

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. május 10, v. 1.0.0

Copyright © 2019 Dr. Bátfai Norbert

Copyright © 2019 Szilágyi Csaba

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

Copyright (C) 2019, Szilágyi Csaba, ninjaraccoon4@gmail.com,

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

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

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

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

COLLABORATORS

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert ÁCs Szilágyi, Csaba	2019. december 12.	

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
0.1.0	2019-02-27	Feladatak elkészítésének megkezdése.	nraccoon
0.1.1	2019-03-04	Első feladatcsokor (2.fejezet) befejezve.	nraccoon
0.1.2	2019-03-10	Harmadik fejezet elkészítve.	nraccoon
0.1.3	2019-03-16	Negyedik fejezet elkészítve	nraccoon

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.1.4	2019-03-26	Ötödik fejezet befejezve.	nraccoon
0.1.5	2019-04-02	Hatodik fejezet befejezve.	nraccoon
0.1.6	2019-04-07	Hetedik fejezet befejezve.	nraccoon
0.1.7	2019-04-18	Olvasónaplók kidolgozva.	nraccoon
0.1.8	2019-04-23	Nyolcadik fejezet elkészítve.	nraccoon
0.1.9	2019-04-27	Kilencedik fejezet elkészült.	nraccoon
0.2.0	2019-05-09	Feladatok finomításai, és véglegesítései.	nraccoon
1.0.0	2019-05-10	A könyv első teljes kiadása.	nraccoon

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	9
2.6. Helló, Google!	10
2.7. 100 éves a Brun téTEL	11
2.8. A Monty Hall probléma	12
3. Helló, Chomsky!	14
3.1. Decimálisból unárisba átváltó Turing gép	14
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	15
3.3. Hivatalos nyelv	16
3.4. Saját lexikális elemző	17
3.5. l33t.l	18
3.6. A források olvasása	19
3.7. Logikus	20
3.8. Deklaráció	21

4. Helló, Caesar!	23
4.1. double ** háromszögmátrix	23
4.2. C EXOR titkosító	25
4.3. Java EXOR titkosító	26
4.4. C EXOR törő	27
4.5. Neurális OR, AND és EXOR kapu	28
4.6. Hiba-visszaterjesztéses perceptron	30
5. Helló, Mandelbrot!	32
5.1. A Mandelbrot halmaz	32
5.2. A Mandelbrot halmaz a std::complex osztállyal	33
5.3. Biomorfok	35
5.4. A Mandelbrot halmaz CUDA megvalósítása	36
5.5. Mandelbrot nagyító és utazó C++ nyelven	37
5.6. Mandelbrot nagyító és utazó Java nyelven	37
6. Helló, Welch!	39
6.1. Első osztályom	39
6.2. LZW	39
6.3. Fabejárás	40
6.4. Tag a gyökér	42
6.5. Mutató a gyökér	44
6.6. Mozgató szemantika	44
7. Helló, Conway!	47
7.1. Hangyszimulációk	47
7.2. Java életjáték	48
7.3. Qt C++ életjáték	48
7.4. BrainB Benchmark	50
8. Helló, Schwarzenegger!	52
8.1. Szoftmax Py MNIST	52
8.2. Mély MNIST	55
8.3. Minecraft-MALMÖ	55

9. Helló, Chaitin!	58
9.1. Iteratív és rekurzív faktoriális Lisp-ben	58
9.2. Gimp Scheme Script-fu: króm effekt	59
9.3. Gimp Scheme Script-fu: név mandala	59
10. Helló, Gutenberg!	61
10.1. Programozási alapfogalmak	61
10.2. Programozás bevezetés	62
10.3. Programozás	64
III. Második felvonás	66
11. Helló, Berners-Lee!	68
11.1. Összehasonlító jellegű olvasónapló	68
11.2. Élmény-olvasónapló	70
12. Helló, Arroway!	72
12.1. OO szemlélet	72
12.2. "Gagyi"	75
12.3. Yoda	77
12.4. Kódolás from scratch	79
13. Helló, Liskov!	82
13.1. Liskov helyettesítés sértése	82
13.2. Szülő-gyerek	86
13.3. Anti OO	89
13.4. Hello, Android!	90
14. Helló, Mandelbrot!	95
14.1. Reverse engineering UML osztálydiagram	95
14.2. Forward engineering UML osztálydiagram	98
14.3. BPMN	99
15. Helló, Chomsky!	102
15.1. Encoding	102
15.2. Full screen	105
15.3. Perceptron osztály	107

16. Helló, Stroustrup!	110
16.1. JDK osztályok	110
16.2. Hibásan implementált RSA törése	111
16.3. Változó argumentumszámú ctor	114
17. Helló, Gödel!	116
17.1. Gengszterek	116
17.2. STL map érték szerinti rendezése	117
17.3. Alternatív Tabella rendezése	118
17.4. GIMP Scheme hack	121
18. Helló, Szünet!	123
18.1. OOCWC Boost ASIO hálózatkezelése	123
18.2. SamuCam	125
18.3. BrainB	128
19. Helló, Lauda!	130
19.1. Port scan	130
19.2. AOP	132
19.3. Junit teszt	134
20. Helló, Calvin!	136
20.1. MNIST	136
20.2. CIFAR-10	141
20.3. Android telefonra a TF objektum detektálója	146
IV. Irodalomjegyzék	155
20.4. Általános	156
20.5. C	156
20.6. C++	156
20.7. Lisp	156

Ábrák jegyzéke

4.1. Double **	24
4.2. MLP tanítása hibavisszaterjesztéses algoritmussal	30
7.1. Sejtablak	49
7.2. BrainB Benchmark	50
12.1. PolarGen.java futása	73
12.2. PolarGen.cpp futása	74
12.3. Gagyi.java futása	76
12.4. Gagyi2.java futása	77
12.5. Yoda.java futása	78
12.6. Yoda2.java futása	78
12.7. BBP formula	79
12.8. PiBBP.java futása	81
13.1. Liskov.java futatása	83
13.2. madar.cpp futatása	84
13.3. megoldasmadar.cpp futatása	86
13.4. Osztaly.java fordítása	87
13.5. Szulo-gyerek.cpp fordítása	88
13.6. Eredmények	89
13.7. Anti.cpp futása	90
13.8. PiBBPBench.java futása	90
13.9. SMNIST háttér szín	91
13.10SMNIST háttér szín	92
13.11SMNIST pont háttér	93
13.12RGB színkód "generálás"	94

14.1. UML Binfa	95
14.2. Code Importing Wizard	96
14.3. Binfa.cpp bevitel	96
14.4. Importálás	97
14.5. Three view	97
14.6. UML diagram	98
14.7. Code Generation Wizard	98
14.8. Code Generation Options	99
14.9. Draw.io kezelőfelülete	100
14.10Megoldás	101
15.1. Fájlnév elváltozás	102
15.2. Fordítási hiba	103
15.3. Fordítás hiba nélkül	104
15.4. MandelbrotHalmazNagyító futása	105
15.5. Full screen game	107
15.6. Mandel	109
16.1. RSA	114
17.1. Alternativ rendezett tabella	120
18.1. SamuCam futása	127
19.1. KapuSzkenner futása	132
19.2. Elvárt értékek kiszámolása papíron	135
20.1. Első szám felismerése	139
20.2. Második szám felismerése	140
20.3. Saját szám felismerése	141
20.4. Cifar 1	143
20.5. Cifar 2	144
20.6. Cifar 3	145
20.7. Beagle	146
20.8. TF Detect 1	147
20.9. TF Detect 2	148
20.10.TF Classify 1	149

20.11.TF Classify 2	150
20.12.TF Detect 3	151
20.13.TF Detect 4	152
20.14.TF Classify 3	153
20.15.TF Classify 4	154

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás forrása:

100% 1 mag: [Ciklus1](#)

0% minden mag: [Ciklus2](#)

100% minden mag: [Ciklus3](#)

Ez a feladat 3 részre bontható. Az egyikben 100%-on kell dolgoztatni egy magot, a másikban 0%-on minden magot, az utolsóban pedig 100%-on minden magot.

Az első rész, egy egyszerű végtelen ciklus megírásával, már készenek is mondható. A *Megoldás forrásban* látható, hogy én a while használatával készítettem el a végtelen ciklusom. Persze, ezt másképp is meg lehet oldani például egy: "for(;;)" ciklus használatával, ami még talán jobb is lehet, bizonyos szempontokból, de a feladat megoldásához tökéletesen megfelel az én első gondolatom is ami egy egyszerű minden igaz while ciklusra épül. Ugyanis ez tökéletesen 100%-on használ ki egy magot futásakor.

```
while (0<1) { }
```

Mivel ez minden igaz lesz ezért a végtelenségig fut a program, és egy magot teljesen lefog.

A második rész már bonyolultabb egy kicsit, mivel itt a while cikluson belül használnunk kell egy sleep() kifejezést.(A mellékelt módon hasonlóan!)

```
while (0<1) {
    sleep(1); }
```

Valamint a forrás elején includeolunk kell az unistd.h header fájlt.

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

Ezen kívül viszont már nincs más dolgunk csak fordítani és futtatni, az eredmény a várt lett, 0% terhelést nehezít a magokra a program.

A feladat utolsó része tűnhet már első ránézésre is a legnehezebbnek, és ez nincs is máshogyan.

Én ennek a feladatnak a megoldására az OpenMp-t használtam, mely minden magra kiterjeszti a terhelést, ezáltal elérhetjük hogy a program futásakor minden magot 100%-on pörgessen.

Az OpenMp használatához includeolunk kell az omp.h könyvtárat!

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

Valamint a végtelen ciklusunk elé kell írnunk a #pragma omp parallel kifejezést!

```
#pragma omp parallel
while(0<1) {}
```

A kódunk fordításához a következő parancssort kell begépelnünk a terminálba:

```
gcc -fopenmp (fájlnév pl:) 3.c -o (fordítás utáni fájlnév pl:) 3
```

Ezután futtatjuk a programot és láthatóvá válik hogy minden magot (én esetben 4 magot) 100%-on pörget.

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.

Tapasztalatok és megfigyelések: Ilyen programot nem lehet írni, ugyanis ellentmondásra jutunk. Egy program nem fogja tudni megmondani hogy a neki paraméteréül adott program végtelenségig fog-e futni vagy lesz olyan pont ahol megáll.

Legjobban talán az tudná szemléltetni ha paraméterként saját magát kapná meg a program.

Mi történne ilyenkor?

Ha megállna a program csak akkor tudná visszaadni az értéket, hogy a végtelenségig fut, de akkor ez nem lenne a valóságnak megfelelő adat, hiszen nem lenne igaz, mivel megállt. A másik eset az lenne, ha nem állna meg a program futása, de ez esetben pedig nem tudná megadni azt hogy Ó a végtelenségig fut, hiszen nem állt még meg. Ezáltal egy fajta paradoxonra jutunk, és el kell ismernünk, hogy nem létezik erre a problémára megoldás.

Eleinte talán nehezen érthető még magyarázzattal is, de többszöri átolvasás után véleményem szerint mindenkinnek ez a példa triviális lesz. Bátfai Tanár úr kódját nézzük a magyarázat mellé a könnyebb megértés végett!

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Változók cseréje kivonás és összeadás műveletek segítségével: [Csere](#)

Két változó értékének a felcserélésére több lehetséges opciónk is van, az egyik legegyszerűbb megoldás erre egy új változó (ún. segéd változó) létrehozása lenne, Aminek értékül adnánk az egyik felcserélendő változót, aztán a két változó értékét egyenlővé tennénk, majd a segéd változóból átraknánk az értéket a másik felcserélendő változóba.

Viszont segéd változó bevezetése nélkül is megoldható, például kivonás és összeadás művelet segítségével.

Három lépésből megoldható a változók értékeinek cserélése a kivonás és az összeadás művelet segítségével is. Első lépésben vesszük a két változó összegét, tegyük fel a két változónk most "a" és "b". Ez alapján első lépésünk a következő legyen:

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

Ezáltal az új "a" értéke az összeg lesz. Ezután következő lépésünk legyen az alábbi:

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

Így "b" értéke már is az eredeti "a" értékét hordozza magába. Ugyanis az összegből vontuk ki a "b" értékét ami így a régi "a" értékét adja. Már csak a régi "b" értéket kell megadni az új "a"-nak amit a következő képpen tehetünk meg:

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

És kész is vagyunk az érték felcsereléssel, most már az "a" hordozza a régi "b" értéket és a "b" a régi "a"-t. Ugyanis az összegből kivontuk az új "b" értéket ami a régi "a" értékkel azonos, tehát a régi "b" értéket adja vissza.

Ez egy egyszerű ám de hasznos feladat, később sokszor alkalmazhatjuk, valamint megértése is triviális, egy általános iskolás számára is.

2.4. Labdapattogás

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

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

Megoldás forrása:

If segítségével való labda pattogtatás: [Labdaif](#)

Logikai utasítás vagy kifejezés használata nélkül való labda pattogtatás: [Labda](#)

If-ek használatával, egy fokkal könnyebb dolgunk van, mint ha nem használhatnánk őket. Ugyanis ennek a feladatnak az a lényege hogy kövessük a "labda" mozgását, és az amint eléri a konzolunknak a "falát" a mozgásának az iránya megforduljon/megváltozzon. If-ek segítségével ezt az ellenőrzést könnyebben vizsgálhatjuk mint a következő esetben ahol nem használhatunk logika utasítást vagy kifejezést.

```
if ( x>=mx-1 ) {           // elérte-e a jobb oldalát?
    xnov = xnov * -1;
}
if ( x<=0 ) {               // elérte-e a bal oldalát?
    xnov = xnov * -1;
}
if ( y<=0 ) {               // elérte-e a tetejét?
    ynov = ynov * -1;
}
if ( y>=my-1 ) {           // elérte-e az alját?
    ynov = ynov * -1;
}
```

A feladathoz a programnak tudnia kell a konzolunk méretét, ugyanis csak így tudja hol a konzol "fala". Úgy képzeljük el hogy a konzol mérete alapján létre hozz egy "táblázatot" ahol a határértékek megegyeznek, így ha oda ér a "labdánk" akkor megváltoztatja irányát, míg bárholt másol a konzolon belül nem fogja megváltoztatni annak mozgását. A program folyamatosan updatei magát és ahol jár a "labda" azt kiprinteli, így tud a "labda" haladni a képernyőnkön. A program tehát nem csinál mászt csak számolja a labda haladását mind x mind pedig y tengelyen való haladását és hogy az adott hely a konzol határa-e vagy sem, amint eléri a határt mozgási irányt változtat, végig követi és printeli a haladást. If-ek segítségével és if-ek nélkül is ugyan ez az alapja a programnak.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

Szóhossz: [Szóhossz](#)

A szóhosszt a bitshift operátor segítségével tudjuk elvégezni. Ami úgy működik hogy a forráskódban szereplő "a"-t lépteti balra és a jobb oldalára pedig 0 értéket ad. Mind addig folytatja ezt míg ki nem nullázza az egészet.

Hogyan tudjuk meg ebből a bitek számát?

Minden shiftelés(léptetés) után növeljük a forráskódban lévő "b" változó értékét 1-el, teljesen a ciklus végéig, tehát ha lezárul akkor a "b" értéke pontosan a bitek számával fog megegyezni, azaz megtudja azt mondani hogy hány bites a szó a számítógépünkön.

```
while (a != 0) {  
    a<<=1;  
    b++;  
}
```

A konzolon való megjelenítéshez pedig a printf-et fogjuk használni, a következő képpen:

```
printf("%d bites a szó.\n", b);
```

A következő képpen adja vissza tehát például: 32 bites a szó.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Page-Rank: [Page-Rank](#)

A Page-Rank egy algoritmus. Nem véletlenül a "Helló, Google!" cím, ugyanis a Page-Rank a Google keresőmotorjának az egyik legfontosabb eleme. 1998-ban fejlesztették ki a google alapítói Larry Page és Sergey Brin. Az algoritmus használata előtt közel sem volt olyan pontos a keressési találat mint napjainkban, szinte ma már a keresőbe beírt szavak alapján az első három találta benne van a keresett oldal, ritka az amikor a találatok között is keresünk kellene a megfelelőt, régebben még nem volt ilyen tökéletes ez. Ez a pontoság napjainkban annak köszönhető hogy a Page-Rank sorba állítja az oldalakat az alapján hogy hány link mutat rájuk valamint a rájuk mutató oldalakra hány oldal mutat. De miért is jó ez?

Bonyolultnak hangzik ez és értelmetlennek viszont mint látjuk a google kereső motor példáján is, ugyancsak hasznos és működése is kiváló. Visszatérve tehát mit is jelent hogy sorba állítja az oldalakat a fent említettek alapján? Azt takrja hogy a legtöbb oldal hivatkozással rendelkező oldal lesz az első és minél kevesebb oldal hivatkozik rá annál lentebbre kerül az adott lap. Ezáltal a legtöbbet keresett oldalak/leghasználtabb/legnépszerűbb találatok lesznek legelöl, így a keresséi találatok tetején 99,9% kb hogy közte van az általunk keresett oldal.

A feladatban egy 4 honlapból álló hálózatra nézzük meg a Page-Rank értéket.

Az alap érték 1/4-ed lesz minden oldalnak mivel ugye 4 oldalt nézünk most. Az oldalak közötti kapcsolatot egy mátrixban tároljuk el, a következőféléképpen:

```
{  
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}  
};
```

Egy fajta táblázati nézetként tudható ez be, a sorok és az oszlopok megfelelő találkozási pontjánál olvasható le a kapcsolatuk. Ezeket az adatokkal a megfelelő matematikai műveletekkel, mint például a mátrixszorzás, kapjuk meg az oldalak Page-Rank értékét. Ezek az értékek egy tömbe lesznek ami az eredmény tömbünk, ezt akár ki is lehet írtani, így láthatóvá téve az eredményeket a terminálon vagy egy output fájlba. A Page-Rank-ot lehet használjuk nap mint nap is akár keresőmotor használatával például, de magát a kódot nem kell alkalmaznunk, viszont érteni igen, tehát ezt a feladatot inkább megértésre és kód olvasásra ajánlanám, jó magam se írtam meg újra a kódöt, internetről szedett kódöt mellékeltem tanulmányozásra.

2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

A Brun téTEL a prímszámokkal azon belül is az ikerprímekkel foglalkozik. De mik is a prímszámok illetve az ikerprímek?

A prímszámok azon természetes számok melyek csak eggyel és önmagukkal oszthatóak. Ilyen például a 2, az 5, a 7 stb. A 2 egyben az első prímszám is valamint az egyetlen páros prímszám is, értelemszerűen a kettő kivételével a többi párosszám nem lehet prímszám mivel kivétel nélkül oszthatóak kettővel. A prímszámok szorzataként minden természetes szám előállítható, ezért nevezik őket a természetes számok építőelemeinek is őket. Továbbá a prímszámok száma végtelen, ezt már Krisztus előtt is tudták.

A ikerprímek azon prímszámpárok amelyeknek a különbsége kettő. Például 5 és 7 ikerprímek hiszen a különbségük kettővel egyenlő. A Brun téTEL erre épül, ugyanis azt nem tudni hogy az ikerprímek száma is végtelen-e vagy van úgy mond egy "utolsó" ikerprím. (Ezt máig napig kutatják még.)

A Brun téTEL azt mondja ki, hogy az ikerprímek reciprokaiból képzett sor összege véges vagy legalábbis végtelen sor konvergens, azaz az így képzett törtszámok egy határt képeznek amit nem lépnek át soha, ezt a határ értékét Brun konstansnak nevezik.

A mellékelt forrásban lévő kód ezt a "határértéket", ún. Brun konstanst próbálja elérni.

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

Az x-nek megadott számig kiszámolja a prímeket.

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

Megnézi a prímek közti különbséget, ez azért fontos mivel így kitudjuk szűrni az ikerprímeket ugyanis azok közti különbség minden kettő.

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

Kitudjuk szedni az első tagot, és az alapján +2-vel a hozzá tartozó párt, az így kapott számok ikerprímek lesznek tehát nekik kell a reciprokuk.

```
t1primes = primes[idx]
t2primes = primes[idx]+2
rt1plust2 = 1/t1primes+1/t2primes
```

Ezeket a törteket összeadva kell kapnunk azt az értéket mely körülbelül az 1,8-hoz közelít minél több számot vizsgálunk át. Ezt a sum-mal tehetjük meg.

```
return(sum(rt1plust2))
```

Persze szemléletesebb a dolog ha ki is rajzoltatjuk őket.

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

A Monty Hall paradoxonra épülő "műsor" lényege az hogy van 3 ajtó, de csak az egyik mögött van értékes nyeremény, a másik kettő mögött értéktelen dolog van, vagy semmi, a paradoxon szempontjából nem lényeges, csak annyi hogy 1 jó és 2 rossz választás van. A játékosnak lehetősége van az egyik ajtót megjelölni, hogy azt szeretné majd kinyitni, de ezután a játékvezető kinyit egy rossz ajtót, és megengedi hogy változtasson a választásán a játékos, de ezt csak egyszer teheti meg. Nyilvánvalóan a játékvezető tudja hogy hol a nyeremény és csak azt az ajtót nyitja ki ami mögött nincs az értékes nyeremény, valamint a játékos által megjelölt (elsőnek kiválasztott) ajtót sem nyithatja a játékvezető.

A Monty Hall probléma/paradoxon tehát egy érdekes téma, ugyanis nem mindenkinél triviális elsőre az hogy érdemes változtatni. Jó magam se értettem hogy miért lenne érdemesebb változtatni a döntésen. Hisz 50-50% esély van hogy értékes vagy sem a mögötte lévő dolog. Hisz egy rossz ajtót "kizár" a játékvezető, aztán újra nálunk a döntés mintha csak 2 ajtó lenne.

De ha jobban belegondolunk valóban igaz az hogy elsőre 66% eséllyel egyik rossz választásra bökünk rá, míg csak 33% az esély hogy egyből a jót találjuk el. Így a szabályoknak megfelelően ha kizárjuk az egyik rossz ajtót, 66% esély arra hogy a változtatás után nyerünk és csak 33% az esély arra hogy ha nem változtatunk akkor nyerünk.

A forrásban szereplő kód áttanulmányozása után is jól látható, hogy mi a "műsor" lényege. Láthatjuk azt is hogy a műsorvezetőnek adott az ajtó ha a játékos elsőre rosszat választ, mivel sem a jót sem a játékos ajtaját nem nyithatja ki.

```
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) ]

}

}
```

Nagyon jó kis gondolkodós példa, a forráskód Bátfai Norbert Tanár úr-é és R nyelvben íródott, de R nyelvet sose látott olvasók számára is olvasható véleményem szerint.

Ehhez a feladathoz ajánlom a "21 Las Vegas ostroma" című filmet, mely az ajánlott filmek között már fentebb is megtalálható volt, ebben szerepelni fog a Monty Hall probléma.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: [Turing gép](#)

A Turing-gép Alan Turing angol matematikus nevéhez kötődik közvetlen, Ő volt az aki megfogalmazta eme algoritmust és leírta, valamint megjelentette egyik cikkében ezt a jelenséget még 1936-ban, amikor már képes lett volna olyan mai számítógépes probléma megoldásra ami akkoriban még nem is létezett mint eszköz sem.

A Turing-gép úgynevezett absztrakt automata amely azt takarja, hogy valóságos digitális számítógépek nagyon leegyszerűsített verziójára hasonlít vagy nevezhető is annak. De bővebben a Turing gépről már magyar nyelven is olvashatunk a [Wikipédia oldalán](#).

```
#include <stdio.h>

int main(){
    int szam;
    printf("Adj meg egy számot: ");
    scanf("%d", &szam);

    for(int i = 0; i < szam; i++) {
        printf("1");
    }
    printf("\n");
    return 0;
}
```

A fenti kód egy egyszerű kód. Hogy is működik és mit is csinál tehát?

Deklarálunk egy "szam" változót majd a standard outputra kiírjuk a felhasználónak hogy adjon meg egy számot, amit megad szám azt pedig a deklarált "szam" változóba mentjük el. Ezután egy for ciklus segítségével a kapott egész számmal megegyező karakter írunk ki a standard outputra. Az én kódomban ez a

karakter az egyes, de lehetne akár a "|", "/", "\" vagy egyébb szemléletes, könnyen számlálható karakter. Tehát a természetes számokat unáris alakba képes megadni, nyilván hátrány hogy csak természetes számokkal tud dolgozni, törteket és negatív számokat sem tud kezelni.

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

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

Megoldás videó:

Megoldás forrása:

A generatív grammatika, generatív nyelvtan azoknak a csoportosítása Noam Chomsky nevéhez köthető, aki az 1960-as években hozta létre ezeket a csoportokat, fő célja a nyelvek modellezése volt.

Az amerikai nyelvész 4 osztályba sorolta, ezt a csoportosítást Chomsky hierarchiának hívják, de hogy is néz ez ki?

- Az általános típus (0. típus).
- A környezetfüggő (1. típus), ez az a típus amivel a feladatsorán foglalkoznunk kell.
- A környezetfüggetlen típus (2. típus).
- Reguláris (3. típus).

Chomsky-féle nyelvosztályok: A nyelvtan és az általa generált nyelv definíciója szerint minden nyelvtanhoz egy egyértelműen meghatározott nyelv tartozik, de megfordítva ez már nem lesz igaz, ugyanis egy nyelvet nem csak egy nyelvtannal generálhatunk le. Erről részletesebben és a generatív nyelvtanról bővebben olvashatunk a [következő oldalon](#).

Két környezetfüggő generatív grammatika, amely a nyelvet generálja például:

S, X, Y változók
a, b, c konstansok

S - abc, S - aXbc, Xb - bX, Xc - Ybcc, bY - Yb, aY - aaX, aY - aa

S (S - axBc)
aXbc (Xb - bX)
abXc (Xc - Ybcc)
abYbcc (bY - Yb)
aYbbcc (aY - aa)
aabbcc

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

```
aaYbbbccc (aY - aa)
aaabbccc
```

Vagy például:

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

```
A (A - aAB)
aAB ( A - aC)
aaCB (CB - bCc)
aabCc (C - bc)
aabbcc
```

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

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

[c89 vs c99](#)

Az utasítások, ha nem jelezzük hogy ne sorba fussanak le, akkor leírásuk sorrendjébe hajtódnak végre. Nem rendelkeznek értékekkel valamint több csoportjuk ismert C nyelvben, ilyenek például:

- Kifejezésutasítások: Az utasítások többsége ide tartozik. Legáltalánosabb kifejezésutasítás a függvényhívás valamint az értékadás.
- Összetett utasítások: Másik nevén a blokk, arra képes hogy egy utasításként kezeljen egyszerre több utasítást, ez azért jó mert vannak programkörnyezetek, ahol a fordítóprogram csak egyetlen utasítással tud dolgozni, egyszerre többet nem tud elfogadni.

- Iterációs utasítások: Egy ciklust határoznak meg. Ebből a típusból a legismertebb a while, do utasítások.

Lehetne sok ilyen kódot írni, de szemléltetésnek elég egy egyszerű for ciklust írnunk ugyanis c89-ben a c99-es for ciklus felépítés nem fog lefordulni.

```
for (int i=0; i<5; i++) {}
```

Ugyan ezt c89-ben így írhatjuk meg:

```
int i=0;
for (i<5; i++) {}
```

Ez számomra meglepő volt, ugyanis én még nem használtam c89-et, én már megszokásból is a c99 féle for ciklust írnám meg, ami persze fordításnál egyből hibát lökne.

3.4. Saját lexikális elemző

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU (15:01-től).

Megoldás forrása:

Lexikális elemző: [Valós számok](#)

A forrás 3 részre van bontva, ennek mentén haladok a magyarázáttal is.

Az első részben includeoljuk az stdio.h könyvtárat, valamint létrehozunk egy változót amely számolja nekünk azt, hogy hány számot olvass be a program.

```
#include <stdio.h>
int realnumbers = 0;
```

Majd a digitnek megadjuk hogy a 0-9-ig terjedő számjegyeket tartalmazza.

```
digit [0-9]
```

A második részében a kódnak, rögzítjük a szabályokat. Azaz itt lesz megadva hogy nagyjából mit csinál a függvény. Segítségünkre lesz ebben például az informatikában sűrűn használt csillag mely tetszőleges számú tetszőleges karaktert jelent általában. (A forrásban a digit mivel előtte áll így tetszőleges számú tetszőleges számjegyet fog takarni.) Majd ugyanebben a részben ki is íratjuk a stringet a printf segítségével, valamint az atof segítségével a string alapján készített számot elkészítjük és ugyan úgy a printf használatával ki is íratjuk.

```
{digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
```

A 3. rész pedig maga a program, mely ugye az int main-től indul, hívja a lexikális elemzőt, majd pedig kiíratja a "realnumber" értékét, ami a végeredmény.

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

Összeségében tehát nem egy bonyolult kód, de hogy tudjuk működtetni?

Először is a .l fileunkat lexeljük a következőképpen:

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

Ezután fordítjuk következőképpen az előbb létrehozott .c fileunkat:

```
gcc realnumber.c -o realnumber -lfl
```

Majd futtatjuk, és a példában szereplő forrás a terminálból fogadd inputot.

```
./realnumber
```

3.5. I33t.l

Lexelj össze egy l33t cipher!

Megoldás videó: https://youtu.be/06C_PqDpD_k

Megoldás forrása:

Leet chiper: [L33t](#)

A feladat megoldásában tutoriált engem: Dankó Zsolt

Flex lexikális elemzővel tudjuk ezt a feladatot is elvégezni, az előző feladatra épül rá igazából, a fordítást és a futattást is ahhoz hasonlóan kell elvégeznünk, tehát valahogy így:

```
lex -o 133t.c 133t.l
```

```
gcc 133t.c -o 133t
```

```
./133t
```

A kód főeleme egy chiper lesz, ezért először is ennek a struktúráját kell felépítenünk, de ne essünk kétségből ehhez a segítségünkre lesz a [következő oldal](#) amelyen találhatunk helyettesítő karaktereket, amiket majd felhasználhatunk a programunkban. A struktúránkban deklarálunk kell egy karaktert (én esetben ez most a c lesz) és azt is meg kell adnunk, hogy az adott karaktert mire tudjuk cserélni majd. A programnak viszont megengedjük, hogy magának válassza ki a négy lehetőség közül, hogy melyik karakterre cserélje, nem pedig mi mondjuk meg neki, hogy mire cserélje az adott betűket vagy számokat.

Felmerülhet továbbá az a kérdés, hogy hogyan tudjuk kezelni a kis- és nagybetűket hogy ez ne legyen probléma?

A `tolower()` függvényt használva nem lesz ilyen gondunk. Ugyanis ez átalakítja az összes nagybetűnket kisbetűvé, így már nem ütközhetünk emiatt problémába.

3.6. A források olvasása

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

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

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



Bugok

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

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása: [Signal](#)

Megoldás video:

Mi ennek a kódnak a feladata/lényege?

A parancsul adott signal-t kapja el ami a jelen esetben a kettes tehát a Ctrl+C, ami ugye alap esetben "bezárna" programunk futását.

Ahhoz hogy signalokkal foglalkozzunk a kódunkban includeolnunk kell a signal.h könyvtárat

```
#include <signal.h>
```

Megtudjuk még azt is tenni hogy elkapáskor csináljon valamit, például a forrásomban azt írja ki hogy "Elkapva 2" ahol a 2 a signalnak a száma.

```
printf("Elkapva! %d\n", sig);
```

Fontos azt is megjegyezni hogy nem a program lesz "megállíthatatlan" csak a Ctrl+C signalt kapja el, tehát például a Ctrl+Z segítségével a programunk megfog állni ugyan úgy mint eddig.

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

Minden x-re igaz az hogy van olyan y, hogy y nagyobb mint x és y prím. Tehát a prímek száma végtelen.

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

Minden x-ra igaz az hogy létezik olyan y, hogy y nagyobb mint x és y prímszám, valamint y rákövetkezőjének a rákövetkezője is prímszám. Tehát az ikerprímek száma végtelen sok.

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

Létezik olyan y hogy minden x-re igaz az, hogy ha x prím akkor kisebb mint y. Tehát minden prímszám esetén találunk olyan számot mely nagyobb mint Ő, tehát a prímszámok száma véges sok.

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

Létezik olyan y amely minden x-ra igaz az hogy ha x nagyobb mint y és x nem prímszám. Tehát van olyan szám amitől minden nem prím szám nagyobb, azaz nincs olyan szám amitől nem lenne nagyobb nem prímszám, ha y prím lenne akkor az előzővel ekvivalens lenne a megoldás.

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

A feladat megoldása számomra nem okozott gondot, mivel első félévben tanultam logikát mint tantárgyat, de azok számára akik nincsenek tisztában ezzel a tantárggyal és az ilyen típusú feladatokkal, azok számára egy érdekes és elgondolkodtató feladat lehet ez.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

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

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

Megoldás videó:

Megoldás forrása: [Deklaráció](#)

egész

egészre mutató mutató

egész referenciajá

egészek tömbje

egészek tömbjének referenciajá (nem az első elemé)

egészre mutató mutatók tömbje

egészre mutató mutatót visszaadó függvény

egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\* háromszögmátrix

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

Megoldás videó: <https://youtu.be/1MRTuKwRsB0>, <https://youtu.be/RKbX5-EWpzA>.

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/Caesar/tm.c](https://bhax.io/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c)

Bátai Norbert Tanár úr kódját fogom bemutatni, valamint elemzeni.

Először is includeolni kell a szükséges könyvtárakat ami a szokásos stdio könyvtár lesz valamint mellette az stdlib könyvtár is kell ugyanis a mallocot majd így érhetjük el.

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

Majd mainen belül az első amit teszünk az az hogy deklarálunk egy integer típusú "nr" változót amelyben majd az alsó háromszög mátrixunknak a sorainak a számát fogjuk tárolni. A forrásban ez 5 lesz, tehát 5 soros lesz az alsó háromszög mátrixunk, tehát 15 értéke lesz.

```
int nr = 5;
```

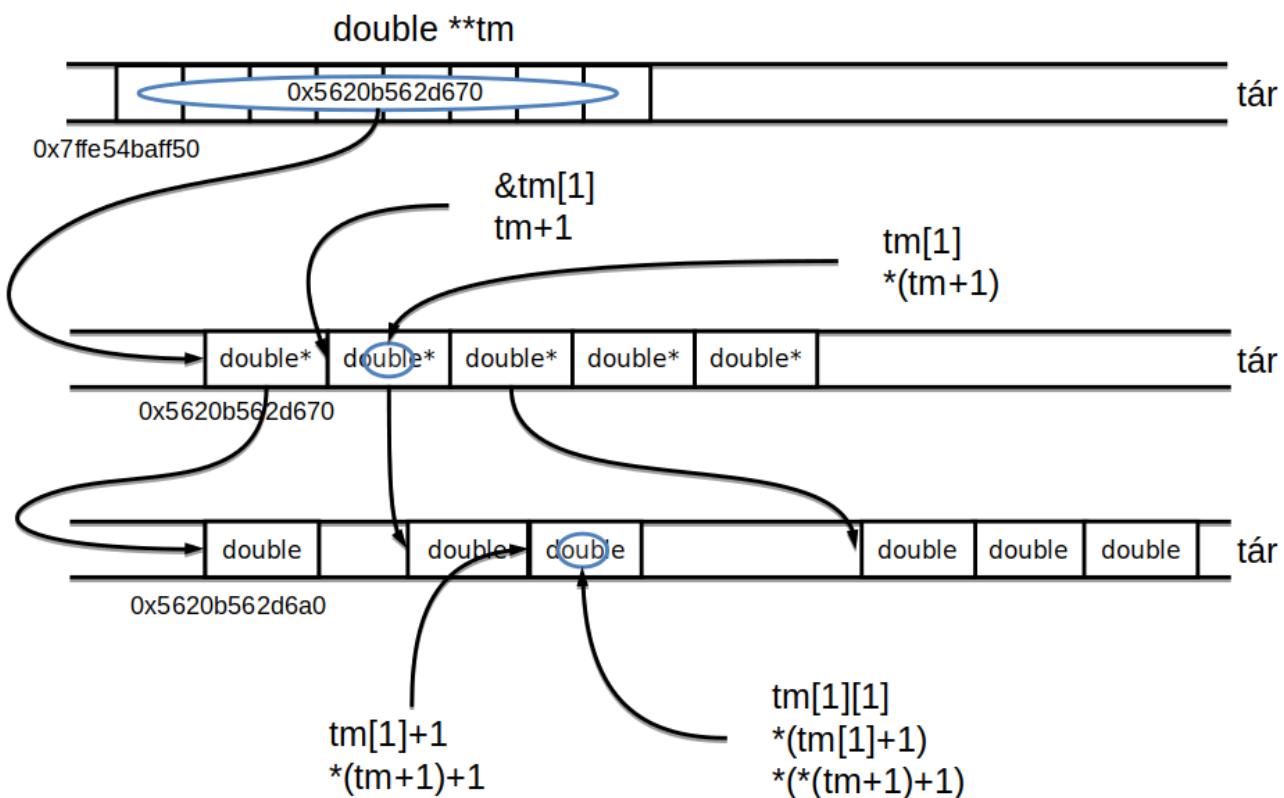
Valamint emellett még deklaráljuk a double \*\*tm-et is. Valamint a tm-nek az aktuális memória címét kiíratjuk.

