

Univerzális programozás

Egy programozós könyv Tony Stark tollából.

Ed. Dékány Róbert, Debre-
cen, 2019. december 11, v.
0.2.11

Copyright © 2019 Dr. Bátfai Norbert

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>
WRITTEN BY	Dékány, Róbert Zsolt	2019. december 2.

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 https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-02-25	Turing csokor kész.	dekrob
0.0.6	2019-03-04	Chomsky csokor elkezdve/kész.	dekrob
0.0.7	2019-03-11	Caesar csokor elkezdve.	dekrob
0.0.8	2019-03-20	Mandelbrot csokor kész.	dekrob

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-03-24	Guttenberg csokor elkezdve.	dekrob
0.1.0	2019-04-03	Welch csokor elkezdve majdnem kész.	dekrob
0.1.1	2019-04-08	Conway / Swarzenegger csokor elkezdve.	dekrob
0.1.2	2019-04-16	Gutenberg folytatás.	dekrob
0.1.3	2019-05-10	Utolsó simítások a könyvön. Szerkesztések.	dekrob
0.2.1	2019-09-15	Helló, Berners - Lee! olvasónapló kész.	dekrob
0.2.2	2019-09-22	Helló, Arroway! csokor kész.	dekrob
0.2.3	2019-09-29	Helló, Liskov! csokor kész.	dekrob
0.2.4	2019-10-06	Helló, Mandelbrot! csokor kész.	dekrob
0.2.5	2019-10-13	Helló, Chomsky! csokor kész.	dekrob
0.2.6	2019-10-20	Helló, Stroustrup! csokor kész.	dekrob
0.2.7	2019-11-03	Helló, Gödel! csokor kész.	dekrob
0.2.8	2019-11-10	Helló, Valami! csokor kész.	dekrob

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.2.9	2019-11-17	Helló, Lauda! csokor kész.	dekrob
0.2.10	2019-11-24	Helló, Calvin! csokor kész.	dekrob
0.2.11	2019-12-01	Végső simítások a könyvön!	dekrob

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

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
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	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun téTEL	14
2.8. A Monty Hall probléma	16
3. Helló, Chomsky!	19
3.1. Decimálisból unárisba átváltó Turing gép	19
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	20
3.3. Hivatalos nyelv	21
3.4. Saját lexikális elemző	22
3.5. l33t.l	22
3.6. A források olvasása	25
3.7. Logikus	26
3.8. Deklaráció	27

4. Helló, Caesar!	29
4.1. double ** háromszögmátrix	29
4.2. C EXOR titkosító	31
4.3. Java EXOR titkosító	33
4.4. C EXOR törő	34
4.5. Neurális OR, AND és EXOR kapu	37
4.6. Hiba-visszaterjesztéses perceptron	37
5. Helló, Mandelbrot!	38
5.1. A Mandelbrot halmaz	38
5.2. A Mandelbrot halmaz a std::complex osztállyal	41
5.3. Biomorfok	42
5.4. A Mandelbrot halmaz CUDA megvalósítása	45
5.5. Mandelbrot nagyító és utazó C++ nyelven	45
5.6. Mandelbrot nagyító és utazó Java nyelven	49
6. Helló, Welch!	53
6.1. Első osztályom	53
6.2. LZW	54
6.3. Fabejárás	63
6.4. Tag a gyökér	66
6.5. Mutató a gyökér	70
6.6. Mozgató szemantika	74
7. Helló, Conway!	87
7.1. Hangyszimulációk	87
7.2. Java életjáték	89
7.3. Qt C++ életjáték	94
7.4. BrainB Benchmark	101
8. Helló, Schwarzenegger!	102
8.1. Szoftmax Py MNIST	102
8.2. Mély MNIST	106
8.3. Minecraft-MALMÖ	111

9. Helló, Chaitin!	112
9.1. Iteratív és rekurzív faktoriális Lisp-ben	112
9.2. Gimp Scheme Script-fu: króm effekt	112
9.3. Gimp Scheme Script-fu: név mandala	118
10. Helló, Gutenberg!	124
10.1. PICI Juhász István	124
10.2. Programozás bevezetés	125
10.3. Programozás	126
III. Második felvonás	127
11. Helló, Olvasónapló!	129
11.1. Java illetve C++ összehasonlítása	129
11.2. Python könyv feldolgozása	131
12. Helló, Arroway!	134
12.1. Az objektumorientált paradigmá alapfoglamai. Osztály, objektum, példányosítás.	134
12.2. OO Szemlélet	134
12.3. Homokozó	137
12.4. „Gagyi”	146
12.5. Yoda	149
12.6. Kódolás from scratch	150
13. Helló, Liskov!	153
13.1. Öröklődés, osztályhierarchia. Polimorfizmus, metódustúlerhelés. Hatáskörkezelés. A be-zárási eszközrendszer, láthatósági szintek. Absztrakt osztályok és interfések.	153
13.2. Liskov helyettesítés sértése	153
13.3. Szülő-gyerek	156
13.4. Anti OO	158
13.5. Hello, Android!	168
13.6. Ciklomatikus komplexitás	173

14. Helló, Mandelbrot!	175
14.1. Modellező eszközök és nyelvek. AZ UML és az UML osztálydiagramja.	175
14.2. Reverse engineering UML osztálydiagram	175
14.3. Forward engineering UML osztálydiagram	176
14.4. Egy esettan	181
14.5. BPMN	183
14.6. TeX UML	184
15. Helló, Chomsky!	187
15.1. Objektumorientált programozási nyelvek programnyelvi elemei: karakterkészlet, lexikális egységek, kifejezések, utasítások.	187
15.2. Encoding	187
15.3. Maurer Rose - full screen	190
15.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció	193
15.5. Perceptron osztály	196
16. Helló, Stroustrup!	199
16.1. Objektumorientált programozási nyelvek típusrendszere (pl.: Java, C#) és 6. hét Típusok tagjai: mezők, (nevesített) konstansok, tulajdonságok, metódusok, események, operátorok, indexelők, konstruktorok, destruktörök, beágyazott típusok.	199
16.2. JDK osztályok	199
16.3. Másoló-mozgató szemantika	202
16.4. Hibásan implementált RSA törése	205
16.5. Változó argumentumszámú ctor	211
16.6. Összefoglaló	214
17. Helló, Godel!	216
17.1. Interfészök. Kölcsönös funkcionális nyelvi elemek. Lambda kifejezések.	216
17.2. Gengszterek	216
17.3. C++11 Custom Allocator	217
17.4. STL map érték szerinti rendezése	217
17.5. Alternatív Tabella rendezése	218
17.6. GIMP Scheme	218
17.7. PHP	225

18. Helló, Valami!	230
18.1. Adatfolyamok kezelése, streamek. I/O állománykezelés. Szerializáció.	230
18.2. FUTURE tevékenység editor	230
18.3. OOCWC Boost ASIO hálózatkezelése	232
18.4. BrainB	233
18.5. OSM térképre rajzolása	234
19. Helló, Lauda!	239
19.1. Kivételkezelés. A fordítást és a kódgenerálást támogató nyelvi elemek (annotációk, attribútumok).	239
19.2. Portscan	239
19.3. Androidos játék	241
19.4. JUnit Teszt	249
19.5. AOP	250
20. Helló, Calvin!	253
20.1. Multiparadigmás nyelvek. Programozás multiparadigmás nyelveken.	253
20.2. MNIST	253
20.3. DEEP MNIST	258
20.4. CIFAR10	265
20.5. Android telefonra TF objektum detektálója	272
IV. Irodalomjegyzék	279
20.6. Általános	280
20.7. C	280
20.8. C++	280
20.9. Lisp	280

Ábrák jegyzéke

2.1. PageRank algoritmus	12
2.2. Bátfai Tanár úr ábrája a Brun-tétel abrázolásáról.	16
3.1. A Turing-gép állapotátmeneti gráfjának egyik rajza.	19
4.1. Bátfai Norbert ábrája a double**háromszögmátrixról az én saját címeimmel.	30
6.1. Bejaras	65
7.1. Hangyszimuláció UML	88
7.2. Bátfai Norbert ábrája a BrainB-ről.	101
8.1. Bátfai tanár úr ábrája a megjelenített számokról a MNIST-ben.	106
11.1. Java compile	129
11.2. Java könyvből való típus összegző táblázat	130
11.3. Többszörös öröklődés	131
12.1. PolarGen program eredménye	136
12.2. Random.class-ban lévő nextGaussian() függvény	137
12.3. LZWBinfa megjelenítése böngészőben.	146
12.4. JDK Integer.java	148
12.5. NullPointerException, ha nem használjuk a Yoda Conditiont.	149
12.6. BBP algoritmus output.	152
13.1. ThiefBank helytelenül működik.	155
13.2. Java-féle futási idők (sec)	160
13.3. C-féle futási idők (sec)	163
13.4. C#-féle futási idők (sec)	165
13.5. C++-féle futási idők (sec)	167

13.6. Összefoglaló táblázat a futási időkről	167
13.7. Emulátor hardver tulajdonságai	168
13.8. Emulátor egyéb tulajdonságai	169
13.9. Android Emulátor eszközök listája	170
13.10SMNIST error	172
13.11PiBBP.java ciklomatikus komplexitása	173
13.12Binfa.java ciklomatikus komplexitása	174
 14.1. Binfa UML osztálydiagram	176
14.2. Termékek létrehozásáért felelős osztályok kapcsolata.	182
14.3. Az programunk output-ja.	182
14.4. computerproducts_out.txt file-ba való exportálás.	183
14.5. BPMN modell.	184
14.6. Real-Time Traffic Analyzer Figure made by LaTeX.	186
 15.1. Mandelbrot nagyító IntelliJ-ben.	188
15.2. File Encoding	189
15.3. javac error a Mandelbrotra	190
15.4. Ismét lefutott.	190
15.5. Maurer Rose Figure.	193
15.6. Módosított OpenGL vizualizáció.	195
15.7. Módosított Mandelbrot-halmaz képe.	198
 16.1. JDK osztályok	201
16.2. A main eredményei	205
16.3. Végeredmény a hibásan implementált RSA-ra.	211
16.4. Végeredmény saját gyakorisággal.	211
16.5. Mint látható megegyezik a két kép mérete.	214
16.6. RSA működése.	215
 17.1. GIMP scheme által kreált króm szöveg.	225
17.2. Tesztkód a hekkerek kiiratására.	229
 18.1. Modified Activity Editor	232
18.2. GPS Tracker Android Application	238
 19.1. Portscan Eredmény localhostra.	241

19.2. Unity Editor	242
19.3. Plane objektumra alkalmazott rácsos material.	243
19.4. Kész Androidos játék.	249
19.5. Sikeres teszt.	250
19.6. Sikertelen teszt.	250
19.7. Aspect output.txt	252
20.1. MNIST dataset.	254
20.2. NN	255
20.3. Hálózat tanítása és tesztelése.	257
20.4. Felismert számok.	258
20.5. Convolutional Neural Network (CNN) ábra	259
20.6. Hasonló eredmény a deep CNN változattal.	263
20.7. TensorBoard gráf ábra.	264
20.8. TensorBoard eloszlás ábra.	265
20.9. CIFAR10 osztályok.	266
20.10Modell szerkezete.	268
20.11Automobile kép.	270
20.12Szarvas kép.	270
20.13IFA kép.	271
20.14CIFAR10 eredmény.	271
20.15Kutyánk felismerése.	273
20.16Óra felismerése.	275
20.17Váza felismerése.	276
20.18Ukulele felismerése.	278

Előszó

A legjobb dolog a programozásba az alkotás szabadsága, az új érték megteremtése. Egy hétköznapi ember fel se tudja fogni, hogy mennyi lehetőség rejlik a programozói szakmában és ez teszi oly vonzóvá. - Dékány Róbert Zsolt

Hogyan forgasd

A könyv célja egy stabil programozási szemlélet kialakítása az olvasóban. Módszere, hogy hetekre bontva ad egy tematikus feladatesokrot. 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.

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 másit 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 csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Végtelen ciklus, egy olyan "állatfajta" a programozás világában, amivel a kisgyerekeket riogatják az iskolákban. Jobb elkerülni, de akarva vagy akaratlanul néha bele-belefutunk. Ha még nem láttál végtelen ciklust, íme egy C nyelven:

```
int main ()
{
    for (;;);
    return 0;
}
```

A fenti kódot a `gcc inf.c -o inf` parancsal fordítjuk, majd `./inf`-el futtatjuk. Ha szépen lefutott a `top` parancs segítségével megtudjuk nézni a CPU állapotát! A futtatott programunk sorában tisztán látszik, hogy a CPU 100%-on pörög.

Ha egy olyan végtelen ciklust szeretnénk, ahol a CPU terheltsége a 0%-hoz közelí, akkor a `sleep (seconds)` függvényt kell használnunk a ciklusmagban. minden egyes ciklus lefutásnál a programszál "altatva" van a `sleep` paraméteréül megadott x másodpercig, így elérve a 0% CPU állapotot.

```
#include <unistd.h>

int main ()
{
    for (;;)
    {
        sleep (1);
    }

    return 0;
}
```

```
}
```

Minden magot 100%-on terhelni többszálaszított végtelen ciklussal tudunk. Ehhez a `#pragma omp parallel` utasítást kell alkalmaznunk a `for` ciklusra, majd a `gcc teszt-feladat.c -o teszt -fopenmp` parancssal fordítjuk. A `top` utasítással le tudjuk ellenőrizni, hogy valóban minden mag 100%-on pörög.

```
int main ()
{
    #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 pszeudókódjára:

```
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.

A lefagy vagy nem fagy le program lényegében arról szól, hogy képesek vagyunk e olyan programot írni, ami eldönti egy programról, hogy be fog-e fagyni. Alan Turing matematikus és programozó 1936 decemberében bebizonyította, hogy ezen program írása nem lehetséges. Technikailag egy kisebb programról el tudjuk dönteni, hogy van-e benne például egy végtelen ciklus, ami a program fagyásához vezethet. Azonban ha nagyobb és bonyolultabb programba gondolkodunk ez szinte lehetetlen.

Talán a jövőben megoldható lesz ez a probléma, ki tudja.

2.3. Változók értékének felcserélése

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

Számtalan olyan feladat vagy probléma létezik, ahol két változók kell felcserélni majd ezekkel dolgozni tovább. Programozókként az egyszerű s nagyszerű elvnek megfelelően 2 különböző módszert nézünk meg a változók felcserélésére.

I. Segédváltozós csere:

```
#include <stdio.h>

int main ()
{
    int a = 99; int b = 45;

    int c = a;
    a = b;
    b = c;

    printf("A értéke: %d\n", a);
    printf("B értéke: %d\n", b);

    return 0;
}
```

Rém egyszerű a történet. C segédváltozónak értékül adjuk magát az A értékét, majd az A értékét egyenlővé teszük a B-vel. Végül B értékét egyenlő lesz C értékével.

II. Segédváltozó nélkül egy kis matekkal:

```
#include <stdio.h>

int main ()
{
    int a = 99; int b = 45;

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

```
b = a - b;  
  
printf("A értéke: %d\n", a);  
printf("B értéke: %d\n", b);  
  
return 0;  
}
```

Egy kis összeadással és kivonással egyszerűen megoldható a két változó felcserélése, aki nem hiszi számolja ki.

Videó: https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk

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.)

Ebben a feladatban Nagy Martin-t tutoráltam!

Labda pattogtatásához elsőként a bejárandon területet kell definiálnunk a programban. A bejárandon terület a terminál ablakunk X és Y koordinátái fogják meghatározni. Ehhez felveszünk két konstans változót a `#define {VÁLTOZÓNÉV}` kulcsszó segítségével. Ez után szükség van egy `positionPrinting(int x, int y)` függvényre, ami a paraméterében átadott X és Y koordináták felhasználásával előállítja a labdakimenetet.

```
#include <math.h>  
#include <unistd.h>  
#include <stdio.h>  
#include <stdlib.h>  
  
#define WIDTH 78  
#define HEIGHT 22  
  
int positionPrinting(int x, int y)  
{  
    int i;  
  
    for(i=0; i<x; i++)  
        printf("\n");  
  
    for(i=0; i<y; i++)  
        printf(" ");  
  
    printf("\u26bd;\n");  
  
    return 0;
```

```
}
```

Ha mindez megvan elkészíthetjük a `main` metódusunkat. Itt két hosszú egész típusú változót kell inicializálnunk. Ezekben tároljuk az aktuális X és Y koordinátákat. Majd írunk egy végtelen ciklust. A ciklus magában először minden egyes lefutásnál töröljük a képernyőt a `system("clear")` függvényel. Ezután rajzolhatjuk a labdánkat az előbb elkészített `positionPrinting` (`abs(HEIGHT - (x++% (HEIGHT*2)))`), függvény segítségével. Majd az `usleep(55000)` függvényel "altatjuk". Ezzel implementáljuk le a látszólag folyamatos labdamozgást.

```
int main()
{
    long int x=0;
    long int y=0;

    while(1)
    {
        system("clear");

        positionPrinting(abs(HEIGHT - (x++% (HEIGHT*2))), abs(WIDTH - (y++% (←
            WIDTH*2))));

        usleep(55000);
    }

    return 0;
}
```

Ezen program után rádöbbenhetünk, hogy elég egy console és bármiféle egyszerűbb játékot leimplementálhatunk benne. Következőképp el tudnék képzelní egy console-os amőba vagy akasztófa játékot. Ha lesz rá időm meg is csinálom. :)

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

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!

Az alábbi Linus Torvalds féle BogoMIPS egy program, ami segít a hibakeresésben illetve ellenőrizhető a számítógép számítási teljesítménye. Ellenőrizve az egymillió utasítás per másodperc végrehajtását végül visszaad egy indexszámot, ami jellemzi a processzorteljesítményt.

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

```
void delay (unsigned long long int loops)
{
    unsigned long long int i;
    for (i = 0; i < loops; i++);
}

int main (void)
{
    unsigned long long int loops_per_sec = 1;
    unsigned long long int ticks;

    printf ("Calibrating delay loop..");
    fflush (stdout);

    while ((loops_per_sec <= 1))
    {
        ticks = clock ();
        delay (loops_per_sec);
        ticks = clock () - ticks;

        printf ("%llu %llu\n", ticks, loops_per_sec);

        if (ticks >= CLOCKS_PER_SEC)
        {
            loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

            printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / ←
                    500000,
                    (loops_per_sec / 5000) % 100);

            return 0;
        }
    }

    printf ("failed\n");

    return -1;
}
```

2.6. Helló, Google!

Írj olyan C programot, amely egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét! Középiskolában a (leg)népszerűbbnek számít az az ember, akinek sok barátja és ismerőse van. Kissé köz hely de igaz. Ezen példát továbbgondolva rájöhettünk a hasonlóságra, ha az interneten található weboldalak

népszerűségét figyeljük meg. Népszerű, ha az adott weboldala sok-sok link mutat. Ezen népszerűségi rangsorolást a Google-féle PageRank algoritmus végzi.

Az, hogy milyen "népszerű" egy adott oldal a PageRank algoritmus által hozzárendelt érték adja meg. Minél nagyobb annál népszerűbb. Továbbiakban tisztába kell lenni azzal, hogy egy weboldala kétféle link definiált. Vannak a rámutató (pointing) linkek, illetve a kimenő (outgoing) linkek. Pointing link az adott weboldala mutató linkek. Az outgoing linkek azon weboldal, ahova a weboldalunk mutat.

$$PR(h_2) = \sum_{h \in B(h_2)} \frac{PR(h)}{N(h)}$$

2.1. ábra. PageRank algoritmus

Elsőkörbe megírjuk a megjelenítő `kiir` függvényt. A paramétereként megkapott double vektoron végigiterál, majd kiiírja a PageRank értékeket az `output`-ra.

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

void kiir (vector<double> tomb)
{
    int i;
    for (i=0; i < tomb.size(); i++)
        printf("PageRank [%d]: %lf\n", i, tomb[i]);
}
```

`tavolsag` függvényt a paraméterben megkapott két vectorból számol egy közelítő távolságértéket. Ezt az értéket fogjuk vizsgálni a továbbiakban. Ha ez az érték közelíti vagy nagyobb a 0.00001 akkor nincs értelme tovább iterálni az a PR értékeket.

```
double tavolsag(vector<double> pagerank, vector<double> ←
    pagerank_temp)
{
    double tav = 0.0;
    int i;
    for(i=0;i < pagerank.size();i++)
        tav += abs(pagerank[i] - pagerank_temp[i]);
    return tav;
```

```
}
```

A main függvényben történik a program lényegi része. Elsőként deklarálunk egy 4x4-es mátrixot. Ebbe lesz eltárolva a mutató linkek szerinti értékek amikkel dolgozni fogunk. Egy PR vektorban fogjuk eltárolni az épp aktuális PageRank értékeket. A PRv vektorban az egyes iterációknál meghatározott PageRank értékeket. Alap esetben minden 1/4, mert 4 weblapunk van. Egy végtelen ciklusban fogjuk végrehajtani az egyes iterációs lépésekét. minden iterációban végigmegyünk a PR illetve a PRv elemeken és mátrixszorzással kiszámoltatjuk a PageRank értékeket. Ezeket a lépéseket addig ismételgetjük, amíg a tavolsag függvény vissza nem tér a megfelelő értékkel. A végén pedig a kiír -el kiiratunk.

```
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}
    };

    vector<double> PR = {0.0, 0.0, 0.0, 0.0};
    vector<double> PRv = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};

    long int i, j, h;
    i=0; j=0; h=5;

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

        for (i=0; i<4; i++)
        {
            double temp=0;

            for (j=0; j<4; j++)
                temp+=L[i][j]*PR[j];

            PRv[i]=temp;
        }

        if ( tavolsag(PR,PRv) < 0.00001)
            break;
    }

    kiir (PR);
}
```

```
    return 0;  
}  
}
```

2.7. 100 éves a Brun tétele

Írj R szimulációt a Brun tétele demonstrálására!

Ebben a feladatban Nagy Martin-t tutoráltam!

Bizonyára mindenki hallott már az prímek fogalmáról. Ezek olyan számok amelyek 1-el és önmagukkal oszthatóak csak. Az ikerprímek ezen prímszámokból vett prímek, amelyeknek különbsége kettő. A prímszámok olyan számok, melyek csak önmagukkal és eggyel osztva nem adnak maradékot. Az iker-prímek pedig olyan príszmámok, melyek különbsége kettő.

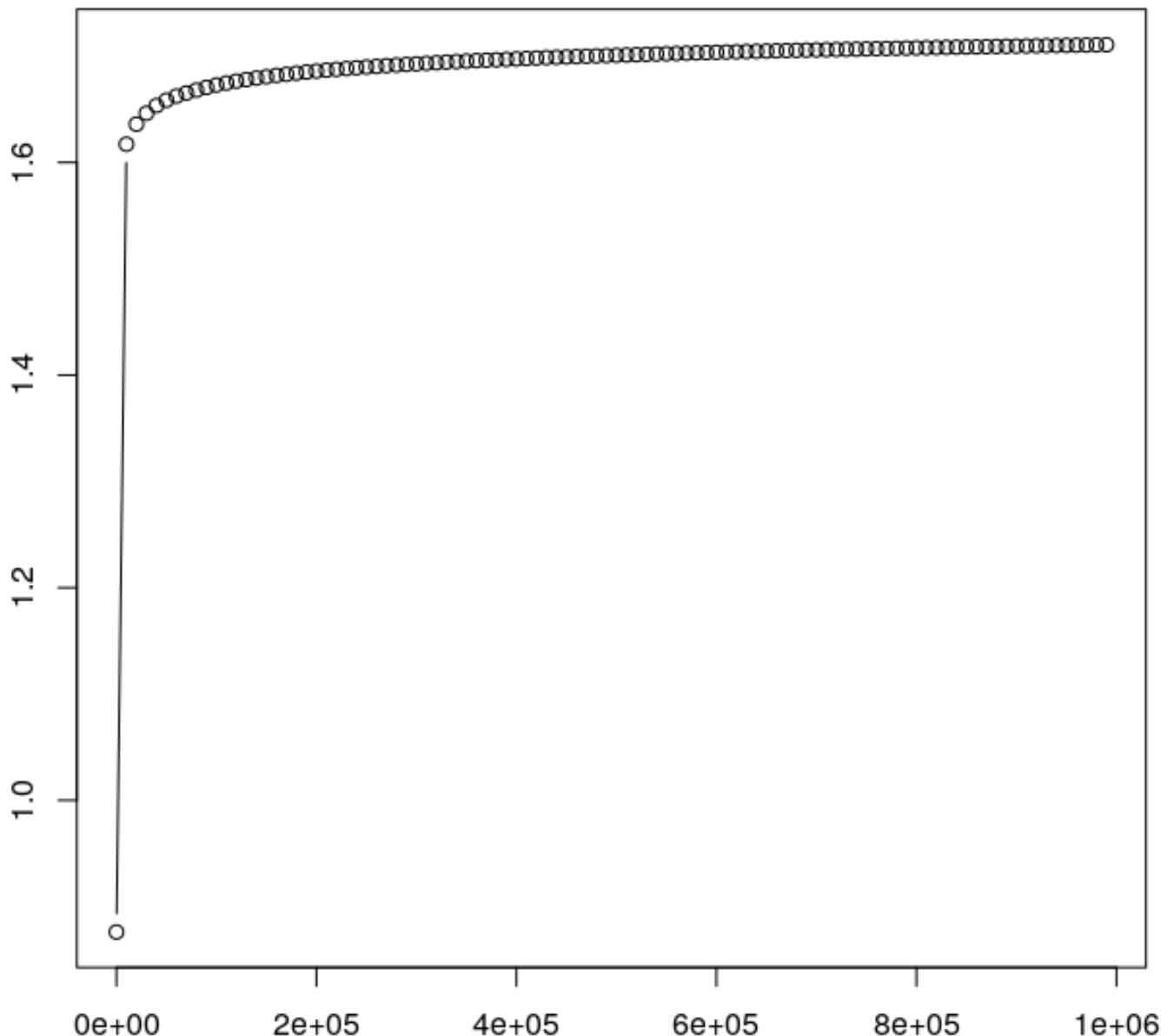
A feladatunk a Brun tétele értelmezéséről és felhasználásáról szól. A Brun tétel kimondja, hogy az ikerprímszámok reciprokát ha elkezdjük összeadogatni akkor az így kapott sor egy Brun-konstanshoz fog konvergálni. Ezen tételek mentén elkészítjük programunkat Matlabban majd ábrázoljuk.

```
# 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")
```

A function(X) függvény a prímelek végigiterál. Megnézi a külöbségüket, majd ha az kettő akkor eltárolja a két ikerprímszámot. Ezután összeadjuk a számokat majd felhasználjuk. Végén pedig a plot parancssal megrajzoltatjuk a grafikont.



2.2. ábra. Bátfai Tanár úr ábrája a Brun-tétel abrázolásáról.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

```
# An illustration written in R for the Monty Hall Problem
# 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/>
#
#   https://bhaxor.blog.hu/2019/01/03/erdos\_pal\_mit\_keresett\_a\_nagykonyvben\_a\_monty\_hall-paradoxon\_kapcsan
#
#   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])

  }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(holvart),1)]

}

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

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

Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan

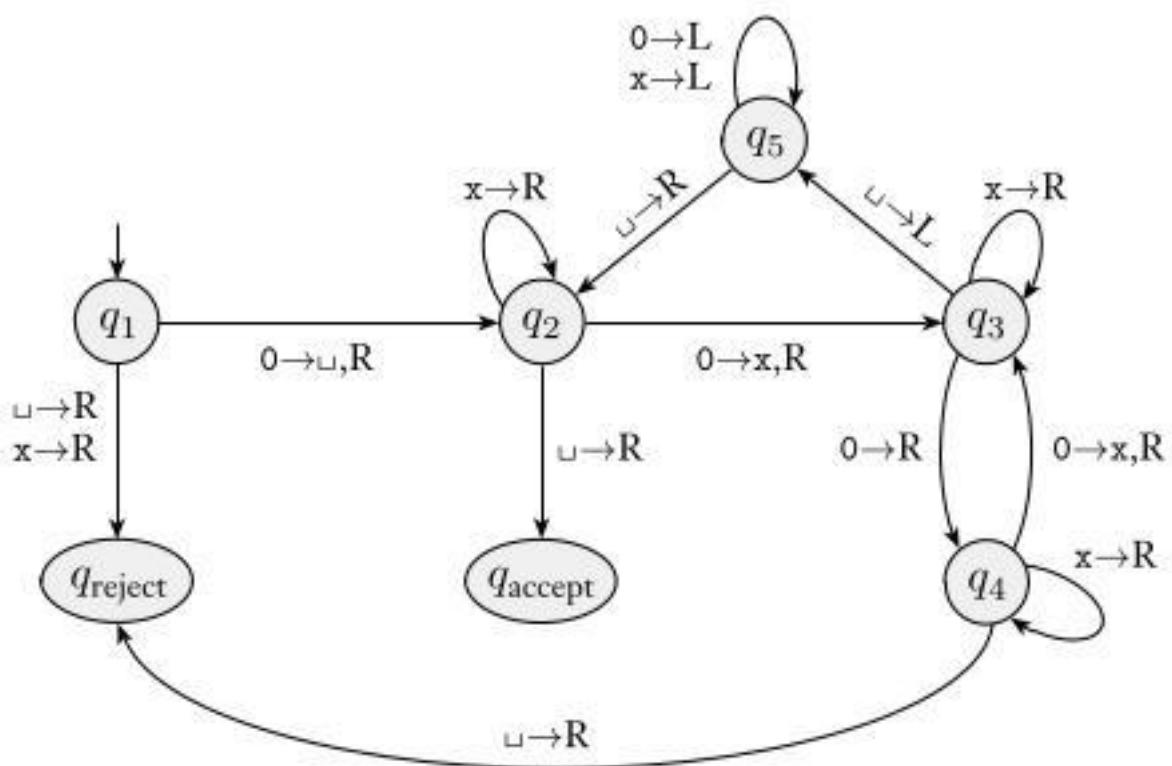
Megoldás később!

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!



3.1. ábra. A Turing-gép állapotátmeneti gráfjának egyik rajza.

Az unáris számrendszer a nevéről eredően az unáris (1) a legnagyobb számjegy. Alsós iskolákban a számolást segítette az újjakon való számolás. Mikor leszámoltunk 5-ig az ujjaink segítségével lényegében az is unáris számrendszerben történt. Az unáris Turing gép egységszakaszokban ábrázolja a számokat. Az

egységszakasz K szakaszra bontható. Ilyen pl. az 5 egységszakasz alapján 1 szakasz megfelel az 1-es szám-jegynek. Ezekhez a szakaszokhoz különböző szimbólumokat/jelentéseket rendelhetünk. Legáltalánosabb példa a vonalakkal jelzett 5-ös egységszakaszok.

C++ KÓD:

```
#include <iostream>

void toUnar(int a)
{
    for(int i=0; i<a; i++)
        std::cout << "1" << std::endl;
}

int main()
{
    int val;
    std::cout << "Type a number in decimal." << std::endl;

    while(std::cin >> val)
    {
        toUnar(val);
    }

    return 0;
}
```

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

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

A generatív nyelvtan megalkotása és elméletének lefektetése Noam Chomsky nyelvész nevéhez fűződik. A lejjebb található kód lényege, hogy miképp tudunk kezdőszimbólumunkból az általunk lefektetett szabály-gyűjtemény felhasználásával, konstansokból felépíteni mondatokat.

Az első:

```
S, X, Y változók
a, b, c konstansok
S -> abc, S -> aXbc, Xb -> bX, Xc -> Ybcc, bY -> Yb, aY -> aaX, aY -> aa
aa

S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
```

```

aYbbcc (aY - aaX)
aaXbbcc (Xb - bX)
aabXbcc (Xb - bX)
aabbXcc (Xc - Ybcc)
aabbYbcc (bY - Yb)
aabYbbccc (bY - Yb)
aaYbbbccc (aY - aa)
aaabbbccc

```

A második:

```

A, B, C változók
a, b, c konstansok
A = aAB, A = aC, CB = bCc, cB = Bc, C = bc

A (A = aAB)
aAB (A = aAB)
aaABB (A = aAB)
aaaABBB (A = aC)
aaaaCBBB (CB = bCc)
aaaabCcBB (cB = Bc)
aaaabCBcB (cB = Bc)
aaaabCBBc (CB = bCc)
aaaabbCcBc (cB = Bc)
aaaabbCBcc (CB = bCc)
aaaabbbCccc (C = bc)
aaaabbbbcccc

```

Miért is nem környezetfüggő? Azért, mert nincs olyan mondat, hogy a mondat elején csak nem terminális jel van.

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.

Különböző szabványok kisebb nagyobb eltéréseket mutathatnak, így jól kell tudnunk, hogy milyen szabványokkal is dolgozunk. Az alábbi kód a C99 szabvány alapján lefut viszont a C89-el nem. A kódunkat szokásos gcc-vel fordítjuk, de ahhoz, hogy megtudjuk nézni a különbségeket a `-std:c89` illetve a `-std:c99` kapcsolókat kell alkalmaznunk.

```

int main()
{
    for(int i = 0; i < 5; i++)
    {
        /* code */
    }
}

```

```
    }

    return 0;
}
```

Ha lefuttattuk C89 szabvány szerint, akkor az alábbi hibaüzenetet kaptuk: error: 'for' loop initial declarations are only allowed in C99 or C11 mode

A régi szabvány szerint a for fejlécében nem megengedett a változó deklaráció. Viszont, ha a deklarációt a foron kívülre helyezzük, akkor minden ok. Íme:

```
#include <unistd.h>

int main()
{
    int i;

    for(i = 0; i < 5; i++)
    {
        /* code */
    }

    return 0;
}
```

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!

3.5. l33t.l

Lexelj össze egy l33t cipher!

A megadott egyszerű szabályok segítségével elő tudunk állítani egy olyan kimenetet, ahol különféle karakterek(láncok) kicsérélődnek a szabályban megadott karakterekre. minden a szabályban specifikált bemenetre megcsinálja a cserét egyéb esetekben leírja a karakterláncot változatlanul.

```
/*
Forditas:
$ lex -o 1337d1c7.c 1337d1c7.l
```

Futtatas:

```
$ gcc 1337d1c7.c -o 1337d1c7 -lfl  
(kilépés az input vége, azaz Ctrl+D)
```

Copyright (C) 2019

Norbert Bátfai, 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/>>.

```
*/  
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <ctype.h>  
  
#define L337SIZE (sizeof 1337d1c7 / sizeof (struct cipher))  
  
struct cipher {  
    char c;  
    char *leet[4];  
} 1337d1c7 [] = {  
  
{'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  
};  
  
%}  
%%  
. {  
  
    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);

    }
%%

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

Kimenet lehet például:

```
~ hello
~ h3110
```

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);
```

Ha a SIGINT nem volt figyelmen kívül hagyva akkor a jelkezelő kezelje. Ha figyelmen kívül volt hagyva tövábbra is maradjon úgy.

ii.

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

A fejlécébe hiányzik az i deklaráció, ha ez megvan akkor működik.

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

A fejlécébe hiányzik az i deklaráció, ha ez megvan akkor működik.

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

A kód hibamentes, mert már létrehoztuk a változókat és mutatókat.

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

Ugyanaz, mint az előzőnél.

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

A printf függvény ki fog írni 2 decimális számot ha már megvan az f függvény, az a változó, és ha az a változó megfelelő típusú az f függvényhez. Arra kell figyelni hogy ha az f függvény visszatérési értéke nem int akkor a kiírt értékek nem biztos hogy pontosak lesznek.

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

A printf ki fogja írni az f függvény visszatérési értékét a-ra, és a értékét.

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

A kiiratás megtörténik viszont az f függvény most az a változó memória címével fog dolgozni nem az a értékével.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$ (\forall x \exists y ((x < y) \wedge (y \text{ prim}))) $  

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

$ (\exists y \forall x (x \text{ prim}) \supset (x < y)) $  

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, https://youtu.be/AJSXOQFF_wk

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a;
```

- egészre mutató mutató

```
int *b;
```

- egész referencia

```
int &c;
```

- egészek tömbje

```
int T[3];
```

- egészek tömbjének referencia (nem az első elemé)

```
int (&T)[3] = T;
```

- egészre mutató mutatók tömbje

```
int *T[3];
```

- egészre mutató mutatót visszaadó függvény

```
int *func();
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int *(*func)();
```

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

```
int *(*(*func))();
```

- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

```
int *(*(*func))();
```

Mit vezetnek be a programba a következő nevek?

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

Öt elemű tömb deklaráció.

- ```
int a;
```

Egész declaráció.

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

Egy pointer, ami az 'a' változóra mutat.

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

Az 'a' változó referenciaja.

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

Öt elemű tömb deklaráció.

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

A c tömbre referenciaja.

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

Egy int-re mutató 5 elemű pointer tömb.

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

Egy int-re mutató függvény pointer.

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

Egy int-re mutató függvénypointer pointere.

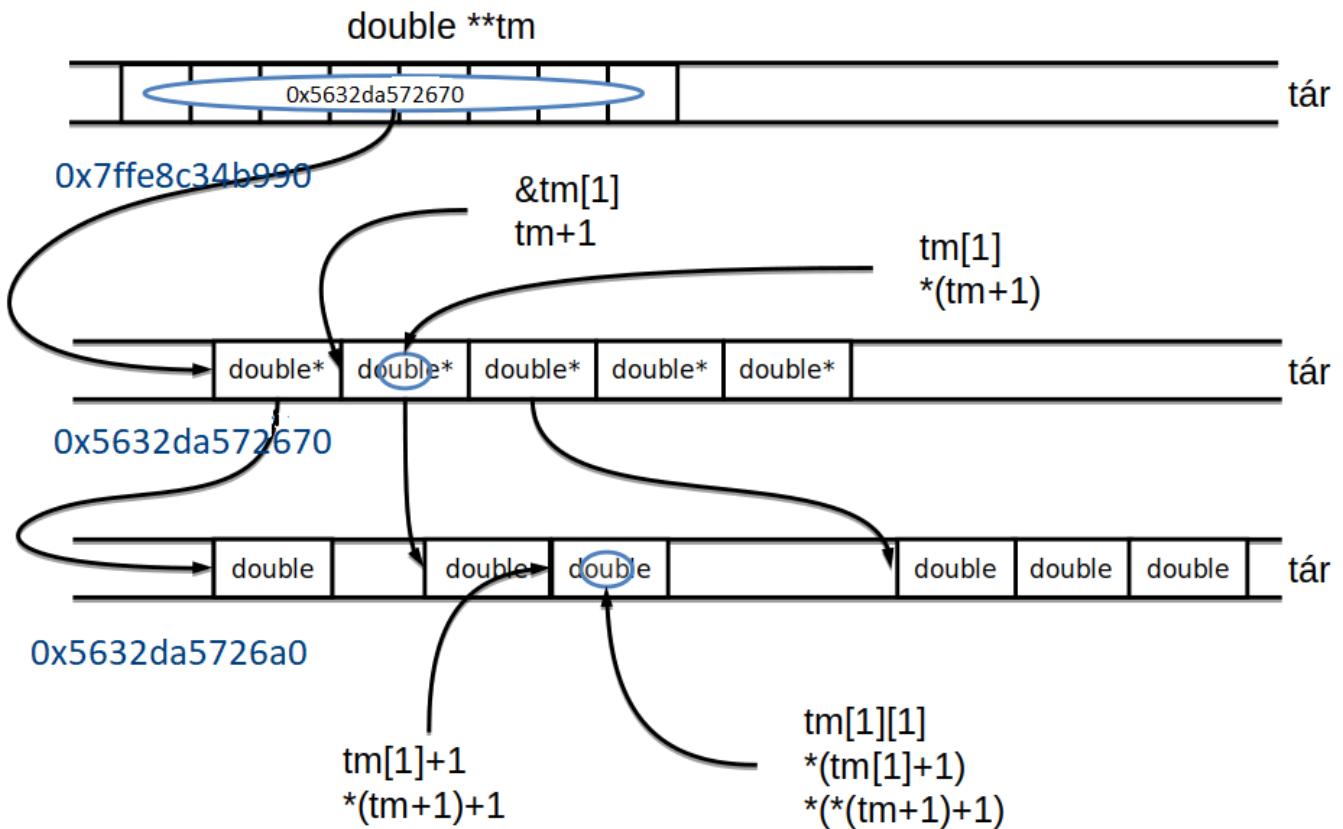
- ```
int (*v(int c))(int a, int b)
```

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

A feladat megoldásánál fontos tisztába lenni, hogy mi is az a háromszögmátrix. Nos, egy olyan mátrix, ahol a főátló felett (felső háromszögmátrix) vagy alatt (alsó háromszögmátrix) csupa nulla szerepel. A feladatban egy megadott 5×5 ös kétdimenziós tömböt szeretnénk megvalósítani. A memória foglalásokat a alsó háromszögmátrix értékei szerint elvégezzük. Ehhez egy ábra készült, ami bemutatja mit "ügyködünk" a memóriában. Az ábrát Bátfai Norbert készítette majd én szerkesztettem a saját értékeimmel. Ha jobban megnézzük akkor hasonlít a mátrix szerkezetéhez a memória foglalás. Ezt is akarta szemléltetni a feladat, hogy a memóriát nem csak egy szalagként tudjuk elképzelni.



4.1. ábra. Bátfa Norbert ábrája a `double**`háromszögmátrixról az én saját címeimmel.

A `double **`háromszögmátrix C változata:

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

int
main ()
{
    int nr = 5;
    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 (*tm[i])) == NULL)
{
    return -1;
}

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

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");
}

tm[3][0] = 42.0;
(*tm + 3)[1] = 43.0; // mi van, ha itt hiányzik a külső
()
*(tm[3] + 2) = 44.0;
*(tm[3] + 3) = 45.0;

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)
    free (tm[i]);

free (tm);

return 0;
}
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Az EXOR titkosítás egy egyszerű titkosítási eljárás. Lényege, hogy a titosítandó szöveg mellé rendelünk

egy titkosító kulcsot (szöveg), majd ezzel végezzük el a titkosítást. Az alábbi C kód az EXOR titkosító algoritmus.

```
#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);
    }
}
```

Működése:

Elsőként a kódban felveszünk 2 konstans változót. A MAX_KULCS 100 lesz a kulcsunk maximális mérete, illetve BUFFER_MERET 256 a maximálisan beolvasható stringek száma. Ezek után a main() függvénybe valósul meg a titkosító algoritmus. El van tárolva a kulcs illetve a bufferbe a szöveg. Ezután a kulcs méretét vizsgáljuk a strncpy függvényel. Ha túl nagy a kulcs mérete, akkor lecípi belőle a szükséges 100 karakternyi részét.

```
#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);
```

Továbbiakban egy while ciklus segítségével folyamatosan olvassuk be a bajtakat a titkosítandó szöveges állományból. A magban a lényegi dolgok történnek. A kiolvasott bajtot össze EXOR-ozza a kulcs adott bajtja segítségével. Az EXOR után a kulcsindexet növeljük. A titkosított bajtakat egy bufferbe olvassuk, majd kiirjuk egy output file-ba.

```
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!

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

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
```

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

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);

}
}

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

EXOR titkosítás Java nyelven megvalósítva. Itt specifikusan Java megvalósításokat használunk az implementációban de a kód sok része átültethető C-ből. Itt is a kulcs segítségével össze EXOR-ozzuk a szöveget. Majd léptetjük a kulcsindexet. A titkosított szöveget a `write` segítségével kiiírjük egy külön file-ba.

4.4. C EXOR törő

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

```
#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;  
}
```

Az exor törő minden esetben egy karakter sorozatból megadott kulcs alapján próbálja visszafejteni a szöveget. Bruteforce módszert használva minden lehetséges kombinációval kipróbálva töri fel. A tiszta szöveg előállítását is egy algoritmus végzi, ami egyes szavak alapján megpróbál értelmes szöveget visszafejteni.

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Ebben a feladatban Nagy Martin tutorált engem!

A matematikában a Mandelbrot-halmaz azon c komplex számokból áll (a „komplex számsík” azon pontjainak mértani helye, halmaza), melyekre az alábbi (komplex szám értékű) x_n rekurzív sorozat:

$$x_1 := c$$

$$x_{n+1} := (x_n)^2 + c$$

nem tart végtelenbe, azaz abszolút értékben (hosszára nézve) korlátos. Ez a komplex számokon egy nevezetes fraktálalakzatot formál. A továbbiakban ezt fogjuk leimplementálni C++ nyelven.

Először szükség van egy Makefile-ra, ami a .cpp file-ből előállítja a megfelelő kimenetet. Jelen esetben létrehozza az output-ot és a képet. Íme így néz ki a Makefile:

```
all: mandelbrot clean

mandelbrot.o: mandelbrot.cpp
    @g++ -c mandelbrot.cpp `libpng-config --cflags`

mandelbrot: mandelbrot.o
    @g++ -o mandelbrot mandelbrot.o `libpng-config --ldflags`

clean:
    @rm -rf *.o
    @./mandelbrot
    @rm -rf mandelbrot
```

Ahhoz, hogy előállíthassuk a képünket szükség van a png++/png.hpp header file-ra. Ha ez megvan, akkor a GeneratePNG(int tomb[N][M]) eljárás fogja legenerálni a kimenet.png állományt. Ezt ugy teszi meg, hogy az előállítanó kép mérete fix 500x500-as képpontú. Egy forciklus végigmegy az eljárás paraméterében átadott mátrixon, majd pixelről pixelre színez az értékek alapján. A paraméterként átadott mátrix a Mandelbrot-halmaz a komplex számsíkon vett pontjait tartalmazza.

```
void GeneratePNG( int tomb[N][M] )
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y] ←
                ], tomb[x][y]);
        }
    }
    image.write("kimenet.png");
}
```

Létrehozunk egy Komplex nevű struktúrát majd a re és im mezőkben el fogjuk tárolni a komplex számunk halmazán vett valós és képzetes egységeket.

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

A main() függvénybe van megírva a a Mandelbrot-halmaz algoritmusa.

```
/*
 * Program: Mandelbrot halmaz
 * Dátum: 2014. február. 26.
 * Tutor: Szabó Attila
 * Tutoriált: Tuza József
 */

#include <png++/png.hpp>

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

void GeneratePNG( int tomb[N][M] )
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
```

```
for (int y = 0; y < M; y++)
{
    image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], ←
        tomb[x][y]);
}
image.write("kimenet.png");
}

struct Komplex
{
    double re, im;
};

int main()
{
    int tomb[N][M];

    int i, j, k;

    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;

    struct Komplex C, Z, Zuj;

    int iteracio;

    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            C.re = MINX + j * dx;
            C.im = MAXY - i * dy;

            Z.re = 0;
            Z.im = 0;
            iteracio = 0;

            while (Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < ←
                255)
            {
                Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
                Zuj.im = 2 * Z.re * Z.im + C.im;
                Z.re = Zuj.re;
                Z.im = Zuj.im;
            }

            tomb[i][j] = 256 - iteracio;
        }
    }
}
```

```
    GeneratePNG(tomb);

    return 0;
}
```

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

Lényegében ugyanaz, mint a felső implementáció, viszont még ott mi írtunk egy külön struktúrát a Komplex számok kezelésére, itt a már meglévő `std::complex` osztályal fogunk dolgozni illetve az ő metódusaival. Cpp kód kicsit átírva `std::complex` osztályra.

```
/*
 * Program: Mandelbrot halmaz komplex osztályval
 * Dátum: 2014. március. 5.
 * A feladatot Szabó Attila és Tuza József által készített ←
 * alapfeladat alapján
 * Dalmadi Zoltán módosította
 */

#include <png++/png.hpp>
#include <complex>

const int N = 500;
const int M = 500;
const double MAXX = 0.7;
const double MINX = -2.0;
const double MAXY = 1.35;
const double MINY = -1.35;

void GeneratePNG(const int tomb[N][M])
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y] ←
                , tomb[x][y]);
        }
    }
    image.write("kimenet.png");
}

int main()
```

```
{  
    int tomb[N][M];  
  
    double dx = (MAXX - MINX) / N;  
    double dy = (MAXY - MINY) / M;  
  
    std::complex<double> C, Z, Zuj;  
  
    int iteracio;  
  
    for (int i = 0; i < M; i++)  
    {  
        for (int j = 0; j < N; j++)  
        {  
            C.real(MINX + j * dx);  
            C.imag(MAXY - i * dy);  
  
            Z = 0;  
            iteracio = 0;  
  
            while (abs(Z) < 2 && iteracio++ < 255)  
            {  
                Zuj = Z * Z + C;  
                Z = Zuj;  
            }  
  
            tomb[i][j] = 256 - iteracio;  
        }  
    }  
  
    GeneratePNG(tomb);  
  
    return 0;  
}
```

5.3. Biomorfok

```
// Verzio: 3.1.3.cpp  
// Forditas:  
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3  
// Futtatas:  
// ./3.1.3 biomorf.png 800 800 10 -2 2 -2 2 .285 0 10  
// Nyomtatas:  
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-  
// footer="BATF41 HAXOR STR34M" --right-footer="https://bhaxor. ↵
```

```
blog.hu/" --pro=color
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, 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
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9\_Iss5\_2305--2315\_Biomorphs\_via\_modified\_iterations.pdf
//

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

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

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

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

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

double dx = ( xmax - xmin ) / szelesseg;
double dy = ( ymax - ymin ) / magassag;

std::complex<double> cc ( reC, imC );

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

// 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 ) );
}

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 biomorf példákon végéggmenve úgy tapasztaljuk, hogy a logikája hasonló a Mandelbrot halmaz kódjához így egy kis átalakítással létrehozható a biomorf. Mi az a biomorf? Egy elő szervezetre (pl. baktériumra, egysejtűre) hasonlító forma vagy modell. Nem feltétlen jelent elő organizmust.

Tanulságok, tapasztalatok, magyarázat...

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

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!

Ebben a megoldásban a GUI létrehozására és kezelésére SFML grafikus library-t használunk. Ezt a csomagot telepítjük a sudo apt-get install libsfml-dev parancssal. Logika ugyanaz, mint a korábbi

Mandelbrot os példáknál. void generate_mandelbrot_set(sf::VertexArray vertexarray, int pixel_shift_x, int pixel_shift_y, int precision, float zoom) függvény fogja legenerálni a MBH_t egy Vertex array segítségével illetve minden az aktuális paraméterekként átadott értékek alapján. Pl. Ha zoom történik, akkor ujra meghívódik a függvény az éppen aktuális MBH részeként.

