az krdezett llapotban lv nyolcszomszdok szma.
 *
 * @param rcs a sejttr rcs
 * @param sor a rcs vizsglt sora
 * @param oszlop a rcs vizsglt oszlopa
 * @param llapor a nyolcszomszdok vizsglt llapota
 * @return int a krdezett llapotbeli nyolcszomszdok szma.
 */
int SejtSzal::szomszedokSzama(bool **racs,
                                int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
    // A nyolcszomszdok vgigzongorza:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsglt sejtet magt kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejtrabl szlnek szomszdai
                // a szembe oldalakon ("peridikus hatrfelttel")
                int o = oszlop + j;
                if(o < 0)
                    o = szelesseg-1;
                else if(o >= szelesseg)
                    o = 0;

                int s = sor + i;
                if(s < 0)
                    s = magassag-1;
                else if(s >= magassag)
                    s = 0;

                if(racs[s][o] == allapot)
                    ++allapotuSzomszed;
            }
    return allapotuSzomszed;
}

/***
 * A sejttr idbeli fejldse a John H. Conway fle
 * letjtk sejtautomata szablyai alapjn trtnik.
 * A szablyok rszletes ismertetst lsd pldul a
 * [MATEK JTK] hivatkozban (Cskny Bla: Diszkrt
```

```
* matematikai jtkok. Polygon, Szeged 1998. 171. oldal.)  
*/  
void SejtSzal::idoFejlodes() {  
  
    bool **racsElotte = racsok[racsIndex];  
    bool **racsUtana = racsok[(racsIndex+1)%2];  
  
    for(int i=0; i<magassag; ++i) { // sorok  
        for(int j=0; j<szelesség; ++j) { // oszlopok  
  
            int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);  
  
            if(racsElotte[i][j] == SejtAblak::ELO) {  
                /* 1 l marad, ha kett vagy hrom l  
                szomszedja van, klnben halott lesz. */  
                if(elok==2 || elok==3)  
                    racsUtana[i][j] = SejtAblak::ELO;  
                else  
                    racsUtana[i][j] = SejtAblak::HALOTT;  
            } else {  
                /* Halott halott marad, ha hrom l  
                szomszedja van, klnben l lesz. */  
                if(elok==3)  
                    racsUtana[i][j] = SejtAblak::ELO;  
                else  
                    racsUtana[i][j] = SejtAblak::HALOTT;  
            }  
        }  
    }  
    racsIndex = (racsIndex+1)%2;  
}  
  
/** A sejttr idbeli fejldse. */  
void SejtSzal::run()  
{  
    while(true) {  
        QThread::msleep(varakozas);  
        idoFejlodes();  
        sejtAblak->vissza(racsIndex);  
    }  
}  
  
SejtSzal::~SejtSzal()  
{  
}
```

Elsősorban hivatkozunk a sejtszal.h fájlra, amit az előbb már átbeszéltünk.

A konstruktorban nem történik tulajdonképpen semmi érdekes, beállítjuk a racsIndexet 0-ra.

Nézzük mi történik a szomszedokSzama() függvényben, visszaadja az elemeknek a számát, a szomszéd-ságból, tehát abból a 8-ból ami körülötte van.

Figyeljuk meg az idoFejlodes() függvényt, kialakítja a pillanatot a rácsba, majd végigmegy a sejtekben és megvizsgálja sz összesét. Mivel szabályok vannak, ezért ezeket figyelembe veszi, tehát akkor lesz élő a sejt ha pontosan 2 vagy 3 szomszédja van.

Ha ezzel végzett akkor a racsIndexet lépteti.

sejtablak.h

```
#ifndef SEJTABLAK_H
#define SEJTABLAK_H

#include <QMainWindow>
#include <QPainter>
#include "sejtszal.h"

class SejtSzal;

class SejtAblak : public QMainWindow
{
    Q_OBJECT

public:
    SejtAblak(int szelesseg = 100, int magassag = 75, QWidget *parent = 0);

    ~SejtAblak();
    // Egy sejt lehet 1
    static const bool ELO = true;
    // vagy halott
    static const bool HALOTT = false;
    void vissza(int racsIndex);

protected:
    // Kt rcsot hasznunk majd, az egyik a sejtrr llapott
    // a t_n, a msik a t_n+1 idpillanatban jellemzi.
    bool ***racsok;
    // Valamelyik rcsra mutat, technikai jelleg, hogy ne kelljen a
    // [2][][]-bl az els dimenzit hasznlni, mert vagy az egyikre
    // lltjuk, vagy a msikra.
    bool **racs;
    // Megmutatja melyik rcs az aktulis: [rcsIndex][][][]
    int racsIndex;
    // Pixelben egy cella adatai.
    int cellaSzelesseg;
    int cellaMagassag;
    // A sejtrr nagysga, azaz hnyszor hny cella van?
    int szelesseg;
    int magassag;
    void paintEvent(QPaintEvent*);
```

```
void siklo(bool **racs, int x, int y);
void sikloKilovo(bool **racs, int x, int y);

private:
    SejtSzal* eletjatek;

};

#endif // SEJTABLAK_H
```

A fájlunk egy header fáj mint ahogyan a sejtszal.h is az volt. Beimportáljuk a szükséges osztályokat, a QMainWindow-t és a QPainter-t, valamint hivatkozunk a sejtszal.h fájlra.

Létrehozzuk a sejtAblak osztályt, aminek kiterjesztése a QMainWindow. A publikus részben találjuk a konstruktort valamint a destruktort is. Rajtuk kívül látunk még 2 tagot az egyikben az élő sejtet adjuk meg ez az ( ELO = true ), a másikban pedig értelemszerűen a halott sejtet adjuk meg ( HALOTT = false ).

Protected részben látjuk a rácsokat, valamint a cellaSzelesseg, cellaMagassag, szelesseg, magassag változókat. Továbbá láthatunk még 3 függvényt, a paintEvent(), siklo() és a sikloKilovo().

sejtablak.cpp

```
#include "sejtablak.h"

SejtAblak::SejtAblak(int szelesseg, int magassag, QWidget *parent)
: QMainWindow(parent)
{
    setWindowTitle("A John Horton Conway-fle letjtk");

    this->magassag = magassag;
    this->szelesseg = szelesseg;

    cellaSzelesseg = 6;
    cellaMagassag = 6;

    setFixedSize(QSize(szelesseg*cellaSzelesseg, magassag*cellaMagassag));

    racsok = new bool**[2];
    racsok[0] = new bool*[magassag];
    for(int i=0; i<magassag; ++i)
        racsok[0][i] = new bool [szelesseg];
    racsok[1] = new bool*[magassag];
    for(int i=0; i<magassag; ++i)
        racsok[1][i] = new bool [szelesseg];

    racsIndex = 0;
    racs = racsok[racsIndex];

    // A kiindul racs minden cellja HALOTT
    for(int i=0; i<magassag; ++i)
```

```
for(int j=0; j<szelesseg; ++j)
    racs[i][j] = HALOTT;
// A kiindul racsra "ELOlnyeket" helyeznk
//siklo(racs, 2, 2);

sikloKilovo(racs, 5, 60);

eletjatek = new SejtSzal(racsok, szelesseg, magassag, 120, this);
eletjatek->start();
}

void SejtAblak::paintEvent(QPaintEvent*)
{
QPainter qpainter(this);

// Az aktulis
bool **racs = racsok[racsIndex];
// racsot rajzoljuk ki:
for(int i=0; i<magassag; ++i) { // végig lpked a sorokon
    for(int j=0; j<szelesseg; ++j) { // s az oszlopok
        // Sejt cella kirajzolsa
        if(racs[i][j] == ELO)
            qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                               cellaSzelesseg, cellaMagassag, Qt::black);
        else
            qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                               cellaSzelesseg, cellaMagassag, Qt::white);
        qpainter.setPen(QPen(Qt::gray, 1));
    }
}
qpainter.end();
}

SejtAblak::~SejtAblak()
{
    delete eletjatek;

    for(int i=0; i<magassag; ++i) {
        delete[] racsok[0][i];
        delete[] racsok[1][i];
    }

    delete[] racsok[0];
    delete[] racsok[1];
```

```
delete[] racsok;

}

void SejtAblak::vissza(int racsIndex)
{
    this->racsIndex = racsIndex;
    update();
}

/***
 * A sejttrbe "ELOlnyeket" helyeznk, ez a "sikl".
 * Adott irnyban halad, msolja magt a sejttrben.
 * Az ELOlny ismertetst lsd pldul a
 * [MATEK JTK] hivatkozsban (Cskny Bla: Diszkrt
 * matematikai jtkok. Polygon, Szeged 1998. 172. oldal.)
 *
 * @param    racs    a sejttr ahov ezt az llatkt helyezzk
 * @param    x        a befoglal tglal bal fels sarknak oszlopa
 * @param    y        a befoglal tglal bal fels sarknak sora
 */
void SejtAblak::siklo(bool **racs, int x, int y) {

    racs[y+ 0][x+ 2] = ELO;
    racs[y+ 1][x+ 1] = ELO;
    racs[y+ 2][x+ 1] = ELO;
    racs[y+ 2][x+ 2] = ELO;
    racs[y+ 2][x+ 3] = ELO;

}

/***
 * A sejttrbe "ELOlnyeket" helyeznk, ez a "sikl gy".
 * Adott irnyban siklkat l ki.
 * Az ELOlny ismertetst lsd pldul a
 * [MATEK JTK] hivatkozsban /Cskny Bla: Diszkrt
 * matematikai jtkok. Polygon, Szeged 1998. 173. oldal./,
 * de itt az bra hibs, egy oszloppal told mg balra a
 * bal oldali 4 sejtes ngyzetet. A helyes gy rajzt
 * lsd pl. az [LET CIKK] hivatkozsban /Robert T.
 * Wainwright: Life is Universal./ (Megemlhetjk, hogy
 * minden tartalmaz kt felesleges sejtet is.)
 *
 * @param    racs    a sejttr ahov ezt az llatkt helyezzk
 * @param    x        a befoglal tglal bal fels sarknak oszlopa
 * @param    y        a befoglal tglal bal fels sarknak sora
 */
void SejtAblak::sikloKilovo(bool **racs, int x, int y) {

    racs[y+ 6][x+ 0] = ELO;
```

```
racs[y+ 6][x+ 1] = ELO;
racs[y+ 7][x+ 0] = ELO;
racs[y+ 7][x+ 1] = ELO;

racs[y+ 3][x+ 13] = ELO;

racs[y+ 4][x+ 12] = ELO;
racs[y+ 4][x+ 14] = ELO;

racs[y+ 5][x+ 11] = ELO;
racs[y+ 5][x+ 15] = ELO;
racs[y+ 5][x+ 16] = ELO;
racs[y+ 5][x+ 25] = ELO;

racs[y+ 6][x+ 11] = ELO;
racs[y+ 6][x+ 15] = ELO;
racs[y+ 6][x+ 16] = ELO;
racs[y+ 6][x+ 22] = ELO;
racs[y+ 6][x+ 23] = ELO;
racs[y+ 6][x+ 24] = ELO;
racs[y+ 6][x+ 25] = ELO;

racs[y+ 7][x+ 11] = ELO;
racs[y+ 7][x+ 15] = ELO;
racs[y+ 7][x+ 16] = ELO;
racs[y+ 7][x+ 21] = ELO;
racs[y+ 7][x+ 22] = ELO;
racs[y+ 7][x+ 23] = ELO;
racs[y+ 7][x+ 24] = ELO;

racs[y+ 8][x+ 12] = ELO;
racs[y+ 8][x+ 14] = ELO;
racs[y+ 8][x+ 21] = ELO;
racs[y+ 8][x+ 24] = ELO;
racs[y+ 8][x+ 34] = ELO;
racs[y+ 8][x+ 35] = ELO;

racs[y+ 9][x+ 13] = ELO;
racs[y+ 9][x+ 21] = ELO;
racs[y+ 9][x+ 22] = ELO;
racs[y+ 9][x+ 23] = ELO;
racs[y+ 9][x+ 24] = ELO;
racs[y+ 9][x+ 34] = ELO;
racs[y+ 9][x+ 35] = ELO;

racs[y+ 10][x+ 22] = ELO;
racs[y+ 10][x+ 23] = ELO;
racs[y+ 10][x+ 24] = ELO;
racs[y+ 10][x+ 25] = ELO;
```

```
    racs[y+ 11][x+ 25] = ELO;  
}
```

A forráskód elején hivatkozunk a sejtblak.h állományukra.

Nézzük a konstruktort, a lényegi dogol tulajdonképpen itt történnek majd. Itt kerülnek beállításra a szélesség és magasság változók. A celláknak a mérete fixen van megadva, valamint beállításra kerül az ablak mérete, ezt le is fixáljuk.

Meghívásra kerül a sikloKilovo() függvény ezzel rajzolhatunk.

Forrásunkban a SejtSzalt példányosítjuk, megadjuk neki a paramétereket, ezt követően meghívjuk, ezáltal létrejön a start() függvény, és Ő fogja meghívni a run() függvényt, így elundul a játékunk.

A paintEvent() függvény végighalad a rácson, majd kirajzolja az összes sejtet, színezésnél fontos az, ha a sejt él, akkor fekete a színezés, viszont ha a sejt halott, akkor a színezés fehér.

A destrukturban tulajdonképpen törlésre kerül sor.

A forráskódunk végén a siklo() és a sikliKilovo() függvények létrehozzák az élőlényeket, amik szerepelnek bennük.

main.cpp

Nézzük is a főprogramunkat:

```
#include <QApplication>  
#include "sejtblak.h"  
#include <QDesktopWidget>  
  
int main(int argc, char *argv[])  
{  
    QApplication a(argc, argv);  
    SejtAblak w(100, 75);  
    w.show();  
  
    return a.exec();  
}
```

Beimportáljuk a szükséges osztályokat, a QApplication valamint a QDesktopWidget-t, ezenkívül még hivatkozunk a sejtblak.h állományra.

A main részben példányosítás történik mégpedig a QApplication-t példányosítjuk melynek odaadjuk az argumentumokat. Példányosítjuk továbbá a SejtAblakot is, neki oadunk a méreteket. (100 jelöli az oszlopot, 75 a sort.)

A show() függvényel megtekinthetjük a játékunkat, majd a forráskód végén a visszatérési érték az exec() lesz, ha bezárjuk az ablakot akkor tér majd vissza, és az értéke 0 lesz.

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

[BrainBThread.h](#)

[BrainBThread.cpp](#)

[BrainBWin.h](#)

[BrainBWin.cpp](#)

[main.cpp](#)

Tanulságok, tapasztalatok, magyarázat...

A BrainB egy projekt, melynek az lenne a célja, lényege, hogy a jövendőbelé esportoló játékosokat feltalálni, megkeresni. Ahhoz ezeket a játékosokat megtaláljuk, ehhez készült ez a program, amely az agynak a kognitív képességét figyeli, méri fel.

A feladatunk nekünk játékosoknak, hogy van egy felirat a Samu Entropy karakteren kell tartanunk az egérnek a mutatóját, azonban ez egyre nehezebb, mivel újabb és újabb karakterek jelennek meg a kijelzőnk. Ezek ismeretében nézzük meg mik is történnek az adott fájlokban:

[BrainBWin.h](#)

Ebben a header fájlban találhatjuk a konstruktort, a destruktort és a bemenetet észlelő függvényeket. Ezek azok a függvények, melyek észlelik azokat a bemeneteket amik billentyűzetről, vagy egéről érkeznek. Megtalálható még ebben a fájlban azt a függvényt is ami azért felel, hogy megjelenítse számunkra a tartalmat.

[BrainBWin.cpp](#)

Ebben a fájlban tulajdonképpen az előzőleg ismertetett header fájlt dekrarálja, és definiálja, a változókat, illetve a különböző függvényeket is.

[BrainBThread.h](#)

Ebben a fájlban dekraláljuk a szükséges oszzályokat, a Thread osztályban történik a lényeg. A forráskódunkban megadjuk a Hero osztályt, továbbá a BrainBThread osztályt is létrehozzuk. Megalkotjuk továbbá a hősök vektorát, illetve adunk még meg pár változót. Ebben a fájlban többek között olyan függvényeket is használunk mint például a getT() ami visszaadja a time-ot, illetve a finish() függvény, ő pedig azt elleőrzi, hogy van-e még időnk, nem járt-e le.

[BrainBThread.cpp](#)

```
BrainBThread::BrainBThread ( int w, int h )
{
    dispShift = heroRectSize+heroRectSize/2;

    this->w = w - 3 * heroRectSize;
    this->h = h - 3 * heroRectSize;

    std::srand ( std::time ( 0 ) );

    Hero me ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100,
              this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 9 );
```

```
Hero other1 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
    ) - 100,
    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        5, "Norbi Entropy" );
Hero other2 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
    ) - 100,
    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        3, "Greta Entropy" );
Hero other4 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
    ) - 100,
    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        5, "Nandi Entropy" );
Hero other5 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
    ) - 100,
    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        7, "Matyi Entropy" );

heroes.push_back ( me );
heroes.push_back ( other1 );
heroes.push_back ( other2 );
heroes.push_back ( other4 );
heroes.push_back ( other5 );

}
```

Ez a fájl hivatkozik a már előzőleg megismert BrainBThread.h fájlra. Ebben a fájlban megalkotjuk Samu-t illetve hozzáadunk még 4 “hőst” és hozzáadjuk őket a heroes vektorához. Így megalakulnak a hősök.

```
void BrainBThread::pause()
{
    paused = !paused;
    if ( paused ) {
        ++nofPaused;
    }
}

void BrainBThread::set_paused ( bool p )
{
    if ( !paused && p ) {
        ++nofPaused;
    }
}
```

```
    paused = p;  
}
```

A program további részében a desktruktort vizsgáljuk illetve megismerkedünk különböző függvényekkel ilyen például a set\_paused() függvény.

Végezetül pedig nézzük meg egy kicsit részletesebben a main részét a programnak.

```
#include <QApplication>  
#include <QTextStream>  
#include <QtWidgets>  
#include "BrainBWin.h"  
  
int main ( int argc, char **argv )  
{  
    QApplication app ( argc, argv );
```

A main résznek az elején, már a szokásosnak mondható importálásokat végezzük. Nézzük mit/miket is kell beimportálnunk: QApplication, QTextStream, QtWidgets és végül az utoljára hivatkozunk a BrainBWin.h fájlra.

A main-ben deklarálunk egy app-ot melynek típusa QApplication, ez egy objektum, és odaadjuk neki a paraméterlistát.

```
QTextStream qout ( stdout );  
    qout.setCodec ( "UTF-8" );  
  
    qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( " ↵  
        Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;  
  
    qout << "This program is free software: you can redistribute it and ↵  
        /or modify it under" << endl;  
    qout << "the terms of the GNU General Public License as published ↵  
        by the Free Software" << endl;  
    qout << "Foundation, either version 3 of the License, or (at your ↵  
        option) any later" << endl;  
    qout << "version.\n" << endl;  
  
    qout << "This program is distributed in the hope that it will be ↵  
        useful, but WITHOUT" << endl;  
    qout << "ANY WARRANTY; without even the implied warranty of ↵  
        MERCHANTABILITY or FITNESS" << endl;  
    qout << "FOR A PARTICULAR PURPOSE. See the GNU General Public ↵  
        License for more details.\n" << endl;  
  
    qout << QString::fromUtf8 ( "Ez a program szabad szoftver; ↵  
        terjeszthető illetve módosítható a Free Software" ) << endl;  
    qout << QString::fromUtf8 ( "Foundation által kiadott GNU General ↵  
        Public License dokumentumában leírtak;" ) << endl;
```

```
qout << QString::fromUtf8 ( "akár a licenc 3-as, akár (tetszőleges) ←
    későbbi változata szerint.\n" ) << endl;

qout << QString::fromUtf8 ( "Ez a program abban a reményben kerül ←
    közreadásra, hogy hasznos lesz, de minden" ) << endl;
qout << QString::fromUtf8 ( "egyéb GARANCIA NÉLKÜL, az ←
    ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ" ) << endl;
qout << QString::fromUtf8 ( "ALKALMAZHATÓSÁGRA való származtatott ←
    garanciát is beleértve. További" ) << endl;
qout << QString::fromUtf8 ( "részleteket a GNU General Public ←
    License tartalmaz.\n" ) << endl;

qout << "http://gnu.hu/gplv3.html" << endl;