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

A man 3 malloccal megtudjuk nézni hogy a malloc mit ad vissza. Láthatjuk hogy visszaad egy pointert, ha valami hibát észlel akkor pedig NULL-t add vissza, így ha azzal egyenlő akkor return -1-elünk így a hiba esetén "kilök" a program.

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

Ezzel ráállítjuk a tm mutatót a memóriában malloc által lefoglalt terültre, persze ha az tudott foglalni, értelem szerűen ha nem tudott a program hibát fog lökni, viszont ha tudott akkor pedig kiírja a lefoglalt tár memóriacímét.



4.1. ábra. Double \*\*

A képet Bátfai Norbert készítette, ami a következő oldalon fellelhető: <https://youtu.be/1MRTuKwRsB0>

A következő kódcsipet pedig, egy for ciklus ami maga a foglalás igazából, ha nr értéke 5 akkor 5-ször fog lefutni és mindeniknek megfelelőnyi bájtot foglal le, első sorban ugye i=0 azaz 1 doublenek 8 bájtnyit, i=1 esetén a második sorban két doublenek 2\*8=16 bájtnyit stb...

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

A legvégén pedig felszabadítjuk a lefoglalt memóriát, ezt a free függvény segítségével tesszük meg valami a for ciklusok használatával, majd magát a tm mutatót is felszabadítjuk a free használatával.

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

free (tm);
```

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: [Titkosító](#)

[Titkosítandó szöveg](#)

[Titkosított szöveg](#)

A titkosító azt csinálja hogy a bemenetként kapott szöveget emberi szem számára olvashatatlaná változtatja még pedig a bitek száma alapján. Ehhez egy úgynevezett kulcsot használ amit szintén a felhasználó ad meg, a kulcs és a törő segítségével pedig az olvashatatlan karakter halmazból újra visszatudjuk majd állítani a szövegünket.

```
#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);

}
}
```

Mint láthatjuk maga a titkosító, nem túl bonyolult és hosszúnak sem mondható egyáltalán. Valamint maga a titkosítás ideje se huzamos, emberek számára szinte azonnali. Ez majd a törésnél már nem lesz megfigyelhető.

### 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: [Java titkosító](#)

Az előző feladat javás megvalósítása szükséges, ez alapjába véve nem lenne annyira nehéz feladat, mert ugyan vannak különbségek C és java között de átírni egy ekkora kódcsípetet nem sok idő ha ismerjük a javát és a c-t is. Számomra azért volt nehezebb feladat ez, mivel nem ismertem a javát, így először is a javával kellett ismerkednem utána a feladat forráskódjával.

Elsőnek is a legszembe tűnőbb az hogy a java objektum-orientált programozási nyelv, így már egy "Hello, World!" megírása is teljesen más mint C-ben, ugyanis classokat kell létrehozni és abban kell deklarálni/kidolgozni minden.

```
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 olvasottBájt = 0;

 while((olvasottBájt =
 bejövőCsatorna.read(buffer)) != -1) {

 for(int i=0; i<olvasottBájt; ++i) {

 buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
 kulcsIndex = (kulcsIndex+1) % kulcs.length;
 }

 kimenőCsatorna.write(buffer, 0, olvasottBájt);
 }
 }

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

```
try {

 new ExorTitkosító(args[0], System.in, System.out);

} catch(java.io.IOException e) {

 e.printStackTrace();

}

}
}
```

Végül, ahhoz hogy futtassuk telepítenünk kell az openjdk-8-at amit a következőképpen tudunk leszedni és feltelepíteni:

```
sudo apt-get install openjdk-8-jdk
```

A futtatáshoz pedig ezt kell begépelni a terminálba:

```
javac ExorTitkosító.java
java ExorTitkosító kulcs (pl:12345678) <tiszta.txt >titkos.txt
```

## 4.4. C EXOR törő

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

Megoldás forrása: [Törő](#)

[Titkosítandó szöveg](#)

[Titkosított szöveg](#)

A törő feladata a fentebb említett titkosító által létrehozott "titkos" szöveg visszafejtése, ugyanis a titkosító által kreált karakter halmaz értelmetlen és olvashatatlan az emberek számára.

A törő kódja már pici bonyolultabb és jóval hosszabb a titkosító társánál. Valamint futási ideje nagyságrendekkel nagyobb a titkosításnál. A törés a fordítás módjától, a szöveg és a kulcs hosszától is nagyban függ. A forrásoknál az általam titkosított szöveget is linkeltem, valamint annak a titkosított verzióját is, ezekben jól megfigyelhető milyen is az átalakítás. Ennek a szövegnek a titkosításának a törése körülbelül 20 percet vett/vesz igénybe ha a következő módon fordítjuk a törőt:

```
gcc -o t t.c
```

És csak körülbelül 3 percbe telik ugyan ez ha így fordítjuk, és azután futtatjuk csak:

```
gcc t.c -O3 -o t -std=c99
```

Ez nagyszerűen szemlélteti hogy a fordítás módja már önmaga is meghatározó tud lenni a törési idő csökkenésére/növelésére.

## 4.5. Neurális OR, AND és EXOR kapu

R

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

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

Ez a feladat az R programozáshoz kapcsolódik megint. Itt építünk fel egy tanulni képes neurális hálót, mely a neuronról kapta nevét, ami az agyunkban lévő sejt, ami az információ feldolgozásáért és kezeléséért felelős. Itt is ugyan ez lesz a helyzet, a feladatban a VAGY az ÉS és a kizárt vagy azaz EXOR-t fogjuk neki úgy mond megtanítani.

A VAGY művelet betanítása lesz az első, ahhoz hogy tudjunk dolgozni a neuralnet könyvtárra lesz szükséünk. A VAGY-nak a működését szemléltetjük az a1 a2 és az OR segítségével, ha az a1 és az a2 is 0 akkor az OR is 0 lesz, tehát ha minden operandus 0 akkor a VAGY logikai művelet is 0-át ad vissza, egyébként pedig 1-et fog vissza adni. Azaz ha legalább az egyik operandus 1 akkor az OR is 1 lesz. (Nyilván ez magába foglalja azt is hogy mind az a1 mind az a2 1 akkor is 1 lesz az OR.)

```
library(neuralnet)

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

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

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

plot(nn.or)

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

Az ÉS művelet is hasonlóképpen betanítható és működtethető, annyi eltéréssel hogy itt akkor fog 1-ét visszaadni a logikai művelet ha minden operandus 1, különben 0-val tér vissza. Tehát ha legalább az egyik operandus 0 akkor az ÉS is 0 lesz. (Ez pedig magába foglalja azt is hogy minden a1 minden a2 0 akkor az ÉS is 0.)

```
library(neuralnet)

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

orand.data <- data.frame(a1, a2, OR, AND)

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

plot(nn.orand)
```

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

Az EXOR már kicsit másabb, eleinte mivel nem működött megfelelően fel is hagytak a neurális hálókkal, mert úgy voltak vele az emberek hogy ha egy ilyen egyszerű és sűrűn használt logikai műveletre nem lehet megtanítani akkor nem is annyira érdemes használni. Persze azóta megfejtették hogy mi is lehet a hiba, és most már képesek lehetünk neki megtanítani az EXOR azaz a kizáró vagy műveletet. Mindössze létre kell hozni rejtett neuronokat amelyek segítik a tanulásban.

```
library(neuralnet)

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

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

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ←
 output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

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

Az ábra igaz bonyolultabbá válik a rejtett neuronok miatt, viszont csak így érhetjük el a helyes eredményeket, te hát a megfelelő helyeken az 1-et és a 0-át, ha nem használnánk rejtett neuronokat és a VAGY vagy az ÉS logikai művelthez hasonlóan próbálnánk megoldani a tanítását, olyan 0,5 körüli értékeket kapnánk ami számunkra nem megfelelő. De nézzük meg a "hibás" forráskódot is:

```
library(neuralnet)

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

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

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

plot(nn.exor)

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

Ez az ami nem működne megfelelően, de hála a rejtett neuronoknak már az EXOR művelet megtanítása se egy lehetetlen dolog. Az utolsó "hibás" kódot tehát el is felejthetjük.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: <https://youtu.be/XpBnR31BRJY>

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

A feladat értelmezésében és megoldásában tutoráltam Nagy Krisztiánt!

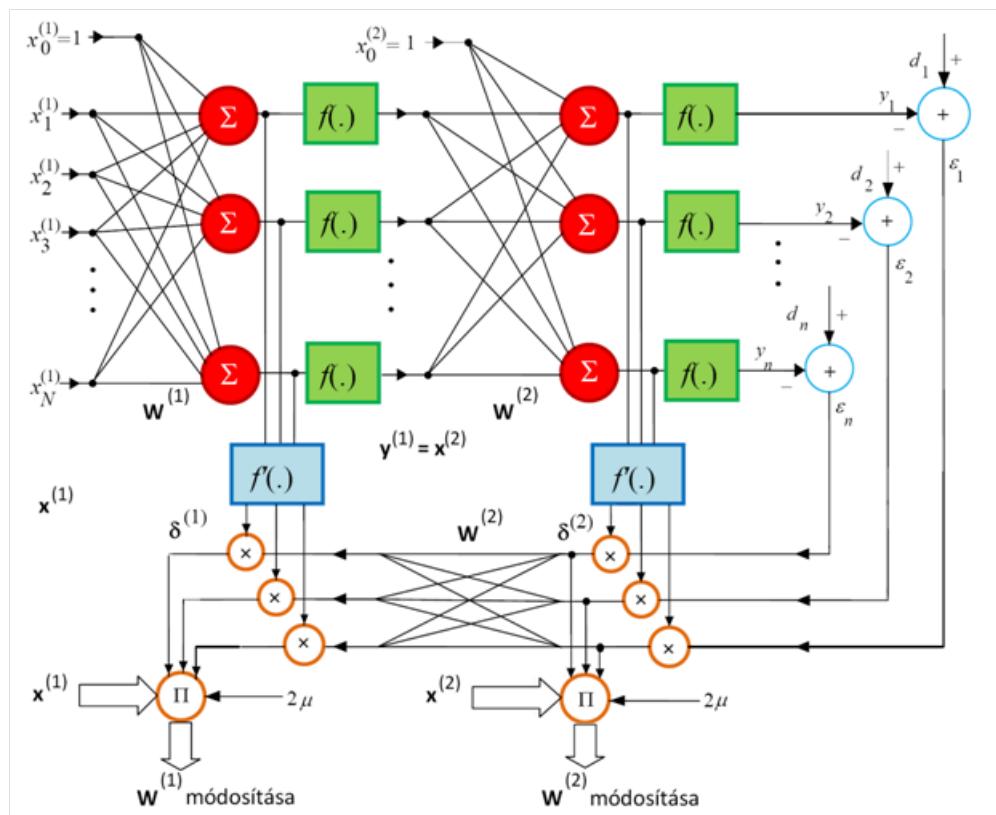
A perceptron egy algoritmus ami a gépi tanulásban játszik fontos szerepet. A bináris osztályzók tanulásában is fontos szerepet játszanak, ugyanis ezek munkaköre az hogy eltudja dönteneti/el is dönti, hogy az input specifikus osztályhoz tartozik-e vagy sem. A perceptronról bővebben angolul olvashatunk a [Wikipédia oldalán](#).

Továbbá még kiemelném a perceptron felépítését, mely 3 fő részből áll:

- A retinának nevezett első elem, ami a bemeneti jeleket fogadó cellákat tartalmazza.
- Az asszociatív cellák, melyek összegzik a hozzájuk érkező jeleket, impulzusokat.
- A döntési cellák rétege a perceptronok kimenetele. Az asszociatív cellákhoz hasonló képpen működnek ezek is.

Részletesebben erről és egyéb perceptron információról olvashatunk magyar nyelven a [következő oldalon](#).

A hibavisszaterjesztéses algoritmus az MLP architektúra alapján származtható. Maga az algoritmus egy tanuló eljárás. Az MLP tanítása az algoritmussal egy szemléletes képen:



4.2. ábra. MLP tanítása hibavisszaterjesztéses algoritmussal

A képet a következő oldalról szűrtam be: <http://mialmanach.mit.bme.hu/neuralis/ch04s02>

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása: [bhax/attention\\_raising/CUDA/mandelpngt.cpp](https://github.com/bhax/attention_raising/blob/main/mandelpngt.cpp) nevű állománya.

A Mandelbrot halmaz Benoit Mandelbrot nevéhez fűződik, aki még 1980-ban fedezte fel azt a komplex számsíkon, akkor el is neveztek róla.

De mit is nevezünk komplex számoknak?

Azok a számok, amelyek "nem létező számok" úgynevezett imaginárius azaz "képzeletbeli" számok. Ezekre azért van szükségünk hogy kitudjuk fejezni magunkat olyan helyzetben is amire nem létezik valós megoldás. Mint például a páros gyökkitevőjű gyök alatti negatív szám értéke. Ugyanis még középiskolában is azt tanítják, hogy a páros gyökkitevőjű gyök alatti negatív számot nem tudjuk értelmezni, de egyetemen már a komplex számok bevezetésével ezeket is tudjuk értelmezni.

Hogyan láthatjuk meg a Mandelbrot halmazt?

Hát úgy, hogy az origó középpontú négyzetbe lefektetünk egy négyzet rácsot például 600x600 vagy 800x800 stb. Valamint ezután kiszámoljuk, hogy a lefektetett rács pontjai mely komplex számoknak felelnek meg. A  $z_{n+1} = z_n^2 + c$ , ( $0 \leq n$ ) képletet felhasználva kapjuk majd meg, még pedig úgy hogy a  $c$  lesz a képletben a vizsgált rácspont, míg a  $z_0$  lesz az origó. Alkalmazva a képletet a továbbiakban:

- $z_0 = 0$
- $z_1 = 0^2 + c = c$
- $z_2 = c^2 + c$
- $z_3 = (c^2 + c)^2 + c$
- $z_4 = ((c^2 + c)^2 + c)^2 + c$
- ... s így tovább.

Vagyis a Mandelbrot halmaz megkeresése és kirajzolása úgy működik hogy kiindulunk az origóból ( $z_0$ -ból) és onnan ugrunk a rács első pontjába a  $z_1 = c$ -be, aztán ettől a  $c$ -től függően a további  $z$ -kbe fogunk ugrani és azokat is vizsgáljuk. Mind addig ugrálunk míg ki nem érünk a 2 sugarú körből, ha ez megtörténik akkor a vizsgált rácspont nem lesz eleme a Mandelbrot halmaznak. Mivel végtelenségig nem tudjuk megvizsgálni ezért csak véges számú  $z$  elemet nézünk meg minden rácsponthoz. Viszont ha ez idő alatt nem lép ki a körből akkor feketére színezzük a rácspontot, ezzel jelezve azt hogy az a  $c$  rácspont a Mandelbrot halmaz része. Aztán lépünk tovább a következő rácspontra majd a következőre és így tovább. Végeredményként a Mandelbrot halmaz elemei fekete színnel lesznek jelölve.

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

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása:

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhax/attention\\_raising/Mandelbrot/3.1.2.cpp](#) nevű állománya.

Egyik legfontosabb különbség az hogy c++-ban tudjuk includeolni a complex könyvtárat ami nagyban megkönnyíti a munkánkat ugyanis biztosít nekünk complex típust így nem kell komplex számot "tartanunk". De nézzük is meg a kódot!

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

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

 int szelesseg = 1920;
 int magassag = 1080;
 int iteraciosHatar = 255;
 double a = -1.9;
 double b = 0.7;
 double c = -1.3;
 double d = 1.3;

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

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

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

double dx = (b - a) / szelesseg;
double dy = (d - c) / magassag;
double reC, imC, reZ, imZ;
int iteracio = 0;

std::cout << "Számítás\n";

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

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

 // c = (reC, imC) a halo racspontjainak
 // megfelelő komplex szám

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

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

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

 ++iteracio;
 }

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

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

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

Ehhez a feladathoz én Bátfai Norbert Tanár úr kódját használtam és értelmeztem. Szerintem a kommentelésből és a videóból könnyen megérthető hogy mit is csinál a program. Valamint az előző feladat alapján már nem fog nehezünkre esni a kód értelmezése. Hisz igazából ugyan az pár kisebb nagyobb különbséggel.

A CUDA-s verzióhoz képest ez annyival másabb (főként), hogy ez csak egy magiszálon fut míg a CUDA-s megvalósításnál felossza rácsra, és minden rácsban vannak blokkok, és a blokkokon belül van például 100 szál, ahol végig egyszerre fognak dolgozni, ezáltal sokkal, de sokkal gyorsabb lesz maga a folyamat a CUDA-s megoldásban.

A CUDA-s megvalósításról későbbi feladatban részletesebben fogunk beszélni! Így ha még nem világos mi is az a CUDA vagy milyen a CUDA-s megoldás nem kell kétségebe esni.

## 5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbqRzY76E>

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

A biomorfokra rátaláló Clifford Pickover meg volt győződve hogy természeti törvényre bukkant. 1986-ban volt ez, akkor valóban természeti formáknak tűntek ezek, ezért is "bio"morfok.

Az előző kódhoz hasonló lesz ez is, egy két kisebb-nagyobb változtatás kell csak minden össze.

Mint például a következő kódcsípetben jól látható hogy az előző kód a,b,c,d változói itt is jelen vannak csak xmin, xmax, ymin és ymax néven:

Valamint még hozzá adunk egy "reC", egy "imC" és egy R változót is, így már az "argc==9" helyett is "argc==12"-öt írunk.

```
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]);

}
```

Az előzőek említett 3 új változó közül kettő a komplexikáló függvénynek a paramétere fog majd lenni, a következő féleképpen:

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

A változtatások miatt a számláló for ciklusunkat is kicsit át kell írni és a beléágazott while ciklusunkat is átírjuk for ciklusra, ahol hasznosítjuk a 3. új változónkat az R-t, ez a következő képpen fog kinézni a kódunkban:

```
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));
}
```

Ha  $z_n$  valós vagy imaginárius része nagyobb mint  $R$  akkor az iteráció a ciklus i változója lesz.

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: [bhax/attention\\_raising/CUDA/mandelpngc\\_60x60\\_100.cu](bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu) nevű állománya.

Ehhez a feladathoz fontos, hogy milyen hardverünk van, ugyanis ehhez a feladathoz nvidia kártyára lesz szükségünk, mivel csak azok a videókártyák rendelkeznek Cudacorral.

A CUDA kártya ellentétben a fenti feladattal, Ő nem CPU-n dolgozik hanem GPU-n (a GPU=Graphics processing unit azaz grafikai processzor/feldolgozó egység). Olyan műveleteket is megtud csinálni a CPU helyett manapság mint például a videókódolás, és nem csak hogy áttudja venni, de sokkal gyorsabban végzi is el azt. Ugyanis ahogy már a fentebbi feladatban említettem a CUDA-s megvalósításnál felossza rácsokra, és minden rácson vannak blokkok, amelyeken belül van például 100 szál, ahol végig egyszerre

fognak dolgozni. Azaz szinte mondható az hogy külön pontonként/pixelenként fog számolni, persze ez nem teljesen igaz, de egy erős túlzással állíthatjuk ezt.

Hogyan tudjuk fordítani?

nvcc fájlnév (pl. mandelpngc\_60x60\_100.cu) -lpng16 -O3 -o kimeneti fájlnév (például: mandelpngc)

Ezután futtathatjuk is, érdemes a 2. feladat forrásával futtatni, és megnézni a két forrás futása közti különbséget. Mit fogunk tapasztalni?

Azt tapasztaljuk hogy többszörösen gyorsabb a CUDA-s megoldás, a cuDecore-tól is számít persze ez, de 50-100x-os sebességgel végzi el ugyan azt amit a 2. feladat.

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

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

Megoldás videó: Illetve [https://bhaxor.blog.hu/2018/09/02/ismerkedes\\_a\\_mandelbrot\\_halmazzal](https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal).

Megoldás forrása: [https://bhaxor.blog.hu/2018/09/02/ismerkedes\\_a\\_mandelbrot\\_halmazzal](https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal)

Már a nevéből is látható hogy ez bizony valamit nagyítani fog nekünk, de hogy és mit?

Először is a libqt4-dev csomagra lesz szükségünk, ennek telepítése után tudjuk csak elvégezni a feladatot.

Ha viszont ezzel rendelkezünk és egy mappában van az összes szükséges fájl, akkor a qmake parancssal készíthetünk egy makefile-t, amit a make parancssal lefuttathatunk. Majd a programunkat tudjuk futtatni a ./Frak prancsal.

Az eredmény frakablakok lesznek, amelyeken különféle fraktált alakzatok láthatóak, köztük kinagyított képek az eredeti halmazból, innen a nagyító kifejezés.

Továbbá meg lehet azt oldani hogy egy bizonyos gomb lenyomásakor nagyítson a képünkön, ezáltal még részletesebben lássuk a kirajzolódást.

## 5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJnssY>, 4:27-től. Illetve [https://bhaxor.blog.hu/2018/09/02/ismerkedes\\_a\\_mandelbrot\\_halmazzal](https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal)

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518>

Alapvetően ugyan az mint az előző feladat, annyi különbséggel ugye bár hogy más a programozási nyelv így nem árt a kódot átírni/újraírni különben nem is működne az.

Először is szükségünk lesz a jdk8 csomagra a javás verzió elkészítéséhez és használatához, ezt a következőképpen tudjuk beszerezni:

```
sudo apt-get install openjdk-8-jdk
```

A javában include-álni/importolni máshogyan kell mint c++-ban, itt jelen esetben az extend-del fogunk. A MandelbrotHalmaz.javát úgy tudjuk használni a kódunkban ha azt a következőképpen importáljuk bele:

```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz{...}
```

Az előző feladatban már említett nagyítást szintén itt is be bindelhetjük valamelyik gombra, annak lenyomása után pedig egy "közelebbi" részletesebb képet kaphatunk a kirajzolt alakzatunkról.

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: [Polargen C++-ba](#), [Polargen Javába](#)

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

Ehhez a feladathoz szükségünk lesz a randomizálásra. A randomizálást c++ a következőképp tudjuk alkalmazni, először is a cstdlib mellett szükségünk van a ctime-ra is, tehát includeolni kell ezt, ugyanis az idő függvényében tudunk teljesen random számot készíteni az srandom által, különben a program minden ugyan azt a random számot generálná, így viszont a futtatás ideje alapján minden újat és újat.

A kódban látható hogy folyamatosan figyeljük a tárolt számokat is, ezáltal azt is elkerüljük ha véletelenül már szerepelne a random számunk.

A matematikai háttér jelenleg nem érdekel minket, legalábbis a feladat is megkér arra hogy erre most ne térd ki, tehát nem túl részletesen csak lényegre törően azt csinálja a programunk, hogy ha nincs eltárolva a randomizált számunk akkor lefut az algoritmus és vissza ad egy random számot két különböző változóba. Ahonnan az egyik ki lesz íratva a másik pedig el lesz tárolva. Amennyiben el van tárolva egyszerűen kiíratjuk.

Valamint azt is megfigyelhetjük, ami a feladatban szerepel, tehát a JDK forrásaiban is hasonó felépítéssel vannak meg a kódok a miénkhez, tehát egy tapasztalt programozó is úgy oldaná meg ezt mint mi a jelenlegi tudásunkkal. Tehát mondhatni hogy ha ezt értjük akkor ugyanolyan természetes ez nekünk mint a Sun programozóinak.

### 6.2. LZW

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

Megoldás videó:

Megoldás forrása: [C Binfa](#)

Az LZW (Lempel-Ziv-Welch nevéhez köthető, innen ered) egy veszteségmentes tömörítési algoritmus. A tömörítés alapja az hogy a kódoló csak egy szótábeli indexet küld át. Erről részletesebben magyar nyelven olvashatunk a [Wikipédia oldalán](#).

A bináris fa egy olyan adatszerkezet amely belső és külső csúcsok hierarchikus elrendezésből áll, ezzel megalkotva a szülő-gyermekek elrendezést, mint az általános fa adatszerkezet. Fontos viszont kiemelni hogy minden "szülőnek" maximuma két gyermek lehet, ez az amivel eltér a többi fa adatszerkezettől, valamint a gyermeket megkülönböztetni úgy tudjuk a fában hogy bal vagy jobb oldalákat képezik-e a szülőnek. Tehát fontos a sorrendjük is, bejárásra több példát is fogunk látni a következő feladatokban, ott majd részletesebben megismерkedünk a bejárás fogalmával és mintjével.

Hogyan tudjuk fordítani a binfát?

Egyszerűen annyi mint bármelyik másik c forráskódot, tehát valahogy így:

```
g++ z.c -o z
```

Hogyan tudjuk futtatni?

A futtatás se különbözik igazán, minden össze megkell neki adni egy bemeneti és kimeneti fájlt is, valahogy így:

```
./z bemenetifájl -o kimeneteifájl
```

## 6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: [Preorder bejárás](#), [Postorder bejárás](#)

Az alap, tehát a fentebbi forráskód szerinti inorder bejárás:

```
void
kiir (BINFA_PTR elem)
{
 if (elem != NULL)
 {
 ++melyseg;
 _melyseif (melyseg > maxg);
 max_melyseg = melyseg;
 kiir (elem->jobb_egy);
 // ez a postorder bejáráshoz képest
 // 1-el nagyobb mélység, ezért -1
 for (int i = 0; i < melyseg; ++i)
 printf ("---");
 printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↔ ,
```

```
 melyseg - 1);
 kiir (elem->bal_nulla);
 --melyseg;
}
}
```

A bejárás megváltoztatása a forróskódban egy egyszerű sor kicséréssel valósítható meg, ugyanis a program lineárisan halad a végrehajtáson és ha előre rakjuk pl a gyökér elem vizsgálatát és csak utána a bal, s majd jobb ág vizsgálatát már is preorderben lesz a bejárás. Ha végére rakjuk a gyökér elem vizsgálatát nyilván posztorder bejárást fog eredményezni.

Preorder bejárás:

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

 // ez a postorder bejáráshoz képest
 // 1-el nagyobb mélység, ezért -1
 for (int i = 0; i < melyseg; ++i)
 printf ("---");
 printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↔
 ,
 melyseg - 1);
 kiir (elem->bal_nulla);
 kiir (elem->jobb_egy);
--melyseg;
 }
}
```

És posztorder bejárással is:

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

 kiir (elem->jobb_egy);
 kiir (elem->bal_nulla);
 // ez a postorder bejáráshoz képest
 // 1-el nagyobb mélység, ezért -1
 for (int i = 0; i < melyseg; ++i)
```

```
printf ("---");
 printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
 ,
 melyseg - 1);
--melyseg;
}
}
```

A név nem véletlenszerű hiszen az angol szavakból ered, "pre" mint elől "post" mint hátul és "in" mint benne/középen, és a gyök elem helyzetét árulja el a bejárás során.

## 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: [Tag a gyökér](#)

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

A kód alapján jól látható hogy nem mutató a gyökér (mint ahogy az a következő feladatban szerepelni fog), hanem a feladat leírása szerint egy Tree és egy beágazott Csomópont osztály.

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

class LZWBinFa
{
public:

 LZWBinFa ()
 {
 gyoker = new Csomopont;
 gyoker = fa;
 }
//~LZWBinFa.....
//a Csomopont class:
private:
 class Csomopont
 {
public:
 Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
 { };
```

```
~Csomopont ()
{
 { return balNulla; }
 { return jobbEgy; }
 { balNulla = gy; }
 { jobbEgy = gy; }
 char getBetu () const
 { return betu; }

private:
 char betu;
 Csomopont *balNulla;
 Csomopont *jobbEgy;
 Csomopont (const Csomopont &);
 Csomopont & operator= (const Csomopont &);

};

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

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

void szabadit (Csmopont * elem)
{
 if (elem != NULL)
 {
 szabadit (elem->egyesGyermek ());
 szabadit (elem->>nullasGyermek ());
 delete elem;
 delete Csmopont;
 }
}
```

## 6.5. Mutató a gyökér

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

Megoldás videó:

A feladat megoldásában tutoriáltam: Győri Márk Patrikot.

Megoldás forrása: [Mutató a gyökér](#)

Első lépésként át kell írnunk a gyökeret hogy mutató legyen, ehez minden össze annyit kell tennünk hogy csillagot teszünk elé.

```
Csomopont *gyoker;
```

Ezután ha megpróbáljuk fordítani láthatjuk hogy kismilliónyi hibát jelez, ezeket kell javítanunk.

Kezdjük már is a legelején, a konstruktort írjuk át valahogyan így:

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

Ezt ha megléptük akkor sikeresen helyet foglaltunk a memóriában a csomópontnak amire a gyökér mutat, továbbá a fa mutatót is ráállítottuk a gyökérre.

Mivel már pointer a gyökér ezért a szabadításnál ' .' helyett ' ->' operátort kell alkalmaznunk, írjuk hát át ezt is!

```
{
 szabadit (gyoker ->egyesGyermekek());
 szabadit (gyoker ->nullasGyermekek());
}
```

Valamint, szintén a gyökér pointer léte miatt a kódban szereplő

```
"&gyoker"
```

kifejezéseket is át kell írnunk simán "gyoker"-re, ugyanis nekünk nem a pointer címére lesz szükségünk sehol hanem arra a címre amire mutat, tegyük meg hát ezeket a lépéseket is!

## 6.6. Mozgató szemantika

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

Megoldás videó:

A feladat megoldásában tutoriált engem: Győri Márk Patrik

Megoldás forrása: [Mozgató szemantika](#)

```
LZWBinFa & operator= (const LZWBinFa & cp) {
 if(&cp != this)
 rekurzioIndutasa(cp.gyoker);
 return *this;
};
```

Ehhez a feladathoz azt kell tennünk hogy létrehozunk egy operátort, aminek az a feladata hogy másolja le magát ha nem a gyökeret tartalmazza. Valamint a másolás rekurzióval végződik, mely azt fogja tenni hogy a fa minden egyes ágát újra létrehozza viszont egy másik gyökérre. A kód csípetekből jól fog látszani, viszont a kódott a tutoromtól (Győri Márk Patriktól) kaptam, nem pedig saját.

```
void rekurzioIndutasa(Csomopont csm) {
 if(csm.nullasGyermek()) {
 fa = &gyoker;
 Csomopont *uj = new Csomopont ('0');
 fa->ujNullasGyermek (uj);
 fa = fa->nullasGyermek();
 std::cout << "GYOKER: nullas van" << std::endl;
 rekurzioAzAgakon(csm.nullasGyermek());
 }
 if(csm.egyesGyermek()) {
 fa = &gyoker;
 Csomopont *uj = new Csomopont ('1');
 fa->ujEgyesGyermek (uj);
 fa = fa->egyesGyermek();
 std::cout << "GYOKER: egyes van" << std::endl;
 rekurzioAzAgakon(csm.egyesGyermek());
 }
}

void rekurzioAzAgakon(Csomopont * csm) {
 if (csm->nullasGyermek()) {
 std::cout << "====van nullas" << std::endl;
 Csomopont *uj = new Csomopont ('0');
 fa->ujNullasGyermek(uj);
 }
 if (csm->egyesGyermek()) {
 std::cout << "====van egyes" << std::endl;
 Csomopont *uj = new Csomopont ('1');
 fa->ujEgyesGyermek(uj);
 }
 Csomopont * nullas = fa->nullasGyermek();
 Csomopont * egyes = fa->egyesGyermek();
 if(nullas) {
 fa = nullas;
 rekurzioAzAgakon(csm->nullasGyermek());
 }
 if(egyes) {
 fa = egyes;
 rekurzioAzAgakon(csm->egyesGyermek());
```

```
}
```

```
}
```

A rekurzioInditasa függvény fogja elindítani a rekurziót, ha van nullás gyermeké akkor azon fog elsőként tovább futni, majd ha van egyes gyermeké akkor arra is meghívásra fog kerülni. A fő eljárást maga a rekurzioAzAgakon függvény fogja elvégezni nekünk, ez fog átfutni az összes ágon, és majd Ő fogja létrehozni az új csomópontokat.

```
LZWBinFa binFa2;
 binFa2 = binFa;
```

A másolás az egyenlőség jel operátorral meghívva történik, így az alap binFa átmásolódik a binFa2-be.

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyszimulációk

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

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

Megoldás forrása:

[Hangya osztály](#)

[AntWin osztály](#)

[AntWin](#)

[Antthread osztály](#)

[Antthread](#)

[Main](#)

A hangya osztályban a hangyának a tulajdonságai vannak. A hangyának van oszlopa és sora ami a kódban x és y, valamint a kódban lévő dir ami a hangya irányát fogja képezni. Ezekből fogja tudni a program hogy hol van és merre tart a "hangyánk".

```
class Ant
{
public:
 int x;
 int y;
 int dir;

 Ant(int x, int y) : x(x), y(y) {
 dir = qrand() % 8;
 }
};
```

Nem pixelekkel dolgozik a program hanem cellákkal. Alapértelmezetten 6x6 pixeles egy cella.

```
cellWidth = 6;
cellHeight = 6;
```

Az AntWinbe van az AntThread osztályra mutató. A grid int\*\*\* a két rácsra mutat. Azért van két rácsunk hogy minden hangya egyszerre lépjen ehhez majd szükségünk lesz a gridIdx-re. Valamint még a feromonnak a maximum és minimum értékét tartalamazza az AntWin. Az AntWinnek van konstruktur és destrukturja is, valamint van close esetnje ami a bezáráshoz kell, és keypresseventje ami a pauseolásra képes, ami a kódban P-re van bindelve.

## 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Ehhez a feladathoz is szükségünk lesz a jdk8-ra, de ezzel már rendelkezünk, korábbi fejezetben ennek beszerzése levan írva, ugyanis már kellett használnunk javás feladathoz.

Az életjáték elvet John Horton Conway nevéhez kötjük. Aki 1970-ben létrehozta saját sejtautomatáját, aminek megadhatunk bizonyos feltételeket az életbe maradáshoz. Így egy fajta életet hozunk létre, amit megfigyelhetünk, ugyanis dimanikusan változik, kihalnak, születnek újak, vannak olyanok is amelyek stagnálnak. És a legritkább az amikor egyfajta ismétlődéses stagnálást eredményeznek a csoportulásuk ezeknek a kis "élőlényeknek".

Több szabályrendszer is létezik, a mi példákban Conway 3 szabályát feldolgozó automatát használjuk. Ennek szabályai a következők:

1.szabály: Csak 2 vagy 3 szomszéddal rendelkező sejtek maradnak életben.

2.szabály: Ha egy szomszédnak 3+ szomszédja van túlnépesedés miatt kihal. Valamint ha 2-nél kevesebb akkor pedig szintén meghal, magányosság miatt.

3.szabály: Megszületik egy sejt ha üres a cella és 3 élő sejt szomszédja van a cellának.

Ezen szabályok mellett az egyik legjobb és legérdekesebb eset az úgynevezett "siklókilövő" kódját linkelem ide, a feladathoz hűen java nyelven, majd a következő feladatban szintén ez megtalálható lesz c++ nyelven is. Valamint a következő feladatban a program futásakor készített kép is megtalálható lesz.

Forráskód: [Siklókilövő](#)

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: [Sejtablak](#), [Sejtablak header](#)

A feladat ugyan az mint az előző csak c++-ban. Itt is szintén a Conway 3-as szabályrendszer fogjuk alkalmazni, ezáltal a szabályok ugyanazok, tehát a születés az életbenmaradás és a kihalás esete ugyan úgy fordul elő mint előbbiekbén.