```
// Forrás: https://github.com/SullyChen/Mandelbrot-Set-Plotter

#include "SFML/Graphics.hpp"

//resolution of the window
const int width = 1280;
const int height = 720;

//used for complex numbers
struct complex_number
{
    long double real;
    long double imaginary;
};

//mandelbrot komplex alapján legenerál egy mandelbrot halmazt
void generate_mandelbrot_set(sf::VertexArray& vertexarray, int ←
    pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            //scale the pixel location to the complex plane for ←
            //calculations
            long double x = ((long double)j - pixel_shift_x) / ←
                zoom;
            long double y = ((long double)i - pixel_shift_y) / ←
                zoom;
            complex_number c;
            c.real = x;
            c.imaginary = y;
            complex_number z = c;
            int iterations = 0; //keep track of the number of ←
            //iterations
            for (int k = 0; k < precision; k++)
            {
                complex_number z2;
                z2.real = z.real * z.real - z.imaginary * z. ←
                    imaginary;
                z2.imaginary = 2 * z.real * z.imaginary;
                z2.real += c.real;
```

```
        z2.imaginary += c.imaginary;
        z = z2;
        iterations++;
        if (z.real * z.real + z.imaginary * z.imaginary <-
            > 4)
            break;
    }
    //color pixel based on the number of iterations
    if (iterations < precision / 4.0f)
    {
        vertexarray[i*width + j].position = sf::: ←
            Vector2f(j, i);
        sf::Color color(iterations * 255.0f / (←
            precision / 4.0f), 0, 0);
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision / 2.0f)
    {
        vertexarray[i*width + j].position = sf::: ←
            Vector2f(j, i);
        sf::Color color(0, iterations * 255.0f / (←
            precision / 2.0f), 0);
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision)
    {
        vertexarray[i*width + j].position = sf::: ←
            Vector2f(j, i);
        sf::Color color(0, 0, iterations * 255.0f / ←
            precision);
        vertexarray[i*width + j].color = color;
    }
}
}
```

A main() metódusba kezeljük le a GUI generálását. Először is sf:: ablaknak beállítunk címet, ablakméretet stb. Ezután a default értékekkel (zoom, precision, x_shift, y_shift) legeneráltatjuk a MBH-t a generate_mandelbrot_set(...) függvényel. Ameddig az ablak nyitott állapotba van addig két eseményt figyel. 1. Ha rákattintunk az X gombra, akkor zárja be az ablakot. 2. Ha bal gombbal belekattintunk a MBH-ba, akkor az átadott értékek segítségével újra legeneráltatja a MBH-t 2X nagyítással.

```
int main()
{
    sf::String title_string = "Mandelbrot Set Plotter"; //ablak ←
        címe
    sf::RenderWindow window(sf::VideoMode(width, height), ←
```

```
title_string); //ablak objektum(létrehozza az ablakot a ←
               megadott méretekkel és címmel)
window.setFramerateLimit(30); //frissített ablak/s vagy ←
                               ilyesmi
sf::VertexArray pointmap(sf::Points, width * height);

//értékek inicializálása
float zoom = 300.0f;
int precision = 100;
int x_shift = width / 2;
int y_shift = height / 2;

//legenerálja a mbh-t
generate_mandelbrot_set(pointmap, x_shift, y_shift, ←
                         precision, zoom);

/***
*
*
*
*
* */
while (window.isOpen())
{

    //ciklikusan figyeli az előforduló különböző event-eket ←
    //, ha egy olyan esemény következik be, hogy ←
    //rákattolunk az X gombra, akkor bezárja az ablakot
    sf::Event event;
    while (window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed)
            window.close();
    }

    //ha a bal egérgommbal kattintunk, akkor az egér ←
    //helyére nagyít az alábbi algoritmus segítségével. ←
    //Minden nagyítás után újra legenerálja a mbh-t.
    //zoom into area that is left clicked
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
    {
        sf::Vector2i position = sf::Mouse::getPosition( ←
                                                       window);
        x_shift -= position.x - x_shift;
        y_shift -= position.y - y_shift;
        zoom *= 2;
        precision += 200;
    }
}
```

```
#pragma omp parallel for
for (int i = 0; i < width*height; i++)
{
    pointmap[i].color = sf::Color::Black;
}
generate_mandelbrot_set(pointmap, x_shift, y_shift, ←
precision, zoom);
}
window.clear();
window.draw(pointmap);
window.display();
}

return 0;
}
```

5.6. Mandelbrot nagyító és utazó Java nyelven

Ebben a feladatban Lovász Botondot tutoráltam!

```
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.awt.event.*;

public class Mandelbrot extends JFrame implements ←
ActionListener
{

    private JPanel ctrlPanel;
    private JPanel btnPanel;
    private int numIter = 50;
    private double zoom = 130;
    private double zoomIncrease = 100;
    private int colorIter = 20;
    private BufferedImage I;
    private double zx, zy, cx, cy, temp;
    private int xMove, yMove = 0;
    private JButton[] ctrlBns = new JButton[9];
    private Color themeColor = new Color(150,180,200);

    public Mandelbrot() {
        super("Mandelbrot Set");
        setBounds(100, 100, 800, 600);
        setResizable(false);
```

```
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        plotPoints();

        Container contentPane = getContentPane();
        contentPane.setLayout(null);

        ctrlPanel = new JPanel();
        ctrlPanel.setBounds(600,0,200,600);
        ctrlPanel.setBackground(themeColor);
        ctrlPanel.setLayout(null);

        btnPanel = new JPanel();
        btnPanel.setBounds(0,200,200,200);
        btnPanel.setLayout(new GridLayout(3,3));
        btnPanel.setBackground(themeColor);

        ctrlBtns[1] = new JButton("up");
        ctrlBtns[7] = new JButton("down");
        ctrlBtns[3] = new JButton ("left");
        ctrlBtns[5] = new JButton("right");
        ctrlBtns[2] = new JButton("+");
        ctrlBtns[0] = new JButton("-");
        ctrlBtns[8] = new JButton(">");
        ctrlBtns[6] = new JButton("<");
        ctrlBtns[4] = new JButton();

        contentPane.add(ctrlPanel);
        contentPane.add(new imgPanel());
        ctrlPanel.add(btnPanel);

        for (int x = 0; x<ctrlBtns.length;x++) {
            btnPanel.add(ctrlBtns[x]);
            ctrlBtns[x].addActionListener(this);
        }

        validate();

    }

    public class imgPanel extends JPanel{
        public imgPanel(){
            setBounds(0,0,600,600);

        }
        @Override
```

```
public void paint (Graphics g){  
    super.paint(g);  
    g.drawImage(I, 0, 0, this);  
}  
}  
  
public void plotPoints(){  
    I = new BufferedImage(getWidth(), getHeight(),  $\leftarrow$   
        BufferedImage.TYPE_INT_RGB);  
    for (int y = 0; y < getHeight(); y++) {  
        for (int x = 0; x < getWidth(); x++) {  
            zx = zy = 0;  
            cx = (x - 320+xMove) / zoom;  
            cy = (y - 290+yMove) / zoom;  
            int iter = numIter;  
            while (zx * zx + zy * zy < 4 && iter > 0) {  
                temp = zx * zx - zy * zy + cx;  
                zy = 2 * zx * zy + cy;  
                zx = temp;  
                iter--;  
            }  
            I.setRGB(x, y, iter | (iter << colorIter));  
        }  
    }  
}  
  
public void actionPerformed(ActionEvent ae){  
    String event = ae.getActionCommand();  
  
    switch (event){  
    case "up":  
        yMove-=100;  
        break;  
    case "down":  
        yMove+=100;  
        break;  
    case "left":  
        xMove-=100;  
        break;  
    case "right":  
        xMove+=100;  
        break;  
    case "+":  
        zoom+=zoomIncrease;  
        zoomIncrease+=100;  
        break;  
    case "-":  
        zoom-=zoomIncrease;  
        zoomIncrease-=100;  
        break;  
    }
```

```
        case ">":
            colorIter++;
            break;
        case "<":
            colorIter--;
            break;
    }

    plotPoints();
    validate();
    repaint();
}
public static void main(String[] args)
{
    new Mandelbrot().setVisible(true);
}
}
```

A program az Eclipse gui library-t (swing, awt) használok, ami nagyban segíti az ablakozó rendszer létrejöttét. A Mandelbrot osztályt származtatjuk a JFrame osztályból így elérjük a JFram tulajdonságait. A konstrutorba létrehozzuk az ablakot a megfelelő méretekkel illetve felpakoluk a gombokat és a Mandelbrot halmazt megjelenítő ablakocskát. A `plotPoints()` függvény rajzolatja ki a Mandelbrot halmazt a mb algoritmus alapján. Továbbiakban minden egyes gombok által elérhető eseményekre feliratkozunk és megmondjuk hogy azon eseményre mi történjen. Ilyen eseményekre többnyire a MB halmaz értékeit változtatjuk specifikusan. Például ha nagyítunk vagy kicsinyítünk stb. minden eseménykor újrageneráltatjuk a MB-halmazt.

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.

```
public class PolarGenerator {  
    boolean nincsTarolt = true;  
    double tarolt;  
    public PolarGenerator() {  
  
        nincsTarolt = true;  
    }  
    public double kovetkezo() {  
        if(nincsTarolt) {  
            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);  
  
        tarolt = r*v2;  
        nincsTarolt = !nincsTarolt;  
  
        return r*v1;  
    }  
}
```

```
        } else {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }

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

A létrehozott `PolarGenerator` objektum példányának a `kovetkezo()` függvényét ha meghívjuk akkor 10 random tranzformált számok kapunk vissza. A lényegi matematikai eljárás nem fontos a számunkra hiszen az Objektum Orientált Paradigma szépen elrejti előlünk. Mi egyszerűen kapunk egy függvényt, amit tetszőlegesen felhasználhatunk a "boldog tudatlanságban". Ezt nevezzük egységbezárásnak.

6.2. LZW

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
typedef struct node{
    char c;
    struct node* left;
    struct node* right;
} Node;

Node* fa;
Node gyoker;

#define null NULL

Node* create_empty()
{
    Node* tmp = &gyoker;
    tmp->c= '/';
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
```

```
Node* create_node(char val)
{
    Node* tmp = (Node*)malloc(sizeof(Node));
    tmp->c=val;
    tmp->left = null;
    tmp->right = null;
    return tmp;
}

void insert_tree(char val)
{
    if(val=='0')
    {
        if(fa->left == null)
        {
            fa->left = create_node(val);
            fa = &gyoker;
            //printf("Inserted into left.");
        }
        else
        {
            fa = fa->left;
        }
    }
    else
    {
        if(fa->right == null)
        {
            fa->right = create_node(val);
            fa = &gyoker;
            //printf("Inserted into left.");
        }
        else
        {
            fa = fa->right;
        }
    }
}

void inorder(Node* elem, int depth)
{
    if(elem==null)
    {
        return;
    }
    inorder(elem->left, depth+1);
    if(depth)
    {
        char *spaces;
```

```
spaces =(char*) malloc(sizeof(char)*depth*2+1);
for(int i=0;i<depth;i+=2)
{
    spaces[i]='-';
    spaces[i+1]='-';
}
spaces[depth]='\0';

printf("%s%c\n",spaces,elem->c);
}
else
{
    printf("%c\n",elem->c);
}
inorder(elem->right,depth+1);
}

void preorder(Node* elem,int depth)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
    preorder(elem->left,depth+1);
    preorder(elem->right,depth+1);
}
void postorder(Node* elem,int depth)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left,depth+1);
```

```
postorder(elem->right, depth+1);
if(depth)
{
    char *spaces;
    spaces = (char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0; i<depth; i+=2)
    {
        spaces[i] = ' ';
        spaces[i+1] = ' ';
    }
    spaces[depth*2] = '\0';

    printf("%s%c\n", spaces, elem->c);
    free(spaces);
}
else
{
    printf("%c\n", elem->c);
}
}

void destroy_tree(Node* elem)
{
    if(elem==null)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if(elem->c == gyoker.c)
    {

    }
    else
    {
        free(elem);
    }
}

void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}

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

```
    srand(time(null));
    fa = create_empty();
    //gyoker = *fa;
    for(int i=0;i<10000;i++)
    {
        int x=rand()%2;
        if(x)
        {
            insert_tree('1');
        }
        else
        {
            insert_tree('0');
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1],"--preorder")==0)
        {
            preorder(&gyoker,0);
        }
        else if(strcmp(argv[1],"--inorder")==0)
        {
            inorder(&gyoker,0);
        }
        else if(strcmp(argv[1],"--postorder")==0)
        {
            postorder(&gyoker,0);
        }
        else
        {
            usage();
        }
    }
    else
    {
        usage();
    }
    destroy_tree(&gyoker);
    return 0;
}
```

A fenti programban az LZW algoritmussal kódolt binfa változata van megírva. Ez a feladat több részből áll, amit a továbbiakban részletekre bontva taglalunk.

```
typedef struct node
{
    char c;
    struct node* left;
    struct node* right;
```

```
    } Node;
```

```
    Node* fa;
    Node gyoker;
```

```
#define null NULL
```

```
Node* create_empty()
{
    Node* tmp = &gyoker;
    tmp->c= '/';
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
```

```
Node* create_node(char val)
{
    Node* tmp = (Node*)malloc(sizeof(Node));
    tmp->c=val;
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
```

Létrehozunk egy adatstruktúrát, aminek a neve és típusa Node lesz. Ebbe három property lesz. char c property-be tároljuk el az input karaktert. A left property egy saját node-ra mutató pointer. Ugyanez a left property-re. Ezután definiálunk egy Node* fa pointer objektumot és egy Node típusú objektumot.

A create_empty() függvény lényege, hogy létrehoz egy új Node* típusú pointer objektumot, aminek beállítjuk a bal, jobb gyermekét nullára majd a függvény visszatér ezzel a pointer objektummal.

A create_node(char val) függvény lényege, hogy paraméterként kapott char érték alapján először helyet foglal a memóriában, majd a val értékét eltárolja az legfoglalt memóriacímen. Beállítja a jobb és bal fiát nullára, majd visszatér egy Node* pointerrel.

```
void insert_tree(char val)
{
    if(val=='0')
    {
        if(fa->left == null)
        {
            fa->left = create_node(val);
            fa = &gyoker;
            //printf("Inserted into left.");
        }
        else
        {
            fa = fa->left;
        }
    }
}
```

```
        else
        {
            if (fa->right == null)
            {
                fa->right = create_node(val);
                fa = &gyoker;
                //printf("Inserted into right.");
            }
            else
            {
                fa = fa->right;
            }
        }
    }
```

A `insert_tree(char val)` eljárás a paraméterében megkapott érték alapján felépít egy ÚJ csomópontot a csomópont éppen aktuális jobb vagy bal fiával. Ha pl. a val értéke 1 akkor megnézi, hogy az fa ponter objektumban a aktuális csomópontnak van-e 1-es gyermekje. Ha null az érték a jobb gyermeknél akkor beállítja az 1-es értéket a jobb gyermekhez. Ez után a fát ráállítjuk a binfa gyökerére. Ha viszont már van 1-es gyermekje az aktuális node-nak akkor továbbhalad a jobb gyermekre és beállítja erre a fa mutatóját. Ugyanezen logika mentén játszódik le a 0-ás érték esetén.

```
void inorder(Node* elem, int depth)
{
    if (elem==null)
    {
        return;
    }
    inorder(elem->left, depth+1);
    if (depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth]='\0';

        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
    inorder(elem->right, depth+1);
}
```

```
void preorder(Node* elem, int depth)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
    preorder(elem->left, depth+1);
    preorder(elem->right, depth+1);
}
void postorder(Node* elem, int depth)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left, depth+1);
    postorder(elem->right, depth+1);
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n", spaces, elem->c);
        free(spaces);
    }
    else
    {
```

```
        printf("%c\n", elem->c);
    }
}
```

Ezekbe a metódusokba van megírta az három fabejárás(inorder, postorder, preorder). Rekurzív rendezések. Rendezéstől függ, hogy melyik algoritmus alapján járjuk be a fát. Pl. Inorder: Mindig a bal oldalt vizsgáljuk majd ha megvan a legutolsó bal elem akkor visszatér az ō szülejéhez majd pedig a jobb gyerekéhez. Ezt az agoritmust minden node-on lejátsza rekurzívan.

```
void destroy_tree(Node* elem)
{
    if (elem==NULL)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if (elem->c == gyoker.c)
    {

    }
    else
    {
        free(elem);
    }
}
```

A `destroy_tree(Node* elem)` függvény a felesleges memóriacímeket szabadítja fel. Ha elem null értékű akkor üres a visszatérési értéke. Ha viszont a paraméterként átadott csomópont nem null akkor önmagát meghívja a jobb illetve majd a bal fiára. Utánna a `free(elem)` metódussal felszabadítja a jobb vagy éppen a bal fia memóriacímét.

```
int main(int argc, char** argv)
{
    srand(time(NULL));
    fa = create_empty();
    //gyoker = *fa;
    for (int i=0;i<10000;i++)
    {
        int x=rand()%2;
        if (x)
        {
            insert_tree('1');
        }
        else
        {
            insert_tree('0');
        }
    }
    if (argc == 2)
    {
```

```
if(strcmp(argv[1], "--preorder")==0)
{
    preorder(&gyoker, 0);
}
else if(strcmp(argv[1], "--inorder")==0)
{
    inorder(&gyoker, 0);
}
else if(strcmp(argv[1], "--postorder")==0)
{
    postorder(&gyoker, 0);
}
else
{
    usage();
}
else
{
    usage();
}
destroy_tree(&gyoker);
return 0;
}
```

Az összes eddig szébtontogatott részek felhasználása a `main()` metóduson belül történik.

6.3. Fabejárás

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

Az előző feladatba részletezve van mindenkom fabejárás. Azt felhasználva illetve kiegészítve rakkom be ehhez a feladathoz.

```
void inorder(Node* elem, int depth)
{
    if(elem==NULL)
    {
        return;
    }
    inorder(elem->left, depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0; i<depth; i+=2)
        {
            spaces[i] = ' - ';
        }
    }
}
```

```
        spaces[i+1]='-';
    }
    spaces[depth]='\0';

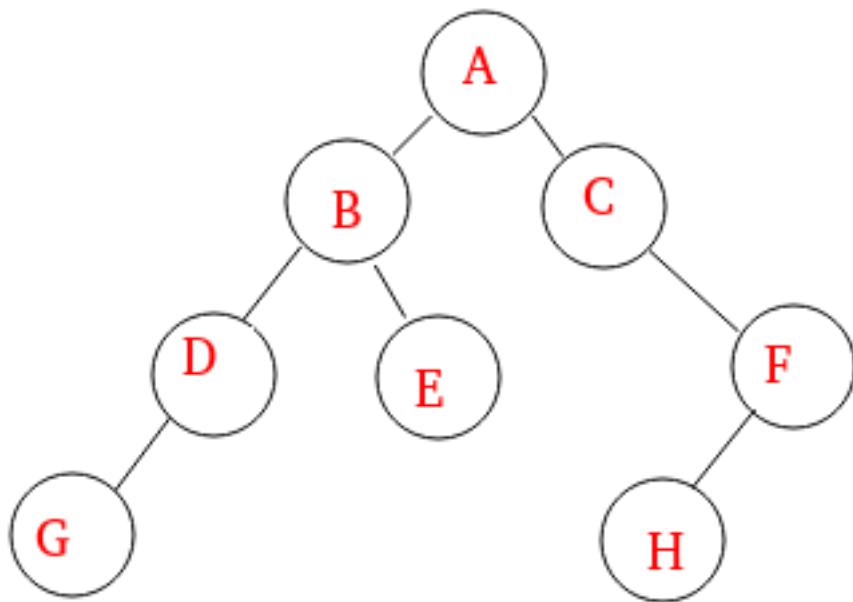
    printf("%s%c\n", spaces, elem->c);
}
else
{
    printf("%c\n", elem->c);
}
inorder(elem->right, depth+1);
}

void preorder(Node* elem, int depth)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
    preorder(elem->left, depth+1);
    preorder(elem->right, depth+1);
}
void postorder(Node* elem, int depth)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left, depth+1);
    postorder(elem->right, depth+1);
    if(depth)
    {
        char *spaces;
```

```
spaces =(char*) malloc(sizeof(char)*depth*2+1);
for(int i=0;i<depth;i+=2)
{
    spaces[i]='-';
    spaces[i+1]='-';
}
spaces[depth*2]='\0';

printf("%s%c\n", spaces, elem->c);
free(spaces);
}
else
{
    printf("%c\n", elem->c);
}
}
```



INORDER (LPR): G,D,B,E,A,C,H,F

PREORDER (PLR): A,B,D,G,E,C,F,H

POSTORDER (LRP): G,D,E,B,H,F,C,A

6.4. Tag a gyökér

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

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string.h>

#define null NULL

class Binfa
{
private:
    class Node
    {
    public:
        Node(char c=' ')
        {
            this->c=c;
            this->left = null;
            this->right = null;
        }
        char c;
        Node* left;
        Node* right;
    };
    Node* fa;
}

public:
    Binfa(): fa(&gyoker)
    {

    }

    void operator<<(char c)
    {
        if(c=='0')
        {
            if(fa->left == null)
            {
                fa->left = new Node('0');
                fa = &gyoker;
            }
            else
            {
                fa = fa->left;
            }
        }
    }
}
```

```
        }
    }
else
{
    if(fa->right == null)
    {
        fa->right = new Node('1');
        fa = &gyoker;
    }
else
{
    fa = fa->right;
}
}

void preorder(Node* elem,int depth=0)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
    preorder(elem->left,depth+1);
    preorder(elem->right,depth+1);
}

void inorder(Node* elem,int depth=0)
{
    if(elem==null)
    {
        return;
    }
    inorder(elem->left,depth+1);
```

```
if (depth)
{
    char *spaces;
    spaces = (char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0;i<depth;i+=2)
    {
        spaces[i]='_';
        spaces[i+1]='-';
    }
    spaces[depth*2]='\0';

    printf("%s%c\n",spaces,elem->c);
}
else
{
    printf("%c\n",elem->c);
}
inorder(elem->right,depth+1);
}

void postorder(Node* elem,int depth=0)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left,depth+1);
    postorder(elem->right,depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='_';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
}

void destroy_tree(Node* elem)
{
    if(elem==null)
```

```
{  
    return;  
}  
destroy_tree(elem->left);  
destroy_tree(elem->right);  
if(elem->c=='/') delete elem;  
}  
  
Node gyoker;  
};  
  
void usage()  
{  
    printf("Használat: ./binfa KAPCSOLÓ\n");  
    printf("Az KAPCSOLÓ lehet:\n");  
    printf("--preorder\tA bináris fa preorder bejárása\n");  
    printf("--inorder\tA bináris fa inorder bejárása\n");  
    printf("--postorder\tA bináris fa postorder bejárása\n");  
}  
  
int main(int argc, char** argv)  
{  
    srand(time(0));  
    Binafa bfa;  
    for(int i=0;i<100;i++)  
    {  
        int x=rand()%2;  
        if(x)  
        {  
            bfa<<'1';  
        }  
        else  
        {  
            bfa<<'0';  
        }  
    }  
    if(argc == 2)  
    {  
        if(strcmp(argv[1], "--preorder")==0)  
        {  
            bfa.preorder(&bfa.gyoker);  
        }  
        else if(strcmp(argv[1], "--inorder")==0)  
        {  
            bfa.inorder(&bfa.gyoker);  
        }  
        else if(strcmp(argv[1], "--postorder")==0)  
        {  
            bfa.postorder(&bfa.gyoker);  
        }  
    }  
}
```

```
        }
    else
    {
        usage();
    }
}
else
{
    usage();
}
bfa.destroy_tree(&bfa.gyoker);
return 0;
}
```

Ebben a változatban tagként van definiálva a csomópont gyökér. Mivel tag így, hogy elérjük szükségünk van a referenciajára. mindenhol, ahol szükség van a gyökér tagra ott alkalmazni kell a referencia operátort.

6.5. Mutató a gyökér

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

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string.h>

#define null NULL

class Binfa
{
private:
    class Node
    {
public:
    Node(char c='/')
    {
        this->c=c;
        this->left = null;
        this->right = null;
    }
    char c;
    Node* left;
    Node* right;
};

Node* fa;

public:
```

```
Binfa()
{
    gyoker=fa=new Node();
}

void operator<<(char c)
{
    if(c=='0')
    {
        if(fa->left == null)
        {
            fa->left = new Node('0');
            fa = gyoker;
        }
        else
        {
            fa = fa->left;
        }
    }
    else
    {
        if(fa->right == null)
        {
            fa->right = new Node('1');
            fa = gyoker;
        }
        else
        {
            fa = fa->right;
        }
    }
}

void preorder(Node* elem,int depth=0)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';
        cout<<spaces;
    }
    if(elem->left)
        cout<<'('<<elem->left->val<<') ';
    if(elem->right)
        cout<<'['<<elem->right->val<<'] ';
}
```

```
        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
    preorder(elem->left, depth+1);
    preorder(elem->right, depth+1);
}

void inorder(Node* elem, int depth=0)
{
    if(elem==null)
    {
        return;
    }
    inorder(elem->left, depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
    inorder(elem->right, depth+1);
}

void postorder(Node* elem, int depth=0)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left, depth+1);
    postorder(elem->right, depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
```

```
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
}

void destroy_tree(Node* elem)
{
    if(elem==null)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if(elem->c=='/')
        delete elem;
}

Node* gyoker;

};

void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n" );
}

int main(int argc, char** argv)
{
    srand(time(0));
    Binfa bfa;
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            bfa<<'1';
        }
        else
        {
            bfa<<'0';
        }
    }
}
```

```
        }
    else
    {
        bfa<<'0';
    }
}
if(argc == 2)
{
    if(strcmp(argv[1], "--preorder")==0)
    {
        bfa.preorder(bfa.gyoker);
    }
    else if(strcmp(argv[1], "--inorder")==0)
    {
        bfa.inorder(bfa.gyoker);
    }
    else if(strcmp(argv[1], "--postorder")==0)
    {
        bfa.postorder(bfa.gyoker);
    }
    else
    {
        usage();
    }
}
else
{
    usage();
}
bfa.destroy_tree(bfa.gyoker);
return 0;
}
```

Lényegében a Node gyoker tagok változtattuk át Node* gyoker-re. A konstruktort átalakítjuk úgy hogy nem referenciát adunk át, hanem a gyökérnek egy új helyet osztunk fel a memóriában a new operátor segítségével. Továbbá a gyökér minden előfordulásánál ki kell venni a referencia szerinti hivatkozásokat.

6.6. Mozgató szemantika

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

Megoldás videó:

```
// z3a9.cpp
//
// Copyright (C) 2011, 2012, Bátfai Norbert, nbatfai@inf.unideb.hu, ←
// 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/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public ←
// License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) ←
// későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos ←
// lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY ←
// CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve ←
// .
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU ←
// General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.

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

#include "Csomopont.hpp"

class LZWBInFa
{
public:
    LZWBInFa()
    {
        gyoker = new Csomopont('/');
    }
}
```

```
        fa = gyoker;
    }

LZWBinFa(const LZWBinFa &forras):LZWBinFa()
{
    std::cout << "Masolo konstruktor" << std::endl;

    if (gyoker != nullptr)
    {
        szabadit(gyoker);
        std::cout << "Masolo konstruktor" << std::endl;
        gyoker = copy(forras.gyoker, forras.fa);
    }
}

LZWBinFa(LZWBinFa &&forras)
{
    std::cout << "Move ctor" << std::endl;

    gyoker = nullptr;

    *this = std::move(forras);
}

LZWBinFa& operator=(const LZWBinFa &forras)
{
    if (this == &forras)
    {
        return *this;
    }

    if (forras.gyoker == nullptr)
    {
        return *this;
    }

    szabadit(gyoker);

    gyoker = copy(forras.gyoker, forras.fa);

    return *this;
}

LZWBinFa& operator=(LZWBinFa &&forras)
{
    std::cout << "Move assignment" << std::endl;
    std::swap(gyoker, forras.gyoker);
    return *this;
}
```

```
~LZWBinFa()
{
    szabadit (gyoker);
}

void operator<<(const 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()
{
    melyseg = 0;

    kiir(gyoker, std::cout);
}

void kiir (std::ostream &os)
{
    melyseg = 0;

    kiir(gyoker, os);
}
```

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

friend std::ostream &operator<<(std::ostream &os, LZWBInFa &bf)
{
    bf.kiir(os);

    return os;
}

private:
    Csomopont *fa;

    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;

    void kiir(Csomopont *elem, std::ostream & os)
    {
        if (elem != nullptr)
        {
            // InOrder
            ++melyseg;
            kiir(elem->egyesGyermek(), os);

            for (int i = 0; i < melyseg; ++i)
            {
                os << "---";
            }

            os << elem->getBetu() << "(" << melyseg - 1 << ")" << endl;

            kiir(elem->>nullasGyermek(), os);

            --melyseg;
        }
    }

    Csomopont *copy (const Csomopont *forras, const Csomopont * ←
                     regifa )
    {
        Csomopont* masolt = nullptr;

        if (forras != nullptr)
        {
            masolt = new Csomopont(forras->getBetu());

            masolt->ujEgyesGyermek(copy(forras->egyesGyermek(), ←
```

```
        regifa));  
  
    masolt->ujNullasGyermek(copy(forras->nullasGyermek(), ←  
        regifa));  
  
    if (regifa == forras)  
    {  
        fa = masolt;  
    }  
}  
  
return masolt;  
}  
  
void szabadit (Csomopont *elem)  
{  
    if (elem != nullptr)  
    {  
        // Free the children, then ourselves.  
        // PostOrder  
        szabadit(elem->egyesGyermek());  
        szabadit(elem->>nullasGyermek());  
  
        delete elem;  
    }  
}  
  
protected:  
    Csomopont *gyoker;  
  
    int maxMelyseg;  
    double atlag, szoras;  
  
    void rmelyseg(Csomopont *elem);  
    void ratlag(Csomopont *elem);  
    void rszoras(Csomopont *elem);  
};  
  
int LZWBInFa::getMelyseg()  
{  
    melyseg = maxMelyseg = 0;  
    rmelyseg(gyoker);  
  
    return maxMelyseg - 1;  
}  
  
double LZWBInFa::getAtlag()  
{  
    melyseg = atlagosszeg = atlagdb = 0;
```

```
ratlag(gyoker);
atlag = ((double) atlagosszeg) / atlagdb;

return atlag;
}

double LZWBinFa::getSzoras()
{
    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 != nullptr)
    {
        ++melyseg;

        if (melyseg > maxMelyseg)
        {
            maxMelyseg = melyseg;
        }

        rmelyseg(elem->egyesGyermek());
        rmelyseg(elem->>nullasGyermek());

        --melyseg;
    }
}

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

```
++melyseg;

ratlag(elem->egyesGyermek());
ratlag(elem->>nullasGyermek());
--melyseg;

if ((elem->egyesGyermek() == nullptr) && (elem->leftrightarrow
nullasGyermek() == nullptr))
{
    ++atlagdb;
    atlagosszeg += melyseg;
}
}

void LZWBInFa::rszoras (Csomopont *elem)
{
    if (elem != nullptr)
    {
        ++melyseg;

        rszoras(elem->egyesGyermek());
        rszoras(elem->>nullasGyermek());

        --melyseg;
        if ((elem->egyesGyermek() == nullptr) && (elem->leftrightarrow
nullasGyermek() == nullptr))
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag)) ↔
;
        }
    }
}

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

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;

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;
    }
}

LZWBinFa binFa_copy = binFa;

kiFile << binFa;

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

kiFile<<"#####másolt ←
#####\n";
kiFile << binFa_copy;

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

LZWBinFa binFa_move = std::move(binFa_copy);
kiFile<<"#####mozgatott ←
#####\n";
kiFile << binFa_move;
kiFile << "depth = " << binFa_move.getMelyseg () << std::endl;
kiFile << "mean = " << binFa_move.getAtlag () << std::endl;
kiFile << "var = " << binFa_move.getSzoras () << std::endl;

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

return 0;
}
```

Ebben a megoldásban a Csomópont osztály ki lett írva külön header/cpp fileokba!

Ha alkalmazunk másolást a copy constructort használjuk. A C++ szerint, ha másolunk akkor szükségünk

lehet mozgatásra is. Ezt definiálja a move constructor. Illetve ha ezeket használjuk, akkor szükség van destructorra is. Ez a hármas az úgynevezett mozgató szemantika. Ezek a LZWBInfa osztályban így néznek ki:

```
LZWBInFa::LZWBInFa(const LZWBInFa &forras) : LZWBInFa()
{
    std::cout << "Masolo konstruktor" << std::endl;

    if (gyoker != nullptr)
    {
        szabadit(gyoker);
        std::cout << "Masolo konstruktor" << std::endl;
        gyoker = copy(forras.gyoker, forras.fa);
    }
}

LZWBInFa::LZWBInFa(LZWBInFa &&forras)
{
    std::cout << "Move ctor" << std::endl;

    gyoker = nullptr;

    *this = std::move(forras);
}

LZWBInFa& LZWBInFa::operator=(const LZWBInFa &forras)
{
    if (this == &forras)
    {
        return *this;
    }

    if (forras.gyoker == nullptr)
    {
        return *this;
    }

    szabadit(gyoker);

    gyoker = copy(forras.gyoker, forras.fa);

    return *this;
}

LZWBInFa& LZWBInFa::operator=(LZWBInFa &&forras)
{
    std::cout << "Move assignment" << std::endl;
    std::swap(gyoker, forras.gyoker);
    return *this;
}
```

```
~LZWBinFa ()  
{  
    szabadit (gyoker);  
}
```

Felhasználásuk pedig a main metódus végén történik.

```
LZWBinFa binFa_copy = binFa;  
  
kiFile << binFa;  
  
kiFile << "depth = " << binFa.getMelyseg () << std::endl;  
kiFile << "mean = " << binFa.getAtlag () << std::endl;  
kiFile << "var = " << binFa.getSzoras () << std::endl;  
  
kiFile<<"#####másolt" <<  
        #####\n";  
kiFile << binFa_copy;  
  
kiFile << "depth = " << binFa_copy.getMelyseg () << std::endl;  
kiFile << "mean = " << binFa_copy.getAtlag () << std::endl;  
kiFile << "var = " << binFa_copy.getSzoras () << std::endl;  
  
LZWBinFa binFa_move = std::move(binFa_copy);  
kiFile<<"#####mozgatott" <<  
        #####\n";  
kiFile << binFa_move;  
kiFile << "depth = " << binFa_move.getMelyseg () << std::endl;  
kiFile << "mean = " << binFa_move.getAtlag () << std::endl;  
kiFile << "var = " << binFa_move.getSzoras () << std::endl;  
  
kiFile.close ();  
beFile.close ();  
  
return 0;
```

Fontos megemlíteni, hogy a LZWBinfa mozgatásánál, hogy az std::move(binFa_copy) csak egy jobbérték referenciát csinál a binFa_copy-ból. A jobbérték refenrencia után hívódik meg a move_ctor. Hogy mi mikor fut le a kódban elhelyezett nyomkövető üzenetek segítenek. Ebben az esetben a következő:

```
dekanyrobert@dekanyrobert:~/Dokumentumok/progl/binfa/vedes$ ./lzw <<  
befile.txt -o kifile  
Masolo konstruktor
```

Masolo konstruktor
Move ctor
Move assignment

A mozgató konstruktorban a referenciaiként megkapott LZWBinfát egy még nem létező objektumba másoljuk majd az eredeti fát kinullázzuk (nullptr-re állítjuk az értékét). Ezek után a régi fa lesz az mozgatott új fánk.

```
LZWBInFa(LZWBInFa &&forras)
{
    std::cout << "Move ctor" << std::endl;

    gyoker = nullptr;

    *this = std::move(forras);
}
```

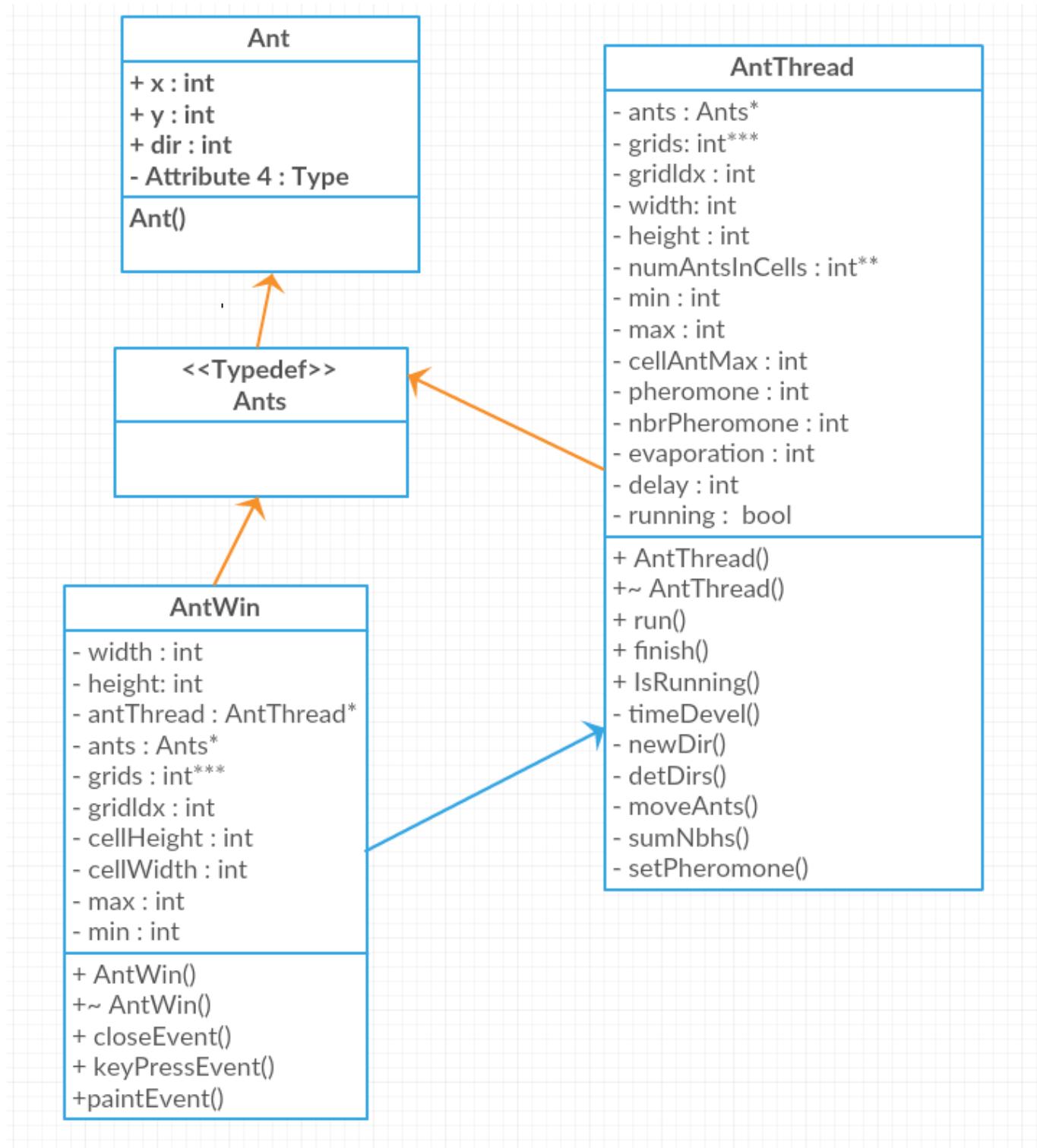
7. fejezet

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>



7.1. ábra. Hangyasimuláció UML

Az UML ábrában minden blokk egy osztályt jelent. Blokkon belül 3 tagrészt különítünk el. Fentről lefelé haladva: osztálynév, tulajdonságok, viselkedés. A + vagy minusz jelek a láthatóságot jelentik. + ha más osztályok láthatják illetve - ha nem (private).

Ant

Az Ant osztály fogja létrehozni a hangya objektumokat. Vannak különböző mezői, ami a hangyára jellemző illetve egy `Ant()` függvénye.

Ant Thread

Ez az osztály írja le a hangya egyed tulajdonságait. A tulajdonságok meghatározzák pl. az egyes egyedek elhelyezkedését, viselkedését, mozgását stb. Valamit ezekhez a tulajdonságokhoz párosulnak függvények. Ilyen pl. a `run()`, `newDir()`, `detDirs()` függvények, amik a hangya mozgását írják le.

Ant Win

Ebben az osztályban történik meg a rácsvonalak valamint a hangyák és feromon útvonalak kirajzoltatása. Tartalmaz továbbá egy `AntThread` pointert is, ami az egyes hangya egyedet tulajdonságait és viselkedéseit írják le.

7.2. Java életjáték

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

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.Shape;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.ImageObserver;
import java.text.AttributedCharacterIterator;
import java.util.ArrayList;
import java.awt.Event;
public class game_of_life extends JFrame {
    RenderArea ra;
    private int i;

    public game_of_life() {
        super("Game of Life");
        this.setSize(1005, 1030);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
        this.setResizable(false);
        ra = new RenderArea();
        ra.setFocusable(true);
        ra.grabFocus();
```

```
        add(ra);
        ra.edit_mode = true;
        ra.running = true;
    }
    public void update() {
        ArrayList<ArrayList<Boolean>> entities = new ArrayList<ArrayList<Boolean>>();
        int size1 = ra.entities.size();
        int size2 = ra.entities.get(0).size();
        for(int i=0;i<size1;i++)
        {
            entities.add( new ArrayList<Boolean>());
            for(int j=0;j<size2;j++)
            {
                int alive = 0;

                if(ra.entities.get((size1+i-1)%size1).get((size2+j-1)%size2)) alive++;
                if(ra.entities.get((size1+i-1)%size1).get((size2+j)%size2)) alive++;
                if(ra.entities.get((size1+i-1)%size1).get((size2+j+1)%size2)) alive++;
                if(ra.entities.get((size1+i)%size1).get((size2+j-1)%size2)) alive++;
                if(ra.entities.get((size1+i)%size1).get((size2+j+1)%size2)) alive++;
                if(ra.entities.get((size1+i+1)%size1).get((size2+j-1)%size2)) alive++;
                if(ra.entities.get((size1+i+1)%size1).get((size2+j)%size2)) alive++;
                if(ra.entities.get((size1+i+1)%size1).get((size2+j+1)%size2)) alive++;

                /*for(int k=-1;k<2;k++)
                {
                    for(int l = -1; l < 2 ;l++)
                    {
                        if(!(k==0 && l == 0))
                        {
                            if(ra.entities.get((size1+i+k)%size1).get((size2+j+l)%size2)) alive++;
                        }
                    }
                }*/
                if(ra.entities.get(i).get(j))
                {
                    if(alive < 2 || alive > 3)
                    {
                        //ra.entities.get(i).set(j,false);
                        entities.get(i).add(false);
                    }
                }
            }
        }
    }
}
```

```
        }
        else
        {
            entities.get(i).add(true);
        }
    }
    else
    {
        if(alive == 3)
        {
            //ra.entities.get(i).set(j,true);
            entities.get(i).add(true);
        }
        else
        {
            entities.get(i).add(false);
        }
    }
}
ra.entities = entities;
}

class RenderArea extends JPanel implements KeyListener {
    public ArrayList<ArrayList<Boolean>> entities;
    public int diff;
    public boolean edit_mode;
    public boolean running;
    public RenderArea() {
        super();
        setSize(1000, 1000);
        setVisible(true);
        setBackground(Color.WHITE);
        setForeground(Color.BLACK);
        setLocation(0, 0);
        diff = 20;

        this.addMouseListener((MouseListener) new MouseListener <-
        () {

            @Override
            public void mouseReleased(MouseEvent arg0) {

            }

            @Override
            public void mousePressed(MouseEvent arg0) {
                clicked(arg0);
            }

            @Override
```

```
public void mouseExited(MouseEvent arg0) {  
}  
  
}  
  
@Override  
public void mouseEntered(MouseEvent arg0) {  
}  
  
}  
  
@Override  
public void mouseClicked(MouseEvent arg0) {  
}  
});  
this.addKeyListener(this);  
entities = new ArrayList<ArrayList<Boolean>>();  
for(int i=0;i<1000/diff;i++)  
{  
    entities.add(new ArrayList<Boolean>());  
    for(int j=0;j<1000/diff;j++)  
    {  
        entities.get(i).add(false);  
    }  
}  
  
}  
void clicked(MouseEvent arg0)  
{  
    System.out.println("Button "+(arg0.getButton()== 1 ? " ←  
        Left" : "Right"));  
    System.out.println("X:"+arg0.getX()/diff);  
    System.out.println("Y:"+arg0.getY()/diff);  
    if(edit_mode)  
    {  
        entities.get(arg0.getX()/diff).set(arg0.getY()/diff ←  
            ,!entities.get(arg0.getX()/diff).get((arg0.getY ←  
                ()/diff)));  
        this.update(this.getGraphics());  
    }  
}  
}  
@Override  
public void keyTyped(KeyEvent e) {  
    //System.out.println(e.getKeyChar());  
}  
  
}@Override  
public void keyReleased(KeyEvent e) {  
    System.out.println("Key pressed:"+e.getKeyChar());  
    if(e.getKeyChar()=='e')
```

```
{  
    edit_mode = !edit_mode;  
}  
else if(e.getKeyChar()=='q')  
{  
    this.running = false;  
}  
else if(e.getKeyChar()=='c')  
{  
    if(edit_mode)  
{  
        for(int i=0;i<this.entities.size();i++)  
        {  
            for(int j=0;j<this.entities.get(1).size();j++)  
            {  
                this.entities.get(i).set(j,false);  
            }  
        }  
        this.update(this.getGraphics());  
    }  
}  
}  
  
}@Override  
public void keyPressed(KeyEvent e) {  
    //System.out.println(e.getKeyChar());  
}  
@Override  
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    g.clearRect(0, 0, 1000, 1000);  
    for(int i=0;i<1000;i+=diff)  
    {  
        g.drawLine(i, 0, i, 1000);  
    }  
    for(int j=0;j<1000;j+=diff)  
    {  
        g.drawLine(0, j, 1000, j);  
    }  
    for(int i=0;i<1000;i+=diff)  
    {  
        for(int j=0;j<1000;j+=diff)  
        {  
            if(entities.get(i/diff).get(j/diff))  
            {  
                g.setColor(Color.BLACK);  
            }  
        }  
    }  
}
```

```
        else
        {
            g.setColor(Color.WHITE);
        }

        g.fillRect(i+2, j+2, diff-3, diff-3);
    }
}

private static final long serialVersionUID = 1L;

}

private static final long serialVersionUID = 1L;
public static void main(String args[])
{
    game_of_life gol = new game_of_life();
    while(gol.ra.running)
    {
        if(!gol.ra.edit_mode)gol.update();
        try{Thread.sleep(200);}
        catch(Exception ex)
        {
        }
        gol.ra.update(gol.ra.getGraphics());
    }
    gol.dispose();
}
}
```

Lényegében majdnem ugyanaz van megírva, mint a C++ verzióban. Annyi, hogy a Java-s verzióban a beépített gui library-keket (swing, awt) használom. A program lényege az ez alatt lévő C++ életjátékba van kifejtve. Továbbiakban hozzá szeretnék írni

7.3. Qt C++ életjáték

Most Qt C++-ban!

Az életjátékot John Conway Cambridge Egyetem matematikusa találta ki. Ez egy nullszemélyes játék. Lényege, hogy a játékos megad kezdő alakzatot vagy alakzatokat és ha elindítjuk egy számítás eredményeként bizonyos feltételek mellett új alakzatot kapunk. Sejtautomaták közé tartozik ez a fajta játék. Szabályok:

1. Túléli a sejt(kocka), ha a közvetlen közelébe 2 vagy 3 szomszédja van.
2. A sejt elpusztul, ha 2-nél kevesebb vagy 3-nál több szomszédja van. Az előbbit túlnépesedésnek a utobbit elszigetelődésnek nevezük.
3. Új sejt születik minden olyan cellában, amelynek környezetében párom sejt található.

Jellegzetes alakzat a Bill Gosper féle "siklóágyú", amely időközönként siklókat lő ki.

Tanulságok, tapasztalatok, magyarázat...