QRect rect = QApplication::desktop()->availableGeometry();
BrainBWin brainBWin ( rect.width(), rect.height() );
brainBWin.setWindowState ( brainBWin.windowState() ^ Qt::←
    WindowFullScreen );
brainBWin.show();
return app.exec();
```

Ezt követően a qout-ot fogjuk hozzuk létre, ő a standart outpatra fog kerülni. Beállítjuk neki, hogy az írás az UTF-8 legyen, majd kiíratjuk vele az információkat.

Miután a kiíratások megtörténtek, látunk egy objektumot, ennek a neve Qrect, ahol egy téglalapot fogunk létrehozni, pontosan akkorát, hogy az befedje az egész területet az asztalon. Ezt követően példányosítás történik mégpedig a BrainBWin osztályt fogjuk példányosítani.

A forráskódunk végén beállítjuk, hogy a megjelnő ablakunk full screen-ben jelenjen meg, és a show függvényel elérjük azt, hogy ez látható is legyen.

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

### 8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndl8>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9. fejezet

# Helló, Chaitin!

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

Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

Megoldás forrása:

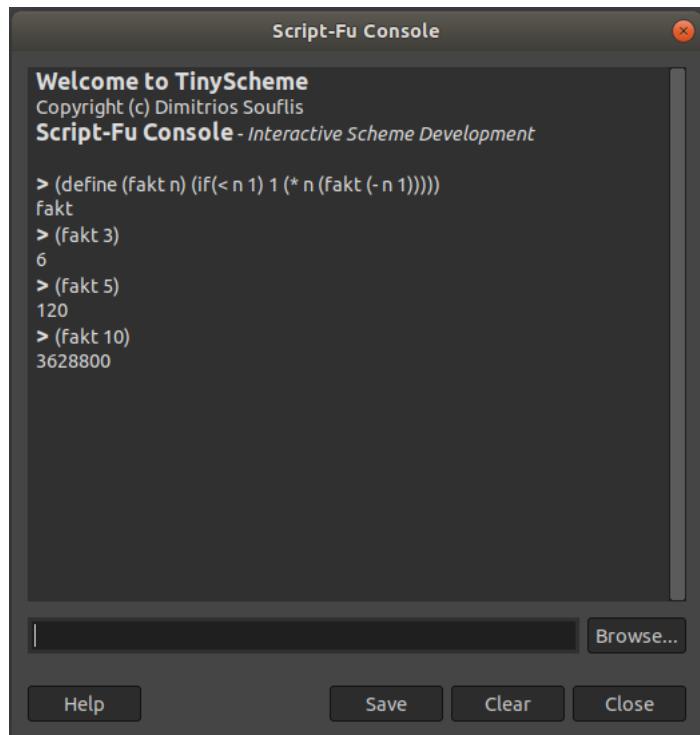
Mi is az a Lips? A Lips egy programozási nyelv, ami már elég régi, ám még ma is használható nyelv. Érdekkességeként 1958-ban jelent meg ez a nyelv, elsősorban a meseterséges intelligencia kedvelőinek okozott hatalmas öröömöt.

Most nézzük, hogyan is tudjuk őt “előcsalni”. Szükségünk lesz a GIMP-nevű alkalmazásra, ha ezzel nem rendelkezünk akkor telepítenünk kell. A GIMP-en belül a szűrők között találjuk, azon belül us a Script-Fu fájlba kell keresni, és az ott található Conosle-val tudjuk elindítani.

Miután megnyitottuk, nagyon fontos információ, hogy azokat a műveleteket, amiket el szeretnénk vele végeztetni, például egy összeadásnál nem szokványos módon kell megadni, hogy  $5 + 5$ , hanem úgy, hogy  $+ 5 5$ . Fontos hogy ezeket zárójelek közé írjuk, tehát a helyes megoldás a  $(+ 5 5)$ . Ha ezt tisztáztuk, akkor nézzük is a számunkra érdekes faktoriálist.

A függvényt amit használni szeretnénk a faktoriális esetén, a define szóval kell bevezetnünk, és adunk neki egy nevet jelen esetben a fakt nevet kapta a függvényünk. Az n az egy paraméter, az n helyére kell majd írnunk azt a számot amelyik faktoriálisra kívánccsak vagyunk. Ezt követően az if-el feltételt vizsgálunk, ha ez a feltétel sikerül, akkor visszakapjuk az első értéket, ha pedig nem sikerülne, akkor azt, ami utána következik, esetünkben ez az n paraméter a fakt fügvénnyel. Ezzel észrevehetjük azt, hogy a beírt kódunk az rekurzív is mert meghívja önmagát, és iteratív is mivel az amit végrehajt az megismétlődik.

Most pedig nézzük meg, hogy hogyan is működik



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

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

Megoldás videó: [https://youtu.be/OKdAkI\\_c7Sc](https://youtu.be/OKdAkI_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)

Tanulságok, tapasztalatok, magyarázat...

## 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\\_hackelese\\_a\\_scheme\\_programozasi\\_nyelv](https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_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)

Tanulságok, tapasztalatok, magyarázat...

# 10. fejezet

## Helló, Gutenberg!

### 10.1. Juhász István: Magas szintű programozási nyelvek 1

#### 1.2 Alapfogalmak

A programozási nyelvnek 3 szintje van: ismerjük a gépi nyelvet, a magas szintű nyelvet, valamint az assembly nyelvet is.

Ebben a könyvben leginkább a magas szintű programozásról van szó, ezekből láthatunk forrásokat.

Számomra érdekes volt, hogy egy program amit magas szintű programnyelven van megírva, abból a kódóból lehet tárgyprogramot készíteni. Ezekhez különböző lépésekkel kell teljesítenünk, de ha ezeket végigzongorázunk, akkor kapunk egy gépi kódot, és ebből alakul ki egy olyan kód amit már lehet futtatni is. Fontos lehet még, hogy a programozási nyelvek 2 csoportra osztjuk, imperatív valamint dekleratív.

#### 1.4 A jegyzetben alkalmazott formális jelölésrendszer

Azt tudatja velünk, hogy milyen jelölésrendszert használ a könyv.

#### 2.4 Adattípusok

A nyelvek közül ismerünk kettőt vannak olyanok amelyek típusosak, és vannak olyanok amelyek nem. Alapvetően egy adattípust 3 részre lehet osztani. Ez a 3 rész a tartomány, a műveletek, és a reprezentáció.

Adattípusoknak 2 “fő” csoportja van, vannak az egyszerű típusok, ō lehet karakteres, logikai, egész és valós is.

Az összetett típusban igazán két érdekes van az egyik a rekord, a másik pedig a tömb.

#### 2.5 A nevesített konstans

Fontos tudni azt, hogy a konstansoknak 3 komponensük van. Ezek pedig a név, a típus, és az érték. Igazából ez azért jó és hasznos, mert ha meg akarunk jeleníteni egy értéket több helyen is akkor ezt, ilyen formában meg tudjuk tenni.

#### 2.6 A változó

A konstanshoz hasonlóan, a változónak is vannak komponensei, egész pontosan a változónak 4 komponense van: Ez a 4 pedig a név, a cím, az attribútumok, és az érték. A névvel a változóra szoktunk hivatkozni.

Attribútumok közül érdemes megjegyezni a típust. Az attribútumot a forráskódban deklaráció formájában adjuk meg.

A cím lényegében tudatja velünk azt, hogy az érték hol található, hol a helye. Az érték komponenssel pedig értéket adunk meg, ezt megtehetjük egy művelettel is akár, illetve ezt meg is kaphatja programozásban gyakori értékkedés a kezdőértékkedés.

## 10.2. Kernigan-Ritchie: A C programozási nyelv

### Típusok

Mivel ebben a könyvben a C nyelvről van szó, ezért egy kis szót ejtenék 1-2 adattípusról, melyek nagyon alapvetőek a programozásban, és szinte minden forráskódban találkozhatunk valamelyikükkel.

Nézzük is kik ők: a char, melynek a mérete nem túl nagy, egész pontosan 1 byte, ő karakterkészletnek egy eleme található benne.

A következő adattípus igen gyakori szinte minden forráskódban megtalálható, ő az int, emiről azt érdemes tudni, hogy ő egy egész szám.

Még 2 darab adattípusunk maradt, az egyik a float, ő egy lebegőpontos szám, ami annyiban különbözik az int-től, hogy ő egyszeres pontosságban tehát 1 tizedes jeggyel kezeli a számokat.

A végére maradt a double a double is lebegőpontos szám hasonlóan a floathoz, annyi különbséggel, hogy ő kétszeres pontossággal kezeli a számokat ( 2 tizedes jegy).

### Vezérlési szerkezetek

Ezek az utasítások különböző műveletek sorrendjét fogják meghatározni. Különböző utasításokat ismerünk. Az egyik leggyakoribb, ami nagyon sok forráskódban előfordul, az az if-else utasítást. Itt egy választás, döntés elé állítjuk a felhasználót. Az if helyes használata ha nyitunk utána egy zárójelet és abba írjuk be a kifejezést, ami ha teljesül, akkor további utasításokra lehet számítani.

Az else utasítás ez nem szükséges, vagyis el lehet hagyni, ezt legfőképp akkor használjuk, hogy ha mindenféleképpen szeretnénk valami utasítást adni, ugyanis az else akkor fog "szóba" kerülni, ha az if-es rész, kifejezés nem teljesül.

Igen népszerű még a programozásban a while illetve a for utasítás. Róluk azt érdemes tudni, hogy mind a ketten ciklusok, a while után is zárójelben adjuk meg a kifejezést hasonlóan mint az if-nél és az addig fog futni ameddig azt az utasításban megadtuk, kivéve ha 0-át adunk neki meg, mert akkor nem fog vizsgálni.

A for ciklus sokak számára lehet ismert az egyik leggyakrabban előforduló utasítás, főleg azoknak akik most kezdenek bele a programozásba. Szóval a for ciklus után szintén zárójelben adjuk meg az utasításokat, itt függvényhívásokat, értékkedésokat, vagy egy reálciós kifejezést szoktunk megadni.

Következő a break utasítás: A break utasítással azt érjük el, hogy ki fog lépni a belső ciklusból, ha ezt az utasítást adjuk a forráskódunkban.

Ismerős lehet még a do-while utasítás. Ő is egy ciklus, ennek az utasításnak a "különlegessége", hogy minden esetben le fog futni.

## 10.3. Benedek-Levendovszky: Szoftverfejlesztés C++ nyelven

### A C++ nem objektumorientált újdonságai

A C és a C++ között az egyik különség az, hogy ha a függvénynek nem adunk paramétert, tehát üres, így C++-ban kell adni paraméterként oda kell adni a void szót, ha azt akarjuk, hogy ne legyen paraméterünk, míg C-ben ezt elég volt üresen hagyni.

A különbség még a visszatérési értékben is megállapítható, ugyanis ha C-ben elfelejtünk visszatérési értéket adni, vagy nem is áll szándékunkban, akkor a visszatérési érték az int. Nos C++-ban ez nem így működik, ha nem írunk visszatérési értéket, akkor a programunk szólni fog, ugyanis így ez hibás lesz.

## 2.1 A main függvény

A main függvénynek 2 megadási módja van, az egyik az, ha a paramétere üres, tehát a zárójelbe nem írunk semmit, a másik módja pedig az, ha a zárójelben megtaláljuk az argc változót, valamint mellette ott van még az argv vektor. A visszatérési értéke 0.

### 2.1.3 A bool típus

A C++-ban megtalálhatjuk a bool típust, ō egy logikai értéket tárol, ilyen kulcsszavakat látunk a C++-ban a bool-al kapcsolatban: bool, a true, és a false.

## 2.2 Függvények túlterhelése

A C++-ban már lehetőségünk van arra, hogy függvényeket terheljünk túl, ezt úgy tehetjük meg, hogy a függvényt nem a neve azonosítja mint azt tette sima C-ben hanem C++ ban a függvényt a név mellett az argumentumlista is azonosítja. A könyv említést tesz az extern használatáról, ami lenyégeiben azért fontos, mert ha akarunk egy C++ függvényt használni C-ben akkor ezt, így tehetjük meg, hogy meghívjuk az extern-el.

## 2.4 Paraméteradás referenciatípussal

C-ben a paraméteradás, az csak értékkel lehetéges, C++-ban azonban ez máshogyan van, itt hivatkozás szerint adjuk meg, a függvények esetében. Ez azért lehet számunkra hasznos, ha a paramétert nem csak használni szeretnénk, hanem szeretnénk benne módosítani. Ezt jelezük, úgy hogy egy a paraméter előtt raktunk egy & jelet, ezzel bevezetjük a referenciatípust.

## **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. Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven

Tanulságok, tapasztalatok, magyarázat...

### 11.2. Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II

#### 1. Nyojc lap alatt a nyelv körül

A Java nyelv hasonló a C++-hoz hiszen ō is egy objektumorientált nyelv, ezt tulajdonképpen annyit jelent, hogy a program amit Java nyelvben írtunk meg objektumokból és osztályokból áll. Az osztályokról érdemes még annyit tudni, hogy változók és metódusok alkotják, hasonlóan a C++ nyelvhez.

A könyv elején a már számunkra is ismerő példával a Hello World-el nyitnak, ezen keresztül szemléltetik számunkra hogyan is működik a fordítója a Java-nak. Ebben az érdekesség az, hogy ez nem úgy működik mint a C++ -ban, hiszen a Java-ban egy úgynevezett Java Virtuális Gép értelmezi a bájtkódöt, és így történik a fordítás.

A példaprogramban egy osztály definíció van, ahol csupán csak egy metódus van meghatározva. Találkozunk a már jól ismert main metódussal, ez továbbra is a programnak a “fő” metódusa. A könyv szerint ha a programot elindítjuk, akkor lényegében ez a fordítóprogram ( Virtuális gép ), azt fogja fordítani, amit megadtunk osztály annak fogja a main metódusát fordítani.

A könyvben továbbá találkozhatunk a változókkal, megjegyzésekkel, valamint konstansokkal is. A változók típusai a következők lehetnek: a char ō egy 16 bites karakter, a boolean ami true vagy false logikai értékkel bír, a byte ami 8 bites egész szám, a short ami 16 bites egész szám, az int ami 32 bites egész szám, a long ami 64 bites egész szám, és végül a 2 lebegőpontos szám ugye a float 32 bites lebegőpontos szám, és a double ami 64 bites lebegőpontos szám.

A megjegyzésről érdemes tudni, hogy egy, de akár több sorosak is lehetnek, ezekre részletesen mutat példát a könyv. Ami érdekes lehet ha egy soros megjegyzést akarunk alkalmazni akkor ezt a // jelöléssel tehetjük

meg. Azonban ha a megjegyzésünk hosszabb ( több mint egy sor ) akkor van egy nyitó jelölés ez az `/*` és a megjegyzésünk végén is van egy jelölés ez pedig ugyan ez csak felcserélve, tehát `*/`.

A konstans megadása a `final` szóval lehetséges.

Az osztályok: A javaban az osztály a `class` szóval hozható létre. Az osztálynak az adattagjait és metódusait számunkra szimpatikus sorrendben sorolhatjuk fel, ezek tulajdonképpen a függvények és a változók. A könyv említ egy típust aminek a neve `String`. Ez az osztály a karakterekkel foglalkozik, ebben az érdekkesség az, mivel ez egy osztály így létrehozhatunk objektumot is belőle ehhez a `new`-ra van szükségünk. A deklarálásnál szó esik még `static`-ról is, ami ujdonság nekünk, a lények amikert deklaráltunk az is az osztályhoz tartozik.

A kivételezés a Java-ban a C++-ból már jól ismert szerkezet a try-catch szerkezet, a Java-ban is örömmel használjuk.

Ebben a részben szó esik még a grafikai megvalósításról, itt említésre kerül az AWT (Abstract Window Toolkit).

## 2. Az Alapok

A számítógépeken legfőkébb két karakterkészlet terjedt el, ennek oka a 8 bit-es karakterábrázolás. Ez a két karakterkészlet: az EBDIC és a ASCII. Érdekesség, hogy mind a kettőnek van korlátja, ez a 8 bitből ered, ezért a speciális karaktereket nem ismeri. Az Unicode karakterkészletben azonban már ez lehetséges, hiszen több biten ábrázolja a karaktereket. Ezt az Unicode karakterkészletet használja a Java nyelve is, valamint futtatáskor is, mindenhol.

A Javaban az azonosítás mindenféleképpen betűvel kell kezdődni, és ezt betű vagy szám követheti. Az azonosító tetszőleges hosszú lehet, viszont vannak olyan szavak amiket nem használhatunk, mivel a nyelv is alkalmazza ezeket a szavakat.

A változó deklarálásához kell egy típus és egy változónév, mind a kettő elengedhetetlen. A változóhoz adhatunk értéket ezt az egyenlőség operátorral tehetjük meg, ezt lehetségünk deklaráláskor.

A Java nyelvben ha tömböt szeretnénk használni akkor ezt a `[ ]` jelöléssel tehetjük meg, ami fontos hogy a Javabam ez típus, még ugye a C++-ban mutató típus volt. A tömb indexelése a nyelvben 0-val kezd.

Mivel a Java egy típusos nyelv, ezért a kifejezésekben megvizsgálja, hogy vannak-e benne összeegyeztethető típusok, tehát az automatikus konverziókat azonos típusra tudja-e alakítani. Ha az ellenőrzés a fordítási időben még elvégezhető, ebben az esetben hiba esetén a fordítás meg fog szakadni. Ha konverziós hibára csak futtatási időben kerül sor, akkor egy hibaüzenet fog létrehozni a futtató rendszer.

## 3. Vezérlés

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

A Python programozási nyelv a C programozási nyelvhez képest fiatal, hiszen 1990-ben megalkotta egy bizonyos Guido van Rossum. A Python programozási nyelvben a prototípusok készítése a köz kedvelt, de természetesen az alkalmazások készítésére is használható, ez a nyelv platformfüggetlen, és objektumorientált. A nyelv sajátossága, hogy nincsen szükség fordításra, csupán a forrás elég neki, ugye ezzel ellentében a C és a C++-valamit a Java nyelvben is szükséges a fordítás. Az alkotó szerint ez a nyelv könnyen tanulható.

Fentebb említettem, hogy a Phyton egy könnyen tanulható nyelv, használata is viszonylag egyszerű valamint megbízható is. A hibák javításában is készségesen segít nekünk, a problémákat átláthatóbbá teszi, ezáltal a kódokat gyorsan tudjuk olvasni.

Megismerhetjük a nyelvnek a szintaxisát, amelyből kiderül, hogy itt már nem használjuk a jól megszokott kapcsos zárójelet(ket). További érdekesség lehet, hogy az utasítás(ok) addig tartanak ameddig a sor végéig nem érünk, tehát nem a pontosvesszőt használjuk ahogyan eddig azt tettük. A nyelvben a megjegyzést a kettőskereszttel tudjuk jelölni.

A Phytonban az adatokat egy objektum reprezentálja. Megfigyelhető, hogy nem kell megadnunk adott esetben egy változó konkrét típusát, hanem van olyan okos a nyelv, hogy kitalálja mire gondolunk. A következő adattípusok fordulhatnak elő: sztringek, számok, szótárak és listák. A könyvben láthatunk róluk példát. Az értékadás egyenlőségjellel valósul meg, szó volt még a DEL szóról, amivel váltózó hozzárendelést lehet törölni.

Nyelv eszközei részben a könyvben, találkozunk a print metódussal, itt egy standard kimenetre írhatunk változókat vagy szövegeket is akár. Találkozhatunk ebben a nyelvben is a már jól ismert függvényekkel, a for és a while fügvénnyel. Tudat a könyv velünk további 2 függvényet ezek a : range és xrange függvények.

Lehetőségünk van címkeket létrehozni, ezt a goto fügvénnyel tehetjük meg. A függvényeket tudjuk definiálni ezt a def szóval tehetjük meg. A kivitelezések nél ugye még Javaban a try-catch-et használtuk, addig itt Phytonban ez egy try-except szerkezet.

A legtöbbben a Phytont a mobiltelefonra fejleszthető munkára használják, ehhez azonban létrehoztak különböző modulokat, ezzel segítve a felhasználók munkáit. A modulra például a camera modul, aminek segítségével hozzáférhetünk a telefonunk kamerájához, továbbá a messaging modul, ahol az SMS üzenetekhez férhetünk hozzá, de a sysinfo modul is hasznos, ugyanis ezzel a készüléknkról szóló információkat kérdezhetjük le. Kimaradt még az audio modul, ezzel pedig a mobiltelefonunkon található hangfelvételkhöz enged hozzáférést. Ezek a modulok valóban nagy mértékben megkönnyítik a munkát a felhasználó és a készülék között.

Összegzésként, a könyv számomra felfedett újdonságokat, érdekességeket tetszett a modulos rész a könyvben, és örülök, hogy egy kicsit megismerhettem még ha csak olvasás módján is jelen pillanatban ezt a könyvet.

## 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.! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG> (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az OOP objektumorientált programozás a felhasználó számára egy olyan lehetőséget nyújt, amivel kevéssé bonyolult a program, illetve a megbízhatóságot növeli. A Java valamint a C++ is objektumorientált programozási nyelv.

A feladat megoldása C++ -ban

```
$ more polargen.cpp
#include "polargen.h"

double PolarGen::kovetkezo()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1 = std::rand() / (RAND_MAX + 1.0);
            u2 = std::rand() / (RAND_MAX + 1.0);
            v1 = 2*u1-1;
            v2 = 2*u2-1;
            w = v1*v1+v2*v2;
        }
        while (w > 1);
```

```
        double r = std::sqrt (( -2 * std::log ( w ) ) / w );
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
        return r * v1;
    }
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}

$ more polargenteszt.cpp
#include <iostream>
#include "polargen.h"

int main(int argc, char **argv)
{
    PolarGen pg;
    for (int i = 0; i < 10; ++i)
        std::cout << pg.kovetkezo () << std::endl;
    return 0;
}
```

Amint látható használjuk a scope operátort, ezáltal lehetőségünk van elérni az std-t. Ez azért fontos mert így tudunk gyököt vonni, és random számot is vissza tudunk adni.

A nincsTarolt változó azért fontos, hogy eldönthessük, hogy páros vagy páratlan lépéssel hívjuk meg a függvényt. Ez a függvény a kovetkezo() függvény.

A feladat megoldása Java -ban

```
public class PolarGenerator {

    boolean nincsTárolt = true;
    double tárolt;

    public PolarGenerator() {

        nincsTárolt = true;
    }

    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;
}
}

public static void main(String[] args) {

PolarGenerator g = new PolarGenerator();

for(int i=0; i<10; ++i)
    System.out.println(g.következő());

}
}
```

Azt vehetjük észre, hogy a 2 állományban több azonosságot is fel tudunk fedezni, így a gondolkodásmenet és a megvalósítás a 2 esetben nagyon hasonló.

## 12.2. Homokozó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciakat kell kiirtani és minden márás 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 Servlethez és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 12.3. Gagyí

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 referenciaja) „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:

Tanulságok, tapasztalatok, magyarázat...

```
$ more Gagyí.java
public class Gagyí
{
    public static void main (String []args)
    {
        Integer x = new Integer (3);
        Integer t = new Integer (x - 0);
        System.out.println (x);
        System.out.println (t);
        while (x <= t && x >= t && t != x);
    }
}
$ javac Gagyí.java
$ java Gagyí
3
3
```

A Java ismerekedés megkezdődik, ahol látunk egy x int típusú változót, ami tárol egy értéket, jelen esetben ez az érték a 3. A t típusú változóban pedig tárolásra kerül az x - 0. Amint észrevehető 0 -át vonunk ki az x-ből tehát a két érték amit a változóban tárolunk megegyezik.

Ezt követően ezt a 2 értéket a System.out.println-el kiíratjuk. Ezt követően pedig beletesszük egy while ciklusba .

A futtatás során azt tapasztalhatjuk, hogy elindítottunk egy végtelen ciklust, illetve, hogy minden esetben a 3-om került a kijelzőre.

Ha meg szeretnénk szüntetni a végtelen ciklust, akkor az Integer értékeit kell átírnunk, és ezáltal megszüntetjük azt.

Nézzük a második programot:

```
$ more Gagyí2.java
public class Gagyí2
{
    public static void main (String []args)
```

```
{  
    Integer x = -128;  
    Integer t = -128;  
    System.out.println (x);  
    System.out.println (t);  
    while (x <= t && x >= t && t != x);  
}  
}  
$ javac Gagyi2.java  
$ java Gagyi2  
-128  
-128
```

Amint láthatjuk itt is hasonlóképpen járunk el mint az előző programban, a különbség az, hogy az x-nek -128-at adunk meg, valamint a t változó értéke most nem x - 0, hanem ő is szintén -128. Ez az érték az integerCache tartománynak a része, ezért ez azt eredményezi, hogy nem fogunk végtelen ciklust kapni, nemúgy mint az első esetben.

Az érdekessége, hogy a két memóriacím egyenlő lesz, hiszen az adott számra ugyanazt azt objektumot kapjuk.

Ő már nem lesz végtelen ciklus, hanem egy véges ciklus, mivel a -128 és 127 az értéktartomány, ezért egy létező objektumra fogunk kapni referenciát, ami által nem teljesül az  $x \neq t$ .

## 12.4. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-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:

Tanulságok, tapasztalatok, magyarázat...

A feladat egy programozási stílusra épül, ez a Yoda conditions, ennek lényege, hogy a kifejezés sorrendje az fel van cserélve. Röviden ez annyit jelent, hogy a konstans a kifejezéseknek (egyenlőségeknek) a bal oldalára kerül. Ez zavart jelenthet a kód olvasása közben, ezért nem éppen közkedvelt módszer.

A nevét a már minden által jól ismert Star Wars filmből kapta, mégpedig a sokak által kedvelt, szimpatikus Yoda-ról, akinek a "különlegessége" az volt, hogy a mondanivalóját nem mindig a már megszokott szósorrendet használta.

Erre a készségre alapul ez a programozási nyelv is.

A porgramozási nyelvet arra használják, hogy elkerüljék a null pointeres hibát.

A programunkban nem kell követni a Yoda conditions szabályait.

A forráskód:

```
public class Yoda  
{
```

```
public static void main(String[] args)
{
    String yodastring = null;

    if (stringem.equals("star_wars"))

    {
        System.out.println("nem null");
    }
}
```

Amint látható egy String típusú változót adtunk meg, értékét null-ra állítottuk. Ami érdekess, hogy a String egy null pointer, ezáltal, az a szöveg, hogy: nem null, az nem fog kiiratásra kerülni.

## 12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/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:

Tanulságok, tapasztalatok, magyarázat...

A feladat megoldása a BBP (Bailey-Borwein-Plouffe) algoritmusára épül, ami segít a Pi hexa jegyeinek a számolását végzi.

Nézzük a forráskódot:

```
public class PiBBP {

    public PiBBP(int d) {

        double d16Pi = 0.0d;

        double d16S1t = d16Sj(d, 1);
        double d16S4t = d16Sj(d, 4);
        double d16S5t = d16Sj(d, 5);
        double d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

        Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};
```

```
}
```

Először is létre kell hoznunk a BBP algoritmust, tehát a Pi-hez tartozó objektumot. A Pi a hexa kifejtésében azt jelenti, hogy  $d+1$  hexa jegytől mennyi hexa jegy.

Ezt követően  $\{16^d \text{Pi}\} = \{4 * \{16^d S1\} - 2 * \{16^d S4\} - \{16^d S5\} - \{16^d S6\}\}$  fogjuk kiszámítani.

```
while (d16Pi != 0.0d) {  
  
    int jegy = (int) StrictMath.floor(16.0d*d16Pi);  
  
    if(jegy<10)  
        sb.append(jegy);  
    else  
        sb.append(hexaJegyek[jegy-10]);  
  
    d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);  
}  
  
d16PiHexaJegyek = sb.toString();  
}
```

Tovább haladva láthatunk egy while ciklust, ami addig fut amíg  $d16Pi$  az nem egyenlő  $0.0d$ -vel. Ezt követően egy if feltételt látunk, ahol megvizsgáljuk, hogy a jegy az kisebb-e mint 10.

```
public double d16Sj(int d, int j) {  
  
    double d16Sj = 0.0d;  
  
    for(int k=0; k<=d; ++k)  
        d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);  
  
    return d16Sj - StrictMath.floor(d16Sj);  
}
```

Ebben a részben kiszámoljuk a  $\{16^d S_j\}$ -t. Ehhez szükségünk van a  $d$  paramétere, amely ugye a  $d+1$  hexa jegytől számoljuk az adott hexa jegyeket. Található a programrészletben egy  $j$  paraméterő lesz az  $S_j$ -nek az indexe.