Így például egy 2x2-es cella az élni fog és meg is marad nem fog változni se, míg egy 1x3-as cella pedig 3x1 lesz majd megint 1x3 és így tovább, viszont nem hal meg az egész alakzat.

Hogy is néz ki ez a cellákból álló ablak?

Valahogyan így:



7.1. ábra. Sejtablak

A képet a következő oldalon fellelhetjük, valamint láthatunk még további állapotairól képeket a "siklókilővőnek". Az oldal: [https://progater.blog.hu/2011/03/03/fegyvert\\_a\\_nepnek](https://progater.blog.hu/2011/03/03/fegyvert_a_nepnek)

Maga a forrás pedig úgy működik hogy ilyen "siklókat" hozz létre amik folyamatosan haladnak és betöltik a rendelkezésre álló teret, ha ez a tér véges, akkor addig fognak haladni hogy a "szülők"-ig érnek azok pedig így meghalnak, de a siklók örökre megmaradnak és haladnak.

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

<https://github.com/nbatfai/SamuBrain>

A játékos feladata az hogy a Samu Entropy-n belül lévő kék körben kell a kurzort lenyomva tartani, ez eleinte nem bonyolult, de minél tovább tartjuk rajta annál jobban elkezdenek mozogni a pontok, így követni hol jár egyre nehezebbé válik. A konzolon kiírja ha hibázunk valamint folyamatosan méri hogy mennyi bit/sec a reakció időnk. Tehát érdekes egy program, főként az esportban jártasak számára.

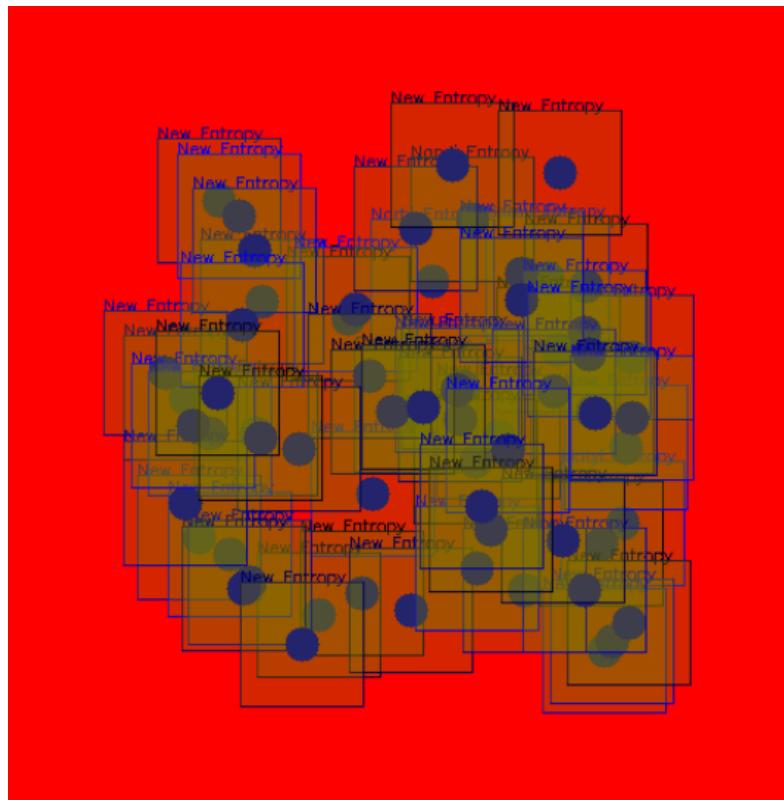
A játék 10 percig fut, de lehetőségünk van előtte is kilépni, és ha hamarabb befejezzük akkor is láthatjuk az eredményeinket.

Fordítani és futtatni úgy tudjuk ha a szükséges fájlok minden egy mappába vannak és a szokásos qmake-s eljárást kell alkalmaznunk itt is. Valahogyan így:

```
qmake BrainB.pro--
make--
. / BrainB --
```

Eleinte kevesebb "ablakkal" kezd, az idő múltával lesz egyre több.

Kezelő felületet így néz ki futás közben:



7.2. ábra. BrainB Benchmark

A kép egy régebbi verziót ábrázol de az újabb belinkelt forráskódú "játék"/"benchmark" is ugyanolyan felépítésű. A képet az interneten a következő oldalon lehetjük fel: <https://github.com/nbatfai/esport-talent-search>

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

Python

Megoldás videó: <https://youtu.be/j7f9SkJR3oc>

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa...  
[https://progpater.blog.hu/2016/11/13/hello\\_samu\\_a\\_tensorflow-bol](https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol)

Forráskód: [Szoftmax](#)

Ebben a feladatban Python nyelvben fogjuk megírni a MNIST-et amivel egy képfelismerő programot készítünk, melynek feladata az lesz hogy felismerje hány pont van a képen. Ehhez viszont szükségünk lesz a Python3 fejlesztő csomagra és a Tensorflowra is.

Hogyan tudjuk telepíteni a Tensorflowt és a Python3 fejlesztői csomagot?

A Python pip package segítségével a következőképpen tehetjük meg ezt:

```
sudo apt update
sudo apt install python3-dev python3-pip
sudo apt-get install python3-matplotlib
sudo pip3 install -U virtualenv
virtualenv --system-site-packages python3
source ./venv/bin/activate
pip install --upgrade pip
pip install --upgrade tensorflow
python -c "import tensorflow as tf; tf.enable_eager_execution(); print(tf. ↵
 reduce_sum(tf.random_normal([1000, 1000])))"
deactivate
```

A MNIST egy adatbázis mely tanítóanyagokat tartalmazz, olyan hálózatok tanítására aminek a feladata valamilyen képfelismeréshez vagy képelemzéshez köthető. Az SMNIST ennek úgynevezett "gyermeke" lett. A Bátfai Tanár úr féle nyílt forráskódú SMNIST-el "játszani" és tesztelgetni szabadon lehetett/kellett. Mi saját SMNIST for human próbálkozást is csináltunk/teszteltünk pár szakon lévő sráccal. Érdekes tud lenni ez, valamint versenyezni is lehet benne. Későbbiekben ez a képfelismerés, arcfelismerés stb gépek számára szerintem nagyobb teret fog behódítani jelenlegi állapotához képest. Ugyanis hasznos funkció lehet akár egy telefonban is. (Már léteznek próbálkozások kihasználására.)

Számomra a Python nyelv és annak használata kevésbé ismert, de más nyelvhez pl C vagy C++-hoz hasonlóan itt is elsőként importáljuk a szükséges könyvtárakat. Ilyen például ebben a feladatban használt Tensorflow könyvtár is.

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

import argparse

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

import matplotlib.pyplot

import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)
```

```
FLAGS = None

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

A modell elkészítése.

```
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(←
 labels = y_, logits = y))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(←
 cross_entropy)

sess = tf.InteractiveSession()
```

A tanítása:

```
tf.initialize_all_variables().run(session=sess)
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("-----")

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("-- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.test. ←
 images,
 y_: mnist.test.labels}))
print("-----")

print("-- A MNIST 42. tesztképenek felismerése, mutatom a számot, a ←
 továbblepéshez 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 MNIST 11. tesztképenek felismerése, mutatom a számot, a ←
 továbblepéshez csukd be az ablakat")

img = mnist.test.images[11]
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, 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()
```

Bonyolult kissé a kód egyrészt mert a hivatalos MNIST példaprogramja az alapja másrészt számomra azért is mert a Python forráskód olvasata még nem megszokott.

## 8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Ezt a feladatot passzolnám!

## 8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndl8>

Megoldás forrása:

[Minecraft-MALMÖ](#)

A Malmo egy platform amely segít nekünk kutatásban a mesterséges intelligencia területén. A feladatban az ágens-programozásba csöppenhetünk bele, melyet a népszerű játékkal a Minecrafttal ötvözve probálhatunk ki. A feladat olyan Ai-t írni Stevenek hogy kikerülje az elé kerülő akadályokat, ezáltal nem csapdába esve haladni a mapon és felfedezni azt.

Tesztelni a mellékelt kódot nem tudtam, ugyanis nem rendelkezem a Minecrafttal, de működnie kell.

Mint mindenhol itt is elsőnek a szükséges könyvtárak importálásával kell kezdenünk:

```
from __future__ import print_function

from builtins import range
import MalmoPython
import os
import sys
import time
```

Majd létre hozunk egy alapértelmezett Malmo objektet:

```
agent_host = MalmoPython.AgentHost()
try:
 agent_host.parse(sys.argv)
except RuntimeError as e:
 print('ERROR:',e)
 print(agent_host.getUsage())
 exit(1)
if agent_host.receivedArgument("help"):
 print(agent_host.getUsage())
 exit(0)
```

```
my_mission = MalmoPython.MissionSpec(missionXML, True)
my_mission_record = MalmoPython.MissionRecordSpec()
```

Majd megkíséreljük a "küldetésünket" elindítani:

```
max_retries = 3
for retry in range(max_retries):
 try:
 agent_host.startMission(my_mission, my_mission_record)
 break
 except RuntimeError as e:
 if retry == max_retries - 1:
 print("Error starting mission:",e)
 exit(1)
 else:
 time.sleep(2)
```

Majd loopolnunk kell addig amíg a mi kis küldetésünk el nem indul:

```
print("Waiting for the mission to start ", end=' ')
world_state = agent_host.getWorldState()
while not world_state.has_mission_begun:
 print(".", end="")
 time.sleep(0.1)
 world_state = agent_host.getWorldState()
 for error in world_state.errors:
 print("Error:",error.text)

print()
print("Mission running ", end=' ')

agent_host.sendCommand("move 1")

stevex = 0
stevez = 0
stevey = 0
steveyaw = 0
stevepitch = 0
elotteidx = 0
elotteidxj = 0
elotteidxb = 0
akadaly = 0
```

Ezután a kód többi része maga az a kódrész ami loopolva lesz, tehát ami majd akkor fut folyamatosan amíg a küldetésünk tart.

```
while world_state.is_mission_running:
 .
 .
 .
#(a forráskód fent szerepel nem csúnyítanám vele a könyvet)
```

```
•
•
•
print()
print("Mission ended")
```

A kódban látható hogy a koordináták segítségével valamint a kód részében az if-ek segítségével tudjuk neki megmondani mikor mit csináljon. Például hogy mikor forduljon vagy mikor kezdjen ugrálni a karakterünk. Ezeknek a finom módosítgatásával tudjuk biztosítani a küldetésünk sikerét, akár különböző biomokon vagy különböző akadályokon való "túlélésen", és leküzdésében.

## 9. fejezet

# Helló, Chaitin!

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

Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

Megoldás forrása:

A feladatot Lisp-ben kell megoldanunk, de hogyan férhetünk hozzá?

Mindössze elég hozzá egy GIMP-et indítani, azon belül pedig nyitni kell egy Script-fu konzolt, és már kezdhetjük is beírni a Lisp-es kódunkat. Na de hogyan is néz ki egy Lisp-es kód?

Először is azt ki kell emelni hogy ezen a nyelven nagyon figyelnünk kell a zárójelezésre, mivel azt könnyű elbénázni, és nagyon fontos mindennek zárójelben kell lennie. De nézzünk is egy egyszerű összeadást Lisp-ben:

```
(+ 2 2)
```

Ez négyet fog nekünk adni. Egy preorder bejárással történt műveletnek tűnik, de egy bonyulultabb egyenletnél láthatjuk hogy nem erről van szó, nézzük is meg!

```
(+ 2 2 2)
```

Ez lesz a Lisp kódban a  $2+2+2$ , míg preorder bejárással  $+ + 2 2 2$  lenne.

Na most hogy kicsit megismertük a Lispet kezdjünk neki a feladatnak!

Először is a faktoriális függvényünket "meg kell tanítanunk". Amit a következő kód beírásával tudunk megtenni:

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

Most már tudja hogy mi a "fakt", így kitudja számolni egy szám faktoriálisát, a példában nézzük meg hogy kell a 4 faktoriálist ezután kiszámolni.

```
(fakt 4)
```

Ez 24-et fog adni, azaz a fakt definiálása és alkalmazása is tökéletes lett ugyanis ez megegyezik  $4!$ -sal.

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

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

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

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

A GIMP script mappájából tudjuk csak a kódunkat működésre bírni, valamint a programon belül is megadjuk elérési helyét ezután már is képesek leszünk kevükk szerint alakítani a színeket a szövegen. Ha futtatjuk a programot akkor kapunk egy default beállítást mindenre de persze ezeken kedvünk szerint módosíthatunk, nem köt minket semmihez. Viszont a feladat köt minket még pedig a króm effektet kell elérnünk.

Na de térijünk a feladatra, első lépésként egy fekete színű háttérrel hozunk létre amin a szövegünk fehér színű lesz. Ezt így érjük el:

```
(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)))
```

Ha ezzel megvagyunk második lépésként elmosssuk a szövegünket, még pedig az úgynevezett Gaussian eljárással.

```
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

3. lépésként a szövegnek az éleit görbítjük le, majd 4. lépésként pedig ezt mossuk el az előbbi módszerrel.

```
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
 (plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

A következő lépésekben lehetőségünk lesz az elején létrehozott fekete háttteret kitörölni a kép invertálása mellett, ami átlátszó lesz tehát maga a kép fog látszani ezáltal, a szöveggel pedig kedvünk szerint foglalkozhatunk, majd az utolsó (a forráskódban 9.) lépés alkalmával lesz lehetőségünk megadni a gradient effektet, amivel sikeresen késznek tudhatjuk a feladatot.

```
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))
```

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

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

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

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

Ezt a feladatot passzolnám!

# 10. fejezet

## Helló, Gutenberg!

### 10.1. Programozási alapfogalmak

[?]

Ezt az olvasónaplót a megadott oldal számok és téma elolvasása alapján írom, alcímként megadom hogy melyik oldalról/melyik fejezet részről szól az adott szakasz. Annak a szakasznak röviden lényegre törően a tartalmát és a vele kapcsolatos egyéb meglátásokat közzök az olvasónaplóban.

#### (11.-15. o.) 1.2 Alapfogalmak:

Az alap fogalmakat foglalja össze, viszont ellentétben a Keringhan-Ritchie könyvvel, itt nem kódokkal és azoknak az elmagyarázásával mutatja be, hanem inkább ilyen elmélet szerűen tanulás nem gyakorlatiasan, tehát a fogalmak pontos korrekt megfogalmazásával. Így már én az első fejezet után is úgy vélem hogy a KernighanRitchie féle oktatás és magyarázás számomra sokkal kedvezőbb és könnyebben elsíjátítható mint ennél a könyvnél, de persze minden embernél, más, ezért főként azoknak ajánlanám ezt a könyvet aki nem a gyakorlatiasabb példákon átívelő tanulást részesíti előnyben. De persze még a végén erre kitérünk haladjunk is tovább.

#### (46.-55. o.) Kifejezések:

A kifejezések azaz a 3. fejezetben, a címből következtetve is helyesen gondolhatjuk azt hogy a kifejezések-ről lesz szó. A kifejezések elméleti tudnivalójával kezd, megtudjuk mit nevezünk például operandusnak, operátornak stb. Valamint még az alakokról is szó esik (prefix,postfix,infix) ugye ezek között minden össze az operátorok és operandusok helyzete a különbség, hogy milyen sorrendbe írjuk őket fel. Azaz ha például egy kifejezés operátorait és operandusait egy bináris fával rajzolnánk fel akkor milyen bejárást alkalmazva jutunk el a felírásokhoz. (értsd úgy hogy preorder=prefix....)

A 3.1 "Kifejezés a C-ben" már inkább gyakorlatiasabb szemléletesebb fejezet rész, itt a címből is addódóan a C-ben lévő kifejezéseket mutatjabe azoknak a helyes használatát, kötései irányát stb.

#### (55.-57. o.) Utasítások:

Ezek az oldalak az utasításokról fognak szólni. Az utasítások építik fel a programot ezért fontos az ismertük. Utasításoknak több fajtája van, mint például értékadó-, ciklusszervező-, hívó-, i/o-, egyéb utasítások. A kijelölt oldalakon ezeknek még nem található részletezése, de a rákövetkező párral oldal taglalja, szerintem érdemes azt is elolvasni hozzá, hisz ott ismerjük meg csak részletesen ezeket az utasításokat valamint ott már forrás kódot is kapunk mellé hogy láthassuk miről is van szó.

**(58.-82. o.) LISP:**

Mint az fentebb említettem érdemes elolvasni, igazából ki is lett jelölve csak másik csokorba. Egészen a 72. oldalig az utasításoknak a részletes szemléltetéséről szól a könyv, én ezt még az előző kijelöléshez soroltam volna. Ugyanis 72. odlalnál a programok szerkezetére tér át a könyv. Ábrás itt-ott forrás kódos módon mutatja be de még mindig inkább elméleti szinten, a paraméterekre is kitér bőségesen, azok megadására és átadására is egyaránt.

**(56.-71. o.)**

Még egyszer fel lett adva, ez a rész összefoglalóan részletesen az utasításokról szól, hasznos és fontos rész a könyvben a két oldalszám közé eső részek.

**(72.-78. o.) és (78.-82. o.)**

Ezek is újra fel lettek adva, nem írnám le még egyszer, fentebb olvasható miről is szól ez az intervallum.

**(82.-85. o.)**

Ez a rész még szintén a programok szerkezetéhez köthető. A blokkal ismerkedünk meg, valamit a hatáskör fogalmát vesszük át. A hatáskör a c++-ban például fontos ugyanis nem mindegy hogy akár egy osztálynál vagy egy function-nél hol deklarálunk, hol használunk függvényeket stb, ezért érdemes megismерkedni vele.

**(112.-113. o.)**

Ez a rész a kivételkezeléssel foglalkozik. Megszakítások kezelését felhozzuk a program szintjére, ennek a részletes ismertetőjét olvashatjuk/tanulmányozhatjuk át ebben a részben.

**(134.-138. o.)**

Ez a rész pedig az I/O-val foglalkozik, tehát az input és output-tal. Azok ismertetésével (input/output állomány), mikor jön létre, mikor szükséges stb. Állományok deklarációjáról, összerendeléséről, megnyitásáról, feldolgozásáról, lezárásáról esik még szó. Majd a 13.1-ben a különböző nyelveknek az I/O eszközeit mutatja be nekünk az író.

**Összesítés:**

Alapjába véve egy hasznos könyv, mely rengeteg információt hordoz magában, de számonra kicsit száraz, szerintem jobb a Kernighan fele könyv mely példákon keresztül tanít. De ennek ellenére ezt is ajánlanám azoknak akik a C nyelvről szeretnének minél több tudást a magukénak érezni.

## 10.2. Programozás bevezetés

**[KERNIGHANRITCHIE]**

Megoldás videó: <https://youtu.be/zmfT9miB-jY>

Én fejezetenként adok egy kis összefoglalót hogy miről is szól az adott fejezet, így teljesíteném az olvasónapló megírását ennél a feladatnál.

**1. Fejezet:**

A könyv az alafogalmakkal és alapismeretekkel indít. Az 1.1-ben a szokásos "Hello, World!" programot írja meg és magyarázza el a "Halló, mindenki" példán keresztül. Majd a változókat és annak típusait nézi át részletesebben az 1.2-ben. Valamint még ugyan itt a kommentekről is olvashatunk továbbá a while

ciklus is megjelenik. Míg az 1.3-ban a for ciklust részletezi a könyv, amit az előző whilehoz hasonlítva mutat be. 1.4-be szimbolikus állandókat mutatja be az olvasóknak a könyv. Az 1.5-ben a karakterekkel foglalkozó függvényekről kapunk egy részletes összefoglalót amely bemutatja többek között a beolvasást, kiírást, sorok/szavak számlálását. Az 1.6 alfejezetben a tömbökről lesz szó, azoknak az alkalmazásáról, a könyv karakter megszámlálása valamint üres hely megszámlálása példáján mutatja be hasznosságukat. Ezután az 1.7-be a függvényeké lesz a főszerep, a hatványozást mutatja be a könyv függvények segítségével. 1.8-ba pedig az argumentumokat taglalja a könyv. A fejezet végén pedig újra, csak azúttal részletesebben még egyszer megnézzük a tömböket és változókat.

## 2. Fejezet:

A 2.fejezet címként (tehát a könyv alcímként) megkapta a "Típusok, operátorok és kifejezések" címet, mely szerintem nagyon lényegretörő, ugyanis valóban ezekről lesz szó. A fejezet nagyon nagy részletességgel taglalja a változónevek, adattípusok és méretek, állandók, aritmetikai operátorok, logikai és relációs operátorok használatát fontosságát. A deklaráció fontoságáról és annak ki nem hagyhatóságáról is ki emeli a 2.4-be. Megismerkedhetünk a típus konverziókkal, valamint különféle operátorokról és kifejezésekről olvashatunk a 2. fejezet második felében, ami nagyon fontos, valamint szerintem a programozás alapjai, ezért érdemes elolvasni figyelmesen.

## 3. Fejezet:

Ez a fejezet a vezérlési szerkezetekkel fog foglalkozni. Az olvasó számára bemutatja a leghasznosabb és legfontosabb utasításokat mint például az if, else if, switch, break, continue, goto. Valamint a ciklusszervezést mutatja meg az elején említett while és for utasítással, valamint a do-while-al ami a névből is láthatóan a whilehoz hasonló felépítésű és működésű, csak egy kisebb változtatással. Még pedig az hogy a do-while egyszer mindenkorban le fog futni, de nem is írok többet, a könyv nagyon jól bemutatja ezt is, szintén egy hasznos és "programozás alapja" fejezetnek mondanám ezt is.

## 4. Fejezet:

A 4. fejezet a függvényekkel kapcsolatos alapfogalmakkal kezd, ugyanis a fejezet a függvények és programok szerekezetét taglalja. Szintén precíz részletes leírást kapunk a dolgokról az eddig megszokott módon. Először az egyszerűbb függvényeket mutatja be majd a nem egéssel visszatérő függvényeket is. A külső változókról is szót ejt valamint a statikus változókat is kifejti a 4.6-ba, míg a 4.7-be a regiszter változókkal ismerkedhet meg az olvasó. Szintén ebben a fejezetben még bemutatásra kerül a tartományok érvényességi szabálya, a blokkstruktúrák és header állományok. Változók inicializálását részletesebben újra elismétli, a fejezet végén pedig a rekurziót mutatja be valamint a C előfeldolgozó rendszert, ahol az állományok beépítése a #define által való makróhelyettesítés a valamint a feltételes fordítás kap szerepet.

## 5. Fejezet:

Az ötödik fejezetben a főtéma a mutatók és a tömbök lesznek. Elsőnek is a mutatókat a címzésekkel mutatja be, majd később a mutatók és tömbök kapcsolatát is, változásoknál az emelhető ki hogy ábrákkal segíti a megértést, ami szerintem nagyon jó, ugyanis könnyebb úgy megérteni ha az ember nem csak olvasni tudja a kódcsipeteket és a magyarázó szöveget hanem kis ábrákat mellé nézve értelmezi a mutatókat például vagy a tömbök számozását 0-tól a memoriában foglalt területen. Ugyan ebben a fejezetben lesz még szó címaritmetikáról karaktermutatókról és függvényekről. Valamint a mutatóra mutató mutatókról is, ami számonra régebben bonyolultnak tűnt, ezt elolvasva hasonló helyzetben lévő embereknek sokat segíthet. Mutatótömbök és több dimenziós többök bemutatása inicializálása, értelmezése több alfejezet is taglalja ezeket. Majd a fejezet végén előkerülnek a parancssor-argumentumok és a bonyolultabb deklarációk is. Ez a fejezet már nem annyira az alapokat részletezi hanem azokra már építve kissé de még mindig új "alap" fogalmakat bevezetve és elmagyarázva mutatja azt be.

## 6. Fejezet:

A 6. fejezet a struktúrák köré épül. Mélyebben belemegyünk ebbe a témaiba, és részletesen, ábrákkal is segítve, a struktúrák működését veszi át, valamint az azokra épülő dolgokat mint például a struktúratömbököt, a struktúra és függvények viszonyát, valamint a struktúra és mutatók együttesét, önhivatkozó struktúrákat. Ez a fejezet már egy haladóbb szint szerintem, itt már számomra is sok új dolog volt és ha csak a mellékelt kódokat nézzük is, azok is már bonyolultabbak mint az előző fejezetben mellékelt kódok, viszont nem kell megijedni ugyanis a szokásos módon ezt is jól elmagyarázza a könyv és meg lehet érteni elolvasás után.

## 7. Fejezet:

A hetedik fejezet főként az adat ki- és bevitelről szól. Annak részletesebb elismétléséről. Megjelenek a függvények részletes leírása, olyanoké mint például a printf vagy scanf, valamint a fejezetben a karaktervizsgáló és -átalakító függvényekről is kapunk egy kis táblázatot. A hibakezelésről is szó esik az stderr és exit függvények mellett. Matematikai függvények és egyéb függvényekről is szó esik még, hogy milyen könyvtárba tartoznak, azaz mit kell includeolnunk hogy elérjük őket, mit tudnak kezdeni a bemenetként ak-pott dolgokkal stb. Valamint a legutolsó alfejezetben a 7.8.7-ben a véletlenszerű számgenerálást ismerjük meg.

## 8. Fejezet:

A 8. fejezetben a UNIX rendszer adatbevitellel és kivitellet ismerkedünk meg. Különböző függvényeket mutat be, számomra ez a fejezet már nem volt annyira érdekes, de azért érdemes ezt is végig olvasni sose tudni mikor vesszük hasznát, de véleményem szerint ez már tényleg inkább csak plusz infók hogy a tudásunkat bővítsük.

## Összesítés:

Összeségében szerintem ez egy nagyon hasznos és precízen összeállított könyv, nagyon sokat tud segíteni a kezdő C programozók számára. Ezért én mindenkihez ajánlanám. Hátrányként esetleg azt tudnám megemlíteni hogy itt-ott vannak hibák elírások, de ez nagyon pici hátrány a sok-sok előny mellett.

A könyvnek van függeléke is mely még több hasznos és fontos dolgot vesz számon. Nem tudtom hogy az olvasónapló vonatkozik rá, de szerintem mindenkihez érdemes beleolvasni. Mivel még több hasznos információval rendelkezik. Egy szó mint száz érdemes olvasgatni mind a fő fejezeteket mind a függelékeket is, nagyon hasznos, szemléletes a nyelv bemutatása benne, valamint figyelmet fentartó módon mutatja be a dolgokat nem pedig untat minket.

## 10.3. Programozás

### [BMECPP]

Ezt az olvasónaplót a kijelölt oldalszámok elolvasása után írom meg.

#### (17.-59. o.):

Ezek az oldalok az osztályokat taglalják valamint az objektumokat. Nagyon részletesen, példákkal, kódokkal dusítva, személy szerint nekem ez a rész sokat segítet, ugyanis rengeteg kérdőjel volt bennem mind az osztályokkal mind az objektumorientáltsággal kapcsolatban, ezek az oldalak pedig választ tudnak adni. Tipikusan olyan rész ahol még tapasztaltként is fellapozhatjuk ha valami nem tiszta.

#### (187.-197. o.):

Ez a rész a kivételkezelésről szól. A könyv ezen az oldalalakon keresztül végig azt taglalja, elégé részletesen valamint forráskódokkal és feladatokkal tűzdeli meg az információ csokrokát, ezáltal az olvasó számára még jobban értelmezhetőek a dolgok. Try-catch blokkokat mutatja be azoknak egymásba ágyazásáról is szó esik valamint az elkapott kivétel újradobásáról is. Hasznos rész, nagyon részletes és szemléletes módon mutatja be a kivételkezelést c++-ban.

### **190. példa:**

A példában láthatóvá válik az hogy mi történik meg ha a felhasználó bemenetként 0-át ad meg vagy ha éppen ellenkezőleg bemenetnek nem nullát ad meg. Mivel a kódunk a beérkező szám reciprokát számolja ki és írja ki, ezért ugye 0 nem lehet mivel nem értelmezünk olyan törtet aminek 0 a nevezője. Ennek segítségével mutatja be hogy lehet használni a try-catchet, ugyanis ha 0-át adunk bemenetnek akkor dobni fog egy hibát amit elkapunk és kiírjuk hogy a bemenetnek nem nullának kell lennie.

### **197. példa::**

A példában azt láthatjuk, hogy az f2 kivételt dob, így f2-ben a definált i lokális változó felszabadul, míg az f1-ben lefoglalt Fifo fifo objektum felszabadul, mert meghívódik a destruktora. Majd lefut a main függvényben lévő catch blokk. Ki emeli továbbá a könyv is azt a fontos tényt hogy ugyan a hiba dobás és elkapás között kód fut le, de semmiképpen ne dobjunk új hibát míg az elsőt el nem kaptuk, mert annak kezelése nem lesz lehetséges.

### **(211.o) 58-as fólia kapcsán:**

Üzenet kezeléssel foglalkozik, valamint az előző try-catch téma épül, egy újabb jó példa eme téma fel-dolgozására. Valamint részletesen taglalja a megoldást, ezzel megértetve az olvasóval a try-catch lényegi működését.

### **Összesítés:**

Összesítésben egy nagyon jó és hasznos könyv, szemléletesen és részletesen mutatja be a C++ tulajdonságait és használatát. Ajánlott belőle szemezgetni, tanulgatni. Hasznos, lényegretörő és szemléletes. Az alapktól kezdődően egy magasabb szintre fel tudunk fejlődni csak ezen könyv mellett is C++ programozási nyelvben.

## **III. rész**

# **Második felvonás**

**Bátf41 Haxor Stream**

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

# 11. fejezet

## Helló, Berners-Lee!

### 11.1. Összehasonlító jellegű olvasónapló

**C++:** Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven Java: Nyékyné Dr. Gáspár Judit et al. Java 2 útikalauz programozóknak 5.0 I-II. o Ebből a két könyvből pár oldalas esszé jellegű kidolgozást kérek, Java és C++ összehasonlítás mentén, pl. kb.: kifejezés fogalom ua., Javában minden objektum referencia, minden dinamikus a kötés, minden függvény virtuális, klónozás stb.

#### Összehasonlító jellegű olvasónapló:

Mindkét könyv (én esetben 3 példány) nagyon részletesen és szemléletesen mutatja be a C++ valamint a Java programozási nyelvet. Bár itt ott már-már elavultnak mondható egy két rész, ugyanis folyamatosan változik ez a szakág, de ennek ellenére egy nagyon jó alapot tud adni a nylevekből, amire az ember már könnyedén építhet saját maga. Nem kell tehát meglepődni ha valamilyen oknál fogva a könyvben egy két kódcsípet már nem fordul le vagy éppen hibát lök vissza, én magam is jártam így olvasás/próbálgatás közben.

A C++ egy általános célú programozási nyelv, amely lehetővé teszi az objektumorientált és a generikus programozást is, de alacsonyabb szintű nyelvi konstrukciókat is támogat. Míg a Java egy teljesen objektumorientált eszközökészletből építkező programozási nyelv.

A Java jelölésrendszerében sok minden átvett a C++-tól de sok figyelmet fordítottak mellette a megbízhatóságra, valamint a biztonságosságra, ezért a két nyelv között vannak kisebb-nagyobb eltérések, melyekre a következő sorokban/oldalakban jobban kitérnek. Ez az olvasónapló ugyanis a feladat leírása alapján is jól látszik, hogy a két nyelvnek az összehasonlításáról fog szólni. Főként különbségeik mentén, de nyilván hasonlóságaikról is szót ejtve.

A Java fordítóprogram egy bájtkódnak nevezett formátumra fordítja le a forráskódot amit majd a Java Virtuális Gép interperterként fog értelmezni, ennek meg van az előnye és a hátránya is, előnye például biztonsági szempontokból mutatkozik leginkább, hátrányaként pedig abszolút kiemelendő a sebesség. Míg a C++ natív kód fordul, ezáltal sokkal gyorsabb, játékoknál is fontos például a gyorsaság így ezért is inkább c++-ban írnak játékfejlesztésnél(pc-re). De ugyanakkor a Java kódok így "hordozhatóak" ugyanis a bájtkód osztályok átvihetőek platformspecifikus JVM-ekre, ellentétben a C++-al. A C++ kódok platformfüggőek még a Java programok platformfüggetlenek, a JVM-nek köszönhetően újrafordítás nélkül is futni fognak.

A Java forráskódokat olvasgatva nyilvánvalóvá válik hogy a szintaxisa a C illetve a C++ nyelvből fejlődött ki. Ugyanakkor néha alapvető pontokban is eltér tőlük. Például a metódusok visszatérési típusának

megadása a Javában kötelező minden metódus estetén, míg a C++-ban ugyanez nem mondható el, ugyanis van ott egy alapértelmezett "int" visszatérési típus, így nem kötelező minden esetben a típus megadása.

A Java 2 Utikalóz könyv első kötetében fel is hívja a figyelmet arra, hogy a "gyakorlottabb" C++ programozók számára az első meglepetés jelentheti a main metódus argumentuma, hiszen a C++-al ellentétben itt egy igazi szövegtömbön "(String[])"-en keresztül kerülnek átadásra a paraméterek. Jómagam is előbb a C++ nyelvvel ismerkedtem meg, így számomra is a Javával való első találkozásnál feltűnt ez. Valamint ami még egy sima "Hello, World!" megírása, fordítása, s futtatása során is megfigyelhetünk, az nem más mint a Java név és forrás közötti kapcsolata. Ugyanis C++-ban nincs szigorú kapcsolat az osztálynevek és a fájlnevek között, viszont ugyan ez nem mondható el a Javánál ahol például egy HelloVilág osztály forráskódjának a HelloVilág.java nevet kell viselnie, ahoz hogy sikeresen fordítani és futtatni tudjuk a kódunkat.

Továbbá szembetűnő az I/O statements használata közötti különbségek is, C++ ugye a

```
cin>>
```

és

```
cout<<
```

kifejezések és jelek használatával könnyedén bekérhetünk vagy épp kiolvashatunk a standard outputról/-ra. A Java esetében azért ez picit komplexebb, ugyanis a beolvasás biteonként történik

```
(System.in)
```

a kiíratás ugyanakkor nem okozz nehézségeket egy egyszerű

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

parancsal történik.

Kommentelés terén a két nyelv ugyanazokat a jeleket ugyanúgy használja, egy különbséggel hogy míg C++-ban nem lehet dokumentációs megjegyzéseket létrehozni, addig Javában használható az. A következő jelöléssel: `/** */`. Hasonló a több soros megjegyzéshez `(* *)` annyi különbséggel, hogy ugyan a fordítóprogram figyelmen kívül hagyja, viszont azok a program dokumentációjába belekerülnek.

További különbség például a memória kezelése a két programozási nyelvnek, ugyanis a javában automatikus takarítás van amit egy űgynévezett szemétygyűjtő eljárás végez. Ennek a működése teljes egészében a futtató rendszer feladata, tehát a felhasználónak/fejlesztőnek nem kell törődni a tárterület felszabadításával. A rendszer felismeri ha már kezd a memória betelni és akkor elindítja ezt az eljárást, vagy éppen akkor ha semmi se fut. Ezzel ellentétben a C++-ban nincs ilyen, ott a memória felszabadítását a programozó végzi, destrukturákkal kell dolgozni, és "manuálisan" törölni a nem használt hivatkozásokat. Ez minden oldalról lehet előny és hátrány is, például a c++ féle módszer előnyösebb lehet az esetben ha kisebb limitált memoriával rendelkező eszközökre akarunk valami nagyobb teljesítményű programot, akár egy játékokat írni, a java féle módszer nagyon fogyasztja a memóriát ami nem kifizetődő, de alap esetben nyilván kényelmesebb és jobb egy automatikus takarítás.

Továbbá a C++ lehetőséget nyújt a többszöri öröklődésre ellentétben a Javával ahol ez nem megtalálható. Valamint a kivételkezelésük is különbözik, míg a c++ a try/catch funkciókkal dobja és kapja el a kivételeket. Addig a Javának különböző kivételkezelése van ugyanis nincs destructor se a nyelvben, valamint a Javában meg kell határozni a try/catch-et ha a funkció deklarálja hogy esély van arra hogy kivételt dob.

A Javában grafikai elemeket létrehozni és használni az AWT(Abstract Window Toolkit) segítségével tudunk. Vagy léztezik egy kicsit fejlettebb, látványosabb elemeket alkotó Swing nevezetű könyvtár. C++-ban

pedig a QT tud segítségünkre válni ha grafikai megjelenítésről van szó. Bár nekem a grafikai része a programozásnak nem túl gyakran használt, de azért mindenképp említésre méltó rész ez is.