```
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <vector>
#include <iostream>
using namespace sf;
using std::vector;
using std::cout;
using std::endl;
class Grid
{
public:
    Grid(unsigned int x = 1000, unsigned int y = 1000, unsigned int diff = ↵
        50) : w(x), h(y), diff(diff)
    {
    }

    void draw(RenderWindow & window)
    {
        for(int i=0; i<w; i+=diff)
        {
            Vertex line[] =
            {
                sf::Vertex(sf::Vector2f(i, 0)),
                sf::Vertex(sf::Vector2f(i, h))
            };
            line[0].color = Color(0,0,0);
            line[1].color = Color(0,0,0);
            window.draw(line, 2, sf::Lines);
        }
        for(int i=0; i<h; i+=diff)
        {
            Vertex line[] =
            {
                sf::Vertex(sf::Vector2f(0,i)),
                sf::Vertex(sf::Vector2f(w,i))
            };
            line[0].color = Color(0,0,0);
            line[1].color = Color(0,0,0);
            window.draw(line, 2, sf::Lines);
        }
    }
    unsigned int w;
    unsigned int h;
    unsigned int diff;
};

class Square
{
public:
    Square()
```

```
{  
}  
}  
Square(int x_pos, int y_pos, float w, bool alive = false)  
{  
    square = new RectangleShape(Vector2f(w,w));  
    square->setPosition(Vector2f(x_pos,y_pos));  
    aliveState = alive;  
}  
/*Square (const Square& other )  
{  
    if(this != &other)  
    {  
        delete this->square;  
        this->square = other.square;  
    }  
}  
Square& operator=(const Square& other)  
{  
    if(this != &other)  
    {  
        delete this->square;  
        this->square = other.square;  
    }  
    return *this;  
}*/  
~Square()  
{  
    delete square;  
}  
void update()  
{  
    if(aliveState)  
    {  
        square->setFillColor(Color::Black);  
    }  
    else  
    {  
        square->setFillColor(Color::White);  
    }  
}  
void setFill(Color c = Color::White)  
{  
    square->setFillColor(c);  
}  
void draw(RenderWindow &window)  
{  
    window.draw(*square);  
}  
RectangleShape* square;
```

```
    bool aliveState;
private:

};

vector<vector<Square*>> update(vector<vector<Square*>> v)
{
    vector<vector<Square*>> tmp ; //= v;
    for(int i=0;i<v.size();i++)
    {
        tmp.push_back(vector<Square*>());
        for(int j=0;j<v[0].size();j++)
        {
            tmp[i].push_back(new Square(v[i][j]->square->getPosition().x,v[  
    i][j]->square->getPosition().y,v[i][j]->square->getSize().x,  
    v[i][j]->aliveState));
        }
    }
    for(int i=0;i<v.size();i++)
    {
        for(int j=0;j<v[0].size();j++)
        {
            int live_neighbours = 0;
            live_neighbours += v[(i-1)%v.size()][(j-1)%v[0].size()]->  
                aliveState;
            live_neighbours += v[(i-1)%v.size()][(j)%v[0].size()]->  
                aliveState;
            live_neighbours += v[(i-1)%v.size()][(j+1)%v[0].size()]->  
                aliveState;
            live_neighbours += v[(i)%v.size()][(j-1)%v[0].size()]->  
                aliveState;
            live_neighbours += v[(i)%v.size()][(j+1)%v[0].size()]->  
                aliveState;
            live_neighbours += v[(i+1)%v.size()][(j-1)%v[0].size()]->  
                aliveState;
            live_neighbours += v[(i+1)%v.size()][(j)%v[0].size()]->  
                aliveState;
            live_neighbours += v[(i+1)%v.size()][(j+1)%v[0].size()]->  
                aliveState;
            //cout << " X:" << i << " y:" << j << " Live neighbours:" <<  
                live_neighbours << endl;
            if(v[i][j]->aliveState)
            {
                if(live_neighbours < 2)
                {
                    tmp[i][j]->aliveState = false;
                }
                else if(live_neighbours > 3)
                {
                    tmp[i][j]->aliveState = false;
                }
            }
        }
    }
}
```

```
        }
    else
    {
        if(live_neighbours == 3)
        {
            tmp[i][j]->aliveState = true;
        }
    }
}
return tmp;
}
void killall(vector<vector<Square*>> &v)
{
    for(int i=0;i<v.size();i++)
    {
        for(int j=0;j<v[0].size();j++)
        {
            v[i][j]->aliveState=false;
        }
    }
}
int main()
{
    RenderWindow window(VideoMode(1000,1000), "Game of Life");
    window.setFramerateLimit(10);
    window.setActive();

    Vector2u size = window.getSize();
    Grid g(size.x,size.y,1000/40);

    int h = g.h/g.diff+1;
    int w = g.w/g.diff+1;
    //Square squares[h][w];
    std::vector<std::vector<Square*>> squares;
    bool edit_mode = true;
    for(int i=0;i<h;i++)
    {
        squares.push_back(vector<Square*>());
        for(int j=0;j<w;j++)
        {
            squares[i].push_back(new Square(i*g.diff+1,j*g.diff+2,g.diff-3) ↔
                );
        }
    }
    //squares[4][5]->aliveState=true;
    while (window.isOpen())
    {
        window.clear(sf::Color::White);
```

```
// check all the window's events that were triggered since the last ←
    iteration of the loop
sf::Event event;
while (window.pollEvent(event))
{
    // "close requested" event: we close the window
    if (event.type == sf::Event::Closed)
    {
        window.close();
    }
    else if (event.type == Event::MouseButtonPressed)
    {
        if (edit_mode && event.mouseButton.button == Mouse::Button::Left)
        {
            /*cout<<event.mousePosition.x<<" "<<event.mousePosition.y<< ←
                endl;
            cout<<event.mousePosition.x/g.diff<< " "<< event. ←
                mouseButton.y/g.diff<<endl; */
            squares[event.mousePosition.x/g.diff][event.mousePosition.y ←
                /g.diff]->aliveState= !squares[event.mousePosition.x/g ←
                .diff][event.mousePosition.y/g.diff]->aliveState;
            cout<< "Changed state on entity at X:"<< event. ←
                mouseButton.x/g.diff << " Y:"<<event.mousePosition.y/g ←
                .diff << " to "<< (squares[event.mousePosition.x/g. ←
                diff][event.mousePosition.y/g.diff]->aliveState? " ←
                Alive" : "Dead")<<endl;
        }
    }
    else if (event.type == Event::KeyPressed)
    {
        if (event.key.code == Keyboard::Q)
        {
            cout<<"Close request received. Application will exit." ←
                <<endl;
            window.close();
        }
        if (edit_mode && event.key.code == Keyboard::C)
        {
            cout<< "Killed all entities." <<endl;
            killall(squares);
        }
        if (event.key.code == Keyboard::E)
        {
            edit_mode = !edit_mode;
            if (edit_mode)
            {
                cout<< "Changed to edit mode."<<endl;
            }
        }
    }
}
```

```
        {
            cout<< "Changed to simulation mode."<<endl;
        }

    }

/*s.draw(window);
s.square->setPosition(Vector2f(s.square->getPosition().x+1,s.square->
getPosition().y));*/
g.draw(window);
for(int i=0;i<h;i++)
{
    for( int j=0; j<w; j++)
    {
        squares[i][j]->draw(window);
    }
}
window.display();
if(!edit_mode) squares = update(squares);
for(int i=0;i<h;i++)
{
    for( int j=0; j<w; j++)
    {
        squares[i][j]->update();
    }
}
}

return EXIT_SUCCESS;
}
```

Fordítás: "g++ *.cpp -o sfml-app -lsfml-graphics lsfml-window -lsfml.system". Amikor már a programunk fut , az esetben kézzel kell rajzolnunk egy alakzatot , majd nyomni egy "e" betűt , aminek következtében a program elkezdi az "életjátékot".

A program SFML ablakozó rendszerre épül. Először létrehozzuk az ablakunkat, majd megrajzoltatjuk a rács-pont rendszert. A update () függvénybe van megírva az egyes szabályok mentén történő matematikai műveletek végrehajtása. Ez ha a szabály teljesül visszatér egy Alive értékkel, ami jellemzi a rácspontot. A többi az SFML-hez tartozó kódokat mutatja. Úgy, mint az egéresemények, kirajzoltatás update stb.

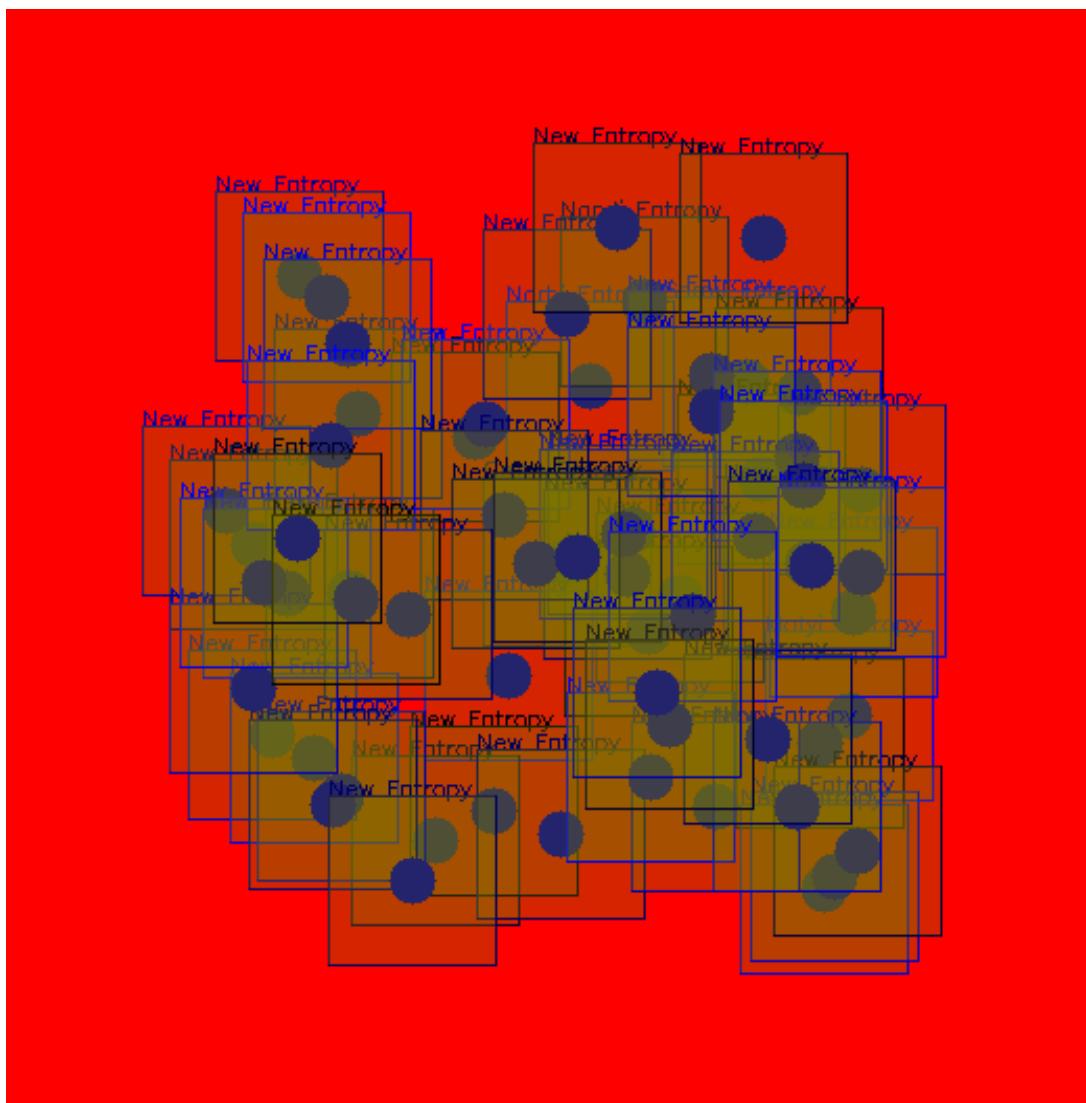
Idő híján a csak a lényegi részt írtam ki. Később bővebben is kifejtem a program működését, mint Java, mint C++ verziókban.

7.4. BrainB Benchmark

Ez a program vagy játék egy készségmérő program, ami azt méri, hogy egy bizonyos objektumot mennyire tudunk lekövetni. Kezdetben egy négyzetben lévő karikára kell rafocuszálni. Cél, hogy az egérgomb folyamatos nyomvatartása mellett kövessük a Samu Entropy nevű objektumot a kurzorral. Bizonyos időközönként új négyzetek (objektumok) jelennek meg a képernyőn, ami nagyban nehezíti a Samu objektum követését. Illetve az egyes objektumok mozgása, rezgése is megnőhet.

A játék során erős koncentrációs és reagálási képesség kell.

Ehhez hasonló képességfelmérés létezik a League of Legends játékon belül is, amit elsőként Veres Dávid Msc hallgató munkájában láttam. A játékon belül különböző behatások érnek minket játék közben. (pl. teamfight, roam, active items stb.) Ezekre mindenkorban figyelni kell, ha hasznos tagjai akarunk lenni a csapatunknak. Ez nyilván nem könnyű így sok gyakorlást és odafigyelést igényel az egyes objektumok követése, figyelése.



7.2. ábra. Bátfai Norbert ábrája a BrainB-ről.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Először is, hogy használni tudjuk le kell töltenünk a TensorFlow-t. A TF telepítéséhez nem kell külön ügyködni, elég ha felmegyünk a hivatalos oldalára és onnan az útmutatók segítségével megcsináljuk. A program lényege, hogy 1-9 ig számokat mutató kis 28x28 as képekből fel kell ismernie az éppen aktuális számjegyet. Ez ugye 784 pontot, azaz 784 db számot jelent. Ezt a 784 számot felfoghatjuk úgy is, mint egy pont koordinátáit a 784 dimenziós térben. Az eredmény meg ugye 0-9-ig egy szám, pontosabban 10 db érték, ami azt mondja meg, hogy rendszerünk az adott bemenetre milyen számot tippel. Ugye ha jó a rendszer, akkor arra a számra fogja mondani a legnagyobb esélyt, ami oda van írva. Ezt úgy kell elképzelni, hogy mondjuk egy írott 6-osra azt mondja, hogy 10%, hogy 8, 20% hogy 9, és 70%, hogy 6-os számot lát.

A Szoftmax python kódja:

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License" ↵
# );
# you may not use this file except in compliance with the ↵
# License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, ↵
# software
# distributed under the License is distributed on an "AS IS" ↵
# BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express ↵
# or implied.
# See the License for the specific language governing ↵
# permissions and
# limitations under the License.
# ↵
=====
```

```
# Norbert Batfai, 27 Nov 2016
# Some modifications and additions to the original code:
# https://github.com/tensorflow/tensorflow/blob/r0.11/←
# tensorflow/examples/tutorials/mnist/mnist_softmax.py
# See also http://progpater.blog.hu/2016/11/13/←
# hello_samu_a_tensorflow-bol
# ←
=====

"""A very simple MNIST classifier.

See extensive documentation at
http://tensorflow.org/tutorials/mnist/beginners/index.md
"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse

# Import data
from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

import matplotlib.pyplot

FLAGS = None

def readimg():
    file = tf.read_file("sajat8a.png")
    img = tf.image.decode_png(file)
    return img

def main(_):
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y = tf.matmul(x, W) + b

    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])
```

```
# The raw formulation of cross-entropy,
#
#   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y))) ←
#   ,
#                   reduction_indices=[1]))
#
# can be numerically unstable.
#
# So here we use tf.nn.softmax_cross_entropy_with_logits on the ←
#   raw
# outputs of 'y', and then average across the batch.
cross_entropy = tf.reduce_mean(tf.nn. ←
    softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize( ←
    cross_entropy)

sess = tf.InteractiveSession()
# Train
tf.initialize_all_variables().run()
print("-- A halozat tanitasa")
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
        print(i/10, "%")
print(" -----")

# Test trained model
print("-- A halozat tesztelese")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1)) ←
    )
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf. ←
    float32))
print("-- Pontossag: ", sess.run(accuracy, feed_dict={x: mnist. ←
    test.images,
                           y_: mnist.test.labels}))
print(" -----")

print("-- A MNIST 42. tesztkepenek felismerese, mutatom a ←
      szamot, a tovabbelipeshez csukd be az ablakat")

img = mnist.test.images[42]
image = img

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib. ←
    .pyplot.cm.binary)
matplotlib.pyplot.savefig("4.png")
```

```
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image ↪
] })

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print(" ↪
-----")

print("-- A saját kezi 8-asom felismerese, mutatom a szamot, a ↪
tovabbolteshez csukd be az ablakat")

img = readimg()
image = img.eval()
image = image.reshape(28*28)

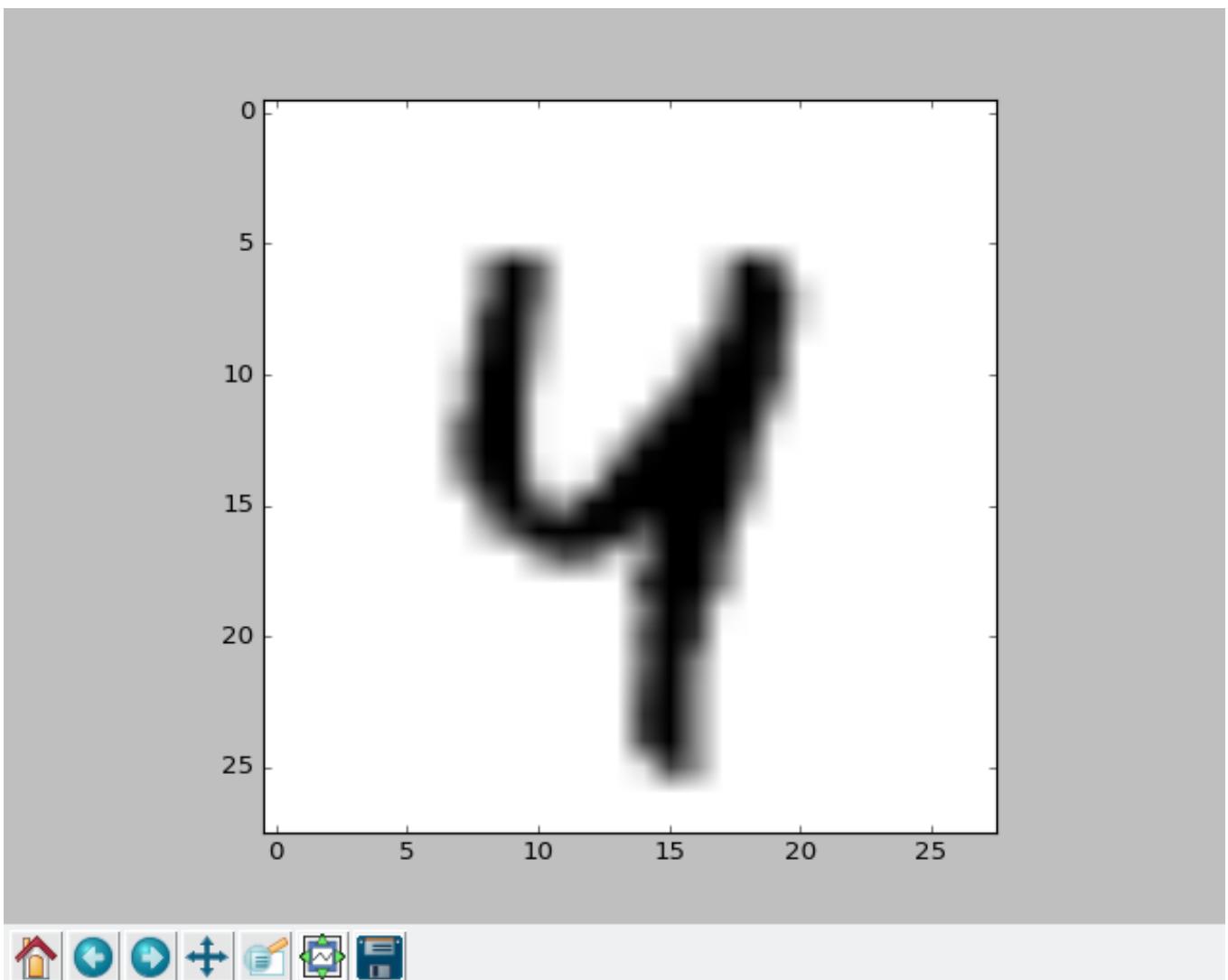
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib ↪
    .pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image ↪
] })

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print(" ↪
-----")

if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--data_dir', type=str, default='/tmp/ ↪
tensorflow/mnist/input_data',
                    help='Directory for storing input data')
FLAGS = parser.parse_args()
tf.app.run()
```

Az fenti kód két részre bontható. Van az első rész, ahol a feltanítjuk a hálózatunkat a felismerni kívánt "objektumokkal". Feltölti a képet, majd ezek alapján egy bizonyos pontosságot belőve meghatározza, hogy az épp milyen objektum. A második rész a tesztelése a hálózatnak, aholis felhasználói inputokat vizsgál a hálózat és eldönti, hogy az inputon melyik számjegy található. Futtatni a python értelmezővel tudjuk. Futtatás után a felismert számot kiírja a kimenetre.



8.1. ábra. Bátfai tanár úr ábrája a megjelenített számokról a MNIST-ben.

8.2. Mély MNIST

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.  
#  
# Licensed under the Apache License, Version 2.0 (the "License"  
#);  
# you may not use this file except in compliance with the  
# License.  
# You may obtain a copy of the License at  
#  
#     http://www.apache.org/licenses/LICENSE-2.0  
#
```

```
# Unless required by applicable law or agreed to in writing, ←
# software
# distributed under the License is distributed on an "AS IS" ←
# BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express ←
# or implied.
# See the License for the specific language governing ←
# permissions and
# limitations under the License.
# ←
=====

"""A deep MNIST classifier using convolutional layers.
See extensive documentation at
https://www.tensorflow.org/get_started/mnist/pros
"""

# Disable linter warnings to maintain consistency with tutorial ←
.
# pylint: disable=invalid-name
# pylint: disable=g-bad-import-order

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys
import tempfile

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

FLAGS = None


def deepnn(x):
    """deepnn builds the graph for a deep net for classifying ←
    digits.

Args:
    x: an input tensor with the dimensions (N_examples, 784), ←
        where 784 is the
        number of pixels in a standard MNIST image.

Returns:
    A tuple (y, keep_prob). y is a tensor of shape (N_examples, ←
        10), with values
    equal to the logits of classifying the digit into one of 10 ←
        classes (the
```

```
    digits 0-9). keep_prob is a scalar placeholder for the ↵
        probability of
        dropout.
    """
# Reshape to use within a convolutional neural net.
# Last dimension is for "features" - there is only one here, ↵
# since images are
# grayscale -- it would be 3 for an RGB image, 4 for RGBA, etc.
with tf.name_scope('reshape'):
    x_image = tf.reshape(x, [-1, 28, 28, 1])

# First convolutional layer - maps one grayscale image to 32 ↵
# feature maps.
with tf.name_scope('conv1'):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

# Pooling layer - downsamples by 2X.
with tf.name_scope('pool1'):
    h_pool1 = max_pool_2x2(h_conv1)

# Second convolutional layer -- maps 32 feature maps to 64.
with tf.name_scope('conv2'):
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)

# Second pooling layer.
with tf.name_scope('pool2'):
    h_pool2 = max_pool_2x2(h_conv2)

# Fully connected layer 1 -- after 2 round of downsampling, our ↵
# 28x28 image
# is down to 7x7x64 feature maps -- maps this to 1024 features.
with tf.name_scope('fc1'):
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Dropout - controls the complexity of the model, prevents co- ↵
# adaptation of
# features.
with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Map the 1024 features to 10 classes, one for each digit
```

```
with tf.name_scope('fc2'):
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])

    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
    return y_conv, keep_prob

def conv2d(x, W):
    """conv2d returns a 2d convolution layer with full stride."""
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    """max_pool_2x2 downsamples a feature map by 2X."""
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')

def weight_variable(shape):
    """weight_variable generates a weight variable of a given shape"""
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    """bias_variable generates a bias variable of a given shape."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def main(_):
    # Import data
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])

    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])

    # Build the graph for the deep net
    y_conv, keep_prob = deepnn(x)

    with tf.name_scope('loss'):
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
            logits=logits,
            labels=y_,
            = = = = =
```

```
    y_conv ←  
) ←  
  
cross_entropy = tf.reduce_mean(cross_entropy)  
  
with tf.name_scope('adam_optimizer'):  
    train_step = tf.train.AdamOptimizer(1e-4).minimize( ←  
        cross_entropy)  
  
with tf.name_scope('accuracy'):  
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf. ←  
        argmax(y_, 1))  
    correct_prediction = tf.cast(correct_prediction, tf.float32 ←  
        )  
    accuracy = tf.reduce_mean(correct_prediction)  
  
graph_location = tempfile.mkdtemp()  
print('Saving graph to: %s' % graph_location)  
train_writer = tf.summary.FileWriter(graph_location)  
train_writer.add_graph(tf.get_default_graph())  
  
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())  
    for i in range(20000):  
        batch = mnist.train.next_batch(50)  
        if i % 100 == 0:  
            train_accuracy = accuracy.eval(feed_dict={  
                x: batch[0], y_: batch[1], keep_prob: 1.0})  
            print('step %d, training accuracy %g' % (i, ←  
                train_accuracy))  
        train_step.run(feed_dict={x: batch[0], y_: batch[1], ←  
            keep_prob: 0.5})  
  
        print('test accuracy %g' % accuracy.eval(feed_dict={  
            x: mnist.test.images, y_: mnist.test.labels, keep_prob: ←  
                1.0}))  
  
    if __name__ == '__main__':  
        parser = argparse.ArgumentParser()  
        parser.add_argument('--data_dir', type=str,  
                            default='/tmp/tensorflow/mnist/input_data',  
                            help='Directory for storing input data')  
        FLAGS, unparsed = parser.parse_known_args()  
        tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

A deep learning (mély tanulás) paradigmán alapuló mély neurális hálózat. A fentiekhez képest több különbség is adódik..

8.3. Minecraft-MALMÖ

Passzolás - Ezt a feladatot a SMNIST kutatásban való részvétellel passzoltam! Link: <https://bit.ly/2HaAhAB>

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Ahhoz, hogy tudjuk hackelni a GIMP-et tisztába kell lenni néhány kifejezéssel.

Kifejezések:

A lisp változók és minden kifejezést kerek zárójelek () közé kell tenni. A zárójelen belüli kifejezés(ek) meghatározott sorrendet követnek. pl.: (- 5 5) A kifejezések mindig egy függvénynel kezdődnek majd utána a megfelelő paraméterek. A Scheme nem veszi figyelembe a szóközöket így szóval külön is tudjuk írni az egyes kifejezéseket. További kifejezés lehet: (* (+ 5 5) (- 10 5))

Függvények:

Függvényeket a `define` kulcsszó segítségével definiálunk. Zárójelek között megadjuk a `define` kulcsszót ezzel jelezve hogy függvénydefinicíó következik, majd név és utánaírjuk a kifejezést. pl.: (`(define (square x) (* x x))`)

Ezek alapján már egyszerűen megtudjuk írni a faktoriális függvényt Lisp-ben.

Ahogy írtuk a függvénydefinicíó a `define` kulcsszóval kezdődik majd megadjuk a fg nevét (`fakt n`). Ezután írjuk a függvény "törzsét", ami kiszámolja az n faktoriálisát. Egybepakolva így néz ki:

```
(define (fakt n) (if(< n 0) 1 (* n (fakt (- n 1)))))
```

9.2. Gimp Scheme Script-fu: króm effekt

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

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome

A program egésze így néz ki:

```
; bhax_chrome3.scm
;
; BHAX-Chrome creates a chrome effect on a given text.
; Copyright (C) 2019
; Norbert Bátfa, batfai.norbert@inf.unideb.hu
; Nándor Bátfa, batfai.nandi@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 <https://www.gnu.org/licenses/>.
;
; Version history
;
; This Scheme code is partially based on the Gimp tutorial
; http://penguinpeta.com/b2evo/index.php?p=351
; (the interactive steps of this tutorial are written in ←
Scheme)
;
; https://bhaxor.blog.hu/2019/01/10/ ←
a\_gimp\_lisp\_hackelese\_a\_scheme\_programozasi\_nyelv
;

(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)
)

; (color-curve)

(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)
  )
)
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-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)
  )
  ;
  ; 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))
```

```
(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)))

;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE ←
    )

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 ←
    0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 ←
    0 0))
(gimp-selection-invert image)

;step 6
(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)

;step 7
(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)))

;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 ←
    25 7 5 5 0 0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

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

```
; (script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 ←
  '(255 0 0) "Crown molding")

(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)0"
  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 program úgy kezdődik, hogy definiálunk egy `color-curve` függvényt. A let kulcsszóval megadunk egy lokális változót, ami egy 8 elemű tömb lesz. Ezután feltöltjük az értékeit különböző értékekkel. Ez lesz a színátmenetért felelős függvény.

```
(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)
)

; (color-curve)
```

A függvény 3 paramétert vár. Magát a szöveget, amit formázni szeretnénk. A szöveg betűstílusát illetve a szöveg méretét. `set!` kulcsszóval beállítunk értékeket a változóknak és a változók globális értékekével válnak. Létrehozunk két változót `text-width` illetve `text-height`-t és beállítjuk az értékeiket 1-re. Majd a `set!` segítségével beállítjuk a további értékeket a paraméterként megkapott 3 érték alapján.

```
(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)
  )
; (text-width "alma" "Sans" 100)
```

Az alábbi programban fog megtörténni a chrome effect leimplementálása. A script-fu-bhx-chrome függvény 7 paramétert vár. Ezek a következők: (script-fu-bhx-chrome "formázandó szöveg" "betűstílus" betűméret szélesség magasság színskála "színezési stílus") A továbbiakban írni fogok még róla. Időhiány stb.

```

(define (script-fu-bhax-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)
)
; 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))

(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)))

```

```
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE ←
)

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 ←
 0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 ←
 0 0))
(gimp-selection-invert image)

;step 6
(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)

;step 7
(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)))

;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 ←
  25 7 5 5 0 0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 ←
' (255 0 0) "Crown molding")
```

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala

Ebben a programban egy mandalát készítünk Scheme segítségével Lisp nyelven. A mandala egy kör-körös stílusú vallási alagzat. A programban egy szöveget fogunk használni a mandala elkészítéséhez. Úgy működik, hogy a fu-bhax-mandala függvény a megadott paraméterek segítségével előállítja a mandalát. A mandalához forgatást használezen kód segítségével. (`(gimp-item-transform-rotate text-layer (/ (*pi* 4) TRUE 0 0)`) Az elforgatott formázott szöveg után megkapjuk a mintát. A kódot illetve lefuttatva az Új/ Létrehozás menüpont alatt GUI-s interface-n tudjuk megadni a paramétereineket.

```
; bhax_mandala9.scm
;
; BHAX-Mandala creates a mandala from a text box.
; Copyright (C) 2019 Norbert Bátfai, 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
;
; This Scheme code is partially based on the Python code
; Pat625_Mandala_With_Your_Name.py by Tin Tran, which is released under the GNU GPL v3, see
; https://gimplearn.net/viewtopic.php?f=1&t=Pat625-Mandala-With-Your-Name-Script-for-GIMP?t=269&p=976
;
; https://bhaxor.blog.hu/2019/01/10/a\_gimp\_lisp\_hackelese\_a\_scheme\_programozasi\_nyelv
;
```

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-width text font fontsize)
(let*
(
  (text-width 1)
)
(set! text-width (car (gimp-text-get-extents-fontname text ←
  fontsize PIXELS font)))
text-width
)
)

(define (text-wh text font fontsize)
(let*
(
  (text-width 1)
  (text-height 1)
)
;;;
(set! text-width (car (gimp-text-get-extents-fontname text ←
  fontsize PIXELS font)))
;;; ved ki a lista 2. elemét
(set! text-height (elem 2 (gimp-text-get-extents-fontname ←
  text fontsize PIXELS font)))
;;
(list text-width text-height)
)
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width ←
height color gradient)
(let*
(
  (image (car (gimp-image-new width height 0)))
  (layer (car (gimp-layer-new image width height ←
    RGB-IMAGE "bg" 100 LAYER-MODE-NORMAL-LEGACY)))
  (textfs)
  (text-layer)
  (text-width (text-width text font fontsize))
)
;;;
```

```
(text2-width (car (text-wh text2 font fontsize)))
(text2-height (elem 2 (text-wh text2 font fontsize)))
;;
(textfs-width)
(textfs-height)
(gradient-layer)
)

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

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

(gimp-context-set-foreground color)

(set! textfs (car (gimp-text-layer-new image text font ←
    fontsize PIXELS)))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width ←
    2)) (/ height 2))
(gimp-layer-resize-to-image-size textfs)

(set! text-layer (car (gimp-layer-new-from-drawable textfs ←
    image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE
0 TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-l
r CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable
tfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 2) TR
0 0)
(set! textfs (car (gimp-image-merge-down image text-layer ←
CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs ←
    image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer ←
CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs ←
    image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
```

```
(set! textfs (car(gimp-image-merge-down image text-layer ←
CLIP-TO-BOTTOM-LAYER)))

(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car(gimp-drawable-width textfs)) ←
100))
(set! textfs-height (+ (car(gimp-drawable-height textfs)) ←
100))

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

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

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

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

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width ←
height RGB-IMAGE "gradient" 100 LAYER-MODE-NORMAL-LEGACY ←
)))
(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM ←
LAYER-MODE-NORMAL-LEGACY GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ ←
height 2) (+ (+ (/ width 2) (/ textfs-width 2)) 8) (/ ←
height 2))

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(set! textfs (car (gimp-text-layer-new image text2 font ←
fontsize PIXELS)))
```

```
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ ←
    text2-width 2)) (- (/ height 2) (/ text2-height 2)))  
  
;(gimp-selection-none image)
;(gimp-image-flatten image)  
  
(gimp-display-new image)
(gimp-image-clean-all image)
)  
)  
  
;(script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Ruge Boogie" ←
  120 1920 1080 '(255 0 0) "Shadows 3")  
  
(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"        "Bátf41 Haxor"
  SF-STRING      "Text2"       "BHAX"
  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"   "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

10. fejezet

Helló, Gutenberg!

10.1. PICI Juhász István

A programozásban használatos nyelvek közül több szintet különböztetünk meg. gépi nyelv assembly szintű nyelv magas szintű nyelv Az előbbi 2 szintről csak említés szintén tanultunk, hiszen a mai időkben a legelterjedtebbek a magas szintű nyelvek. minden magas szintű nyelvet a szintaktika és a szemantika határoz meg. A szintaktika a nyelv által lefektetett és jól meghatározott szabályok gyűjteménye. Ezek formai, nyelv specifikus követelmények. Pl.: Szintaktikai hiba lehet egy ; záró tag elhagyása. A szemantikai szabályok pedig a tartalmi, jelentésbeli formális meghatározások. Pl.: Szemantikai hiba esetén a programunk futhat viszont nem várt működést eredményezhet. minden program az őt futtató processzor utasításkészlete alapján fordul. (gépi kód) Alapesetben a programunk kódja nem gépi kódon íródott így át kell alakítani a forráskódokat a gép számára értelmezhető nyelvezetűre. Erre két megoldás adott a magas szintű nyelvek esetében. fordítóprogramos nyelv interpreteres nyelv A fordítóprogram a forráskódból gép kódú úgynévezett tárgy kódot állít elő. Ez m ég nem futtatható így egy kapcsolatszerkesztő előállítja a tárgykód ból a megfelelő futtatható kód állományt. Interpreteres nyelv esetében pedig a forráskódot sorról sorra értelmezi és hajta végre. Tipikus interpreteres nyelv a Java, ahol a forráskódból köztes .class kód majd gépi kód állítódik elő. minden programozási nyelvhez létezik olyan IDE (Integrated Developer Environment), ami nagyban segíti az adott nyelvben történő hatékony programozást. IDE funkciók a kódszínezés, kódkiegészítés, debuggolás, tesztelés stb. Eddigi tanulmányaim során többnyire objektumorientált nyelvekben programoztam, ami az imperatív nyelvi programozás csoportjába tartozik. Valamint ide tartozik még az eljárásvorientált nyelvek is. Viszont sose hallottam még deklaratív nyelvekről. Ezek többnyire a programozó által meghatározott problémára keresik a megoldást a nyelvi implementációk segítségével. Ezek nem algoritmikus nyelvek.

Az adatabsztraktió első formája az adattípus. Két fajta adattípust különböztetünk meg vannak az egyszerű vagy primitív típusok és az összetett típusok. Az egyszerű típusok közé tartoznak a különféle egész- és lebegőpontos típusok, valamint a logikai típusok. Ezek többnyire literállal rendelkeznek. A másik csoportba tartoznak a jóval összetettebb adatszerkezetek. Ilyen lehet pl. egy felhasználó által definiált adattípus vagy egy tömb.

Három adattagja van { név, típus, érték }. Az ilyen változót a kezdőértékkadás után nem lehet megváltoztatni.

A változónak négy komponense van: a név, az attribútumok, a cím és az érték. A deklarált változók, metódusok, konstansok stb. nevével tudunk hivatkozni a változókra. Különböző megszorítások vannak, amik megszabják a változók szintaktikáját. 1. nem kezdődhet speciális karakterrel. 2. nem lehet már a nyelv által definiált változóneveket adni 3. használhatunk számokat is a változók nevében.

C-ben az aritmetikai típusok az egyszerű típusok, a származtatottak az összetett típusok vannak. Aritmetikai típusok az int, short, long, double, float. A karakteres típusok a char és ide tartozhat a string is ami char-ok tömbje.

Kifejezések olyan "szintaktikai" eszközök, amelyekkel új értékeket adhatunk különböző a programon belül található kód részletekből. Két részből épül fel, az érték és típusból. Kifejezések szintaktikai eszközök, melyekkel új értékekkel adhatunk különböző kódrészletekhez (kifejezésekhez). Összetevői lehetnek operandusok, operátorok, kárójelek akár kapcsos akár szöglletes. Operandusok számát tekintve 3 altípust különböztetünk meg a nyelvben. Az egyoperandusú (pl. ++a), a kétoperandusú (pl. a + a), és a három-operandusú, ami jellegzetesen lehet egy "kicsi if". (pl. ez ? az : macika).

Az utasítások alkotják meg a program egységét. Két csoportjuk van a deklarációs utasítások, és a végrehajtó utasítások. A végrehajtó utasításokból pedig a fordító generálja a kódot. A végrehajtó utasítások a következők: értékkadó, üres ,ugró , elágaztató utasítás stb. A végrehajtó utasítások a ciklusszervező, elágaztató, feltételes, többfeltételes utasítások stb.

A paraméterátadásnak többféle módja is lehet, ezek nyelvfüggőek, hogy melyik nyelv melyiket alkalmazza.

Történhet érték szerint, mint a C-ben például. Ekkor a formális paraméter értékül kapja az aktuális paraméter értékét. Ennél a módszernél a függvényben nem lehet megváltoztatni a aktuális paraméter értékét. Lehet címszerinti a paraméterátadás. Ekkor a formális paraméter címe értékül kapja az aktuális paraméter címét. Ilyenkor a függvényben meg lehet változtatni az aktuális paraméter értékét. Lehet eredmény szerinti átadás is, ekkor a formális paraméter szintén megkapja az aktuális paraméter címét, de nem használja, csak a végén beletölti az adatokat. Létezik még érték-eredmény szerinti, ekkor másolódik a cím szintén, és használja is az adatokat, majd a függvény végén belemásolja a formális paraméterbe az adatokat.

I/O műveleteket különböző input és outputból érkező file vagy/és perifériák kezelésére alkalmazzuk. Akkor alkalmazzuk pl. ha olvasni vagy írni szeretnénk egy file-ba. Vagy esetleg a programunkba egy webkamera által szolgáltatott képet szeretnénk kezelní. A C-ben az IO nem eszköze a rendszernek. Ezeket csak könyvtári függvényekként tudjuk elérni. Az I/O függvények minimálisan eg y karakter vagy karaktercsoport, valamint egy bájt vagy bájtcsoport írását és olvasását adja a kezünkbe.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

V.heti előadás - Vezérlési szerkezetek

Egy nyelv vezérlésátadó utasításai az egyes műveletek végrehajtási sorrendjét határozzák meg. Ha egy kifejezés után pontosvesszőt (;) rakunk akkor az utasítás lesz. Ezeket egy idő után automatikusan alkalmazzuk viszont ha elhagyjuk nagy eséllyel hibát okoz. A ; jel elhagyása tipikus kezdő programozók hibája. :) Vannak utasítások, amelyekhez több utasítás párosulhat, ezeket blokokba rendezzük a kapcsos zárójelek ({}) segítségével. IF_ELSE utasítás egy elágazás, amely egy feltételt vizsgálva vagy az igaz ágba vagy a hamis ágba fordul. (Szintaxis: if(feltétel) {feltétel teljesülése esetén} else {különben}). Van több ágat vizsgáló IF_ELSE IF_ELSE szerkezet is. Ez többféle feltételt vizsgál és több ágba fordul. (Szintaxis: if(feltétel) {feltétel teljesülése esetén} else if(feltétel2) {feltétel2 teljesülése esetén} else {különben}). Létezik többágú elágazás ez a SWITCH. A switch egy feltételt vizsgálva több ágba is fordulhat a feltétel teljesülése esetén. Szintaxisa a switch(feltétel) majd ezt követi a törzs. Itt különböző case ágakba vannak definiálva a lehetséges feltételnek megfelelő értékek és az azokhoz tartozó utasítások. minden case ágat egy (;break;) zár. Ha nem illeszkedik egyik megadott értékre sem, akkor a default ágba fordul. Elágazások

után tipikus vezérlési szerkezetek a ciklusok. Ezek közül hármat különböztetünk meg. A for ciklust, az elől tesztelő ciklust(while) illetve a hátul tesztelő ciklust (do while). FOR ciklus áll egy ciklusfejből. Itt három lezáró taggal szeparált tagok kell megadni. A ciklus kezdeti értékét a végpontot illetve a egy ciklus alatti lépés számát vagy "mértékét" (pl. i++). Ha a fej megvan ugyanúgy, mint az if-nél meg kell adnunk a ciklus törzsét kapcsos zárójelek között. WHILE ciklus először egy feltételt vagy feltételeket vizsgál és ezek alapján, ha teljesül megismétli önmagát különben meg kilép a ciklusból. A feltétel után megadjuk a ciklus törzsét kapcsos zárójelekben. DO WHILE ciklus a while-hoz hasonlóan egy feltételt vizsgálva ismételgeti önmagát, viszont itt a feltételvizsgálat a ciklusmag után helyezkedik el. Ez azt jelenti, hogy amit megadtunk a magban az egyszer mindenkorban lefordul, majd utána vizsgálja a feltételt.

10.3. Programozás

[BMECPP]

Míg C-ben egy függvény deklaráció üres paraméterlistája tetszőleges számú paramétert eredményezhet ugyanez C++-ba a paraméterként megadott void kulcsszó segítségével történik. További ilyen C++ tetszőleges paraméter lehet a (...) paraméter definiálás. A program fő lefutási és indulási pontja a main metódus. Ezt kétféleképpen is definiálhatjuk. Üres paraméterrel (pl. int main()) vagy a paraméterben megadott parancssori argumentumokkal illetve azon számával. (pl. int main(int argc, char* argv[])). Hasonlóképpen, mint a többi magasszintű nyelvben változó deklarációt célszerű ott használni, ahol utasítás áll és használja azt. Ha nem így teszünk akkor warning-ot kaphatunk, ami arra figyelmeztet, hogy nem használt változót deklaráltunk. Vannak olyan előre megírt függvények, amelyeknek alapértelmezett léteznek paraméteri argumentumai, ezeket meg kell adnunk, ha fel szeretnénk használni az adott függvényt. C-ben kizárálag csak érték szerinti paraméter átadás történhet ezzel szemben a C++-ban lehetőség van referencia szerinti paraméterátadásra is. Az OOP lényegeként azt tekintjük, hogy előre megírt felhasznált elemektől építjük fel programunkat. Elemezzük a világot, a környezetet és egy kép alapján lemodellezük azt. Alapfogalmak: osztály - egy hasonló mondon kezelendő objektum halmaz. Objektum - Az osztály egy példánya. Az osztályokat class kulcsszóval hozzuk létre. Az osztálynak több tagja is lehet. Ilyenek a mezők, paraméterek, metódusok ami lehet eljárás vagy függvény. Az OOP-s nyelvek három alap jellemzővel bírnak: Egységbázis, Öröklődés, Többalakúság. Az adatainkat egy egységek szeretnénk kezelni, ezeket szabályozni akarjuk, hogy az egységen kívül más ne tudjon belebabrálni. Az így kapott egységek az objektumok. Ha egy osztályt létrehozunk akkor annak egy példányát szeretnénk máshol is felhasználni, erre szolgál az öröklődés. A szülő - leszármazott viszonyban a leszármazott objektum örökli a szülő minden egyes tulajdonságát. Vannak esetek, amikor a leszármazott felül tudja definiálni a szülő tulajdonságait ezzel egy önmagára specifikus tulajdonságot létrehozva. Ezt nevezzük többalakúságnak. Pl. egy ősosztály, az állnak a mozgás tulajdonságát írtuk le. Ezt a tulajdonságot hozzárendelhetjük más különböző objektumokhoz osztályokhoz is pl.: halak, emlősök akiknek ugyanúgy szükségük van a mozgás tulajdonságára.

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.

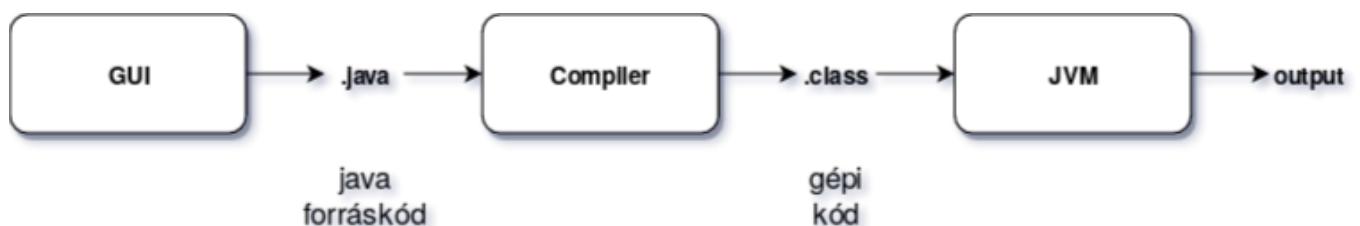
11. fejezet

Helló, Olvasónapló!

11.1. Java illetve C++ összehasonlítása

A feladat a C++ könyv illetve a Java I-II könyv összehasonlítása volt. Az esszében pár oldalas formában összefoglalom, hogy milyen eltérő nyelvi megvalósításokat találhatunk, mint Java, mint C++ oldalon. Elsőkörben a főbb különbségeket tárgyaljuk a két nyelv között majd áttérek az egyéb szintaktikai, szemantikai különbségekre is.

Mindkét nyelv széles körben alkalmazható különböző feladatok, problémák megoldására. A C++ fő vonzeréje, hogy lehetővé teszi a forrás szinten hordozható programok írását. Ez bővebben annyit takar, hogy egy másik gépen a megfelelő platform mellett a program ugyanolyan működéssel bír. A C++ forráskód gépi nyelvre konvertálódik így az csak az adott platformon fordul, ezt hívjuk platform-függő megvalósításnak. Ezzel szemben a Java egy úgynevezett JVM futtató környezetet használ, ami minden platformra egységes program futást eredményez. A java forráskódot a javac compiler egy java virtuális környezetben futatható byte kóddá (.class) fordítja a programokat. A Java így tehát egy platform-független nyelv.



11.1. ábra. Java compile

A Java nyelvi szinten támogatja az egyes osztálykönyvtárakat így széles alkalmazási területet fed le, úgy mint a szálkezelés, GUI programozás, adatbázis elérés stb. A C++ hasonlóan támogatja ezen alkalmazásokat viszont jobban igénybe kell venni a külső, nem nyelvi szintű könyvtárakat.

A C++ nyelv az egyes objektumokat egy memóriaszegmensen elhelyezkedő bájtsorozatnak fogja fel, amiket a mutatók, referenciaiák segítségével könnyedén manipulálhatjuk a memóriában. Ezen esetben beszélünk statikus-, automatikus- és dinamikus memóriáról. A programozó legtöbbször a dinamikus memóriában "matat", ide történik meg a helyfoglalás illetve az egyes felszabadítási mechanizmusok. Ezzel szemben

a Java-ban nincs lehetőség közvetlenül elérni a memóriát, ahogyan a C++-ban tettük. Helyette hivatkozásokon keresztül érjük el a memória tartalmát. Ugyanezen témát érintve a destruktur-hívó mechanizmus hiányzik a Java-ban, helyette egy úgynevezett garbage-collector (szemétgyűjtő mechanizmus) automatikusan menedzseli a memóriát. Ezen mechanizmus átveszi a programozótól a memória kezelés problémáját néhány előnyt és hátrányt hagyva maga után. Előnyei többek között a memóriászemét eltakarítása, memóriariaszivágás megakadályozása és egyéb memória menedzselési feladatok automatikus elvégzése. Hátránya, hogy a szemétgyűjtés stop-the-world módon történik. Az összes Java szálat leállítja azért, hogy ne változzon a heap és a stack. Ez a leállítás sok párhuzamos felhasználás mellett nagymértékben rontja a teljesítményt. Az egyes szemétgyűjtő algoritmusok eltérőek lehetnek. (A garbage collector mechanizmusról pontosabb infókat a (<http://www.jtechlog.hu/2011/12/30/java-memoriakezeles-szemetgyujto.html> oldalon olvastam).

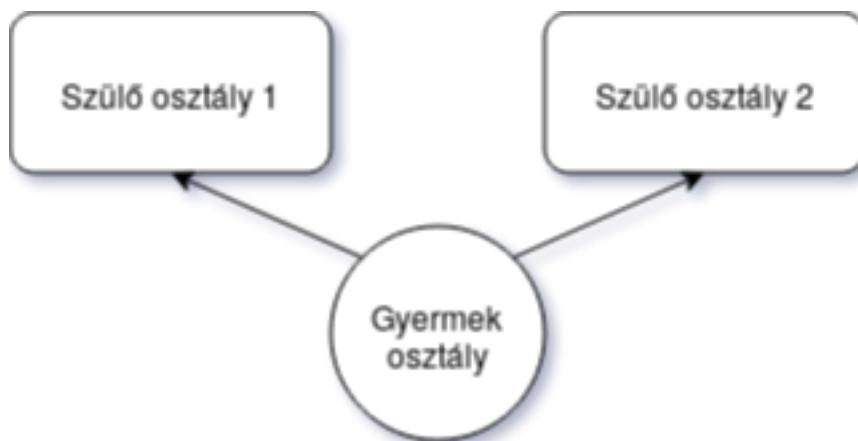
A C++ nyelvben definiált primitív típusok pontos mérete és értéktartománya nem definiált. Ez a hatékony felhasználást célozza meg. Jó példa a char primitív típus (signed/unsigned) tulajdonsága vagy int (16/32/64 bit) mérete. A Java ezeknél szigorúbb szabályokat alkalmaz a primitív típusok definiálására. Mivel futási időben nem hozhatunk létre primitív változókat így a Java bevezette az úgynevezett primitív típusoknak megfelelő Csomagoló osztályokat. Ezek az osztályok tartalmaznak primitív típust fogadó konstruktort és primitív típusra konvertáló metódust valamint a főbb konstansokat. Főbb olyan típusra jellemző metódust és függvényt is definiálnak ezek az osztályok, amik segítik a velük való objektum szintű műveleteket.