```
public long n16modk(int n, int k) {  
  
    int t = 1;  
    while(t <= n)  
        t *= 2;  
  
    long r = 1;  
  
    while(true) {  
  
        if(n >= t) {
```

```
        r = (16*r) % k;
        n = n - t;
    }

    t = t/2;

    if(t < 1)
        break;

    r = (r*r) % k;

}

return r;
}
```

Itt használni fogjuk a bináris hatványozást. Ezt azért kell tennünk, hogy ki tudjuk számolni a  $16^n$  mod k-nak az értékét. Ami fontos, hogy ez a 2 paraméter ( n , k ) közül a k a modulus fogja jelölni, míg az n a hatványmutató.

```
public static void main(String args[]) {
    System.out.print(new PiBBP(1000000));
}
}
```

Tovább haladva a main függvényünkben példányosítást hajtunk végre, mégpedig a BBP algoritmust implementáló objektumot.

A programunkban a d értéke 1000000 ( d = 1000000 ), így ezáltal a Pi hexidecimálisan 1000001 hexa jegyekből kerül kiírásra.

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

    for(int i=1000000; i<1001000; i+=2) {
        PiBBP piBBP = new PiBBP(i);
        System.out.print(piBBP.toString().charAt(0));
        System.out.print(piBBP.toString().charAt(1));
    }
}

$ javac PiBBP2.java
$ java PiBBP2
6C65E52CB459350050E4BB178F4C67A0FCF7BF27206290FBE70F93B828CD939C475C728F2FD
B0CB923CF52C40D631D4DB2E98340AA25A6F07DB685C0A9C04F3F6E667CFD6E1764C83ECA94
E79661FC180E6AEF581987E79E13278712CB01255E8CE4D9E048F782D756370548FB0778323
CF2074C2716D121639F1DD5A31EF6C242676B3783AD528852CCA52A9B4F999C526B0750859A
EEC9CE6635B30996A210CD419D5FD47A4E7AAF906E26A4CCF99A2E493BBB5E7D5E0B94F1519
6DA8CD1A0C57FE03A629B2D5842317C173D163EA8717B46930EE0FE82FEC4B01016F155FB44
6AA6958EAD9265EC0C914CB84755DD1BCE5100C23804D67A787BEC57CD7D8E190B3F55E3D25
58927215504F141AC8B0BA836F7781E19664EFA8B22BEB3816A70F7210E4784A1F377623612
86448CD051BCE3A4CE156D70CDBA256C1A36C38648633C8F13A53405795635084A2DEAF3B90
```

```
66BC3863BB07447DDDBDE5644034A6893E3E1CFDB369631BAA4240D93F17F667F7C51ABF076  
F7C1BB35DECC240153F4817A579CBD1DAC895E8555929D1ADA3C787A0BF2881BBC44C4BE505  
E91FE5A28B9BA47D4845B7639239AD71D8B63BF9D23B2CC88C9D39C033B0482F5F801D778BB  
B734EA8B1BE878D129514BFA5C4A6D60E80CF4B14A2A5673992B1839723054BD44F767B0324  
5F2873973EF6D84B2B96EFC9A
```

Ha módosítjuk a main függvényünket az alábbiak szerint, akkor elérhetünk több jegy pontosságot is. Itt például megkaptuk a Pi hexadecimális kifejtésének, ami 1000001. pozíciójától a kifejti a 1000 hexadecimális számjegyét.

## 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:

Tanulságok, tapasztalatok, magyarázat...

Forrás: <https://reiteristvan.wordpress.com/2011/07/05/s-o-l-i-d-objektum-orientált-tervezési-elvek-3-lsp/>

A Liskov-helyettesítés egy tervezési elv az objektumorientált programozásban

Ez a helyettesítés azért fontos nekünk, mert ha van olyan osztály, ami valaminek a leszármazottja, tegyük fel van egy S osztályunk ennek a T osztály a leszármazottja, akkor az S helyére lehet helyettesítést végrehajtani, abban az esetben, tehát olyan helyekre ahol S típust várnánk.

Nézzük a forráskódot:

```
class Madar {  
};  
  
class Program {  
public:  
    void fgv ( Madar &madar ) {  
    }  
};  
  
class RepuloMadar : public Madar {  
public:  
    virtual void repul() {};  
};
```

```
class Sas : public RepuloMadar
{};

class Pingvin : public Madar
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin );

}
```

A feladat lényege, az lenne, hogy olyan programot írunk ami megséríti a Liskov-helyettesítési elvet. Az előzőökben leírtak alapján a Madár osztály lesz nekünk a T osztály. Az S osztály, tehát az alolsztály pedig a RepuloMadar, Pingvin és a Sas.

A P programot, két osztály alkotja az LSP-ben (Liskov Substitution Principle).

A program elején létrehozzuk a Madár osztályt. Láthatjuk a program nevű osztályt, aki a Madar objeftumot "reptetni" fogja a függvények a segítségével.

Ezek után létrehoz ásra kerül még 2 osztály a RepuloMadar, Sas és a Pingvin osztály, ők lesznek ugye az alosztályok.

A programunkban a Madar nem tud repülni hiába van a leszármazottaknak meg a repul metódusa, mivel a Madar& madar miatt nem lehet meghívni. Így mivel erre odafigyeltünk teljesítettük a Liskov-helyettesítéses elvet.

Próbáljuk meg megsérteni az elvet.

forráskód:

```
class Madar {
public:
    virtual void repul() {};
};

class Program {
public:
    void fgv ( Madar &madar ) {
        madar.repul();
    }
};

class Sas : public Madar
```

```
{ };

class Pingvin : public Madar
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin );

}
```

A program elején létrehozzuk a Madár osztályt, amiben szerepel a repül függvény. Ennek a feladata az, hogy reptesse a madarakat.

A Program osztály megegyezik az előzővel.

Ezt követően létrehozunk 2 osztályt a Sas és a Pingvin alosztályt, ami ugye a Madar osztálynak az alosztályai.

Nézzük mi történik a mainben. A main függvényben reptetjük a madarakat, ami viszont feltünhet, hogy itt is adunk a Pingvin objektumra repul() függvényt, pedig tudjuk, hogy ugyan madár a pingvin, de repülni nem tud.

Ami különböziök tehát a 2 forráskód között, hogy a T osztály az továbbra is megegyezik, illetve az S osztályok is viszont a masodik programrészletben nem a RepuloMadar S osztályban jelent meg a “repülés”. Ezzel azt értük el, hogy a madár az tud repülni a P programunkban.

Ezáltal megsértjük a Liskov-helyettesítés elvét, ugyanis ebben a részben ( kódban) a Pingvin tud repülni, ami tudjuk, hogy lehetetlen.

Java-ban:

```
class Madar {
public

void repul() {
}

static class Program {

    void fgv(Madar madar) {
```

```
        madar.repul();
    }
}

;

static class Sas extends Madar

{
}

;

static class Pingvin extends Madar

{
}

;

public static void main(String[] args) {

    Program program=new Program();
    Madar madar=new Madar();
    program.fgv(madar);

    Sas sas=new Sas();
    program.fgv(sas);

    Pingvin pingvin=new Pingvin();
    program.fgv(pingvin);

}
}
```

## 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:

Tanulságok, tapasztalatok, magyarázat...

A feladat az öröklődésen alapszik, az objektumorientált programozásban ezt azért szeretjük használni, mert már egy meglévő osztálynak tudjuk használni a tulajdonságait, és ezek szerint lehetőségünk van új osztályt létrehozni.

Itt jelen esetben ugye a gyerek osztály fogja jelenteni nekünk az új osztályt, ami tudja használni a szülő osztálynak a metódusait illetve tulajdonságait, viszont ezeket felül is tudja írni ha szükséges és ha kell új akkor azt is létrehozhat. Ugye ez fordítva nem lesz igaz, tehát a szülő az nem örökli meg a gyerek tulajdonságait.

A forrás C++ ban

```
#include <iostream>
using namespace std;
class szulo {
public:
    void uzenet() {cout<<"A szulo uzenete\n";}
};

void fgv ( szulo& os ) {
    os.uzenet();
}

class gyerek : public szulo
{
public: void uzenet() {cout<<"Az gyerek uzenete\n";}
};

int main ( int argc, char **argv )
{
    szulo* szulol = new gyerek();
    szulol->uzenet(); //az ős üzenete

}
```

Létrehozzuk a szülő osztályt, ahol, amiből származik a leszármaztatott osztály.

Tovább haladva láthatjuk, hogy a függvényt, ami paraméterként megkapja a szulo-t.

Ezt követően létrehozzuk a gyerek osztályt, amit a szulo osztályból származtatunk, megkapja a szulo osztály tulajdonságait.

A main részhez érve ki akarjuk íratni, hogy mit mond a szülő, de mivel a szülő nem rendelkezik uzenet paraméterrel, ezért a programunk sajnos nem fog lefordulni.

Nézzük meg Java-ban.

```
class szarmaz
{
    public class szulo
    {
        public void uzenet ()
        {
            System.out.println ("A szulo uzenete");
        }
    }
}
```

```
};

public void fgv (szulo szul)
{
    szul.uzenet ();
}

public class gyerek extends szulo
{
    public void uzenet ()
    {
        System.out.println ("A gyerek uzenete");
    }
};

public static void main (String args[])
{
    szarmaz szarm = new szarmaz ();
    szulo szul = szarm.new gyerek ();
    szul.uzenet ();
    gyerek gyer = szarm.new gyerek ();
    szarm.fgv (gyer);
}
}
```

A Java forráskóban a kötésünk dinamikus ezért lehetőségünk van elérni a gyerek üzeneteit.

### 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/tartalom/tanitok-javat/apas03.html#id561066>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A BBP algoritmussal már találkoztunk az első csokorban a Kódolás from scratch feladatban, ahol a Pi hexajegyeit számoltuk ki.

Most a feladat az, hogy a 0. pozícióból indulva számítva a  $10^6$ ,  $10^7$  és  $10^8$  darab jegyét határozzuk meg minden C,C++,Java és C# nyelven és össze kell hasonlítanunk a futtatási időket.

Forráskód:

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

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
```

```
double d16S6t = 0.0d;

int jegy = 0;

long delta = System.currentTimeMillis();

for(int d=1000000; d<1000001; ++d) {

    d16Pi = 0.0d;

    d16S1t = d16Sj(d, 1);
    d16S4t = d16Sj(d, 4);
    d16S5t = d16Sj(d, 5);
    d16S6t = d16Sj(d, 6);

    d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

    d16Pi = d16Pi - Math.floor(d16Pi);

    jegy = (int)Math.floor(16.0d*d16Pi);

}

System.out.println(jegy);
delta = System.currentTimeMillis() - delta;
System.out.println(delta/1000.0);
}
}
```

```
huri@huri-VirtualBox:~/Liskov/Anti$ gcc PiBPP.c -o PiBPP -lm
PiBPP.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main ()
 ^
huri@huri-VirtualBox:~/Liskov/Anti$ ./PiBPP
6
1.812311
huri@huri-VirtualBox:~/Liskov/Anti$ g++ pibbp.cpp -o pibbp
huri@huri-VirtualBox:~/Liskov/Anti$ ./pibbp
6
1.6512
huri@huri-VirtualBox:~/Liskov/Anti$ javac PiBBPBench.java
huri@huri-VirtualBox:~/Liskov/Anti$ java PiBBPBench
6
1.522
```

```
huri@huri-VirtualBox:~/Liskov/Anti\$ mcs PiBBPBench.cs
huri@huri-VirtualBox:~/Liskov/Anti\$ mono PiBBPBench.exe
6
1.58551
```

Amint látható először a  $10^6$ -ot vizsgáljuk, ez esetben a teljesítmény nem meghatározó tényező, hiszen nagyon rövid időn belül megtörtént a futtatás, ami a képen is látható. Szinte minden a 4 futtatás esetén megegyezik az érték 1,5-2 másodperc közé esik, tényleg nagyon hamar végez a program.

Nézzük mi történik  $10^7$  esetén.

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

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
    double d16S6t = 0.0d;

    int jegy = 0;

    long delta = System.currentTimeMillis();

    for(int d=10000000; d<10000001; ++d) {

        d16Pi = 0.0d;

        d16S1t = d16Sj(d, 1);
        d16S4t = d16Sj(d, 4);
        d16S5t = d16Sj(d, 5);
        d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - Math.floor(d16Pi);

        jegy = (int)Math.floor(16.0d*d16Pi);

    }

    System.out.println(jegy);
    delta = System.currentTimeMillis() - delta;
    System.out.println(delta/1000.0);
}
```

```
huri@huri-VirtualBox:~$ cd Liskov
huri@huri-VirtualBox:~/Liskov$ cd Anti
huri@huri-VirtualBox:~/Liskov/Anti$ gcc PiBPP.c -o PiBPP -lm
PiBPP.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main ()
 ^
huri@huri-VirtualBox:~/Liskov/Anti$ ./PiBPP
7
19.284085
huri@huri-VirtualBox:~/Liskov/Anti$ g++ pibbp.cpp -o pibbp
huri@huri-VirtualBox:~/Liskov/Anti$ ./pibbp
7
19.4233
huri@huri-VirtualBox:~/Liskov/Anti$ javac PiBBPBench.java
huri@huri-VirtualBox:~/Liskov/Anti$ java PiBBPBench
7
18.615
```

```
huri@huri-VirtualBox:~/Liskov/Anti$ mcs PiBBPBench.cs
huri@huri-VirtualBox:~/Liskov/Anti$ mono PiBBPBench.exe
7
18.97523
```

Ahhoz hogy  $10^7$ -ent vizsgálunk a for ciklusban a d értékét át kell írnunk úgy, hogy 7 nulla legyen benne, figyelve arra is, hogy a d kisebb értéket is változtassuk.

Amint a képen is látható, itt a futtatás már jelentősen több időt vett igénybe mint előzőleg hiszen még az előbb maximum 2 másodperc volt az érték most ez az érték már megközelíti a 20 másodpercet tehát egy közel 10x-es ugrást tapasztalhatunk időben, C, C++, Java és C# nyelven is .

Vizsgáljuk meg a  $10^8$  esetén mi történik.

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

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
    double d16S6t = 0.0d;

    int jegy = 0;

    long delta = System.currentTimeMillis();

    for(int d=100000000; d<100000001; ++d) {

        d16Pi = 0.0d;

        d16S1t = d16Sj(d, 1);
```

```
d16S4t = d16Sj(d, 4);
d16S5t = d16Sj(d, 5);
d16S6t = d16Sj(d, 6);

d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

d16Pi = d16Pi - Math.floor(d16Pi);

jegy = (int)Math.floor(16.0d*d16Pi);

}

System.out.println(jegy);
delta = System.currentTimeMillis() - delta;
System.out.println(delta/1000.0);
}
}
```

```
huri@huri-VirtualBox:~/Liskov/Anti$ gcc PiBPP.c -o PiBPP -lm
PiBPP.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main ()
 ^
huri@huri-VirtualBox:~/Liskov/Anti$ ./PiBPP
12
223.175033
huri@huri-VirtualBox:~/Liskov/Anti$ g++ pibbp.cpp -o pibbp
huri@huri-VirtualBox:~/Liskov/Anti$ ./pibbp
12
223.168
huri@huri-VirtualBox:~/Liskov/Anti$ javac PiBBPBench.java
huri@huri-VirtualBox:~/Liskov/Anti$ java PiBBPBench
12
207.629
```

```
huri@huri-VirtualBox:~/Liskov/Anti$ mcs PiBBPBench.cs
huri@huri-VirtualBox:~/Liskov/Anti$ mono PiBBPBench.exe
12
217.949955
```

Ahhoz hogy a futtatáskor helyes eredményt kapunk, ismét át kell írnunk a d értékét, úgy hogy most 8 nulla legyen, ugyanúgy mint előzőleg.

Most tapasztalhatjuk, hogy a futtatási idő viszonylag sok időt vesz igénybe hiszen körülbelül 3,5 percet vett igénybe a futtatás minden a 4 esetben (C, C++, C# és Java nyelveken is).

Érdekesség, hogy  $10^6$  ról  $10^7$ -re 10x lassabban fordult a program, majd  $10^7$ -ről  $10^8$ -ra ismét 10x lassabban fordult a programunk, egyenletes lassulást tapasztalunk.

## 13.4. deprecated - Hello, Android!

Élesszük fel a <https://github.com/nbatfai/SamuEntropy/tree/master/cs> projektjeit és vessünk össze néhány egymásra következőt, hogy hogyan változtak a források!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 13.5. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNIST>

Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 13.6. Hello, SMNIST for Humans!

Fejleszd tovább az SMNIST for Humans projektet SMNIST for Anyone emberre szánt appá! Lásd az smnist2\_kutatasi\_jegyzokonyv.pdf-ben a részletesebb háttérét!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 13.7. 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:

Tanulságok, tapasztalatok, magyarázat...

Ciklomatikus komplexitás alatt szoftvermetrikát értünk ezt Thomas McCabe publikálta 1976-ban.

Nézzük a gráfelmélet szerint a gráfban található független utaknak a maximális számát adja.

Két út akkor független a komplexitás szerint, ha mind a két útban van olyan pont vagy él ami nem található a másik útban.

Képlete:  $M = E - N + 2P$

Az E jelöli a gráf éleinek számát, az N a csúcsok számát a gráfban, a P pedig azokat a komponenseknek a számát amik összefüggenek.

Mint említettem a cilomatikus komplexitás egy szoftvermetrika, ami azt jelenti, hogy forráskód alapján határozza meg egy bizonyos program komplexitását.

Sajnos nem sikerült megfelelően telepítenem a complexityt, ezért egy online megoldást kerestem a probléma megoldására, és találtam is egy analizert, amivel sikerült megnézni egy adott program komplexitását. Ez az online program pedig nem más mint a: Lizard Code Complexity Analyzer.

Először nézzünk egy egyszerűbb példát:

The screenshot shows the Lizard Code Complexity Analyzer interface. On the left, there is a code editor window titled "Try Lizard in Your Browser" containing a .c file with the following code:

```
else = masodik-elszo; // elszo = 10, masodik = 5
printf("Ertekelek a csere utan: ");
printf("elszo:%d, masodik:%d\n",elszo, masodik);
return 0;
}
```

On the right, the analysis results are displayed in a table:

| Code analyzed successfully. |      |             |         |             |    |
|-----------------------------|------|-------------|---------|-------------|----|
| File Type                   | .c   | Token Count | 62      | NLOC        | 13 |
| Function Name               | NLOC | Complexity  | Token # | Parameter # |    |
| main                        | 12   | 1           | 59      |             |    |

Amint látható ez egy egyszerű változó cserés program esetén vizsgálja az egyetelen függvény komplexitását, ami jelen esetben 1. Ez jelöli azt, hogy ez egy egyszerű példa.

Nézzünk egy kicsit bonyolultabbat.

The screenshot shows the Lizard Code Complexity Analyzer interface. On the left, there is a code editor window titled "Try Lizard in Your Browser" containing a .c file with the following code:

```
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal nulla);
        free (elem);
    }
}
```

On the right, the analysis results are displayed in a table:

| Code analyzed successfully. |      |             |         |             |    |
|-----------------------------|------|-------------|---------|-------------|----|
| File Type                   | .c   | Token Count | 426     | NLOC        | 96 |
| Function Name               | NLOC | Complexity  | Token # | Parameter # |    |
| uj_elem                     | 10   | 2           | 42      |             |    |
| main                        | 44   | 5           | 203     |             |    |
| kiir                        | 16   | 5           | 87      |             |    |
| szabadit                    | 9    | 2           | 34      |             |    |

Ez már egy kicsit bonyolultabb forráskód, most a már tavalyról jól ismert binfa forráskódot adtam neki, oda látható, hogy több függvényt vizsgál, és sokkal több adat van a táblázatban, mint az előző egyszerű példánkban, lebontja függvényekre a ciklomatikus komplexitást.

## 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\\_nIERIEOs](https://youtu.be/Td_nIERIEOs). <https://arato.inf.unideb.hu/batfai.norbert/UD/> (28-32 fólia)

Megoldás videó:

Megoldás forrása:

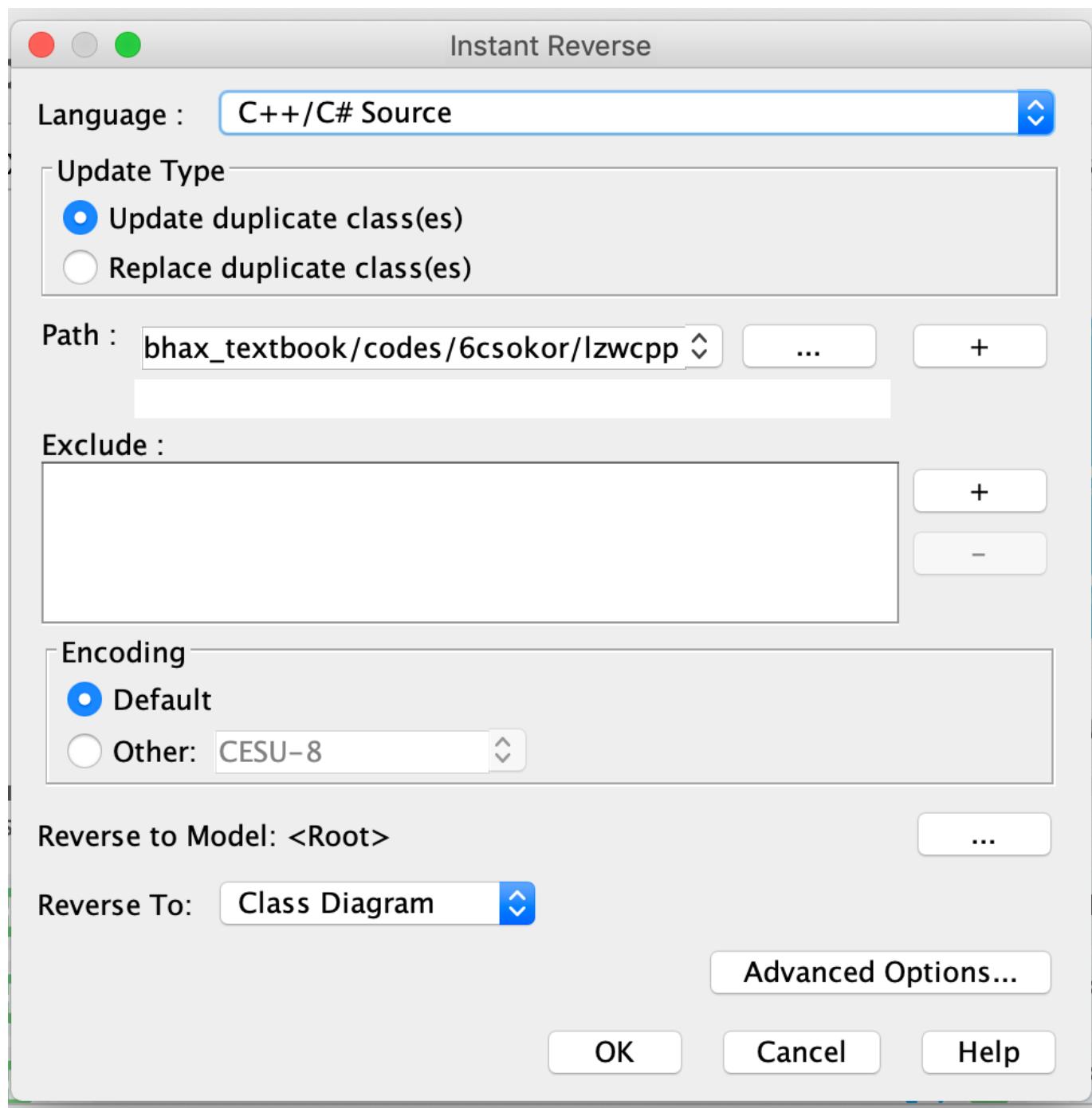
Tanulságok, tapasztalatok, magyarázat...

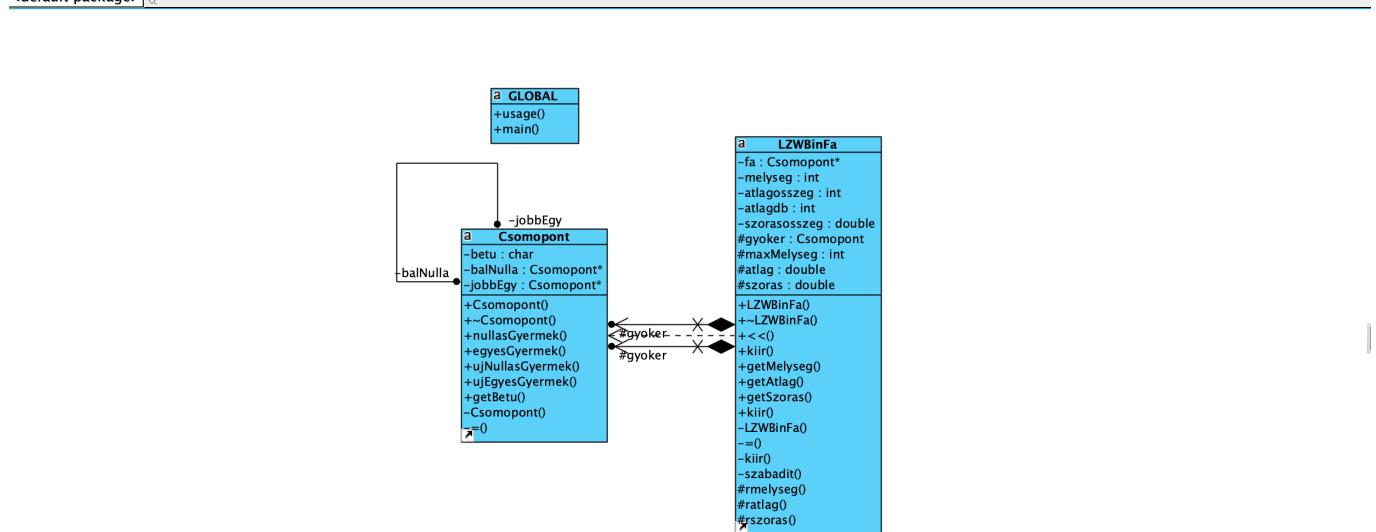
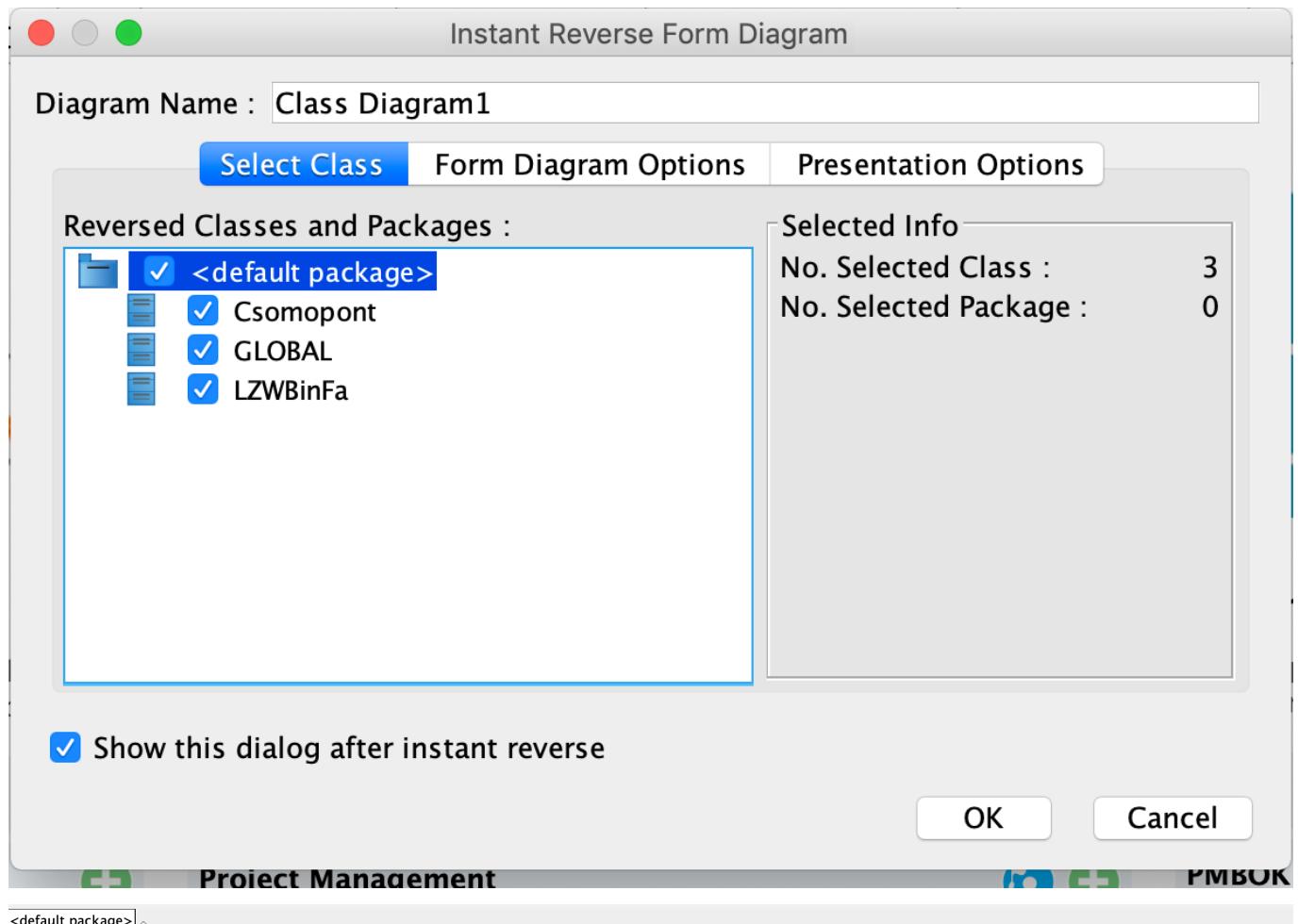
Az UML ( Unified Modeling Language ) -t a kilencvenes évek elején teljesedett ki teljesen, ami egy objektumorientált módszerek alapján kidolgozott modellező eszköz. Az UML lehetőséget biztosít számunkra a fejlesztési elvek, módszerek valamint az eszközök hetékony integrációjára.

Az UML lehetővé teszi számunkra azt, hogy a már meglévő programjainkat meg tudjuk jeleníteni grafikusan is.

A grafikus nyelv úgy alakult ki, hogy a diagramok az UML-ben egy egységes jelölésrendszerbe kerültek, ezáltal az egész egy jól áttekinthető specifikációk, modellek, dokumentációk valamint tervezési készítését segíti az UML. Az UML építőkövei: razzi elemek, diagramok, rajzi elemek közötti relációk.

A feladat végrehajtásához, a Visual Paradigm programot használtam, annak is a Standard változtatát, mivel ez 30 napig ingyenesen elérhető. Ahhoz, hogy megkapjuk az UML diagarammunkat a Tools fülön belül a Code-ben érjük el az Instant Reverse-t, amivel meg tudjuk nézni milyen ábrát kapunk majd. Ennek folyamatát képekben mutatnám be.





Osztálydiagramm: Az osztályokat ennek tartalmát és a kapcsolatrendszernének az összefoglaló diaghrammja az osztálydiagaram. Az osztálydiagrammal leírhatjuk a rendszert fölépítő objektumokat és a közöttük lévő statikus kapcsolatokat is.

A társítások, tehát az asszociációk az két osztály valamint két objektum közötti viszonyt fejeznek ki. Min-

den ilyen társításhoz az irányától függetlenül két osztályt lehet rendelni ( forrás osztály, célosztály )

Ha a diagrammunkban lyukas rombuszt látunk, akkor az UML diagrammunkban aggregációt tapasztalunk, ami azt jelenti, hogy gyenge a tartalmazás. Ha aggrerációt tapasztalunk, akkor a rész az egészhez tartozik, de lehet olyan lehetőség, hogy önmagában is egy létező entitás.

Maradt még a kompozíció ezt egy csúcsára állított rombusz jelenti ami tömve van, ezzel azt takarja hogy a tartalmazás az erős. Az aggrergációval ellentétben a kompozíció esetében a rész önmagában nem valósulhat meg, tehát nem létezhet, hanem csak az egésznek az elemeként.

## 14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

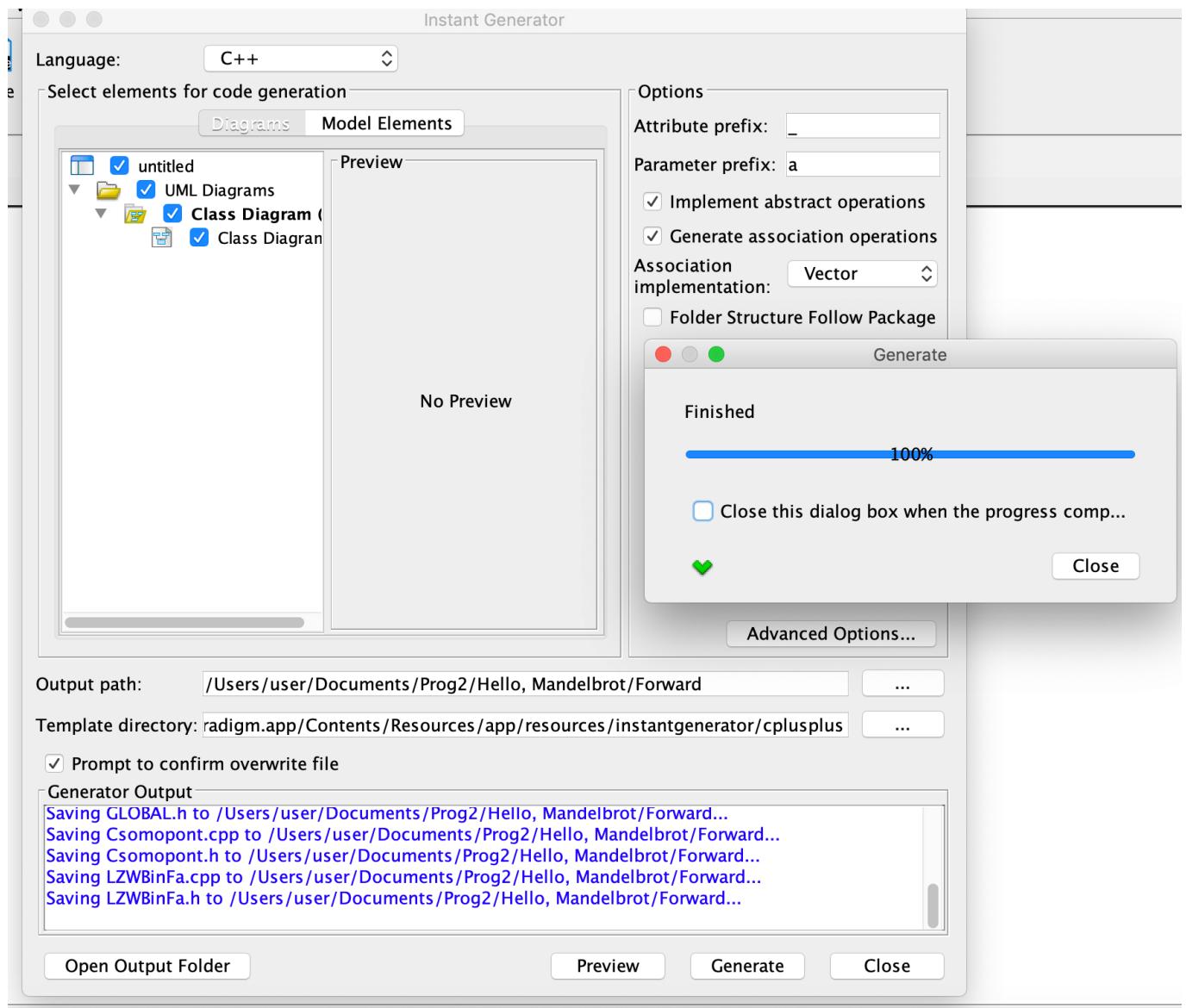
Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az előző feladathoz hasonlóan most is a Visual Paradigm-et használom segítségül a feladat megoldásához. Ugye az előző feladatban egy C++ forráskódból hoztunk létre egy diagrammot, most viszont ezt fordítva kell megtennünk tehát diagramból kell C++ forráskódot létrehoznunk, erre is alkalmas a Visual Paradigm.

A forward engineering megvalósításához hasonlóan mint az előző feladat megint a Tools fülön belül a Code-t kell megkeresnünk. A Code fülön belül pedig az Instant Generatorot kell megkeresnünk. Ha megtaláltuk, akkor az előzőleg létrehozott diagrammból ( ami ugye az lzw binfa volt ) abból fog nekünk egy kódot generálni. Ezt kiválaszthatjuk, hogy milyen formában tegye meg, én most a C++-t választom. Ezt követően megadjuk neki azt, hogy melyik mappába végezzel el a műveletet, majd az eredményt az adott mappában láthatjuk. Ez így néz ki:



A forráskód amit kaptunk:

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

#ifndef _LZWBInFa_h_
#define _LZWBInFa_h_

// #include "Csomopont.h"

class Csomopont;
class LZWBInFa;

class LZWBInFa
{
    private: Csomopont* _fa;
    private: int _melyseg;
    private: int _atlagosszeg;
```

```
private: int _atlagdb;
private: double _szorasosszeg;
protected: Csomopont* _gyoker;
protected: int _maxMelyseg;
protected: double _atlag;
protected: double _szoras;

public: LZWBinFa();

public: LZWBinFa(const LZWBinFa& aForras);

public: LZWBinFa(LZWBinFa&& aForras);

public: LZWBinFa& _(const LZWBinFa& aForras);

public: LZWBinFa& _(LZWBinFa&& aForras);

public: void _LZWBinFa();

public: void _<(const char aB);

public: void kiir();

public: void kiir(std::ostream& aOs);

public: int getMelyseg();

public: double getAtlag();

public: double getSzoras();

private: void kiir(Csomopont* aElem, std::ostream& aOs);

private: Csomopont* copy(Csomopont const* aForras, Csomopont const* ←
aRegifa);

private: void szabadit(Csomopont* aElem);

protected: void rmelyseg(Csomopont* aElem);

protected: void ratlag(Csomopont* aElem);

protected: void rszoras(Csomopont* aElem);
};

#endif
```

Ami megfigyelhető, hogy a header fájlok nagyon pontosak lettek, szinte pontosan megegyezik azzal amit már korábban megírtunk, tehát itt jó munkát végzett a program a diagrammból.

A probléma azonban a cpp fájlból van. Amint látható a forward engineering úgy végezte el, hogy a függvényeknek a törzse üresen marad, mivel a diagramból dolgozta fel a kódot, és csak abból dolgozott a Visual Paradigm. Így megfigyelhetjük azt, hogy a forward engineeringnek ez egy óriási hátránya.

### 14.3. Egy esettan

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladatban a Szofverfejlesztés C++ nyelven című könyvben megtalálható részletet kell feldolgoznunk. Nézzük mi is található benne:

A bevezetésben láthatjuk, hogy az UML-ről lesz szó ( Unified Modeling Language ), itt részletesen bemutatják az osztálydiagrammokat, de ezzel már találkoztunk a feladataink során.

Az osztályok, ami a legfontosabb elem, azok téglalappal vannak jelölve, 3 részre lehet osztani őket, a felső az osztály nevét, a középső az osztálynak az attribútumait, és végül az utolsó osztály pedig tartalmazza az osztály műveleteit.

Szó esik még a láthatóságról is, ezzel adhatjuk meg egy osztály attribútumának szintaxisát. Ami kötelező az a név, láthatóság az pedig 4 féle lehet.

A paraméterlista szintaxisa: irány név: ahol az alapértelmezett érték = típus, fontos megjegyezni, hogy itt az irány az 3 féle lehet. Ez a három irány: Az in, out és végül az inout.

A könyv beszél még a kapcsolatokról is. A kapcsolatot a könyv egy üresháromszög nyíllal jelöli. Az asszociációt ami az osztályok között van azt pedig sima nyíllal jelöljük.

A könyv kitér arra, hogy az asszociációnak két fajtája van : a kompozíció és az aggregáció. A kompozíciót egy tömött robusszal jelöljük, a lényege hogy van egy tartalmazott és egy tartalmazó, ezek együtt jöhetnek létre valamint együtt is szúnhetnek meg. Az asszociációt egy üres rombusszal jelöljük.

Szó esik még a kódgenerálásról is, itt arra tér ki a könyv, hogy akkor jó egy "igazi" osztálydiagramm, hogy ha készíthető belőle kód is. Erre azért az UML-ek képesek is, hasonló feldattal már foglalkoztunk ebben a részben.

Nézzük a feladatot:

A feladat egy program elkészítéséről szól, ahol a feladat egy számítógépes kereskedés a téma. Feladatunk, hogy a számítógépes-alkatrészeket nyílván kell tartanunk. A programnak tudnia kell az állományból való betöltést, a képernyőre listázást, az árképzésnek rugalmas alakítását, valamint pedig az állományba való kiírást is. Ami fontos, hogy úgy kell megadnunk a programot, hogy a jövőben ez még bővíthető is legyen, ha éppen szükség lesz rá.

Nézzük miből épül fel a program: Product ő a termék osztály ebből ComposedProduct, ami az összetett terméket jelöli, a Display értelemszerűen a kijelzőt, a HardDisk pedig a mervelmeznek az osztályát, tehát ez a 3 osztály származtattható a mi Product osztályunkból.

A forráskód:

```
#include "product.h"
#include <string.h>
#include <stdexcept>

using namespace std;

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;
}

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) != &#x1f60e;
        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;
}
```

Amint látható a Product osztálynak 3 változója van, a dateOfAcquisition, name és initialPrice, amit fontos

róluk tudni, hogy ōk protectedek, így kell hozzájuk getterek.

A program célja, hogy az adatfolyamba tudjunk írni és ebből a későbbiekben olvasni is tudjunk, ezért szükségünk van operátorra amivel ezt meg is tehetjük. <><> operátorok ezt lehetővé teszik. Tovább haladva van egy függvényük a GetCurrentPrice ami az adott terméknek az éppen aktuális árát fogja visszaadni nekünk.

A Print függvény fogja elvégezni a kiíratást. Van még a writeParamsToStream függvény valamint a readParamsFromStream függvény, az előbbi az adatfolyamba történő íráshoz szükséges, míg utóbbi az olvasáshoz. Maradt még egy érdekes függvény ez a printParams, ami megadja nekünk a beszerzési dátumot, a termének a korát is megadja valamint a beszerzési árat, illetve a termék aktuális árát is tudatja velünk.

```
#include "display.h"

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;
    }

#include "harddisk.h"

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.cpp-ben megtalálható a printParams függvény amiről már szó esett, hogy pontosan mit is csinál.

A forráskóban minden sor egy adott terméknek az adatait tartalmazza, valamint megfigyelhető az is, hogy sornak az elején, egy adott karakterből álló típuskód áll. Ezek a típuskódok a “h” ami a HardDisket a “d” pedig a Display-t jelöli.

A writeParamsToStream függvényről si volt szó, ugye ő az adtfolyamba való írás a feladata. A loadParamsFromStream függvénynek a feladata, hogy minden termékre beolvassa a közös adatokat.

```
#include "compositeproduct.h"
#include "productfactory.h"

using namespace std;

CompositeProduct::CompositeProduct() : Product() {}

CompositeProduct::~CompositeProduct() {
    for(unsigned i = 0; i < parts.size(); i++)
        delete parts[i];
    parts.clear();
}

void CompositeProduct::addPart(Product *product) {
    parts.push_back(product);
}

void CompositeProduct::printParams(std::ostream &os) const {
    Product::printParams(os);
    os << endl << "Items: ";
    for(unsigned i = 0; i < parts.size(); i++) {
        os << endl << " " << i << ". ";
        parts[i]->print(os);
    }
}

void CompositeProduct::writeParamsToStream(std::ostream &os) const {
    Product::writeParamsToStream(os);
    os << ' ' << parts.size();
    for(unsigned i = 0; i < parts.size(); i++)
        os << endl << *parts[i];
}

void CompositeProduct::loadParamsFromStream(std::istream &is) {
    Product::loadParamsFromStream(is);
    int itemCount;
    is >> itemCount;

    for(int i = 0; i < itemCount; i++) {
        Product* product = ProductFactory::getInstance()->readAndCreateProduct(is);
        if(product) {
            is >> *product;
            addPart(product);
        }
    }
}
```

A compositeproduct.cpp forráskód. A CompositeProduct osztály kerül bevezetésre, ō arra szolgál, hogy

megjelenítse az összetett termékeket. Mivel ez termék ezért a Product osztályból származtatni kell. Ha fel akarunk venni a terméket a listába akkor azt a AddPart(Product\* product) tagfüggvényel tehetjük meg. A kiíratáshoz itt is a printParams függvényt használjuk, valamint azt hogy az összetett termékeket be is tudjuk olvasni, és úgy mondta felül is írjuk ahoz pedig használjuk az előbbiekben már ismert loadParamsFromStream tagfüggvényt.

```
#include "productinventory.h"
#include "productfactory.h"

using namespace std;

ProductInventory::~ProductInventory() {
    emptyProducts();
}

void ProductInventory::emptyProducts() {
    for(unsigned i = 0; i < products.size(); ++i) {
        delete products[i];
    }

    products.clear();
}

void ProductInventory::printProducts(std::ostream& os) const {
    for(unsigned i = 0; i < products.size(); ++i) {
        os << i << ".: ";
        products[i]->print(os);
        os << endl;
    }
}

void ProductInventory::readInventory(std::istream &is) {
    is >> ws;
    while(is.good()) {
        Product* product = ProductFactory::getInstance()->readAndCreateProduct(is);

        if(product) {
            is >> *product;
            addProduct(product);
        }
    }

    cout << "End of reading product items.";
}

void ProductInventory::writeInventory(std::ostream &os) const {
    for(unsigned i = 0; i < products.size(); ++i)
        os << *products[i] << endl;
}
```

```
void ProductInventory::addProduct(Product *product) {
    if (product == NULL)
        throw invalid_argument("ProductInventory::AddProduct - The ←
                               product parameter can not be null.");
    products.push_back(product);
}
```

A productinventory.cpp -ben megismerkedünk a ProductInventory osztálytal. Ennek az osztálynak a feladata, hogy betöltsse a termékek listáját az adott adatfolyamból, tárolja a betöltött termékeket, a termékek adatfolyamba írása, illetve a termékek megjelenítése.

```
#include "productfactory.h"

using namespace std;

ProductFactory* ProductFactory::instance = NULL;

void ProductFactory::Init(ProductFactory* pf) {
    instance = pf;
}

ProductFactory* ProductFactory::getInstance() {
    return instance;
}

Product* ProductFactory::readAndCreateProduct(std::istream &is) {
    if (!is.good())
        return NULL;

    char typeCode;
    is >> typeCode;

    if (!is.good()) {
        if (is.eof()) return NULL;

        cout << "There was an error reading the product items" << ←
             endl;
    }

    return NULL;
}

Product* product = createProduct(typeCode);
if (product == NULL) {
    cout << "Unknown product type." << endl;
}
return product;
}
```

A productfactory.cpp forráskódban a ProductFactory osztálytalálkozunk ezen belül is egy függvénytel, a eadAndCreateProduct, ez lehetővé teszi a termékeknek a termékkódjainak a beolvasását.

```
#include "computerproductfactory.h"
#include "display.h"
#include "harddisk.h"
#include "computerconfiguration.h"

Product* ComputerProductFactory::createProduct (char typeCode) const {
{
    switch(typeCode) {
    case 'd':
        return new Display();
    case 'h':
        return new HardDisk();
    case 'c':
        return new ComputerConfiguration();
    }

    return NULL;
}
```

A computerproductfactory.cpp forráskódban, a hangsúly a termékek létrehozásán van.