Javában minden objektum referencia kivétel nélkül. Nyilvánvalóan ez C++-ra egyáltalán nem igaz.

Operátor túlterhelés is különbség többek közt, ugyanis a Javában ilyenre nincs lehetőség, míg a C++ rendelkezik operátor túlterheléssel. Az operátorok jelentését is megtudjuk változtatni a saját típusainkra. Persze C++-ba se túlterhelhető minden operátor, de nagyon sok igen (a c++ operátorok wikipédia oldalán találhatunk egy táblázatot, ahol láthatjuk hogy az adott operátor túlterhelhető-e vagy sem), ami sokszor jól jöhet a feladatunk elvégzésére. Többek közt ebben a könyvben ahol ez az olvasónapló olvasható, találhatunk olyan feladatot ahol az operátor túlterhelés nagy szerepet kapott. Haználható például két string + jelel való összeadására is, ami Javában alapból működik.

Összefoglaláskép annyi, hogy ugyan van pár lényegi különbség a két nyelv között, mégis sok azonosság fellehető, így az egyik nyelv tudatában már a másik nyelv kevesebb erő feszítéssel is megtanulható. Mindkét nyelv fontos napjainkban és megvan a maga előnye/hátránya, ezeket ismerve kihozhatjuk a maximumot belőlük. A könyvek nagyon részletesek, csak ajánlani tudom őket, nem is csak elolvasásra és abból tanulásra, de például ha "karierünk" során valami olyanba botlunk amit nem tudunk megoldani, nyugodtan fellapozhatjuk újra a megfelelő oldalakat, nagy esélyt látva arra hogy a problémánkat megoldja. Ha bár a könyv maga tartalmaz olyan kódcsípeteket vagy szituációkat ami már napjainkban nem validak, még is elviekin helyes, és nagy segítség a nyelvek elsajátításához felsőbb fokokon is, de alap szinten is megtanulhatunk dolgot persze.

## 11.2. Élmény-olvasónapló

**Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)** Itt a kijelölt oldalakból egy 1 oldalas élmény-olvasónaplóra gondoltam.

### Élmény-olvasónapló:

Számomra a Python egy kissé ismeretlen programozási nyelv, ugyanis még nem sokat használtam, mivel az egyetemi éveim során kezdtem bele a programozás tanulásába, és az ott használt nyelveket (C++, Java) részesítettem előnybe főként. Tehát számomra ez a pár oldal nem csak hasznos információk szerzésére volt alkalmas, de motiválóan is hatott, kedvet teremtett a Python nyelv megismerésére/elsajátítására.

A Python egy általános célú programozási nyelv, melyet 1990-ben Guido van Rossum alkotott meg. Ez a magas szintű programozási nyelv, rengeteg pozitívummal rendelkezik, ugyanis objektumorientált, platformfüggetlen, egyszerű a használata, nincs szükség fordításra és linkelésre, az interpreter interaktívan is használható és még sok más jó tulajdonsággal rendelkezik. Nem véltelen tehát az, hogy 2019-ben a legjobb programozási nyelvek ranglistán az első helyet tudhatja magáénak a Python, ugyanis minden területen jól használható, és a fejlesztők örömmel használják.

A 3.1 ás a 3.2 a fent említett hasznos és fontos tulajdonságai részletezi, és ad alap információt a Python nyelvről. A 3.3 az első "érdekesebb" bekezdés, ugyanis itt egyből kiemeli az egyik hasznos jellemzőjét a Pythonnak, azaz a behúzásalapú szintaxisát, amely eltér a megszokott C++/Java szintaxistól. Ugyanis itt nincs szükség zárójelzésre vagy explicit kulcsszavakra (pl. begin, end), hanem a tab és szóköz kezeli az elválasztásokat. Az utasítások pedig a sor végéig tartanak alap esetben tehát a ";" se kell, persze a megfelelő "\" jelzéssel írhatunk egy utasítást több soron keresztül, ha egyben nem férne el az. Ezek után megismerhetjük a lefgolalt kulcsszavakat, majd a típusokat taglalja a könyv, ahol újabb pozitív jellemzőre

hívja fel az olvasó figyelmét, mi szerint nem kell minden változő típusát explicit megadni, ez szintén a kód tömörsegét segíti elő, tehát egy bonyolultabb kódot is könnyen olvashatunk úgy hogy lényegesen kevesebb sorban van megírva mint egy C++-os vagy Java-s társa.

Majd szó eset a különféle függvényekről, ciklusokról, metódusokról, funkciókról, azoknak a megfelelő használatáról, nyilván eltérés itt is akadt bőven egy C++-hoz képest például, de szerintem egy C++/Java ismeretében és a Python szintaxis tudtában ezek könnyen elsajátíthatóak, és nem okoznak nagy meglepetéseket az olvasó számára. Később pedig az osztályokról és objektumról írt a könyv, de kiemleték azt is hogy "...e könyvnek azonban nem célja a nyelvről teljes referenciát biztosítani, így az érdeklődő Olvasót további kézikönyvek áttanulmányozására biztatjuk.", de az alap osztályok létrehozását használatát, konstruktorkiemelését nagyszerűen leírta, minden részre a mélyebb részekbe nem ment bele.

Modulokról eset szó a 3.3.6-ban, ami főként a mobilkészülékekkel való fejlesztés megkönnyítéséért felelősek, rengeteg hasznos modult tartalmaz melyek közül van olyan például amely a készülékek kamerájával kapcsolatos műveleteket tudja elkészíteni/lekérdezni, vagy például la hangfelvételek készítéséért vagy lejátszásáért felelősek. A 3.3.7 a kivételkezelést szemlélteti röviden, a try-kulcsszóval.

A 3.4 pedig már példákon keresztül taglalja a különböző témákat. Ami szerintem nagyon jó ötlet, ugyanis én önamgamon is észrevettem már hogy sokat segít ha láthatom is az "elmélet" mögötti gyakorlati példákat. A könyv már a különböző témák bemutatásakor is használt kódcsípeketet a szemléltetésre, de ezután még külön gyakorlati példákon keresztül is szemléltette a "nehezebbnek vélt" témaöröket. Persze programozni amúgy se könyvből lehet megtanulni, ergo az olvasottakat gyakorlatban is érdemes kipróbálni saját kézzel írt kódokkal ha lehet. Ugyanis így könnyebben megjegyezzük és megtanuljuk a különböző nyelveket, vagy akár azokon belül a metódusokat, függvényeket, ciklusokat.

Összeségében, szerintem egy nagyon jó alapot nyújtó pár oldal volt ez a könyvből, szemléletes leírás menettel és példákkal. Én személy szerint ajánlanám mindenkinak aki Python tanulásba bele akar kezdeni, és még azoknak is talán akik nem, ugyanis elképzelhető hogy ez a pár oldal meghozza hozzá a kedvet.

## 12. fejezet

# Helló, Arroway!

### 12.1. OO szemlélet

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

Megoldás forrásai:

[PolarGen.java](#)

[PolarGen.cpp](#)

A feladatot én egy boolean típusú változó segítségével oldottam meg, ami tárolta számomra az az információt hogy van-e már kisszámolt véletlenszerű szám tárolva, tehát azzal kezdtem az egészet, hogy létre hoztam egy nincsTarolt változót melynek true-t adtam értékül. Valamint egy double változót is létre hoztam amely a legenerált számot fogja értékül kapni majd minden.

```
boolean nincsTarolt = true;
double tarolt;
```

A számok generálását pedig a példányon a "kovetkezo" metódus meghívásával érhetjük majd el, ami ha van már tárolt érték akkor azt visszaadja valamint a boolean változónkat true-ra állítja.

```
nincsTarolt = !nincsTarolt;
return tarolt;
```

Ha pedig nincs még tárolt értékünk akkor a legenerált két értékből az egyik a "tarolt" változóba landol, ezáltal a "nincsTarolt" változó értékét falsera állítja, míg a másikat visszaadja (kiíratja). Maga a kiszámítási rész is itt található, tehát az a rész amin "spórolunk" ha csak a tárolt szám visszaadása elégges.

```
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;
```

Tehát összegezve egy lefutása alatt két random számot generál, amiből az egyiket eltárolja, így a következő meghívásnál "kíméljük a processzorunkat" ugyanis elég az előző eltárolt számot visszaadnia nem kell újat generálnia. A mi esetünkben így a 20 legenerált szám létrehozásához csak 10-szer kell lefutnia ténylegesen a számítási résznek.

Így néz ki tehát futtatásnál:

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ javac PolarGen.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ java PolarGen
0.8952941717989455
-0.09506103815931427
0.1954374417102932
0.46436713375466543
-0.26835632494969996
-1.0221786656387872
0.09907519786445387
1.9268426898370439
1.0120174549348264
-0.18313718613548552
-1.732016936749558
-0.2439308634259936
0.20934360496856275
1.570624464523647
-0.37590583991230586
-0.2235996691244321
-0.0973826010084923
0.794645487355327
0.6959157234414205
-0.27746273177613423
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ █
```

12.1. ábra. PolarGen.java futása

A feladat kéri ugyan ezt C++-ban is! A Java kód tulajdonában, talán már nem is lesz olyan nehéz a C++-os kód előállítása se, mindössze néhány dolgon kell módosítanunk.

Nyilvánvalóan a legszembe tűnőbb dolgok azok az includeok/importok és ehhez hasonlók, de valójában nem sok a tényleges különbség a C++ és a Javás verzió között. Talán kiemelendő hogy a C++-nál létre kell hoznunk egy destruktort mert itt nem található meg az automatikus takarítás mint a javánál. De maga a felépítés és a számítások hasonlóak, törekedtem az elnevezésekkel és a tagolással is úgy játszani hogy lássuk azt hogy szinte ugyan az a kód. A kódokat feljebb a "Megoldás forrásai" résznél linkeltem!

Így néz ki hát a C++-os PolarGen fordítása és futtatása:

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ g++ -o PolarGen PolarGen.cpp
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$./PolarGen
-0.748237
-0.757155
-0.653762
1.56476
-0.329681
-0.31089
-0.325787
0.171602
2.31587
-0.610129
-0.896631
0.886729
-1.41528
-0.0990208
-0.665003
-2.17302
-0.896296
0.937737
0.931381
0.223461
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ □
```

12.2. ábra. PolarGen.cpp futása

És nézzük meg a JDK forrásban hogy található meg ez:

```
synchronized public double nextGaussian() {
 //See Knuth, ACP, Section 3.4.1 Algorithm C.
 if(haveNextNextGaussian) {
 haveNextNextGaussian = false;
 return nextnextGaussian;
 } else {
 double v1, v2, s;
 do {
 v1 = 2 * nextDouble() - 1; //between -1 and 1
 v2 = 2 * nextDouble() - 1; //between -1 and 1
 s = v1 * v1 + v2 * v2;
 }while (s >= 1 || s == 0);
 double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
 nextNextGaussian = v2 * multiplier;
 haveNextNextGaussian = true;
 return v1 * multiplier;
 }
}
```

Mint láthatjuk hasonlóan gondolkodtak a JDK programozói is, egy dologban tért el, ez nem más mint a synchronized public double használata. Ami azt teszi, hogy a program futását egy szálra korlátozza így ha egyszerre több hívna meg a többi szálnak várakoznia kellene, míg az első szál elvégzi a dolgot.

## 12.2. "Gagyi"

Az ismert formális

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

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

Megoldás forrásai:

[Gagyi.java](#)

[Gagyi2.java](#)

Én ezt a feladatot az Integer osztályra építettem, annak segítségével szemléltetem a feladat megoldását. Tehát importálnom kellet a következőképpen:

```
import java.lang.Number;
```

De nézzük a fentebb belinkelt Gagyi.java nevű kódunkat futás közbe!

### 12.3. ábra. Gagyi.java futása

A kód nem meglepő módon végtelen ciklust hozott létre, ugyanis a while minden egyes eleme igazzal tér vissza, tehát:

```
x<=t == true
x>=t == true
mivel 321 == 321
```

Viszont az is igaz lesz hogy  $t \neq x$ , értékük ugyanis megegyezik, viszont az osztályból két példányt példányosítottunk, tehát nem lehetnek egyenlők, hisz azonos időben a memória különböző területén helyezkednek el nem pedig ugyan ott.

De képesek vagyunk a végtelen ciklus elkerülésére, minden össze az integerek értékének megváltoztatásával. Erre példát láthatunk a Gagyi2.java fájlban. Ami a következő képpen fest:

```
import java.lang.Number;

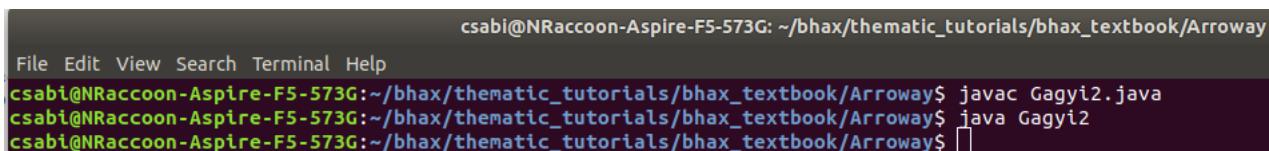
class Gagy12{

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

```
Integer t = 123;
Integer x = 123;

while(x<=t && x>=t && t!=x) {
 System.out.println("Gagyi");
}
}
```

Futás közben pedig nem annyira látványos, de így néz ki:



```
csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic_tutorials/bhax_textbook/Arroway
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ javac Gagyi2.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ java Gagyi2
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$
```

12.4. ábra. Gagyi2.java futása

De miért van ez?

A válasz egyszerű, a fent említett osztály gyorsítótárazási tulajdonságát kihasználva van lehetőségünk erre, ugyanis az integer [-128, 127] intervallumban már egy meglévő objektumról kapunk egy referenciát, ez pont a gyorsítótárazás miatt van, valószínűleg úgy gondolták ebben a tartományban használjuk leginkább a számokat és egy fajta könnyítés képp van ez így. Viszont ez épp elég ahoz hogy a t != x feltétel hamis értékkel térjen vissza, ezáltal a while feltétele is hamisra módusoljon, mivel ha már az egyik nem igaz akkor az egész hamis lesz.

A feladat ugyanakkor kérte itt is hogy nézzük meg a JDK forrást. Ahol a következő látható:

```
public static Integer valueOf(int i) {
 if(i >= IntegerCache.low && i <= IntegerCache.high)
 return IntegerCache.cache[i + (-IntegerCache.low)];
 return new Integer(i);
}
```

Mint az fent említve volt, most láthatjuk kód formájában is, a -129 érték például már kivül fog esni az intervallumon így az már a "return new Integer(i)-vel jön létre nem pedig a poolból kapjuk mint a -128-ig. Ez pedig azt eredményezi hogy különböző memória címen fog létrejönni így a feltételünk teljesül.

## 12.3. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-leáll, ha nem követjük a Yoda conditions-t! Yoda-conditionról bővebben: [https://en.wikipedia.org/wiki/Yoda\\_conditions](https://en.wikipedia.org/wiki/Yoda_conditions)!

Megoldás forrásai:

[Yoda.java](#)

[Yoda2.java](#)

A feladat az volt hogy olyan programot írunk amely leáll, ha nem követjük a Yoda conditionst, azaz java.lang.NullPointerExceptionnel kilök. Én egy egyszerű, de talán szemléletes példát hoztam ennek bemutatására. Valamint a kódot "átírtam" a Yoda conditions követése szerint is, így láthatjuk mi a különbség.

```
public class Yoda{
 public static void main (String[]args) {
 String yodaString = null;
 if (yodaString.equals("Yoda")){
 System.out.println ("Hurrá, egyenlő");
 }
 }
}
```

A fent megjelölt megoldás forrás első forrása (Yoda.java), az a feladat által igényelt fájl, ennek futása NullPointerExceptionnel lefog állni, ugyanis a string amihez hasonlítanánk a másik stringünket egy null pointer valójában, így maga az összehasonlítás nem lesz lehetséges, ugyanis null objektum metódusát akarnánk meghívni.

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ javac Yoda.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ java Yoda
Exception in thread "main" java.lang.NullPointerException
 at Yoda.main(Yoda.java:4)
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$
```

12.5. ábra. Yoda.java futása

```
public class Yoda2{
 public static void main (String[]args) {
 String yodaString = null;
 if ("Yoda".equals(yodaString)){
 System.out.println ("Hurrá, egyenlő");
 }
 }
}
```

A fent belinkelt 2. forrás pedig ugyan ez, csak annyi különbséggel, hogy ez esetben követtem a Yoda conditionst, így lefut gond nélkül a program és egyszerűen a várt "false" értéket fogja vissza adni. Ez azért lehetséges mert ez esetben a meglévő "Yoda" stringhez akarjuk hasonlítani a "null pointer" stringünket, ami egyértelműen nem egyenlő ugyan de az összehasonlítás önmaga lehetséges ellentétbe a másik kóddal.

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ javac Yoda2.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$ java Yoda2
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Arroway$
```

12.6. ábra. Yoda2.java futása

## 12.4. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását!

Megoldás forrása:

[PiBBP.java](#)

A BBP (Bailey-Borwein-Plouf) algoritmus a BBP formulát használja alapul amit 1995-ben talált fel Simon Plouffe. Az algoritmus alkalmas a Pi hexa számjegyeinek meghatározására.

Erről lenne szó:

The formula is derived from the Bailey-Borwein-Plouffe formula for pi, which was published by Simon Plouffe in 1995.

$$\pi = \sum_{k=0}^{\infty} \left[ \frac{1}{16^k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right].$$

The BBP formula gives rise to a **spigot algorithm** for computing the  $n$ th

12.7. ábra. BBP formula

A formulát viszont nem ember találta meg, hiába Simon nevéhez fűződik, egy számítógépes program által lett megtalálva. Persze most már létezik több változata a formulának, amik talán korszerűbbek, jobbak, gyorsabbak.

Nézzük meg a kódot:

```
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);
```

A kód elején létre kell hoznunk váltózokat amik majd a tört elemekkel való számolást fogják nekünk intézni. A d16Sj() metódus segítségével adjuk meg nekik az értéküket.

```
d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;
d16Pi = d16Pi - StrictMath.floor(d16Pi);
```

Ezután kiszámoljuk a megadott Pi értéket, és ezt kerekítjük a StrictMath.floor() metódussal.

```
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();
}
```

A fenti kódcsípetben pedig azt láthatjuk, hogy készítünk egy buffert és egy tömböt, ami majd a hexajegyeket fogja tartalmazni. Az append() metódust is alkalmazzuk, ami azért felelős hogy az új jegyeket hozzáfűzzük a már meglévő Pi jegyekhez. Legvégül pedig az látható hogy az elkészített bufferünket stringé alakítjuk át a toString() metódussal.

```
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;
}

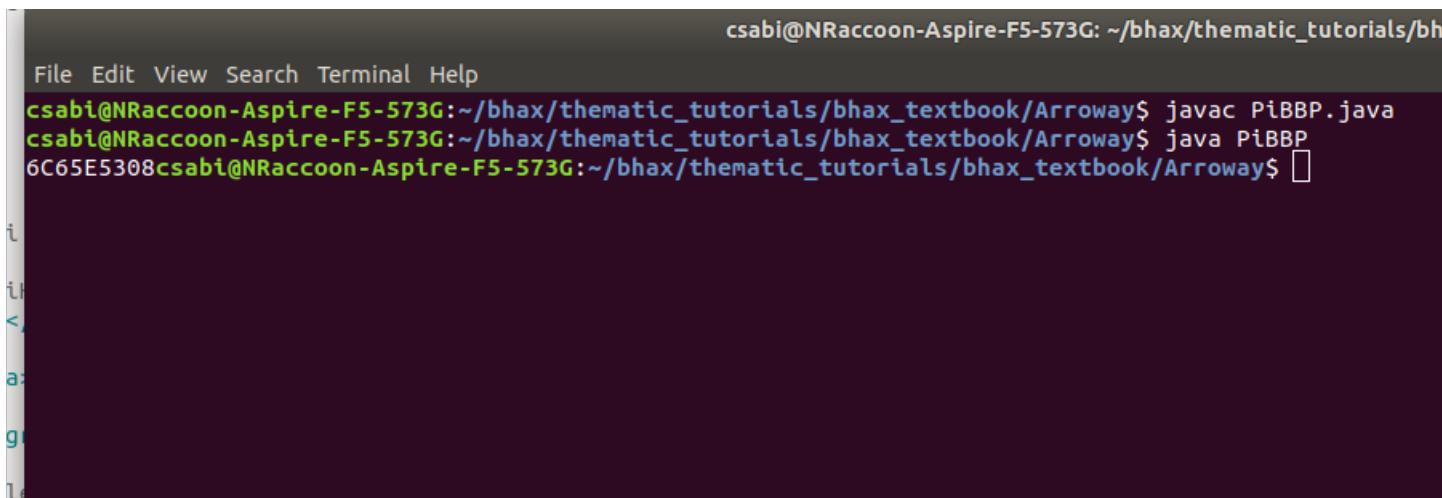
public String toString(){
return d16PiHexaJegyek;
}

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

```
}
```

Majd a metódusokat elkészítjük a tört számok kiszámítására, majd ugye visszaadjuk a kiszámolt hexajegyeket a megadott számjegytől számítva, ez lesz az ami a terminálon látható lesz a "felhasználó" számára.

Tehát így néz ki futtatásnál:



The screenshot shows a terminal window with a dark background and light-colored text. At the top, it displays the command line: csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic\_tutorials/bhax\_textbook/Arroway\$. Below this, the terminal menu bar shows 'File Edit View Search Terminal Help'. The main area of the terminal contains the following text:  
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic\_tutorials/bhax\_textbook/Arroway\$ javac PiBBP.java  
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic\_tutorials/bhax\_textbook/Arroway\$ java PiBBP  
6C65E5308csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic\_tutorials/bhax\_textbook/Arroway\$ █  
  
The text '6C65E5308' is highlighted in blue, indicating it is a copyable selection. The terminal window has a standard window frame with title bars and scroll bars.

12.8. ábra. PiBBP.java futása

Az eredmény a várt lett, visszakapjuk a hexa jegyeket gond nélkül. Még pedig a Pi hexadecimális kifejtését a 1000001. jegytől, ami a "6C65E5308".

# 13. fejezet

## Helló, Liskov!

### 13.1. Liskov helyettesítés sértése

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

#### A feladat megoldásában tutoráltam Schachinger Zsoltot!

A Liskov elv, a S.O.L.I.D. elvek közül az L nyelv jelentése, hogy ha egy programban T altípusa S akkor ahol használható T, lecserélhető T-nek S altípusával anélkül, hogy az hatással lenne a program tulajdonságára. A feladat eme elv megsértése.

A feladatot Bátfai Tanárúr ötlete alapján a madarak repülése/nem repülése által mutatnám be. Mit is értek ez alatt? Nézzük meg a kódot, először Javában:

[Liskov.java](#)

A kód magyarázata:

```
class Madar{
public
 void repul(){
 System.out.println("repul");
 }
}
```

Létrehozunk egy "Madar" osztályt amiben van egy public metódus, "tulajdonság", még pedig a repülés.

```
class Sas extends Madar{}

class Pingvin extends Madar{}
```

A "Madar"-ak osztályának képzünk két gyermek osztályt a "Sas" és a "Pingvin" osztályokat. Amik öröklik a "Madar"-ak osztály "tulajdonságait".

```
class Program{
public
```

```
void reptet(Madar madar) {
 madar.repul();
}
```

Valamint létrehozunk egy "Program" osztályt, mely képes lesz "reptet"-ni madarainkat.

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

 Program program = new Program();
 Madar madar = new Madar();
 program.reptet(madar);

 Sas sas = new Sas();
 program.reptet(sas);

 Pingvin pingvin = new Pingvin();
 program.reptet(pingvin);
}
```

A main-be pedig ki is próbáljuk hogy valóban örökölték-e a repülést, és a "program" valóban megtudja-e őket reptetni.

Ha az egészet fordítjuk és futtatjuk a következő fogad minket:

```
csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic_tutorials/bhax_textbook/Liskov
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ javac Liskov.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ java Liskov
repul
repul
repul
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ █
```

13.1. ábra. Liskov.java futatása

Mi a hiba?

A gond az hogy a pingvin is repül a kód alapján, míg köztudott hogy a pingvinek nem repülnek. De miért van ez?

Azért van, mert a "Madar"-ak osztályból lett leszármaztatva, és az osztályon belül deklarálva van a repülés, így azt is elörökölte mind a sas, mind a pingvin.

Ugyan ez a kód C++-ban (nem taglalnám külön részekre hisz teljesen ugyan az a felépítése):

```
madar.cpp
```

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

```
class Madar{
public:
 void repul() {
 cout << "Repul" << endl;
 }
};

class Program{
public:
 void reptet(Madar &madar) {
 madar.repul();
 }
};

class Sas : public Madar{};

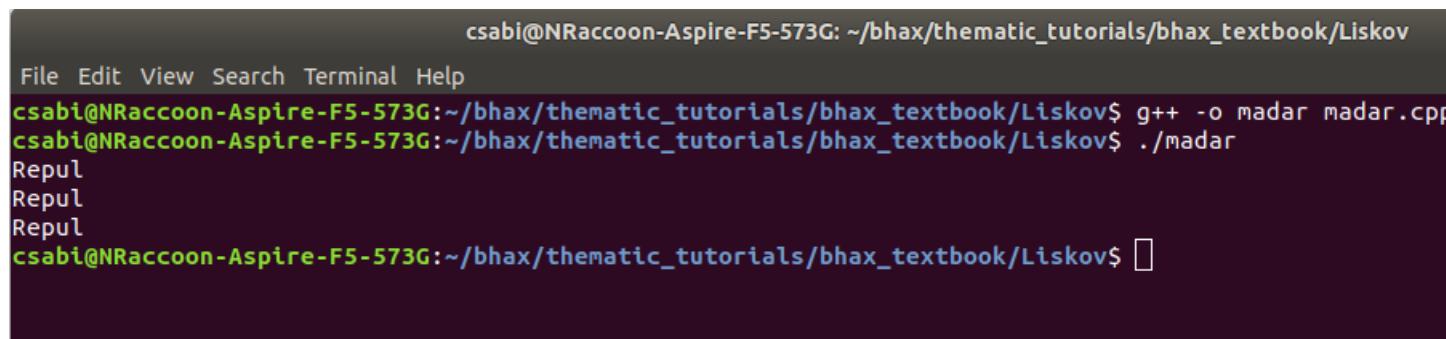
class Pingvin : public Madar{};

int main(){
 Program program;
 Madar madar;
 program.reptet(madar);

 Sas sas;
 program.reptet(sas);

 Pingvin pingvin;
 program.reptet(pingvin);
}
```

C++ verzió futás közben:



```
csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic_tutorials/bhax_textbook/Liskov
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ g++ -o madar madar.cpp
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$./madar
Repul
Repul
Repul
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$
```

13.2. ábra. madar.cpp futatása

C++-nál is ugyan az a helyzet. De van rá megoldás mindkét nyelvben, mivel nagy a hasonlóság én csak C++-ban mutatnám be, hogy is tudjuk ezt a problémát hamar orvosolni.

A logikailag (is) helyes megoldás:

`megoldasmadar.cpp`

```
class Madar{};
```

Létrehozzuk a "Madar" osztályt, ez esetben viszont nem kapja meg a repülés metódusát (a kódban semmit, de például beírható lenne hogy két lábuk van stb...).

```
class RepuloMadar : public Madar{
public:
 virtual void repul() {
 cout << "Repul" << endl;
 }
};
```

Létrehozzunk egy "RepuloMadar"-ak osztályt a "Madar" osztály utódjaként, mely megkapja az előző osztálytól "elvett" repülési metódust.

```
class Program{
public:
 void reptet(RepuloMadar& rmadar) {
 rmadar.repul();
 }
};
```

Ugyanúgy megmarad a "Program" osztályunk, csak a repülőmadarakat lesz képes reptetni most már.

```
class Sas : public RepuloMadar{};

class Pingvin : public Madar{};
```

A "Sas" és a "Pingvin" osztályokat ugyanúgy származtatjuk, annyi különbséggel hogy a "Pingvin" a sima "Madar" osztálytól még a "Sas" a "RepuloMadar"-ak osztálytól fog öröklődni.

```
int main(){
 Program program;
 Sas sas;
 program.reptet(sas);

 Pingvin pingvin;
 program.reptet(pingvin); //enélkül a sor nélkül lefordul és fut a program
}
```

Main-be pedig kerül megint a "tesztelés", megpróbáljuk reptetni a madarakat.

A "helyes" verzió futás közben:

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov
megoldasmadar.cpp: In function 'int main()':
megoldasmadar.cpp:32:24: error: no matching function for call to 'Program::reptet(Pingvin&)'
 program.reptet(pingvin);
 ^
megoldasmadar.cpp:15:7: note: candidate: void Program::reptet(RepuloMadar&)
 void reptet(RepuloMadar& rmadar){
 ^~~~~~
megoldasmadar.cpp:15:7: note: no known conversion for argument 1 from 'Pingvin' to 'RepuloMadar&'
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$
```

13.3. ábra. megoldasmadar.cpp futatása

Azt láthatjuk hogy hibát dob a pingvineknél, hisz azok nem képesek repülni, nem férnek hozzá a repülőmadarak tulajdonságaihoz.

## 13.2. Szülő-gyerek

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

Ebben a feladatban azt szemléltetjük hogy egy szülő referencián keresztül, amely a gyerek objektumára hivatkozik, egyszerűen nem tudjuk meghívni a gyermek metódusát, amit Ő maga nem definiált.

Alap esetben ugye az ősosztály rendelkezik tulajdonságokkal és metódusokkal amit a gyerek örökölni tud tőle, és ezen felül a gyermek osztály rendelkezhet olyan tulajdonságokkal és metódusokkal amivel a szülő osztály nem rendelkezik.

Nézzük meg a feladatot elsőnek Javában:

```
public class Osztaly{

 public static void main(String[] args) {
 Szulo sz = new Gyerek();
 sz.kiir1();
 sz.kiir2();
 }
}

class Szulo{
 public void kiir1(){
 System.out.println("Szulo");
 }
}

class Gyerek extends Szulo{
```

```
public void kiir2(){
 System.out.println("Gyerek");
}
```

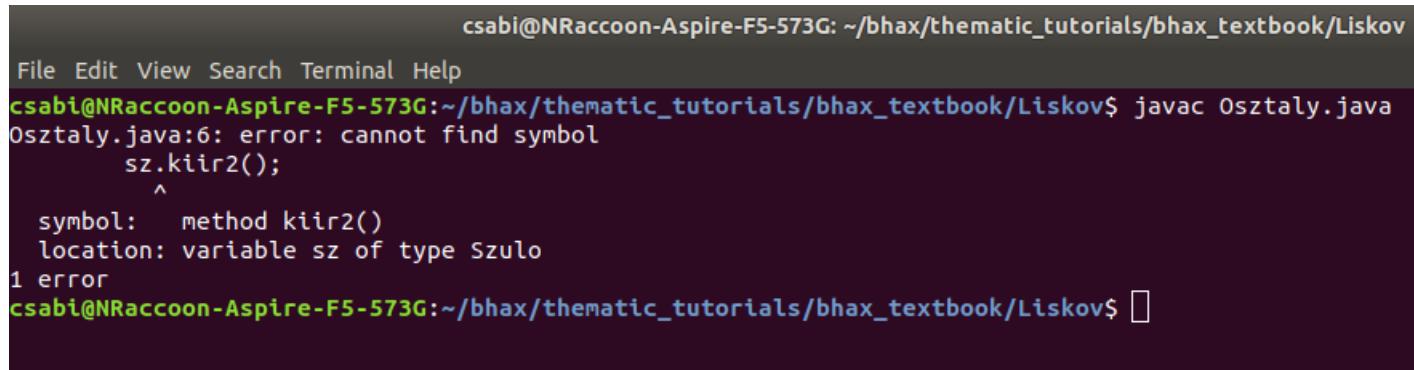
Én ezzel a kóddal szemléltetném, persze máshogyan is megírhattam volna, de szerintem ez szemléletesen mutatja be.

Mit látunk?

Van egy szülő osztályunk amely rendelkezik egy "kiir1" metódussal, valamint egy abból származtatott gyerek osztályunk ami szintén rendelkezik egy metódussal, amivel a szülő nem ez pedig a "kiir2".

```
Szulo sz = new Gyerek();
sz.kiir1();
sz.kiir2();
```

A mainben pedig példányosítunk a fent látható módon, ezután pedig meghívjuk mindkét függvényt rá. Fordításnál a következőket fogjuk látni:



The screenshot shows a terminal window with the following text:  
File Edit View Search Terminal Help  
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic\_tutorials/bhax\_textbook/Liskov\$ javac Osztaly.java  
Osztaly.java:6: error: cannot find symbol  
 sz.kiir2();  
 ^  
 symbol: method kiir2()  
 location: variable sz of type Szulo  
1 error  
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic\_tutorials/bhax\_textbook/Liskov\$

13.4. ábra. Osztaly.java fordítása

Mint azt látjuk hibát dobott, ugyanis erre a példányra a "kiir2"-öt nem tudjuk meghívni mivel az szülő típusú. Ellentétben a "kiir1"-el, azt gond nélkül megtudnánk hívni, és lefordul valamint lefut a program gond nélkül. Javában minden objektum referencia, és a kötés dinamikus, de ezzel nem küldhetjük a gyerek által hozott új üzeneteket. De megnézzük ugyan ezt a példát C++-ban is, hátha van változás.