C++	Java	Csomagolóosztály	méret
bool	boolean	Boolean	logikai
char	-	-	8 bites karakter
wchar_t	char	Character	16 bites Unicode karakter
signed char	byte	Byte	8 bites előjeles szám
short	short	Short	16 bites előjeles szám
long	int	Integer	32 bites előjeles szám
long long	long	Long	64 bites előjeles szám
float	float	Float	32 bites lebegőpontos szám
double	double	Double	64 bites lebegőpontos szám
long double	-	-	> 64 bites lebegőpontos szám

11.2. ábra. Java könyvből való típus összegző táblázat

Java-ban a logikai típushoz nem tartozik egész típusú megfelelő. Pl.: A hamis ! = 0 vagy a igaz ! = 1. A C++-ban bátran tudunk konvertálni ezen két típus között. A Java nyelvben nem definiált a pointer így ennek megfelelően nem tudunk a C++-tól megszokott pointer vagy referencia szintaxist használni a dinamikus memória elérésére. Ezeket csak hivatkozásokon keresztül érhetjük el. Javában a függvénymutatók helyett objektum-referenciákat, visszatérési értékeket, tömböket és interface-ket használunk. A C++-beli const kulcsszó definiált viszont nincs jelentése a Java-ban. Hasonló jelentéssel a final kulcsszó rendelkezik, amely az adattagok, metódusok változatlanságát illetve az osztályra vonatkozó származtatás tiltását jelenti. Sok osztály rendelkezik final minősítéssel ugyanígy az összes primitív típus Csomagoló osztálya is.

Az objektum orientált szemléletmódot minden nyelv támogatja viszont néhány eltérést találhatunk. A C++ többparadigmás (többértelmű) nyelv. Támogatja a procedurális programokat, bonyolult osztályhierarchiákat, objektum orientált elveket, könyvtárakat. A Java nyelv ezzel szemben csak az objektum orientált paradigmát támogatja. Nincsenek globális változók, függvények csak osztály attribútumok, változók, konstansok és metódusok. A C++-ban lehetőség van az egyszeres illetve többszörös öröklődésre. Java-ban nincs támogatva a többszörös öröklődés viszont interface-k segítenek megvalósítani azokat.



11.3. ábra. Többszörös öröklődés

11.2. Python könyv feldolgozása

A Python egy magasszintű programozási szkriptnyelv. 1990-ben Guido van Rossum hozta létre és mára szinte a programozási nyelvek listáján egyeduralkodóként van jelen. Fejlesztők számára számos tulajdonsággal rendelkezik. Magas szintű, dinamikus, objektumorientált és platformfüggetlen, ami nagyban hozzájárul a különböző szintű és összetettségű programok hatékony létrehozásához. Rengeteg csomagot tartalmaz amely segít a komolyabb problémák (pl. MI, gépitanulás, statisztika stb.) megoldásához. A programokhoz nem szükséges fordítási fázis, elegendő a forrást az értelmezőnek átdobni ami automatikusan futtatja az alkalmazást.

A nyelv sajátossága, hogy nem blokkokban gondolkodik, hanem behuzásalapú szintaxison. Az azonos behuzással rendelkező kódok egy csoportba tartoznak így alá/felé rendelést létrehozva. Továbbá nincs end-jel karakter (;) ami jelzi az utasítás végét. minden egyes sort úgynevezett tokenekre bont, amelyek között whitespace karakterek lehetnek. Lehetnek kulcsszavak, literálok stb. és ezeket az értelmező értelmezi.

```
if x > 5 :  
    print('hello world!')  
    print('not hello world!')
```

A Python nyelvbe nincsenek megadott típusok, minden adatot objektumok reprezentálnak. Értékek típusa futási időbe fogalmazódik meg. Ezen típusok lehetnek számok, karakterláncok, ennesek, listák, szótárak. Példa stringre:

```
print u"Yesterday, Pete went %s" ("afk")
```

A típusoknál emlétet ennesek nem mások, mint tetszőleges objektumok gyűjteménye. Lényegében azonos vagy különböző értékeket tudunk tárolni benne. Zárójelek között megadjuk a tagokat:

```
(5, "kettő", true)
```

Listákat szögletes zárójelek [...] között adjuk meg. Ezek ugyanúgy működnek, mint a többi nyelvben. Hozzáfűzhetünk új elemeket is hozzá így dinamikusan növelve a méretét. Az elemeket indexükkel azonosítjuk. Példa a listákra:

```
[3, 4, 5]  
[5, "kettő", true]  
list(5, 4)
```

A könyvtárak (dictionary) kulcs érték párokat tárol. Lényegében egy kiterjesztett listáról beszélünk. Hasonlóan képzelhetjük el, mint PHP-ben az asszociatív tömböket. Kapcsos zárójelek között kettősponttal elválasztva adjuk meg az egyes elemeket.

```
{'a':1, 'b':9}
```

Pythonban léteznek globális és lokális változók. Ezeket egyenlőségjel segítségével inicializálhatjuk. Érték kül bármit adhatunk pl.: objektumokat, típusokat, függvényeket és így tovább.

A nyelv támogatja a már jól megszokott vezérlési szerkezeteket. Ilyen első alap szerkezet az elágazás vagyis az if. Ugyanúgy működik, mint más programozási nyelvekben csak a nyelvi tagolásra kell figyelni. Ez az alábbi módon néz ki:

```
if c > 5:  
    print("asd")  
elif c < 5:  
    print("-asd")  
else:  
    print("5")
```

If kulcsszó után megadjuk a feltételt. Ezután egy kettőspont választja el a "törzset" majd a sorbehúzás szerint végrehajtódik egy utasítás, ha a feltétel teljesül.

Ciklusok közül a for ciklust és a while ciklust említeném meg. A forral végigiterálhatunk egy listán vagy akár egy kulcs érték párokat használó könyvtáron is. for kulcsszó után egy változót adunk meg, amelyben

az épp aktuális elemet tárolja ideiglenesen. Ezután egy in kulcsszó következik majd a lista. A while ciklus hasonlóan működik, mint a többi nyelvben. While kulcsszó után egy feltétel következik. Ha a feltétel teljesük, akkor az alatta elhelyezkedő utasítás hajtódik végre.

```
characters = ["Fizz", "Thresh", "Pyke"]
for x in characters
    print x

for key, value in dic.items():
    print key, '-', value

y = "cat"
while (y == "cat"):
    print "meow:3"
```

A könyv röviden kitér a Python-ban megtalálható objektumorientált eszközökre is. Osztályt az alábbi módon tudjuk definiálni:

```
class Bill():
    def Hello(self, say):
        print "Hello", say
```

Az osztályt a class kucslszóval tudjuk megadni. Utána következik az osztály neve illetve felsorolásképp az ősosztályok listája. Ezután behuzás szinten definiálhatjuk az utasításokat, konstruktorokat stb.

12. fejezet

Helló, Arroway!

12.1. Az objektumorientált paradigma alapfoglalmai. Osztály, objektum, példányosítás.

Az objektumorientált programozás az objektumok osztályozásainak, kapcsolatainak és tulajdonságainak felhasználásával segíti a programfejlesztést. Objektum szinte bármilyen fogalom vagy elem lehet. Lehet egy fizikai objektum (egy fejlesztendő cég terméklistája) vagy épp egy kódon, programon belüli rész. Az attribútumok az objektummal kapcsolatban álló tulajdonságok vagy változók. A műveletek az objektum olyan metódusai, eljárásai vagy függvényei amelyek segítségével módosíthatunk meglévő objektumokon vagy új funkciókat képezzük a segítségükkel. Az objektumokon belüli adatokhoz kizárolag csak az objektumokon belül illetve metódusokon (interface) keresztül férhetünk hozzá. Továbbá az objektum nem léheti át saját hatáskörét magyarul minden objektumnak megvan a jól meghatározott feladatköre. Az objektumok osztályokba csoportosíthatók. minden osztálynak megvan a maga tulajdonságai és műveletei. Új objektumot példányosításon keresztül hozhatunk létre. Ekkor egy osztály új objektumpéldánya jön létre.

12.2. OO Szemlélet

Ebben a részben a polártranszformációs normális generátort kellett megírni Java nyelven. Lényege, hogy egy polármódszerrel előállított számot ad vissza. Ennek a matematika háttere nem érdekes, viszont a Java program része már annál inkább. A program objektumorientált szemléletmóddal készült vagyis van benne osztály, attribútumok és metódusok. Lényege, hogy a PolarGen osztály boolean nincsTarolt attribútuma határozza meg a kovetkezo() függvény double visszatérési értékét. Ha nincs tárolt értékünk akkor két számot állítunk elő. Az egyiket eltároljuk egy változóban ebből következik, hogy a nincsTarol értéke módusul majd a másikkal visszatérünk. Ha viszont van tárolt érték akkor egyszerűen azzal tér vissza a kovetkezo() függvény. A kód:

```
import static java.lang.System.*;  
  
public class PolarGen  
{
```

```
boolean nincsTarolt = true;
double tarolt;

public PolarGen()
{
    nincsTarolt = true;
}

public double kovetkezo()
{
    if(nincsTarolt)
    {
        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);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
        return r * v1;
    }
    else
    {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
}

public static void main(String[] args)
{
    PolarGen pg = new PolarGen();
    for (int i = 0; i < 10; i++)
    {
        out.println(pg.kovetkezo());
    }
}
```

```
/snap/intellij-idea-community/172/jbr/bin/java -java  
-1.082218642676967  
1.375591816758593  
0.5780022253223966  
-0.5736211327291608  
1.6905763165671035  
0.34815541141002515  
-0.6708691365892944  
-1.4722912280495872  
-0.8939030735183057  
-0.3848460646072055
```

```
Process finished with exit code 0
```

12.1. ábra. PolarGen program eredménye

Hasonló megoldás található az OpenJDK-ban a Random.class-ban. Különbség a synchronized kulcs-szó. Lényege, hogy egy időben csak 1 szálon futhat a programkód ezzel megakadályozva a különöböző szálakon történő szimultán módosításokat.

```
    public synchronized double nextGaussian() {
        if (this.haveNextNextGaussian) {
            this.haveNextNextGaussian = false;
            return this.nextNextGaussian;
        } else {
            double v1;
            double v2;
            double s;
            do {
                do {
                    v1 = 2.0D * this.nextDouble() - 1.0D;
                    v2 = 2.0D * this.nextDouble() - 1.0D;
                    s = v1 * v1 + v2 * v2;
                } while(s >= 1.0D);
            } while(s == 0.0D);

            double multiplier = StrictMath.sqrt(v: -2.0D * StrictMath.log(s) / s);
            this.nextNextGaussian = v2 * multiplier;
            this.haveNextNextGaussian = true;
            return v1 * multiplier;
        }
    }
```

12.2. ábra. Random.class-ban lévő nextGaussian() függvény

12.3. Homokozó

Ebben a feladatban a C++ féle LZWBinfa-t kellett átírni Java nyelvre. A legtöbb függvényt könnyű volt átírni csak a pointereket és referenciakat kellett kiiírtani a kódból. Lényegi rész volt, hogy a működése ne változzon. A működést nem részletezném, mert már prog1-ről is rengetegszer feljött úgyhogy mindenkinak a "könyökén" jön ki. Java forráskód a Binfáról:

```
package lzwfa;

import java.io.FileInputStream;

public class Binfa
{
    public Binfa()
    {
        fa = gyoker;
    }

    public void egyBitFeldolg(char b)
```

```
{  
    if (b == '0')  
    {  
  
        if (fa.egyesGyermek() == null)  
        {  
            Csomopont uj = new Csomopont('0');  
            fa.ujNullasGyermek(uj);  
            fa = gyoker;  
        }  
        else  
        {  
            fa = fa.nullasGyermek();  
        }  
    }  
    else  
    {  
        if (fa.egyesGyermek() == null)  
        {  
            Csomopont uj = new Csomopont('1');  
            fa.ujEgyesGyermek(uj);  
            fa = gyoker;  
        }  
        else  
        {  
            fa = fa.egyesGyermek();  
        }  
    }  
}  
  
public void kiir()  
{  
    melyseg = 0;  
    kiir(gyoker, new java.io.PrintWriter(System.out));  
}  
  
public void kiir(java.io.PrintWriter os)  
{  
    melyseg = 0;  
    kiir(gyoker, os);  
}  
  
class Csomopont  
{  
    public Csomopont(char betu)  
    {  
        this.betu = betu;  
        balNulla = null;  
        jobbEgy = null;  
    };
```

```
public Csomopont nullasGyermek()
{
    return balNulla;
}

public Csomopont egyesGyermek()
{
    return jobbEgy;
}

public void ujNullasGyermek(Csomopont gy)
{
    balNulla = gy;
}

public void ujEgyesGyermek(Csomopont gy)
{
    jobbEgy = gy;
}

public char getBetu()
{
    return betu;
}
private char betu;

private Csomopont balNulla = null;
private Csomopont jobbEgy = null;

};

private Csomopont fa = null;

private int melyseg, atlagosszeg, atlagdb;
private double szorasosszeg;

public void kiir(Csomopont elem, java.io.PrintWriter os)
{
    if(elem != null)
    {
        ++melyseg;
        kiir(elem.egyesGyermek(), os);
        for (int i = 0; i < melyseg; i++)
        {
            os.print("----");
        }
        os.print(elem.getBetu());
        os.print("(");
        os.print(melyseg - 1);
    }
}
```

```
        os.println(")\"");
        kiir(elem.nullasGyermek(), os);
        --melyseg;
    }
}

protected Csomopont gyoker = new Csomopont('/');
int maxMelyseg;
double atlag, szoras;

public int getMelyseg()
{
    melyseg = maxMelyseg = 0;
    rmelyseg(gyoker);
    return maxMelyseg - 1;
}

public void rmelyseg(Csomopont elem)
{
    if(elem != null)
    {
        ++melyseg;
        if(melyseg > maxMelyseg)
        {
            maxMelyseg = melyseg;
        }
        rmelyseg(elem.egyesGyermek());
        rmelyseg(elem.nullasGyermek());
        --melyseg;
    }
}

public double getAtlag()
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag(gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}

public double getSzoras()
{
    atlag = getAtlag();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras(gyoker);

    if(atlagdb - 1 > 0)
    {
```

```
        szoras = Math.sqrt(szorasosszeg / (atlagdb - 1));
    }
    else
    {
        szoras = Math.sqrt(szorasosszeg);
    }

    return szoras;
}

public void ratlag(Csomopont elem)
{
    if(elem != null)
    {
        ++melyseg;
        ratlag(elem.egyesGyermek());
        ratlag(elem.nullasGyermek());
        --melyseg;
        if(elem.egyesGyermek() == null && elem.nullasGyermek() == null)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

public void 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));
        }
    }
}

public static void usage()
{
    System.out.println("Usage: lzwtree in_file -o out_file");
}

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

```
if(args.length != 3)
{
    usage();
    System.exit(-1);
}

String inFile = args[0];

if(!"-o".equals(args[1]))
{
    usage();
    System.exit(-1);
}

try
{
    java.io.FileInputStream beFile = new java.io.FileInputStream(new java.io.File(args[0]));

    java.io.PrintWriter kiFile = new java.io.PrintWriter( new java.io.BufferedWriter( new java.io.FileWriter(args[2])));

    byte[] b = new byte[1];

    Binfa binFa = new Binfa();

    while(beFile.read(b) != -1)
    {
        if(b[0] == 0x0a)
        {
            break;
        }
    }
    boolean kommentben = false;

    while(beFile.read(b) != -1)
    {
        if(b[0] == 0x3e)
        {
            kommentben = true;
            continue;
        }

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

```
if(kommentben)
{
    continue;
}

if(b[0] == 0x4e)
{
    continue;
}

for (int i = 0; i < 8; ++i)
{
    if((b[0] & 0x80) != 0)
    {
        binFa.egyBitFeldolg('1');
    }
    else
    {
        binFa.egyBitFeldolg('0');
    }
    b[0] <<= 1;
}

binFa.kiir(kiFile);

kiFile.println("depth = " + binFa.getMelyseg());
kiFile.println("mean = " + binFa.getAtlag());
kiFile.println("var = " + binFa.getSzoras());

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

} catch (Exception e) {
    System.err.print(e.getMessage());
}
}
```

A feladat második része az volt, hogy ezt a Binfa-t emeljük át egy Java Servletbe és irassuk ki a fát egy böngészőbe. Ahhoz, hogy működőképes legyen a Servlet szükség volt egy Tomcat Serverre. Letöltés után integrálni kellett a JRE-hez, amit az Eclipse telepítési könyvtárában találtam. A megfelelő konfiguráció után a 8085-ös porton elértem a Tomcat szervert. Következő lépésként indítottam egy Dinamikus Web Projektet, amit belül csináltam egy `LZWBinfaServlet.java` servlet file-t. Ezen fájl automatikusan legenerálta a szükséges metódusokat, konstruktorkat a megfelelő működéshez. A classom a `HttpServlet` osztályból lett származtatva, illetve kaptunk egy `doGet(...)` eljárást. Ez az eljárás felel a HTTP GET

kérésékről, ami nekünk pont kapóra költött.

Az LZWBinfá forráskódjából minden átírtunk a main függvény kivételével. A main kódjait a doGet(...) eljárásban belül írjuk meg. A kész doGet(...) így néz ki:

```
/*
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
{
    String parameter = request.getParameter("param");

    byte[] b = parameter.getBytes();

    Binfa binFa = new Binfa();

    java.io.PrintWriter kiFile = new java.io.PrintWriter( new java.io.
        BufferedWriter( new java.io.FileWriter("output.txt") ));

    boolean kommentben = false;

    for(int h = 0; h < b.length; ++h)
    {
        if(b[h] == 0x3e)
        {
            kommentben = true;
            continue;
        }

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

        if(kommentben)
        {
            continue;
        }

        if(b[h] == 0x4e)
        {
            continue;
        }

        for (int i = 0; i < 8; i++)
        {
            if((b[h] & 0x80) != 0)
```

```
{  
    binFa.egyBitFeldolg('1');  
}  
else  
{  
    binFa.egyBitFeldolg('0');  
}  
b[h] <= 1;  
}  
}  
  
binFa.kiir(kiFile);  
  
kiFile.println("depth = " + binFa.getMelyseg());  
kiFile.println("mean = " + binFa.getAtlag());  
kiFile.println("var = " + binFa.getSzoras());  
  
kiFile.close();  
  
File file = new File("output.txt");  
FileInputStream fis = new FileInputStream(file);  
  
// TODO Auto-generated method stub  
response.setContentType("text/html");  
  
PrintWriter out = response.getWriter();  
out.println("<!DOCTYPE html>");  
out.println("<html>");  
  
out.println("<head><title>LZWBinfa</title></head>");  
out.println("<body>");  
out.println("<h2>Dékány Róbert LZWBinfa Servlet</h2>");  
try(BufferedReader br = new BufferedReader(new InputStreamReader(fis))) {  
    for(String line; (line = br.readLine()) != null; ) { out.println(←  
    "<p>" + line + "</p>"); } }  
  
out.println("</body></html>");  
}
```

A String parameter = request.getParameter("param"); változóban lesz eltárolva az URI-ból vett param nevű paraméter értéke. Ezt a karakterláncot fogjuk továbbadni a Binfának. A PrintWriter out = response.getWriter(); egy olyan objektummal tér vissza, ami képes kiírni az outputra jelen esetbe HTML kódokat. A továbbiakban csak kiiíratjuk a Stream tartalmát és elhelyezzük a szükséges HTML-t.



12.3. ábra. LZWBinffa megjelenítése böngészőben.

12.4. „Gagyi”

Feladatunk az alábbi feltételre épül:

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

Ha külön az x,t értékének -128-at majd -129-et adunk mit tapasztalunk? A -128 nem teljesük a feltétel nem teljesük és kilép a ciklusból még a -129-nél végtelen ciklusba kerül.

```
public class Gagyi2
{
    public static void main (String[] args)
    {
        Integer x = -129;
        Integer t = -129;

        System.out.println (x);
        System.out.println (t);

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

public class Gagyi3
{
    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);
    }
}
```

Röviden a -128 esetén a x és t objektumok megegyeznek még a -129 esetén két különöböző objektum. A bővebb magyarázatot a JDK Integer.java forrására adja meg.

```
    } private static class IntegerCache {  
        static final int low = -128;  
        static final int high;  
        static final Integer[] cache;  
  
        private IntegerCache() {  
        }  
  
        static {  
            int h = 127;  
            String integerCacheHighPropValue = VM.getSavedProperty( key: "java.lang.Integer.IntegerCache.high");  
            int i;  
            if (integerCacheHighPropValue != null) {  
                try {  
                    i = Integer.parseInt(integerCacheHighPropValue);  
                    i = Math.max(i, 127);  
                    h = Math.min(i, 2147483518);  
                } catch (NumberFormatException var4) {  
                }  
            }  
  
            high = h;  
            cache = new Integer[high - -128 + 1];  
            i = -128;  
  
            for(int k = 0; k < cache.length; ++k) {  
                cache[k] = new Integer(i++);  
            }  
  
            assert high >= 127;  
        }  
    }  
}
```

12.4. ábra. JDK Integer.java

```
@HotSpotIntrinsicCandidate  
public static Integer valueOf(int i) {  
    return i >= -128 && i <= Integer.IntegerCache.high ? Integer. ←  
        IntegerCache.cache[i + 128] : new Integer(i);  
}
```

A public static Integer valueOf(int i) függvény egy Integer objektummal tér vissza. Ha a paraméter i értéke kisebb vagy egyenlő, mint a IntegerCache.high értéke akkor egy IntegerCache poolból adja vissza az objektumot. Viszont, ha nagyobb akkor egy új Integer objektumot ad vissza, ami már új memóriacímmel fog rendelezni.

12.5. Yoda

Ebben a feladatban egy Java programot kellett írni, amely NullPointerException-el leáll, ha nem követjük a Yoda conditiont. A Yoda Condition lényege, hogy a feltétel sorrendje megfordul. Baloldalon a konstans fogal helyet.

```
public class Main {  
  
    public static void main(String[] args)  
    {  
        String name = null;  
        if (name.equals("John")) { /* ... */ }  
    }  
}
```

A felti kód NullPointerException kivételt ad, mert name változó egy nullpointer és hasonlíthatni akarjuk a "John" konstanst, ami nem megengedett.

```
/snap/intelliJ-idea-community/172/jbr/bin/java -javaagent:/snap/int  
Exception in thread "main" java.lang.NullPointerException  
at com.company.Main.main(Main.java:8)  
  
Process finished with exit code 1
```

12.5. ábra. NullPointerException, ha nem használjuk a Yoda Conditiont.

Ha megfordítjuk a feltétel sorrendjét és használjuk a Yoda conditiont akkor már jó. String literalhoz már hasonlíthatunk null értéket, ami jelen esetben false értéket adna. Ezzel a módszerrel különféle hibákat tudunk kiküszöbölni, úgy mint a értékadás-hasonlítás vagy a null értékekből eredő hibák.

```
public class Main  
{  
  
    public static void main(String[] args)  
    {  
        String name = null;  
        if ("John".equals(name)) { /* ... */ }  
    }  
}
```

12.6. Kódolás from scratch

Feladat volt a BBP (Bailey-Borwein-Plouffe) algoritmust a Pi hexa jegyeinek számolását végző osztály megírása egy tudományos közlemény alapján. A pdf-ben sok matematika képlet illetve leírás szerepelt. Ezek megértése időigényes. A közleményt illetve a megadott példákat felhasználva sikerült megírni az algoritmust:

```
package com.company;

public class PiBBP {

    String d16PiHexaJegyek;

    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'};

        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();
    }

    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);
    }

    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;
    }

    public String toString() {

        return d16PiHexaJegyek;
    }

    public static void main(String args[]) {
        System.out.print(new PiBBP(1000000));
    }
}
```

```
/snap/intellij-idea-community/172/jbr/bin/java -j  
6C65E5308  
Process finished with exit code 0
```

12.6. ábra. BBP algoritmus output.

13. fejezet

Helló, Liskov!

13.1. Öröklődés, osztályhierarchia. Polimorfizmus, metódustúlterhélések. Hatáskörkezelés. A bezárási eszközrendszer, láthatósági szintek. Absztrakt osztályok és interfések.

Az öröklődés (inheritance) lehetővé teszi, hogy alosztályok (subclass) használatával hierarchikus kapcsolatot hozunk létre osztályok között. A gyermekosztály örökli a bázisosztály tulajdonságait. Öröklődéssel építhetünk a tulajdonságaira és új tulajdonságokkal ruházhatjuk fel. A többalakúság (polymorphism) alatt azt értjük, hogy eltérő osztályok eltérő viselkedéssel rendelkezhetnek ugyanazon műveletre. Gondoljuk egy bicikli osztályra azon belül egy tisztít metódusra. Az emberi nyelvben a tisztít, mint fogalom általános megfogalmazás ezért nem csak a biciklire tudjuk alkalmazni. Hatáskörkezelés egyik fogalma az egységbázárás (encapsulation). Ez annyit takar, hogy minden osztálynak megvannak a tulajdonságai és metódusai, amik egybe tartoznak ezért célszerű egy egységebe zární őket. (pl.: osztályok). Másik ilyen fogalom az adatelrejtés vagy data hiding. Ezen elv alapján korlátozást szabhatjuk az osztály egyes adatelemeinek elérésére. Ez azért nagyon fontos, hogy a kritikus tulajdonságokat (pl.: banki program, egyenleg) ne lehessen módosítani. Ha módosítani szeretnénk azt az getter, setter metódusokon keresztül tegyük meg. (ha van rá lehetőség) A private, protected, public láthatósági módosítók segítenek az adatelrejtésben illetve az eléréssek szabályozásában. A törzs nélküli metódusokat definiáló osztályokat absztrakt osztályoknak nevezzük. Lényege, hogy az alosztályok interface-nek közös részét adja, konkrét implementációt nem. Ez többnyire az osztályhierarchia tetején álló osztályokból tevődik össze. Az absztrakt osztályok nem példányosíthatóak. Az interfacek olyan referencia típusok, amelyekben csak deklarációk szerepelnek. Ez segíti az absztraktiós szint bevezetését.

13.2. Liskov helyettesítés sértése

Ebben a feladatban a Liskov-féle helyettesítési elv megsértésére kellett egy C++ illetve Java példát írni. Ezen elv a S.O.L.I.D alapeltek egyike, amely elősegíti a tiszta kód készítését. A Liskov-féle helyettesítés arról szól, hogy minden osztály legyen helyettesíthető a leszármazott osztályával anélkül, hogy a program helyes működését befolyásolná. Ha S leszármazottja T-nek, akkor behelyettesíthetjük minden gond nélkül T helyére S-t. Ezen elvet megsértő C++ kód:

```
#include <iostream>

class Bank
{
private:
    unsigned int money;

public:
    virtual void setMoney(int m)
    {
        money = m;
    }

    virtual int getMoney()
    {
        return money;
    }
};

class ThiefBank : public Bank
{
    int getMoney()
    {
        return 0;
    }
};

int main()
{
    Bank *bank = new ThiefBank();

    bank->setMoney(555);

    std::cout << bank->getMoney() << std::endl;

    return 0;
}
```

Mint látható a ThiefBank osztályban a getMoney() metódus megváltozik. Ezáltal nem teljesül az a feltétel, hogy ugyanazon működés kell, hogy legyen. Ha viszont a ThiefBank osztályban a getMoney()

függvényt átírjuk getMonkey()-ra, akkor már a helyes működést tapasztalnánk. :D

```
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ g++ liskov.cpp -o l
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ ./l
0
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ █
```

13.1. ábra. ThiefBank helytelenül működik.

Hasonló példakód a Liskolv-elv megsértésére Java-ban:

```
class LiskovLoveYou
{
    private String welcome;

    public void setWelcome(String welcome)
    {
        this.welcome = welcome;
    }

    public String getWelcome()
    {
        return this.welcome;
    }
}

class LiskovHateYou extends LiskovLoveYou
{
    public String getWelcome()
    {
        return "I hate you so much!";
    }
}

public class Liskov
{
    public static void main(String[] args)
    {
        LiskovLoveYou l = new LiskovHateYou();

        System.out.println("Liskov: " + l.getWelcome());
    }
}
```

Ezen kódcsipetek segítenek rájönni, hogy mennyire sokszínű hibákat követhetünk el kódolás közben. Érdemes a különböző kódolási alapelveket követve kódolni, mert sok fejfájástól megkímélhetjük magunkat

és másokat.

13.3. Szülő-gyerek

Ebben a feladatban a bázisosztály-gyerekosztály kapcsolatra kellett példát készíteni. Igazolni kell, hogy a származtatás ellenére lesznek olyan metódusok, amelyeket az ősosztály nem láthat vagyis "ősön keresztül csak az ős üzenetei küldhetőek". Kód:

```
#include <iostream>

class Bank
{
private:
    unsigned int money;

public:
    virtual void setMoney(int m)
    {
        money = m;
    }

    virtual int getMoney()
    {
        return money;
    }
};

class ThiefBank : public Bank
{
    int getMonkey()
    {
        return 0;
    }
};

int main()
{
    Bank *bank = new ThiefBank();

    std::cout << bank->getMonkey() << std::endl;

    return 0;
}
```

```
}
```

A példában ThiefBank gyermekosztályban egy új getMonkey() metódust van definiálva. A Bank *bank = new ThiefBank(); értékkadásnál a Bank a statikus típus míg a ThiefBank a dinamikus típus. A dinamikus típus futás időben, a statikus típus pedig fordítási időben definiált. A probléma ott keződik, hogy a Monkey() metódus az ősosztályban nem definiált így már fordítás közben megakad. A program az error: class Bank' has no member named getMonkey'-el elszáll. Az előbbi probléma Java kódja:

```
class LiskovLoveYou
{
    private String welcome;

    public void setWelcome(String welcome)
    {
        this.welcome = welcome;
    }

    public String getWelcome()
    {
        return this.welcome;
    }
}

class LiskovHateYou extends LiskovLoveYou
{
    public String getColdWelcome()
    {
        return "I hate you so much!";
    }
}

public class Liskov
{
    public static void main(String[] args)
    {
        LiskovLoveYou l = new LiskovHateYou();

        System.out.println("Liskov: " + l.getColdWelcome());
    }
}
```

Az IntelliJ már a kód beírásánál sípol, hibát jelez, ahogy az várható volt.

13.4. Anti OO

A feladat az előző csokorból megismert PiBBP algoritmusra épít. 0. pozíciótól számított 10^6 , 10^7 , 10^8 darab jegyet kellett meghatározni C, C++, Java és C# nyelveken és összevetni a futási időket! Kissé időigényes volt átírni az adott nyelvre, de végül sikerült a futási időket is lemérni. Itt a Java kód:

```
public class PiBBP {  
  
    String d16PiHexaJegyek;  
  
    long startTimer = System.currentTimeMillis();  
  
    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'};  
  
        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);  
        }  
        long endTimer = System.currentTimeMillis();  
  
        float time = (endTimer - startTimer) / 1000F;  
  
        System.out.println(time + " ms");  
  
        d16PiHexaJegyek = sb.toString();  
    }  
}
```

```
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);  
}  
  
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;  
}  
  
public String toString() {  
  
    return d16PiHexaJegyek;  
}  
  
public static void main(String args[]) {  
    System.out.println("10^6: ");  
    System.out.print("hexa:" + new PiBBP(1000000));  
}  
}
```

```
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ javac PiBBP.java
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ java PiBBP
10^6:
1.554 ms
hexa:6C65E5308
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ javac PiBBP.java
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ java PiBBP
10^7:
17.98 ms
hexa:7AF58A34
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ javac PiBBP.java
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ java PiBBP
10^8:
206.496 ms
hexa:CB7CC4
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ □
```

13.2. ábra. Java-féle futási idők (sec)

C kód:

```
#include <stdio.h>
#include <math.h>
#include <time.h>
/*
 * pi_bbp_bench.c
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 * A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
 * a PiBBPBench osztályt, amit pedig átírtuk C nyelvre.
 *
 */
/*
 * 16^n mod k
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
long
n16modk (int n, int k)
{
    long r = 1;

    int t = 1;
```

```
while (t <= n)
    t *= 2;

for (;;)
{
    if (n >= t)
    {
        r = (16 * r) % k;
        n = n - t;
    }

    t = t / 2;

    if (t < 1)
        break;

    r = (r * r) % k;
}

return r;
}

/* {16^d Sj}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
double
d16Sj (int d, int j)
{
    double d16Sj = 0.0;
    int k;

    for (k = 0; k <= d; ++k)
        d16Sj += (double) n16modk (d - k, 8 * k + j) / (double) (8 * k + j);

    /*
        for(k=d+1; k<=2*d; ++k)
            d16Sj += pow(16.0, d-k) / (double) (8*k + j);
    */

    return d16Sj - floor (d16Sj);
}

/*
 * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
```

```
/*
main ()
{
    double d16Pi = 0.0;

    double d16S1t = 0.0;
    double d16S4t = 0.0;
    double d16S5t = 0.0;
    double d16S6t = 0.0;

    int jegy;
    int d;

    clock_t delta = clock ();

    for (d = 100000000; d < 100000001; ++d)
    {
        d16Pi = 0.0;

        d16S1t = d16Sj (d, 1);
        d16S4t = d16Sj (d, 4);
        d16S5t = d16Sj (d, 5);
        d16S6t = d16Sj (d, 6);

        d16Pi = 4.0 * d16S1t - 2.0 * d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - floor (d16Pi);

        jegy = (int) floor (16.0 * d16Pi);

    }

    printf ("%d\n", jegy);
    delta = clock () - delta;
    printf ("%f\n", (double) delta / CLOCKS_PER_SEC);
}
```

```
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ gcc picpp.cpp -o pi -lm
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ ./pi
6
1.734532
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ gcc picpp.cpp -o pi -lm
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ ./pi
7
20.031083
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ gcc picpp.cpp -o pi -lm
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ ./pi
12
230.616417
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ █
```

13.3. ábra. C-féle futási idők (sec)

C# kód:

```
public class PiBBPBench {

    public static 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);

        /*
        for(int k=d+1; k<=2*d; ++k)
            d16Sj += System.Math.pow(16.0d, d-k) / (double)(8*k + j);
        */

        return d16Sj - System.Math.Floor(d16Sj);
    }

    public static 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;
}

public static void Main(System.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;

    System.DateTime kezd = System.DateTime.Now;

    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 - System.Math.Floor(d16Pi);

        jegy = (int)System.Math.Floor(16.0d*d16Pi);

    }

    System.Console.WriteLine(jegy);
    System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
    System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
}
}
```

```
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ mcs PiBBPBench.cs
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ mono PiBBPBench.exe
6
1,573109
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ mcs PiBBPBench.cs
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ mono PiBBPBench.exe
7
18,199072
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ mcs PiBBPBench.cs
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ mcs PiBBPBench.cs
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ mono PiBBPBench.exe
12
208,712231
(base) drob@drob-laptop:~/Dokumentumok/prog2/cs-examples$ □
```

13.4. ábra. C#-félé futási idők (sec)

C++ kód:

```
#include <iostream>
#include <math.h>
#include <time.h>
/*
 * pi_bbp_bench.c
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 * A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
 * a PiBBPBench osztályt, amit pedig átírtuk C nyelvre.
 *
 */
/*
 * 16^n mod k
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
long n16modk (int n, int k)
{
    long r = 1;

    int t = 1;
    while (t <= n)
        t *= 2;

    if (t > n)
        t /= 2;
```

```
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;
}
double d16Sj (int d, int j)
{
    double d16Sj = 0.0;

    for (int k = 0; k <= d; ++k)
        d16Sj += (double) (n16modk (d - k, 8 * k + j)) / (double) (8 * k + j);

    return d16Sj - floor (d16Sj);
}

int main()
{
    double d16Pi = 0.0;

    double d16S1t = 0.0;
    double d16S4t = 0.0;
    double d16S5t = 0.0;
    double d16S6t = 0.0;

    int jegy;

    clock_t delta = clock ();

    for (int d = 100000000; d < 100000001; ++d)
    {
        d16Pi = 0.0;
```

```
d16S1t = d16Sj (d, 1);
d16S4t = d16Sj (d, 4);
d16S5t = d16Sj (d, 5);
d16S6t = d16Sj (d, 6);

d16Pi = 4.0 * d16S1t - 2.0 * d16S4t - d16S5t - d16S6t;

d16Pi = d16Pi - floor (d16Pi);

jegy = (int) floor (16.0 * d16Pi);

}

std::cout << jegy << '\n';
delta = clock () - delta;
std::cout << static_cast<double>(delta)/CLOCKS_PER_SEC << '\n';
}
```

```
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ g++ picpp.cpp -o p
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ ./p
6
1.66025
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ g++ picpp.cpp -o p
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ ./p
7
19.4103
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ g++ picpp.cpp -o p
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ ./p
12
226.655
(base) drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples$ █
```

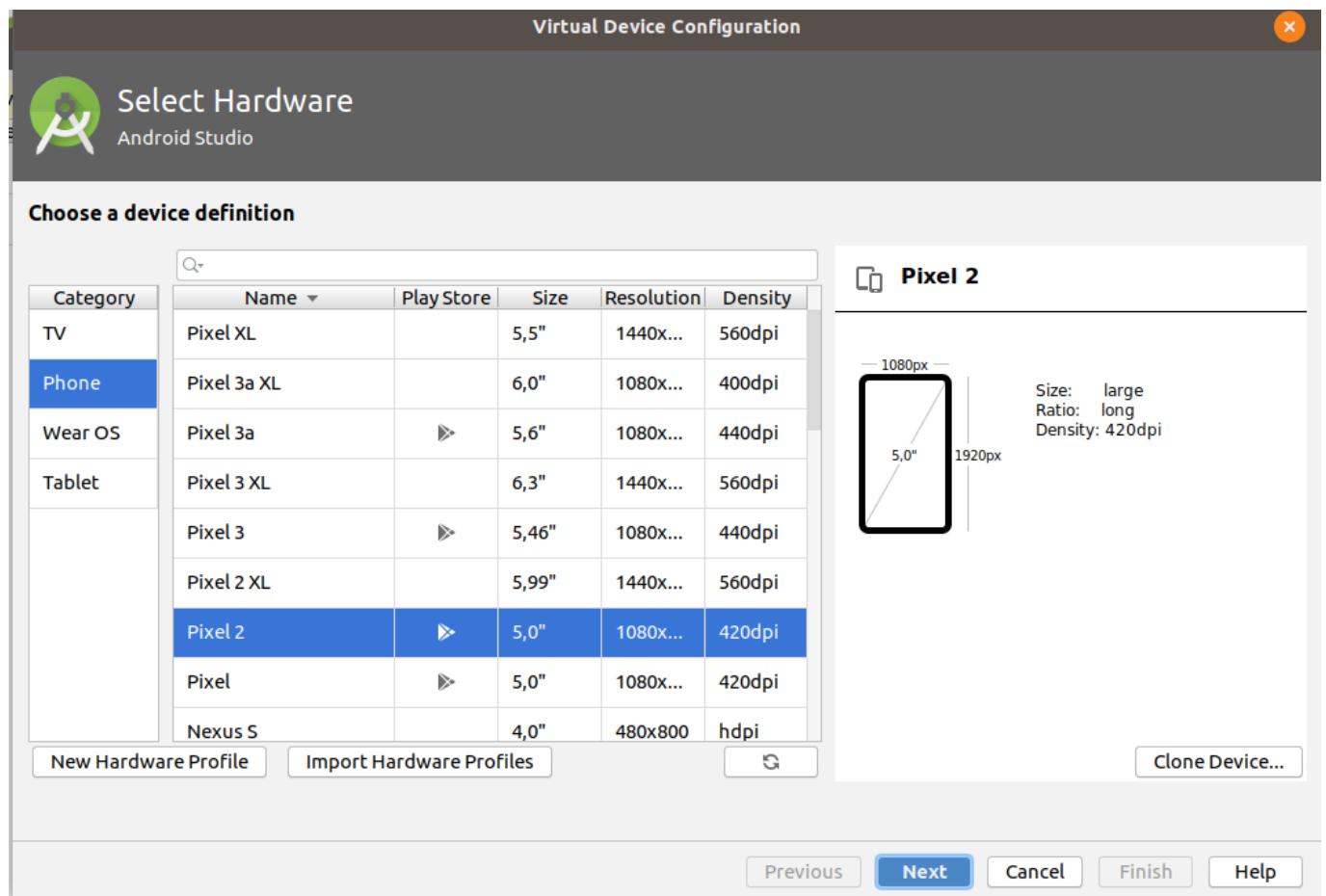
13.5. ábra. C++-félé futási idők (sec)

	C	C++	Java	C#
10^6	1.734	1.660	1.554	1.573
10^7	20.031	19.410	17.98	18.199
10^8	230.616	226.655	206.496	208.712

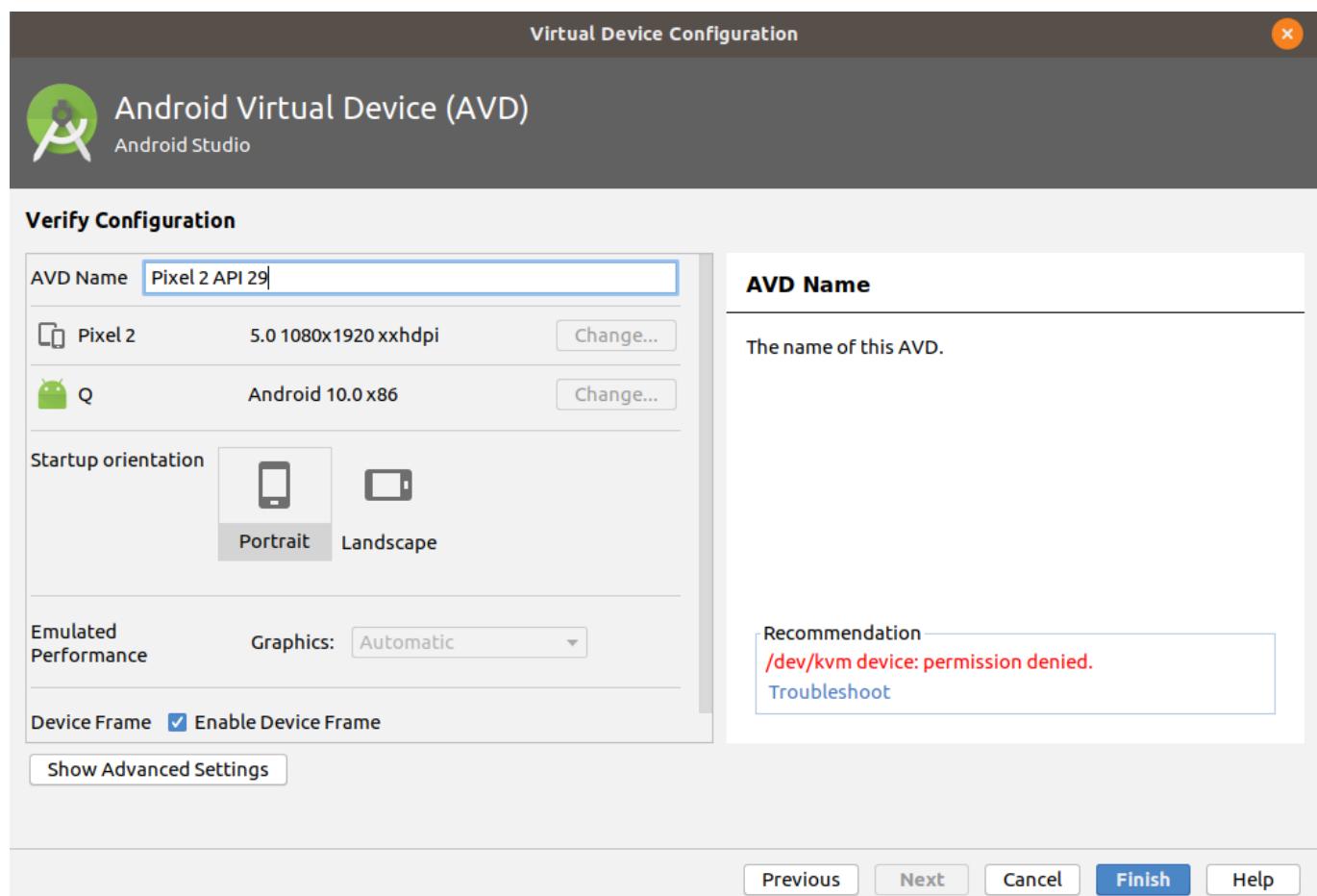
13.6. ábra. Összefoglaló táblázat a futási időkről

13.5. Hello, Android!

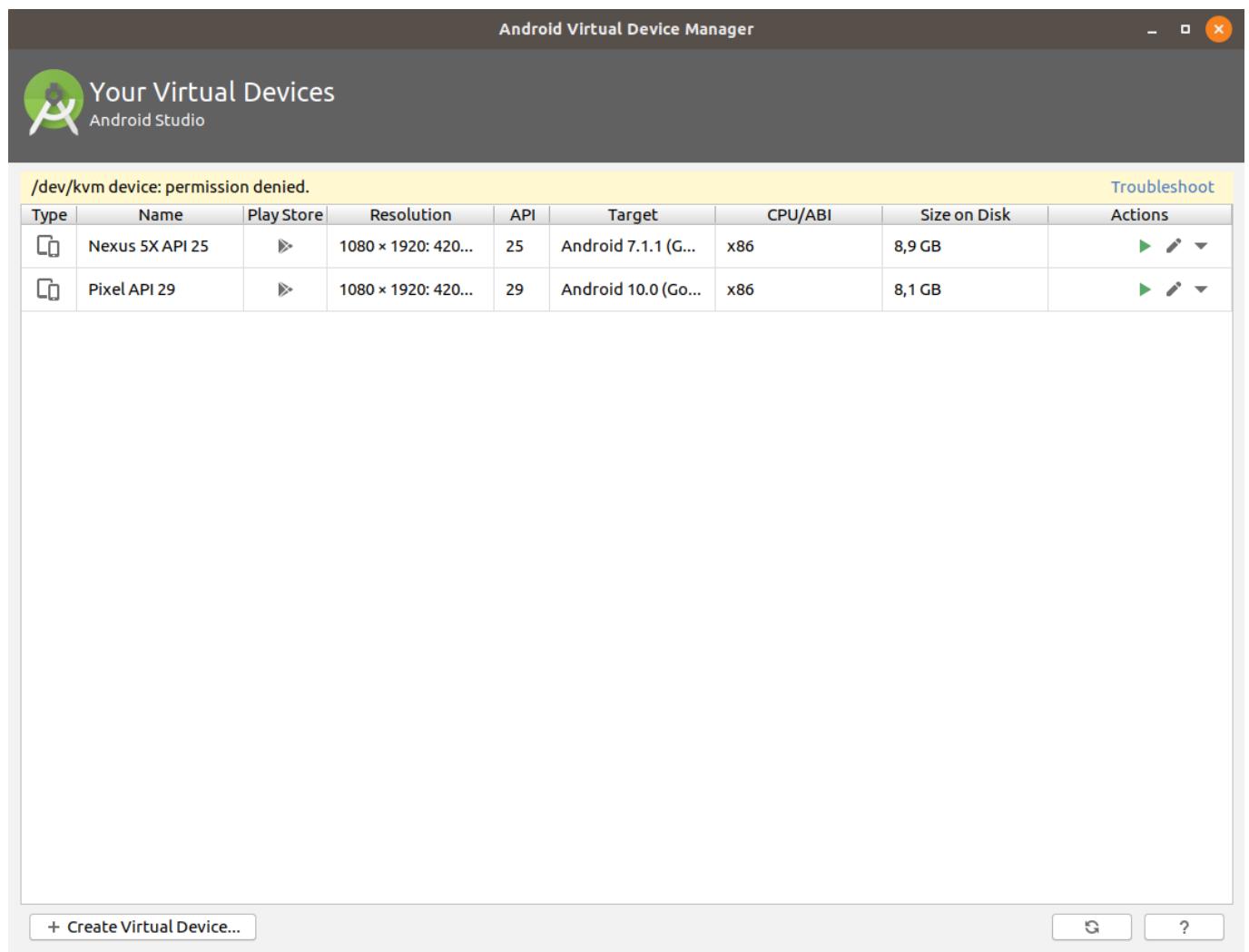
Az Android Studio telepítése után a Settings-be be kellett állítani az Android SDK-t. Ez pár kattintás és opció elfogadásával hamar települt. Következő lépés az Android Emulator konfigurálása volt. Ehhez le kellett tölteni egy emulátor virtualizációs csomagot: sudo apt install qemu-kvm majd hozzáadni a userünket a /etc/group a kvm/ csoporthoz. Ezután az Android Studion belül beállítottunk egy emulátor eszközt a Virtual Device Manager segítségével:



13.7. ábra. Emulátor hardver tulajdonságai



13.8. ábra. Emulátor egyéb tulajdonságai



13.9. ábra. Android Emulátor eszközök listája

Ha az összes lépésen végigmentünk akkor kész vagyunk futtatni Emulátoron keresztül a programunkat. Következő lépésként létrehoztunk egy új projektet, amjd átszabtuk az SMNIST formára a következőképpen. Elsőnek a res/layout/ mappába kellet bemásolni az activity_smniste3.xml file-t. Ez fogja beállítani a megfelelő elrendezést az alkalmazásunknak.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <hu.blog.bhaxor.smnistforhumansexp3.SMNISTSurfaceView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</RelativeLayout>
```

Ezután a app/manifests/ könyvtárba is be kellett másolni az `AndroidManifest.xml` file-t. Ebben a fileban az `activity android:name` attribútuma változott meg. Ezzel mondjuk meg, hogy melyik activity fussion le elsőként.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smnist">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SMNISTE3Activity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

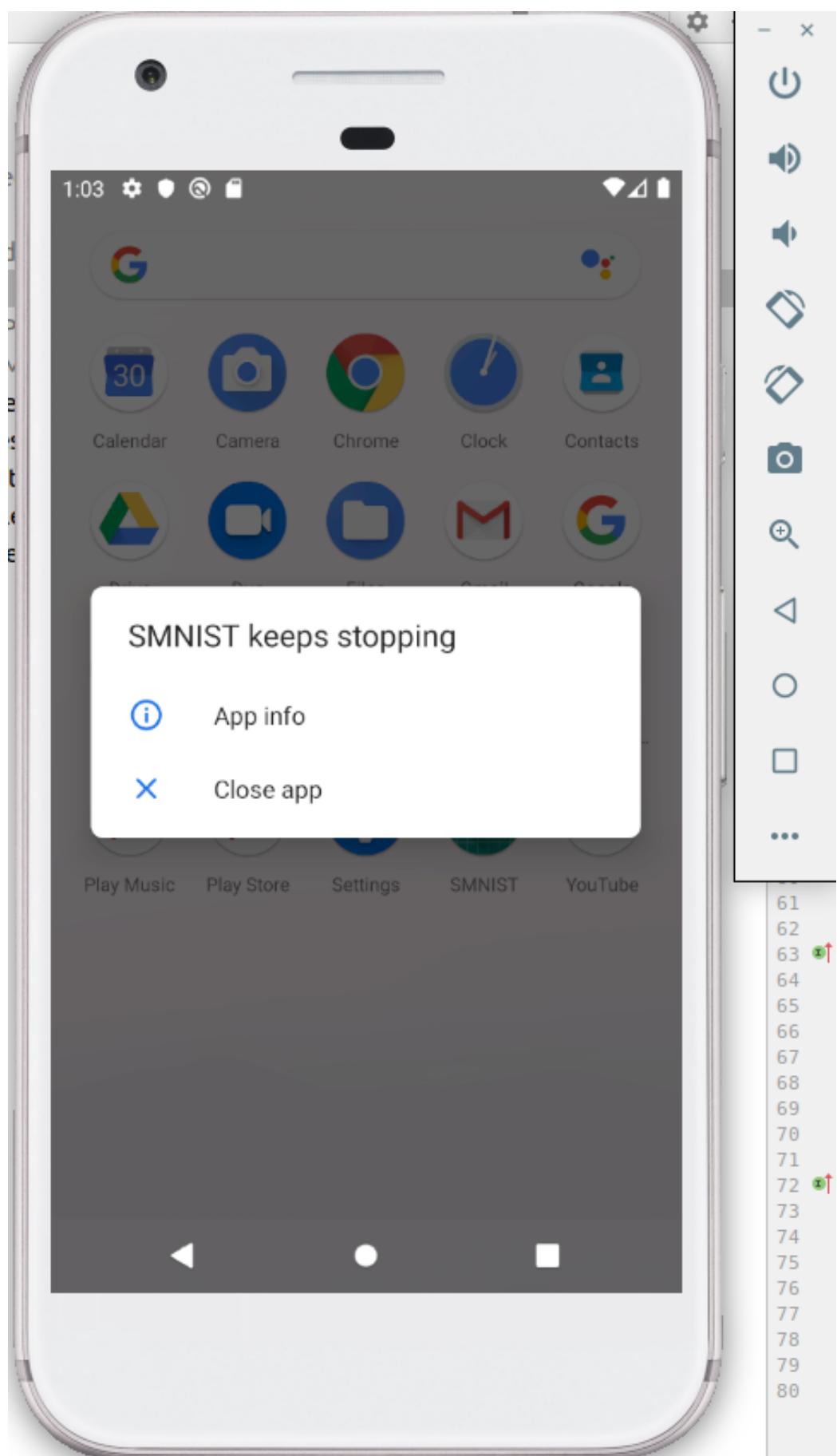
</manifest>
```

A továbbiakban már csak be kellett másolni az app/java/com.example.smnist/ mappába a négy java forrásfile-t. Itt még keletkezett hiba. Az `SMNISTE3Activity` osztály származtatva volt egy már elavult `AppCompatActivity`-ből így lecseréltem egy andoridx-es változatra.

```
import androidx.appcompat.app.AppCompatActivity;

public class SMNISTE3Activity extends AppCompatActivity{
```

Ezek után már lehetett futtatni egy Android Emulátorban. Sajnos az én esetben lefut viszont maga az alkalmazás kicrash-el. Ezt a jövőben javítani szeretném és teljesen bemutatni a helyes működéssel, illetve a módosításokkal.