```
#include <fstream>
#include "productinventory.h"
#include "productfactory.h"
#include "computerproductfactory.h"
#include "display.h"
#include "harddisk.h"
#include "computerconfiguration.h"

using namespace std;

void readInvFileTest (ProductInventory& inv);
void writeInvFileTest (ProductInventory& inv);

int main() {
    try {
        ProductFactory::Init (new ComputerProductFactory);

        //teszt1
        cout<< "Test1: create inventory and printing it to the <<
            screen." << endl;
        time_t currentTime;
        time(&currentTime);
        ProductInventory inv1;
        inv1.addProduct (new Display ("TFT1", 30000, currentTime, 13, <<
            12));
        inv1.addProduct (new HardDisk ("WD", 25000, currentTime, <<
            7500));
```

```
    inv1.printProducts(cout);

    cout<<"Press any key to continue...";  
    cin.get();  
    cout << endl;

    cout << "Test2: loading inventory from a file ( ←  
          computerproducts.txt), printing it,"  
          " and then writing it to a file ( ←  
          computerproducts_out.txt)." << endl;
ProductInventory inv2;  
readInvFileTest(inv2);  
writeInvFileTest(inv2);

cout << endl;  
cout << "Done.";  
  
cin.get();  
  
    return 0;
}  
catch(const std::exception& e) {
    cerr << "There was an error: " << endl;
    cerr << e.what() << endl;
}  
catch(...) {
    cout << "Unexpected error occurred." << endl;
}
}  
  
void readInvFileTest(ProductInventory &inv) {
    ifstream fs("computerproducts.txt");

    if(!fs) {
        cerr << "Error opening file." << endl;
        return;
    }

    inv.readInventory(fs);
    cout << "The content of the file is: " << endl;
    inv.printProducts(cout);
    cout << endl;
}

void writeInvFileTest(ProductInventory &inv) {
    ofstream fs("computerproducts_out.txt");

    if(!fs) {
        cerr << "Error opening file." << endl;
        return;
    }
}
```

```

    }
    inv.writeInventory(fs);
    cout << "The content of the inventory has been written to " <<
        computerproducts_out.txt" << endl;
}

```

A productinventorytest.cpp szolgálja a keretrendszerünket, a ProductInventoryTest ezzel tudjuk tesztelni.

## 14.4. BPMN

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

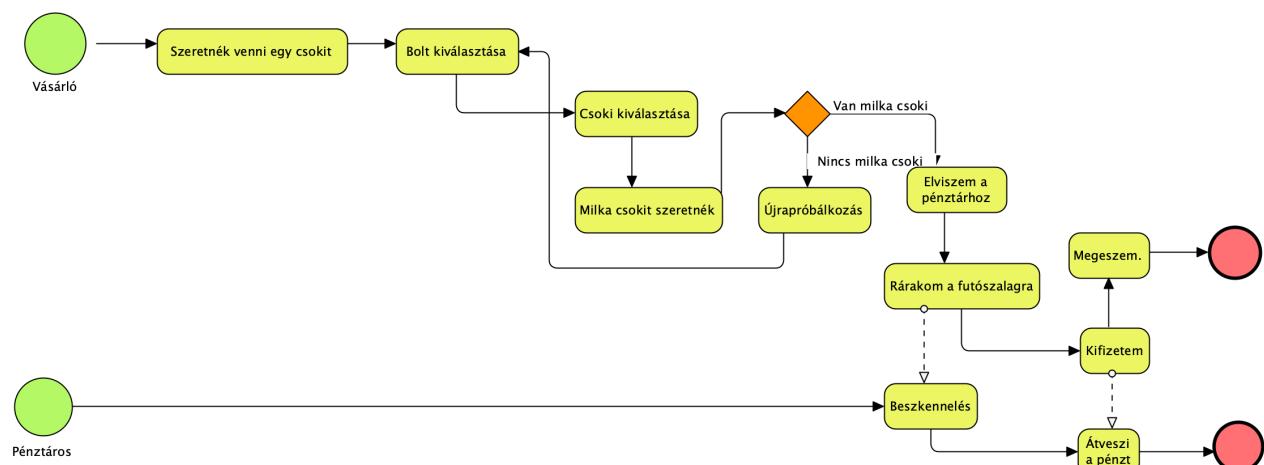
A BPMN egy egységes folyamatábra alapú jelölés üzleti folyamatok modellezéséhez.

A Business Process Model and Notification célja az, hogy az üzleti folyamatot egy egységes grafikai jelölés adjá.

A BPMN nem terjed ki a következők modellezésére: Funkcionális bontás, adat- és információs modellek, stratégiaára, üzleti szabályzat, szervezeti struktúrák és erőforrások.

A folyamatábra készítéséhez a már jól ismert Visual Paradigm-et fogom használni, a folyamatábra készítést pedig a View > Project Browser fülön belül érhetjük el.

Az ábra:



A folyamatábra megrajzolásához segítségül vettetem a Visual Paradigm által adott utasításokat melyet itt lehet elérni: [https://www.visual-paradigm.com/tutorials/businessprocessmodeling.jsp?fbclid=IwAR3\\_-bK0Ddf10HJB](https://www.visual-paradigm.com/tutorials/businessprocessmodeling.jsp?fbclid=IwAR3_-bK0Ddf10HJB)

Az ábrámon 2 sáv látható, az egyik sáv a Vásárló, míg a másik a pénztáros. Az ábrán egy “csoki” vásárlási folyamata látható, ami akkor bonyoldik be egy kicsit amikor 2 lehetséges választási lehetőség van.

Ez pedig az van-e milka csoki vagy nincs, ha van akkor egész könnyen eljuthatunk addig amíg a vásrló megeszi a csokoládét. A bonyolultabb része akkor következik be ugyanis, ha nincs csoki akkor újra kell próbálkoznia, mit ronthatott el, miért nem megfelelő ez a helyzet számára, és így kell eljutni addig a pontig amíg a vásárló megeszi a csokoládét.

## 14.5. BPEL Helló, Világ! - egy visszhang folyamat

Egy visszhang folyamat megvalósítása az alábbi teljes „videó tutoriál” alapján: [https://youtu.be/0OnlYWX2v\\_I](https://youtu.be/0OnlYWX2v_I)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 14.6. TeX UML

Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 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:

Tanulságok, tapasztalatok, magyarázat...

A feladatunk az, hogy a megadott MandelbrotHalmaz forráskódot próbáljuk meg lefordítani linuxban. Nézzük mibe ütközünk ha megpróbáljuk lefordítani:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
UTF-8
    // vizsgáljuk egy adott pont iteraciót:
    ^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
UTF-8
    // vizsgáljuk egy adott pont iteraciót:
    ^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
UTF-8
    // vizsgáljuk egy adott pont iteraciót:
    ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
UTF-8
    // Az egérmutató pozíciója
    ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
UTF-8
    // Az egérmutató pozíciója
    ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
UTF-8
    // Az egérmutató pozíciója
    ^
100 errors
```

Megpróbáltuk lefordítani, és amint látható ez sajnos nem sikerült. A hibaüzenetből látható, hogy pontosan mi is a baj. A probléma az, hogy a program UTF-8 -as kódolásban van, és olyan karakterek találhatóak a forráskódunkban, amiket az UTF-8 as kódolás nem ismer, ezért a fordító sem tudja ezt értelmezni.

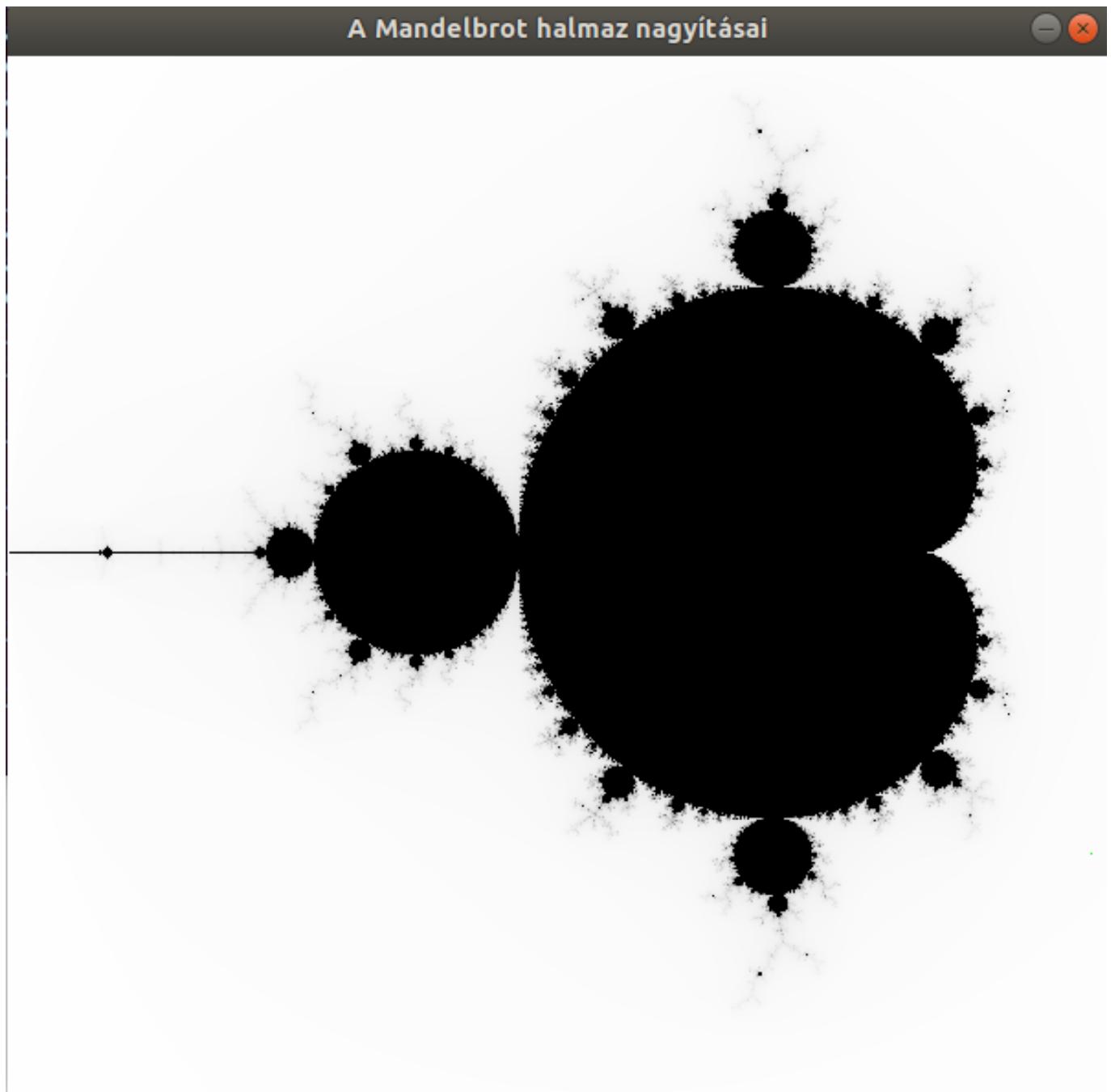
A feladatunk tehát az, hogy olyan kódolást keressünk, amivel le tudjuk fordítani a programot, ami felismeri az ékezetes karaktereket pl: ö. Ha sikerül megtalálnunk ezt a kódolást a fordítónak nem kell, hogy baja legyen.

A megoldás a windows1252-es kódolásra esett ami fontos, hogy ez megegyezik a Latin 1-es betűkészlettel, ezzel elérjük azt, hogy tartalmazni fogja a számunkra szükséges ékezetes karaktereket is.

Az alábbi parancs beírásával a program megfelelően fordult, és a programunk megfelelően működött, alább található is kép a futtatásról.

```
$ $ javac -encoding windows1252 MandelbrotHalmazNagyító.java
```

Kép a futtatásról:



A forráskód:

```
$ more MandelbrotHalmazNagyító.java
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    private int x, y;
    private int mx, my;

    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
        int szélesség, int iterációsHatár) {
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        addMouseListener(new java.awt.event.MouseAdapter() {
```

```
public void mousePressed(java.awt.event.MouseEvent m) {
    x = m.getX();
    y = m.getY();
    if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
        mx = 0;
        my = 0;
        repaint();
    } else {
        MandelbrotIterációk iterációk =
            new MandelbrotIterációk(
                MandelbrotHalmazNagyító.this, 50);
        new Thread(iterációk).start();
    }
}

public void mouseReleased(java.awt.event.MouseEvent m) {
    if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
        double dx = (MandelbrotHalmazNagyító.this.b
                    - MandelbrotHalmazNagyító.this.a)
                    /MandelbrotHalmazNagyító.this.szélesség;
        double dy = (MandelbrotHalmazNagyító.this.d
                    - MandelbrotHalmazNagyító.this.c)
                    /MandelbrotHalmazNagyító.this.magasság;
        new MandelbrotHalmazNagyító(
            MandelbrotHalmazNagyító.this.a+x*dx,
            MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
            MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
            MandelbrotHalmazNagyító.this.d-y*dy,
            600,
            MandelbrotHalmazNagyító.this.iterációsHatár);
    }
}
});

addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent m) {
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});

public void pillanatfelvétel() {
    java.awt.image.BufferedImage mentKép =
        new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKép.getGraphics();
    g.drawImage(kép, 0, 0, this);
    g.setColor(java.awt.Color.BLACK);
```

```
g.drawString("a=" + a, 10, 15);
g.drawString("b=" + b, 10, 30);
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iterációsHatár, 10, 75);
if(számításFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyítás_");
sb.append(++pillanatfelvételszámláló);
sb.append("_");
sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");

try {
    javax.imageio.ImageIO.write(mentKép, "png",
        new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
    e.printStackTrace();
}
}

public void paint(java.awt.Graphics g) {

    g.drawImage(kép, 0, 0, this);
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }

    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
}

public int getX() {
    return x;
}
```

```
public int getY() {
    return y;
}

public static void main(String[] args) {
    new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
```

## 15.2. OOCWC lexer

Izzítsuk be az OOCWC-t és vázoljuk a <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src> lexert és kapcsolását a programunk OO struktúrájába!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 15.3. 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 tettek meg, akkor írassd ki és magyarázd meg a használt struktúratömb memória foglalását!)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Nézzük mi is az a Leet. Az úgynevezett "leet speak" a 80-as években alakult ki, melynek lényege az volt, hogy bizonyos karaktereket, betűket átalakít az eredetihez hasonló karakterré. Ez azért műkás, mert így nagyon nehéz dolgunk van akkor amikor ki szeretnénk olvasni azt, hogy pontosan mit is írtunk, főleg annak aki ezt a kódolást nem ismeri.

A leet az már ismerős lehet számunkra, ugyanis volt a témaival kapcsolatban egy feladat a Prog1-es cso-korban is. Akkor az volt a feladatunk, hogy a Lex készítse el számunkra a kódot, azon szabalyok alapján amiket mi is megadtunk. Most viszont ezt a kódot nekünk kell megírnunk.

A forráskód:

```
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();
}

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

    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);
        }
    }
}
```

A forráskódunk elején azt láthatjuk, hogy megvizsgáljuk az argumentumoknak a számát. Ezt olyan formában tesszük meg, hogy ha az argumentumoknak a száma nem egyezik meg kettővel, akkor felszólítjuk a felhasználót, hogy valamit rosszul csinál, megmondjuk neki, hogyan kellene cselekednie.

Abban az esetben viszont amikor az argumentumok száma az kettő, ekkor létrehozásra kerül egy FileReader valamint egy FileWriter, ezek szolgálják a fájloknak a beolvasását, és a kiíratását. Ezt követően láthatunk egy while ciklust amiben azt látjuk hogy addig kell beolvasnunk a fájt amíg véget nem ér, majd ezt követően kiíratjuk a megadott karaktereknek a leet megfelelőjét. Ha ezzel végzett a programunk akkor bezárja a fáj kiírót és beolvasót is.

Térjünk át a LeetCipher osztályra. Található az osztályban egy String típusú tömb, ō tárolja nekünk a

leetnek a karaktereit. Továbbá találkozhatunk egy újabb String tömbbel értelemszerűen ő pedig a számokat tárolja nekünk és azoknak a leet jeleit.

Ami fontos, hogy található még egy metódus a cipher ő fogja elvégezni a karakterek, betűk átalakítását. Ez úgy fog megvalósulni, hogy eldönti, hogy ami következik az szám-e vagy betű és ennek a bizonyos karakterek visszaadja a megfelelő leetjet.

A program fordítás és futtatás után így néz ki:

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/Leet$ gedit test
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/Leet$ javac Leet.java
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/Leet$ cat test
EZ ITT EGY SZOVEG AMIVEL TESZTELJUK AZT HOGY A PROGRAMUNK MEGFELELOEN MUKODIK E
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/Leet$ java Leet test testout
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/Leet$ cat testout
32 177 36`/ 520\|36 4|V|1\|3| 735273|_||_|< 427 |-|06`/ 4 |>|206|24|V||_|\\||<
|V|36|=3|3|03|\| |V||_|<0[]1|< 3
```

## 15.4. 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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Tutor: [Rácz András](#)

A feladatunk ebben a fealadtban az volt, hogy olyan java-s programot kellett írni, ami teljes képernyős. Ebben segített nekem Rácz András, az ő korábbi projektjét használtuk fel a feladat megoldásához. Itt csak az `egyjatekos()` függvényről fogok szót ejteni, mivel a többkátékos módban nem érhető el a grafikus felület.

Az egymással párhuzamos módban nagyon fontos az első sor, mivel létrehozásra kerül egy `GraphicsDevice`, ami abban segít, hogy megvalósoljuk a teljes képernyőt, amit mi igazán szeretnénk. Ezt követően létrehozásra kerül a `KeyListener`, aminek szerepe akkor lesz ha ESC gombot lenyomjuk, ez azért fontos, hogy amikor ezt a gombot megnyomja az adott felhasználó akkor a programunk ki fog lépni értelemszerűen.

```
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) {}
};
```

Ezt követően létrehozásra kerül egy JFrame, aminek a funkciója az, hogy ő maga az alkalmazásnak az ablaka, és hozzá adjuk a frame-hez a KeyListeneret, majd ezt követően létrehozunk még pár gombot. Letrehozzuk még a szövegmezőt is és ezekhez is hozzárendeljük a KeyListeneret. Ez azért fontos nekünk, mert a KeyListener csak akkor fog működni, ha hozzá van rendelve ahoz az elemhez ami éppen a középpontban van, ezért nem elég csak magához a Framehez hozzárendelni. Tehát ha a programunk vár egy kattintásra, de ehhez a gombhoz nincsen hozzárendelve a KeyListener, akkor bizony az alkalmazásunk nem fog bezárulni, hiába nyomjuk majd meg az ESC gombot.

```
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);
```

Ne nézzük, hogyan is működik a teljes képrenyőre való váltás. Először is meg kell néznünk azt, hogy a számítógépünk támogatja-e a teljesen képernyőt. Erre szolgál a gd.isFullScreenSupported() függvény. Ha a számítógépünk támogatja, akkor undercorated-re kell állítanunk a frame-t ami elengedhetetlen az igazi teljes képrenyőhöz, majd ezt követően a setFullScreenWindow() függvénnyel teljes képernyőre állítjuk a frame-t.

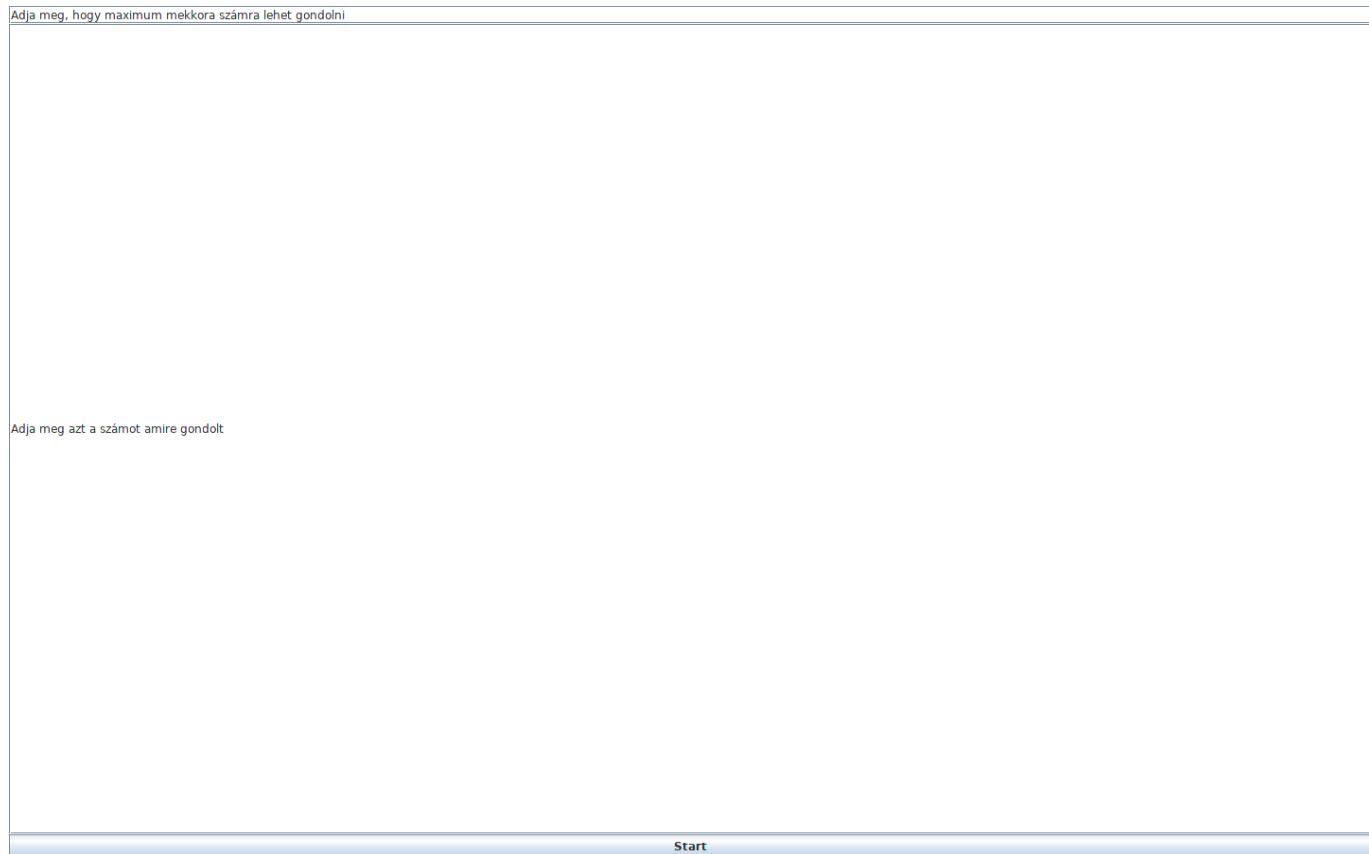
```
if (gd.isFullScreenSupported()) {
    options.setUndecorated(true);
    gd.setFullScreenWindow(options);
}
else{
    System.err.println("Nem jó");
    options.setSize(600, 200);
    options.setVisible(true);
```

```
}
```

Nézzük a programot fordítás és futtatás után:

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/Full
Screen$ javac Client.java
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/Full
Screen$ java Client
Egyjátékos vagy Többjátékos módban szeretnél játszani?
egyjatekos
```

A programunk teljes képrenyőben:



## 15.5. 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:

Tanulságok, tapasztalatok, magyarázat...

Először nézzük meg mi is az a Paszigráfia Rapszódia. Ez egy olyan mesterséges nyelvet próbál kialakítamni, amely kapcsolatot, kommunikációt hoz létre a homunkulusz és a mesteséges homunkulusz között.

A Paszigráfia Rapszódia vizualizáció alapja az SMNIST-hez köthető, viszont ott a pöttyöknek a számosságán volt a hangsúly, de itt most számunkra a helyzetük a fontos.

A feladat forrása a Bátfai Norbert által megadott forrás volt ezekben kellett változtatásokat elvégezni. Ebben az OpenGL alapú vizualizációban, 3 kocka található, bennünk található négyzet betűk, és ezel minden irányban forgathatóak.

Mindenekelőtt azonban, hogy működésre bírjuk a programot, ahoz fel kell telepítenünk a boostot, ezt az alábbi parancssal tehetjük meg: *sudo apt-get install libboost-all-dev*

Szükségünk van még továbbá, az OpenGL-re is ahoz hogy a programunk megfelelően működjön, ehhez pedig a következő parancs szükséges: *sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev*

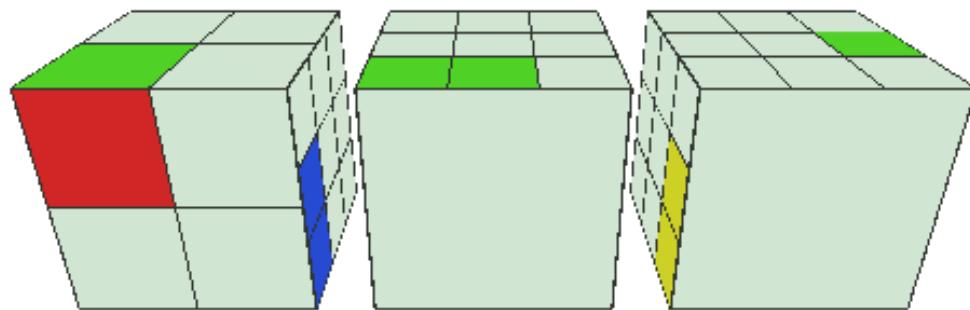
```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/OpenG  
L$ sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

Ha a telepítésekkel megvagyunk, akkor most már tudjuk futtatni a programot. Nézzük hogyan kell fordítani és futtatni a programot (ehhez segítséget kapunk magában a program forráskódjában):

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/OpenG  
L$ g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut  
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Chomsky/OpenG  
L$ ./para 3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:0:1:0:1:0:1:0:2:2:0:1:1:1:  
3:2:1:0:2:0:2:1:1:1:2:3:0:1:1:1:0:3:3:0:1:0:2:1:0:1:0:2:2:0:0:0:1:3:1:0:1:3:2:  
1:0:2:0:3:3:0:1:0:2:1:0
```

A fordítás és futtatás után láthatujuk, hogy a vizualizációban, látható három kocka melyeknek nincsen hátterük, valamint a kockákban, egy-két négyzet színezve van. Ahhoz hogy a kockákat forgatni tudjuk, nyilak segítségével tehetjük ezt meg. Ha ki szeretnénk választani, hogy a 3 kocka közül melyiken végezzünk módosításokat, azt a "0", "1", "2" gombokkal tehetjük meg, értelemszerűen a 0 az első kocka az 1 a második és a 2 pedig a harmadik kocka. A kockáknak a méreteit tudjuk csökkenteni valamint növelni is ezt a "+", és a "-" gombokkal érhetjük el növelés a "+" gomb csökkentés a "-" gomb.