Tehát itt is volna ugyan ez C++-ban:

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

class Szulo{
public:
 void kiir1(){
 cout << "Szulo" << endl;
 }
};
```

```
class Gyerek : public Szulo{
public:
 void kiir2(){
 cout << "Gyerek" << endl;
 }
};

int main(){

Szulo* sz = new Gyerek();

sz->kiir1();
sz->kiir2();

}
```

A felállás változatlan, ugyanúgy rendelkezünk egy szülő és egy gyerek osztállyal, minden kettőnek ugyanúgy megvan a maga metódusa, és a mainbe ugyanúgy példányosítás után meghívjuk őket rá.

Nézzük meg tehát ennek a fordítását:

```
csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic_tutorials/bhax_textbook/Liskov
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ g++ -o szulo-gyerek szulo-gyerek.cpp
szulo-gyerek.cpp: In function 'int main()':
szulo-gyerek.cpp:23:6: error: 'class Szulo' has no member named 'kiir2'; did you mean 'kiir1'?
 sz->kiir2();
 ^~~~~_
 kiir1
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$
```

13.5. ábra. Szulo-gyerek.cpp fordítása

Mit látunk?

Az eredmény itt is ugyan az, a "kiir2"-öt nem lehet meghívni a példányra; (A "kiir1" itt is gondnálkül meghívható lenne, fordulna és futna a program hiba nélkül.). C++-ban a "virtual" kulcsszóval tudjuk elérni a dinamikus kötést, úgy hogy az ősben a viselkedést virtuálisra deklaráljuk. Valamint itt is igaz hogy az ősosztály referenciaján vagy mutatóján keresztül csak az ős üzeneteit képes továbbítani.

C++-ban több opciónk is lett volna a példányosításnál, én a mutatóval/pointerrel dolgoztam! Lehetséges opcionális paraméterek:

```
Szulo& sz = new Gyerek(); //referencia
Szulo* sz = new Gyerek(); //mutató
Szulo sz = new Gyerek(); //objektum
```

### 13.3. Anti OO

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

Az előző fejezetben már a BBP algoritmusról eset szó, most pedig ebből indulunk ki. A feladat az lesz hogy összehasonlítsük 4 különböző programozási nyelvben a futási időket. De minek a futását?

A BBP algoritmus Pi hexadecimális kifejetésének a 0. pozíciótól számított 1000000, 10000000, 100000000 darab jegyét. Ezeket a kódokat a feladatban megemlített oldalról vettettem, ugyanis így biztos, hogy egy működő, ráadásul ugyanúgy működő kódról van szó. Tehát más feladatuktól eltérően itt Bátfai Tanár úr kódját használtam mindenhol.

Az eredmény annyira nem eltérő mint talán azt gondolnánk. Először is egy kis táblázattal szeretném szemléltetni az eredményeket:

|      | $10^6$   | $10^7$    | $10^8$     |
|------|----------|-----------|------------|
| Java | 1.67     | 19.604    | 224.494    |
| C#   | 3.327    | 24.728    | 296.946    |
| C    | 1.866485 | 22.014892 | 251.213007 |
| C++  | 1.870038 | 21.871267 | 251.460414 |

13.6. ábra. Eredmények

Mint az láthatjuk nálam a java volt a leggyorsabb, még a C# a leglasabb, a C és a C++ között pedig nem figyelhető meg nagy eltérés, bár szerintem ez nem meglepő eredmény.

A C# utolsó helyének lehet az is az oka nálam, hogy azt Windowson futattam, de alapvetőleg is "hátránnal" indult egyértelműen.

Nem teljesen volt fair azért ez az összehasonlítás, mert például a Javának vannak automatikus "optimalizáló" tulajdonságai, de valószínűleg nem tér el sokkal az igazságtól az eredmény.

A végére pedig belinkelném a kódokat, habár nem saját kódok, azért lehessen látni hogy mivel dolgoztam.

C és a C++:

[Anti.c](#)

[Anti.cpp](#)

Valamint a futásuk után:(nem sok a különbség ezért csak a c++)

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ g++ -o Anti Anti.cpp
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$./Anti
6
1.870038
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ g++ -o Anti Anti.cpp
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$./Anti
7
21.871267
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ g++ -o Anti Anti.cpp
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$./Anti
12
251.460414
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$
```

13.7. ábra. Anti.cpp futása

A C# kód:

[Anti.cs](#)

Valamint a java kód:

[PiBBPBench.java](#)

Valamint a java futás alatt:(C#-ról nincs képem mivel windows alatt csináltam és nem készült screenshot)

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ javac PiBBPBench.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ java PiBBPBench
6
1.67
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ javac PiBBPBench.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ java PiBBPBench
7
19.604
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ javac PiBBPBench.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$ java PiBBPBench
12
224.494
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Liskov$
```

13.8. ábra. PiBBPBench.java futása

## 13.4. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/-SMNISTforHumansExp3/app/src/main> Apró módosításokat eszközölj benne, pl. színvilág.

**A feladat megoldásában tutorált engem Győri Márk Patrik!**

A feladatot android studióval és azon belül emolátorral végeztem el Windowson. Ennek a beállításában segített Márk. Ugyanis én még nem használtam ezt a programot soha sem. De épp ezért talán az egyik legizgalmasabb feladat volt ez.

Mivel maga az eredeti kód se akarta az igazságot itt is a Márk által kissé módosított kódot használtam, remélem ez nem probléma. Részletesebben valószínűleg az Ő könyvében olvashatunk a kódról, hogy mit is módosított rajta vagy hogy bírta életre.

A feladat az volt hogy az SMNIST újra élesztését követően kicsit játszunk el vele, változtassuk meg pl a háttér színét vagy a szöveg színt/méretet hasonlók.

Én a színek megváltoztatását választottam, a következő képeken látható hogy én főként a háttér megjelenési színét változtattam, mind a főháttér, mind pedig annak a háttérnek a színét amin a pontok megjelennek.

A fő háttér színét állítottam át, a pirossal aláhúzott rész került módosításra az eredmény érdekében. Alapjába véve a piros és a zöld volt a fő háttér színei, én a következő két képen látható szín árnyalatokat használtam:

```
private float height = 1000;
protected float fromx;
protected float fromy;

private android.view.SurfaceHolder surfaceHolder;
private android.view.ScaleGestureDetector scaleGestureDetector;
private float scaleFactor = 1.0f;

private boolean running = true;
private static int decisionTimeLimit = 1000;

private int semValue = 0;
private static final int SEM_VALUE_LIMIT_START = 3;
private static int semValueLimit = SEM_VALUE_LIMIT_START;
private static final int NUM_OF_DIGITS = 10;
private float[] semValueDots = new float[2 * NUM_OF_DIGITS];

private float[] digitsCoords = new float[2 * NUM_OF_DIGITS];

private int successCnt = 0;
private boolean armed = true;
private long armedTime = 0;

private Random rand = new Random();

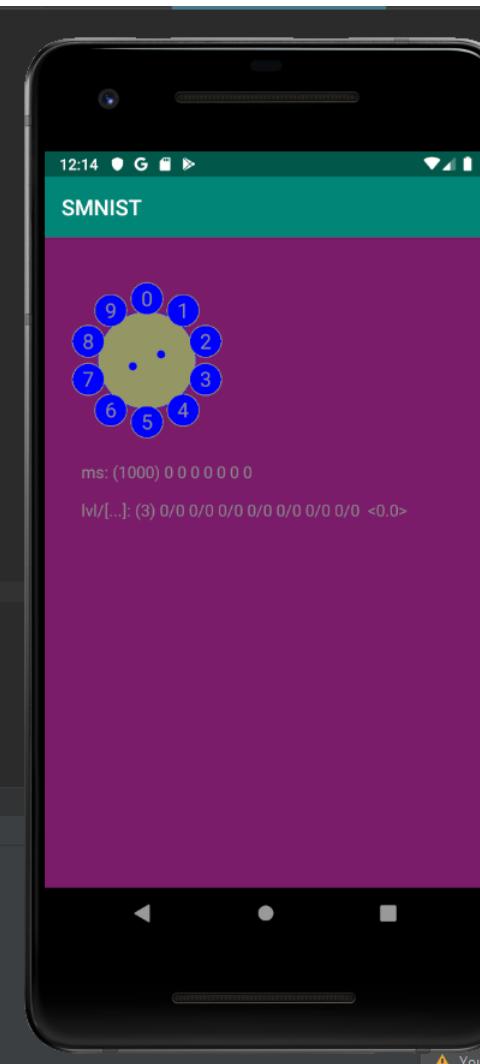
int[] bgColor =
{
 android.graphics.Color.rgb(red: 127, green: 28, blue: 104),
 android.graphics.Color.rgb(red: 54, green: 88, blue: 76)
};

int bgIdx = 0;

private int decisionTimeDelaySum = 0;
private int semValueSum = 0;

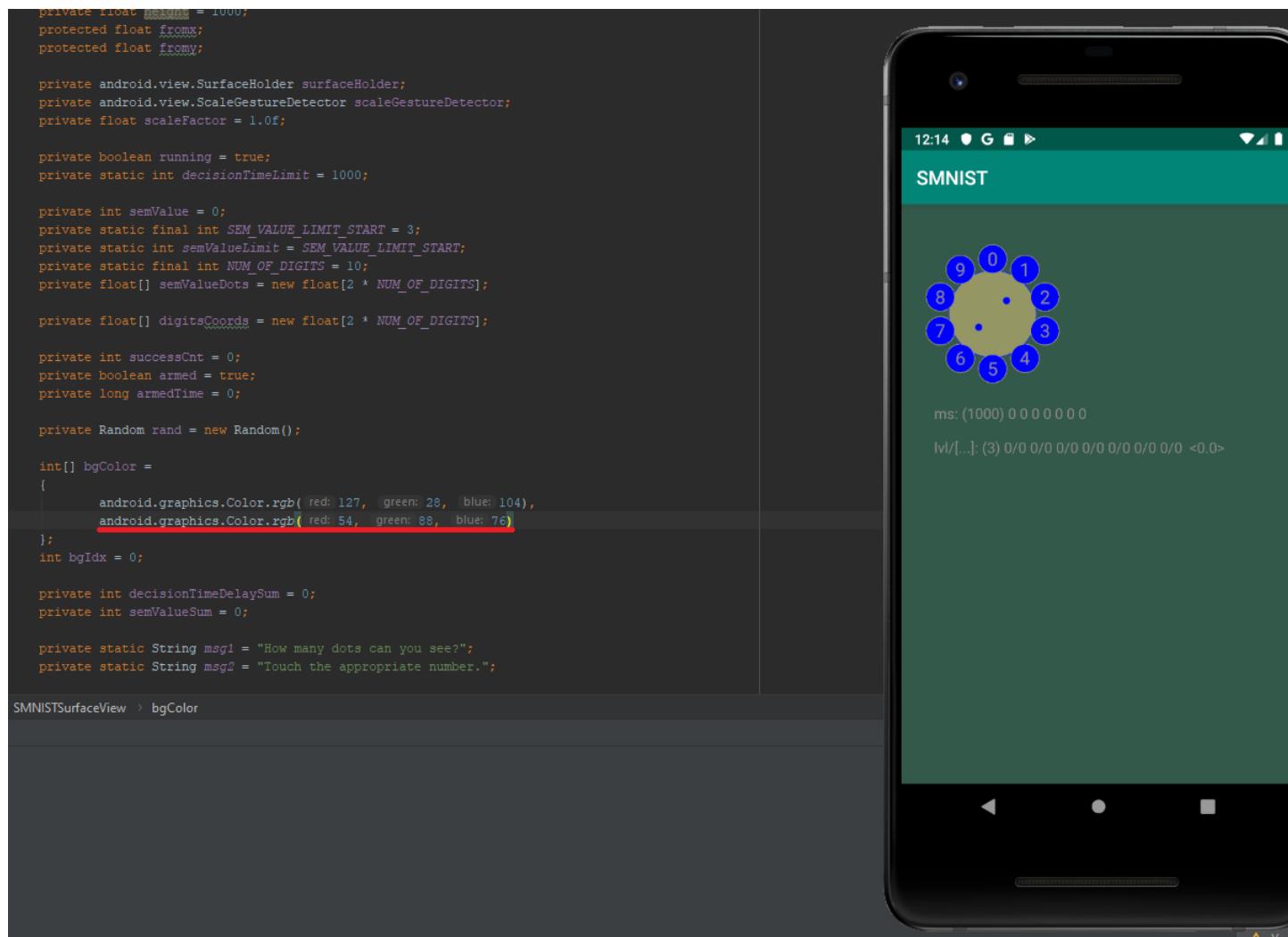
private static String msg1 = "How many dots can you see?";
private static String msg2 = "Touch the appropriate number.";
```

SMNISTSurfaceView > bgColor



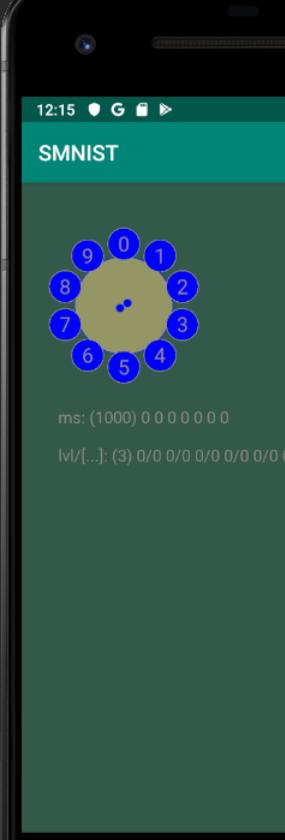
13.9. ábra. SMNIST háttér szín

Valamint a másik fő háttért is módosítottam, RGB színkód alapján:



13.10. ábra. SMNIST háttér szín

Továbbá a pontok megjelenése alatti háttér réteg színét is módosítottam, ez eredetileg nem RGB színkóddal volt megadva, hanem a "YELLOW" szóval ami angolul a citromsárgát jelenti, így a sárgáról módosítottam a lent látható színre a hátteret. Én jobbnak láttam RGB színkóddal megadni ezt is, mivel árnyalatokat és "nem megszokott" színeket is betudtam állítani így!



```
gory > smnist > SMNISTSurfaceView
```

```
STE3Activity.java x ScaleAdapter.java x SMNIST x app x settings.gradle x gradle.properties x activity_smniste3.xml x strings.xml x SMNISTSurfaceView.java x Surface
```

```
 height = newy;
```

```
}
```

```
private void cinit(android.content.Context context) {
```

```
 textPaint.setColor(android.graphics.Color.GRAY);
```

```
 textPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
```

```
 textPaint.setAntiAlias(true);
```

```
 textPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
```

```
 textPaint.setTextSize(50);
```

```
 msgPaint.setColor(android.graphics.Color.GRAY);
```

```
 msgPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
```

```
 msgPaint.setAntiAlias(true);
```

```
 msgPaint.setTextAlign(android.graphics.Paint.Align.LEFT);
```

```
 msgPaint.setTextSize(40);
```

```
 dotPaint.setColor(android.graphics.Color.BLUE);
```

```
 dotPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
```

```
 dotPaint.setAntiAlias(true);
```

```
 dotPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
```

```
 dotPaint.setTextSize(50);
```

```
 borderPaint.setStrokeWidth(2);
```

```
 borderPaint.setColor(android.graphics.Color.rgb(red: 148, green: 149, blue: 100));
```

```
 fillPaint.setStyle(android.graphics.Paint.Style.FILL);
```

```
 fillPaint.setColor(android.graphics.Color.BLUE);
```

```
 surfaceHolder = getHolder();
```

```
 surfaceHolder.addCallback(new SurfaceEvents(surfaceView: this));
```

```
 scaleGestureDetector = new android.view.ScaleGestureDetector(context, new ScaleAdapter(surfaceView: this));
```

```
 for (int i = 0; i < digitsCoords.length / 2; ++i) {
```

```
 digitsCoords[2 * i] = (float) (150 * Math.cos(-Math.PI / 2.0 + i * (12.0 / 10.0) * (Math.PI / 6.0)));
```

```
 digitsCoords[2 * i + 1] = (float) (150 * Math.sin(-Math.PI / 2.0 + i * (12.0 / 10.0) * (Math.PI / 6.0)));
```

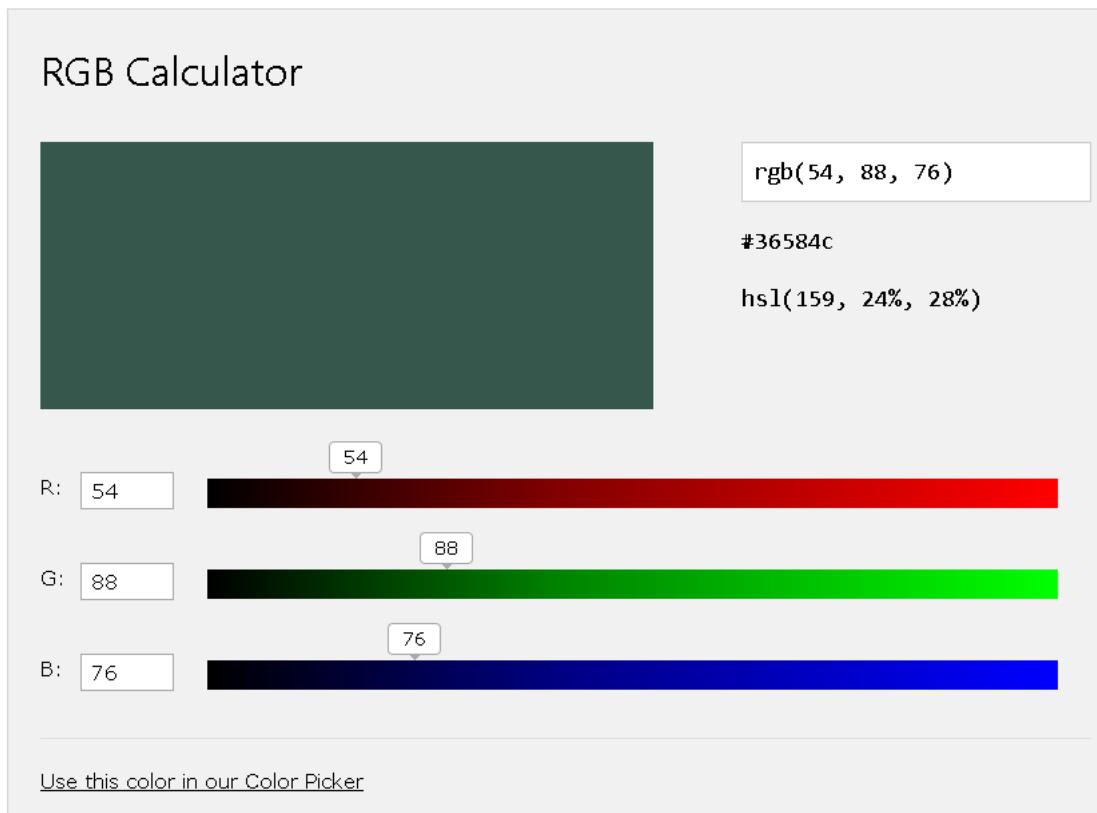
```
 }
```

```
 SMNISTSurfaceView > bgColor
```

13.11. ábra. SMNIST pont háttér

Persze emellett lehet módosítani sok más is, de én ezeket találtam a legszemléletebbnek, ezért ezeket módosítottam a feladat megoldása során.

A színeket RGB színkód alapján adtam meg, de mivel fejből nyilvánvalóan nem tudok olyan sok RGB színkódot (csak az "alapokat", de kicsit eltérőt szerettem volna), ezért a következő online RGB Calculatort vettettem igénybe. A képen pont az egyik fő háttér színének az RGB-jét "számoltam ki" vele.



13.12. ábra. RGB színkód "generálás"

## 14. fejezet

# Helló, Mandelbrot!

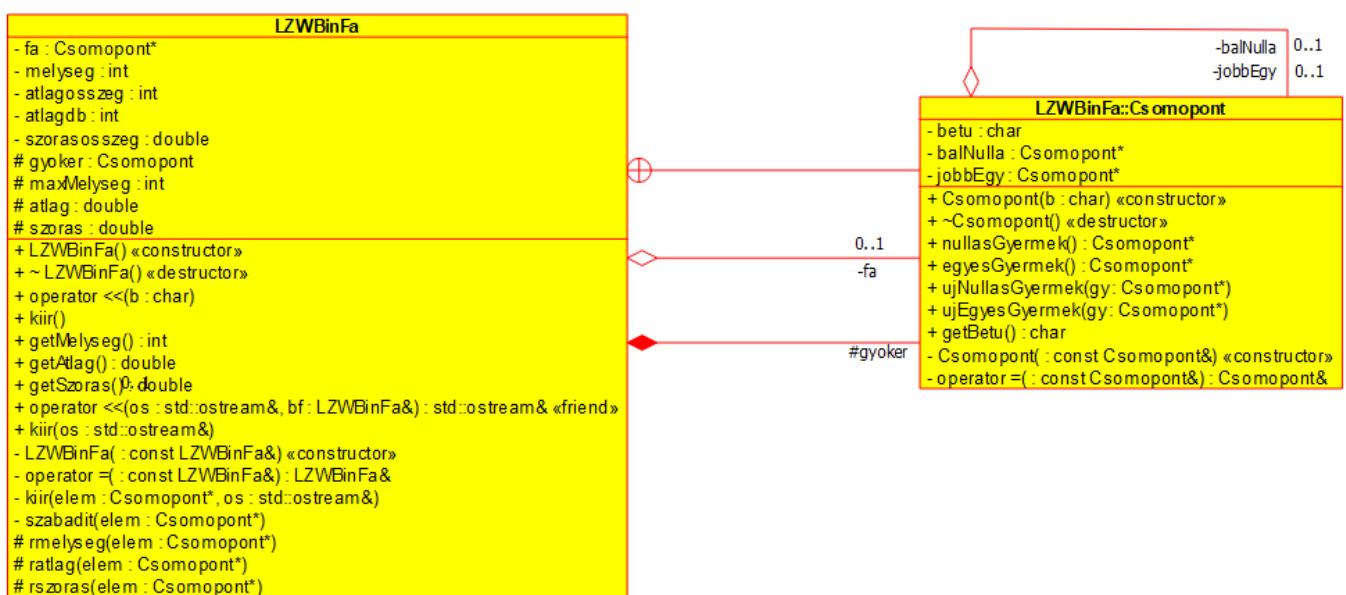
### 14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: [https://youtu.be/Td\\_nIERIEOs](https://youtu.be/Td_nIERIEOs). [https://arato.inf.unideb.hu-batfai.norbert/UDPROG/deprecated/Prog1\\_6.pdf](https://arato.inf.unideb.hu-batfai.norbert/UDPROG/deprecated/Prog1_6.pdf) (28-32 fólia)

A feladatban a jól ismert C++-os binfát kellet UML osztálydiagramként felrajzolni. A forrásból kellett legenerálni pontosabban. A következő forrást generáltam le az Umbrello segítségével:

[Binfa.cpp](#)

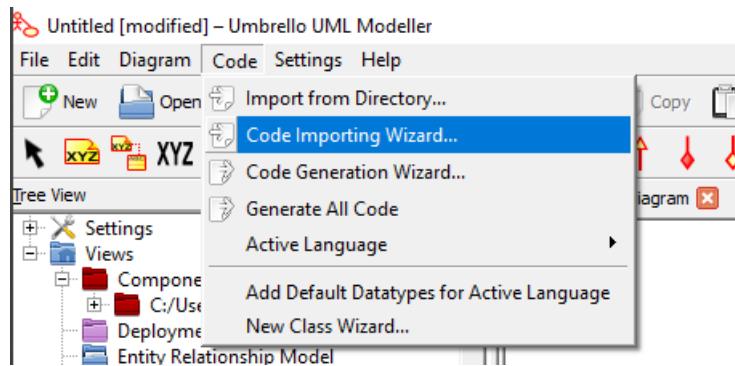
És így néz ki a végeredmény:



14.1. ábra. UML Binfa

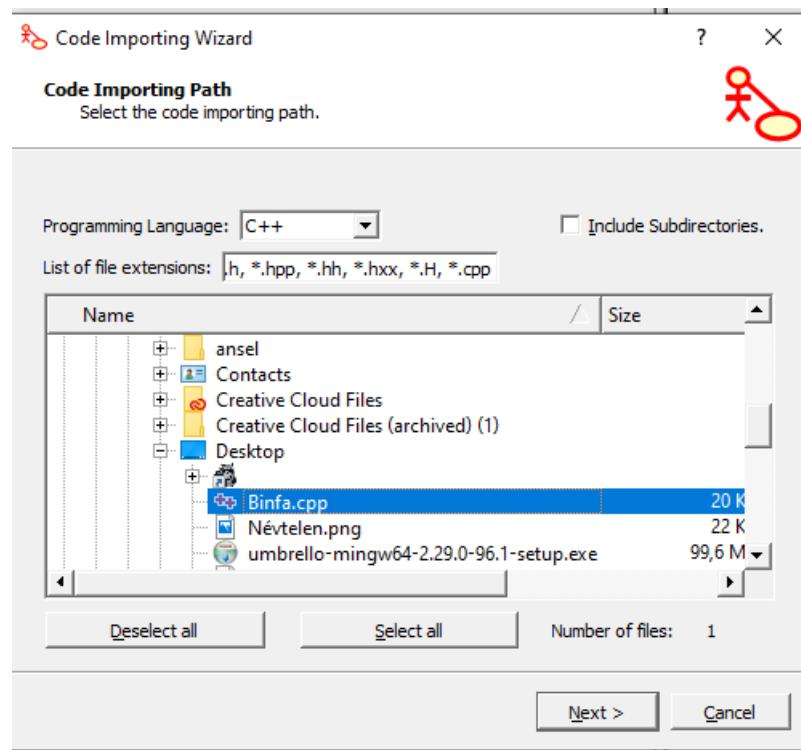
De hogy is jött ez létre?

Először is a "Code" fül alatt a Code Importing Wizardot kellett megnyitni:



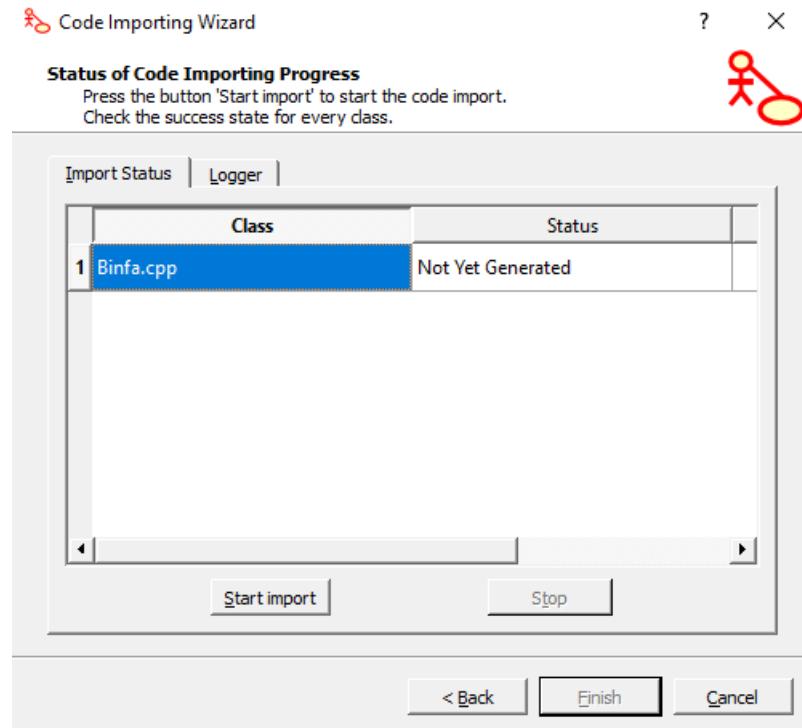
14.2. ábra. Code Importing Wizard

Majd itt kellett bővíteni a "List of file extensions"-t a "\*.cpp"-vel ezáltal kitudtuk keresni és "berakni" a "Binfa.cpp"-t.



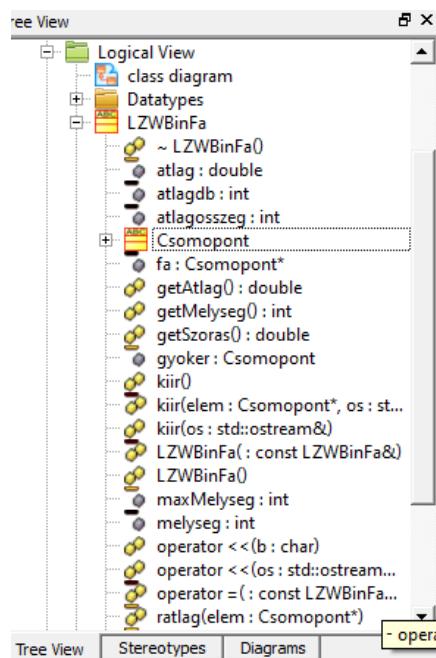
14.3. ábra. Binfa.cpp bevitele

Ezután a "Start import" gombra kattintva kezdhattük meg az importálást.



14.4. ábra. Importálás

Majd importálás után a "Finish" gombra kattintva, a Three view-ba láthatuk is az importált elemeket, amiket behúzva, tudtuk a megfelelő "elemeket/vonalakat" a helyére rakni, és eredményeként kaptuk meg a fent látható képet.



14.5. ábra. Three view

## 14.2. Forward engineering UML osztálydiagram

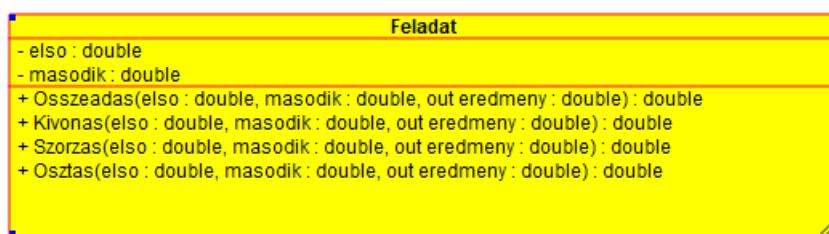
UML-ben tervezünk osztályokat és generálunk belőle forrást!

Ez a feladat az előző feladat fordítója igazából. Tehát most rajzolunk egy UML diagrammot és abból generáljuk le a hozzá tartozó kódot.

Először is létre kell hoznunk egy osztályt és abba kell beaddolnunk minden egyes változót, osztályt, függvényt, metódust, minden.

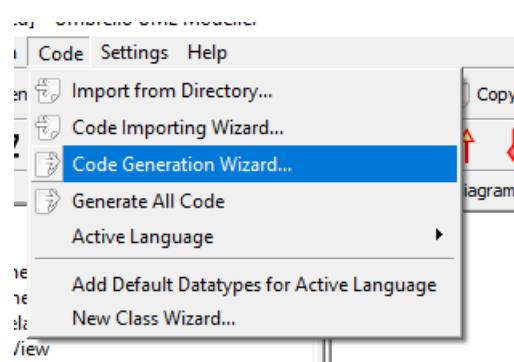
Én még nem csináltam sose ilyet, ezért egy "egyszerű" kis UML diagrammot készítettem, ami két privát változóval, és 4 darab publikus függvényvel rendelkezik.

Amint hozzáadtuk a megfelelő elemeket valami ilyesmit kell látnunk:



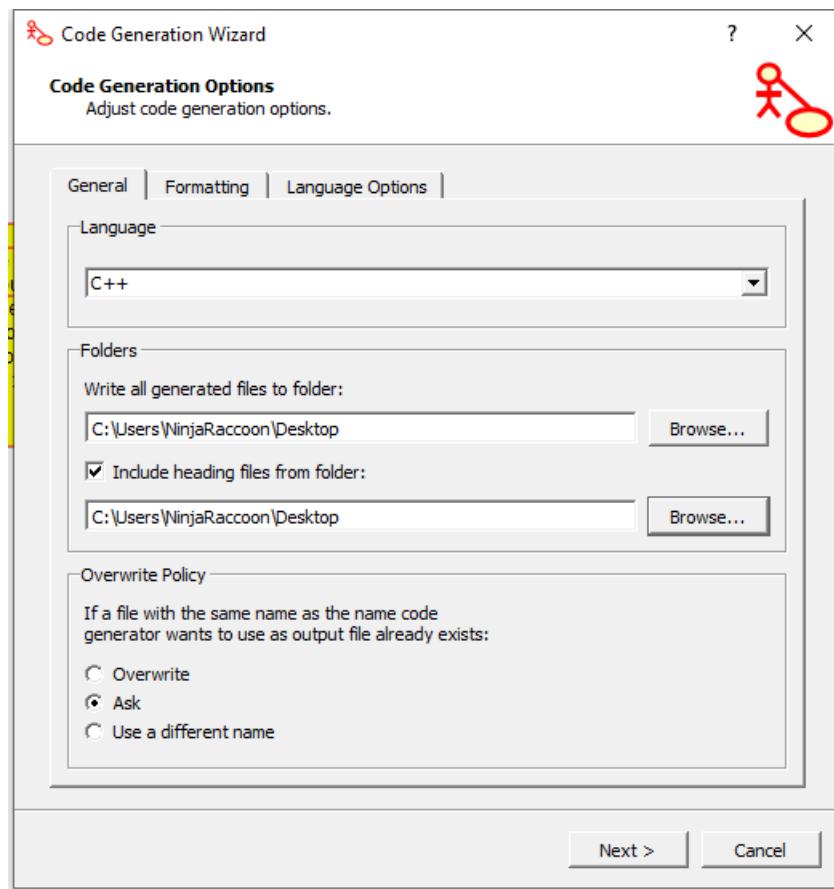
14.6. ábra. UML diagram

Ezután a az előző feladathoz hasonlóan a "Code" fül lenyitásával kezdünk, csak ez esetben viszont a Code Generation Wizard menüpontot kell választanunk.



14.7. ábra. Code Generation Wizard

Majd a következő ablak fogad minket:



14.8. ábra. Code Generation Options

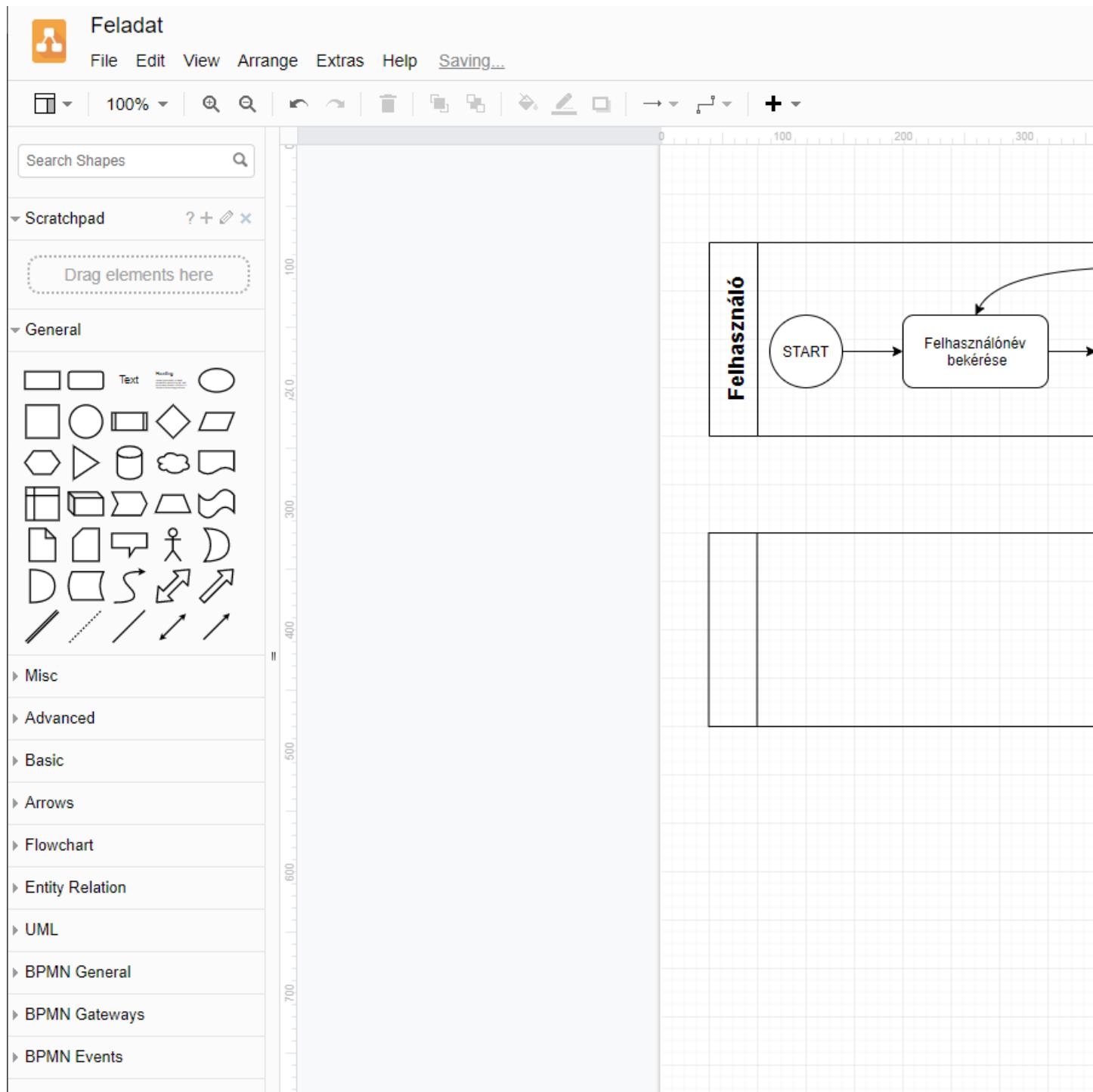
Itt tudjuk beállítani a forrás nyelvét, tehát hogy mire generálja a diagrammunk, hogy hova tegye minden ezt, és egyéb beállításokat. Next gomb segítségével haladhatunk az oldalokon, ha mindenkel megvagyunk a Generate gomb segítségével le is generálja nekünk egy header fájl és egy .cpp fájl formájában, ha a C++-t választottuk.

### 14.3. BPMN

Rajzolunk le egy tevékenységet BPMN-ben! [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated-Prog2\\_7.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated-Prog2_7.pdf) (34-47 fólia)

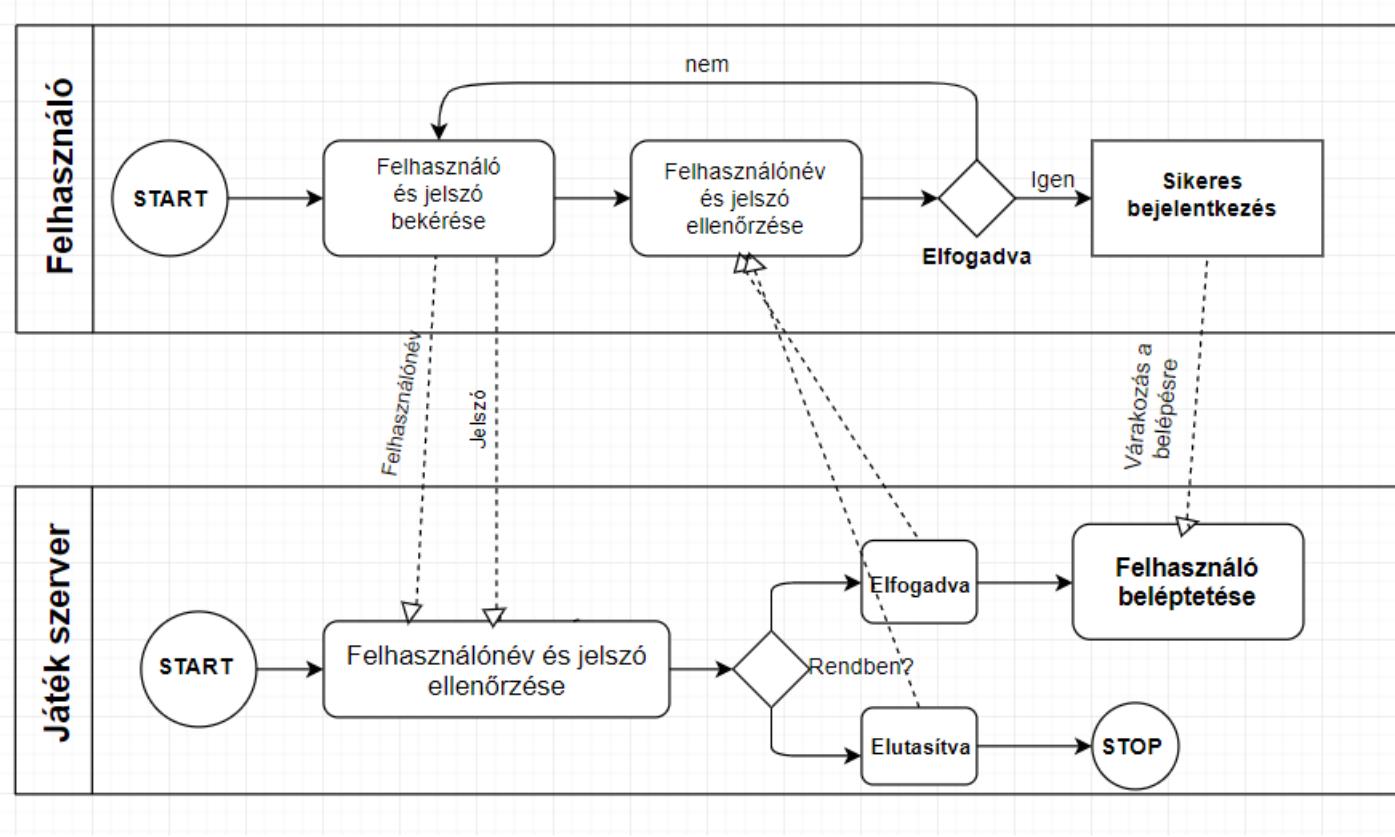
Ebben a feladatban valamelyen irl folyamatot/tevékenységet kellett ábrázolni. Én egy játékba bejelentkezést ábrázoltam. Az ábrázoláshoz a Google draw.io-ját alkalmaztam. Ugyanis könnyen kezelhető, és kis erő befektetéssel is szép ábrákat lehet vele készíteni.

Kezelőfelülete a következőképpen néz ki:



14.9. ábra. Draw.io kezelőfelülete

Én úgy gondolom, sok magyarázatot nem igényel az elkészített ábra, ugyanis jól látható a folyamat haladása, az elágazások és a szerver valamint a felhasználó közötti adat csere is könnyen értelmezhető. Itt van tehát a megoldásom:



14.10. ábra. Megoldás

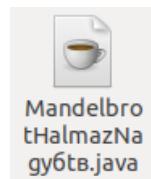
## 15. fejezet

# Helló, Chomsky!

### 15.1. Encoding

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

A feladatban a fenti linken megtalálható MandelbrotHalmaz fájlokat kellett letölteni és használni. Már letöltés után is szembetűnik valami, még pedig az hogy a fájl nevével nincs minden rendben, mert így néz ki:



15.1. ábra. Fájlnév elváltozás

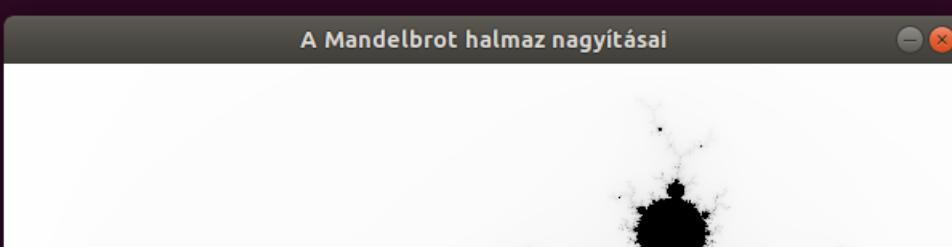
Ezeket a hibás neveket visszaírjuk az eredetire, ezután megpróbáljuk futtatni őket, a futtatás eredménye a következő lesz:

```
csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic_tutorials/bhax_textbook/Chomsky2
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Chomsky2$ javac MandelbrothHalmazNagyító.java
MandelbrothHalmazNagyító.java:2: error: unmappable character (0xED) for encoding UTF-8
 * MandelbrothHalmazNagyító.java
 ^
MandelbrothHalmazNagyító.java:2: error: unmappable character (0xF3) for encoding UTF-8
 * MandelbrothHalmazNagyító.java
 ^
MandelbrothHalmazNagyító.java:4: error: unmappable character (0xED) for encoding UTF-8
 * DIGIT 2005, Javat tanított
 ^
MandelbrothHalmazNagyító.java:5: error: unmappable character (0xE1) for encoding UTF-8
 * Bétfai Norbert, nbatfai@inf.unideb.hu
 ^
MandelbrothHalmazNagyító.java:9: error: unmappable character (0xED) for encoding UTF-8
 * A Mandelbrot halmazt nagyított és kirajzolta osztály.
 ^
MandelbrothHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding UTF-8
 * A Mandelbrot halmazt nagyított és kirajzolta osztály.
 ^
MandelbrothHalmazNagyító.java:9: error: unmappable character (0xE9) for encoding UTF-8
 * A Mandelbrot halmazt nagyított és kirajzolta osztály.
 ^
MandelbrothHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding UTF-8
 * A Mandelbrot halmazt nagyított és kirajzolta osztály.
 ^
MandelbrothHalmazNagyító.java:9: error: unmappable character (0xE1) for encoding UTF-8
 * A Mandelbrot halmazt nagyított és kirajzolta osztály.
 ^
MandelbrothHalmazNagyító.java:11: error: unmappable character (0xE1) for encoding UTF-8
 * @author Bétfai Norbert, nbatfai@inf.unideb.hu
 ^
MandelbrothHalmazNagyító.java:14: error: unmappable character (0xED) for encoding UTF-8
public class MandelbrothHalmazNagyító extends MandelbrothHalmaz {
 ^
MandelbrothHalmazNagyító.java:14: error: unmappable character (0xF3) for encoding UTF-8
public class MandelbrothHalmazNagyító extends MandelbrothHalmaz {
 ^
MandelbrothHalmazNagyító.java:15: error: unmappable character (0xED) for encoding UTF-8
 /** A nagyítandó kijelölt területet bal felső sarka. */
 ^
MandelbrothHalmazNagyító.java:15: error: unmappable character (0xF3) for encoding UTF-8
 /** A nagyítandó kijelölt területet bal felső sarka. */
 ^
MandelbrothHalmazNagyító.java:15: error: unmappable character (0xF6) for encoding UTF-8
 /** A nagyítandó kijelölt területet bal felső sarka. */
 ^
MandelbrothHalmazNagyító.java:15: error: unmappable character (0xFC) for encoding UTF-8
 /** A nagyítandó kijelölt területet bal felső sarka. */
 ^
```

## 15.2. ábra. Fordítási hiba

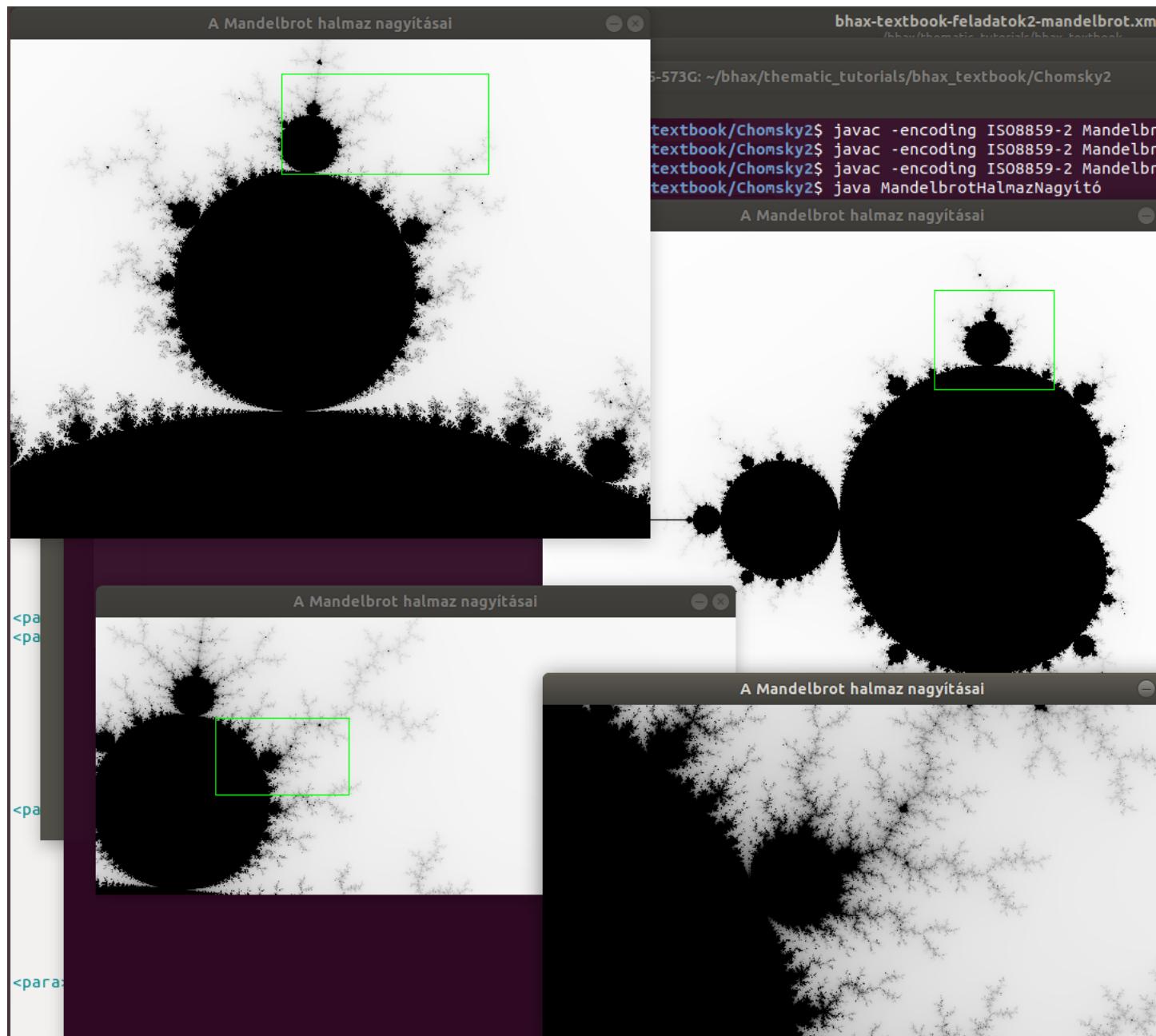
Hogy miért van ez, azt a hibakód egyértelműen elárulja, azaz az UTF-8-as kódolásban nem megtalálható karaktereket tartalmaz. És itt kezdődik igazából a feladat, ezt a problémát kell orvosolnunk. Amit az "-encoding" kapcsolóval lehetünk meg. Haználata egyszerű minden össze az "-encoding" kapcsoló után írni egy kódolás megnevezését ami tartalmazza a magyar ő, ű és egyéb ékezetes betűket. Erre a célra teljesen megfelel az ISO8859-2 (Latin 2) kódolás, így én azt használom. Tehát a következőképpen kell fordítanunk:

```
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Chomsky2$ javac -encoding ISO8859_2 Chomsky2.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Chomsky2$ javac -encoding ISO8859_2 MandelbrotHalmaz.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Chomsky2$ javac -encoding ISO8859_2 MandelbrotHalmaz2.java
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Chomsky2$ java MandelbrotHalmaz2
A Mandelbrot halmaz nagyításai
```



15.3. ábra. Fordítás hiba nélkül

Így láthatjuk hogy hiba nélkül lefordulnak a fájlaink és még futtatni is tudjuk azt.



15.4. ábra. MandelbrotHalmazNagyító futása

## 15.2. Full screen

Készítsünk egy teljes képernyős Java programot!

Ez a feladat kicsit eltér a többitől, mert itt grafikus területen is dolgozni fogunk többet, használunk több képet is a megoldáshoz. Valamint teljes képernyőn ábrázolunk nem pedig csak egy ablakban. Valamint mozgatható karakterünk lesz, és az Esc billentyű lenyomásával kiléphetünk a "játékból". A feladathoz Bátfa Tanár úr Labirintus játékát veszem alapul.

A különböző KeyEventeket nem részletezném, a forrásban szépen fel van kommentelve, hogy mi miért van, a feladatra koncentrálva a teljes képernyő megoldását részletezném.

A teljesképernyős mód nem minden eszközön érhető el, ezért meg kell néznünk hogy át tudunk-e váltani full screen módba.

```
boolean fullScreenTámogatott = graphicsDevice.isFullScreenSupported();
```

Ha lehetséges akkor át is váltunk, valamint elkérjük a képernyő adatait, ez alatt a frekvenciát a magasságot a szélességet (felbontást), színmélységet értjük. Nyilvánvalóan ez fontos hisz, különböző adatokkal rendelkezik minden eszköz, a mi példánkban például a felbontás 1024x768 lesz, persze ez átállítható/személyre szabható.

```
if(fullScreenTámogatott) {
 graphicsDevice.setFullScreenWindow(this);
 java.awt.DisplayMode displayMode
 = graphicsDevice.getDisplayMode();
 szélesség = displayMode.getWidth();
 magasság = displayMode.getHeight();
 int színMélység = displayMode.getBitDepth();
 int frissítésiFrekvencia = displayMode.getRefreshRate();
 System.out.println(szélesség
 + "x" + magasság
 + ", " + színMélység
 + ", " + frissítésiFrekvencia);
```

Lehetséges képernyő beállításokat elkérjük valamint megnézzük hogy a fentebb említett felbontást támogatja-e ugyanis a képein erre a felbontásra készültek.

```
java.awt.DisplayMode[] displayModes
 = graphicsDevice.getDisplayModes();
boolean dm1024x768 = false;
for(int i=0; i<displayModes.length; ++i) {
 if(displayModes[i].getWidth() == 1024
 && displayModes[i].getHeight() == 768
 && displayModes[i].getBitDepth() == színMélység
 && displayModes[i].getRefreshRate()
 == frissítésiFrekvencia) {
 graphicsDevice.setDisplayMode(displayModes[i]);
 dm1024x768 = true;
 break;
}
```

Ha nem lehetséges elérni ezt a felbontást azt közöljük a felhasználóval a következőképpen:

```
if(!dm1024x768)
 System.out.println("Nem megy az 1024x768, de a példa ←
 képméretei ehhez a felbontáshoz vannak állítva.");
```

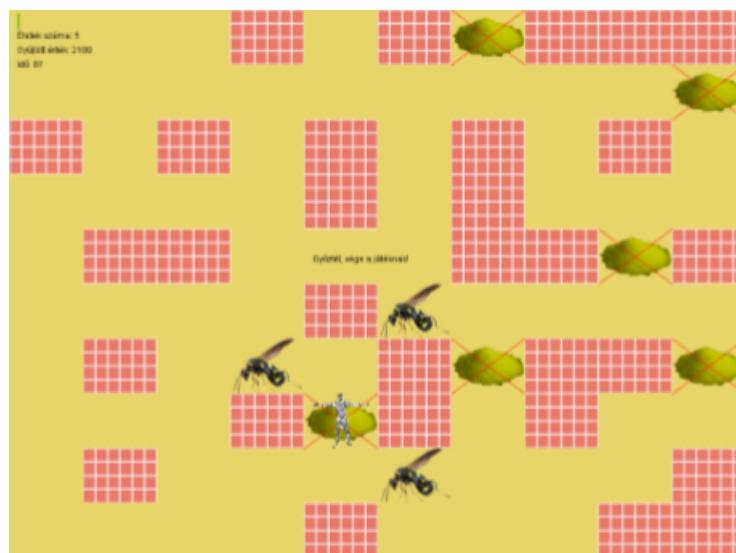
A feladat szempontjából a kód ezen része volt fontos, de érdemes megnézni az egész kódot és a kommentelését, mert a kis játékunk irányítása és a világ megalkotása is szépen ki van vitelezve.

Felcommentelt fő forrás fájl:

### LabirintusJáték

A Chomsky2 mappán belül a Labirintus mappában megtalálhatóak a szükséges képek és egyébb források is, azokat most nem linkelném!

Játék kinézete futás közben:



15.5. ábra. Full screen game

## 15.3. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! Lásd <https://youtu.be/XpBnR31BRJY>

A perceptron egy algoritmus ami a gépi tanulásban játszik fontos szerepet. A bináris osztályzók tanulásában is fontos szerepet játszanak, ugyanis ezek munkaköre az hogy eltudja dönteni/el is dönti, hogy az input specifikus osztályhoz tartozik-e vagy sem. A perceptronról bővebben angolul olvashatunk a [Wikipédia oldalán](#).

Továbbá még kiemelném a perceptron felépítését, mely 3 fő részből áll:

- A retinának nevezett első elem, ami a bemeneti jeleket fogadó cellákat tartalmazza.
- Az asszociatív cellák, melyek összegzik a hozzájuk érkező jeleket, impulzusokat.
- A döntési cellák rétege a perceptronok kimenetele. Az asszociatív cellákhoz hasonló képpen működnek ezek is.

Részletesebben erről és egyéb perceptron információról olvashatunk magyar nyelven a [következő oldalon](#).

A fenti részeket a Prog1-es perceptronos feladatomból emeltem át, egy kis ismétlés/ismertetés szempontjából. A mostani feladatunk viszont az lesz, hogy kapjon egy képet bemenetként (A mandelbrot.cpp által létrehozott Mandelbrot-halmaz kép tökéletes lesz.) és ennek a képnak fogja venni az egyik színkódját ami majd a többrétegű perceptronunk bemenete lesz.

Első lépésként is include-olnunk kell a megfelelő könyvtárakat. Mivel szükségünk lesz a többrétegűségre ezért kell az mlp (Multi Layer Perceptron) könyvtára is, és mivel a PNG képpel fogunk dolgozni ezért a png könyvtárra is szükség lesz, ezeket így tudjuk majd elérni:

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

A main elején a képünk beolvasása történik, így tudunk rajta dolgozni, még hozzá a "get\_width" és a "get\_height" segítségével. Valamint a new operátor segítségével létrehozzuk a perceptronunkat.

```
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, 1);
```

Ezután egy double típusú változót hozzunk létre. Valamint a for ciklusok végig mennek a kép szélesség és magassági pontokon. Miután végigmentünk a képpontokon, az image változó tárolni fogja a képállomány vörös színkomponensét. A value pedig azt a double típusú számot amit majd kiíratunk a végén.

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

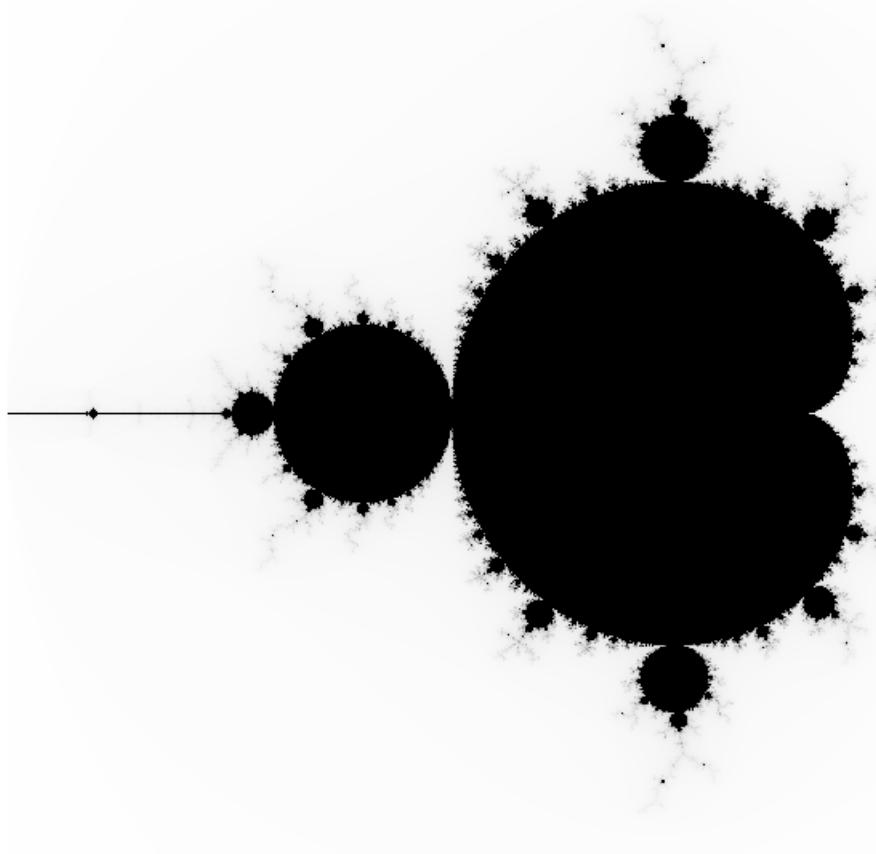
double value = (*p)(image);

cout << value << endl;
```

A kódunk végén pedig elvégezzük a szükséges hely felszabadítást a memóriában, amit a következő képpen teszünk:

```
delete p;
delete [] image;
```

Ha a kódunkat megfelelően fordítjuk, és rendelkezünk a szükséges fájlokkal akkor a következő fog minket fogadni png fájlként:



15.6. ábra. Mandel

A megfelelő fordítás és futtatás egyébiránt a következő féleképpen néz ki:

```
g++ mlp.hpp main.cpp -o mandel -lpng -std=c++11
./mandel mandel.png
```

# 16. fejezet

## Helló, Stroustrup!

### 16.1. JDK osztályok

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

Ezt a feladatot a feladat leírásában ajánlott fénykard kódóból kiindulva készítem el. Ezt a kódot Bátfai Tanár úr repójából szedtem, amelyet ide lentre be is linkelek:

<https://github.com/nbatfai/future/blob/master/cs/F7/fenykard.cpp?fbclid=IwAR1bLKhMxq-HIXphsQKDexVKUoPIkOyskYQE9kbCO4uU> Fénykard

De mi is fontos nekünk a fénykard kódóból?

A legfontosabb a "read\_acts" nevű függvény lesz, ugyanis ezen kell majd módosítanunk hogy a feladatot elvégezhessük.

```
void read_acts(boost::filesystem::path path, std::map<std::string, int> &←
acts)
{
 if (is_regular_file(path)) {

 std::string ext(".props");
 if (!ext.compare(boost::filesystem::extension(path))) {

 std::string actpropspath = path.string();
 std::size_t end = actpropspath.find_last_of("/");
 std::string act = actpropspath.substr(0, end);

 acts[act] = get_points(path);

 std::cout << std::setw(4) << acts[act] << " " << act << std::endl;
 }
 }
 else if (is_directory(path))
```

```
for (boost::filesystem::directory_entry & entry : boost::filesystem::<-
 directory_iterator(path))
 readActs(entry.path(), acts);

}
```

Amint láthatjuk ez lesz felelős a rekurzív bejárásért, és az összes .props fájlt kilistázná nekünk, de a feladat nem ezt kéri hanem a .java fájlokat. Eszerint kell módosítani!

```
void readClasses(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))
 readClasses(entry.path(), classes);
}
```

A különbség jól látszik, maga a függvény neve is readClasses lett, persze ez nem lényeges, de viszont actokra nem lesz szükség, ugye most az osztályok fontosak nekünk, így magába a bejárásba, a kódba se törődünk az actokkal, azt a részt egyszerűen "classes.push\_back(path.string());" cseréljük az if ágban, hisz ez nekünk a feladat teljesítéséhez elegendő. Az elseifbe pedig csak a neveket írjuk át, mármint az acts-okat classes-re.

Tehát maga a kód azt fogja tenni nekünk egész egyszerűen, hogy a bekapott állományszerkezeten végig megy és a keresett .java kiterjesztésű fájlokat megkeresi, és azt eltárolja, ha mappa a vizsgált elem akkor azt "megnyitja" és azon belül vizsgálja az elemeket, tehát rekurzívan bejárja az egész állományszerkezetet, és az összes .java kiterjesztésű fájlt eltárolja nekünk. Jelen esetben vissza is adván azt a standard outputon.

## 16.2. Hibásan implementált RSA törése

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_3.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_3.pdf) (71-73 fólia) által készített titkos szövegen.

A feladat az hogy egy hibás RSA kódolást hozunk létre, vagyis hogy ne lehessen RSA kódolást visszafejteni, először is mi is az az RSA kódolás?

Az RSA egy titkosító algoritmus amit 1976-ban fejlesztettek ki, de mai napig az egyik leggyakrabban, hanem a leggyakrabban használt titkosítási eljárás. Moduláris számelmélet és a prímszám elmélet tétele adja az alapját. Viszont én nem taglalnám ezeket a matematikai tételeket most, aki megszeretné nézni ezeket a Wikipédia oldalán fellelheti azt.

Ami számunkra fontosabb, az az hogy mi is kell a titkosításhoz vagy hogy is működik az, én utána néztem ezeket, mert eddig én se ismertem. A lényeg az hogy két kulcs van egy nyilvános és egy titkos, a nyilvános kulcs az nyílt tehát mindenki ismeri és segítségével lehet titkosítani, viszont csak a titkos kulcs segítségével feltörhetőek azok.

RSA kódolót az interneten találhatunk több féle képpen kidolgozva is, én a feladat megoldásához és bemutatásához Győri Márk által módosított RSA kódolót/törőt fogom használni.

Először is a megfelelő könyvtákat importálnunk kell, többek között ilyen lesz a buffer reader és filereader ami az olvasáshoz lesz szükséges, valamint ami nagyon fontos a BigInteger.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map.Entry;
```

Nézzük is meg mi fog szerepelni a mainben!

```
public class rsa_chiper {
 public static void main(String[] args) {
 int bitlength = 2100;

 SecureRandom random = new SecureRandom();

 BigInteger p = BigInteger.probablePrime(bitlength/2, random);
 BigInteger q = BigInteger.probablePrime(bitlength/2, random);

 BigInteger publicKey = new BigInteger("65537");
 BigInteger modulus = p.multiply(q);

 String str = "this is a perfect string".toUpperCase();
 System.out.println("Eredeti: " + str);

 byte[] out = new byte[str.length()];
 for (int i = 0; i < str.length(); i++) {
 char c = str.charAt(i);
 if (c == ' ')
 out[i] = (byte)c;
 else
 out[i] = new BigInteger(new byte[] {(byte)c}).modPow(publicKey, ←
 modulus).byteValue();
 }
 String encoded = new String(out);
 System.out.println("Kodolt: " + encoded);

 Decode de = new Decode(encoded);
 System.out.println("Visszafejtett: " + de.getDecoded());
 }
}
```

Ugye a fő osztály az rsa\_chiper lesz amiben a main is szerepel. Mint azt látjuk a mainbe történik a szövegünk titkosítása, valamint a kulcs is itt kerül meghatározásra. A titkosítani kívánt szöveg jelen esetünkben a "this is a perfect string" lesz. Ezt titkosítva próbálja majd a program visszafejtjeni. Nyilván több kevesebb sikkerrel :D. A titkosítás betűről betűre történik a következő csipetben:

```
byte[] out = new byte[str.length()];
for (int i = 0; i < str.length(); i++) {
 char c = str.charAt(i);
 if (c == ' ')
 out[i] = (byte)c;
 else
 out[i] = new BigInteger(new byte[] {(byte)c}).modPow(publicKey, ←
 modulus).byteValue();
}
```

A titkosított szöveg úgy áll elő, hogy a visszaadott bytokból számunkra olvashatatlan szöveget generál a program.

A visszafejtéshez viszont szükségünk lesz egy betű gyakorisági listára, mivel az algoritmus a betűk gyakorisága szerint helyettesíti be a karaktereket, így érdemes a listát úgy elkészíteni hogy megfelelő legyen a betűgyakoriság. (Például a magyar nyelvben az e betű a leggyakoribb betű.) Ez a lista kódban így néz ki:

```
private void loadFreqList() {
 BufferedReader reader;
 try {
 reader = new BufferedReader(new FileReader("freq.txt"));
 String line;
 while((line = reader.readLine()) != null) {
 String[] args = line.split("\t");
 char c = args[0].charAt(0);
 int num = Integer.parseInt(args[1]);
 this.charRank.put(c, num);
 }
 } catch (Exception e) {
 System.out.println("Error when loading list -> " + e.getMessage());
 }
}
```

Ez a függvény beolvassa a gyakoriság listát, és ha meghívásra kerül karakterenként megvizsgálja az elemeket, ha a vizsgált betű már benne van a listába akkor +1-el növeli a gyakoriságának az értékét, ha pedig nincs benne akkor belerakja azt 1-es kezdő értékkel. Whitespace karakterek esetén a vizsgálás tovább megy, a spaceket nem számolja, csakis a betűket! Értelem szerűen a legyakrabban használt betűnek lesz így a legnagyobb értéke, és a helyettesítésnél az algoritmus a nagyobb értékű karaktereket helyezik prioritásba.

Ezután már csak annyit tesz a program hogy a nextFreq függvénytel egy egyszerű maximum kiválasztásos rendszerrel a listából behelyettesíti a karaktereket, és a behelyettesítetett "kiveszi" belőle, így a lsita a végére kiürül mi meg ha minden jól megy az eredeti szöveget vagy ahoz hasonló megoldást kapunk, de persze egyáltalán nem pontos ez. A listától függően lehet aránylag türhető megoldás vagy egyszerűen nem is hasonló az eredmény.

```
private char nextFreq() {
 char c = 0;
 int nowFreq = 0;
 for(Entry<Character, Integer> e : this.charRank.entrySet()) {
 if (e.getValue() > nowFreq) {
 nowFreq = e.getValue();
 }
 }
}
```

```

 c = e.getKey();
 }
}
if (this.charRank.containsKey(c))
 this.charRank.remove(c);
return c;
}

```

A getDecode függvény vissza adja nekünk a "visszafejtett" szövegünket, ám az eredmény jelen esetben se lenyűgöző annyira...

Fordítása után ha futtatjuk, és a megfelelő listát a mappában tároljuk (jelen esetben "freq.txt" formában), akkor futtatásnál a következőt fogjuk látni:

```

csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic_tutorials/bhax_textbook/Stroustrup
File Edit View Search Terminal Help
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Stroustrup$ java rsa_chiper
Eredeti: THIS IS A PERFECT STRING
Kodolt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Visszafejtett: SCTI TI I AEIEFS ISITR
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Stroustrup$

```

16.1. ábra. RSA

### 16.3. Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

A feladat az előző csokor Perceptronos feladatjára épül vagyis abból ki lehet indulni. Így egyes részei meg is egyeznek a kódnak.

Első lépésként is include-olunk kell a megfelelő könyvtárakat. Mivel szükségünk lesz a többrétegűségre ezért kell az mlp (Multi Layer Perceptron) könyvtára is, és mivel a PNG képpel fogunk dolgozni ezért a png könyvtárra is szükség lesz, ezeket így tudjuk majd elérni:

```

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

```

A main elején a képünk beolvasása történik, így tudunk rajta dolgozni, még hozzá a "get\_width" és a "get\_height" segítségével. Valamint a new operátor segítségével létrehozzuk a perceptronunkat.

```

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, 1);
```

Ezután egy double típusú mutatót hozzunk létre. Valamint a for ciklusok végig mennek a kép szélesség és magassági pontokon. A képpontokat az "image" tárolni fogja a képalomány vörös színkomponensét (mivel a kódban ezt adtuk meg).

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

Ez eddig az előző csokor perceptronos feladatával ekvivalens, itt jön be a különbség

Ugyanis most nem egy value double típusú változóra van szükségünk, mivel nem arra vágyunk, hogy egy értéket írjon ki nekünk, hanem egy képet akarunk generáltatni vele. Így tehát a double\* típusra lesz szükségünk. Így tehát szükségünk lesz újabb két for ciklusra ami szintén a megszokott feladatot kapja, azaz az egyik a kép magassági pontjain még a másik a szélességi pontjain lépked. Ez ugye azért fog kelleni hogy az új képünk a megfelelő adatokat kapja meg és az alapján generálja le a képünket. Valamint output png kiterjesztésű képet létrehozza nekünk a write függvény, és ez lesz az ami nekünk kell.

```
double* newPicture = (*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 = newPicture[i*png_image.get_width()+j];

png_image.write("output.png");
```

Kép generálást végzünk, a for ciklusok segítségével megkapja a megfelelő színadatokat. A write függvény pedig létre is hozza output néven a kimenetet png formátumban.

A kódunk végén pedig elvégezzük a szükséges hely felszabadítást a memóriában, amit a következő képpen teszünk:

```
delete p;
delete [] image;
```

Nyilván az mlp.hpp-be is módosítanunk kell a kódon, hogy megfelelően működjön a programunk.

```
double* operator() (double image [])
```

Ugye egyértelmű a változtatás a fenti csipeten is, hisz már nem sima double hanem double\*-ot add vissza a ()operátor.

# 17. fejezet

## Helló, Gödel!

### 17.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban <https://youtu.be/DL6iQwPx1Yw>

Ebben a feladatban az std::sort() függvényt fogjuk alkalmazni. Ez a függvény 3 paramétert vár, az első az hogy mettől a második az hogy meddig szeretnénk rendezni a vektort/tömböt. A 3. pedig az, hogy mi alapján szeretnénk rendezni.

Épp ezért, a 3. paraméter miatt, meg kell ismerkedjünk a lambdával, ugyanis a feladatunk az hogy rendezzük a lambdával a gengsztereket.

A lambda szintaxisa a következő:

```
[] (paraméterek) -> visszatérés típusa {utasítások}
```

A felépítése a következő: [] jelek közé a függvényen kívüli változókat kell megadni amiket elszeretnénk érni, aztán a paramétereket kell megadnunk, majd a visszatérést.

De nézzük is meg hogy akkor a jelenesetben az std::sort() függvény és a lambda függvény hogyan épül fel a gengszterekkel.

```
std::sort (gangsters.begin(), gangsters.end(), [this, cop] (Gangster x, ←
 Gangster y,) {
 return dst (cop, x.to) < dst (cop, y.to);
}
};
```

Tehát mit láthatunk?

Talán abban megegyezhetünk, hogy maga a sort függvény nem meglepő, első paraméterként megkapta a gengszterek elejét míg második paraméterként megkapta azok végét a begin() és az end() függvények segítségével, így tehát az egészet fogja vizsgálni, 3. paraméterként pedig megkapta a fent említett és várt lambda függvényt! Viszont nézzük meg inkább a lambdát talán ez az ami érdekesebb.

Külső változónak megadtuk a "this"-t, tehát az aktuális objektumot, és a "cop" objektumot.

```
[this, cop]
```

Paraméternek megadtuk a Gangster x és y objektumot, amik ugye a sort-tal minden összelesznek hasonlítva.  
(Gangster x, Gangster y,)

A vizsgálat pedig a szerint fog történni hogy melyik gengszter van közelebb a rendőrhöz.