13.10. ábra. SMNIST error

13.6. Ciklomatikus komplexitás

Ezen csokor utolsó feladataként egy általam választott program ciklomatikus komplexitását kellett kiszámolnom. A ciklomatikus komplexitás egy számérték, amelyet a program komplexitása (bonyolultsága) határoz meg. A komplexitás számítása gráfelméleten alapul.

A matek képlete a következő: $M = E - N + 2P$

A komplexitás lemérésére egy Lizard nevű open-source code complexity analyzer-t használlok. Github: <https://github.com/terryin/lizard>. Továbbiakban a PiBBP.java és a Binfa.java komplexitását mértem le. Az előbbi NLOC értéke 57, az utóbbi értéke pedig 250. Ebből is látszik, hogy a Binfában sokkal komplexebb vezérlési szerkezetek vannak implementálva. A kiadott értékekkel böngészve sok érdekes adatot szolgáltat a Lizard analyzer:

```
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ lizard PiBBP.java -x"./test/*"
=====
NLOC      CCN      token      PARAM      length      location
-----
 23        3       223        1        37 PiBBP::PiBBP@7-43@PiBBP.java
   6        2       70         2         9 PiBBP::d16Sj@45-53@PiBBP.java
 17        5       88         2        26 PiBBP::n16modk@55-80@PiBBP.java
   3        1        8         0         4 PiBBP::toString@82-85@PiBBP.java
   4        1       33         1         4 PiBBP::main@87-90@PiBBP.java
1 file analyzed.
=====
NLOC      Avg.NLOC      AvgCCN      Avg.token      function_cnt      file
-----
 57        10.6        2.4        84.4          5      PiBBP.java
=====
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or parameter_count > 100)
=====
Total nloc      Avg.NLOC      AvgCCN      Avg.token      Fun Cnt      Warning cnt      Fun Rt      nloc Rt
-----
 57        10.6        2.4        84.4          5            0        0.00        0.00
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ █
```

13.11. ábra. PiBBP.java ciklomatikus komplexitása

```
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ lizard Binfa.java -x"./test/*"
=====
 NLOC   CCN   token  PARAM  length  location
 -----
    4      1      9      0       4 Binfa::Binfa@5-8@Binfa.java
   29      4     104     1      30 Binfa::egyBitFeldolg@10-39@Binfa.java
    5      1     26      0       5 Binfa::kiir@41-45@Binfa.java
    5      1     22      1       5 Binfa::kiir@47-51@Binfa.java
    6      1     21      1       6 Binfa::Csomopont::Csomopont@55-60@Binfa.java
    4      1      8      0       4 Binfa::Csomopont::nullasGyermek@62-65@Binfa.java
    4      1      8      0       4 Binfa::Csomopont::egyesGyermek@67-70@Binfa.java
    4      1     11      1       4 Binfa::Csomopont::ujNullasGyermek@72-75@Binfa.java
    4      1     11      1       4 Binfa::Csomopont::ujEgyesGyermek@77-80@Binfa.java
    4      1      8      0       4 Binfa::Csomopont::getBetu@82-85@Binfa.java
  18      3     107      2      18 Binfa::kiir@98-115@Binfa.java
    6      1     21      0       6 Binfa::getMelyseg@121-126@Binfa.java
   14      3     51      1      14 Binfa::rmelyseg@128-141@Binfa.java
    7      1     32      0       7 Binfa::getAtlag@143-149@Binfa.java
   16      2     68      0      19 Binfa::getSzoras@151-169@Binfa.java
   15      4     66      1      15 Binfa::ratlag@171-185@Binfa.java
   15      4     78      1      15 Binfa::rszoras@187-201@Binfa.java
    4      1     14      0       4 Binfa::usage@203-206@Binfa.java
   70     13    386      1      88 Binfa::main@209-296@Binfa.java
1 file analyzed.
=====
 NLOC   Avg.NLOC  AvgCCN  Avg.token  function_cnt  file
 -----
  250      12.3     2.4      55.3        19  Binfa.java
=====
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or parameter_count > 100)
=====
Total nloc   Avg.NLOC  AvgCCN  Avg.token  Fun Cnt  Warning cnt  Fun Rt  nloc Rt
-----
  250      12.3     2.4      55.3        19          0     0.00     0.00
(base) drob@drob-laptop:~/Dokumentumok/prog2/java$ 
```

13.12. ábra. Binfa.java ciklomatikus komplexitása

14. fejezet

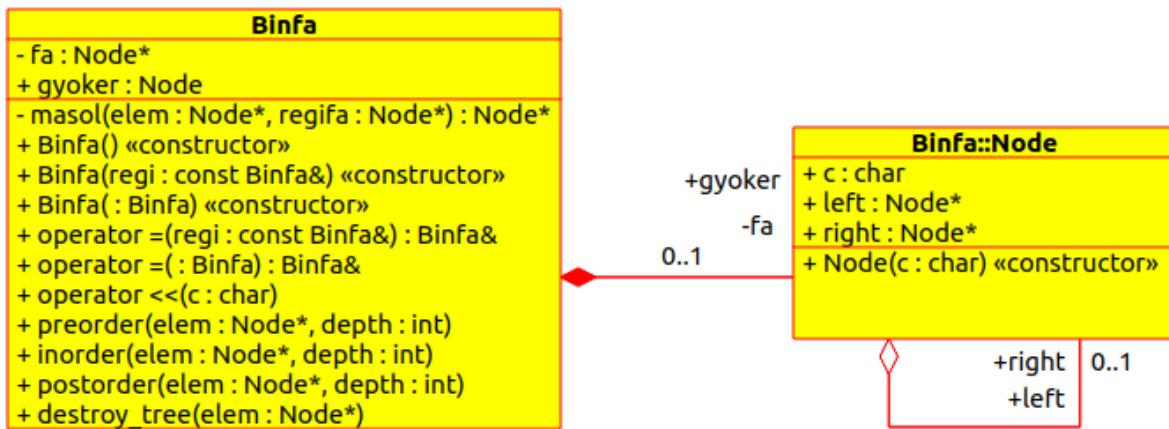
Helló, Mandelbrot!

14.1. Modellező eszközök és nyelvek. AZ UML és az UML osztálydiagramja.

A modellező nyelvek szerepe mindenkor hangsúlyos volt a szoftverfejlesztésben. UML (Unified Modelling Language) jelentése Egységes Modellező Nyelv. Mint a neve is sugallja egy olyan megoldást kíván amely egységesen segíti a szoftverfejlesztés egyes folyamatait (tervezés, dokumentálás, kommunikáció stb.). Továbbá egy olyan megoldást szolgál, amelyet a megrendelő és a fejlesztők is egységesen értenek. A különböző programozási elemekhez különböző szimbólumokat rendel. Gondolok itt tipikusan a rombusz alakzatot az IF elágazás reprezentánsa. Rengeteg előre definiált nyelvi eszközt adnak az UML nyelvek. A folyamatok irányát és az egyes elemek viszonyát nyilak adják. Rengeteg strukturális diagram fajtákat készíthetünk el: Osztálydiagram, objektumdiagram, komponens diagram stb. Illetve vannak viselkedési diagramok: Eset diagramok, aktivitási diagramok stb. minden típusnak megvan a sajátos nyelvi elemei és szabályai.

14.2. Reverse engineering UML osztálydiagram

Ebben a feladatban a tavalyi prog1 LZWBinfa C++ forráskódjából kellett UML osztálydiagramot készíteni. Majd beszélni kellett egy kicsit az aggregáció és kompozíció fogalmakról majd ezek kapcsolatáról a kód illetve a diagram között. Ezt a feladatot érdekesnek és hasznosnak találtam, mert pár kattintással hasznos UML diagramot hozhattam létre. Ez egyfajta leellenőrzésként is szolgálhat az egyes UML-es feladatoknál illetve a munka világában is. Ezen feladat elkészítéséhez a Tanár Úr által ajánlott Umbrello UML modellező szoftvert választottam. Telepítés után a Kódimportáló varázsló... segítségével importáltam a binfa forráskódját majd behúztam az osztálydiagram megjelenítőbe. A kiadott UML ábra:



14.1. ábra. Binfa UML osztálydiagram

Az aggregáció egy rész-egész kapcsolat, ahol az egyik egész-objektum tartalmazza, vagy birtokolja a részmásikat. A rész-objektum létezése az egész-objektumtól függ. Beszélhetünk gyenge- illetve erős aggregációról. Az előbbi jele egy üres rombusz míg az utóbbié egy teli rombusz. Ezeket az osztálydiagramom is szépen szemlélteti. Az erős aggregációt szokás nevezni még kompozíciónak. Kompozícióról beszélünk a Csomópont esetén hiszen önmagában nincs értelme "létezni".

14.3. Forward engineering UML osztálydiagram

Ebben a feladatban egy már kész UML osztálydiagramból kellett generáltatni forráskódot. Ehhez a feladathoz a Binfa java verzióját használtam. Ehhez legeneráltattam az Kódimport varázslóval az UML ábrát, majd az Umbrellaban kexportáltam az UML ábra szerint. A következő fain .java forráskódot kaptam:

```

import java.io.PrintWriter;
import Binfa.Csomopont;

/**
 * Class Binfa
 */
public class Binfa {

    //
    // Fields
    //

    protected Csomopont gyoker;
    private Csomopont fa;
    private int melyseg;
}

```

```
private int atlagosszeg;
private int atlagdb;
private double szorasosszeg;

//  
// Constructors  
//  
public Binfa () { }

//  
// Methods  
//  
  
//  
// Accessor methods  
//  
  
/**  
 * Set the value of gyoker  
 * @param newVar the new value of gyoker  
 */  
protected void setGyoker (Csomopont newVar) {  
    gyoker = newVar;  
}  
  
/**  
 * Get the value of gyoker  
 * @return the value of gyoker  
 */  
protected Csomopont getGyoker () {  
    return gyoker;  
}  
  
/**  
 * Set the value of fa  
 * @param newVar the new value of fa  
 */  
private void setFa (Csomopont newVar) {  
    fa = newVar;  
}  
  
/**  
 * Get the value of fa  
 * @return the value of fa  
 */  
private Csomopont getFa () {  
    return fa;  
}
```

```
/**  
 * Set the value of melyseg  
 * @param newVar the new value of melyseg  
 */  
private void setMelyseg (int newVar) {  
    melyseg = newVar;  
}  
  
/**  
 * Get the value of melyseg  
 * @return the value of melyseg  
 */  
private int getMelyseg () {  
    return melyseg;  
}  
  
/**  
 * Set the value of atlagosszeg  
 * @param newVar the new value of atlagosszeg  
 */  
private void setAtlagosszeg (int newVar) {  
    atlagosszeg = newVar;  
}  
  
/**  
 * Get the value of atlagosszeg  
 * @return the value of atlagosszeg  
 */  
private int getAtlagosszeg () {  
    return atlagosszeg;  
}  
  
/**  
 * Set the value of atlagdb  
 * @param newVar the new value of atlagdb  
 */  
private void setAtlagdb (int newVar) {  
    atlagdb = newVar;  
}  
  
/**  
 * Get the value of atlagdb  
 * @return the value of atlagdb  
 */  
private int getAtlagdb () {  
    return atlagdb;  
}  
  
/**  
 * Set the value of szorasosszeg
```

```
* @param newVar the new value of szorasosszeg
*/
private void setSzorasosszeg (double newVar) {
    szorasosszeg = newVar;
}

/**
 * Get the value of szorasosszeg
 * @return the value of szorasosszeg
 */
private double getSzorasosszeg () {
    return szorasosszeg;
}

//  

// Other methods  

//  

/***
 */
public void Binfa()
{
}

/***
 * @param b
 */
public void egyBitFeldolg(char b)
{
}

/***
 */
public void kiir()
{
}

/***
 * @param os
 */
public void kiir(java.io.PrintWriter os)
{
}

/***
 * @param elem
*/
```

```
* @param          os
*/
public void kiir(Binfa.Csomopont elem, java.io.PrintWriter os)
{



/***
 * @return         int
 */
public int getMelyseg()
{
}

/***
 * @param          elem
 */
public void rmelyseg(Binfa.Csomopont elem)
{
}

/***
 * @return         double
 */
public double getAtlag()
{
}

/***
 * @return         double
 */
public double getszoras()
{
}

/***
 * @param          elem
 */
public void ratlag(Binfa.Csomopont elem)
{
}

/***
 * @param          elem
 */
```

```
public void rszoras(Binfa.Csomopont elem)
{
}

/**/
public static void usage()
{
}

/**/
* @param      args
*/
public static void main(String[] args)
{
}

}
```

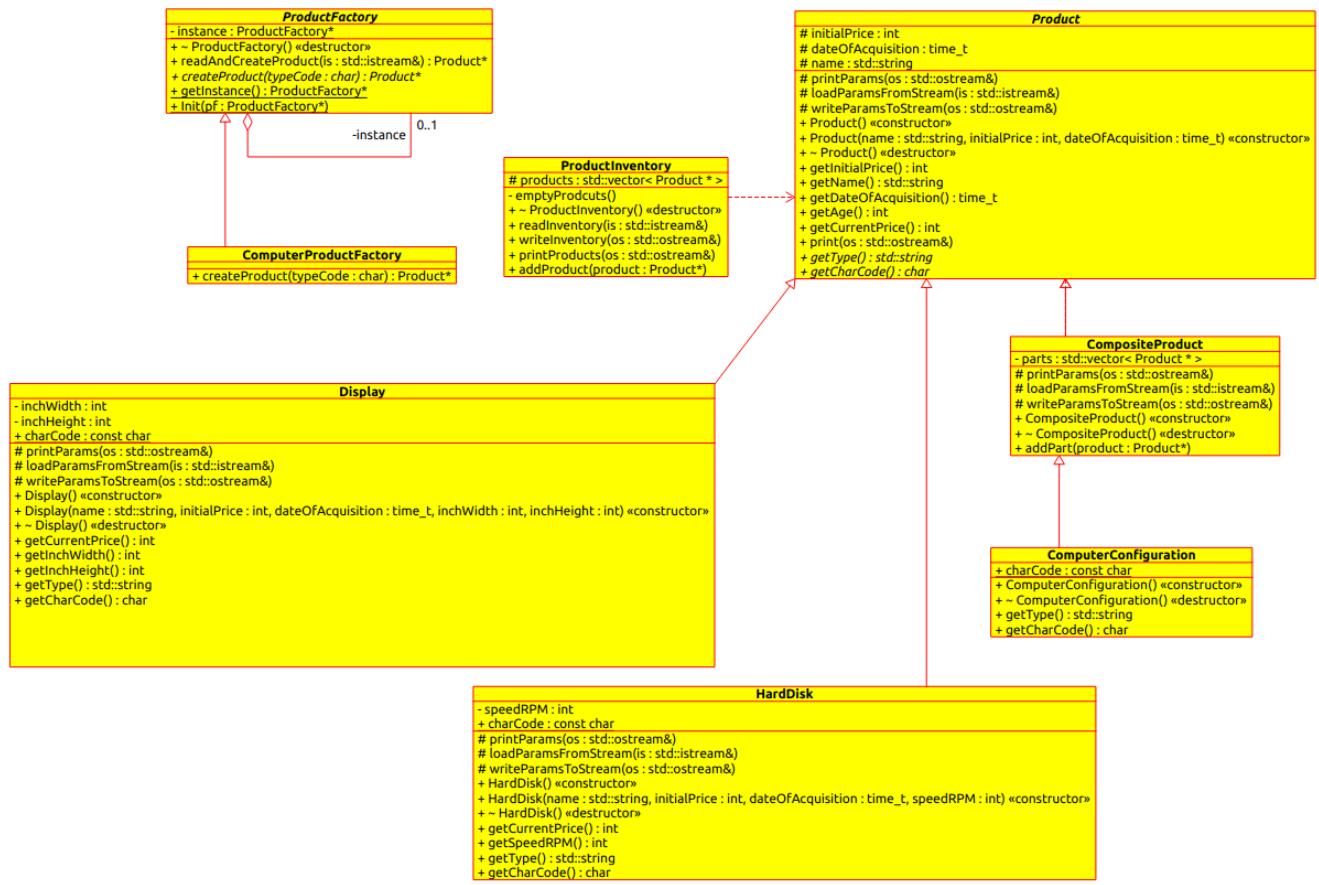
A visszakapott kódból látszik, hogy csak függvénydeklarációkat generált ami hűen tükrözi a Binfa osztály szerkezeti felépítését. Ezekhez függvénydefiníciókat nem kapunk hiszen azt majd nekünk kéne megírni. Tervezési fázisban nagyon hasznosak tudnak lenni az UML diagramok erre is próbált rávilágítani ezen két feladat.

14.4. Egy esettan

Ezt a feladatot a *Szoftverfejlesztés C++ nyelven* című könyv 14. fejezetét (427-444 elmelet, 445-469 az esettan) kellett feldolgozni.

A feladat áttekintése: Egy kereskedés számítógép-alkatrészek és számítógép-konfigurációk értékesítésével foglalkozik. Elsődleges feladtunk egy olyan alkalmazás elkszítése, amely lehetőséget biztosít a kereskedés alkatrészeinek és konfigurációinak nyilvántartására. Ennek keretében támogatnia kell a termékek állományból való betöltését, képernyőre történő listázását, állományba való kiírását és az árképzés rugalmas kialakítását.

Ezt a feladatot olvasva olyan érzésem volt, mintha hasonló UML-ból írnám meg a forráskódot csak ez szöveges formátumú. Jól követhető, hasznos ábrák alapján viszonylag könnyen megírható a program. Az egyes osztályok működését nem részletezném, mert a könyv jól leírja helyette készítettem az Umbrello UML szoftver segítségével UML osztálydiagramot. A diagram:



14.2. ábra. Termékek létrehozásáért felelős osztályok kapcsolata.

```

(base) dроб@роб-laptop:~/Документы/прог2/cpp-examples/esettan$ g++ -c *.cpp
(base) dроб@роб-laptop:~/Документы/прог2/cpp-examples/esettan$ g++ *.o -o test
(base) dроб@роб-laptop:~/Документы/прог2/cpp-examples/esettan$ ./test
Test1: create inventory and printing it to the screen.
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20191007, Age: 0, Current price: 30000, InchWidth: 13, InchHeight: 12
1.: Type: HardDisk, Name: WD, Initial price: 25000, Date of acquisition: 20191007, Age: 0, Current price: 25000, SpeedRPM: 7500
Press any key to continue...

Test2: loading inventory from a file (computerproducts.txt), printing it, and then writing it to a file (computerproducts_out.txt).
End of reading product items.The content of the file is:
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20011001, Age: 6579, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 4754, Current price: 28000, InchWidth: 10, InchHeight: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age: 4754, Current price: 70000
Items:
0. Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20011001, Age: 6579, Current price: 24000, InchWidth: 12, InchHeight: 13
1. Type: HardDisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 4754, Current price: 28000, SpeedRPM: 7000
3.: Type: HardDisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20050228, Age: 5333, Current price: 20000, SpeedRPM: 7000

The content of the inventory has been written to computerproducts_out.txt

Done.
(base) dроб@роб-laptop:~/Документы/прог2/cpp-examples/esettan$ 
  
```

14.3. ábra. Az programunk output-ja.

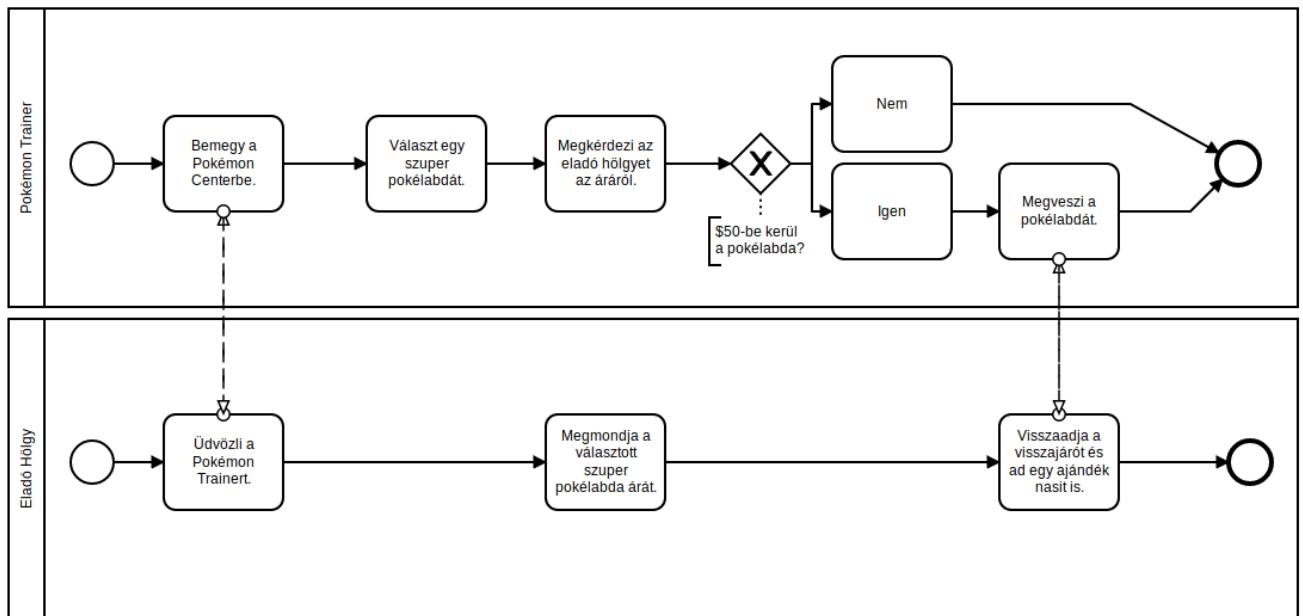
```
computerproducts_out.txt

1 d TFT1 30000 20011002 12 13
2 d TFT2 35000 20061001 10 10
3 c ComputerConfig1 70000 20061001 2
4 d TFT3 30000 20011002 12 13
5 h WesternDigital 35000 20061001 7000
6 h Maxtor 25000 20050301 7000
7
```

14.4. ábra. computerproducts_out.txt file-ba való exportálás.

14.5. BPMN

Ebben a feladatban egy tetszőleges üzleti folyamatot kellett modellezni BPMN-ben. A BPMN (Business Process Model and Notations) egy olyan leírás amely különböző üzleti folyamatokat modellez. A modellek reprezentálásához hasonló speciális grafikai elemeket használ, mint az UML. Ezek az Eventek (kör, tele-kör), Activity-k (négyzet), Gateway (trapéz), Associations (nyíl, szaggatott-nyíl). A neten sokféle BPMN szoftver megtalálható. Az én példámat a <https://demo.bpmn.io/> oldalon készítettem el. Az ábra:



14.5. ábra. BPMN modell.

Ezen az ábrán a két folyamatot külön blokk tartalmazza. Ezek a Pokemon Trainer és a Eladó hölgy. Mindkét folyamat egy start eseményre indul. Folytatásképp a Pokemon Trainer belép a Pokémon Center-be és ezzel párhuzamban az Eladó Hölgy köszön neki. Itt asszociáció jön létre, amelyet az modell jól mutat. Folytatódik a folyamat és a Trainer kiválaszt egy Pokélabdát, majd megkérdezi az árat. Itt két ágon lehet tovább a folyamat. Ha \$50-ra kerül, akkor megveszi a labdát, de ha nem annyi akkor elhagyja a Pokémon Centert (kilép a programból). Az árat az Eladó Hölgy határozza meg. Majd ha a feltétel igaz ágba fordul, akkor kiszolgálja a Pokémon Trainert (itt szintén asszociáció jön létre) majd kilép a folyamatból. Egyszerű de nagyszerű példa a BPMN lényegére. :D

14.6. TeX UML

Ebben a feladatban az OOCWC (rObOCar World Championship) projektről kellett diagramot készíteni LaTeX-ben. Ehhez a www.overleaf.com online latex editort használtam. Sok bolyongás és fejfájás mellett találtam egy ábrát, amely bemutatja, hogy milyen részekből tevődik össze a real-time traffic analyzer. Ezt az ábrát készítettem el TikZ package (LaTeX) segítségével. A forrás:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{tikz}

\tikzstyle{video} = [rectangle, minimum width=3cm, minimum height=1cm, text centered, draw=black, fill=blue!20]
```

```
\tikzstyle{camera} = [rectangle, rounded corners, minimum width=3cm, ←
    minimum height=2cm, text centered, draw=black, fill=green!40]

\tikzstyle{axi} = [rectangle, minimum width=3cm, minimum height=1cm, text ←
    centered, draw=black, fill=pink!80]

\tikzstyle{gps} = [rectangle, rounded corners, minimum width=2.5cm, minimum ←
    height=2cm, text centered, draw=black, fill=blue!65]

\tikzstyle{zynq} = [rectangle, minimum width=4cm, minimum height=3cm, text ←
    centered, draw=black, fill=green!15]

\tikzstyle{reset} = [rectangle, minimum width=2cm, minimum height=1cm, text ←
    centered, draw=black, fill=yellow!70]

\tikzstyle{axi2} = [rectangle, minimum width=3cm, minimum height=1cm, text ←
    centered, draw=black, fill=pink!80]

\tikzstyle{gsm} = [rectangle, rounded corners, minimum width=3cm, minimum ←
    height=2cm, text centered, draw=black, fill=yellow!40]

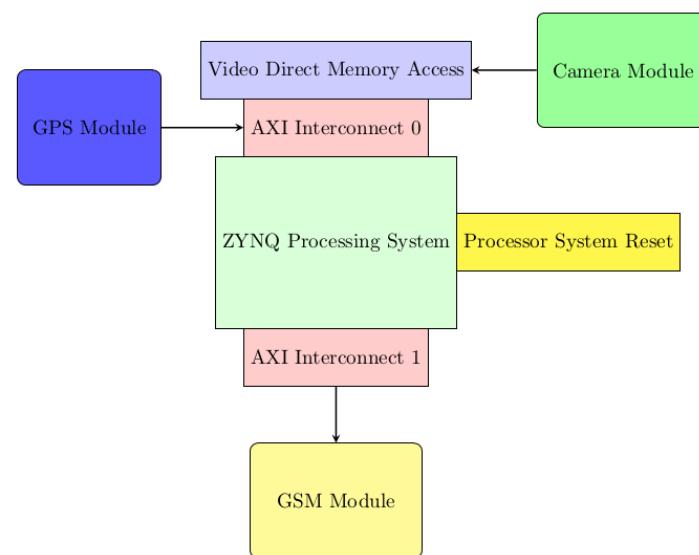
\tikzstyle{arrow} = [thick,->,>=stealth]

\begin{document}
\maketitle
\begin{tikzpicture}[node distance=2cm]
\node (video) [video] {Video Direct Memory Access};
\node (camera) [camera, right of=video, xshift=3cm] {Camera Module};
\node (axi) [axi, below of=video, yshift=1cm] {AXI Interconnect 0};
\node (gps) [gps, left of=axi, xshift=-2.3cm] {GPS Module};
\node (zynq) [zynq, below of=axi, yshift=0.0010cm] {ZYNQ Processing System ←
    };
\node (reset) [reset, right of=zynq, xshift=2.05cm] {Processor System Reset ←
    };
\node (axi2) [axi2, below of=zynq, yshift=0cm] {AXI Interconnect 1};
\node (gsm) [gsm, below of=axi2, yshift=-0.5cm] {GSM Module};
\draw [arrow] (camera) -- (video);
\draw [arrow] (gps) -- (axi);
\draw [arrow] (axi2) -- (gsm);
\end{tikzpicture}
\end{document}
```

OOCWC - Real-Time Traffic Analyzer

Dékány Róbert Zsolt

October 7, 2019



14.6. ábra. Real-Time Traffic Analyzer Figure made by LaTeX.

15. fejezet

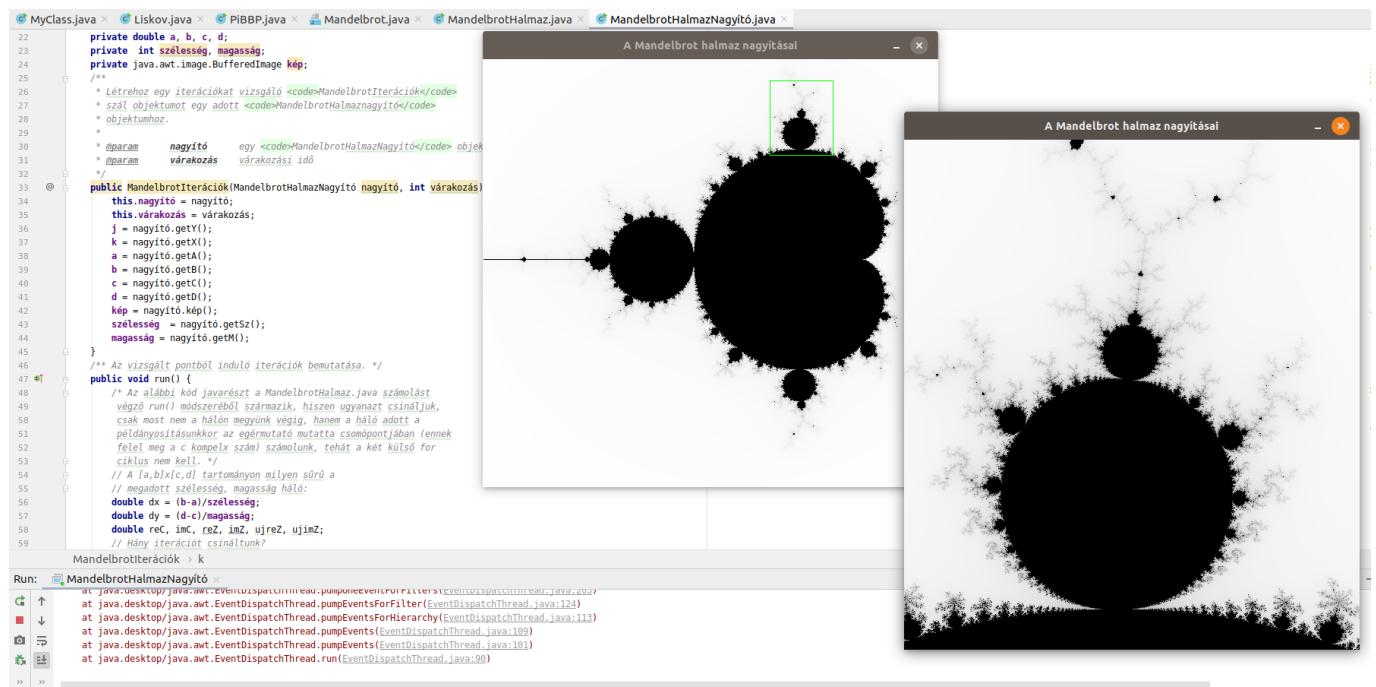
Helló, Chomsky!

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

A programozásban a karakterkészletek azon betűk, számok, szimbólumok írásának fajtát jelenti, amely egységes kódolási karakter gyűjteményt foglal magába. Legelterjedtebb az ASCII kódolás. Ebbe nincs benne a magyar ékezes betűk így konvenciák szerint ékezetmentes betűket használunk a kódolásban. Viszont a készítendő programunk magyar nyelvi támogatású akkor célszerű beállítani egy általános Unicode karakterkódolást, amely már tartalmazza az ékezes betűket is. Lexikális elemek a nyelv építőelemei, amelyek kifejezésekké, utasításokká állnak össze. Ezen elemek a nyelv által definiált típusok, kulcsszavak, szimbólumok, kifejezések összessége. Az utasítások helyessége erősen függ a nyelvtől illetve a használt programozási konvencióktól (pl. camelcase). Ilyen lehet pl. az, hogy változó neve nem kezdődhet számmal, nem lehet már előre definiált kulcsszó illetve egyedinek, sokatmondónak kell lennie. Utasítás lehet deklaráció illetve definíciós kifejezés. Deklaráció lehet egy változó deklaráció, amelyet lényegében csak létrehozunk, legfoglaljuk a memóriacímeket. Ezt ha használni szeretnénk akkor a programkódban definiáljuk. Például növeljük az értékét. Ugyaníg létrehozhatunk a függvényeket, attribútumokat. A definíciókat globális- illetve lokális névterben helyezhetjük el kód blokokban.

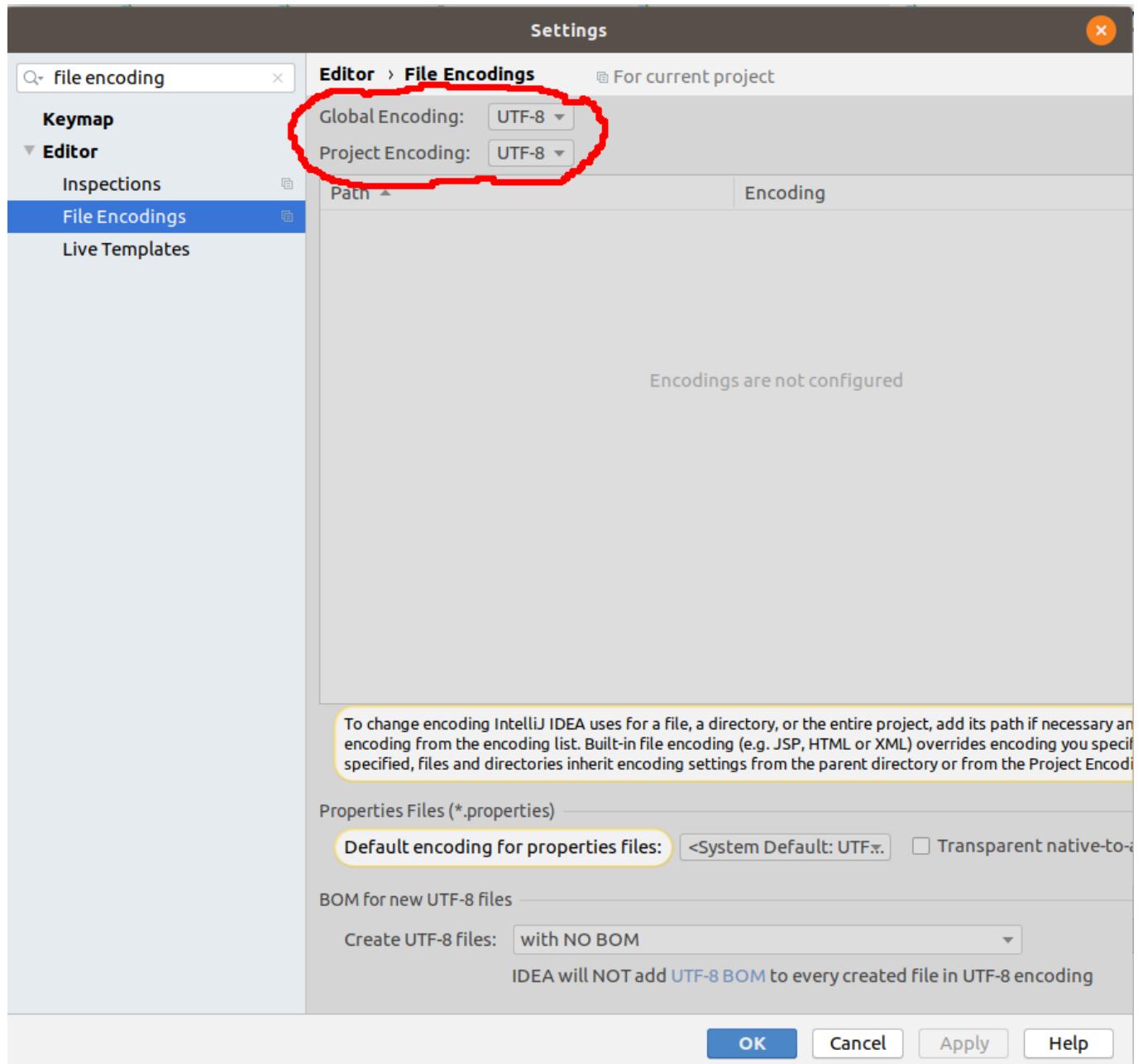
15.2. Encoding

Ebben a részben elméleti részel párhuzamosan az java encoding-ról lesz szó. A feladatban a java nyelven megírt Mandelbrot-halmaz nagyító forráskódját kellett teleírni magyar ékezes betűkkel. Az IntelliJ-ben ezt tapasztaltam:



15.1. ábra. Mandelbrot nagyító IntelliJJ-ben.

Tökéletesen lefutott, semmi hiba nem keletkezett. Ez azért történt, mert az IDE-ben default encoding alapértelmezetten UTF-8 kódolású, ami tartalmaz magy karakterkészletet. A Settings/File Encodings-ban jól látszik:



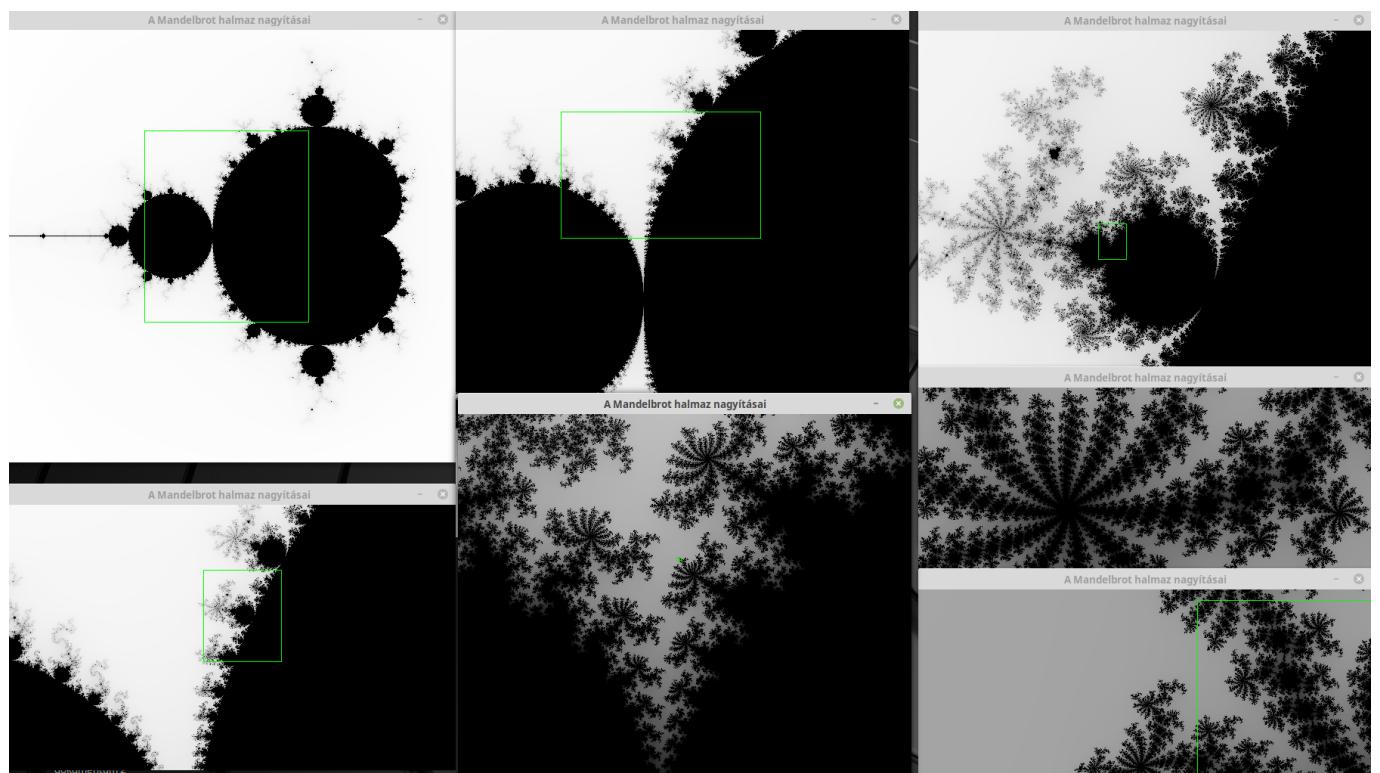
15.2. ábra. File Encoding

Gondolom, ha ilyen egyszerű lenne nem kaptunk volna ilyen feladatot így a parancssori javac fordítóval is lefordítottam. Az eredmény eltér az előbbiből:

```
(base) drob@drob-laptop:~/Dokumentumok/prog2$ javac MandelbrotHalmazNagyító.java MandelbrotIterációk.java
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xED) for encoding UTF-8
 * MandelbrotHalmazNagyító.java
      ^
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xF3) for encoding UTF-8
 * MandelbrotHalmazNagyító.java
      ^
MandelbrotHalmazNagyító.java:4: error: unmappable character (0xED) for encoding UTF-8
 * DIGIT 2005, Javat tanárok
      ^
MandelbrotHalmazNagyító.java:5: error: unmappable character (0xE1) for encoding UTF-8
 * Bétfai Norbert, nbtfai@inf.unideb.hu
      ^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xED) for encoding UTF-8
```

15.3. ábra. javac error a Mandelbrotra

A helyes megoldás, ha az alábbi parancsot adjuk ki: `javac -encoding "ISO-8859-2" MandelbrotHalmazNagyító.java MandelbrotIterációk.java`. Ezzel elérünk, hogy az ékezetes betűket felismerje és futtassuk a kódokat. Az eredmény:



15.4. ábra. Ismét lefutott.

15.3. Maurer Rose - full screen

Ebben a feladatban Java nyelven kellett írni egy full screen alkalmazást. A megadott példa több szempontból se volt működőképes így saját példaalkalmazást készítettem a feladatra. Valamelyik este pont néztem

a CodingTrain Youtube Csatornát, ahol megláttam és megtetszett a Maurer Rózsa vizualizációja. Így ezt választottam. Lényegében ha jól értelmeztem, akkor a rózsa egy olyan sinuszoid, ami egy polár koordináta rendszerben van ábrázolva. (ezt még pontosítom). Lényegében a megadott matematikai algoritmussal két bemenetet várok egy N és D számokat, amelyek megadják a rózsa tulajdonságait. A forráskódja:

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.Stroke;
import java.awt.geom.Path2D;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import java.awt.Toolkit;

@SuppressWarnings("serial")
public class RosePanel extends JPanel{
    private static Toolkit tk = Toolkit.getDefaultToolkit();
    private static int PREF_W = (int) tk.getScreenSize().getWidth();
    private static int PREF_H = (int) tk.getScreenSize().getHeight();
    private static final int MAX = 100;
    private static final double SCALE = 150.0;
    private static final double DELTA_X = PREF_W / 2;
    private static final double DELTA_Y = PREF_H / 2;
    private static Color ROSE_COLOR = Color.blue;
    private static final Stroke ROSE_STROKE = new BasicStroke(1f);
    private Path2D path = new Path2D.Double();

    public RosePanel() {

        float N = 6;
        float D = 71;

        for (float i = 0; i < 2 * Math.PI; i+= ((2 * Math.PI) / 360))
        {

            float k = i * D;
            float r = 250 * (float)Math.sin(N * k);
            float x = r * (float)Math.cos(k) + (float)DELTA_X;
            float y = r * (float)Math.sin(k) + (float)DELTA_Y;

            if (i == 0) {
                path.moveTo(x, y);
            } else {
                path.lineTo(x, y);
            }
        }
    }
}
```

```
        }
        path.closePath();

    }

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setColor(ROSE_COLOR);
    g2.setStroke(ROSE_STROKE);
    g2.draw(path);
}

@Override
public Dimension getPreferredSize() {
    if (isPreferredSizeSet()) {
        return super.getPreferredSize();
    }
    return new Dimension(PREF_W, PREF_H);
}

private static void createAndShowGui() {

    JFrame frame = new JFrame("Maurer Rózsa App");
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    RosePanel mainPanel = new RosePanel();
    frame.getContentPane().add(mainPanel);
    frame.setUndecorated(true);
    frame.setAlwaysOnTop(true);
    frame.pack();
    //frame.setLocationByPlatform(true);

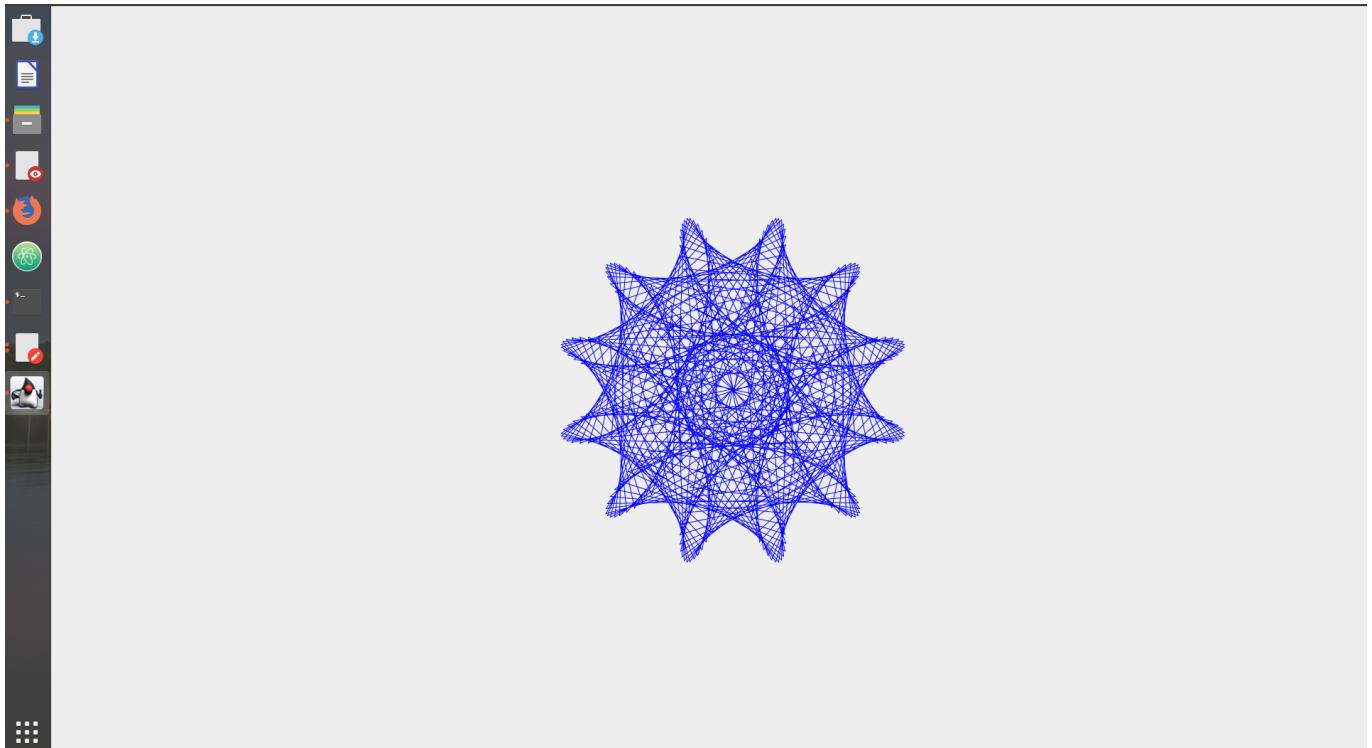
    frame.setVisible(true);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGui();

        }
    });
}
}
```

A kódban egy már meglévő ablakos alkalmazás szolgáltatta a vázat. A RosePanel() konstruktur fogja

kiszámolni a N és D értékekkel a rózsát majd egy path-ba rakom bele (Path2D). Ezután a paintComponent (Graphics) metódus végzi el a Rózsa 2D-s megvalósítását. Ehhez egy Grapic2D objektumot hozok létre, módosításokat hajtok végre rajta. Majd a konstruktor az ábrát visszaadja. A Dimension illetve a createAndShowGUI() végzi el a teljes képernyő létrehozását. Ehhez a már jól ismert JFrame osztály egy objektumát hívjuk segítségül. Ezt a kis program eddig csak megjelenít. A jövőben tervezek létrehozni egy kis animációt, amely az N és D értékek változásával az ábra is változik. A full screen app:



15.5. ábra. Maurer Rose Figure.