Így néz ki a három kockánk:

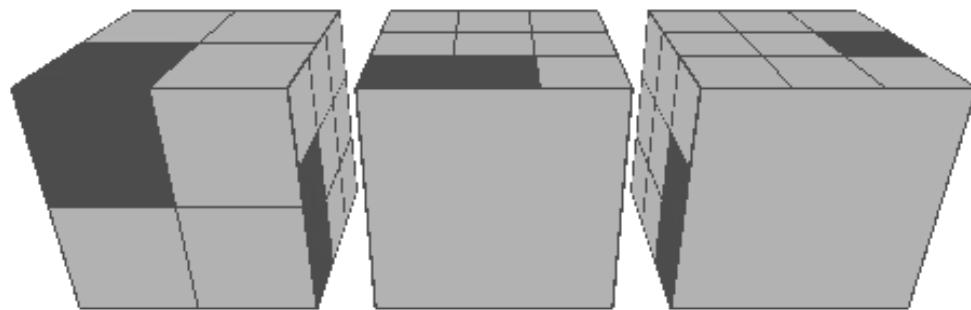
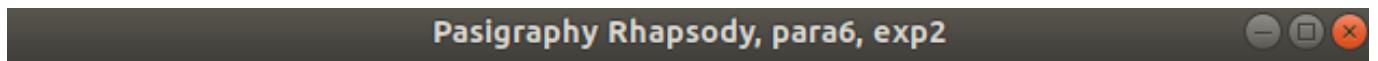
**Pasigraphy Rhapsody, para6, exp2**

Nézzük a modósításokat. Ha szeretnénk az ablak méretén változtani akkor azt a w mint width és h mint height-ben tehetjük meg, úgy hogy átírjuk a változók értékeit.

```
int w = 1024;  
int h = 768;
```

Én amin változtattam, az a kockáknak a színvilága, amihez módosítanunk kellett a glColor3f() függvényt. Amint látható ennek a függvénynek három paramétere van. Ez a három paraméter az RGB-nek megfelelően ugye a Red Green Blue. Tehát az első az a piros a második a zöld a harmadikkal pedig a kék színeknek az "intenzitását" állíthatjuk be. Az én értékeim a következőek voltak: 0.7,0.7,0.7 valamint 0.3,0.3,0.3

Nézzük mi történt ekkor a színekkel:



A következő módosításom pedig az irányításhoz köthető. Ugye a kockákat a nyilakkal tudtuk forgatni, ugye a fel le jobbra és a balra nyilakkal. Ehhez a `keyboard()` függvényt kellett módosítanunk, úgy hogy most már a forgatás a WASDQR billentyűkkel történjen.

```
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 == 't' ) {
        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 );

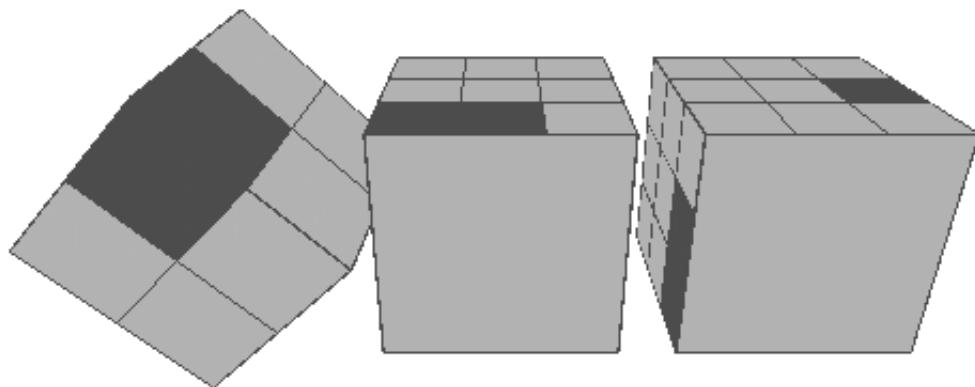
    }

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 == 'r' ) {
    cubeLetters[index].rotz += 5.0;
} else if ( key == 'q' ) {
    cubeLetters[index].rotz -= 5.0;
}

glutPostRedisplay();

}
```

A fogatásról kép:

**Pasigraphy Rhapsody, para6, exp2**

A módosítások után örömmel láthatjuk, hogy sikeresen jártunk el, hisz a WASDQR billentyűkkel sikeresen tudunk forgatni, valamint a színeket is sikerült módosítanunk az általam kedvelt szürke színekre.

A program teljes forráskódja:

```
// Esport Language (PaRa), first experiments
// This is a rapid prototype for planning Pasigraphy Rhapsody (Paszigráfia ←
// Rapszódia, PaRa)
//
// para6.cpp
//
// Copyright (C) 2019 Norbert Bátfai, nbatfa@gmail.com, batfai.norbert@inf. ←
// unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Initial hack
//
// g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut
//./para ↲
3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:0:1:0:1:0:2:2:0:1:1:1:3:2:1:0:2
//
#include <iostream>
#include <vector>
#include <boost/tokenizer.hpp>
#include <GL/glut.h>

class PaRaCube
{
public:
    GLfloat rotx = 0.0f;
    GLfloat roty = 0.0f;
    GLfloat rotz = 0.0f;
    int nn[6] = {1,2,3,4,8,10};
    std::vector<int> cc[6];
};

int index = 0;
bool transp {false};
GLdouble fovy = 70;
int w = 640;
int h = 480;

std::vector<PaRaCube> cubeLetters;

void drawPaRaCube ( int idx )
{
    glPushMatrix();

    int d = cubeLetters.size() /2 ;
    glTranslatef ( ( idx-d ) *2.5f, 0.0f, 0.0f );

    glRotatef ( cubeLetters[idx].rotx, 1.0f, 0.0f, 0.0f );
    glRotatef ( cubeLetters[idx].roty, 0.0f, 1.0f, 0.0f );
    glRotatef ( cubeLetters[idx].rotz, 0.0f, 0.0f, 1.0f );
```

```
glBegin ( GL_QUADS );

    glColor3f ( 0.7f, 0.7f, 0.7f );

    glVertex3f ( -1.0f, 1.0f, 1.0f );
    glVertex3f ( 1.0f, 1.0f, 1.0f );
    glVertex3f ( 1.0f,-1.0f, 1.0f );
    glVertex3f ( -1.0f,-1.0f, 1.0f );

    glVertex3f ( 1.0f, 1.0f, 1.0f );
    glVertex3f ( 1.0f, 1.0f,-1.0f );
    glVertex3f ( 1.0f,-1.0f,-1.0f );
    glVertex3f ( 1.0f,-1.0f, 1.0f );

    glVertex3f ( -1.0f, 1.0f, 1.0f );
    glVertex3f ( -1.0f, 1.0f,-1.0f );
    glVertex3f ( 1.0f, 1.0f,-1.0f );
    glVertex3f ( 1.0f, 1.0f, 1.0f );

    glVertex3f ( -1.0f, 1.0f, 1.0f );
    glVertex3f ( -1.0f, 1.0f,-1.0f );
    glVertex3f ( -1.0f,-1.0f,-1.0f );
    glVertex3f ( -1.0f,-1.0f, 1.0f );

    glVertex3f ( -1.0f, 1.0f,-1.0f );
    glVertex3f ( 1.0f, 1.0f,-1.0f );
    glVertex3f ( 1.0f,-1.0f,-1.0f );
    glVertex3f ( -1.0f,-1.0f,-1.0f );

    glVertex3f ( -1.0f,-1.0f, 1.0f );
    glVertex3f ( 1.0f,-1.0f, 1.0f );
    glVertex3f ( 1.0f,-1.0f,-1.0f );
    glVertex3f ( -1.0f,-1.0f,-1.0f );

    glEnd();

    glBegin ( GL_LINES );

        glColor3f ( 0.3f, 0.3f, 0.3f );

        for ( int i=0; i<=cubeLetters[idx].nn[0]; i++ ) {

            glVertex3f ( -1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ), ←
                         1.005f );
            glVertex3f ( 1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ), ←
                         1.005f );
        }
        for ( int i=0; i<=cubeLetters[idx].nn[0]; i++ ) {

            glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ), 1.0f , ←
                         1.005f );
        }
    
```

```
        glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ), -1.0f ←
                     , 1.005f );

    }

glEnd();

for ( int i {0}; i<cubeLetters[idx].cc[0].size() /2; ++i ) {
    glBegin ( GL_QUADS );

    glColor3f ( 0.3f, 0.3f, 0.3f );

    glVertex3f ( 1.0f- ( cubeLetters[idx].cc[0][2*i]+1 ) * ( ←
                     2.0/cubeLetters[idx].nn[0] ),
                  1.0f- ( cubeLetters[idx].cc[0][2*i+1]+1 ) * ( ←
                     2.0/cubeLetters[idx].nn[0] ), 1.002f );
    glVertex3f ( 1.0f-cubeLetters[idx].cc[0][2*i]* ( 2.0/ ←
                     cubeLetters[idx].nn[0] ),
                  1.0f- ( cubeLetters[idx].cc[0][2*i+1]+1 ) * ( ←
                     2.0/cubeLetters[idx].nn[0] ), 1.002f );
    glVertex3f ( 1.0f-cubeLetters[idx].cc[0][2*i]* ( 2.0/ ←
                     cubeLetters[idx].nn[0] ),
                  1.0f-cubeLetters[idx].cc[0][2*i+1]* ( 2.0/ ←
                     cubeLetters[idx].nn[0] ), 1.002f );
    glVertex3f ( 1.0f- ( cubeLetters[idx].cc[0][2*i]+1 ) * ( ←
                     2.0/cubeLetters[idx].nn[0] ),
                  1.0f-cubeLetters[idx].cc[0][2*i+1]* ( 2.0/ ←
                     cubeLetters[idx].nn[0] ), 1.002f );

    glEnd();
}

glBegin ( GL_LINES );
glColor3f ( 0.3f, 0.3f, 0.3f );

for ( int i=0; i<=cubeLetters[idx].nn[1]; i++ ) {

    glVertex3f ( 1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[1] ) ←
                  , 1.0f );
    glVertex3f ( 1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[1] ) ←
                  , -1.0f );
}

for ( int i=0; i<=cubeLetters[idx].nn[1]; i++ ) {

    glVertex3f ( 1.005f, 1.0f , 1.0f-i* ( 2.0/cubeLetters[idx]. ←
                     nn[1] ) );
    glVertex3f ( 1.005f, -1.0f , 1.0f-i* ( 2.0/cubeLetters[idx] ←
                     ].nn[1] ) );
}
```

```
}

glEnd();

for ( int i {0}; i<cubeLetters[idx].cc[1].size() /2; ++i ) {
    glBegin ( GL_QUADS );

    glColor3f ( 0.3f, 0.3f, 0.3f );

    glVertex3f ( 1.002f, 1.0f-cubeLetters[idx].cc[1][2*i]* ( ←
        2.0/cubeLetters[idx].nn[1] ),
                 1.0f- ( cubeLetters[idx].cc[1][2*i+1]+1 ) * ( ←
                     2.0/cubeLetters[idx].nn[1] ) );
    glVertex3f ( 1.002f, 1.0f-cubeLetters[idx].cc[1][2*i]* ( ←
        2.0/cubeLetters[idx].nn[1] ),
                 1.0f-cubeLetters[idx].cc[1][2*i+1]* ( 2.0/ ←
                     cubeLetters[idx].nn[1] ) );
    glVertex3f ( 1.002f, 1.0f- ( cubeLetters[idx].cc[1][2*i]+1 ←
        ) * ( 2.0/cubeLetters[idx].nn[1] ),
                 1.0f-cubeLetters[idx].cc[1][2*i+1]* ( 2.0/ ←
                     cubeLetters[idx].nn[1] ) );
    glVertex3f ( 1.002f, 1.0f- ( cubeLetters[idx].cc[1][2*i]+1 ←
        ) * ( 2.0/cubeLetters[idx].nn[1] ),
                 1.0f- ( cubeLetters[idx].cc[1][2*i+1]+1 ) * ( ←
                     2.0/cubeLetters[idx].nn[1] ) );

    glEnd();
}

glBegin ( GL_LINES );
glColor3f ( 0.3f, 0.3f, 0.3f );

for ( int i=0; i<=cubeLetters[idx].nn[2]; i++ ) {

    glVertex3f ( -1.0f, 1.005f , 1.0f-i* ( 2.0/cubeLetters[idx] ←
        ].nn[2] ) );
    glVertex3f ( 1.0f, 1.005f , 1.0f-i* ( 2.0/cubeLetters[idx]. ←
        nn[2] ) );
}

for ( int i=0; i<=cubeLetters[idx].nn[2]; i++ ) {

    glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[2] ), 1.005f ←
        , -1.0f );
    glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[2] ), 1.005f ←
        , 1.0f );
}

glEnd();

for ( int i {0}; i<cubeLetters[idx].cc[2].size() /2; ++i ) {
```

```
glBegin ( GL_QUADS );

    glColor3f ( 0.3f, 0.3f, 0.3f );

    glVertex3f ( 1.0f-cubeLetters[idx].cc[2][2*i]* ( 2.0/ ←
        cubeLetters[idx].nn[2] ),
                  1.002f , 1.0f-cubeLetters[idx].cc[2][2*i+1]* ( ←
                      2.0/cubeLetters[idx].nn[2] ) );
    glVertex3f ( 1.0f- ( cubeLetters[idx].cc[2][2*i]+1 ) * ( ←
        2.0/cubeLetters[idx].nn[2] ),
                  1.002f , 1.0f-cubeLetters[idx].cc[2][2*i+1]* ( ←
                      2.0/cubeLetters[idx].nn[2] ) );
    glVertex3f ( 1.0f- ( cubeLetters[idx].cc[2][2*i]+1 ) * ( ←
        2.0/cubeLetters[idx].nn[2] ),
                  1.002f , 1.0f- ( cubeLetters[idx].cc[2][2*i] ←
                      +1]+1 ) * ( 2.0/cubeLetters[idx].nn[2] ) );
    glVertex3f ( 1.0f-cubeLetters[idx].cc[2][2*i]* ( 2.0/ ←
        cubeLetters[idx].nn[2] ),
                  1.002f , 1.0f- ( cubeLetters[idx].cc[2][2*i] ←
                      +1]+1 ) * ( 2.0/cubeLetters[idx].nn[2] ) );

    glEnd();
}

glBegin ( GL_LINES );
glColor3f ( 0.3f, 0.3f, 0.3f );

for ( int i=0; i<=cubeLetters[idx].nn[3]; i++ ) {

    glVertex3f ( -1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[3] ←
        ), 1.0f );
    glVertex3f ( -1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[3] ←
        ), -1.0f );
}
for ( int i=0; i<=cubeLetters[idx].nn[3]; i++ ) {

    glVertex3f ( -1.005f, 1.0f , 1.0f-i* ( 2.0/cubeLetters[idx] ←
        ].nn[3] ) );
    glVertex3f ( -1.005f, -1.0f , 1.0f-i* ( 2.0/cubeLetters[idx] ←
        ].nn[3] ) );
}

glEnd();

for ( int i {0}; i<cubeLetters[idx].cc[3].size() /2; ++i ) {
    glBegin ( GL_QUADS );
    glColor3f ( 0.3f, 0.3f, 0.3f );

    glVertex3f ( -1.002f, 1.0f- ( cubeLetters[idx].cc[3][2*i]+1 ←
        ) * ( 2.0/cubeLetters[idx].nn[3] ),
```

```
        1.0f-cubeLetters[idx].cc[3][2*i+1]* ( 2.0/ ←
            cubeLetters[idx].nn[3] ) );
    glVertex3f ( -1.002f, 1.0f-cubeLetters[idx].cc[3][2*i]* ( ←
        2.0/cubeLetters[idx].nn[3] ),
        1.0f-cubeLetters[idx].cc[3][2*i+1]* ( 2.0/ ←
            cubeLetters[idx].nn[3] ) );
    glVertex3f ( -1.002f, 1.0f-cubeLetters[idx].cc[3][2*i]* ( ←
        2.0/cubeLetters[idx].nn[3] ),
        1.0f- ( cubeLetters[idx].cc[3][2*i+1]+1 ) * ( ←
            2.0/cubeLetters[idx].nn[3] ) );
    glVertex3f ( -1.002f, 1.0f- ( cubeLetters[idx].cc[3][2*i]+1 ←
        ) * ( 2.0/cubeLetters[idx].nn[3] ),
        1.0f- ( cubeLetters[idx].cc[3][2*i+1]+1 ) * ( ←
            2.0/cubeLetters[idx].nn[3] ) );

    glEnd();
}

glBegin ( GL_LINES );
	glColor3f ( 0.3f, 0.3f, 0.3f );

for ( int i=0; i<=cubeLetters[idx].nn[4]; i++ ) {

    glVertex3f ( -1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ), ←
        -1.005f );
    glVertex3f ( 1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ), ←
        -1.005f );
}
for ( int i=0; i<=cubeLetters[idx].nn[4]; i++ ) {

    glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ), 1.0f, ←
        -1.005f );
    glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ), -1.0f ←
        , -1.005f );
}

glEnd();

for ( int i {0}; i<cubeLetters[idx].cc[4].size() /2; ++i ) {
    glBegin ( GL_QUADS );
    glColor3f ( 0.3f, 0.3f, 0.3f );

    glVertex3f ( 1.0f- ( cubeLetters[idx].cc[4][2*i]+1 ) * ( ←
        2.0/cubeLetters[idx].nn[4] ),
        1.0f-cubeLetters[idx].cc[4][2*i+1]* ( 2.0/ ←
            cubeLetters[idx].nn[4] ), -1.002f );
    glVertex3f ( 1.0f-cubeLetters[idx].cc[4][2*i]* ( 2.0/ ←
        cubeLetters[idx].nn[4] ),
        1.0f-cubeLetters[idx].cc[4][2*i+1]* ( 2.0/ ←
            cubeLetters[idx].nn[4] ), -1.002f );
```

```
glVertex3f ( 1.0f-cubeLetters[idx].cc[4][2*i]* ( 2.0/ ←
    cubeLetters[idx].nn[4] ),
             1.0f- ( cubeLetters[idx].cc[4][2*i+1]+1 ) * ( ←
                 2.0/cubeLetters[idx].nn[4] ), -1.002f );
glVertex3f ( 1.0f- ( cubeLetters[idx].cc[4][2*i]+1 ) * ( ←
    2.0/cubeLetters[idx].nn[4] ),
             1.0f- ( cubeLetters[idx].cc[4][2*i+1]+1 ) * ( ←
                 2.0/cubeLetters[idx].nn[4] ), -1.002f );

glEnd();
}

glBegin ( GL_LINES );
	glColor3f ( 0.3f, 0.3f, 0.3f );

for ( int i=0; i<=cubeLetters[idx].nn[5]; i++ ) {

    glVertex3f ( -1.0f, -1.005f , 1.0f-i* ( 2.0/cubeLetters[idx] ←
        ].nn[5] ) );
    glVertex3f ( 1.0f, -1.005f , 1.0f-i* ( 2.0/cubeLetters[idx] ←
        ].nn[5] ) );
}

for ( int i=0; i<=cubeLetters[idx].nn[5]; i++ ) {

    glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[5] ), -1.005 ←
        f , 1.0f );
    glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[5] ), -1.005 ←
        f , -1.0f );

}

glEnd();

for ( int i {0}; i<cubeLetters[idx].cc[5].size() /2; ++i ) {
    glBegin ( GL_QUADS );
    glColor3f ( 0.3f, 0.3f, 0.3f );

    glVertex3f ( 1.0f-cubeLetters[idx].cc[5][2*i]* ( 2.0/ ←
        cubeLetters[idx].nn[5] ),
                 -1.002f , 1.0f-cubeLetters[idx].cc[5][2*i+1]* ←
                     ( 2.0/cubeLetters[idx].nn[5] ) );
    glVertex3f ( 1.0f-cubeLetters[idx].cc[5][2*i]* ( 2.0/ ←
        cubeLetters[idx].nn[5] ),
                 -1.002f , 1.0f- ( cubeLetters[idx].cc[5][2*i ←
                     +1]+1 ) * ( 2.0/cubeLetters[idx].nn[5] ) );
    glVertex3f ( 1.0f- ( cubeLetters[idx].cc[5][2*i]+1 ) * ( ←
        2.0/cubeLetters[idx].nn[5] ),
                 -1.002f , 1.0f- ( cubeLetters[idx].cc[5][2*i ←
                     +1]+1 ) * ( 2.0/cubeLetters[idx].nn[5] ) );
    glVertex3f ( 1.0f- ( cubeLetters[idx].cc[5][2*i]+1 ) * ( ←
        2.0/cubeLetters[idx].nn[5] ),
```

```
        -1.002f , 1.0f-cubeLetters[idx].cc[5][2*i+1]* ←
        ( 2.0/cubeLetters[idx].nn[5] ) );
```

```
        glEnd();
    }
```

```
    glPopMatrix();
}
```

```
void draw ( void )
{
    glClearColor ( 1.0f, 1.0f, 1.0f, 1.0f );

    if ( transp )
        glDisable ( GL_DEPTH_TEST );
    else
        glEnable ( GL_DEPTH_TEST );

    glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    glMatrixMode ( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt ( 0.0f, 3.0f, 6.0f ,0.0f ,0.0f ,0.0f ,0.0f ,1.0f ,0.0f );

    for ( int i {0}; i<cubeLetters.size(); ++i )
        if ( i == index ) {
            drawPaRaCube ( i );
        } else {
            drawPaRaCube ( i );
        }

    glutSwapBuffers();
}

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 == 't' ) {
        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 == 'r' ) {
    cubeLetters[index].rotz += 5.0;
} else if ( key == 'q' ) {
    cubeLetters[index].rotz -= 5.0;
}

glutPostRedisplay();

}

void skeyboard ( int key, int x, int y )
{
    if ( key == GLUT_KEY_UP ) {
        cubeLetters[index].rotx += 5.0;
    } else if ( key == GLUT_KEY_DOWN ) {
        cubeLetters[index].rotx -= 5.0;
```

```
    } else if ( key == GLUT_KEY_RIGHT ) {
        cubeLetters[index].roty -= 5.0;
    } else if ( key == GLUT_KEY_LEFT ) {
        cubeLetters[index].roty += 5.0;
    } else if ( key == GLUT_KEY_PAGE_UP ) {
        cubeLetters[index].rotz += 5.0;
    } else if ( key == GLUT_KEY_PAGE_DOWN ) {
        cubeLetters[index].rotz -= 5.0;
    }

    glutPostRedisplay();
}

void reshape ( int width, int height )
{
    w = width;
    h = height;

    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective ( fovy, ( float ) w/ ( float ) h, .1f, 1000.0f );
    glViewport ( 0, 0, w, h );
    glMatrixMode ( GL_MODELVIEW );
}

int
main ( int argc, char *argv[] )
{

    for ( int i {1}; i<argc; ++i ) {
        std::string s ( argv[i] );
        std::vector<int> nums;
        boost::char_separator<char> separator ( ":" );
        boost::tokenizer<boost::char_separator<char>> items ( s, separator );

        for ( const auto& token : items )
            nums.push_back ( atoi ( token.c_str() ) );

        int ii=0;
        for ( int cui {0}; cui<nums[0]; ++cui ) {
            PaRaCube prc;
            for ( int s {0}; s<6; ++s ) {
                ++ii;
                prc.nn[s]=nums[ii];
                ++ii;
                int noc = nums[ii];
                for ( int coi {0}; coi<noc; ++coi ) {
                    ++ii;
                }
            }
        }
    }
}
```

```
        prc.cc[s].push_back ( nums[ii] );
        ++ii;
        prc.cc[s].push_back ( nums[ii] );
    }
}

cubeLetters.push_back ( prc );

}

glutInit ( &argc, argv );
glutInitWindowSize ( w, h );
glutInitWindowPosition (
    ( glutGet ( GLUT_SCREEN_WIDTH )-w ) /2,
    ( glutGet ( GLUT_SCREEN_HEIGHT )-h ) /2 );
glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
glutCreateWindow ( "Pasigraphy Rhapsody, para6, exp2" );
glutReshapeFunc ( reshape );
glutDisplayFunc ( draw );
glutKeyboardFunc ( keyboard );
glutSpecialFunc ( skeyboard );

glutMainLoop();
return 0;
}
```

## 15.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Lásd vis\_prel\_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 15.7. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! Lásd <https://youtu.be/XpBnR31BRJY>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# 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:

Tanulságok, tapasztalatok, magyarázat...