```
{
 return dst (cop, x.to) < dst (cop, y.to);
}
```

Így tehát ha x közelebb van a cop-hoz mint y akkor egy true értékkel tér vissza, ezért a vektor elején a rendőrhöz legközelebb álló gengszterek lesznek a vektor végén pedig a legtávolabb állók, így ha a futást befejezte egy teljesen pontos rendezést végez el nekünk.

Tehát a sort és a lambda függvények segítségével, a gangsters vektor elemeit rendőrökötől való távolságuk szerint rendezni tudtuk (növekvő sorrendbe).

## 17.2. STL map érték szerinti rendezése

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

Az STL a Standard Template Library rövidítése, ami egy könyvtár, amely különböző algoritmusokat, tárolókat tartalmaz. Mint például a vector a list vagy a feladatban szereplő map is itt található.

Mi is az a map?

A mapok asszociatív tárolók. A tárolt elemek benne rendelkeznek egy adat értékkel és egy kulcs értékkel. A tárolt adatok ezen kulcs értékek szerint sorba vannak rakva. Forráskódban így néz ki a map:

```
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;
}
```

Létrehozzuk a sort\_map nevű függvényt, amelynek a visszatérési értéke vector párok. Ez string és int párok lesznek, az std::pair-nek hála ez nem probléma hogy különböző típusú párokra van szükségünk. A

függvény paraméterként pedig megkapja a fent említett std::map referenciaját értékül. (mely szintén string és int párokból áll)

```
std::vector<std::pair<std::string, int>> sort_map (std::map <std::string, int> &rank)
```

Következő lépésként létrehozunk egy üres vektort mely a függvény visszatérési értéke lesz, és tartalmazni fogja majd nekünk a string és int párokat.

```
std::vector<std::pair<std::string, int>> ordered;
```

Következő lépében egy for ciklust hozunk létre, mely bejárja nekünk a rank nevű map-et, értékpárok után keresve. Úgy keresi őket hogy egy if-el megnézi hogy tartozik-e második értékpár az adott elemhez, ha igen akkor létre is hozza nekünk az std::pair adatszerkezetet, a megtalált értékeket megadva megfelelő helyre elsőnek és másodiknak. Majd ezt a létrehozott párt, eltárolja az ordered vektorba, amit fent létrehoztunk.

```
for (auto & i : rank) {
 if (i.second) {
 std::pair<std::string, int> p {i.first, i.second};
 ordered.push_back (p);
 }
}
```

Utolsó lépésként pedig az ordered vektorunkat rendezzük, még pedig az első feladatban megismert módszerrel, azaz a sort és a lambda használatával.

```
std::sort (std::begin (ordered), std::end (ordered),
[=] (auto && p1, auto && p2) {
 return p1.second > p2.second;
}
);
```

Mint azt látjuk a rendezés itt is az elejtől a végéig tart, tehát a vektor összes elemét rendezzük megint. 3. paraméterként szintén egy lambda alakú kitevésünk van. Ez esetben a [] között egy "=" szerepel, ami arra utal, hogy másolással fogja most átvenni a változókat nem pedig referenciaival. A paraméterei két auto típusú érték, visszatérési értéke pedig megint boolean típusú, aminek az értékét a p1 és a p2 összehasonlítása fogja megadni.

Ezután már csak a rendezett vektorunkat adjuk vissza:

```
return ordered;
```

### 17.3. Alternatív Tabella rendezése

Mutassuk be a [https://progpater.blog.hu/2011/03/11/alternativ\\_tabella](https://progpater.blog.hu/2011/03/11/alternativ_tabella) a programban a java.lang Interface ComparableT szerepét!

A feladat az hogy készítsünk egy alternatív tabella rendezést a labdarúgó bajnoksághoz, ami ellentében a megszokott rendszerrel nem úgy pontoz hogy a győzelem 3 pont a döntetlen 1 pont a vereség pedig 0,

hanem számításba veszi azt is hogy ki ellen történt a meccs. Így kiküszöbölte azt a problémát, hogy a 3 legerősebb csapat legyőzése is ugyanúgy 9 pontot ér mint a 3 leggyengébb csapat legyőzése.

Ismerős lehet ez nekünk, ami nem véletlen, ugyanis nagyon hasonlít a PageRankhoz, amivel már korábban dolgoztunk. Igazából az alternatív tabella is a PageRankon alapul, mint a google kereső motorja.

Interface Comparable<T>

A Java Comparable interfészét a felhasználó által definiált típusokat rendezésre használjuk. A T annak az objektumnak lesz a típusa, amihez majd hasonlítni akarjuk ezt az interfészt implementáló objektumot. Ezt az interfészt a java.lang-ba fogjuk megtalálni, ezt a feladat is közli. És ha megnézzük egyetlen metódusa lesz ami nem más mint a "compareTo(Object)", ebben fogjuk tehát a rendezést is definiálni.

A feladathoz először is szükségünk lesz a Wiki2Matrix.java-ra amit én a megjelölt oldalról szedtem. Itt kell megadni a következők szerint a bajnokság adatait. Üres=0, zöld=1, srága=2, piros=3. Ezt fordítva és futatva kapunk egy linkmátrixot, amit majd az AlternativTabella.javába kell beraknunk a megfelelő helyre, valamint ott módosítani kell a csapatokat, ugyanis egy 2011-es adat szerepel benne, ami nyilván már nem állja meg a helyét, hisz az mb1 változott azóta sokat, sok csapat került ki és be.

A linkmátrixról nem csatolnék képet, de a fájlaimat belinkelem ha valakit érdekel meg lehet őket tekinteni benne:

[Wiki2Matrix](#)

[Alternativ Tabella](#)

Miután a linkmátrixot megadtam és módosítottam 2019 májusi adat szerint a fájl tartalmát, fordíthatjuk és futathatjuk, nem meglepő módon az eredmény tábla, nem egyezik meg a tabellával.

```
File Edit View Search Terminal Help
0.10012662589051297
0.06878211118661758
0.10386252835000312
0.04913107160326017
0.06645491025329221
0.08637923881305701
0.07237774283587364
0.0955772509354724
0.06954335859344934
0.10437182655130854
0.10182010008631162

Csapatok rendezve:

| -
| Ferencvaros
| 74
| MTK
0.1043
Videoton
61
Honved
0.1038
-
Debrecen
51
Haladas
0.1018
-
Honved
49
Videoton
0.1001
-
Ujpest
48
Kisvarda
0.0955
-
Mezokovesd
44
Puskas Akademia
0.0863
-
Puskas Akademia
40
Ferencvaros
```

17.1. ábra. Alternatív rendezett tabella

De miért is van ez vagy hogyan is van ez?

Alapvetőleg a csapatokat a következő szabályok szerint tudják tabellán ábrázolni:

- 1) pontszám;
- 2) több győzelem;
- 3) jobb gólkülönbség;
- 4) több szerzett gól;
- 5) az egymás ellen játszott mérkőzések pontkülönbsége;
- 6) az egymás ellen játszott mérkőzések gólkülönbsége;
- 7) az egymás ellen játszott mérkőzéseken az idegenben szerzett több gól;
- 8) a fair play értékelésében elért jobb helyezés;

## 9) sorsolás

Ezzel ellentében a mi rendezésünk figyelmebe veszi azt is ki ellen játszott a csapat. Így például márás jobb helyezést ért el összesítésben az MTK, ugyanis az erősebb csapatok elleni győzelem itt többet ért.

A JDK forrásainban is találhatunk ezzel a témával kapcsolatos dolgot, ami a megoldásunk helyeségét alá támasztja, a következő sorok azok:

```
"Lists (and arrays) of objects that implement this interface can be sorted ←
 automatically by {@link Collection#sort(List) Collections.sort} (and { ←
 @link Arrays#sort(Object[]) Arrays.sort}). Objects that implement this ←
 interface can be used as keys in a {@linkplain SortedMap sorted map} or ←
 as elements in a {@linkplain SortedSet sorted set}, without the need to ←
 specify a {@linkplain Comparator comparator}"
```

Ha magyarárunk kellene fordítanunk valami illesmit takarna:

Ezt a felületet megvalósító objektumok listái (és tömbjei) automatikusan ← rendezhetők a {@link Collection#sort(List) Collections.sort} (és a { ← @link Arrays#sort(Object[]) Arrays.sort}) alapján. Ezt a felületet ← megvalósító objektumok kulcsokként használhatók a {@linkplain SortedMap ← sorted map} vagy elemekként a {@linkplain SortedSet sorted set}, anélkül ← , hogy meg kellene adni a {@linkplain Comparator comparator}-t.

## 17.4. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témát (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

**Ezt a feladatot az előző félévben készítettem el, csak átemeltem ide, valamint újra kipróbáltam de ezt a 3 feladaton kívüli +1 feladatnak számom, így nem annyira részletes lehet mint ami már a prog2-es elvárás lenne.**

Gimp Scheme Script-fu: króm effekt

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

A GIMP script mappájából tudjuk csak a kódunkat működésre bírni, valamint a programon belül is megadjuk elérési helyét ezután már is képesek leszünk kevünk szerint alakítani a színeket a szövegen. Ha futtatjuk a programot akkor kapunk egy default beállítást mindenre de persze ezeken kedvünk szerint módosíthatunk, nem köt minket semmihez. Viszont a feladat köt minket még pedig a króm effektet kell elérnünk.

Na de térdünk a feladatra, első lépésként egy fekete színű hátteret hozunk létre amin a szövegünk fehér színű lesz. Ezt így érjük el:

```
(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)))
```

Ha ezzel megvagyunk második lépésként elmosuk a szövegünket, még pedig az úgynevezett Gaussian eljárással.

```
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

3. lépésként a szövegnek az éleit görbítjük le, majd 4. lépésként pedig ezt mossuk el az előbbi módszerrel.

```
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

A következő lépésekben lehetőségünk lesz az elején létrehozott fekete háttérrel kitörölni a kép invertálása mellett, ami átlátszó lesz tehát maga a kép fog látszani ezáltal, a szöveggel pedig kedvünk szerint foglalkozhatunk, majd az utolsó (a forráskódban 9.) lépés alkalmával lesz lehetőségünk megadni a gradient effektet, amivel sikeresen késznek tudhatjuk a feladatot.

```
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))
```

## 18. fejezet

# Helló, Szünet!

### 18.1. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/-justine/rcemu/src/carlexer.ll>

Scanf-fel már a foglalkoztunk ezen a könyven belül, de még is hogy néz ki ez vagy mi is ez?

Az std::scanf adatok beolvasására alkalmas, egy null végződésű string bufferből. Az std::scanf függvény:

```
int scanf(const char* buffer, const char* format, ...);
```

Akkor tér vissza a függvény, ha a formátumsztring által meghatározott számú adatot beolvast, vagy ha hibát észlel, azaz az adatbevitel nem felel meg a formátumsztring előírásainak. Visszatérési értéke egy int lesz, ami azt adja meg hogy hány argumentumot sikerült beolvasni, amennyiben sikeres a beolvasás. Input hiba esetén EOF-al tér vissza.

A buffer egy olyan mutató, ami egy null végződésű karakter stringre mutat, ahonnan mi olvasunk, míg a format szintén mutató egy null végződésű karakter stringre, de ő azt határozza meg hogyan olvassuk a bemenetet.

A Lexer három fő részből áll, ezek a deklarációk, a szabályok és a kiegészítő funkciók. Ezek tudatában nézzük meg hogy a Lex program hogyan hasznája az std::scanf-et.(A kódcsipetek a feladat szövegében megjelölt `carlexer.ll`-ből vannak kiemelve!)

```
{POS} {WS} {INT} {WS} {INT} {WS} {INT} {
 std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
 m_cmd = 10001;
}
```

Ahol a "POS" és a "WS" tűnik idegenebbnek az "INT" mellett, ők a következőképpen vannak definiálva, szintén a `carlexer.ll`-ben:

```
POS "<pos"
WS [\t] *
```

Tehát a POS a pos-szal kezdődő szavakat jelöli, míg a WS a tab karaktert jelenti "[\t]" és mivel ott a csillag így akárhányszor szerepelhet, akár 0-szor is de akár 20-szor is. Így tehát a {POS} {WS} {INT} {WS} {INT} {WS} {INT} kifejezés a következő képpen néz ki:

```
<pos [\t]* {INT} [\t]* {INT} [\t]* {INT}
```

Azaz olyan szöveget fog elfogadni ami pos-szal kezdődik 3 int van benne és az intek között bármennyi \t szerepel (0 is akár).

```
{
 std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
 m_cmd = 10001;
}
```

Ebben a részben pedig az történik hogy az std::scanf függvényből az yytext tartalmazza azt a szövegcsepetet amire a elxer egyezést talált, beolvassuk az értéket az m\_id from és a to-ba. Végül pedig az m\_cmd-t 1001-re állítjuk. Ezek a változók header fájlban vannak definiálva.

És igazából a következő 3 is hasonlóképpen működik, azaz keressük a megadott szövegrészt majd ha megegyezést talál, akkor a megadott változókba beolvassuk az adatokat.

```
{CAR}{WS}{INT} {
 std::sscanf(yytext, "<car %d", &m_id);
 m_cmd = 1001;
}
{STAT}{WS}{INT} {
 std::sscanf(yytext, "<stat %d", &m_id);
 m_cmd = 1003;
}
{GANGSTERS}{WS}{INT} {
 std::sscanf(yytext, "<gangsters %d", &m_id);
 m_cmd = 1002;
}
```

A végén lévő részek és hasonlóak ezekhez, viszont a következő egy érdekesebb, bonyulultabb rész, így még én azt emelném ki részletezésre.

```
{ROUTE}{WS}{INT}{WS}{INT}({WS}{INT})* {
 int size{0};
 int ss{0};
 int sn{0};

 std::sscanf(yytext, "<route %d %d%n", &size, &m_id, &sn);
 ss += sn;
 for(int i{0}; i<size; ++i)
 {
 unsigned int u{0u};
 std::sscanf(yytext+ss, "%u%n", &u, &sn);
 route.push_back(u);
 ss += sn;
 }
 m_cmd = 101;
}
```

Terjedelmeben is ez a leghosszabb példa, és ránézésre is különbözik a felépítése, de nézzük is meg hogy ez mi is, vagy mire képes.

```
{ROUTE} {WS} {INT} {WS} {INT} ({WS} {INT})* {
 int size{0};
 int ss{0};
 int sn{0};
```

Először is deklarálunk 3 változót, ezek közül a size lesz az ahol tároljuk majd a megadott kifejezés által illesztett teljes szöveg hosszát. Az ss az eddig feldolgozott szövegrész méretét fogja tartalmazni, míg az sn-ben pedig az aktuális feldolgozott szöveg mérete lesz karakterekben mérve. Ezekre azért van szükség mert van egy ({WS}{INT})\* tagunk, ami a {WS} és {INT}-ből alkotható összes kombinációt képes létrehozni, ez lehet akár semmi de lehet akár egymást váltva 10 {WS} és 10 {INT}, vagy csak valamelyikból 1, szóval tényleg minden fajta kombinációja ezeknek.

```
std::sscanf(yytext, "<route %d %d%n", &size, &m_id, &sn);
ss += sn;
```

Az első scanfbe beolvassuk a szöveg méretét a size változónkba, a következő integert pedig az m\_id-be, sn-be mint azt fentebb leírtam tároljuk a beolvasott karakterek számát. Miután lement az első scanf növeljük ss-be a feldolgozott karakterek számával az értéket.

```
for(int i{0}; i<size; ++i)
{
 unsigned int u{0u};
 std::sscanf(yytext+ss, "%u%n", &u, &sn);
 route.push_back(u);
 ss += sn;
}
```

Ezután jön a for ciklus, ami a szöveg méretéig megy, tehát végig az egész szövegen. Így a következő scanf-ben yytext-hez hozzáadjuk az ss értékét, így onnan folytatjuk a beolvasást ahol abba hagytuk. Beolvassunk egy unsigned integrert u-ra, sn-ben szintén tároljuk a beolvasott karaktereket, u-t pedig betessük a route nevű vektorunkba, ss-t pedig sn-el növeljük, így a következő lefutásnál megint helyesen tudja folytatni onnan ahol abba hagytuk.

```
m_cmd = 101;
```

És legvégül pedig az m\_cmd váltózót még 101-re állítjuk.

Így a route vektorunk tartalmazni fogja az integrereket a szöveg végéig.

## 18.2. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>

A feladatban a webkamerát kezelő kódokat kell bemutatnunk! Igazából arcokat fog leolvasni nekünk.

Először is be kell szerezni a projektet amit egyszerűen megtudunk a git clone segítségével a következőképpen:

```
git clone https://github.com/nbatfai/SamuCam.git
```

Ezután a mappába lépve a következőt kell beírnunk:

```
wget https://github.com/Itseez/opencv/raw/master/data/lbpcascades/ ←
lbpcascade_frontalface.xml
```

Ezzel egy .xml fájlunk lesz, ezután mehet is a qmake.

```
~/Qt/5.12.2/gcc_64/bin/qmake SamuLife.pro
```

Nyilván onnan indítjuk ahova telepítettük a Qt-t és nyilván azzal a verzióval amivel rendelkezünk!

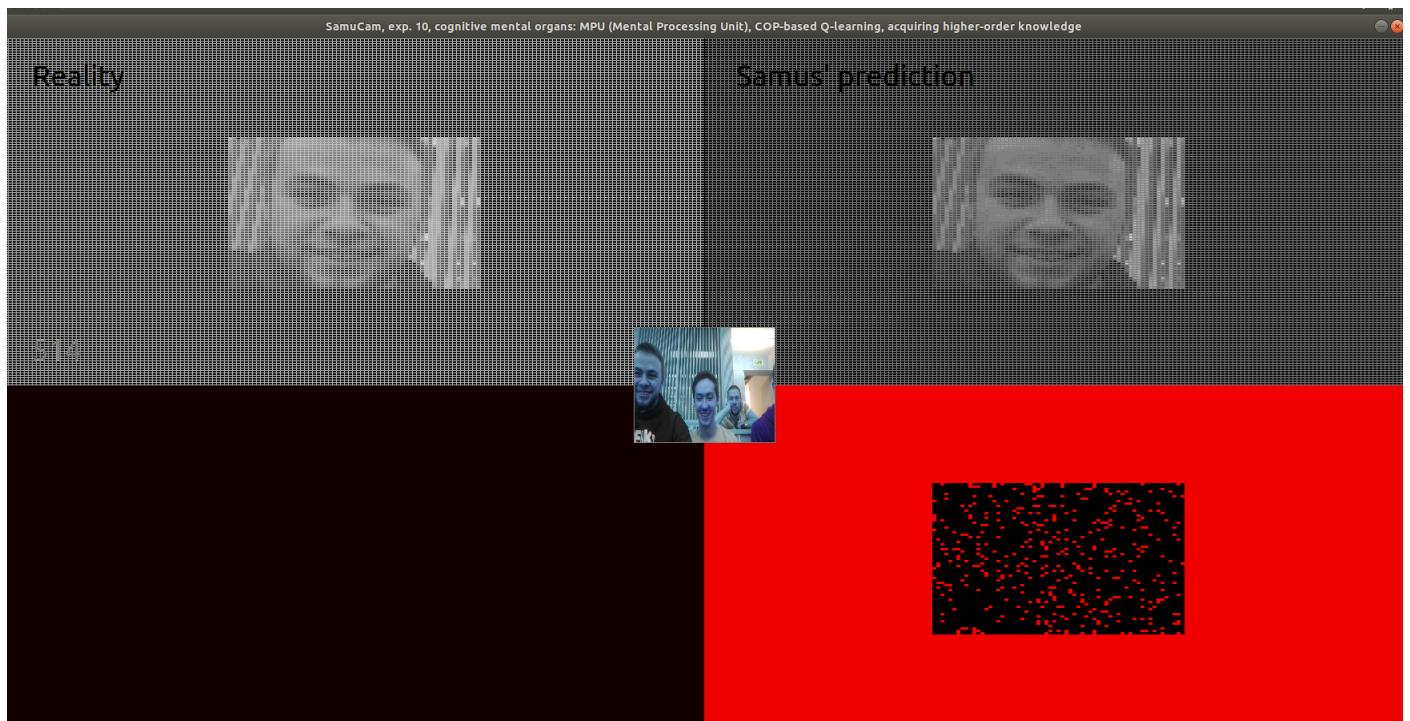
Ezután make, és a ./SamuCam parancsal már fut is a programunk.

Legalábbis ha kamerával rendelkezünk és azt észleli a program, én a laptopom kameráját használtam és ehez egy kis módosítást vittem a kódba, még pedig a videotreamet írtam át 0-ra így egyből alapértelmezetten azt használja, és nem kell bajlódni az ip-vel. A követekző helyen tettek ezt meg:

```
void SamuCam::openVideoStream()
{
 videoCapture.open (0); //itt módosítottam

 videoCapture.set (CV_CAP_PROP_FRAME_WIDTH, width);
 videoCapture.set (CV_CAP_PROP_FRAME_HEIGHT, height);
 videoCapture.set (CV_CAP_PROP_FPS, 10);
}
```

Így már kisebb nagyobb sikereket tudhatunk magunknak, az egyik legjobb felismerését be is rakom ide, tehát így néz ki futás közben:



18.1. ábra. SamuCam futása

A SamuCam osztályunk a Qthread osztályból van származtatva, ezt a SamuCam.h fájlból tudjuk megállapítani.

```
class SamuCam : public QThread
{
 Q_OBJECT

public:
 SamuCam (std::string videoStream, int width, int height);
 ~SamuCam();

 void openVideoStream();
 void run();

private:
 std::string videoStream;
 cv::VideoCapture videoCapture;
 int width;
 int height;
 int fps;

signals:
 void faceChanged (QImage *);
 void webcamChanged (QImage *);
};

};
```

## 18.3. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esport-talent-search>

A BrainB egy esport tehetség felmérő kis program, ha lehet illet mondani, például MOBA játékoknál ismert "karakterelvesztés" gyakoriságát mérifel vagy csökkenti a BrainB használata.

Mi is az a karakterelvesztés?

A karakterelvesztés főként MOBA(Multiplayer Online Battle Arena) játékoknál jellemző, mint például a LoL(League of Legends) esetében. Maga a fogalom elég pontos, ugyanis a szó szoros értelmében elveszítjük a karakterünket a sok akció közepette, például skill-ek általi effektek más player karakterei vagy egyéb "kitakarás" mellett. Maga a BrainB ezzel foglalkozik leginkább, ugyanis a feladat az lesz a programban hogy egy úgy nevezett Samu Entropy-n kell tartanunk az egeret, az eredményünk annál jobb lesz minél jobban ment ez a program 10 perces futása alatt. A 10 perc elteltével fogjuk látni eredményünk.

Ahhoz, hogy a feladatot megoldjuk szükségünk lesz a libqt4-dev, opencv-data és libopencv-dev csopamgok telepítésére. Valamint ugye a Qt-ra. De mivel első fejezetben a BraniB már szerepelt ezzel rendelkezünk, de ha még sem akkor a <https://www.qt.io> oldalról beszerezhető.

Viszont most kicsit részletesebben kell vizsgálnunk mint akkor, először is tisztázni kellene mi az a slot és mi az a signál.

A slotok a függvényekre hasonlítanak, meg elhet őket hívni, definícióval is rendelkeznek ugyanúgy, és paramétereik is lehetnek, egyedül visszatérési értékkel nem rendelkeznek. Míg a signal-ok olyan függvények amik nem rendelkeznek se definícióval se deklarációval. Paraméterekkel ők is rendelkezhetnek de visszatérési értékkel nem hasonlóan a slotokhoz. Viszont meghívni őket az "emit" kulcsszó használatával kell, nem lehet csak egyszerűen mint a függvényeket.

Slot-signál párokat tudunk létrehozni, bár ehhez teljesülni kell annak a feltételnek hogy a paramétereik típusai páronként megegyezzenek, vagy a slot ne rendelkezzen paraméterrel és a signálét át lehet adni neki. Ilyen párosokra kell néznünk példát a BrainB programban, nézzük is meg, a következő kód rész a [BrainBWin.cpp](#)-ből lett kiemelve!

```
BrainBWin::BrainBWin (int w, int h, QWidget *parent) : QMainWindow (←
 parent)
{
 // setWindowTitle(appName + " " + appVersion);
 // setFixedSize(QSize(w, h));

 statDir = appName + " " + appVersion + " - " + QDate::currentDate() ←
 .toString() + QString::number (QDateTime::←
 currentMSecsSinceEpoch());

 brainBThread = new BrainBThread (w, h - yshift);
 brainBThread->start();

 connect (brainBThread, SIGNAL (heroesChanged (QImage, int, int) ←
),
 this, SLOT (updateHeroes (QImage, int, int)));
}
```

```
 connect (brainBThread, SIGNAL (endAndStats (int)),
 this, SLOT (endAndStats (int)));
}
```

Mint az a fenti példában látjuk a connect segítségével fog létrejönni a slot-signal párnak. A connect szintaktikája a következő: az első paraméter lesz az az objektum ami a signalt küldi, a második pedig maga a signal amit kezelni akarunk, a harmadik paraméter pedig egy mutató lesz ami a szignált kezelő objektumra mutat, a negyedik paraméter maga a slot lesz.

```
connect (brainBThread, SIGNAL (heroesChanged (QImage, int, int)) ,
 this, SLOT (updateHeroes (QImage, int, int)));
```

Így a fenti kód részlet azt fogja eredményezni, hogy a brainBThread objektum heroesChanged singálja ha aktiválódik akkor ez az objektum a BrainBWin az updateHeroes slottal fogja lekezelni. Azaz a kiszámolt értékeket a szignál viszi magával amiket az updateHeroes átvesz majd és azoknak segítségével végzi el a dolgát, azaz a megjelenítést/ablakok frissítését.

```
connect (brainBThread, SIGNAL (endAndStats (int)),
 this, SLOT (endAndStats (int)));
```

Míg a második connectben ugyancsak a brainBThread objektum, de ez esetben endAndStats signálját kezeljük, ami akkor aktiválódik ha a program futási ideje befejeződik, ilyenkor ez az objektum (BrainBWin) fogja kezelni az endAndStats slottal, ami bezárja majd nekünk az ablakot és leállítsa a program futását.

```
void BrainBThread::run ()
{
 while (time < endTime) {

 QThread::msleep (delay);

 if (!paused) {

 ++time;

 devel();

 }

 draw();

 }

 emit endAndStats (endTime);
}
```

A fenti kódrészlet pedig már a [BrainBThread.cpp](#)-ből van, a második "connect" által létrejött slot-signalat használja, amit már a fent említett módon az "emit" kulcsszóval hív meg.

## 19. fejezet

# Helló, Lauda!

### 19.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! <https://www.tankonyvtar.hu-hu/tartalom/tkt/javat-tanitok-javat/ch01.html#id527287>

A feladathoz a kódot készen kaptuk ami a következő volt:

```
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");

 }
 }
}
```

De mit is csinál ez, hogyan működik?

Egy for ciklust látunk ami 1024-ig megy 0-tól, ezeken a portszámokon szereplő TCP kapukat fogjuk vizsgálni, pontosabban rajtuk keresztül kapcsolatot létrehozni. Ha a figyeli üzenetet kapjuk vissza az annyit jelent, hogy azon a porton "nyitott" a szerverrel való kapcsolat, ha pedig ez nem igaz lök egy exceptiont amiből mi annyit látunk hogy a "nem figyeli" üzenetet kapjuk vissza.

De beszéljünk egy kicsit a kivételkezelésről is. Akkor keletkezik kivétel ha a program végre hajtódásakor keletkező esemény megszakítja az utasítások végrehajtását. Tehát a try blokkban szereplő kódunkban valami hibát észlel, "megakad" ekkor ugrik át a catch blokkra ami kezeli a kivételeket, mi esetünkben az Exception a paramétere, amely egy olyan kivételkezelési osztály ami magába foglalja az összes többi kivételkezelési osztályt. A programunkban ha az adott porton nem figyeli egyik folyamat sem akkor kivétel kerül eldobásra. Amit a catch blokk észlel és egy "figyelmeztető" kiírással jelzi is azt számunkra. A JDK-ban a kivételkezelésről találhatunk egy ilyet amit érdemes áttanulmányozni ugyan is mostani esetüknél is előjön egy része:

```
public Socket(String host, int port)
 throws UnknownHostException, IOException
{
 this(host != null ? new InetSocketAddress(host, port) :
 new InetSocketAddress(InetAddress.getByName(null), port),
 (SocketAddress) null, true);
}
```

Kiemelendő viszont ennél a feladatnál hogy ne engedjük rá mindenre, hisz fenyegetésnek észlelhető, ezért csak saját gépre/szerverre, vagy ismert gépre/szerverre akivel megbeszéltük!

Ha fordítjuk és futtatjuk a programunkat akkor a következőt fogjuk látni tehát:

```
985 nem figyeli
986 nem figyeli
987 nem figyeli
988 nem figyeli
989 nem figyeli
990 nem figyeli
991 nem figyeli
992 nem figyeli
993 nem figyeli
994 nem figyeli
995 nem figyeli
996 nem figyeli
997 nem figyeli
998 nem figyeli
999 nem figyeli
1000 nem figyeli
1001 nem figyeli
1002 nem figyeli
1003 nem figyeli
1004 nem figyeli
1005 nem figyeli
1006 nem figyeli
1007 nem figyeli
1008 nem figyeli
1009 nem figyeli
1010 nem figyeli
1011 nem figyeli
1012 nem figyeli
1013 nem figyeli
1014 nem figyeli
1015 nem figyeli
1016 nem figyeli
1017 nem figyeli
1018 nem figyeli
1019 nem figyeli
1020 nem figyeli
1021 nem figyeli
1022 nem figyeli
1023 nem figyeli
csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Lauda$
```

19.1. ábra. KapuSzkennner futása

## 19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Az AOP jelentése, Aspect-oriented programming, azaz askpektus orientált programozás, ezt a nevet azért kapta, mert segítségével különböző aspektusokból tudjuk megfigyelni, hogy hogyan is viselkedik a kód, anélkül hogy abba belenyúlnánk. Az OO-hoz képest egy magasabb szintű absztrakciót vezet be. Azért hasznos, mivel nincs szükség a kódunkba belenyúlkálni és ott átírni/beleírni dolgokat, hanem egy külön fájlba megírjuk, majd egy complier segítségével összefüzzük a fő állományunkkal. Futtatás után pedig minden két fájlnak külön-külön olvasható lesz a logja.

A feladat a LZWBinFa Java átiratába kell belefűzni egy AspectJ-s scriptet. Ami a kiir függvényen fog változtatni, még pedig annyiban hogy az inorder (eredeti) kiíratás mellett preorderben is kiírásra kerül a kimenet.

Ennek a megvalósításához, létrehozzunk egy pointcut-ot ami itt a feladat miatt a kiir() függvény lesz, és egyben az Aspect számára ezek lesznek a joinpoint-ok. Így az fog történi hogy a call() függvény hívással meghívjuk a kiir függvényt ami már az AspectJ-s kód szerint is le fog zajlani, míg a kiir függvényig minden ugyanúgy lemegey az LZWBinFa-ba, majd ezek után az after() függvény segítségével minden az eredeti lefutást mint pedig az AspectJ-s "módosítást", azaz a preorderben kiírást is megejelenítjük. Így tehát eredményképpen az eredeti inorder kiírás mellé preorderben is látható lesz a kimenet, anélkül hogy az eredeti forrásban bármit is módosítottunk volna.

```
package binfa;

import java.io.FileNotFoundException;
import java.io.IOException;

public aspect order {

 int melyseg = 0;

 public pointcut travel(LZWBInFa.Csomopont elem, java.io.PrintWriter os)
 : call(public void LZWBInFa.kiir(LZWBInFa.Csomopont, java.io.←
 PrintWriter)) && args(elem,os);

 after(LZWBInFa.Csomopont elem, java.io.PrintWriter os) throws IOException ←
 : travel(elem, os)
 {

 java.io.PrintWriter kiPre = new java.io.PrintWriter(
 new java.io.BufferedWriter(new java.io.FileWriter("preorder.txt")))
 ;

 melyseg = 0;
 preorder(elem, kiPre);

 kiPre.close();
 }

 public void preorder(LZWBInFa.Csomopont elem, java.io.PrintWriter p) {
 if (elem != null) {
 ++melyseg;
 for (int i = 0; i < melyseg; ++i) {
 p.print("----");
 }
 p.print(elem.getBetu());
 p.print("(");
 p.print(melyseg - 1);
 p.println(")");
 preorder(elem.egyesGyermekek(), p);
 preorder(elem.nullasGyermekek(), p);

 --melyseg;
 }
 }
}
```

```
 }

}
```

A fenti forrás maga az \*.aj fájlunk, mint azt fentebb említettem az elején pointcutot létrehozza valamint a call segítségével meghívja a szükséges függvényt, majd meghatározza a kimenetet az after függvény törzsében. Magát az inorderból preorderbe történő átalakítást pedig a preorder függvényünk végzi, ha a szükséges fájlokkal rendelkezünk (LZWBinFa.java, order.aj, text.txt...), azok fordítása után, s futattása utána megfog jelenni a preorder.txt-nk ami tartalmazni fogja a binfa preorder bejárása szerinti kiírást.

### 19.3. Junit teszt

A [https://progpater.blog.hu/2011/03/05/labormeres\\_oththon\\_avagy\\_hogyan\\_dolgozok\\_fel\\_egy\\_pedat](https://progpater.blog.hu/2011/03/05/labormeres_oththon_avagy_hogyan_dolgozok_fel_egy_pedat) poszt kézzel számított mélységet és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban). Először is tisztázzuk, mi is az a Junit teszt? Pontosabban mi a Junit?

A Junit egy keretrendszer, amit egységesztelésre használnak, a Java programozási nyelvnél

De nézzük is meg, hogy a gyakorlatban hogy néz ez ki!

```
@org.junit.Test
```

Már rögtön a 3. sorban ez fogad minket, de mi is ez? Igazából a @-cal kezdődő sorok jelölik a metódusok kezdetét, amit a Junit tesztfuttatójának futtatnia kell. Így ezek segítségével tudjuk a programot bármelyik "élethelyzetében" letesztni.

Tehát a teszteset úgy fog felépülni, hogy egy @ annotációval kezdődik, majd meg kell adni a tesztelendő metódust, ami meghívásra fog kerülni és az eredményt összehasonlíthatjuk az elvárásainkkal, így biztosra menve azzal, hogy úgy működik a metódus, ahogy azt mi szeretnénk.

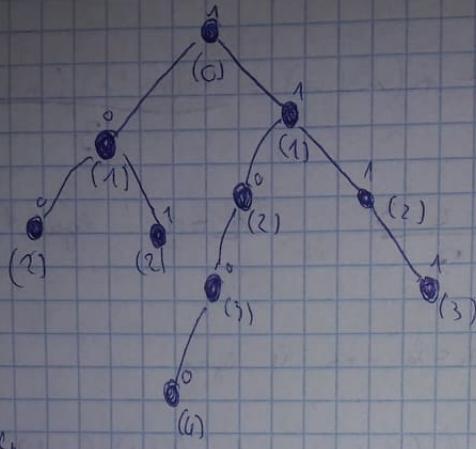
Már léteznek egyéb annotációk és egyéb tesztelési lehetőségek Juniton belül, de a feladathoz ennyi elegendő Nézzük is meg magát a tesztelési részt a forráskódba

```
public void testBitFeldolg() {
 for (char c : "01111001001001000111".toCharArray())
 {
 binfa.egyBitFeldolg(c);
 }
 org.junit.Assert.assertEquals(4, binfa.getMelyseg(), 0.0);
 org.junit.Assert.assertEquals(2.75, binfa.getAtlag(), 0.001);
 org.junit.Assert.assertEquals(0.957427, binfa.getSzoras(), 0.0001);
}
```

Mint láthatjuk a testBitFeldolg függvényünk fogja végezni a teszttünket. Bitenként dolgozzuk fel a megadott tömböt, az egyBitFeldolg függvény segítségével.

Ha ezekkel megvagyunk jön az "összemérés", azaz a következő 3 sor.

Az összehasonlítást az assertEquals függvény fogja képezni, aminek 3 paramétere lesz, az első paraméter az az elváreredmény lesz, a második paraméter a program által kapott érték lesz, míg a 3. paraméter a két érték közötti eltérés megengedett értéke, ami a mi esetünkben nagyon-nagyon alacsony.



Mélyszög: 4

$$\text{Vflang: } \frac{2+2+4+7}{4} = \frac{11}{4} = 2,75$$

$$\text{Sekundás: } \frac{1}{(4-1)} * \left\{ (2-2, 75) * (2-2, 75) + (2-2, 75) * (2-2, 75) + \right. \\ \left. (4-2, 75) * (4-2, 75) + (3-2, 75) * (3-2, 75) \right\} = \\ \frac{2}{3} * (0,5625 + 0,5625 + 1,5625 + 0,0625) = 0,9524$$

19.2. ábra. Elvárt értékek kiszámolása papíron

Az eredmény a várt, a program helyesen működik! A papíron szereplő elvárt számítási eredményeket adja vissza a program is.

## 20. fejezet

# Helló, Calvin!

### 20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, [https://progpater.blog.hu/2016/11/13/hello\\_samu\\_a\\_tensorflow](https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow)-bol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Mi is az az MNIST? Az MNIST igazából egy kézzel írott arab számjegyeket tartalmazó adatbázis mely több ezer képállományt tartalmaz, ezek közül van tanulási és teszt képállomány. A tanulási képekből tanulja meg a gép a számjegyeket, majd ezt tudjuk tesztelni hogy mennyire is tanulta meg helyesen.

Nézzük is meg ezt a tanulási folyamatot a forrás alapján:

```
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.utils import to_categorical, np_utils
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
```

Először is importálnunk kell a szükséges könyvtárakat. Bár ezelőtti lépésként az szerepel hogy minden "beszerezzünk", jómagam is sok minden leszedtem (lehet volt köztük olyan is ami nem is kellett :DD), többek közt szükség lesz a python3-dev-re és a python3-pip-re, a pip segítségével megnézhetjük hogy az importált könyvtárak megvannak-e nekünk, ha nincs akkor el is kezdi szedni nekünk azt. Én Keras-t fogok használni ezt a következő képpen tudjuk letölteni ha már minden más megvan:

```
sudo pip3 install keras
```

A következő "feladatunk" az volt hogy betöltsük az adatbázist, hogy tudjunk vele dolgozni, ezért a forrásban a következő sor felelt:

```
(train_X, train_Y), (test_X, test_Y) = tf.keras.datasets.mnist.load_data()
```

Most már rendelkezésünkre áll az adatbázis, nézzük tovább a forrást..