15.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció

A Paszigráfia Rapszódia Dr. Bátfai Norbert kutatási projektjének egyike. Lényege, hogy lehetővé teszi egy valódi ember és egy mesterséges lény között a kommunikáció létrejöjjön. Ebben a projektben a PaRa vizualizációs lehetőségeit vettük elő. A feladat az volt, hogy egy ilyen vizualizációt valósítunk meg OpenGL segítségével majd az eredeti programot módosítsuk. Én az alábbi módosításokat végeztem el a kódban:

```
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(0.0f, 0.0f, 0.0f); //ezzel elérem, hogy RGB szerint ←
    fekete legyen az alapja a kockának
```

```
for ( int i {0}; i<cubeLetters[idx].cc[0].size() /2; ++i )
{
    glBegin ( GL_QUADS );

    glColor3f(0.5f, 0.0f, 1.0f);

    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 );

    glColor3f(1.0f, 0.99f, 0.0f);

    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 );

    glColor3f(0.0f, 0.99f, 0.0f);

    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 );

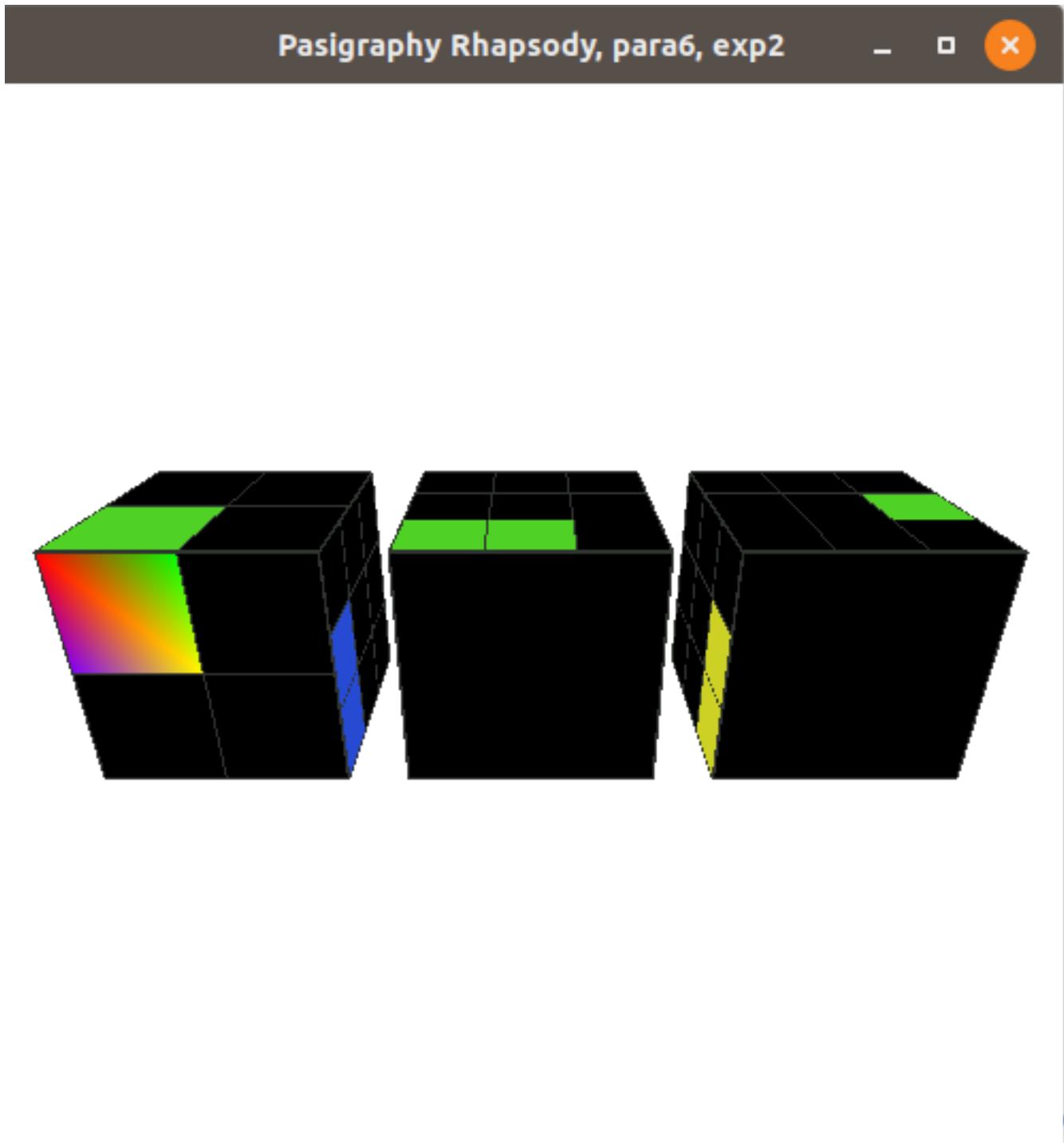
    glColor3f(1.0f, 1.1f, 0.0f);

    glColor3f(1.0f, 0.0f, 0.0f);

    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();
}
```

A kockánk 0 indexű oldalához hozzárendelt glColor3f-et irtam át illetve egészítettem ki. Az így kapott kockára egy szép átmenetet kaptam végeredményül. A sínskála meghatározásához egy online segédletet használtam. Íme a végeredmény:



15.6. ábra. Módosított OpenGL vizualizáció.

Hasonlóan epic lett, mint az Igazság Ligájában a 3 kocka...For the Justice :D

15.5. Perceptron osztály

Ebben a feladatban a jól ismert prog1-es Perceptron osztályos feladatot kellett feléleszteni. Tavaly skippelem ezt a feladatot így számomra ismeretlen volt, hogy mégis mit takar ez az egész. A prog2 labor órán Feri jól elmagyarázta, hogy mit kellene itt csinálni. Lényege, hogy a forrás kép RGB kódjait (mátrixát) bemásoljuk a neurális háló input rétegébe majd az utolsó réteg visszaad egy számot. A feladat az volt, hogy ne egy számot kapunk vissza, hanem a forrás kép mátrixát átírva generálunk egy más színvilágú képet. Először a ml.hpp-t kellett átírni az operator () túlterhelést, hogy ne egy double-t adjon vissza, hanem egy image tömböt. Ebben már a változott értékek szerepelnek majd. Kódban:

```
double* operator() ( double image [] )
{
    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
    {

        #ifdef CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );

        #else

        #pragma omp parallel for
        for ( int j = 0; j < n_units[i]; ++j )
        {
            units[i][j] = 0.0;

            for ( int k = 0; k < n_units[i-1]; ++k )
            {
                units[i][j] += weights[i-1][j][k] * units[i-1][k];
            }

            units[i][j] = sigmoid ( units[i][j] );
        }

        #endif
    }

    // return sigmoid ( units[n_layers - 1][0] );

    for( int i = 0; i < n_units[n_layers - 1]; i++)
        image[i] = units[n_layers - 1][i];

    return image;
}
```

Ezután átírtuk a main.cpp-t. A túlterhelt () operátor segítségével előállítunk egy double tömböt ami tartalmazni fogja a módosított kép értékeit. Ezután a készítendő új kép minden egyes RGB objektumának blue értékéhez ezeket rendeljük hozzá. Ezzel elérünk, hogy a képen megváltozott a háttér színe. Továbbiakban az új kép RGB objektumaihoz nyúltam és minden egyes fekete tartományba eső R G B objektumnak egy random számot adtam. Ezzel egy szép színes mandelbrot halmazt kaptam eredményül. Itt a main:

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

using namespace std;

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];

    double* copyImage = (*p) (image);

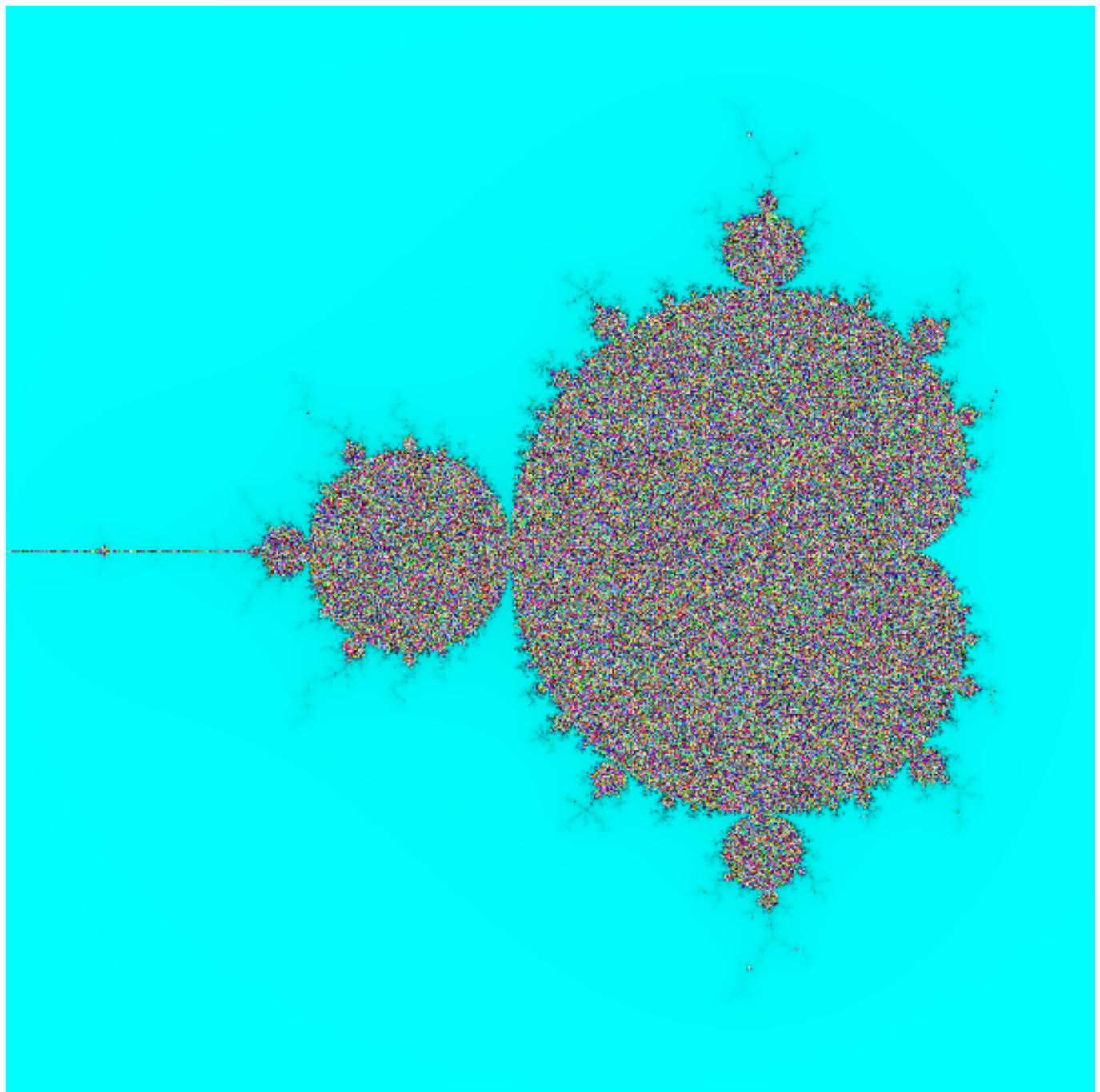
    for(int i = 0; i < png_image.get_width(); ++i)
        for(int j = 0; j < png_image.get_height(); ++j)
            png_image[i][j].red = copyImage[i*png_image.get_width()+j];

    for(int i = 0; i < png_image.get_width(); i++)
    {
        for(int j = 0; j < png_image.get_height(); j++)
        {
            if((int)png_image[i][j].red < 50 && (int)png_image[i][j].green < 50 && (int)png_image[i][j].blue < 50)
            {

                png_image[i][j].red = rand() % 255 + 0;
                png_image[i][j].green = rand() % 255 + 0;
                png_image[i][j].blue = rand() % 255 + 0;
            }
        }
    }

    png_image.write("modified.png");
```

```
delete p;  
  
delete [] image;  
  
}
```



15.7. ábra. Módosított Mandelbrot-halmaz képe.

16. fejezet

Helló, Stroustrup!

16.1. Objektumorientált programozási nyelvek típusrendszere (pl.: Java, C#) és 6. hét Típusok tagjai: mezők, (nevesített) konstansok, tulajdonságok, metódusok, események, operátorok, indexelők, konstruktörök, destruktörök, beágyazott típusok.

Az OOP-ban a mezők az osztályok adattagjai. Ezek az osztály működését általában meghatározzák így nem célszerű ha az értékük szabadon változtathatók. Ezért célszerű ezeknek az adattagoknak korlátozni az elérését tipikusan private accessorral. A konstansok olyan nem mutable változók, amelyek értéka a program teljes életciklusa alatt nem változtatható. Tipikus példa a PI érték kiemelése konstansként vagy épp a statikus MAX értékek. A tulajdonságok vagy property-k C#-ban tipikus implementáció található. Ezeket getter, setter-ek amelyek az osztály adattagjaihoz biztosít elérést metóduson keresztül. Ezekkel változtatni vagy ki tudjuk venni az értékeket. A metódus olyan kódblokk, amely utasításokat tartalmaz. Két fajtájuk van az eljárás és a függvény. Az eljárásnak nincs visszatérési értéke míg a függvénynek van. minden metódusnak van visszatérési értéke, neve és paramétere. A C# ban létezik beépített events, amelyek a különböző action-ökre (gomblenyomás, kattintás stb.) reagál. Az alkalmazásoknak reagálniuk kell ezekre az eventekre, amikor azok bekövetkeznek. Pl.: Ha megnyomjuk a Belépés gombot akkor írja ki, hogy logged. Az operátorok olyan beépített speciális karakterek, amelyek műveletek hajtanak végre. Vannak egyoperandusú, kétoperandusú operátorok. Operátorok többek között a matematikai-, logikai-, kapcsolati-operátorok. Konstruktör olyan speciális metódus, amely az osztály példányasításakor automatikusan lefut. A definícióban az neve az osztály nevével azonos. Hogy példányasításkor melyik konstruktör fog lefutni, az a paraméter-, számától vagy épp típusától függ. A destruktör fogja felszabadítani az erősforrásokat.

16.2. JDK osztályok

Ebben a feladatban a JDK src.zip file-ból kellett kiírni a .java kiterjesztésű osztályokat. A feladat egy bizonyos fénykard programra épül. Hogy miért ez a neve ne kérdezze senki. A programnak végig kell iterálnia a saját mappán belül elhelyezett/kicsomagolt src könyvtáron és kiírnia a .java fileokat. Ebben segítset nyújtott a boost filesystem és filestream könyvtár. A kód így nézne ki:

```
#include <stdio.h>
#include <string>
#include <vector>
#include <iostream>

#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>

using namespace std;

void readJavaExt ( boost::filesystem::path path, vector<string> & classes )
{
    if ( is_regular_file ( path ) )
    {

        std::string ext ( ".java" );
        if ( !ext.compare ( boost::filesystem::extension ( path ) ) )
        {
            classes.push_back(path.string());
        }
    }
    else if ( is_directory ( path ) )
        for ( boost::filesystem::directory_entry & entry : boost::filesystem::directory_iterator ( path ) )
            readJavaExt ( entry.path(), classes );
}

vector<string> getClass ( string path )
{
    vector <string> classes;

    boost::filesystem::path root (path);

    readJavaExt(root, classes);

    return classes;
}

int main(int argc, char const * argv[])
{
    boost::filesystem::path full_path(boost::filesystem::current_path());

    string root = full_path.string() + "/" + "src";

    vector<string> roots = {root};
```

```
vector<string> classes = getClass ( root );  
  
for(const auto & i : classes)  
{  
    cout << i << endl;  
}  
cout << " ←  
-----\n ←  
";  
cout << "Összesen " << classes.size() << ".java file található az src-  
ben.\n";  
  
return 0;  
}
```

A `vector < string > getClass (string path)` fogja visszaadni azt a string vektort amely tartalmazni fogja az abszolút elérést az egyes .java fileoknál. Paraméterként megkapja a root elérést. Létrehozunk egy string vektort, amivel visszatérünk. Majd meghívjuk a `readJavaExt (root, classes)` függvényt, ami elvégzi a tényleges keresést végzi.

A `JreadJavaExt(root, classes)` két paramétert vár. A root-ot fogja felhasználni a root elérés meghatározásához, ez azért szerencsés mert nem statikus az elérés, hanem bármilyen elérést meg tudunk adni. A `classes` pedig egy referencia, ami a `getClass` függvényben definiált vektor referenciajá lesz. A törzsbén vizsgáljuk: ha fájlelérést kapunk akkor megnézzünk, hogy .java-e a kiterjesztése. Ha igen akkor belerakjuk a vektorba. Ez a: `classes.push_back(path.string());` Ha viszont könyvtár elérést kapunk, akkor egyel bentebb megyünk a könyvtárstruktúrán majd rekurzívan újra hívjuk a függvényt.

A main-be először meghatározzuk az aktuális root elérést: `boost::filesystem::path full_path (boo` Ezután egy vektorba beletettsük az összes .java osztályt a `getClass()` metódus segítségével. Ezen a vektoron végigiterálva kiiratjuk az összes osztály elérést. Amit az alábbi kép is mutat:

```
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/MarkId.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/Timer.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/CodeSectionProcessor.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/JavaCallSiteRelocationInfo.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/JavaMethodInfo.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/arch64/AArch64ELFMacroAssembler.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/aarch64/AArch64InstructionDecoder.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/Main.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/jdk.tools.jaotc/src/jdk/tools/jaotc/AOTCompiler.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.aot/module-info.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/module-info.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/Console.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/ExternalEditor.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/JavacPackagesHelper.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/NashornCompleter.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/PackagesHelper.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/HistoryObject.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/PropertiesHelper.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/JrtPackagesHelper.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/EditableObject.java  
/home/drob/Dokumentumok/prog2/cpp-examples/fenykard/src/jdk.scripting.nashorn.shell/jdk/nashorn/tools/jjs/Main.java  
-----  
Összesen 17845 .java file található az src-ben.  
drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples/Fenykard$
```

16.1. ábra. JDK osztályok

16.3. Másoló-mozgató szemantika

Feladatként egy választott kódcsipetből kellett elmagyarázni a másoló-,mozgató- konstruktur és szemantika működését. A feladatot egy verem tulajdonságait implementáló példaprogram segítségével készítettem el. Amikor egy objektumról szeretnénk másolatot készíteni akkor jöhet képbe a mozgató konstruktur. A mozgató konstruktur egy referenciát vár paraméterül, amely megegyezik a osztály típusával. Célja a másolat létrehozása. Alapból, ha nem készítünk saját implementációt akkor a fordító automatikusan bitenkénti másolást végez. Ez ugyan létrehozza a másolatot viszont minden másolandó tartalom pointere ugyanazon memóriaterületre fog hivatkozni. Ez viszont egy hibás működés. Ezt a másolást sekély másolásnak (shallow copy)-nak hívjuk. Ha implementálunk egy saját másoló konstruktort akkor a fordító azt fogja használni. Mély másolásnak (deep copy) nevezünk azt, amikor a dinamikus adattagokat másoljuk. Ezek külön dinamikus memóriacímmel másolódnak így nem lesz azonos hivatkozás. A másoló

```
stack::stack(const stack & s) : cap(s.cap), sz(s.sz), v(nullptr)
{
    std::cout << __PRETTY_FUNCTION__ << " [ " << this << " ] [ " << &s << " ] " << ↵
        std::endl;

    v = new value_type [cap]; //lefoglalunk egy új v példányt

    value_type *dst = v;
    value_type *begin = s.v;
    value_type *end = begin + s.sz;

    for (value_type *src = begin; src != end; src++)
    {
        *(dst++) = *src;
    }
}
```

A másoló konstruktur egy bal érték referenciát kap paraméterül. A készülő stack objektum cap és sz adattagjai a másolt objektum értékeit kapja meg. A készülő v tagot meg kinullázzuk. A törzsben elsőként lefoglalunk a v-nek egy új példányt. Ezt követi egy úgynevezett iterátoros bejárás, amely végigiterál az s objektumon és a megfelelő értékeket beállítja. Végül az src dereferenciával megkapjuk a másolt példányt.

A másoló értékkopírozás (copy assignment) a következőként működik:

```
stack & stack::operator = (const stack & s)
{
    std::cout << __PRETTY_FUNCTION__ << " [ " << this << " ] [ " << &s << " ] " << ↵
        std::endl;

    if(cap != s.cap)
    {
        delete [] v;
        cap = s.cap;
        v = new value_type [cap];
```

```
}

sz = s.sz;

value_type *dst = v;
value_type *begin = s.v;
value_type *end = begin + s.sz;

for (value_type *src = begin; src != end; src++)
{
    *dst++ = *src;
}

return *this;
}
```

Hasonlóan balérték referenciát kap paraméterként. A törzsben egy check következik, ahol megnézem, hogy a stack mérete megegyezik-e a másolandó méretével. Ha nem akkor törlöm a régi verem objektumot, majd az új mérettel létrehozok egy újat. Ugyanígy következik egy iterátoros bejárás, majd végül visszatérünk az objektummal. * Hasonló megoldást használ az std::copy(...) is.

A mozgató konstruktor (move ctor) paraméterként egy jobbérték referenciát vár, amelyet egy `&&` jelöl. A mozgatás lényege, hogy egy objektum erőforrásait elvesszük azon célból, hogy hozzákapcsoljuk egy másik objektumhoz anélkül, hogy új területet foglalnánk. Az áthelyezés után már nincs szükség arra az objektumra, akitől elvettük. Célszerű alkalmazni olyan esetekben, amikor ideiglenes, megsemmisítés előtt álló objektumtól akarjuk elvenni az erőforrásokat. A move ctor kód:

```
stack::stack(stack && s) : cap(s.cap), sz(s.sz), v(s.v)
{
    std::cout << __PRETTY_FUNCTION__ << " [ " << this << " ] [ " << &s << " ] " << ←
        std::endl;

    s.cap = 0;
    s.sz = 0;
    s.v = nullptr;
}
```

A kódban jól látszik, hogy a tényleges s objektum értékeit hozzárendeljük az új stack tagjaihoz. Majd a törzsben a régi s objektum értékeit kinullázzuk. A mozgató szemantikánál hasonlóan felszabadítjuk a régi objektumot úgy, hogy áthúzzuk minden egyes tagját. Végül visszatérünk az új stack objektummal. A move assignment kódja:

```
stack & stack::operator = (stack && s)
{
    std::cout << __PRETTY_FUNCTION__ << " [ " << this << " ] [ " << &s << " ] " << ←
        std::endl;
```

```
    delete [] v;

    cap = s.cap;
    sz = s.sz;
    v = s.v;

    s.cap = 0;
    s.sz = 0;
    s.v = nullptr;

    return *this;
}
```

A main függvény az alábbi módon néz ki a programban:

```
#include <iostream>
#include <stdexcept>
#include <algorithm>

#include "stack.h"

stack f(stack const & s)
{
    stack result = s;
    result.push(333);
    return result;
}

int main()
{
    std::cout << "stack a(4);\n";
    stack a(4);

    std::cout << "stack b = f(a);\n";
    stack b = f(a);

    std::cout << "b = f(a);\n";
    b = f(a);

    std::cout << "stack c = std::move(f(a))" << '\n';
    stack c = std::move(f(a));

    std::cout << "return 0" << '\n';
    return 0;
}
```

```
(base) drob@drob-laptop:~/Dokumentumok/OOP-C++$ ./stack
stack a(4);
stack::stack(stack::size_type)[0x7ffd0edf7140]
stack b = f(a);
stack::stack(const stack&)[0x7ffd0edf7160] [0x7ffd0edf7140]
b = f(a);
stack::stack(const stack&)[0x7ffd0edf71a0] [0x7ffd0edf7140]
stack& stack::operator=(stack&&)[0x7ffd0edf7160] [0x7ffd0edf71a0]
stack::~stack()[0x7ffd0edf71a0]
stack c = std::move(f(a))
stack::stack(const stack&)[0x7ffd0edf71a0] [0x7ffd0edf7140]
stack::stack(stack&&)[0x7ffd0edf7180] [0x7ffd0edf71a0]
stack::~stack()[0x7ffd0edf71a0]
return 0
stack::~stack()[0x7ffd0edf7180]
stack::~stack()[0x7ffd0edf7160]
stack::~stack()[0x7ffd0edf7140]
(base) drob@drob-laptop:~/Dokumentumok/OOP-C++$ □
```

16.2. ábra. A main eredményei

16.4. Hibásan implementált RSA törése

Feladatként egy olyan programot kellett megírni, amely hibásan implementált RSA-ra betű gyakoriság alapján képes valamelyest visszafejteni a titkosított szöveget. Ehhez a Bátfai Tanár Úr RSA programja remek alapot adott. Ezt segédletként megkaptuk a feladathoz. Az RSA hátteréről illetve kifejtéséről az Összefoglaló résznél teszek említést. Itt most a program főbb felépítéseit, magyarázatát taglaljuk.

Több részből tevődik össze a program. Elsőként a kulcsokat generáló osztályt írjuk meg. Milyen kulcsokról beszélünk? A publikus és privát kulcsról. Ezek egy komoly matematikai algoritmus segítségével állnak elő. Először két BigInteger számot generálunk, amit a p és q prímszámok lesznek. Ennek a két prímnek a szorzata fogja megadni, az N-et. Az N lesz a modolusa a public és private kulcsoknak. A BigInteger z = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE)); fogja megadni az N-re vett relatív prímek számát. A d változó értéke fogja tartalmazni a titkos kulcsot. Az e változó értéke pedig a nyilvános kulcs.

```
package rsa;
import java.math.*;
import java.util.*;
import java.io.*;

class KulcsPar
```

```
{  
    BigInteger d, e, m;  
  
    public KulcsPar()  
    {  
        int meretBitekben = 700 * (int) (Math.log((double)10) / Math.log((  
            double)2));  
  
        BigInteger p = new BigInteger(meretBitekben, 100, new Random());  
  
        BigInteger q = new BigInteger(meretBitekben, 100, new Random());  
  
        m = p.multiply(q);  
  
        BigInteger z = p.subtract(BigInteger.ONE).multiply(q.subtract(  
            BigInteger.ONE));  
  
        do  
        {  
            do  
            {  
                d = new java.math.BigInteger(meretBitekben, new Random());  
            }  
            while(d.equals(BigInteger.ONE));  
  
        }  
        while(!z.gcd(d).equals(BigInteger.ONE));  
  
        e = d.modInverse(z);  
    }  
}
```

Ha megvan a kulcsgenerálás jöhet a sima titkosítás. Ez a folyamat a következőképp néz ki: A bufferből kivesszük az elemeket és a titkos tömbbe a már nyilvános kulcs N hatványára elemelt byteja fog kerülni. A decryption (visszafejtés) ugyanilyen módszerrel történik csak a privát kulcsot emeljük N-re. ÉS végül visszakaptuk a titkosított szöveget.

```
public class RSA  
{  
  
    public static void main(String[] args) throws IOException  
{  
  
        KulcsPar jszereplo = new KulcsPar();  
  
        String tisztaSzoveg = "Jurassic Park is a 1993 American science fiction ←  
            adventure film directed by Steven Spielberg and produced by ←
```

```
Kathleen Kennedy and Gerald R. Molen.";  
  
System.out.println("Eredeti szöveg: " + tisztaSzoveg);  
System.out.print("Visszafejtett szöveg: ");  
  
byte[] buffer = tisztaSzoveg.getBytes();  
  
BigInteger[] titkos = new BigInteger[buffer.length];  
  
for(int i = 0; i < titkos.length; ++i)  
{  
    titkos[i] = new BigInteger(new byte[] {buffer[i]});  
    titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo.m);  
}  
  
for(int i = 0; i < titkos.length; ++i)  
{  
    titkos[i] = titkos[i].modPow(jSzereplo.d, jSzereplo.m);  
    buffer[i] = titkos[i].byteValue();  
}  
}  
}
```

Ebből hogyan is tudjuk visszafejteni nem megengedett módszerekkel a szöveget? Sehogy! Vagyis van egy módja de ahhoz hibásan kell implementálni az egész folyamatot. Az ötlet az, hogy betűnként titkosítjuk a szöveget. Ekkor a lehetőségek számát belátható minimum alá helyezzük és hasonló lesz a titkosított szöveg. Ilyen esetben gyakoriságot számolva a titkosított szövegből visszakaphatunk valid részleteket az eredeti szövegből. Izgalmas. Lássuk hogyan is néz ez ki:

```
PrintWriter writer = new PrintWriter("output.txt");  
for(int idx = 0; idx < tisztaSzoveg.length(); ++idx)  
{  
    Character tisztaSzoveg = tisztaSzoveg.charAt(idx);  
    tisztaSzoveg = Character.toLowerCase(tisztaSzoveg);  
    byte[] buffer = tisztaSzoveg.toString().getBytes();  
    BigInteger[] titkos = new BigInteger[buffer.length];  
    byte[] output = new byte[buffer.length];  
  
    for(int i = 0; i < titkos.length; ++i)  
    {  
        titkos[i] = new BigInteger(new byte[] {buffer[i]});  
        titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo.m);  
  
        output[i] = titkos[i].byteValue();  
        writer.print(titkos[i]);  
    }  
    writer.println();  
}
```

```
writer.close();  
  
BufferedReader inputStream = new BufferedReader(new FileReader("output. ↵  
    txt"));  
int lines = 0;  
String line[] = new String[100000];  
  
while((line[lines] = inputStream.readLine()) != null)  
{  
    lines++;  
}  
  
inputStream.close();
```

A karakterek kezeléséhez és a gyakorisági statisztikához egy Karakterek osztályt készítünk. Ebben getter-setter metódusokat implementálunk a karakterekhez és titkosított tartalomhoz. Az increment() eljárás növeli a karakter gyakoriságát illetve a getGyak() visszaadja a tényleges gyakoriságot.

```
class Karakterek  
{  
    private String encrypted;  
    private char karakter = ' ';  
    private int gyakorisag = 0;  
  
    public Karakterek(String str, char k)  
    {  
        encrypted = str;  
        karakter = k;  
    }  
  
    public Karakterek(String str)  
    {  
        encrypted = str;  
    }  
  
    public void setEncrypted(String str)  
    {  
        encrypted = str;  
    }  
  
    public String getEncrypted()  
    {  
        return encrypted;  
    }  
  
    public void setKarakter(char k)
```

```
{  
    karakter = k;  
}  
  
public char getKarakter()  
{  
    return karakter;  
}  
  
public void increment()  
{  
    gyakorisag += 1;  
}  
  
public int getGyak()  
{  
    return gyakorisag;  
}  
}
```

A karakterek gyakoriságát az alábbi algoritmus számolja:

```
Karakterek kar[] = new Karakterek[1000];  
  
boolean found;  
kar[0] = new Karakterek(line[0]);  
int count = 1;  
  
for(int i = 1; i < lines; i++)  
{  
    found = false;  
    for(int j = 0; j < count; j++)  
    {  
        if(kar[j].getEncrypted().equals(line[i]))  
        {  
            kar[j].increment();  
            found = true;  
            break;  
        }  
    }  
    if(!found)  
    {  
        kar[count] = new Karakterek(line[i]);  
        count++;  
    }  
}  
  
for(int i = 0; i < count; i++)  
{
```

```
for(int j = i + 1; j < count; j++)
{
    if(kar[i].getGyak() < kar[j].getGyak())
    {
        Karakterek x = kar[i];
        kar[i] = kar[j];
        kar[j] = x;
    }
}
}
```

A kar[] tömbbe kerülnek az egyes Karakter objektumok. Egy forciklussal végigiterálunk az output.txt titkosított szöveget tartalmazó file-on és a Karakter osztály segéd metódusaival elkészítjük a gyakoriságot. Ezután következik a sorba rendezés. Ha megnézzük hogyan is megy a betű gyakoriság amelyhez viszonyítani akarjuk akkor csökkenő sorrendbe vannak rendezve az egyes betűk. Ezt nekünk is megkell tennünk a készített objektumainkon így meg is tettük. A javasolt betű gyakoriság elkészítéséhez a javasolt oldalt használtam: https://mdickens.me/typing/letter_frequency.html

```
BufferedReader f = new BufferedReader(new FileReader("gyak.txt"));

char[] karakter = new char[31];
//char[] karakter = new char[69];
int karCount = 0;
String k;

while((k = f.readLine()) != null)
{
    karakter[karCount] = k.charAt(0);
    karCount++;
}
f.close();

for(int i = 0; i < count; i++)
{
    kar[i].setKarakter(karakter[i]);
}

for(int i = 0; i < lines; i++)
{
    for(int j = 0; j < count; j++)
    {
        if(line[i].equals(kar[j].getEncrypted()))
        {
            System.out.print(kar[j].getKarakter());
        }
    }
}
System.out.println();
```

A kész gyak.txt-t beolvastam egy karakterek char típusú tömbbe majd ezt felhasználva egy ciklussal összevettem a titkosított gyakorisággal. Íme az eredmény, amely nem az igazi de hellyel-közzel fellelhető valami:

Eredeti szöveg: Jurassic Park is a 1993 American science fiction adventure film directed by Steven Spielberg and produced by Kathleen Kennedy and Gerald R. Molen.
Visszafejtett szöveg: vgiarros daic or a "yyk ameiosat rsoetse woshout anbethgie wolm noieshen fp rhebet rdoelfei, atm diungsen fp cah'leet cettenp atm ,eialn i. mulet.

16.3. ábra. Végeredmény a hibásan implementált RSA-ra.

Továbbiakban Feri mondta, hogy készíthetünk a saját gyakoriságot a titkosítandó szövegre. Én ezt el is készítettem a <https://www.browserling.com/tools/letter-frequency> oldal segítségével majd a gyak.txt-t erre átírva a következő eredményt kaptam:

Eredeti szöveg: Jurassic Park is a 1993 American science fiction adventure film directed by Steven Spielberg and produced by Kathleen Kennedy and Gerald R. Molen.
Visszafejtett szöveg: 1pdassic badk is a jgg3 aoedican science ficlimn arvenlpde fito ridecler uy slevien sbietued. anr bdmrpcer uy kalheen kennery anr .edatr d9 omten9

16.4. ábra. Végeredmény saját gyakorisággal.

Ennél már sokkal jobban kivehető mi is volt az eredeti szöveg. :)

16.5. Változó argumentumszámú ctor

Ebben a feladatban a Perceptron-t kellett ismét elővenni. Az osztályt úgy kellett alakítani, hogy ne egy számmal térjen vissza hanem egy hasonló méretű pixeltömbbel. Majd ezt felhasználva készítünk egy az eredeti képpel megegyező méretű képet.

```
Perceptron ( int nof, ... )
{
    n_layers = nof;

    units = new double*[n_layers];
    n_units = new int[n_layers];

    va_list vap;

    va_start ( vap, nof );

    for ( int i {0}; i < n_layers; ++i )
    {
        n_units[i] = va_arg ( vap, int );
        if ( i )
            units[i] = new double [n_units[i]];
    }
}
```

```
}

va_end ( vap );

weights = new double**[n_layers-1];

#ifndef RND_DEBUG
std::random_device init;
std::default_random_engine gen {init() };
#else
std::default_random_engine gen;
#endif

std::uniform_real_distribution<double> dist ( -1.0, 1.0 );

for ( int i {1}; i < n_layers; ++i )
{
    weights[i-1] = new double *[n_units[i]];

    for ( int j {0}; j < n_units[i]; ++j )
    {
        weights[i-1][j] = new double [n_units[i-1]];

        for ( int k {0}; k < n_units[i-1]; ++k )
        {
            weights[i-1][j][k] = dist ( gen );
        }
    }
}
}
```

Mint látható a Perceptron konstruktör paraméterében egy furcsa megadás van. `Perceptron (int nof, ...)` Ezt a megadási módöt úgy hívjuk, hogy vátozó számú paraméter lista. Az `int nof` paraméter változó fogja megmondani a tényleges paraméter számot. Ezután egy ... jelölés következik. Ahhoz, hogy dolgozni tudjuk a paraméterekkel 1 típus és 3 metódus áll a rendelkezésünkre. A `va_list` definiálja a változó típust. Ezen típussal deklarált változó fogja hordozni a paraméter változókról az információt. A `va_start()` inicializálja a változókat. A `va_args()` fogja visszaadni a paraméter változókat. Ez így nézhet ki: `str=va_arg(vl,char*)`. A felhasználás végén pedig kötelezően le kell zárni a `va_end()` metódussal a paraméter listát. Ezeket az alkotókat láthatjuk a fenti kódban is.

A `main.cc` kódja sokat nem változott az előző hetihez képest. Viszont az jól látszik, hogy itt kihasználtunk a paraméterlista előnyeit. A Perceptron konstruktora 4 paramétert fog továbbpasszolni majd ezekkel különböző műveletek hajtódnak végre: `Perceptron* p = new Perceptron (3, size, 256, size)`

```
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];  
  
double* copyImage = (*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].red = copyImage[i*png_image.get_width()+j];  
  
  
for(int i = 0; i < png_image.get_width(); i++)  
{  
    for(int j = 0; j < png_image.get_height(); j++)  
    {  
        if((int)png_image[i][j].red < 50 && (int)png_image[i][j].green ←  
            < 50 && (int)png_image[i][j].blue < 50)  
        {  
  
            png_image[i][j].red = rand() % 255 + 0;  
            png_image[i][j].green = rand() % 255 + 0;  
            png_image[i][j].blue = rand() % 255 + 0;  
        }  
    }  
}  
  
png_image.write("output.png");  
  
delete p;  
  
delete [] image;  
}
```

```

drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples/perceptron$ ll
összesen 900
drwxr-xr-x 2 drob drob 4096 okt 21 00:34 .
drwxrwxr-x 6 drob drob 4096 okt 20 22:13 ..
-rw-rw-r-- 1 drob drob 1129 okt 14 00:42 main.cpp
-rw-r-xr-x 1 drob drob 100464 okt 13 18:30 mandel*
-rw-r--r-- 1 drob drob 66839 okt 13 18:30 mandel.png
-rw-rw-r-- 1 drob drob 2382 okt 13 20:50 mandelpng.cpp
-rw-rw-r-- 1 drob drob 8193 okt 14 00:40 ml.hpp
-rw-r--r-- 1 drob drob 287912 okt 21 00:34 modified.png
-rw-r--r-- 1 drob drob 292172 okt 13 23:19 output.png
-rw-r-xr-x 1 drob drob 134448 okt 21 00:34 perc*
drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples/perceptron$ identify mandel.png
mandel.png PNG 600x600 600x600+0+0 8-bit sRGB 66.8KB 0.000u 0:00.000
drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples/perceptron$ identify output.png
output.png PNG 600x600 600x600+0+0 8-bit sRGB 292KB 0.000u 0:00.000
drob@drob-laptop:~/Dokumentumok/prog2/cpp-examples/perceptron$ 

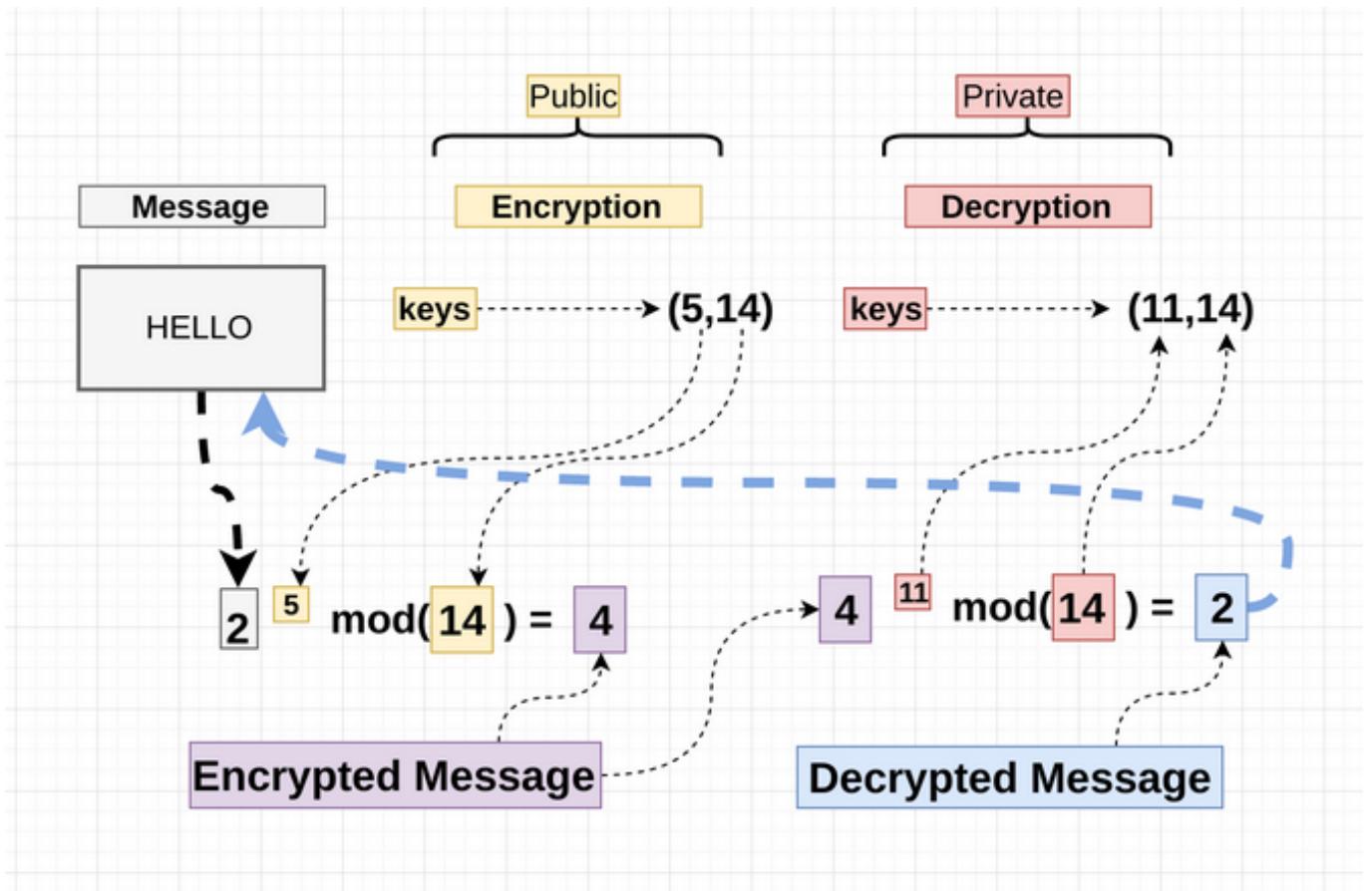
```

16.5. ábra. Mint látható megegyezik a két kép mérete.

16.6. Összefoglaló

Az összefoglalót az RSA ról készítettem el, mert erről lehet hosszabban írni. Hasznosnak találtam, mivel z RSA-eljárás nyílt kulcsú (vagyis „aszimmetrikus”) titkosító algoritmus, melyet 1976-ban Ron Rivest, Adi Shamir és Len Adleman fejlesztett ki (és az elnevezést nevük kezdőbetűiből kapta). Ez napjaink egyik leggyakrabban használt titkosítási eljárása. Előnye, hogy a titkosítás folyamatát elválasztja a dekódolástól. A titkosítás paramétere vagyis az e és d nem azonos így nem határozható meg a dekódolásra szánt paraméter. Röviden az üzenetet a nyilvános kulcs segítségével kódoljuk, amit visszafejteni csak egy titkos kulcs segítségével lehet. A kulcsokat a következő lépésekkel állnak össze: Véletlenszerűen kiválasztunk két nagy (bigInteger) prímszámot ezek a p és q. Következőnek ki kell számolnunk a encrypt és decrypt re szánt modulust, ami az N. Majd $\varphi(N)$ kiszámoljuk hány relatív prím szám van N-re. Ezt felhasználva készítünk egy e nyilvános kulcsot, ami az $\text{Inko}(e, \varphi(N))=1$ képletből áll össze. A titkos kulcs egy olyan d * e, amelyet elosztva $\varphi(N)$ -el 1 maradékot fog adni. Vagyis képlet szerint a következő: $\text{de} \equiv 1 \pmod{\varphi(N)}$. Tegyük fel, hogy B üzenetet akar küldeni A-nak. B az A személy nyilvános kulcsa segítségével titkosítja az üzenetet. A titkosított c szöveg a publikus kulcs n hatványára emelt m érték vagyis $c = m^e \pmod{N}$. A ezután saját titkos kulcsát, d-t használva vissza tudja fejteni m-et c-ből a következőképpen: $m = c^d \pmod{N}$.

A kódoló (küldő) és a dekódoló (fogadó) félnek nem kell semmilyen titkos jelszót vagy kulcsot cserélnie egymással a kommunikáció során. Ehelyett minden egyes felhasználó rendelkezik egy kulcspárral, mellyel a biztonságos kommunikáció létrejöhét. Az egyik kulcsot magánkulcsnak (privát kulcsnak), a másikat nyilvános kulcsnak (publikus kulcsnak) nevezzük. (Szakirodalomban használatosak még a „nyilvános és titkos kulcs” kifejezések is.) A két kulcs ugyanazon kulcsgenerálási eljárásból származik, elválaszthatatlanul összetartoznak, de egyik a másikból nem következtethető ki. A magánkulcsot (privát kulcsot) minden felhasználónak titkosan kell kezelnie, míg a nyilvános kulcsot (publikus kulcsot) mindenivel ismertetni kell, akivel kommunikálni szeretnénk (kulcsadatbázis segítségével). Az aszimmetrikus titkosítás esetén a feleknek nincs szükségük megbízható csatornára vagy személyes találkozásra, csak a kulcs hitelességét, tulajdonosához való tartozását kell bizonyítani.



16.6. ábra. RSA működése.

17. fejezet

Helló, Godel!

17.1. Interfészek. Kollekciók. Funkcionális nyelvi elemek. Lambda kifejezések.

Az interface hivatott kiterjeszteni az absztrakciót. Ezek absztrakt osztályok, amelyek test nélküli metódusokat tartalmaz. Ezt az osztály más osztályok implementálják és kifejtik a metódusokat (ami kötelező). Az interfacek előnyei, hogy elrejtik a megvalósítást és csak a lényegi részt adja meg. Másik fontos előnye, hogy a többszörös öröklődést nem támogató nyelvekben is meg tudjuk ezt valósítani. Collection-ok vagy gyűjtemények olyan programozási eszközök, amelyek több különböző típusú adatokat tárol. Nem csak a tárolás a fontos, hanem az adatokkal végezhető műveletek így azokra különböző metódusokat szolgáltat. A Collectionek több adatszerkezete van: List, Deque, Set, Map. Műveleteik lehetnek: add(), remove(), size(), isEmpty(), contains(). Lambda kifejezések olyan megvalósítást segítő kifejezés, amely jóval egyszerűbbé, olvashatóbbá teszi kódunkat. Használtatára egy példa: button.addActionListener(event -> System.out.println("gomb lenyomva")); A példában a button objektumra esemény feliratkozás történik, majd paraméterében megmondjuk mi is történjen. Az event egy helyettesítés, amely egy actionPerformed(ActionEvent event) speciális metódus. Ennek adjuk át a -> lambda operátor jobb oldalán álló kódot.

17.2. Gengszterek

A OOCWC projekt gengszterek rendezését kellett megnézni lambda kifejezéssel. Hasonló az STL map-es feladathoz itt is a jól bevált sort lambda kifejezéssel rendezünk. Ez az általunk módon néz ki:

```
std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( Gangster x, ←
    Gangster y ) ←
{
    return dst ( cop, x.to ) < dst ( cop, y.to );
} );
```

Két Gangster objektumot vár paraméternek majd megézzük a dst (cop, x.to) függvény kiszámolja a paraméterként megadott rendőr és gangster távolságát. Ugyanezt megnézzük y gangsterre. És ha az x gangster közelebb van a cop-hoz, mint az y gangster akkor tér vissza igazzal.

17.3. C++11 Custom Allocator

Ezt a feladatot SKIPPELEM a Tanárúr által kirakott PHP-s skippelési lehetőségével. A feladatot egy új sectionként definiáltam.

17.4. STL map érték szerinti rendezése

Ebben a feladatban a fénykard rank rendező algoritmusában kellett rámutatni a map érték szerinti rendezésére. A feladat fontos, ha tisztában vagyunk a map-el. Ez egy olyan asszociatív tároló amely kulcs érték - mapped érték párokat tárol. Szintaxisa a következő: `std::map <std::string, int> &rank`. A map-nek különböző segédmetódusai vannak. A feladat szempontjából kettő érdekes. A `begin()` amely az első elemmel tér vissza a map-ből. Illetve a `end()` ami visszatér azzal az elemmel ami az utolsót követi. Most lássuk a kódot:

```
std::vector<std::pair<std::string, int>> sort_map ( std::map <std::string, int> &rank )
{
    std::vector<std::pair<std::string, int>> ordered;

    for ( auto & i : rank ) {
        if ( i.second ) {
            std::pair<std::string, int> p {i.first, i.second};
            ordered.push_back ( p );
        }
    }

    std::sort (
        std::begin ( ordered ), std::end ( ordered ),
        [ = ] ( auto && p1, auto && p2 ) {
            return p1.second > p2.second;
        }
    );

    return ordered;
}
```

Mint látható a `sort_map` függvény paraméterként egy `<std::string, int>` map-et vár. Általánosságba kulcs érték rendezünk, de most feladatban érték szerint kell rendezni. Ehhez létrehozunk egy hasonló map típusú vektort `ordered` néven. Ebbe a vektorba foreach iterátor segítségével átrakjuk a paraméterként megadott map értékeit a vektorba. Ezt a vektort innenstől könnyen lehet érték szerint rendezni. A `sort` lambda kifejezéses "kombóval" rendezzük a vektort úgy, hogy a belső iterátornál a `p1.second` értékét vizsgáljuk. Ha nagyobb, mint a `p2.second` érték (ami lényegében a tényleges int értékek) akkor lesz a kifejezés igaz. Majd a végén a `sort_map` függvény visszatér az érték szerint rendezett vektorral.

17.5. Alternatív Tabella rendezése

Feladatként el kellett magyarázni az `java.lang Interface Comparable<T>` szerepét az magyar labdarugó bajnokság alternatív tabelláján. A `Comparable<T>` egy interface amelyet az implements kulcsszóval tudunk implementálni a classhoz. Ez az interface egyetlen függvényt definiál, a `public int compareTo(T o);`. Ez a függvény a paraméterként átadott T objektumot hasonlítja össze egy másik objektummal. Visszatérésként egy számot kapunk a két objektum relációjától függően. Három fajta áll fent: less than (visszaad negatívat), equals(visszaad nullát), greater than(visszaad pozitívat).