A feladat elkészítéséhez, aminek a célja az, hogy a JDK programunknak az összes osztályát listázzuk ki, segítségül vettet a Bátfai Tanár Úr Future projekteji között a fenykard.cpp állományra.

Nézzük is a fenykard.cpp forráskódját:

```
void read_acts ( boost::filesystem::path path, std::map < ←
                  std::string, int > &acts )
{

    if ( is_regular_file ( path ) ) {

        std::string ext ( ".props" );
        if ( !ext.compare ( boost::filesystem::extension ( path ) ) ←
            ) {

            std::string actpropspath = path.string();
            std::size_t end = actpropspath.find_last_of ( "/" ) ←
                ;
            std::string act = actpropspath.substr ( 0, end );

            acts[act] = get_points ( path );

        }

    } else if ( is_directory ( path ) )
        for ( boost::filesystem::directory_entry & entry : boost::←
              filesystem::directory_iterator ( path ) )
```

```
        read_acts ( entry.path(), acts );  
  
    }  
  
std::map <std::string, int> acts_map ( std::vector<std::string> folder_names )  
{  
    std::map <std::string, int> acts;  
  
    for ( const auto & node : folder_names ) {  
  
        boost::filesystem::path root ( node );  
        read_acts ( root, acts );  
  
    }  
  
    return acts;  
}
```

Ami érdekes lehet számunkra az a `read_acts` függvény. Ez a függvény azt csinálja, hogy végigmegy az állomány szerkezetén és egy bizonyos kiterjesztésű fájlokat keres, most jelenleg a `.pros` kiterjesztésű fájlokat, és ha ezeket a fájlokat megtalálta, akkor azt ami ezekben a fájlokban található információkat fogja kiolvasni majd pedig eltárolni.

Nézzük a `boost.cpp` állományt.

Először is, ha még nem tettük meg akkor telepítsük a `boost`-ot a géünkre.

```
#include <iostream>  
#include <string>  
#include <boost/filesystem.hpp>  
#include <boost/foreach.hpp>  
  
class Feldolgoz {  
  
private:  
    std::string _path;  
    int _count = 0;  
  
public:  
    Feldolgoz(std::string filePath) :_path(filePath) {}  
  
    void travel(boost::filesystem::path path) {  
        boost::filesystem::directory_iterator it{path}, eod;  
        BOOST_FOREACH(boost::filesystem::path const& p, std::make_pair(it, eod)) {  
            if (boost::filesystem::is_regular_file(p) && boost::filesystem::extension(p.string()) == ".java") {
```

```
        std::cout << p << std::endl;
        _count++;
    }
    else if(boost::filesystem::is_directory(p)) travel(p);
}
}

std::string getPath() {
    return _path;
}

int getCount() {
    return _count;
}

};

void usage() {
    std::cout << "./boost <mappa>\n";
}

int main(int argc, char** argv) {
    if (argc < 2) {
        usage();
        return -1;
    }

    Feldolgoz* obj = new Feldolgoz (argv[1]);
    obj->travel(obj->getPath());
    std::cout << obj->getCount() << std::endl;
}
```

A boost.cpp állománnyal tehát azt érjük el, hogy ami a JDK -ban megtalálható src.zip-ben található fájlokat amik .java állományok, őket fogjuk kilistázni. Természetesen akkor, ha kicsomagoltuk az src.zip könyvtárat. Ha ezt sikerült vegrehajtani, akkor ezt követően kiíratásra kerül a JDK osztályok száma.

A Boost könyvtár azért jó, azért hasznos mert e könyvtár segítségével végig tudunk menni a könyvtárszerkezeten rekúrzívan, ami azt jelenti, hogy azokban az állományokban amik .java-ra végződnek, ezekben a fájlokban lévő osztályokból kigyűjtött információ felgyorsul, és egyben le is egyszerűsödik.

A programot akkor fordíthatjuk is, ha sikerült kicsomagolnunk a src.zip-t. Ezt követően ha megadjuk argumentumként az src mappát, akkor futtatáskor ha minden jól megy megkapjuk a JDK osztályoknak a számát

A parancs amivel futthatjuk a boost-ot:

```
$ g++ boost.cpp -o vegig -lboost_system -lboost_filesystem - ←
  lboost_program_options -std=c++14
$ ./vegig src
```

Nézzük mit is kaptunk eredményként:

```
"src/java.desktop/sun/swing/CachedPainter.java"
"src/java.desktop/sun/swing/FilePane.java"
"src/java.desktop/sun/swing/SwingAccessor.java"
"src/java.desktop/sun/swing/DefaultLayoutStyle.java"
"src/java.desktop/sun/swing/PrintingStatus.java"
"src/java.desktop/sun/swing/SwingUtilities2.java"
"src/java.desktop/sun/swing/icon/SortArrowIcon.java"
"src/java.desktop/module-info.java"
"src/jdk.management.jfr/jdk/management/jfr/SettingDescriptorInfo.java"
"src/jdk.management.jfr/jdk/management/jfr/ConfigurationInfo.java"
"src/jdk.management.jfr/jdk/management/jfr/internal/FlightRecorderMXBeanProvider.java"
"src/jdk.management.jfr/jdk/management/jfr/EventTypeInfo.java"
"src/jdk.management.jfr/jdk/management/jfr/Stringifier.java"
"src/jdk.management.jfr/jdk/management/jfr/package-info.java"
"src/jdk.management.jfr/jdk/management/jfr/StreamManager.java"
"src/jdk.management.jfr/jdk/management/jfr/FlightRecorderMXBean.java"
"src/jdk.management.jfr/jdk/management/jfr/RecordingInfo.java"
"src/jdk.management.jfr/jdk/management/jfr/FlightRecorderMXBeanImpl.java"
"src/jdk.management.jfr/jdk/management/jfr/MBeanUtils.java"
"src/jdk.management.jfr/jdk/management/jfr/StreamCleanupTask.java"
"src/jdk.management.jfr/jdk/management/jfr/Stream.java"
"src/jdk.management.jfr/module-info.java"
"src/jdk.editpad/jdk/editpad/EditPadProvider.java"
"src/jdk.editpad/jdk/editpad/EditPad.java"
"src/jdk.editpad/module-info.java"
"src/jdk.internal.ed/jdk/internal/editor/spi/BuildInEditorProvider.java"
"src/jdk.internal.ed/jdk/internal/editor/external/ExternalEditor.java"
"src/jdk.internal.ed/module-info.java"
18332
huri@huri-VirtualBox:~/Downloads/openjdk-13_linux-x64_bin/jdk-13/lib$ █
```

## 16.2. Másoló-mozgató szemantika

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 videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 16.3. Hibásan implementált RSA törése

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: [https://arato.inf.unideb.hu/batfai.nlp/71-73\\_folia.pdf](https://arato.inf.unideb.hu/batfai.nlp/71-73_folia.pdf) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Tutor: [Vámossi Patrik](#)

Két új osztályt tartalmaz a JDK a BigDecimalt, és a BigInteger-t. Ezt a két osztályt lényegében azért hozták létre, hogy megkönnyítsék a dolgunkat, ha a Float a programunkban nem lenne számunkra megfelelő pontosságú.

Ejtsünk szót erről a 2 osztályról egy kicsit részletesebben.

Nézzük meg elszőször a BigInteger osztályt: Ennek az osztálynak a középpontjában a számelmélet van. Többet között lehet vele vizsgálni a nagyobb értékű kongruenciákat, és az ezekhez tartozó műveleteket is, lehet tesztelni vele prímteszteket is, valamint alkalmas még a elég nagy prímszámoknak az előállításához is. Ez a matematika terület egyesült a kriptográfiával, tehát kitér és foglalkozik azzal, hogy egy üzenetet biztonságosan el tudjunk küldeni.

Most pedig nézzük meg a másik új osztályt a BigDecimalt.

A BigDecimal osztály feladata az, hogy tud ábrázolni olyan valós számokat melyeknek az előjelük tetszőleges pontosságú. Fontos tudni, hogy ez a JDK 1.1-es verzióban, a java.math csomagban megtalálható, majd ezt kesőbb ki is egészítették a lebegőpontossággal is és azoknak a műveletével is. Ez az osztály úgy működik, hogy a típus az egy tetszőleges egész értékből áll valamint egy 32 bites egészből, de olyanból ami nem negatív, ez az úgy nevezett skálázó faktorból áll a BigDecimal.

Nézzi mi is az az RSA: Az RSA a nyílt kulcsú rejtjelzésre alapszik, ennek az lenne a lényege, hogy a titkosítást el akarja választani a dekódolástól. Ezt egy úgynevezett "titkosító" algoritmussal oldja meg, mégpedig úgy, hogy amit kódoláshoz használunk paramétert az nem fog megegyezni azzal a paraméterrel amit a dekodolás paraméternél használunk. Ezért azok által a paraméterek által amit használtunk a kódoláshoz, nem fogjuk tudni meghatározni az, hogy pontosan milyen paraméter szükséges a dekódolásunkhoz.

RSA: Az RSA-t Ron Rivest, Adi Shamir és Len Adleman fejlesztette ki 1976-ban. Az RSA elnevezés a nevük kezdőbetűiből jött létre, alakult meg. Ez az algoritmus napjainkban az egyik leghasználtabb titkosító eljárás. Ennek az eljárásnak az elméleti alapja, modurális számelmélet, és a prímszámelmélet tételei adják. Ez egy asszimetrikus algoritmus ami egy matematikai tételeken alapszik ez a téTEL pedig a Fermat-tétel, ami kimondja azt, hogy ha adott egy p prímszám és ez a prímszám nem osztója egy q egész számnak, akkor ebben az esetben  $q^{(p-1)-1}$  az osztható lesz a p prímszámmal. Nézzük meg 2 véletlenszerű prímszámot legyen p és q. A p értéke legyen 61 a q értéke pedig 53. Ezt követően számoljuk ki n-t melynek értéke:  $n = pq$ . Ugye ez  $61 \cdot 53$  ami 3233. Tovább követve kell egy olyan e szám, ahol  $e > 1$  és ez a szám relatív a  $(p-1)(q-1)$ -hez tehát gyorsan behelyettesítve a  $60 \cdot 52$  ami 3120-hoz relatív. Ez az e szám legyen: 17 Majd számítsuk ki a d-t a  $d = 1 \bmod \phi(n)$ . A d értéke 2753 lesz. Most pedig száoljunk:  $17 \cdot 2753 = 46801 = (15 \cdot 3120) + 1$ . A nyilvános kulcs:  $N = 3233$ ,  $e = 17$ . Ebben az esetben az eljárás a következő:  $c = m^e \bmod N = m^{17} \bmod 3233$ . A titkos kulcs  $N = 3233$ ,  $d = 2753$ , Ebben az esetben a dekódoló eljárás pedig:  $m = c^d \bmod N = 2753 \bmod 3233$ .

Az RSA titkosításnál a legjellemzőbb eljárás a Solovay-Strassen teszt, valamint a Miller-Rabin teszt. Nézzük meg jobban a Miller-Rabin tesztet: Ahol n egy páratlan prím, valamint  $n-1 = 2^r \cdot s$ , itt ebben az esetben az r egy páratlan szám. Legyen a egy tetszőlegesen választott egész, ezt úgy tegyük meg, hogy  $\gcd(a, n) = 1$ . Ebben az esetben az fog történni, hogy  $a^r \equiv 1 \pmod{n}$ , minden  $a^{2^r \cdot j} \equiv -1 \pmod{n}$ , és ez igaz lesz bármilyen j értékre.

Nézzük meg ezt a tesztet: A bemenő számunk legyen egy 3 nál nagyobb vagy egyenlő amely páratlan egész valamint legyen egy t paraméter ami nagyobb egynél, ő lesz az úgynevezett biztonsági paraméter. A kimenő paraméterünk pedig az lesz, azt fogjuk kapni, hogy az adott n számunk az prím vagy össztett. Ezek az RSA összegzés befejeződött.

Nézzük egy Hibásan implementált RSA törlését. Itt segítségül vettem a tanár úr által megadott [fóliák](#) alapján a kulcsgenerálást.

Nézzük meg ennek a forrását:

```
public class RSA{
    public static void main(String[] args)
    {
        int meretBitekben = (int) (700 * (int) (java.lang.Math.log( (double) 10)) / java.lang.Math.log((double) 2));

        System.out.println("Méret bitekben: ");
        System.out.println(meretBitekben);

        java.math.BigInteger p_i = new java.math.BigInteger(meretBitekben, 100, new java.util.Random() );

        System.out.println("p_i");
        System.out.println(p_i);
        System.out.println("p_i hexa");
        System.out.println(p_i.toString(16));

        java.math.BigInteger q_i = new java.math.BigInteger(meretBitekben, 100, new java.util.Random() );

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

        java.math.BigInteger m_i = p_i.multiply(q_i);

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

        java.math.BigInteger z_i = p_i.subtract(java.math.BigInteger.ONE).multiply(q_i.subtract(java.math.BigInteger.ONE));

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

        java.math.BigInteger d_i;
        do {
            do {
                d_i = new java.math.BigInteger(meretBitekben, new java.util.Random());
            } while(d_i.equals(java.math.BigInteger.ONE));
        } while(!z_i.gcd(d_i).equals(java.math.BigInteger.ONE));

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

        java.math.BigInteger e_i = d_i.modInverse(z_i);

        System.out.println("e_i");
```

```
        System.out.println(e_i);
    }
}
```

Miután lefordítottuk a programot a futtatást követően megkapjuk azokat a kulcsokat amik a hibás implementáció alapján jöttek létre.

Kép az esetről:

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Stroustrup/RSA$ javac RSA.java
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Stroustrup/RSA$ java RSA
Méret bitekben:
2019
p_i
524534588642345912162662490743776329734110126535664405539769965572942393774965183465419818830517511120898918969318721505759121174064230600835
8001113890240788620723814364858363080790132429268229820829096078562590257106067219383932966797290904939291332215397511203638478069402184467614
10568410019676541473870246890791789432558372147253275770261260374705112897307638099307616594827648957642570372145297736923330782950438803671094
5266518244657071941406660737215267203435337320344916636207941030183063596641805310202923479907105028731164928999681695661825437300527513489765
280360407061147880103562795519918307
p_i hexa
6f89ba620d7f44fa5949f88d921a0205d533411ab5b144d74c914fa41dd025a9961e63eeded55e2df1addb1bd1178eb135b33ed33c846c404e8ffde8ad0b592efeedb224c
31f1822d0de77b5e4a3bb61ef687b6d1bbf08454a57c67807943d53610bdd71db0af7877d1416b9261d1dedb942f37b8e01f75ab5909d8b299bc80d8c2aeae25c704a0d04e11ff
209822bc93a20ffb6ee3becaf92bc1466f4dc07d4b13eaee791a7c97e8b5b3cecf386c7d88e10f7567819f16764fee1596d4e731ecc1454b6a4c0c3dd48a88590fabaa88f9f6fe2a
139a906334990a0feac8794ea23899a6b2fd9643d301846635360881b82d992496b1cc98940e3
q_i
32332933721165681374535895749197385418906671833532398643831664030301098621454780796513755267805614202455271448219708152211921003355286275789025
96328365513512158635632949036659262558796102274839501306474051322426868153046348683538864575893221339834768388240849853008199027945508962778
9149773169547587306981161238465095100245328026979211509659891001801593755378765810345909985378730260127169144256278719425797480636067204915008
5839970251647679606343858839398853004384253313658666087560377374382954890975402251369184554661675963879003884669947482121584102663194318292820
342299610676548881635735225736300773
m_i
169597420890318753635485386057821216307320218829200766801587837490506409069760464095315651081193415153245442867518536430555667883010945425059
65113650822277308284645623985935524311143550804054378189595539725180243404175528763754982587876072020540618117436058204367426548329372632336446
54409901608378810479613078703271275232778164741288145363084751261511690368376302573565319490310263083735515988587552242581741888488240628677952
06867981178557803406483800091634039691872748268130602468033401072119638774274831777119617476670931644322790466367069052690139818723841157580463
3446290838831780108303522746122492745537232466799884484144145794867976870553662345216948513392091159746536480427945071758604919164125178341951
4084699237821252214119049386018526713184459458287527662375620047939548387654192941765727420603260917589662403580493000365355712264872034025680
26938122047493280594259643476572311266640235032986048944755228140400703738756300424523493547691657806206822458801139148078373477408537723547617
4759508824438156432208346311640681546953631189116888138723324970321086511394305527812795075414896051117530331890533412187906631879859511236997
18735173453602084341708517048849462893822173311313330277620861340951311
z_i
169597420890318753635485386057821216307320218829200766801587837490506409069760464095315651081193415153245442867518536430555667883010945425059
65113650822277308284645623985935524311143550804054378189595539725180243404175528763754982587876072020540618117436058204367426548329372632336446
54409901608378810479613078703271275232778164741288145363084751261511690368376302573565319490310263083735515988587552242581741888488240628677952
06867981178557803406483800091634039691872748268130602468033401072119638774274831777119617476670931644322790466367069052690139818723841157580463
34462908388317801083035227461224926607508398813996118933420600971292958478235977853327905464020657163872495465198199691779429832527993464765781
```

Nézzük a titkosítást. Mag az RSA titkosításához szükségünk lesz két kulcsra. Az egyik a titkos a másik pedig egy nyilvános kulcs legyen. A forrás:

```
public class RSA2{

    static 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(meretBitekben, 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 forráskódunk elején látható, hogy létrehoztuk a KulcsPar osztályt. Ebben az osztályban létrehozásba kerül egy modulus (m), valamint létrehoztuk még a kitevőt(d), és ennek az inverzét(e) ebben az osztályban, ez a 3 tag került létrehozásra. Ebben az esetben amint látható, a p és a q lesz a két nagy prímszám, és ha ezt a két prímszámot összeszorozzuk, akkor megkapjuk a modulusnak az értékét.

Nézzük tovább a forráskódot:

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

    KulcsPar jSzereplo = new KulcsPar(); ←

    String tisztaSzoveg = "A Barcelona harom nullara nyert a hetvegen az ←
        Eibar otthonaban."; ←

    byte[] buffer = tisztaSzoveg.getBytes(); ←
    java.math.BigInteger[] titkos = new java.math.BigInteger[buffer.length ←
        ]; ←

    for(int i = 0; i < titkos.length; i++) ←
    { ←
        titkos[i] = new java.math.BigInteger(new byte[]{buffer[i]}); ←
        titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo.m); ←
    } ←
}
```

Mivel titkosítunk, ezért egy titkos szöveget kell feltöltenünk. Viszont ezt úgy kell megtennünk, hogy az ASCII kódolást használjuk.

```
for(int i = 0; i < titkos.length; i++) ←
{
    titkos[i] = new java.math.BigInteger(new byte[]{buffer[i]}); ←
    titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo.m); ←
}
```

A for ciklus első sorában történik a feltöltés a tiszta szöveggel. Ugye az ASCII kódolással történik. A for ciklus második sorában használjuk a modpow függvényt, ezzel a függvénytel átalakítjuk nagy integerré,

ennek az első paramétere az exponens, ez ugye d nek az inverze legyen e emiatt fontos, így sikerülhet a visszafejtés. Ez volt az első paramétere, a második paramétere pedig a modulus.

Nézzük magát a visszafejtést:

```
for(int i = 0; i < titkos.length; i++)
{
    titkos[i] = titkos[i].modPow(jSzereplo.d, jSzereplo.m);
    buffer[i] = titkos[i].byteValue();

}
System.out.println(new String(buffer));
}
```

Itt az történik, hogy a modPow függvény visszaalakítja a nagyuntegereket az ASCII kóddá ugye az inverz miatt. Ezt követően a for ciklus második sorában feltöltjük a buffer tömböt az ASCII kódok megfelelő betűivel, majd ezt követően String készül belőle.

Ha a programot sikeresen fordítottuk és futtattuk, akkor az RSA alapján a visszafejtett szöveget fogjuk visszakapni.

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Strousstrup/RSA$ javac RSA2.java
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Strousstrup/RSA$ java RSA2
A Barcelona harom nullara nyert a hetvegen az Eibar otthonaban.
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Strousstrup/RSA$
```

## 16.4. Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladatunk nagyon hasonló mint az eddigi Perceptronos feladataink. A feladat megoldásához segítségül vettettem a mandel.cpp állományunkat, ahol létrehozunk egy kétdimenziós halmaznak a png ábáját.

Nézzük a forrást:

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

```
int main (int argc, char **argv)
{
    png::image<png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width() *png_image.get_height();

    Perceptron* p = new Perceptron (3, size, 256, size);

    double* image = new double[size];

    for (int i {0}; i<png_image.get_width(); ++i)
        for (int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width() + j] = png_image[i][j].red;

    double* newimage = (*p) (image);

    for (int i = 0; i<png_image.get_width(); ++i)
        for (int j = 0; j<png_image.get_height(); ++j)
            png_image[i][j].blue = newimage[i*png_image.get_width() + j];

    png_image.write("kimenet.png");

    delete p;
    delete [] image;
}
```

A programunk elején látható, hogy szükségünk lesz bizonyos könyvtárakra: Ezek a könyvtárak pedig a libpng és a libpng++, tehát ha még nincs telepítve, akkor ezt tegyük meg. Ezen kívül még szükségünk van png++/png.hpp fájlra is. Valamint includeoljuk még az elején a mlp.hpp-t is azért mert a perceptronunk az többrétegű.

Nézzük mit tartalmaz a main függvényünk: Az első parancssori argumentumot adjuk meg, ez alapján kerül beolvasásra majd a képállományunk. A kép méretét egy változóban tároljuk el, a get\_width és a get\_height szorzata adja majd a pontos képméretünket. Ezt követően példányosítás következik, a new operátornak a segítségével tehetjük ezt meg.

Tovább haladva létrehozunk egy double mutatót. Ebben láthatunk egymásba ágyazott for ciklusokat. Amint látható egy egyik for ciklus az végig meg a kép szélességén, míg a másik for ciklus végig meg a magasságán. Ezt követően az image tárolja a képállományunk piros vörösnek a színkomponensét.

Ezt követi a kép generálás. ehhez ismét szükségünk van a már előzőleg ismert double mutatóra. Majd megint a két for ciklussal végigmegyünk a képnek a szélésségén és a magasságán is, valamint megkapja majd az új színeket is. A write függvényünk arra szolgál, hogy ezáltal létre tudjuk hozni a képünket, mellyek a formátuma png.

A fordításhoz hasznájuk a C++11 szabványt, majd fordítsuk és futtassuk a programunkat.

Nézzük, hogyan működik a programunk.