```
class Csapat implements Comparable<Csapat> {

    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }

    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Ahogy látható a kódóból is implementáltuk a `Comparable`-t a class `Csapatra`. Ezután kifejtettük a `compareTo(Csapat csapat)`. Itt az osztály attribútumként megadott `ertek`-et hasonlítjuk össze a paraméterként átadott `Csapat` objektum értékével. A már korábban leírt módon történik a visszatérés.

17.6. GIMP Scheme

Feladat az volt, hogy a tavalyi gimp scheme króm effectes példát kellett feléleszteni. A program egésze így néz ki:

```
; bhax_chrome3.scm
;
; BHAX-Chrome creates a chrome effect on a given text.
; Copyright (C) 2019
; Norbert Bátfai, batfai.norbert@inf.unideb.hu
; Nándor Bátfai, batfai.nandi@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 <https://www.gnu.org/licenses/>.
;
; Version history
;
; This Scheme code is partially based on the Gimp tutorial
; http://penguinpeta.com/b2evo/index.php?p=351
; (the interactive steps of this tutorial are written in ←
; Scheme)
;
; https://bhaxor.blog.hu/2019/01/10/ ←
; a\_gimp\_lisp\_hackelese\_a\_scheme\_programozasi\_nyelv
;

(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)
)

; (color-curve)

(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)
)
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-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)
)

;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))

(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)))
```

```
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE ←
)

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 ←
 0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 ←
 0 0))
(gimp-selection-invert image)

;step 6
(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)

;step 7
(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)))

;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 ←
  25 7 5 5 0 0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 ←
' (255 0 0) "Crown molding")

(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)0"
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 program úgy kezdődik, hogy definiálunk egy `color-curve` függvényt. A let kulcsszóval megadunk egy lokális változót, ami egy 8 elemű tömb lesz. Ezután feltöltjük az értékeit különböző értékekkel. Ez lesz a színátmenetért felelős függvény.

```
(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)
)
; (color-curve)
```

A függvény 3 paramétert vár. Magát a szöveget, amit formázni szeretnénk. A szöveg betűstílusát illetve a szöveg méretét. `set!` kulcsszóval beállítunk értékeket a változóknak és a változók globális értékekké válnak. Létrehozunk két változót `text-width` illetve `text-height`-t és beállítjuk az értékeiket 1-re. Majd a `set!` segítségével beállítjuk a további értékeket a paraméterként megkapott 3 érték alapján.

```
(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)
)
)

; (text-width "alma" "Sans" 100)
```

Az alábbi programban fog megtörténni a chrome effect leimplementálása. A script-fu-bhax-chrome függvény 7 paramétert vár. Ezek a következők: (script-fu-bhax-chrome "formázandó szöveg" "betűstílus" betűméret szélesség magasság színskála "színezési stílus") A továbbiakban írni fogok még róla. Időhiány stb.

```
(define (script-fu-bhax-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)
)

;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))

(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)))

;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE ←
    )

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 ←
    0 1 TRUE)
```

```
; step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

; step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 ←
    0 0))
(gimp-selection-invert image)

; step 6
(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)

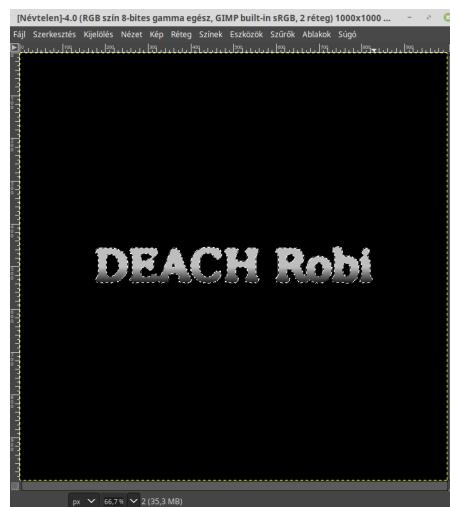
; step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM ←
    LAYER-MODE-NORMAL-LEGACY GRADIENT-LINEAR 100 0 ←
    REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- ←
        height (/ height 3)))

; step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 ←
    25 7 5 5 0 0 TRUE FALSE 2)

; step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 ←
' (255 0 0) "Crown molding")
```



17.1. ábra. GIMP scheme által kreált króm szöveg.

17.7. PHP

Ebben a feladatban a DEAC 100 hours challenge PHP kódjában kellett asszociatív rendezést végezni. A teljes kód így nézne ki:

```
<?php
session_start();
if(!isset($_SESSION["hackerin"]) && $_SESSION["hackerin"] == true)
{
header("location: index.php");
exit;
}
$hackername = $_SESSION["hackername"];
$hackertime = $_SESSION["hackertime"];
$time = $_GET["time"];
if(!empty($time))
{
$hackertime = $time;
$hackerfile = "/tmp/".$_SESSION["hackername"];
file_put_contents($hackerfile, $time);
}
?>
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<style>
p {
margin-top: 0px;
text-align: center;
```

```
font-size: 55px;
font-variant: small-caps;
}
img {
display: block;
margin-left: auto;
margin-right: auto;
}
</style>
</head>
<body>
<p id="ido">Streameljünk 100 óra DEAC játékot!</p>
<img src = "http://deac.hu/upload/5b376e152b0b5.png" width="300">
<p id="hanyan" class="smallcap">Hány DEAC-Hackers streamel?</p>
<div align="center">
Hajrá, <?php echo $hackername; ?>!
</div>
<div align="center">
<button onclick="refresh100 ()">FRISSÍT</button>
<button onclick="stop100 ()">LEÁLLÍT</button>
</div>
<div align="center">
<?php
$hackers = array(
"MatyiBatfai"=>" ",
"NorbertBatfai"=>" ",
"BotondL Lovasz"=>" ",
"LajosNagy"=>" ",
"RobiDekany"=>" "
);
foreach($hackers as $h => &$p)
{
    $hackerfile = "/tmp/".$h;
    $time = 0;
    if(file_exists($hackerfile))
        $time = file_get_contents($hackerfile);
    $p = $time;
}
arsort($hackers);
foreach($hackers as $h => $p)
{
    $hours = (int)($p / (1000*60*60));
    $minutes = (int)((($p - $hours*1000*60*60) / (1000*60));
    $seconds = (int)((($p - $hours*1000*60*60 - $minutes*1000*60) / 1000));
    echo "Hacker: " . $h . ", idő: " . $hours . " óra " . $minutes . " perc ←
          " . $seconds . " mp";
    echo "<br>";
}
?>
</div>
```

```
<script>
var limit = 1000*60*60*100;
var limit = 1000*60*60*100;
var refreshlimit = 1000*60*5;
var nofhackers = 1;
var time = <?php echo $hackertime; ?>;
var localtime = 0;
setInterval(function() {
time = time + nofhackers*1000;
localtime = localtime + 1000;
var hours = Math.floor(time / (1000*60*60));
var minutes = Math.floor((time - hours*1000*60*60) / (1000*60));
var seconds = Math.floor((time - hours*1000*60*60 - minutes*1000*60) / ←
1000);
document.getElementById("ido").innerHTML = hours + " óra "
+ minutes + " perc " + seconds + " mp ";
if(nofhackers==1)
document.getElementById("hanyan").innerHTML = nofhackers + " hacker ←
streamel";
else
document.getElementById("hanyan").innerHTML = nofhackers + " hacker ←
streamel párhuzamosan";
if (time >= limit) {
document.getElementById("ido").innerHTML = "A 100 óra stream teljesítve!";
}
if(localtime >= refreshlimit){
localtime = 0;
refresh100();
}
}, 1000);
function stop100() {
window.location.href = "stop.php?time=" + time;
}
function refresh100() {
window.location.href = "100.php?time=" + time;
}
</script>
</body>
</html>
```

Alapból csak tömbbejárást láthattunk a programban, ami nem rendezi idő szerint a hekkereket. A megfelelő helyeket átírva sikerült idő szerint rendezni az associatív tömböt.

```
<?php
$hackers = array(
"MatyiBatfai"=>" ",
"NorbertBatfai"=>" ",
"BotondLovasz"=>" ",
"LajosNagy"=>" ",
```

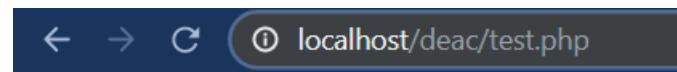
```
"RobiDekany"=>"
);
foreach($hackers as $h => &$p)
{
    $hackerfile = "/tmp/".$h;
    $time = 0;
    if(file_exists($hackerfile))
        $time = file_get_contents($hackerfile);
    $p = $time;
}
arsort($hackers);
foreach($hackers as $h => $p)
{
    $hours = (int)($p / (1000*60*60));
    $minutes = (int)((($p - $hours*1000*60*60) / (1000*60));
    $seconds = (int)((($p - $hours*1000*60*60 - $minutes*1000*60) / 1000));
    echo "Hacker: " . $h . ", idő: " . $hours . " óra " . $minutes . " perc ←
          " . $seconds . " mp";
    echo "<br>";
}
?>
```

A kiiratásba kellett kicsit átirni a kódot. Lényegében vannak a hekkerek egy assoc tömbben. Ez a tömb azért kell, hogy hozzárendelük a filerendszerben eltárolt fileokhoz az egyes hekkereket. Én úgy oldottam meg, hogy először a már meglévő assoc tömbbe érték párájába beletöltöttem a megfelelő time értékeket. Majd ezt az egészet rendeztem az arsort(\$hackers) függvényrel. Majd ahogy az eredeti példában is szépen kiirattam az adatokat. Csináltam egy példakódöt, ami szemlélteti kicsiben is. Később megcsinálom az egész programot, mert nem nehéz :)

```
<?php
$hackers = array(
    "MatyiBatfai"=>"",
    "NorbertBatfai"=>"",
    "DekanyRobert"=>"",
    "L LovaszBotond"=>"",
    "ValakiMali"=>"",
);
$a = 0;

foreach($hackers as $h => &$p)
{
    $p = 5000000 + $a;
    $a += 200000;
}
arsort($hackers);
foreach($hackers as $h => $p)
{
    $time = 0;
```

```
$time = $p;  
$hours = (int) ($time / (1000*60*60));  
$minutes = (int)((($time - $hours*1000*60*60) / (1000*60));  
$seconds = (int)((($time - $hours*1000*60*60 - $minutes*1000*60) / ←  
1000);  
echo "Hacker: " . $h . ", idő: " . $hours . " óra " . $minutes . " ←  
perc " . $seconds . " mp";  
echo "<br>";  
}  
?>
```



17.2. ábra. Tesztkód a hekkerek kiiratására.

18. fejezet

Helló, Valami!

18.1. Adatfolyamok kezelése, streamek. I/O állománykezelés. Seralizáció.

Az adatfolyamok vagy más néven streamek olyan objektumok, amelyek megnyit egy “utat” egy külső információforrás hoz (file, memória, socket). Ezen adatfolyam objektumok képesek írni, olvasni a forrásról. Két fő objektum fajtát különböztetünk meg: Karakter folyamok, amelyek 16 bites karakterekkel való műveleteket teszik lehetővé. Az olvasáshoz főképp a Reader osztályból származó FileReader osztályt használjuk. De ott van még a BufferedReader is. Íráshoz a Writer osztályból származó FileWriter osztályt alkalmazzuk. Ezek az osztályok bármely karaktert képesek Unicode karakterként kezelní. Másik objektum fajta a io műveletekhez a Bináris folyamok. Ezen folyamok főképp bináris adatfolyamok írásához, olvasásához használható. 8 bitenként írunk, olvasunk. Az egyes bináris io objektumok az InputStream és OutputStream ősosztályok leszármazottjai. A sorosítás során létrehozunk egy állományt és egy sorosítást kezelő objektumot. Ez gondoskodik arról, hogy az általunk kiválasztott objektumban tárolt adatok az állományba kerüljenek sorban egymás után. Láncolt lista és körkörös hivatkozások kezelésére is képes. A helyreállítás során megnyitjuk a korábban mentett adatokat tartalmazó állományt, és létrehozunk egy sorosítást kezelő objektumot. Ez gondoskodik a mentett adatok visszatöltéséről. Ha a mentés óta megváltoztattuk az adatok típusát megadó osztályt, akkor a visszatöltés nem lehetséges.

18.2. FUTURE tevékenység editor

Ebben a feladatban a FUTURE-höz létrehozott tevékenységeditoron kellett valamit változtatni. Maga a feladat nem lett volna nehéz viszont a javafx modulokkal sokat bajlódtam. Mint kiderült hiányzott a gépen lévő java11-ből a javafx így manuálisan kellett hozzáadni a compile-hoz. Ehhez egy Makefile-t használtam. Íme a shellscript:

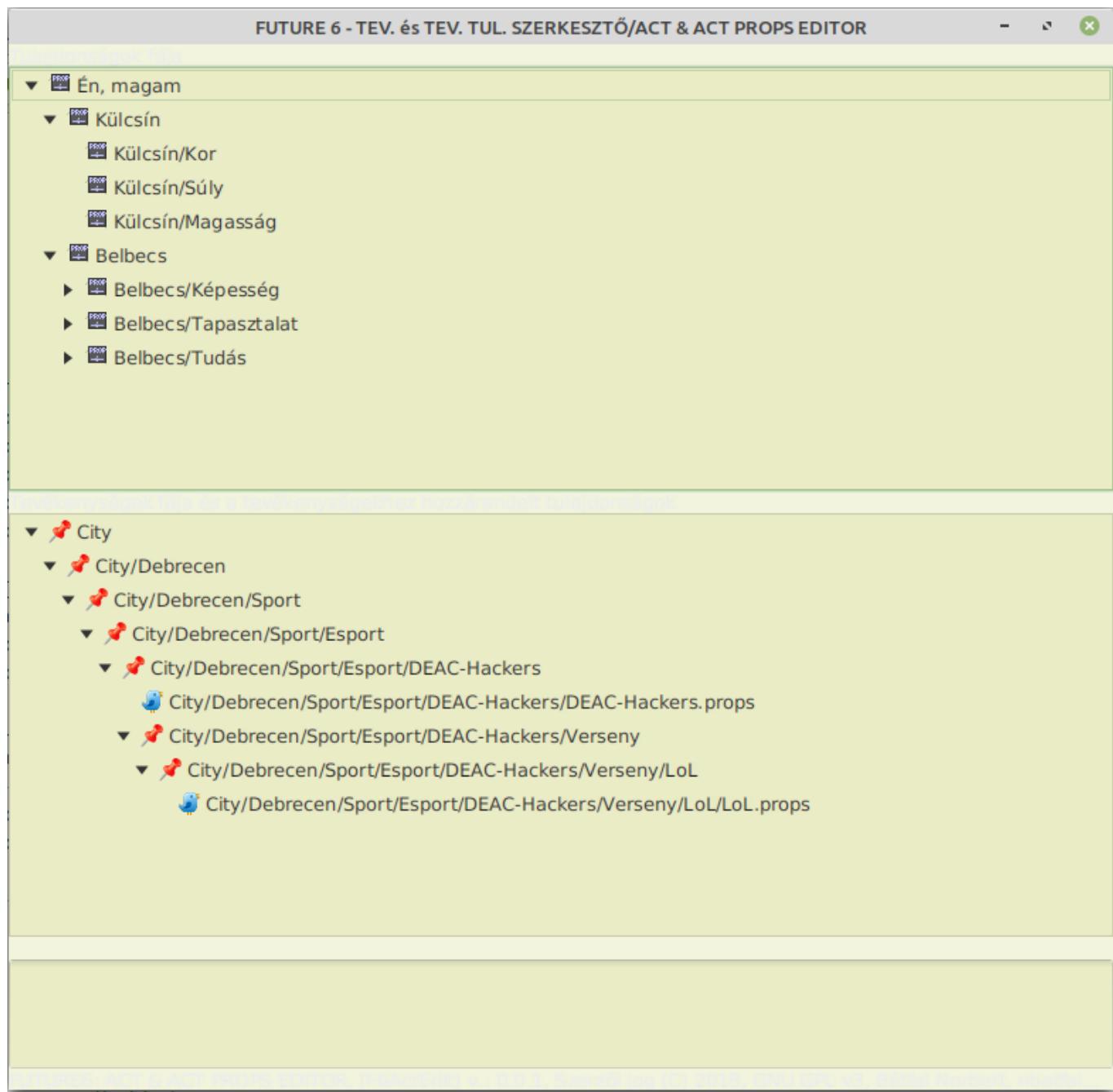
```
#!/usr/bin/env bash
export PATH_TO_FX="/home/drob/Letöltések/javafx-sdk-11.0.2/lib"
java_modules="javafx.controls,javafx.graphics,javafx.media,javafx.swing,←
    javafx.web"
rm -f *.class
```

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac \
--module-path $PATH_TO_FX \
--add-modules=$java_modules ActivityEditor.java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java \
--module-path $PATH_TO_FX \
--add-modules=$java_modules ActivityEditor
rm -f *.class
```

Ezt a `./run.sh` parancsal futtattam. Következőnek hozzáadtam egy css-file-t, hogy a kinézetét testre tudjam szabni. Ehhez a .java-ba az alábbi parancsot kellett addolni: `scene.getStylesheets().add("style.css")`. Ez hozzácsatolja a már létrehozott style.css fileomat a programhoz. A css file így néz ki:

```
.root {
    -fx-accent: #a3be8c;
    -fx-focus-color: -fx-accent;
    -fx-base: #eaec6;
    -fx-control-inner-background: -fx-base;
    -fx-control-inner-background-alt: -fx-base;
}
.label{ -fx-text-fill: #eceff4; }
.separator * .line { -fx-background-color: #4c566a;
    -fx-border-style: solid;
    -fx-border-width: 3px;
}
.scroll-bar{ -fx-background-color: -fx-accent; }
.button:default { -fx-base: -fx-accent; }
.table-view {
    -fx-background-color: derive(-fx-base, 10%);
    -fx-selection-bar-non-focused: derive(-fx-base, 85%);
}
.table-view .column-header .label{ -fx-font-weight: none; }
```

Illetve kicseréltem az iconokat, amit a végeredmény jól szemléltet.



18.1. ábra. Modified Activity Editor

18.3. OOCWC Boost ASIO hálózatkezelése

Ebben a feladatban egy OOCWC kódban kellett bemutatni a `sscanf` függvény szerepét. Mint a neve elárulja egy beolvasási mechanizmust hajt végre. Ez a függvény egy inputról (stringről) olvassa be az adatot majd formázottan eltárolja egy megadott elérésre. A nevében lévő `f` a formatted kulcsszóra utal. Ha nincs alaból elérési útvonal adva akkor a default outputra adja vissza a formázottan beolvasott adatot. Általános formája: `int sscanf (const char * s, const char * format, ...);`, ahol a

visszatérési érték az eltárolt elemek száma. Ha nem sikerült az adatot olvasni egy EOF-al tér vissza. Egy kódcsipet a példakódból:

```
%option c++
%option noyywrap

% {
#define YY_DECL int justine::robocar::CarLexer::yylex()
#include "carlexer.hpp"
#include <cstdio>
#include <limits>
% }

INIT  "<init"
INITG "<init guided"
WS    [ \t]*
WORD   [^-:\n \t()]{2,}
INT    [0123456789]+
FLOAT  [-.0123456789]+
ROUTE  "<route"
CAR    "<car"
POS    "<pos"
GANGSTERS "<gangsters"
STAT   "<stat"
DISP   "<disp>"
%%

...
{POS}{WS}{INT}{WS}{INT}{WS}{INT}
{
  std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
  m_cmd = 10001;
}

...
```

Mint látható egy olyan illesztést várunk, hogy egy POS-al kezdődik majd tabuláltan tagolva mögötte három integer sorakozik. A mi kis sscanf függvényünk egy yytext tárolóba tárolja a formázott értékeket. Ezután történik a regex kifejezés ami megmondja, hogy formázza az inputot. A string-be úgynevezett formatting escapes-eket (%d vagy %u) használunk. Ezzel megadjuk a beolvasott érték helyét a string-ben. Majd a move command-ot 10001-re állítjuk. A példakód többi részében és hasonlóan működik a sscanf függvény.

18.4. BrainB

Ezt a feladatot skippelem a PHP-s feladattal amit a 6. héten csináltam meg.

18.5. OSM térképre rajzolása

A feladat az volt, hogy készítsünk egy saját OSM ábrát. Feladatnak én egy GPS trackert választottam. Lényege, hogy a készített androidos applikáció mp-enként határozza meg az aktuális helyzetemet a Google Maps-en. A helyzetekre rendre markereket helyez el így kialakítva a márt járt utaimat. :)

Kezdésként Android Studioba létrehoztam egy új projektet azon belül egy Google Maps Activity-t. minden munkát megelőzően a Google Developer portálon generáltatni kellett egy saját Android SDK-hoz való API kulcsot. Ha ez megvan akkor a létrejött projektben a `google_maps_api.xml` fileba bemásoltuk az API keyt. Ez szükséges, hogy igénybe tudjuk venni a Google Maps Services-t. A program manifest állományban megadtam engedélyként a helyszolgáltatásokat, hiszen ez szolgáltatja majd az helyadatokat a készülékről.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.locationtracker">

    <!--
        The ACCESS_COARSE/FINE_LOCATION permissions are not required to ←
        use
        Google Maps Android API v2, but you must specify either coarse or ←
        fine
        location permissions for the 'MyLocation' functionality.
    -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" ←
        />
    <uses-permission android:name="android.permission. ←
        ACCESS_COARSE_LOCATION" />

    ...

```

A program logikája a `MapsActivity.java`-ba van megírva. A `LocationManager` objektum fogja meghívni a készülékünk Location Service-ét. `locationManager.requestLocationUpdates (LocationManager MIN_TIME, MIN_DIST, locationListener)`; metódus másodpercenként lekéri a helyzetünket és ha 2 méter távolságot mozdultunk akkor meghívja a `locationListener action-t`. Ez az action különböző metódusokat implementál. Nekiünk a legérdekesebb a `onLocationChanged (Location location)`. Ez minden egyes pozícióváltásnál kirajzol egy markert a helykoordinátákhoz és még a nézetet is odapozicionálja. Ezzel elértük, hogy minden egyes mozgást figyel az FBI és részletes helyzetjelentést készít. ;) A `MapsActivity` forrása:

```
package com.example.locationtracker;

import androidx.core.app.ActivityCompat;
import androidx.fragment.app.FragmentActivity;

import android.Manifest;
import android.app.Activity;
import android.content.pm.PackageManager;
```

```
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    private LocationListener locationListener;

    private LocationManager locationManager;

    private final long MIN_TIME = 1000; // 1 sec
    private final long MIN_DIST = 3; // 1 meter

    private LatLng latLng;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, PackageManager.PERMISSION_GRANTED);
        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_COARSE_LOCATION}, PackageManager.PERMISSION_GRANTED);
    }

    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
    
```

```
* This is where we can add markers or lines, add listeners or move the ←
    camera. In this case,
* we just add a marker near Sydney, Australia.
* If Google Play services is not installed on the device, the user ←
    will be prompted to install
* it inside the SupportMapFragment. This method will only be triggered ←
    once the user has
* installed Google Play services and returned to the app.
*/
@Override
public void onMapReady(GoogleMap googleMap) {

    mMap = googleMap;

    locationListener = new LocationListener() {

        @Override
        public void onLocationChanged(Location location) {

            try
            {
                LatLng = new LatLng(location.getLatitude(), location. ←
                    getLongitude());
                mMap.addMarker(new MarkerOptions().position(latLng). ←
                    title("Robi"));
                mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
            }
            catch (SecurityException e)
            {
                e.printStackTrace();
            }
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle ←
            extras)
        {

        }

        @Override
        public void onProviderEnabled(String provider)
        {

        }

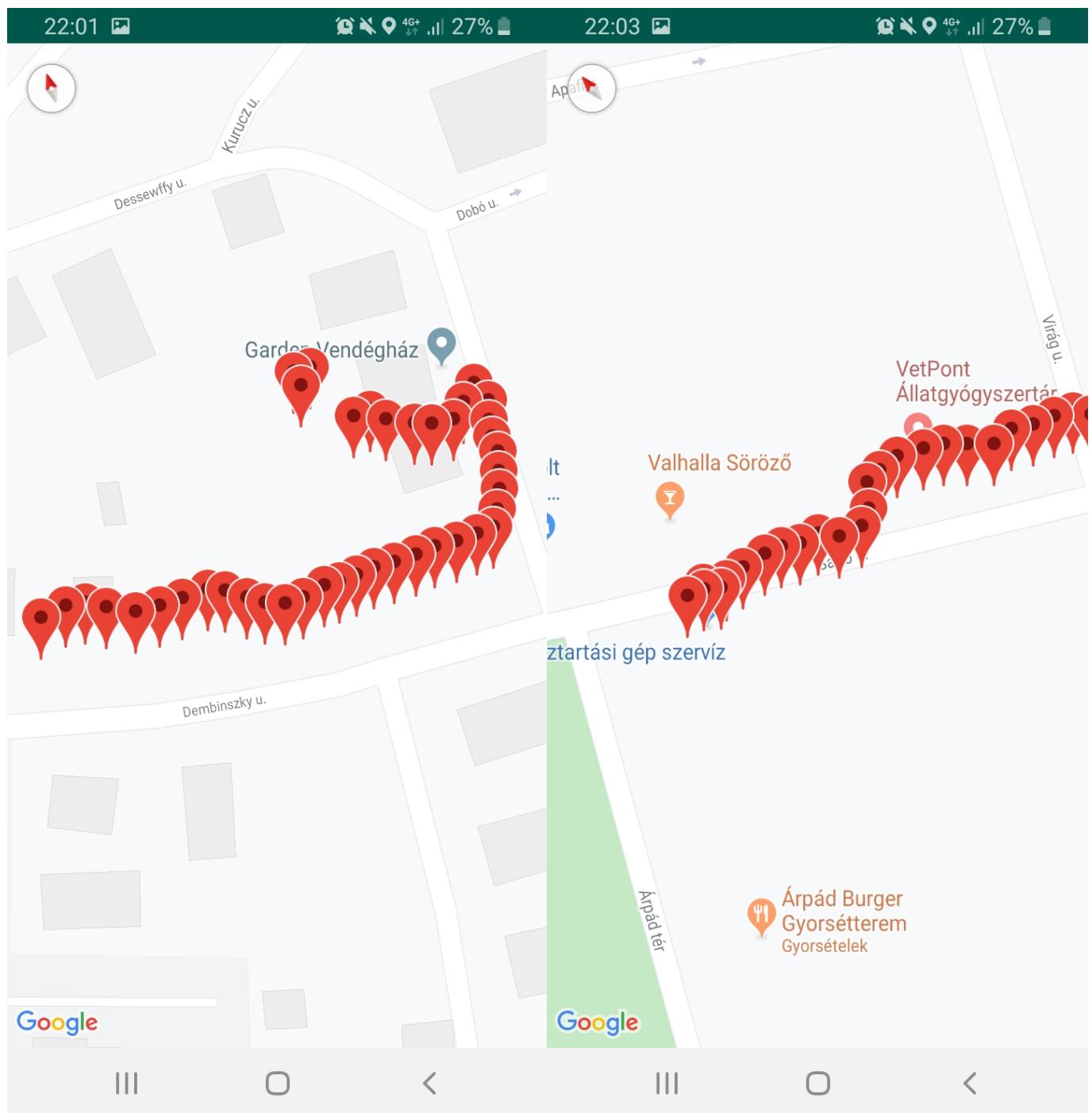
        @Override
        public void onProviderDisabled(String provider)
        {

    };
}
```

```
locationManager = (LocationManager) getSystemService(←
    LOCATION_SERVICE);

try
{
    locationManager.requestLocationUpdates(LocationManager.←
        GPS_PROVIDER, MIN_TIME, MIN_DIST, locationListener);
}
catch (SecurityException e)
{
    e.printStackTrace();
}

}
```



18.2. ábra. GPS Tracker Android Application

19. fejezet

Helló, Lauda!

19.1. Kivételkezelés. A fordítást és a kódgenerálást támogató nyelvi elemek (annotációk, attribútumok).

A kivételkezelés a nemvárt hibák detektálására használatos módszer. A nemvárt hibáknak többféle fajtája lehet. Ilyen pl.: túlindexelés, hibás fájl stb. Ezeket muszáj módszeresen kezelni, hiszen senki se akarja ha egy szép hibás file exceptionnel elszállna a kedves ügyfél programja. Szintaktikája a try-catch illetve ehhez jöhet finally. A try blokkban helyezkedik el a kritikus kódunk, amely hibát dobhat. A catch(Exception e) blokkban kezeljük le a bekövetkezett hibát. Az Exception minden kivétel ősosztálya. A finally a try utáni vezérléssel lefut. Ezt használhatjuk a be nem zárt adatfolyamok lezárására. Lehetőségünk van saját kivételest írni, ezt mindenkor az ősosztályból kell származtatni. Illetve dobhatunk manuálisan kivételt a throw kulcsszóval. A Java annotációk olyan metaadatot tartalmaznak amelyeket a java forráskódokhoz rendeljük. Az annotációk számos fontos adatot tartalmazhatnak. Ezek fordítónak fontos információk, amelyek befolyásolják a működést és hibakeresést. Az annotációkról egy egész hosszú lista áll rendelkezésre így ezeket érdemes elolvasgatni. Szintaktikája a @ (kukac) amely után az annotáció lefoglalt kulcsszava áll. Pl.: @Override, @SuppressWarning

19.2. Portscan

Ebben a feladatban a forrásként megadott port szkennelő kódban kellett rámutatni a kivételkezelés szerepére. A program lényege, hogy parancssori argumentumként megadott host-on végignézzük 1023 ig a portokat és ha valamelyik port szabad akkor egy új stream socket objektumot hoz létre arra a hostra és portra. A program forráskódja:

```
public class KapuSzkenner {  
  
    public static void main(String[] args) {  
  
        for(int i=0; i<1024; ++i)  
  
            try {
```

```
java.net.Socket socket = new java.net.Socket(args[0], i);

System.out.println(i + " figyeli");

socket.close();

} catch (Exception e) {

    System.out.println(i + " nem figyeli");

}

}

}
```

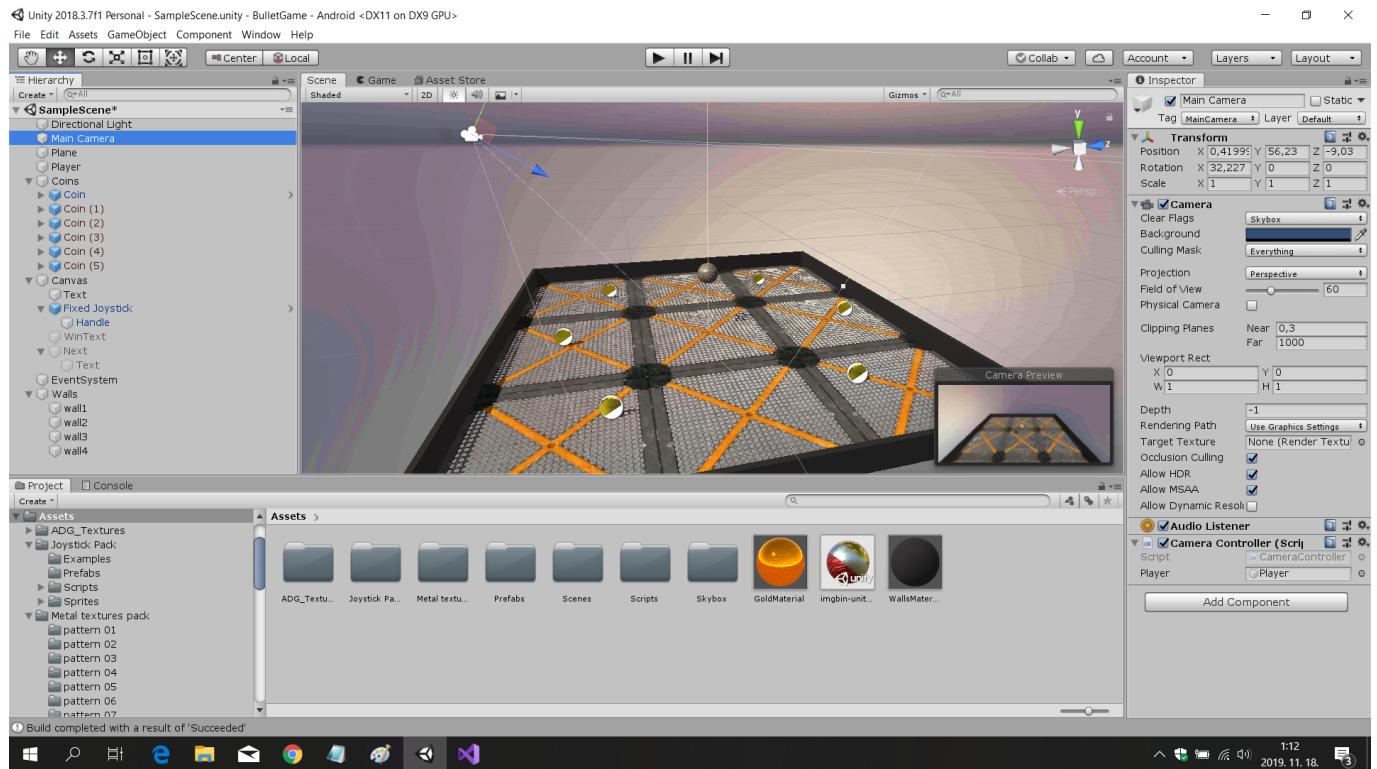
A kivételkezelés fontossága abban rejlik, hogy mikor egy új socket objektumot létrehozásakor számos hiba léphet fel. Ezeket a hibákat megfelelően le kell kezelni hisz nem szeretnénk ha csak úgy kicrasheljen a programunk. Hibákat kivételelosztályok írják le. A mi Socket objektumunk 4 féle exceptionnel térhet vissza. UnknownHostException: Akkor dobja ha a host nincs meghatározva az adott IP címre. IOException: akkor dobja ha nem sikerült létrehoznia az socket objektumot. SecurityException: akkor dobja ha a Socket.checkConnect metódus nem engedélyez különböző műveleteket. IllegalArgumentException: akkor dobja ha portszámkként a meghatározott 0-65535 tartományon kívülre eső portszámot definiálunk. Ezek a különböző kivételelosztályok az Exception űzosztály leszármazottjai tehát a programunk catch ágába éppen az aktuális gyermekosztály "kapódik" el.

```
620 nem figyeli
621 nem figyeli
622 nem figyeli
623 nem figyeli
624 nem figyeli
625 nem figyeli
626 nem figyeli
627 nem figyeli
628 nem figyeli
629 nem figyeli
630 nem figyeli
631 figyeli
632 nem figyeli
633 nem figyeli
634 nem figyeli
635 nem figyeli
636 nem figyeli
637 nem figyeli
638 nem figyeli
639 nem figyeli
á640 nem figyeliba léphet fel.
j641 nem figyeli. Hibákat
l642 nem figyelition: Akkor
s643 nem figyeli az az socket
l644 nem figyeli a műveleteket
```

19.1. ábra. Portscan Eredmény localhostra.

19.3. Androidos játék

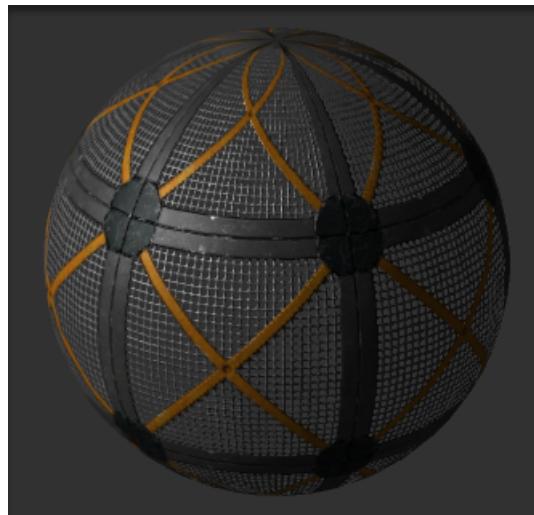
Ebben a feladatban egy androidos játékot kellet készteni. Ez a feladatot nagyon élveztem, mert én magas is érdeklődök a 2D/3D játékfejlesztés irányt. Illetve van pár hobbiprojekt, amit meg szeretném valósítani. (typic programmer things :D) Na de vágunk is bele. Ezt a feladatot Unity Game Engine segítségével készítettem el. Játékként egy labdás játékot hoztam el, amit egy kurzor segítségével irányíthatunk. Fel kell szedni 15 coint. Ha megvan, akkor egy Gratulálok, nyertél felirat jelentik meg illetve egy folytatás gomb, ha szeretnénk tovább játszani. Egy kis preview, hogy hogyan is nézett ki a szerkesztői felület:



19.2. ábra. Unity Editor

Bal oldalon találhatóak a játékot alkotó egyes objektumok (pl.: Player, Main Camera stb.). Középen helyezkedik el a preview/game nézet. Ebben lehet szerkeszteni az egyes objektumokat, nézetet váltani, jelenlegi játék preview-t megtekinteni stb. Alul találhatóak az Asset-ek. Ezen mappastruktúrában tároljuk az egyes esszenciális elemeket. Materialok, Modellek illetve Scriptek. Ezek jelentőségét később fejtem ki. Végül jobb oldalon található a Property ablak, ahol az egyes objektumok tulajdonságait találjuk/állíthatjuk be.

Az Asseten belül szoktuk elhelyezni a Materialokat (.mat) Ezek lényegében stílusdefiníciók, amelyek több tulajdonságból tevődnek össze. Drag/drop-al egyszerűen beállíthatjuk az egyes objektumot kinézetét ezekkel. A programhoz használtam, minden saját mind Asset Store-ból szedett Materialokat. Plane objektumra alkalmazott material nézete:



19.3. ábra. Plane objektumra alkalmazott rácsos material.

A játék fő logikáját a c# scriptek adják. Ezeket konvenció szerint az Asset-ekhez soroljuk. A felvehető pénzérmék forgásáért fele az alábbi script:

```
&#xA0;<using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rotator : MonoBehaviour
{
    // Update is called once per frame
    void Update()
    {
        transform.Rotate(new Vector3(15, 30, 45) * Time.deltaTime);
    }
}
```

Ez a script a Coin-ok van rendelve. Az Update egy speciális metódus, amely a játék során minden egyes frameupdatekor lefut. Ebben egy forgató tranzformációt alkalmazok egy adott 3d térfélen pozícióra a deltatime függvényben. Ezzel fainak forognak a coinok, szuper!

Következőben azt szerettem volna megoldalni, hogy a kamera kövesse a golyó mozgását. Sikerült is íme a kód:

```
&#xA0;<using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    public GameObject player;
```

```
private Vector3 offset;

// Start is called before the first frame update
void Start()
{
    offset = transform.position - player.transform.position;
}

// Update is called once per frame
void LateUpdate()
{
    transform.position = player.transform.position + offset;
}
```

A player objektum és a kamera távolságának különbsége adja az offsetet, amely minden egyes frameupdate-t után hozzáadódik a player távolságához így követve a mozgását.

Player osztály fogja megvalósítani minden funkcionálitást, ami a karakterünket érinti. A OnTriggerEnter (OnTriggerEnter other) függvény fogja lekezelni azt, ha a karakterünk érti az egyik coint. Először megvizsgáljuk, hogy a pontszámunk kisebb-e mint 15, ha igen akkor a pontszámunkat növeljük ha viszont elértek a 15 pontot akor deaktiváljuk a vezérlést és megjelenítjük a győztél feliratot illetve a gombot.

```
&#xA0;<using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

public class Player : MonoBehaviour
{
    public float speed;

    public float jumpForce;

    public Text pointText;

    public Joystick js;

    public Text winText;

    public Button btn;

    private bool canJump = true;

    private Rigidbody rb;

    private int points;
```

```
// Start is called before the first frame update
void Start()
{
    rb = GetComponent<Rigidbody>();

    points = 0;

    SetCountText();
}

// Update is called once per frame
void Update()
{
    if (transform.position.y < 46)
    {
        transform.position = new Vector3(0, 50, 0);
    }
}
private void FixedUpdate()
{
    float moveHorizontal = js.Horizontal;
    float moveVertical = js.Vertical;

    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

    rb.AddForce(movement * speed);
}

private void OnCollisionEnter(Collision collision)
{
    canJump = true;
}

private IEnumerator LateCall(Collider other)
{
    yield return new WaitForSeconds(5f);

    other.gameObject.SetActive(true);
}

private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Coin"))
    {
        other.gameObject.SetActive(false);
        points++;
        SetCountText();

        if (points == 15)
```

```
        {
            winText.gameObject.SetActive(true);
            btn.gameObject.SetActive(true);
            js.gameObject.SetActive(false);
        }

        StartCoroutine(LateCall(other));
    }
}

public void SetCountText()
{
    pointText.text = "Pontok: " + points.ToString();
}
}
```

A Joystick osztály fog felelni a joystick megjelenítéséért illetve a vezérlésért. Ezt a fajta vezérlést fogjuk alkalmazni a golyó mozgatására. A moveCharacter(Vector2d direction) metódus a paraméterében megkaptott joystick pozíciót felhasználva fogjuk mozgatni a karaktert: player.Translate(direction * speed * Time.deltaTime);

```
&#xA0;using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class JoyStick : MonoBehaviour
{
    public Transform player;
    public float speed = 5.0f;
    private bool touchStart = false;
    private Vector2 pointA;
    private Vector2 pointB;

    public Transform circle;
    public Transform outerCircle;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            pointA = Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, Camera.main.transform.position.z));

            circle.transform.position = pointA * -1;
            outerCircle.transform.position = pointA * -1;
            circle.GetComponent<SpriteRenderer>().enabled = true;
        }
    }
}
```

```
        outerCircle.GetComponent<SpriteRenderer>().enabled = true;
    }
    if (Input.GetMouseButton(0))
    {
        touchStart = true;
        pointB = Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, Camera.main.transform.position.z));
    }
    else
    {
        touchStart = false;
    }

}
private void FixedUpdate()
{
    if (touchStart)
    {
        Vector2 offset = pointB - pointA;
        Vector2 direction = Vector2.ClampMagnitude(offset, 1.0f);
        moveCharacter(direction * -1);

        circle.transform.position = new Vector2(pointA.x + direction.x, pointA.y + direction.y) * -1;
    }
    else
    {
        circle.GetComponent<SpriteRenderer>().enabled = false;
        outerCircle.GetComponent<SpriteRenderer>().enabled = false;
    }
}

void moveCharacter(Vector2 direction)
{
    player.Translate(direction * speed * Time.deltaTime);
}
```

A ButtonClick osztály foglalkozik majd a "Nyertél" gomb funkcionálisával. A Start() függvényben a tempBtn objektum fogja tartalmazni a gomb referenciáját. A gomb lenyomásakor egy action fog lefutni. Erre az action-re feliratkozunk: tempBtn.onClick.AddListener(TaskOnClick). Ha rágattintunk a gombra a TaskOnClick() metódus fog lefutni. Ebbe a label,gomb eltünése és a joystick megjelenítése van megírva.

```
&#xA0;&#xA0;using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.UI;

public class ButtonClick : MonoBehaviour
{
    public Button myBtn;

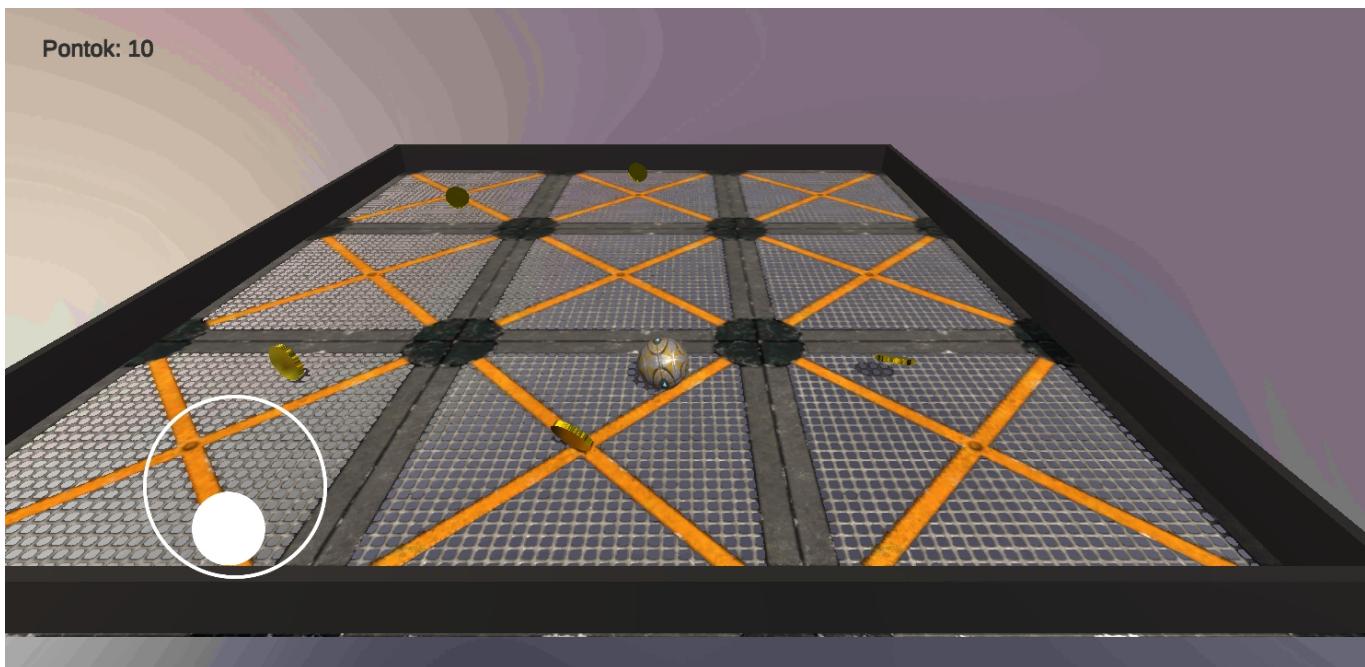
    public Text winText;

    public Joystick js;

    // Start is called before the first frame update
    void Start()
    {
        Button tempBtn = myBtn.GetComponent<Button>();
        tempBtn.onClick.AddListener(TaskOnClick);
    }

    void TaskOnClick()
    {
        winText.gameObject.SetActive(false);
        myBtn.gameObject.SetActive(false);
        js.gameObject.SetActive(true);
    }
}
```

...



19.4. ábra. Kész Androidos játék.

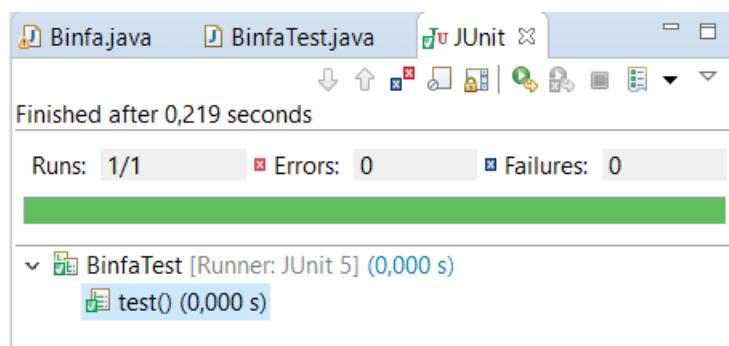
19.4. JUnit Teszt

Ebben a feladatban tesztelést kellett végezni a java Binfa projekten. Az Eclipse-ben ez könnyen megvalósítható volt. Létrehozásnál megadtunk egy osztálynevet, amelybe írjuk majd a tesztünket. A teljes teszt így néz ki:

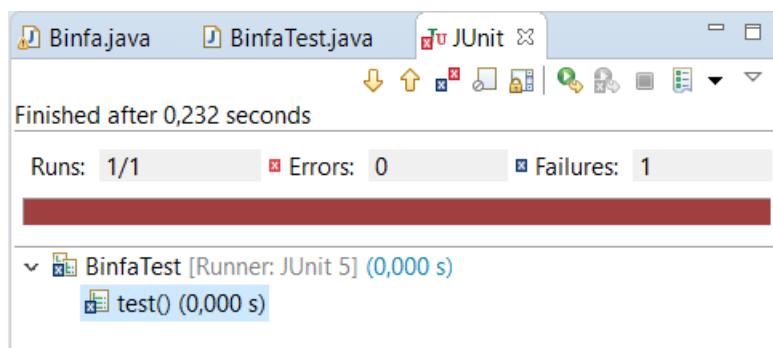
```
&#xA0;&#xA0;import static org.junit.jupiter.api.Assertions.*;  
  
import org.junit.jupiter.api.Test;  
  
class BinfaTest  
{  
  
    @Test  
    void test()  
    {  
        char[] mockData = "0101010101110111010110101111".toCharArray();  
  
        Binfa binfa = new Binfa();  
  
        for(char c : mockData)  
        {  
            binfa.egyBitFeldolg(c);  
        }  
    }  
}
```

```
// tesztelés
assertEquals(4, binfa.getMelyseg());
assertEquals(3.25, binfa.getAtlag());
assertEquals(0.5, binfa.getSzoras());
}
}
```

A tesztmetódusunkat egy @Teszt annotációval láttuk el. Ezután megadtuk a Binfa tesztinputját, ami szolgáltatja az adatokat. A binfa objektumba szépen felépítük a fát a tesztadatainkkal, majd az assertEquals statikus metódussal végrehajtuk a tesztet. Ez két paramétert vár. Az első a valid érték, amit a második paraméterbe írt függvény kell, hogy visszaadjon. Ha hasonló értéket kap akkor a teszt sikeres viszont ha különböző az érték a teszt megbukott. Íme egy-egy példa:



19.5. ábra. Sikeres teszt.



19.6. ábra. Sikertelen teszt.

19.5. AOP

A feladat az Aspektusorientált programozásra ad egy példát. Az AOP egy magasabb szintű absztrakciót megvalító paradigma. Lényege, hogy a feladatokat aspektusonként kezeljük. Az aspektusokat aspektusnyelvek írják le amely megegyezhet a komponensnyelvvel. A komponensnyelv jelen esetben a java. Ha egy kód részletet kívánunk beleszóni vagy épp elvenni a programba. A beszúrandó kód részletet a megfelelő

helyre szúrjuk be. Ezeket a helyeket csomópontoknak vagy más néven join pointoknak nevezik. A feladatot AspectJ-ben készítjük el. Ennek a támogatottsága már megszűnt ezért vissza kellett állítanom a jdk-t Java 8-ra. Packageket is újra kellett tenni ezután már gondnálkül tudtam használni az AspectJ-t. Fordítás-ként az ajc compilert alkalmazzuk. Két külön file-ra lesz szükségünk az egyik a komponens (.java) ami a Binfa.java a másik pedig az aspektj file (.aj). Elsőnek az utóbbit írjuk meg:

```
privileged aspect BinfaAspect
{
    void around(Binfa fa, Binfa.Csomopont elem, BufferedWriter os):
        call(public void Binfa.kiir(Binfa.Csomopont, BufferedWriter)) && target( ←
            fa) && args(elem, os)
    {
        if (elem != null)
        {
            try
            {
                ++fa.melyseg;
                for (int i = 0; i < fa.melyseg; ++i)
                    os.write("----");
                os.write(elem.getBetu () + " (" + (fa.melyseg - 1) + ") \n");

                fa.kiir(elem.egyesGyermek (), os);
                fa.kiir(elem.nullasGyermek (), os);
                --fa.melyseg;
            }
            catch (Exception e)
            {
                System.out.err(e.getMessage());
            }
        }
    }
}
```

Az around függvény helyettesítés végez. A helyettesítésképp a call függvénytörzsében megírt kódrészlet fog fordulni. A call függvény paramétereiben megadott target a minden fa típusra illeszkedni fog. Az args pedig a megadott két argumentumú metódusra lefordul egyéb esetben nem tudjuk, hogy melyikre kéne hasonlítani. Ebből szépen látszik a "nézőpont" szerinti programozás lehetőségei. Fordítás: ajc Binfa.java BinfaAspect.aj Futtatása: java -classpath ./aspectjrt.jar:. Binfa.java input.txt -o output.txt A végeredményen látszik, hogy megváltozott a fabejárás, pont ahogyan várható volt. Eredmény:

```
atrop@atrop-VirtualBox:~/DOKUMENTUMOK/java/aspect$ cat output
---/(0)
-----1(1)
-----1(2)
-----1(3)
-----0(2)
-----0(3)
-----0(4)
-----0(1)
-----1(2)
-----0(2)
depth = 4
mean = 2.75
var = 0.9574271077563381
```

19.7. ábra. Aspect output.txt

20. fejezet

Helló, Calvin!

20.1. Multiparadigmás nyelvek. Programozás multiparadigmás nyelveken.

A programozási nyelveket főképp a nyelv által használt paradigma szerint osztályozzuk. Ilyen pl.: A Java egy magasszintű objektum-orientált programozási nyelv. Sokféle paradigmát különböztetünk meg és párról már említést is tettünk a könyv írása során. A nyelv által használt paradigma meghatározza a nyelv funkcionálitását és elemeit. A multiparadigmás nyelvek több paradigma használnak illetve ezek határozzák meg a funkcióit. Ezek többrétegű feladatok megoldására alkalmasak, amelyek alapból több paradigma keveredésével tevődik össze. A programozó az adott feladat megoldásához több paradigma közül válogathat így elmosva az egyes paradigmák között elhúzódó határokat.

20.2. MNIST

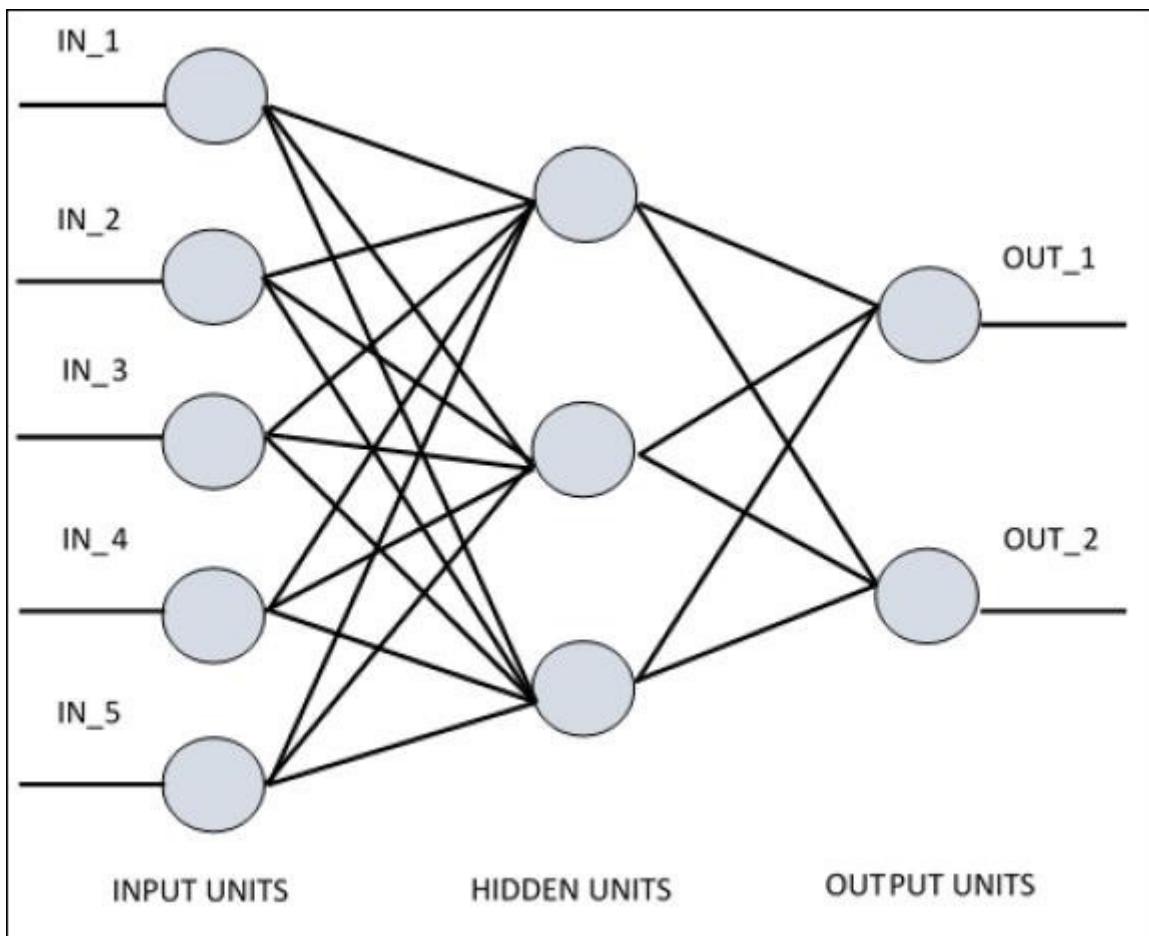
Ebben a csokorban a gépi tanulás világával fogunk megismerkedni. Elsőként az MNIST-el. Ez egy hatalmas dataset amelyben 0-9 -ig terjedő kézzel írott számjegyek vannak. 28x28 px méretű képek szürkeárnyalatos kent vannak ábrázolva. 60.000 training- és 10.000 tanító képet tartalmaz. Ezeket tipikusan a gépi tanulás számos területén alkalmazzák.



20.1. ábra. MNIST dataset.

MNIST-et felhasználva készítsünk el egy olyan programot, amely saját írott számunkat képes felismerni. Ezt a Tensorflow open-source AI könyvtár, amely segít megvalósítani az ehhez a feladathoz szükséges komponenseket. A Tensorflow esszenciális részét képezi a python megléte. így ha még nem tettük meg a szükséges függőségek telepítését, tegyük meg! Ehhez nagy segítséget nyújt a Tensorflow hivatalos oldala. A kódok készítésére és szerkesztésére a Jupyter alkalmazást használtam, amely elég elterjedt az egyes gépi-tanulásos feladatra.

Egy hagyományos többrétegű neurális háló egy bemeneti(input), egy kimeneti (output) és rejtett (hidden) rétegekből állnak össze. Az egyes rétegek között teljes a kapcsolat, amely azt jelenti, hogy minden pont, minden ponttal össze van kötve. Az egyes bemenetek különböző súlyokkal vesznek részt a kimenet meghatározásában. Ezután a súlyokat egy aktivációs függvényre alkalmazva 0-1 közötti lebegőpontos számot kapunk eredményül. Ezek fontos szerepet kapnak a tanítási-vég fázisban.



20.2. ábra. NN

Ezeket a modellek Perceptron modellnek nevezik. Ezekről már korábban is hallottunk, de így közelebb került mindenkihez a fogalom. Ezen háttér birtokában felépítjük a kis MNIST python kódunkat. Elsőként a szükséges tensorflow és a MNIST data_set-et kellett beimportálni. Helyenként deprecated volt így a TF v1. régi verzióval készült a példa. A data-set-et lokálisan kapta meg. Ezután kell egy olyan prepare() függvény, amely a saját képünket lekezeli is megfelelő formában visszaszolgáltatja a programunknak. Itt a cv2.imread végzi a beolvasást. Ezután, ha a kép nem 28x28-ba érkezik akkor átméretezzük. Ezután elkészítjük a szükséges normalizált mátrixot a megadott [?, 784] dimenzióra.

```
def prepare():
    filepath = "C:/Users/teaisawesome/Pictures/mnist/nulla.png";
    IMG_SIZE = 28
    img = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = img.astype('float32')
    img = img.reshape(28*28)
    img = 255-img
    img /= 255
    return img
```

A main függvénybe az mnist változó kapja meg a MNIST data-set-et. Ezután több különböző deklaráció található meg. Az x egy placeholder, amely majd a saját képünket fogja megkapni értékként. A W tartalmazza a súlyokat. A b a valószínűséget hivatott megadni. Ezután következik a tanulási fázis a Softmax és a GradientDescent optimalizációs függvény segítségével.

```
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(←
    logits=y, labels=y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(←
    cross_entropy)
```

A hálózat tanítása 1000x100-assával történik minden egyes tesztadatra.

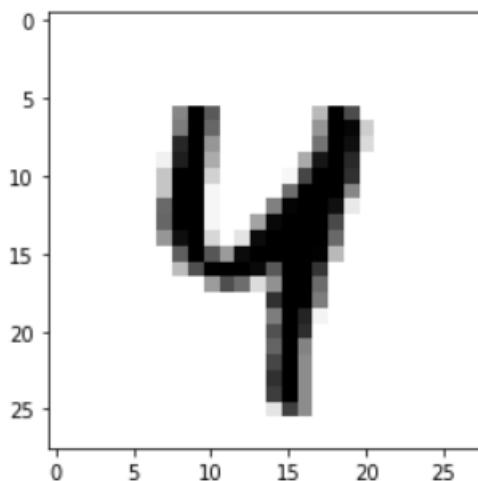
```
print("-- A halozat tanitasa")
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
        print(i/10, "%")
```

Ezután kíváncsian várjuk, hogy milyen lett pontosan lett tanítva a hálózatunk. Ezt a accuracy változóba tároljuk a reduce_mean(...) segítségével.

```
-- A hálózat tanítása
0.0 %
10.0 %
20.0 %
30.0 %
40.0 %
50.0 %
60.0 %
70.0 %
80.0 %
90.0 %

-----
-- A hálózat tesztelése
-- Pontosság: 0.9191

-----
-- A MNIST 42. tesztképenek felismereése, mutatom a számot:
```



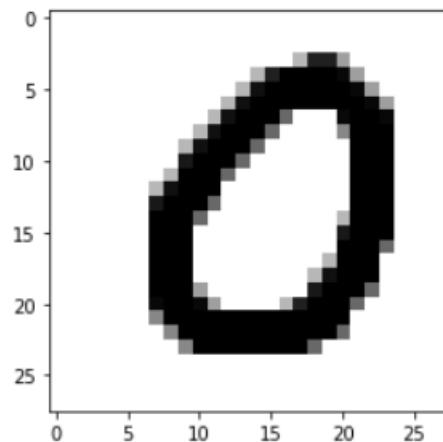
```
-- Ezt a hálózat ennek ismeri fel: 4
```

20.3. ábra. Hálózat tanítása és tesztelése.

Mostmár kész vagyunk a modellel, be is tanítottuk így következhet a számfelismerés varázsa. Egy saját rajzolt 0-ast készítettem erre a célra, ezt fogja bemenetként kapni a program és megpróbálja felismerni a számot a betanított modellünk alapján. Először felkészítjük a képet a már előre megírt prepare() függvényünkkel. Ezt kirajzolatjuk az imshow() matplot függvénygel. Ezután a classification változóba "megjósoltatjuk", hogy éppen melyik számjegyre adja vissza a maximumot. Ez egy objektum, amely tartalmazza a max indexet, amely a jóslott számjegyet jelenti. Ezt kiprinteljük az output-ra és tá'dá' sikeres.

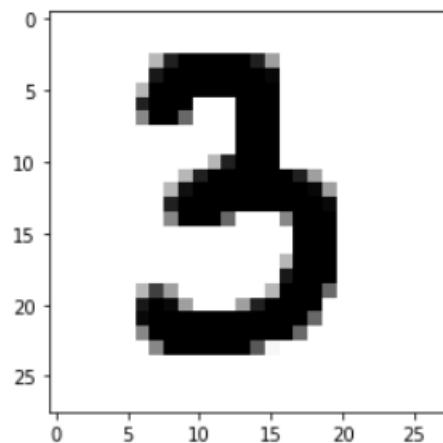
```
-- Ezt a halozat ennek ismeri fel:  4
```

```
-- A saját kezi 0-asom felismerese, mutatom a szamot:
```



```
-- Ezt a halozat ennek ismeri fel:  0
```

```
-- A saját kezi 3-asom felismerese, mutatom a szamot:
```



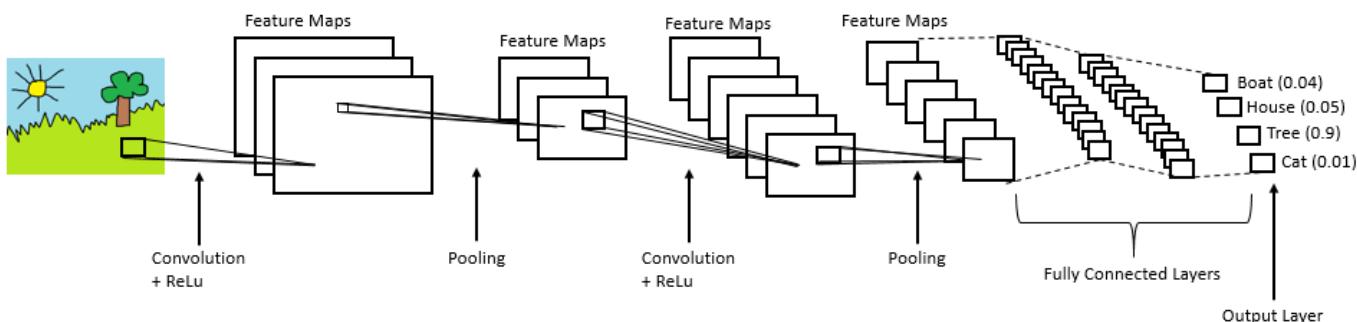
```
-- Ezt a halozat ennek ismeri fel:  3
```

20.4. ábra. Felismert számok.

20.3. DEEP MNIST

Ezen a példában a CNN-el vagyis Konvolúciós Neurális Hálózattal fogunk dolgozni. A konvolúciós hálókkal számos képfelismerési-, objektum detektáló- stb. feladat megvalósítható. Egy ilyen hálózat több

layerből tevődik össze. A rétegek között nincs teljes kapcsolat hanem az előző réteg részhalmazt szolgáltat. Egy konvoluciós ablak segít végigmérni a bemeneti részhalmazokat. Ez az ábrán jól látható. Ezen kívül fontos a Pooling layer, amely térbeli összevonást alkalmaz így csökken a dimenzió. Emiatt már mélyebb CNN layer is alkalmazható rá. Van Max-, Average-, Sum Pooling. Többnyire a Max-et használjuk. ReLu layer egy aktivációs réteg, amely segít 0-1 közötti non-linear értékek kiszámolását. Ez lényegében egy sigmoid. Fully Connected layer egy hagyományos neurális hálózat, amelynek minden neuronja hozzá van kötve a következő réteg további neuronjaihoz. Ez egy modellt képez amely után egy sigmoid vagy softmax aktivációs függvény van. Mindezeket összefoglalva láthatjuk ezen az ábrán:



20.5. ábra. Convolutional Neural Network (CNN) ábra

Most nézzük meg a python kódot illetve elemezgessük kicsit.

Már előzőleg be importáltuk a MNIST tesztadatokat az import input_data-val úgy ahogy az előző példában is. Ezután egy előfeldolgozó prepare() függvényt definiálunk, amely a képünkbeli elkészíti a megfelelő vektort. Ez is hasonló az előző feladathoz.

```
def prepare():
    filepath = "C:/Users/teaisawesome/Pictures/mnist/nulla.png";
    IMG_SIZE = 28
    img = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = img.astype('float32')
    img = img.reshape(28*28)
    img = 255-img
    img /= 255
    return img
```

A továbbiakban a CNN hálózat modellhez több különböző réteget fogunk megadni. Először definiálunk egy deepnn(x) függvényt, amely leírja a modellünk struktúráját. Mint ahogy írtam több különböző layert határozunk meg ezek a következők: Reshape layer: Az inputot átformálja a [-1, 28, 28, 1] dimenzióra illetve a grayscale konverziót.

```
def deepnn(x):
    with tf.name_scope('reshape'):
```

```
x_image = tf.reshape(x, [-1, 28, 28, 1])
```

Convolution 1 layer: Ez az első konvolúciós réteg, amely az inputra kiszámolja a skalárszorzatot (konvolúciót). Ezt a súlyokkal kiszámoljuk a 32-es activation map-et. Feladata a jellemzők kiemelése és számítása. Továbbá itt használjuk a ReLu aktivációs függvényt, amely a (0,1) közé igazítást végez.

```
with tf.name_scope('conv1'):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
```

Pooling 1 layer: Csökkenti a reprezentáció méretét 2x szeresére. Max Pooling-ot hajtunk végre.

```
with tf.name_scope('pool1'):
    h_pool1 = max_pool_2x2(h_conv1)
```

Convolution 2 layer: Második konvolúciós réteg. Itt már a 32-es activation map-et kiszámoljuk 64-re.

```
with tf.name_scope('conv2'):
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
```

Pooling 2 layer: ismét csökkenti a méretet. Max Pooling-ot hajtunk végre.

```
with tf.name_scope('pool2'):
    h_pool2 = max_pool_2x2(h_conv2)
```

Fully Connected 1 layer: minden input össze van kötve a réteg teljes neuron hálózatával, lényegében egy klasszikus NN. A 28x28-as képeket tovább alakítjuk [7,7,64, 1024] map alapján [1024]-es dimenzióra.

```
with tf.name_scope('fc1'):
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

Dropout layer: Lényege, hogy a tanuló fázisban random neuronokat nem vesz figyelembe. Ez az úgynevezett “overfitting” probléma megelőzésére szolgál.

```
with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Fully Connected 2 layer: A beérkező inputra ([1024] map) elkészíti az osztályozást. 0-9 számjegyekre.

```
with tf.name_scope('fc2'):
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])

    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
return y_conv, keep_prob
```

Ezzel elkészült a modell szerkezete. Most megnézzük, hogy a main többi elemét. Az mnist változóba importáljuk a már előre behúzott MNIST adatokat. Az x változóba lesz elkészítve a model. Az y változóba a loss és optimalizálót definiáljuk majd. Ezután a már előre megírt deepn(x) függvény segítségével szépen felépítjük a gráfokat. Láthatjuk, hogy létrehozunk még különöző loss és optimizer függvényeket. Jelen esetbe Adam optimizert használjuk, hogy gyorsítjuk a tanulás folyamatát.

```
def main(_):
    # Import data
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])

    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])

    # Build the graph for the deep net
    y_conv, keep_prob = deepnn(x)

    with tf.name_scope('loss'):
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                               logits=y_conv)
    cross_entropy = tf.reduce_mean(cross_entropy)

    with tf.name_scope('adam_optimizer'):
        train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
        correct_prediction = tf.cast(correct_prediction, tf.float32)
    accuracy = tf.reduce_mean(correct_prediction)
```

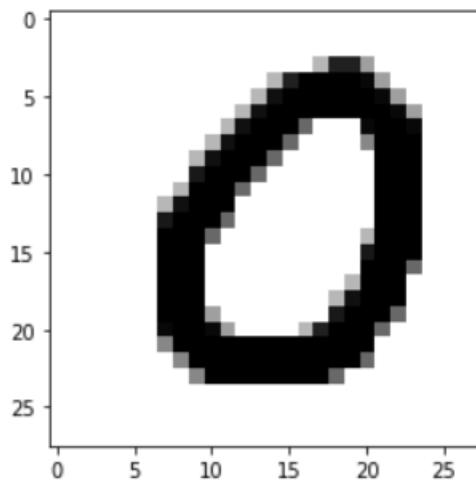
Ha minden megvan készen állunk a tanításra. Az alábbi kód a jól megszokott munkamenetek révén 20.000-szer iterál végig a tanulási folyamat alatt ez jóval hosszabb időt is vett igénybe, mint az előző feladatban.

```
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())  
    for i in range(20000):  
        batch = mnist.train.next_batch(50)  
        if i % 100 == 0:  
            train_accuracy = accuracy.eval(feed_dict={  
                x: batch[0], y_: batch[1], keep_prob: 1.0})  
            print('step %d, training accuracy %g' % (i, train_accuracy))  
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})  
  
    print('test accuracy %g' % accuracy.eval(feed_dict={  
        x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))  
    saver = tf.train.Saver()  
    saver.save(sess, "model.ckpt")
```

A tanítás kb. 20 percig tartott. Mint ahogy látható az ábrán is 0.9928-as a pontosság, ami elég jónak számít. Ezután az előző feladatból átmásoltam a prepare() függvényt illetve áadtam a modellnek a következőképpen:

```
print("-- A saját kezi 0-asom felismerese, mutatom a szamot:")  
  
img = prepare()  
image = img  
  
matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm.binary)  
#matplotlib.pyplot.savefig("8.png")  
matplotlib.pyplot.show()  
  
classification = sess.run(tf.argmax(y_conv, 1), feed_dict={x: [image], keep_prob: 1.0})  
  
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])  
print("-----")
```

```
step 19300, training accuracy 1
step 19400, training accuracy 1
step 19500, training accuracy 1
step 19600, training accuracy 0.98
step 19700, training accuracy 1
step 19800, training accuracy 1
step 19900, training accuracy 1
test accuracy 0.9928
-- A saját kezi 0-asom felismerése, mutatom a szamot:
```

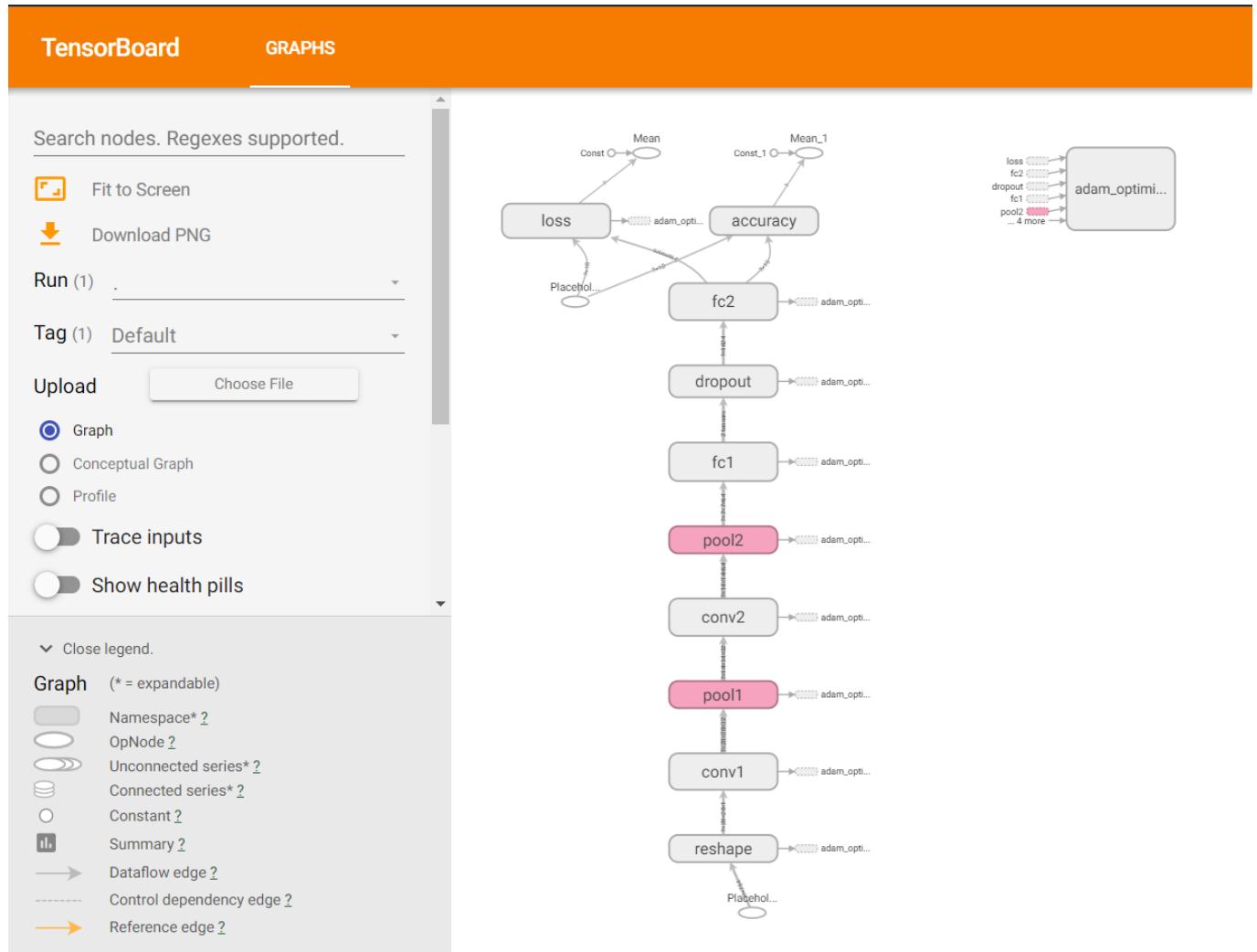


```
-- Ezt a halozat ennek ismeri fel: 0
```

20.6. ábra. Hasonló eredmény a deep CNN változattal.

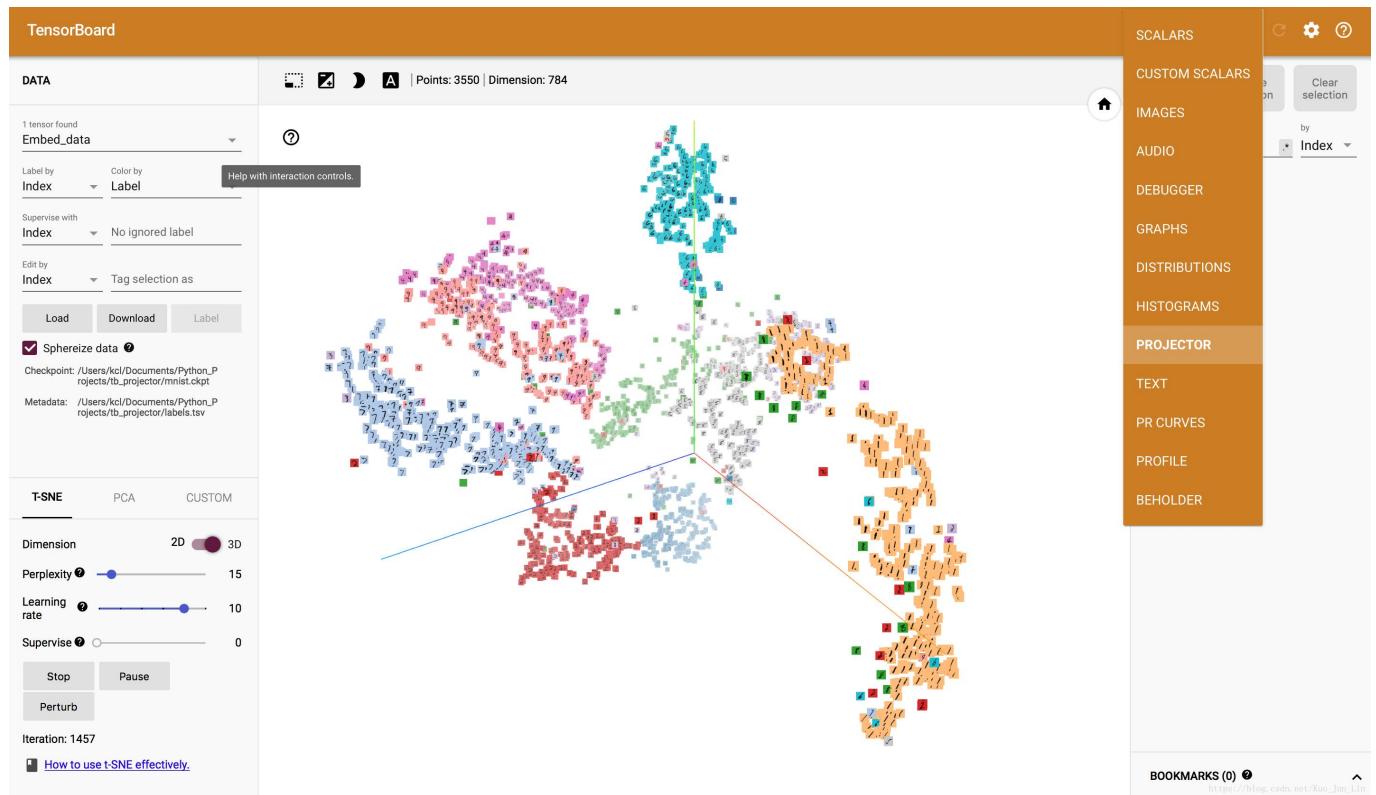
Az alábbi TensorBoard ábra jól szemlélteti az irányított gráfot illetve a tényleges deep CNN-hez tartozó layerek megléte. Nagyon jó elemző funkciók vannak benne. Az egyes elemeket fókuszba helyezve további részletet mutatott az elemre vonatkozóan. A TensorBoard-os megjelenítéshez a rajzolt irányított gráfot ki kellett menteni. A futtató paracs a következő: `tensorboard --logdir path/to/logs` Íme a kód:

```
graph_location = tempfile.mkdtemp()
print('Saving graph to: %s' % graph_location)
train_writer = tf.summary.FileWriter(graph_location)
train_writer.add_graph(tf.get_default_graph())
```



20.7. ábra. TensorBoard gráf ábra.

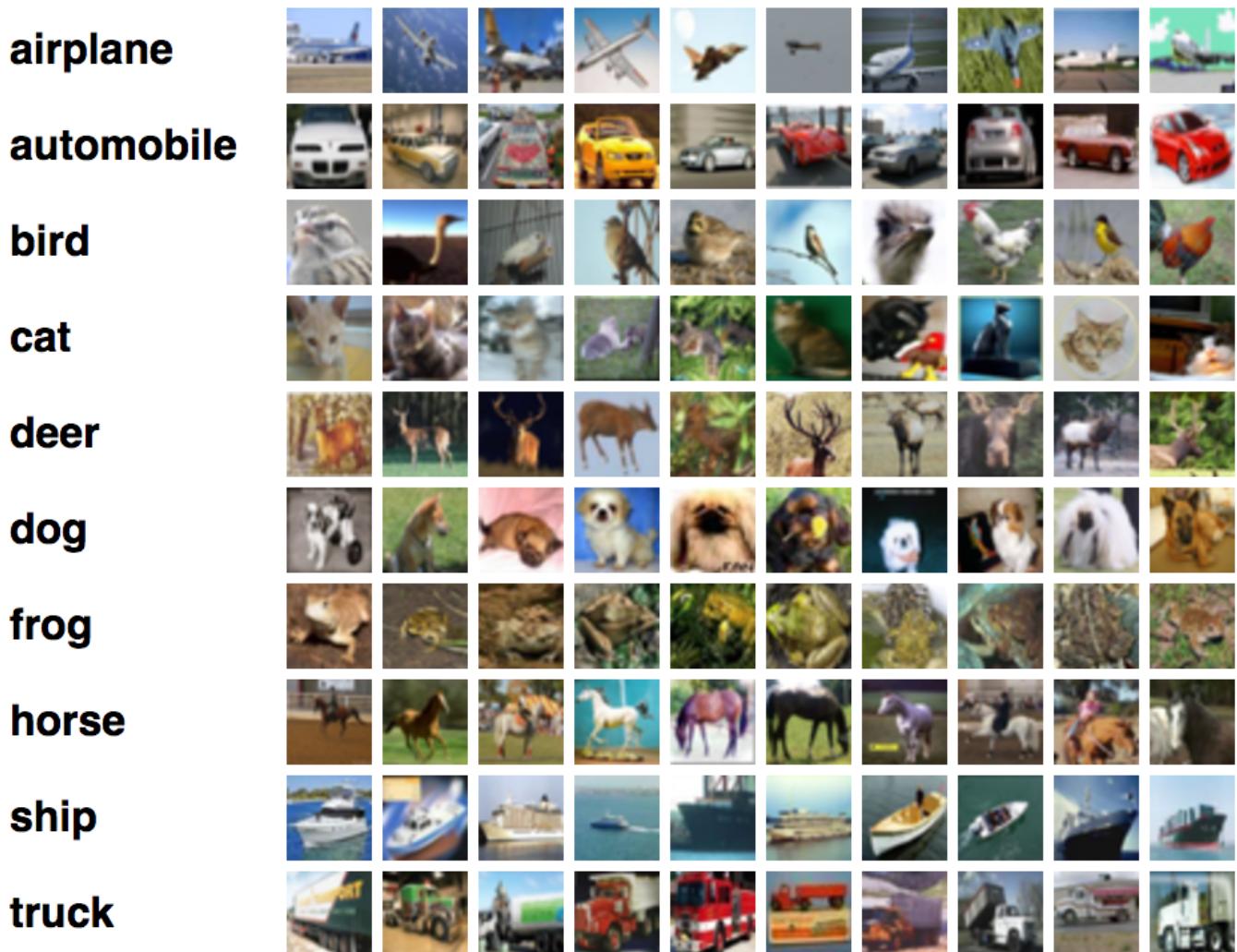
Hasonló remek kimutatásokat lehet készíteni a TensorBoard-hoz íme egy eloszlást szemléltető 3D-s modell. (A kép nem a sajátom csak példaként tettem be! A kép megtalálható, ha beírjuk azt hogy tensorboard mnist examples).



20.8. ábra. TensorBoard eloszlás ábra.

20.4. CIFAR10

A CIFAR10 dataset egy 6000 32x32px méretű képekből áll, amelyek 10 osztályba vannak sorolva.



20.9. ábra. CIFAR10 osztályok.

A feladat az volt, hogy a CIFAR adatbázist felhasználva a programunk képes legyen felismerni az általunk adott képeket. Fontos, hogy a képünk a 10 classba legyen sorolható. Ehhez ismét deep CNN-t használunk. Az elméleti hátteréhez tekints vissza a deep MNIST-es példához. A példához a Keras top-level TF könyvtárat használtam, amely jóval leegyszerűsödött a kód. A konstans deklarálás után betöljtük a CIFAR10 adatbázist. Majd teszteljük, hogy visszakaptuk-e a train-test párokat.

```
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32
BATCH_SIZE = 128
NB_EPOCH = 20
NB_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
OPTIM = RMSprop()
```

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

Ezután a train-test képeket hozzárendeljük a 10 osztályokhoz, amely a kimenetnél lesz fontos.

```
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)
```

Lényegében az adatokat normalizált vektorba töltjük vagyis felkészítjük az adatokat a feldolgozásra. (Lényegében pont az, amit én a prepare() függvényben csináltam)

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Ezt követi a model elkészítése. A model változó most már egy lineáris layer-tároló lesz, ahova addolhatjuk az egyes deep CNN layereket. Láthatjuk, hogy elsőként Conv2D konvolúciós réteget definiálunk. Ezt 3x3-as lépésekre osztjuk illetve shape-ként (32,32,3) van definiálva. Ezt követi egy ReLu aktivációs réteg. Ami után egy maxpooling-ot hajtunk végre a következő rétegen. Ezután egy 0.25 ratel random-kiválasztást alkalmaz az inputra, amellyel megelőzni kívánjuk az “overfitting” problémát. A Flatten() függvény “össze-nyomja” a beérkező inputshape-t. Pl.: (None, 64, 32, 32) => (None, 65536) A Dense(512) függvény egy egyszerű neurális háló réteg, amely 512 méretre redukálja az inputot. A többi layer tovább finomítja a az inputot, amely a Softmax layer megadja a jóolt kimenetet.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=(IMG_ROWS, ←
    IMG_COLS, IMG_CHANNELS)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_2 (Conv2D)	(None, 32, 32, 32)	896
activation_2 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_2 (Dropout)	(None, 16, 16, 32)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 512)	4194816
activation_3 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130
activation_4 (Activation)	(None, 10)	0
<hr/>		
Total params: 4,200,842		
Trainable params: 4,200,842		
Non-trainable params: 0		

20.10. ábra. Modell szerkezete.

A tanulás a compile() és fit() keras függvények segítségével történik. A compile függvény konfigurálja fel a modelltunket a megadott funkcionális argumentumok segítségével. A fit() függvény tanítja fel a modellt. Itt a batch_size a lépésszámot jelenti az epochs pedig a max iterációt.

```
model.compile(loss='categorical_crossentropy', optimizer=OPTIM, metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, validation_split=VALIDATION_SPLIT, verbose=VERBOSE)
score = model.evaluate(X_test, Y_test, batch_size=BATCH_SIZE, verbose=VERBOSE)
print("Test score:", score[0])
print("Test accuracy:", score[1])
```

Ha a tanítás megtörtént akkor tesztelni tudjuk a hálózatunkat: Elsőként a open() függvény segítségével betöljük a képet az állományrendszerből majd a resize() segítségével 32x32 méretűvé méretezzük. Ezután egy numpy array-t készítünk a képünkben majd ezt átdobjuk a model.predict_classes() függvénynek, amely visszaadja a megjósolt osztályt, ahova tartozik a kép.

```
from PIL import Image
import numpy as np

classes = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", ←
    "horse", "ship", "truck"]

img = Image.open("automobile.jpg")
img = img.resize((32, 32))
image = np.array(img)
result = model.predict_classes(image.reshape((1, 32, 32, 3)))
print(" -----")
print("Kép: Bátfai Tanár Úr legendás matchboxa.")
print( model.predict_proba(image.reshape(1, 32, 32, 3)) )
print( classes[result[0]] )

# [[0. 1. 0. 0. 0. 0. 0. 0. 0.]]
# automobile

img = Image.open("deer.png")
img = img.resize((32, 32))
image = np.array(img)
result = model.predict_classes(image.reshape((1, 32, 32, 3)))
print(" -----")
print("Kép: Szarvas")
print( model.predict_proba(image.reshape(1, 32, 32, 3)) )
print( classes[result[0]] )

# [[0. 0. 0. 0. 1. 0. 0. 0. 0.]]
# deer

img = Image.open("IFA.jpg")
img = img.resize((32, 32))
image = np.array(img)
result = model.predict_classes(image.reshape((1, 32, 32, 3)))
print(" -----")
print("Kép: Saját választott képem egy IFA-ról.")
print( model.predict_proba(image.reshape(1, 32, 32, 3)) )
print( classes[result[0]] )

# [[0. 0. 0. 0. 0. 0. 0. 0. 1.]]
# truck
```



20.11. ábra. Automobile kép.



20.12. ábra. Szarvas kép.



20.13. ábra. IFA kép.

Kép: Bátfai Tanár Úr legendás matchboxá.
[[0. 1. 0. 0. 0. 0. 0. 0. 0.]]
automobile

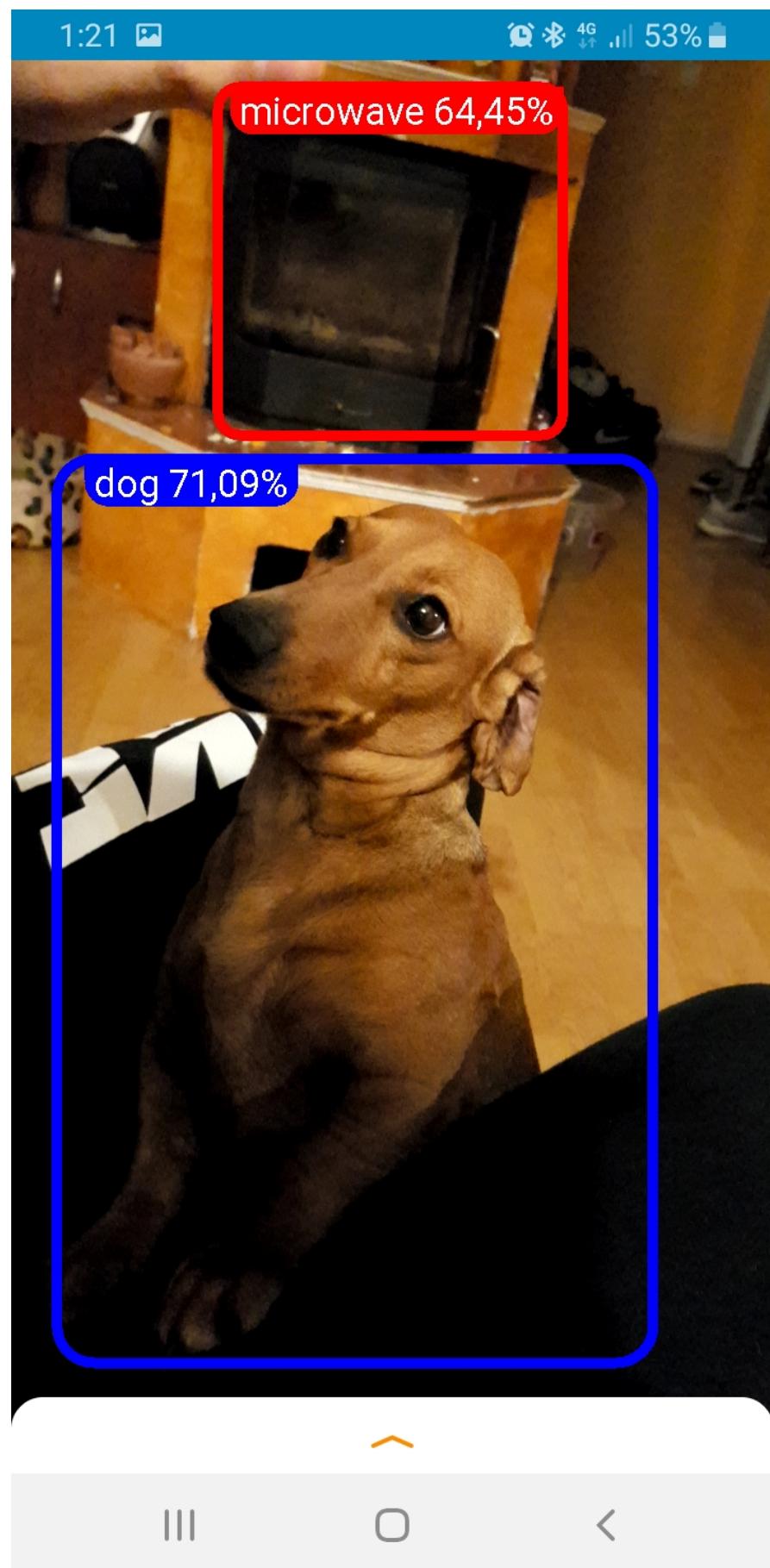
Kép: Szarvas
[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
deer

Kép: Saját választott képem egy IFA-rol.
[[0. 0. 0. 0. 0. 0. 0. 0. 1.]]
truck

20.14. ábra. CIFAR10 eredmény.

20.5. Android telefonra TF objektum detektálója

A feladat csak annyi volt, hogy próbáljuk a ki az Androidos TensorFlow objektum felismerő alkalmazását. Sokat szórakoztam vele, voltak egész jó és kevésbé jó eredmények. Itt az éppen a kutyánkon próbáltam ki és egész jól felismerte.

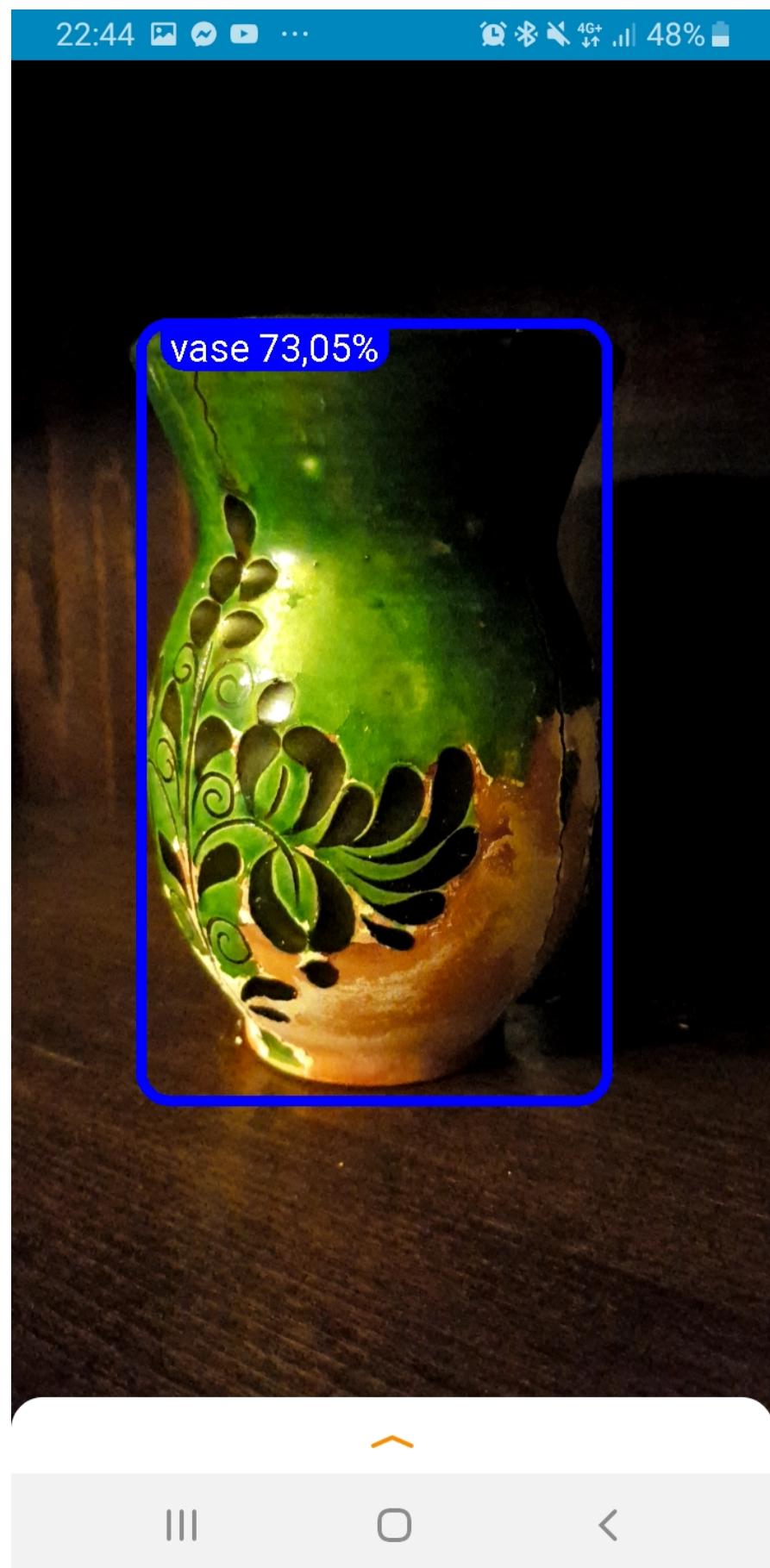


20.15. ábra. Kutyánk felismerése.

Okosórát is elég jól feismerte.



20.16. ábra. Óra felismerése.



20.17. ábra. Váza felismerése.

Akadtak kevésbé jó eredmények. Sajnálatos módon nem sikerült felismernie az ukulelét, de még gitár se illett rá.



20.18. ábra. Ukulele felismerése.

IV. rész

Irodalomjegyzék

20.6. Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

20.7. C

[KERNIGHANRITCHIE] Kernighan, Brian W. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

20.8. C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

20.9. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, 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.