The screenshot shows a Linux desktop environment with several windows open:

- A terminal window titled "bhax-textbook-feladatok2-lli" with the following text:

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_Textbook/codes/Strousstrup/Argument...  
Nézzük a fórumon! Nézzük a fórumon!  
</para> mentum Ctor$ g++ mlp.hpp main.cpp -o perceptron -lpng -std=c++14  
</para> huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_Textbook/codes/Strousstrup/Argu  
<programli...> huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_Textbook/codes/Strousstrup/Argu  
mentum Ctor$ ls  
main.cpp mandel.png mlp.hpp output.png perceptron  
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_Textbook/codes/Strousstrup/Argu  
#include <iostream> #include "mlp.hpp" #include <png++/png.h> #include <cmath>
```
- A code editor window titled "bhax-textbook-feladatok2-stroustrup.xml" showing C++ code.
- Two image viewers side-by-side. The left viewer shows a black and white fractal image (Mandelbrot set) on a white background. The right viewer shows the same fractal image on a yellow background.

## 16.5. Összefoglaló

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

Megoldás video:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# 17. fejezet

## Helló, Gödel!

### 17.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világ bajnokságban <https://youtu.be/DL6iQwPx1Yw> (8:05-től)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Robocar World Championship, egy platform aminek segítségével lehetőségünk adódik bizonyos robotauto-kutatásra, valamint a forgalom-algoritmusránk általánosítása. Az új modelleknek és a ezeknek a tesztelését, egy bizonyos Robocar City Emulator teszi/tette volna lehetővé.

A feladatunkhoz, a rendezéshez szükség van a [GitHub projektet](#) erre a projektre. Számunkra a `carlexer.ll` forráskódra van szükségünk és az ebben található, `scnaf` függvény lehet érdekes.

Ahhoz hogy definiálni tudjunk egy vagy akár több soros függvényeket szükségünk van a C++11-ben megtalálható lambda kifejezésekhez. Ezeket a függvényeket azért használjuk mert a továbbiakban nem szükségesek, tehát arra jók, hogy ezeket a függvényeket csak egyszer használjuk, ezért jellemzően ezeknek a függvényeknek így általában nevet sem adunk neki.

A lambda kifejezéseknek van egy általános alakjuk ami így néz ki:

```
[ ] (int x, int y) { return x + y; }
```

Nézzük mi mit is jelent ebben a kifejezésben: A szöglletes zárójel azt jelenti, hogy egy lambda kifejezésről lesz szó. A kerek zárójelekkel tudjuk megírni a függvényt amire szükségünk van. Majd a kapcsos zárójelek között van a függvényünknek a törzse ezen zárójelek között pedig megtalálhatjuk a return-t ez az utasítás pedig arra jó, hogy meghatározzuk a függvényünknek az értékét és típusát.

A `myshmclient.cpp` fájlban találunk egy `gangsters` vektort ezt kellene nekünk sorba rendezni, lambda kifejezéssel. Nézzük is meg a példát:

```
std::sort (gangsters.begin(), gangsters.end(), [this, cop] ( ←
    Gangster x, Gangster y)
{
    return dst (cop, x.to) < dst (cop, y.to);
```

```
    } );  
  
    void sort (RandomAccessIterator first, RandomAccessIterator last, ←  
               Compare comp);
```

Nos a forráskódból a gangsters vektor fog rendezésre kerülni, ehhez szükség van a sort függvényre aminek 3 paramétere van a begin és az end, a harmadik pedig a lambda kifejezés. Ennek a kifejezésnek a paramétere két Gangster az x és y. A kifejezésünk booleanal tér vissza abban az esetben, ha igaz akkor akkor az x gengster és a rendőrnek a távolsága kisebb, mint a másik tehát az y gengster és rendőr távolsága. Így elmondható az, hogy a vektor az a távolság alapján van rendezve.

## 17.2. 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:

Tanulságok, tapasztalatok, magyarázat...

## 17.3. STL map érték szerinti rendezése

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Tutor: [Rácz András](#)

A feladatunk az, hogy rendezzük a kulcs-érték párokat kell rendeznünk. Ezt alapvetően kulcs szerint tennénk meg, viszont most érték szerint kell.

Nézzük a programunk forráskódját

```
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;  
    }  
}
```

Amint láthatjuk a forráskódban, a kulcs-érték párokat egy osztályban tároljuk el, melyeknek van két tagváltozójuk, ezek pedig a kulcs és az érték változók. Fontos tudni róluk, hogy ezek a változók privát voltózók, így megtalálhatóak a setterek és a getterek is amik hozzájuk tartoznak.

Ezt követően találhatunk egy `toString` metódust is, ami azért felelős, hogy kiírja az objektumoknak a tagválzotónak az értékeit. Fontos még tudnunk azt is, hogy a konstruktur az egy `String` tömböt fog kapni, melynek az első eleme lesz az adott objektumnak a kulcsa, a második eleme pedig az objektumnak az értéke.

Nézzük a beolvasást, valamint a rendezést. Úgy kezdjük az egészet, hogy meg kell számolnunk azt, hogy hány darab kulcs érték párunk van a feladat.txt fájlunkban ezt követően pedig egy `RandomAccessFile`-al be is olvassuk azt a Map tömbünkbe.

Nézzük tovább a forráskódot:

```
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();
```

Találhatunk az elején négy darab változót. Az első az a RandomAccessFile, ō fogja a kulcs-érték párokat tartalmazó fájlnak tartalmazni az elérési útját. Ez egy sor String, ez a beolvásásnál az egy-egy sorát fogja tárolni, mégpedig egy Map[] tömb, ezekbe kerülnek a kulcs-érték párok valamint még egy int db, ennek segítségével tudjuk megadni a fájlban lévő soroknak a számát és megadja nekünk még a tömbnek a hosszát.

Ezt követően inicializáljuk a RandomAccessFile-t valamint a db változót is szintén, majd pedig egy for ciklussal megszámoljuk a soroknak a számát. Ezt követően ismét inicializálás történik mégpedig a Map tömbre itt a db változó értékét 0-ra álltjuk, majd ezt követően visszaugrunk a fájlnak az elejére. Végigmegyünk a fájlnak a sorain, ahol a kulcs-érték párok azok vesszővel vannak elválasztva egymástól, így a konstruktornak át fogunk adni egy két elemű tömböt amit, a vesszőnél sor.split(", ") segítsével tehetjük ezt meg. Ha ezt megettük akkor kiíratjuk az eredeti tömbünket a for each ciklussal, ezt pedig a shell rendezéssel fogjuk rendezni, majd pedig kiíratjuk 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);
    }
}
}
```

Nézzük a programot fordítás és futtatás után hogyan működik:

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Godel$ javac stlmap/*.java
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Godel$ java stlmap/Stlmap
eredeti értékek:
kulcs=a, ertek=11
kulcs=f, ertek=43
kulcs=d, ertek=32
kulcs=z, ertek=1
kulcs=t, ertek=10
kulcs=c, ertek=4
kulcs=k, ertek=25
kulcs=l, ertek=45
kulcs=m, ertek=17
kulcs=p, ertek=90
kulcs=s, ertek=8
kulcs=h, ertek=7
kulcs=i, ertek=99
kulcs=g, ertek=77
kulcs=x, ertek=73
kulcs=y, ertek=21
kulcs=q, ertek=13

Rendezett értékek:
kulcs=z, ertek=1
kulcs=c, ertek=4
kulcs=h, ertek=7
kulcs=s, ertek=8
kulcs=t, ertek=10
kulcs=a, ertek=11
kulcs=q, ertek=13
kulcs=m, ertek=17
kulcs=y, ertek=21
kulcs=k, ertek=25
kulcs=d, ertek=32
kulcs=f, ertek=43
kulcs=l, ertek=45
kulcs=x, ertek=73
kulcs=g, ertek=77
kulcs=p, ertek=90
kulcs=i, ertek=99
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Godel$ █
```

## 17.4. Alternatív Tabella rendezése

Mutassuk be a [https://progpater.blog.hu/2011/03/11/alternativ\\_tabella](https://progpater.blog.hu/2011/03/11/alternativ_tabella) a programban a java.lang Interface Comparable

<T>

szerepét!

Megoldás videó:

Megoldás forrása: Bevezető: [https://hu.wikipedia.org/wiki/Alternativ\\_tabella](https://hu.wikipedia.org/wiki/Alternativ_tabella)

Tanulságok, tapasztalatok, magyarázat...

Tutor: <https://drive.google.com/file/d/1rvjmVVzlApo0n3aeaLBBZOZNXLKs1eTr/view?fbclid=IwAR1wQJZfDoXMpG0GMFKewS6QxyqFIPWtW2-s75liwhkQrwDizzNGPgM5pgI>  
Tutor: Kincs Ákos

A feladat megoldásához alternatív tabellát kell készítenünk. Nézzük mi az alternatív tabella: Ezt jellemzően futballbajnokságokhoz készítenek ez egyfajta sorrend viszont ez nem a győzelem: 3 pont döntetlen: 1 pont és a vereség: 0 pont szerint jár el, hanem figyelembe veszi azt, hogy ki ellen érte el az eredményt, ez tulajdonképpen azért, mert ha egy esetleges győzelmet éppen az aktuális rivális ellen szerzünk, akkor az számunkra is sokkal értékesebb lehet.

Ez az alternatív tabella a Google algoritmusán, mégpedig a PageRank-en alapszik. Ez nekünk annyit jelent, hogy az a csapat kerül előrébb, amely az első csapatok közül tud pontot/pontokat szerezni.

Kép az eredeti és az alternatív tabelláról:

| Hagyományos          | pont | Alternatív           | rang   |
|----------------------|------|----------------------|--------|
| Videoton             | 40   | Videoton             | 0,0841 |
| Ferencváros          | 34   | Debreceni VSC        | 0,0828 |
| Paks                 | 31   | Paksi FC             | 0,0725 |
| Debreceni VSZC       | 31   | BFC Siófok           | 0,0698 |
| Zalaegerszegi TE     | 30   | Budapest Honvéd      | 0,0698 |
| Kaposvári Rákóczi    | 29   | Ferencváros          | 0,0689 |
| Lombard Pápa         | 27   | Győri ETO            | 0,0650 |
| Kecskeméti TE        | 24   | Újpest               | 0,0636 |
| Újpest               | 23   | Zalaegerszegi TE     | 0,0636 |
| Győri ETO            | 23   | MTK Budapest         | 0,0596 |
| Budapest Honvéd      | 22   | Kaposvári Rákóczi    | 0,0570 |
| MTK Budapest         | 22   | Lombard Pápa         | 0,0561 |
| Vasas                | 21   | Szombathelyi Haladás | 0,0559 |
| Szombathelyi Haladás | 20   | Vasas                | 0,0543 |
| BFC Siófok           | 18   | Kecskeméti TE        | 0,0439 |
| Szolnoki MÁV         | 9    | Szolnoki MÁV FC      | 0,0330 |

Kép forrása: [link](#)

Nézzük a feladatunkat:

```
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;
        }
        return 0;
    }
}
```

```
        return 1;
    } else {
        return 0;
    }
}
```

Láthatjuk a java.lang package Comparable interface-t ezt pedig a Csapat osztály fogja implementálni. Az interface azért fontos, hogy tudjuk a függvényeket listákra és tömbökre is alkalmazni. Ahhoz hogy a két objektumot összehasonlítsuk szükségünk lesz a compareTo függvényre így az aktuális a hívott fügvénnyel össze tudjuk hasonlítani.

Láthatjuk, ha a hívó értékünk az kisebb mint az amit paraméterként átadott objektumé, akkor a visszatérési érték -1, ugyanez csak fordítva viszont nyilvánbalóan 1-es értékkel tér vissza. Ha megegyeznek akkor pedig a visszatérési érték 0.

Nézzük ez miért is jó nekünk:

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList( ←
    csapatok);
java.util.Collections.sort(rendezettCsapatok);
```

Nézzük mi is történik itt pontosan: Létrejön a Csapat típusú rendezettCsapatok nevű lista, melyet a sort metódussal rendezni fogunk. Ez a metódus azonban csak olyan listákon működnek, melyeknek a tagjai azok implementálják a Comparable interfacet.

Programunk a fordítás és futtatás után:

```
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Godel/Alternativ$ javac AlternativTabella.java
huri@huri-VirtualBox:~/bhax/thematic_tutorials/bhax_textbook/codes/Godel/Alternativ$ java AlternativTabella
iteracio...
norma = 0.05918759643574294
psszeg = 1.0
iteracio...
norma = 0.014871507967965858
psszeg = 0.9999999999999999
iteracio...
norma = 0.006686056768932613
psszeg = 1.0
iteracio...
norma = 0.007776749198562133
psszeg = 1.0
iteracio...
norma = 0.007661483555477314
psszeg = 0.9999999999999999
iteracio...
norma = 0.007501657116770109
psszeg = 0.9999999999999998
iteracio...
norma = 0.007532790726590474
psszeg = 0.9999999999999998
iteracio...
norma = 0.007538144256251714
psszeg = 0.9999999999999998
iteracio...
norma = 0.007535025315190922
psszeg = 1.0
iteracio...
norma = 0.00753510193655861
psszeg = 0.9999999999999997
iteracio...
norma = 0.0075353297234758716
psszeg = 0.9999999999999998
iteracio...
norma = 0.007535284725802345
psszeg = 0.9999999999999999
iteracio...
norma = 0.007535275242694286
psszeg = 0.9999999999999996
iteracio...
```

```
Csapatok rendezve:  
| -  
| Videoton  
| 40  
| Videoton  
| 0.0841  
| -  
| Ferencvaros  
| 34  
| Debreceni VSC  
| 0.0828  
| -  
| Paks  
| 31  
| Paksi FC  
| 0.0725  
| -  
| Debreceni VSC  
| 31  
| BFC Siofok  
| 0.0698  
| -  
| Zalaegerszegi TE  
| 30  
| Budapest Honved  
| 0.0697  
| -  
| Kaposvari Rakoczi  
| 29  
| Ferencvaros  
| 0.0689  
| -  
| Lombard Papa  
| 27  
| Gyori ETO  
| 0.0649  
| -  
| Kecskemeti TE  
| 24  
| Ujpest  
| 0.0636
```

```
| Ujpest
| 23
| Zalaegerszegi TE
| 0.0636
|
| Gyori ETO
| 23
| MTK Budapest
| 0.0595
|
| Budapest Honved
| 22
| Kaposvari Rakoczi
| 0.0570
|
| MTK Budapest
| 22
| Lombard Papa
| 0.0560
|
| Vasas
| 21
| Szombathelyi Haladas
| 0.0559
|
| Szombathelyi Haladas
| 20
| Vasas
| 0.0543
|
| BFC Siofok
| 18
| Kecskemeti TE
| 0.0439
|
| Szolnoki MAV
| 9
| Szolnoki MAV FC
| 0.0329
| -
```

## 17.5. Prolog családfa

Ágyazd be a Prolog családfa programot C++ vagy Java programba! Lásd para\_prog\_guide.pdf!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 17.6. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témat (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

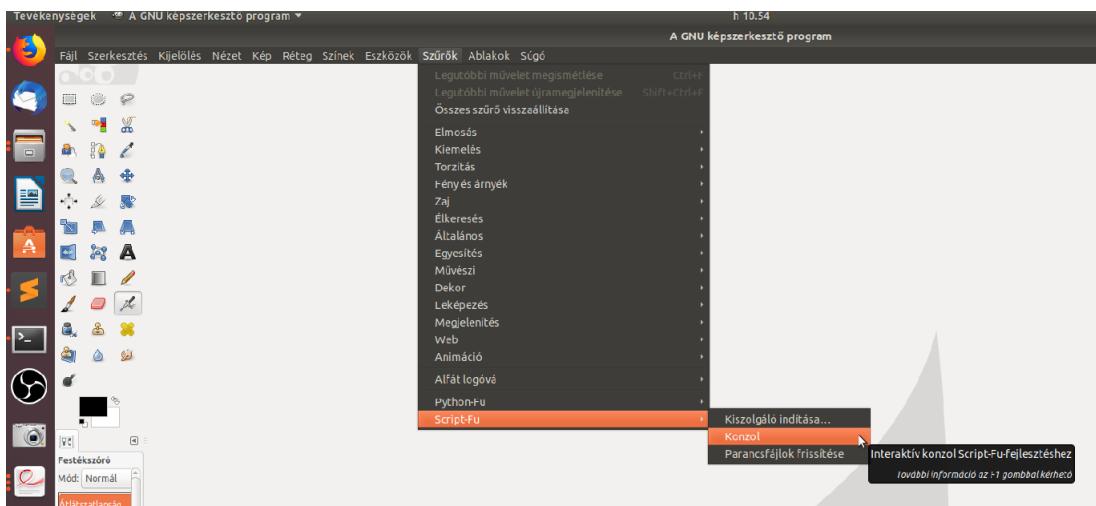
Megoldás videó:

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)

Tanulságok, tapasztalatok, magyarázat...

A feladatban a Lips programnyelvvel fogunk foglalkozni ezen belül is a Scheme nyelvvel. Ez a nyelv 1970-ben jelent meg, de mind a mai napig használjk.

A feladatunkban a Script-Fu-t használjuk, arra, hogy a GIMP képszerkeztő programhoz írunk egy scriptet ami lehetővé teszi a szövegünknek az effektekézését.



Nézzük a .scm fájl forrását:

```
(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)
)
```

Definiálásra kerül a color-curve függvény, ezzel pedig egy tömb kerül feltöltésre. Ennek a tömbnek nyolc különböző értéje van.

Nézzük tovább a forráskódot:

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

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

(list text-width text-height)
)
)

(define (script-fu-phax-chrome text 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-width (car (text-wh text font fontsize)))
    (text-height (elem 2 (text-wh text font fontsize)))
    (layer2)
)
)
```

Tovább haladva két függvényt látunk, van egy elem függvény, ami a megadott listából visszaadja a paraméterként megadott sorszámu elemet. A másik függvény a text-wh ennek a függvénynek a visszatérési értéke egy lista amiben szerepel a fotnak a szélessége valamint a magassága is.

Ezt követően definiáljuk a scriptet, és itt észrevehetjük azt, hogy a scriptnek a nevét szóközzel elválasztva fogják követni a paramétereink. Tovább haladva láthatjuk a let\* függvényt ez pedig két részből áll. A varlist részben azokat a változókat deklaráljuk, amikre a későbbiekben még hasznunkra lehet. A car függvény pedig a GIMP függvények és eljárások által ad vissza a listából, és ennek a listának az első elemét fogja nekünk visszaadni.

Nézzük tovább:

```
;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND )
(gimp-context-set-foreground '(255 255 255))
```

A szöveget amihez hozzá szeretnénk nálni beállítunk neki fehért színt valamint egy fekete háttér. Ahhoz, hogy meg tudjuk oldani a feladatot, a GIMP-nek az eljárásböngszőjét kell használnunk, és ha beírjuk a szükséges függvényt a keresőbe akkor sok információval leszünk gazdagabbak. A gimp-image-insert-layer függvény például egy új réteget ad hozzá a képhez a paraméterek pedig: kép, réteg, szülő réteg( ez nincs tehát ide 0 kerül.), és pozíció( ez egyébként itt is 0. mert azt a réteget szeretnénk ami legfelül van.).

Az előtér színhez szükségünk van a gimp-context-set-foreground függvényre, és mivel ez fekete ezért az RGB színt 0,0,0-át adunk meg. A gimp-drawable-filllayer függvénnyel kitöljtük előtérszínnel, majd a ..set-foreground függvénnyel fehérre állítjuk az előszínt.

```
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
    height 2) (/ text-height 2)))

(set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←
    LAYER)))
```

A gimp-layer-set-offsets -el a réteget középre tudjuk helyezni. Amit az utolsó sorban látjuk a gimp-image-merge-down függvénytel a rétegeinket "összefűlhetjük" ezt pedig a layerben tároljuk el.

Nézzük meg mi történt:



Nézzük mi történik a Gauss-elmosás

```
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```



A következő lépésekben a színekkel fogunk foglalkozni.

```
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
```

Megadjuk, hogy a műveletet a rétegen végezze el, ezt követően már a színeket kell állítani, nézzük hogyan néz ez ki:



Ismét Gauss-elmosást hajtunk végre, azonban most a sugár kissebb mint az előző esetben, nézzük:



A következő lépés már egy kicsit érdekesebb. Itt szín szerinti kijelölés lesz és ezt a kijelölést invertáljuk is

```
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))  
(gimp-selection-invert image)
```

A kijelölés a gimp-image-select-color-al történik a gimp-selection-invert pedig a kijelölést megfordítja. nézzük:



Ebben a lépésben létrehozzuk az átlátszó kijelölést:

```
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←  
    LAYER-MODE-NORMAL-LEGACY)))  
(gimp-image-insert-layer image layer2 0 0)
```



Most ezt az átlátszó kijelölést ki fogjuk tölni egy átmenettel:

```
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT- ↵
    LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ↵
        3)))
```



Ezt követően buckaleképzést fogunk alkalmazni a következőképpen:

```
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ↵
    0 TRUE FALSE 2)
```

A layer2-n hozzuk létre a buckaleképezést, úgy hogy a másik réteget felhasználjuk bemenetként.



Az utolsó lépésben a színbögrékkel kell jatszanunk, nézzük hogyan:

```
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))
```

Itt egyfajta fényesebb hatást szeretnénk elérni, és úgy tűnik ez sikerült is nézzük:



Nézzük még kicsit a scriptet:

```
(script-fu-register "script-fu-bhax-chrome"
  "Chrome3"
  "Creates a chrome effect on a given text."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 19, 2019"
  ""
  SF-STRING      "Text"        "Bátf41 Haxor"
  SF-FONT        "Font"        "Sans"
  SF-ADJUSTMENT  "Font size"   '(100 1 1000 1 10 0 1)
  SF-VALUE       "Width"       "1000"
  SF-VALUE       "Height"      "1000"
  SF-COLOR       "Color"       '(255 0 0)
  SF-GRADIENT    "Gradient"   "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```

A script-fu-register függvény segítségével elérhetjük azt, hogy könnyebben használjuk a scriptet és nem szeretnénk használni a script konzolt. Ennek a függvénynek megadtuk az értékeket is. Végszóként a megírásra kerül a menübe regisztráló függvény, ami annyit tesz, hogy a menüből kitudjuk választani grafikusan a scriptet amit használni szeretnénk.

## 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:

Tanulságok, tapasztalatok, magyarázat...

#### 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:

Tanulságok, tapasztalatok, magyarázat...

#### 18.3. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/Samu>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

---

## 18.4. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esport-talent-search>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 18.5. OSM térképre rajzolása

Debrecen térképre dobunk rá cuccokat, ennek mintájára, ahol én az országba helyeztem el a DEAC hekkereket: <https://www.twitch.tv/videos/182262537> (de az OOCWC Java Swinges megjelenítőjéből: <https://github.com/emulator/tree/master/justine/rcwin> is kiindulhatsz, mondjuk az komplexebb, mert ott időfejlődés is van...)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 19. fejezet

# Helló, Schwarzenegger!

### 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:

Tanulságok, tapasztalatok, magyarázat...

### 19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 19.3. Android Játék

Írunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Helló, Android!” feladatára!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 19.4. Junit teszt

A [https://propater.blog.hu/2011/03/05/labormeres\\_othon\\_avagy\\_hogyan\\_dolgozok\\_fel\\_egy\\_pedat\\_poszt\\_kézzel\\_számított\\_mélységét\\_és\\_szórását\\_dolgozd\\_be\\_egy\\_Junit\\_tesztbe](https://propater.blog.hu/2011/03/05/labormeres_othon_avagy_hogyan_dolgozok_fel_egy_pedat_poszt_kézzel_számított_mélységét_és_szórását_dolgozd_be_egy_Junit_tesztbe) poszt kézzel számított mélységét és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 19.5. OSCI

Készíts egyszerű C++/OpenGL-es megjelenítőt, amiben egy kocsit irányítasz az úton. A kocsi állapotát minden pillanatban mentsd le. Ezeket add át egy Prolog programnak, ami egyszerű reflex ágensként adjon vezérlést a kocsinak, hasonlítsd össze a kézi és a Prolog-os vezérlést. Módosítsd úgy a programodat, hogy ne csak kézzel lehessen vezérelni a kocsit, hanem a Prolog reflex ágens vezérelje!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 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:

Tanulságok, tapasztalatok, magyarázat...

### 20.2. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a <https://arato.inf.unideb.hu/batfai.norbert/deeplearning/mnist.html> fóliáját!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 20.3. CIFAR-10

Az alap feladat megoldása, +saját fotót is ismerjen fel, [https://progpater.blog.hu/2016/12/10/hello\\_samu\\_a\\_cifar-10\\_tf\\_tutorial\\_peldabol](https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tf_tutorial_peldabol)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 20.4. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 20.5. SMNIST for Machines

Készíts saját modellt, vagy használj meglévőt, lásd: <https://arxiv.org/abs/1906.12213>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 20.6. Minecraft MALMO-s példa

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lásd pl.: <https://youtu.be/bAPSu3Rndl8>, [https://bhaxor.blog.hu/2018/11/29/eddig\\_csaltunk\\_de\\_innen\\_tol\\_mi](https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innen_tol_mi), <https://bhaxor.blog.hu/2018/10/28/minecraft>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## **IV. rész**

### **Irodalomjegyzék**

## 20.7. Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

## 20.8. C

[KERNIGHANRITCHIE] Kernighan, Brian W. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

## 20.9. C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

## 20.10. 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.