```
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)
```

Ezek a sorok a vektorok elkészítéséért felelősök, melyeket a reshape függvény segítségével hozunk megfelelő formára. Az az 28 darab 28 db elemet tartalmazó vektorra, ez a 2. és a 3. paraméterre a függvénynek, az első paraméter a "-1" ami azt fogja jelenteni hogy minden egyes tagra értelmezni kell, alapvetően ide az kerülne hány darabot kell konvertálni (például ha az első 10-et akarjuk csak akkor egy 10-est kellene írni ide egész egyszerűen). AZ utolsó azaz a 4. paraméter pedig a kép színcsatornájával köthető össze, mivel mi grayscale képeket használunk így az egyes kerül ide, de ha színes azaz RGB képeket használnánk akkor a 3-as számot javasolt ide írni.

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255
```

A következő kódcsípet, azért felelős hogy a tanulás minnél gyorsabban lemenjen. Igazából egyes pixelek értékeinek módosítása ez.

```
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)
```

One hot kódolásra is szükségünk van ugyanis a kódunk nem működik kategórikus adatokkal. Ezzel a kódolással a számjegyek 0-9-ig 9db nulla és 1db 1-es egítségével lesz leírva, különbség köztük az lesz hogy az 1-esünk minden más helyen fog állni, így azt kell nézni majd hogy hányadik helyen áll az egyes.

```
model = Sequential()

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
```

Ezután felépítjük a szekvenciális modellünket. A rétegek egymásra helyezése rendre az add() függvény segítségével fog zajlani. Paraméterei a következők lesznek: első paraméterként a nuronok számát kell megadni (64), a második paraméter a detektor (3x3), a harmadik paramétere pedig az ún. input\_shape lesz, ami a mi esetünkben 28x28 grayscale-s képek lesznek. Ezután a következő sorban aktiválunk egy

relu-t (Rectified Linear Unit). Majd a következő sorban pedig azt adjuk meg hogy mennyi adat kerüljön feldolgozásra egyszerre, ezt a pool\_size-al tudjuk elérni. Mely egy vertikális és egy horizontális értéket vár paraméterként. Ezután a compile fügvénnyel meg is indítjuk a tanulási folyamatot.

```
model.fit(train_X, train_Y_one_hot, batch_size=64, epochs=1)

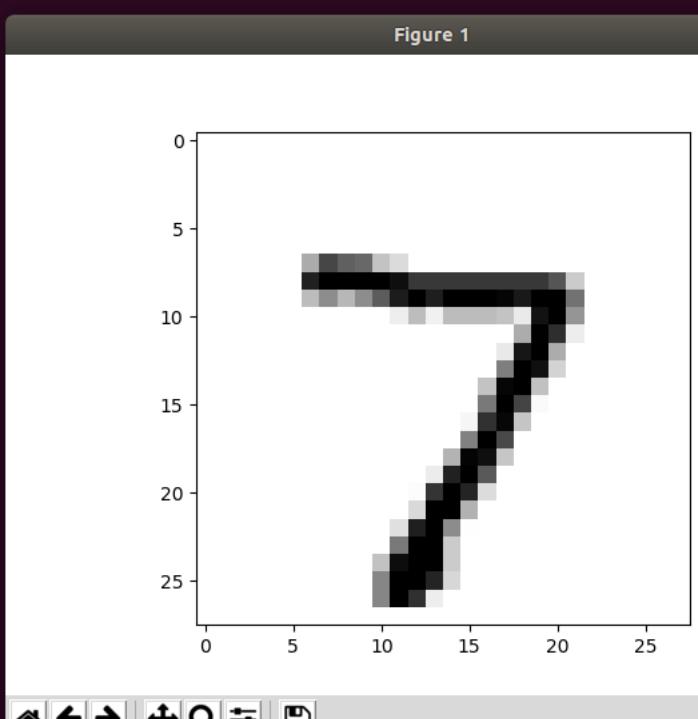
test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)

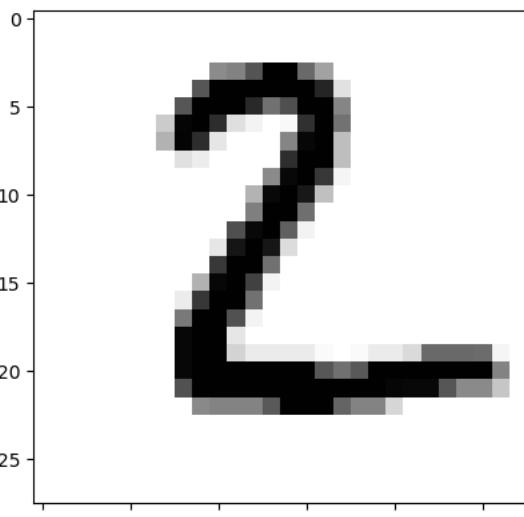
print(np.argmax(np.round(predictions[0])))
plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
print(np.argmax(np.round(predictions[1])))
plt.imshow(test_X[1].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
img = Image.open('szam.png').convert("L")
img = np.resize(img, (28,28,1))
im2arr = np.array(img)
im2arr = im2arr.reshape(1,28,28,1)
print(np.argmax(np.round(model.predict(im2arr))))
plt.imshow(im2arr[0].reshape(28,28),cmap = plt.cm.binary)
plt.show()
```

Ezután beállítjuk a jellemzőket, mint például az epochs, ami azért felelős, hogy hányszor hajtsa végre a tanítási folyamatot, minél nagyobb ez a szám annál pontosabb eredményt kapunk, de a mi esetünkben 1-en hagyva és 90%+ pontossággal dolgozik a program. Valamint a kiíratásokat is itt adjuk meg, amit majd futattásnál látni fogunk. Valamint a feladat alapján egy saját kézzel írott számjegyet és beadunk neki, és megnézzük hogy azt is felismeri-e. Nézzük is hogy boldogult a számjegyekkel.

```
File Edit View Search Terminal Help
(csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Calvin$ python3 MNIST.py
Using TensorFlow backend.
2019-11-24 16:05:33.103341: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2019-11-24 16:05:33.123920: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2712000000 Hz
2019-11-24 16:05:33.124477: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x4a17dc0 executing computations on platform Host. Devices:
2019-11-24 16:05:33.124532: I tensorflow/compiler/xla/service/service.cc:175] StreamExecutor device (0): Host, Default Version
Epoch 1/1
60000/60000 [=====] - 41s 691us/step - loss: 0.1452 - accuracy: 0.9560
10000/10000 [=====] - 2s 206us/step
Test loss 0.05043979652519338
Test accuracy 0.9829000234603882
7
Figure 1
```

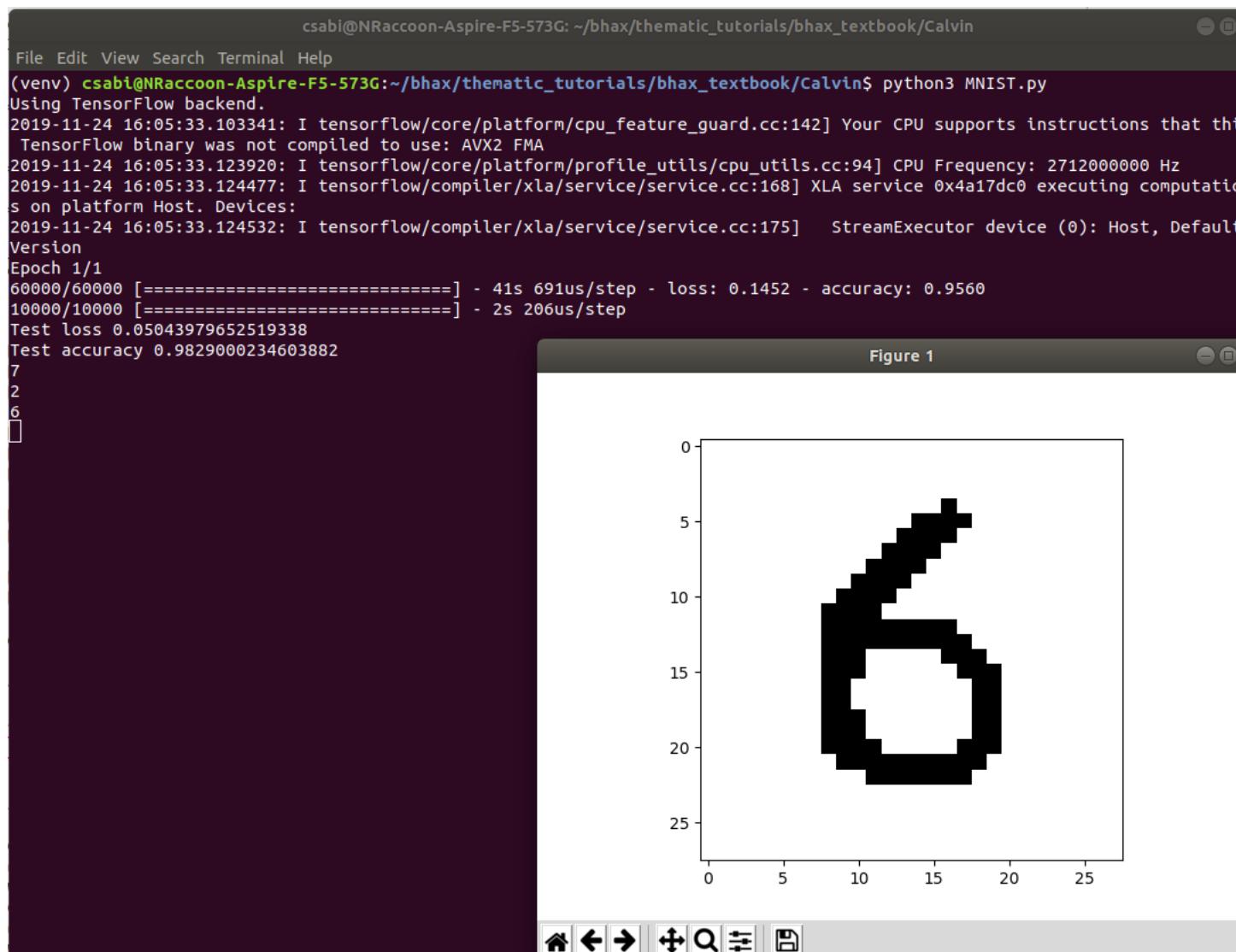


20.1. ábra. Első szám felismerése

```
File Edit View Search Terminal Help
(csabi@NRaccoon-Aspire-F5-573G:~/bhax/thematic_tutorials/bhax_textbook/Calvin$ python3 MNIST.py
Using TensorFlow backend.
2019-11-24 16:05:33.103341: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2019-11-24 16:05:33.123920: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2712000000 Hz
2019-11-24 16:05:33.124477: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x4a17dc0 executing computations on platform Host, Devices:
2019-11-24 16:05:33.124532: I tensorflow/compiler/xla/service/service.cc:175] StreamExecutor device (0): Host, Default Version
Epoch 1/1
60000/60000 [=====] - 41s 691us/step - loss: 0.1452 - accuracy: 0.9560
10000/10000 [=====] - 2s 206us/step
Test loss 0.05043979652519338
Test accuracy 0.9829000234603882
7
2
Figure 1

```

20.2. ábra. Második szám felismerése

Mint azt látjuk gond nélkül felismerte a számjegyeket, most pedig nézzük meg a saját kézzel írott arab számon felismeri-e?



20.3. ábra. Saját szám felismerése

A válasz igen, nem okozott neki gondot felismerni a 6-osom.

## 20.2. CIFAR-10

Az alap feladat megoldása, +saját fotót is ismerjen fel, [https://progpater.blog.hu/2016/12/10/hello\\_samu\\_a\\_cifar-10\\_tfTutorial\\_peldabol](https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tfTutorial_peldabol)

A feladat az előzőhez hasonló lesz, csak nem számokat kell felismernie, hanem tárgyakat, élőlényeket, egyéb objektumokat. Valamint még az is kiemelendő hogy ez esetben színes képekkel fogunk dolgozni és 32x32-es méretben.

Mivel a feladat hasonló így a kód maga se sokban különbözik az előzőtől, így a már leírtakat nem tagalálnám újra ebben a feladatban is, hanem inkább a különbségekre hívjam fel a figyelmet, kezdjük is a legelején.

```
(train_X,train_Y), (test_X,test_Y) = cifar10.load_data()
```

Az első lényeges különbség hogy másik adatbázist töltünk be, ez szerintem nem meglepő, de azért fontos. Nyilván mivel színesek a képek és a méretük is más, így a következő sorok is módosításra kerültek az előző kódhoz képest:

```
train_X = train_X.reshape(-1, 32, 32, 3)
test_X = test_X.reshape(-1, 32, 32, 3)
```

Ugye az első paraméter továbbra is -1 mert minden tagra értendő, a 2. és a 3. 32-re módosul mivel 32x32-es lesz a képek mérete, a 4. pedig 3-ra módosul mivel a képek színesek.

```
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))

model.add(Dense(10))
model.add(Activation('softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=sgd, ←
 metrics=['accuracy'])
img = Image.open('beagle.jpg').convert("L")
img = np.resize(img, (32,32,3))
im2arr = np.array(img)
im2arr = im2arr.reshape(1,32,32,3)
```

Az add függvényt használjuk arra, hogy modellünkhez újabb réteget adjunk hozzá. Például egy konvolúciós réteget adunk hozzá a modellünkhez az első sorban. Amely paraméterül a neuronok számát kapja, majd az úgynévezett detektort pedig második paraméterként.

Így az input\_shape függvény paraméterei is változnak a mostani jellemzőknek megfelelően. Illetve emeltünk a neuron számon is.

```
cifar_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', '←
 frog', 'horse', 'ship', 'truck']
print(cifar_classes[np.argmax(np.round(predictions[0]))])
```

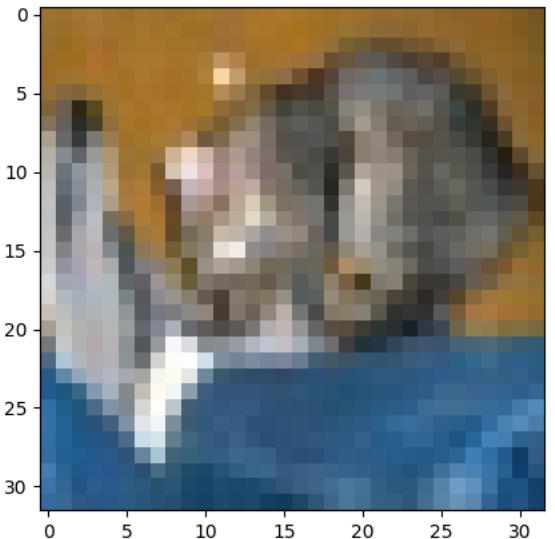
Az egyik legérdekesebb különbség pedig egyértelműen a class tömb, amit kézzel adtunk meg, ez azokat az objektum neveket tartalmazza amiről találhatunk képet az adatbázisban, ugyanis nyilvánvaló hogy nem fog felismerni mondjuk egy mosómedvét ha arról nem volt kép a tanítás során, valószínűleg macskának nézné. Mivel a feladat itt is kér saját képet, így ezeket figyelembe véve kell majd képet választani is!

Nézzük is hogyan boldogult velük a programunk:

```
csabi@NRaccoon-Aspire-F5-573G: ~/bhax/thematic_tutorials/bhax_textbook/Calvin
File Edit View Search Terminal Help
Epoch 7/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.1806 - accuracy: 0.9381
Epoch 8/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.1090 - accuracy: 0.9633
Epoch 9/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0753 - accuracy: 0.9749
Epoch 10/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0549 - accuracy: 0.9826
Epoch 11/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0414 - accuracy: 0.9867
Epoch 12/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0357 - accuracy: 0.9884
Epoch 13/25
50000/50000 [=====] - 157s 3ms/step - loss: 0.0234 - accuracy: 0.9931
Epoch 14/25
50000/50000 [=====] - 157s 3ms/step - loss: 0.0202 - accuracy: 0.9938
Epoch 15/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0145 - accuracy: 0.9960
Epoch 16/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0198 - accuracy: 0.9936
Epoch 17/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0093 - accuracy: 0.9974
Epoch 18/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0041 - accuracy: 0.9989
Epoch 19/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0011 - accuracy: 0.9998
Epoch 20/25
50000/50000 [=====] - 158s 3ms/step - loss: 3.4042e-04 - accuracy: 1.0000
Epoch 21/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.6994e-04 - accuracy: 1.0000
Epoch 22/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.3524e-04 - accuracy: 1.0000
Epoch 23/25
50000/50000 [=====] - 159s 3ms/step - loss: 1.1491e-04 - accuracy: 1.0000
Epoch 24/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.0091e-04 - accuracy: 1.0000
Epoch 25/25
50000/50000 [=====] - 158s 3ms/step - loss: 9.0517e-05 - accuracy: 1.0000
10000/10000 [=====] - 9s 884us/step
Test loss 2.568876153755188
Test accuracy 0.7102000117301941
cat

```

Figure 1



20.4. ábra. Cifar 1

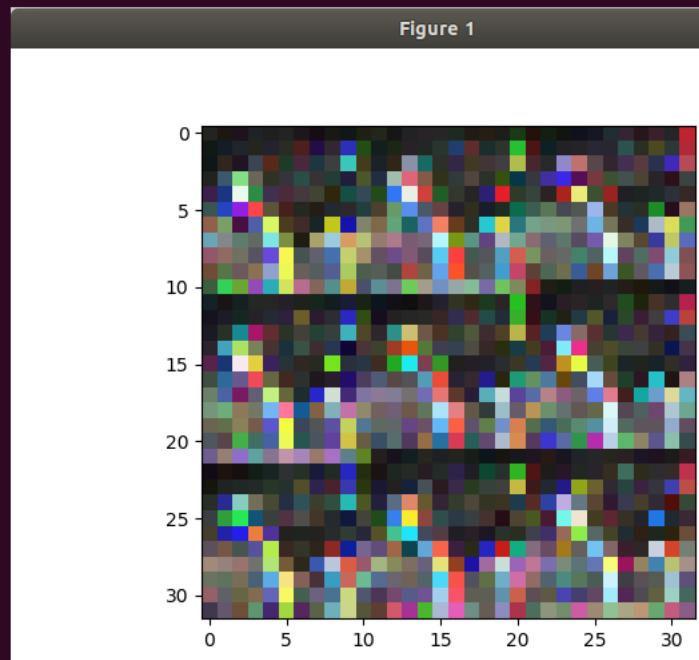
```
File Edit View Search Terminal Help
50000/50000 [=====] - 158s 3ms/step - loss: 0.1806 - accuracy: 0.9381
Epoch 8/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.1090 - accuracy: 0.9633
Epoch 9/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0753 - accuracy: 0.9749
Epoch 10/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0549 - accuracy: 0.9826
Epoch 11/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0414 - accuracy: 0.9867
Epoch 12/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0357 - accuracy: 0.9884
Epoch 13/25
50000/50000 [=====] - 157s 3ms/step - loss: 0.0234 - accuracy: 0.9931
Epoch 14/25
50000/50000 [=====] - 157s 3ms/step - loss: 0.0202 - accuracy: 0.9938
Epoch 15/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0145 - accuracy: 0.9960
Epoch 16/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0198 - accuracy: 0.9936
Epoch 17/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0093 - accuracy: 0.9974
Epoch 18/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0041 - accuracy: 0.9989
Epoch 19/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0011 - accuracy: 0.9998
Epoch 20/25
50000/50000 [=====] - 158s 3ms/step - loss: 3.4042e-04 - accuracy: 1.0000
Epoch 21/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.6994e-04 - accuracy: 1.0000
Epoch 22/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.3524e-04 - accuracy: 1.0000
Epoch 23/25
50000/50000 [=====] - 159s 3ms/step - loss: 1.1491e-04 - accuracy: 1.0000
Epoch 24/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.0091e-04 - accuracy: 1.0000
Epoch 25/25
50000/50000 [=====] - 158s 3ms/step - loss: 9.0517e-05 - accuracy: 1.0000
10000/10000 [=====] - 9s 884us/step
Test loss 2.568876153755188
Test accuracy 0.7102000117301941
cat
deer
□
```

Figure 1

20.5. ábra. Cifar 2

```
File Edit View Search Terminal Help
Epoch 8/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.1090 - accuracy: 0.9633
Epoch 9/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0753 - accuracy: 0.9749
Epoch 10/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0549 - accuracy: 0.9826
Epoch 11/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0414 - accuracy: 0.9867
Epoch 12/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0357 - accuracy: 0.9884
Epoch 13/25
50000/50000 [=====] - 157s 3ms/step - loss: 0.0234 - accuracy: 0.9931
Epoch 14/25
50000/50000 [=====] - 157s 3ms/step - loss: 0.0202 - accuracy: 0.9938
Epoch 15/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0145 - accuracy: 0.9960
Epoch 16/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0198 - accuracy: 0.9936
Epoch 17/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0093 - accuracy: 0.9974
Epoch 18/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0041 - accuracy: 0.9989
Epoch 19/25
50000/50000 [=====] - 158s 3ms/step - loss: 0.0011 - accuracy: 0.9998
Epoch 20/25
50000/50000 [=====] - 158s 3ms/step - loss: 3.4042e-04 - accuracy: 1.0000
Epoch 21/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.6994e-04 - accuracy: 1.0000
Epoch 22/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.3524e-04 - accuracy: 1.0000
Epoch 23/25
50000/50000 [=====] - 159s 3ms/step - loss: 1.1491e-04 - accuracy: 1.0000
Epoch 24/25
50000/50000 [=====] - 158s 3ms/step - loss: 1.0091e-04 - accuracy: 1.0000
Epoch 25/25
50000/50000 [=====] - 158s 3ms/step - loss: 9.0517e-05 - accuracy: 1.0000
10000/10000 [=====] - 9s 884us/step
Test loss 2.568876153755188
Test accuracy 0.7102000117301941
cat
deer
truck 9

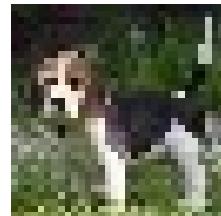
```



20.6. ábra. Cifar 3

A tanulást 25x végeztettem el vele, ugyanis egy tanulásból az első két képet határozottan airplane-nek ítélt

meg míg a saját képem "truck"-nak, ezután 10/15x-ös epochs-sal is kipróbáltam, ott a fenti eredményeket adta vissza, a fentiek viszont 25x-ösek. Tehát nem épp a legjobb, mert az éppen elhetséges hogy az első egy macska, hisz azt én magam se látom ki mi is akar lenni, viszont valószínűséggel a második kép egy madarat ábrázol, hiába pixelekből hasonlít egy szarvasra, de a saját képem bizonyítja hogy közel sem tökéletes ez alapján az adatbázis esetén, hisz a beagle képem egyértelműen kutyát ábrázol, de ezt mindig truck-nak adta vissza. Eredeti kép így nézet ki, ezt adtam be neki (2,5-es nagyításban):

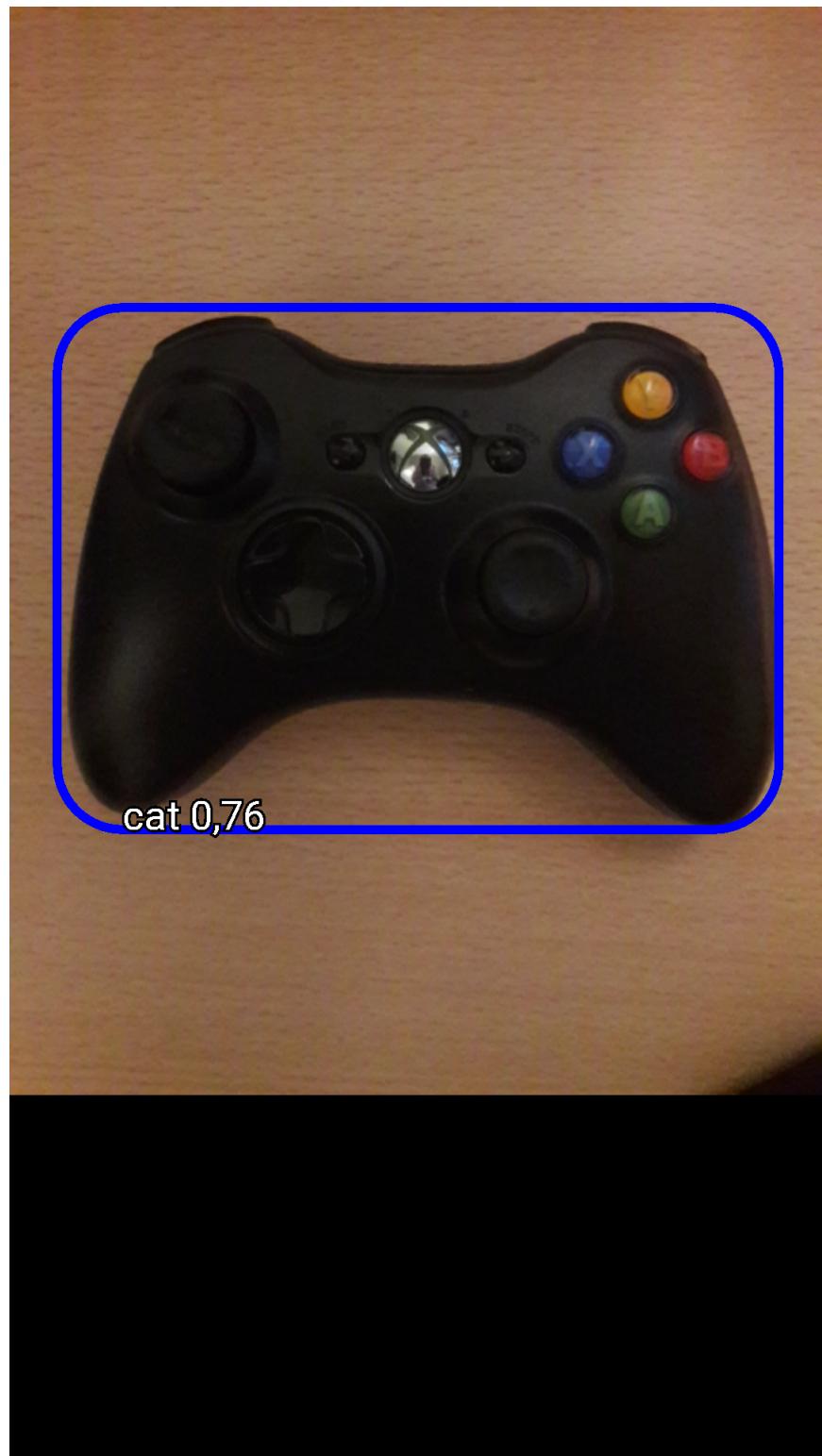


20.7. ábra. Beagle

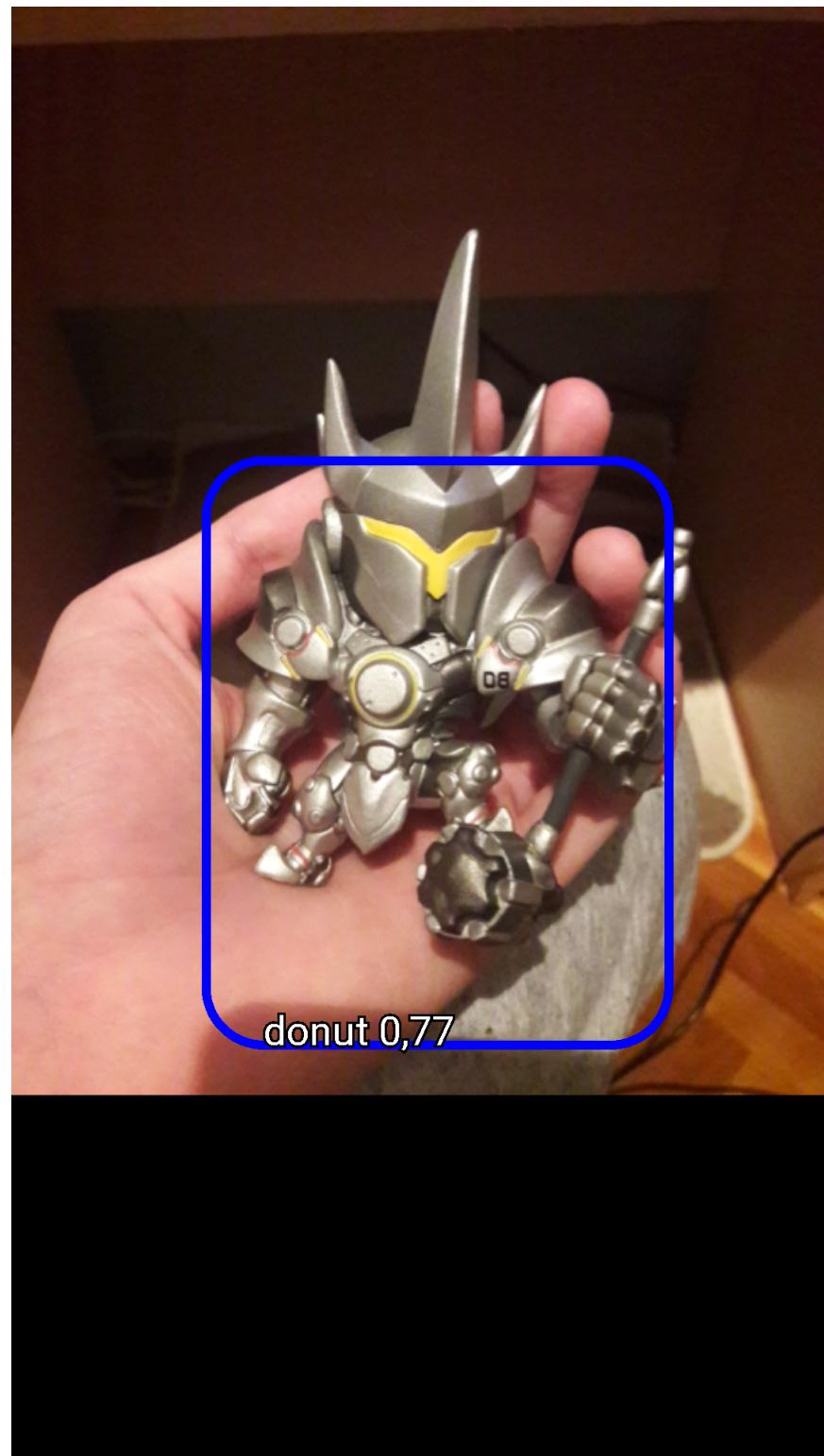
### 20.3. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Itt minden össze annyi volt a feladatunk hogy szerezzük be az "app"-ot és próbáljuk ki. Én a TF Detect és a TF Classify-t próbáltam ki, több dolgon is, azokból most párat bemutatnék:

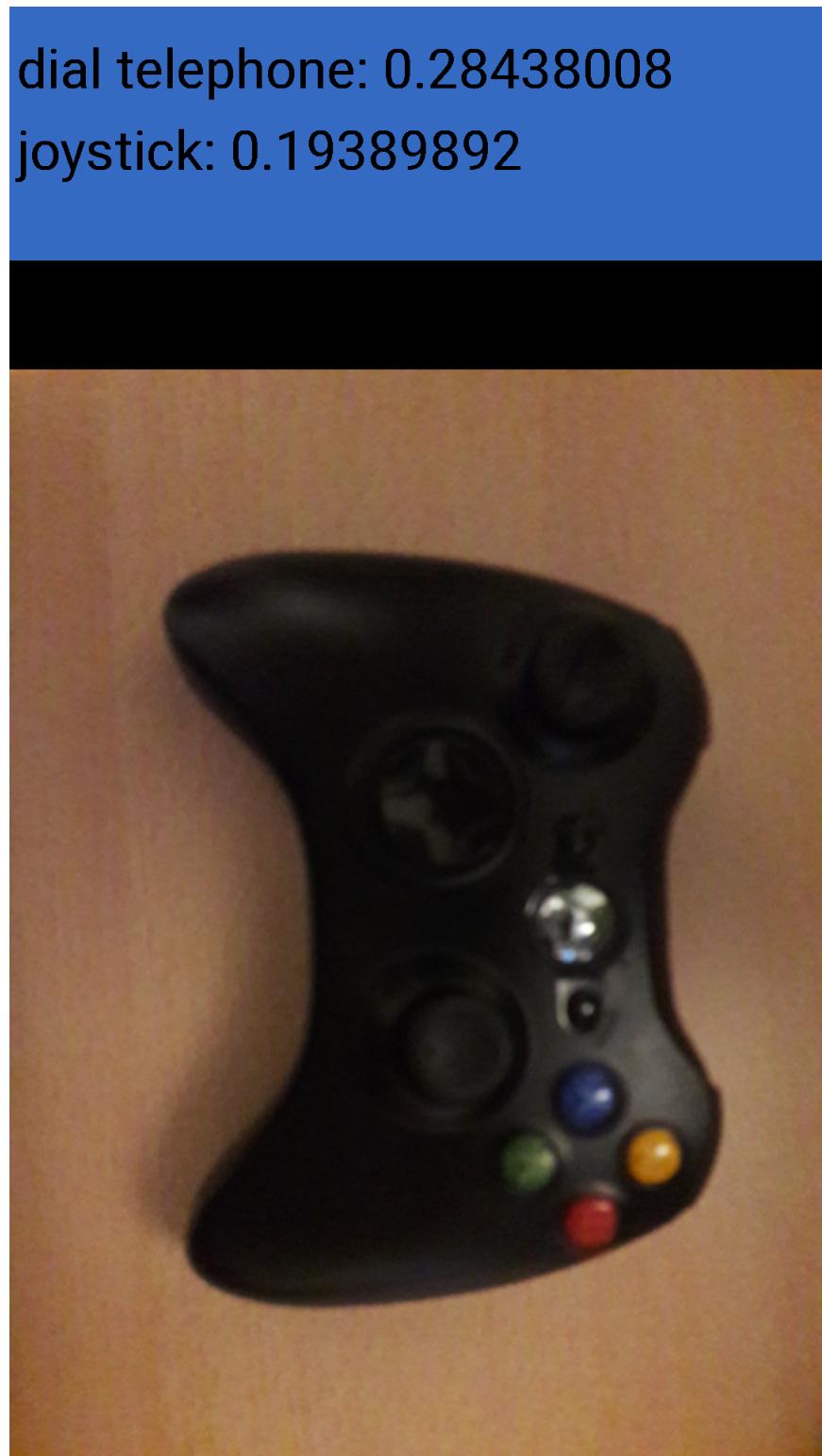


20.8. ábra. TF Detect 1

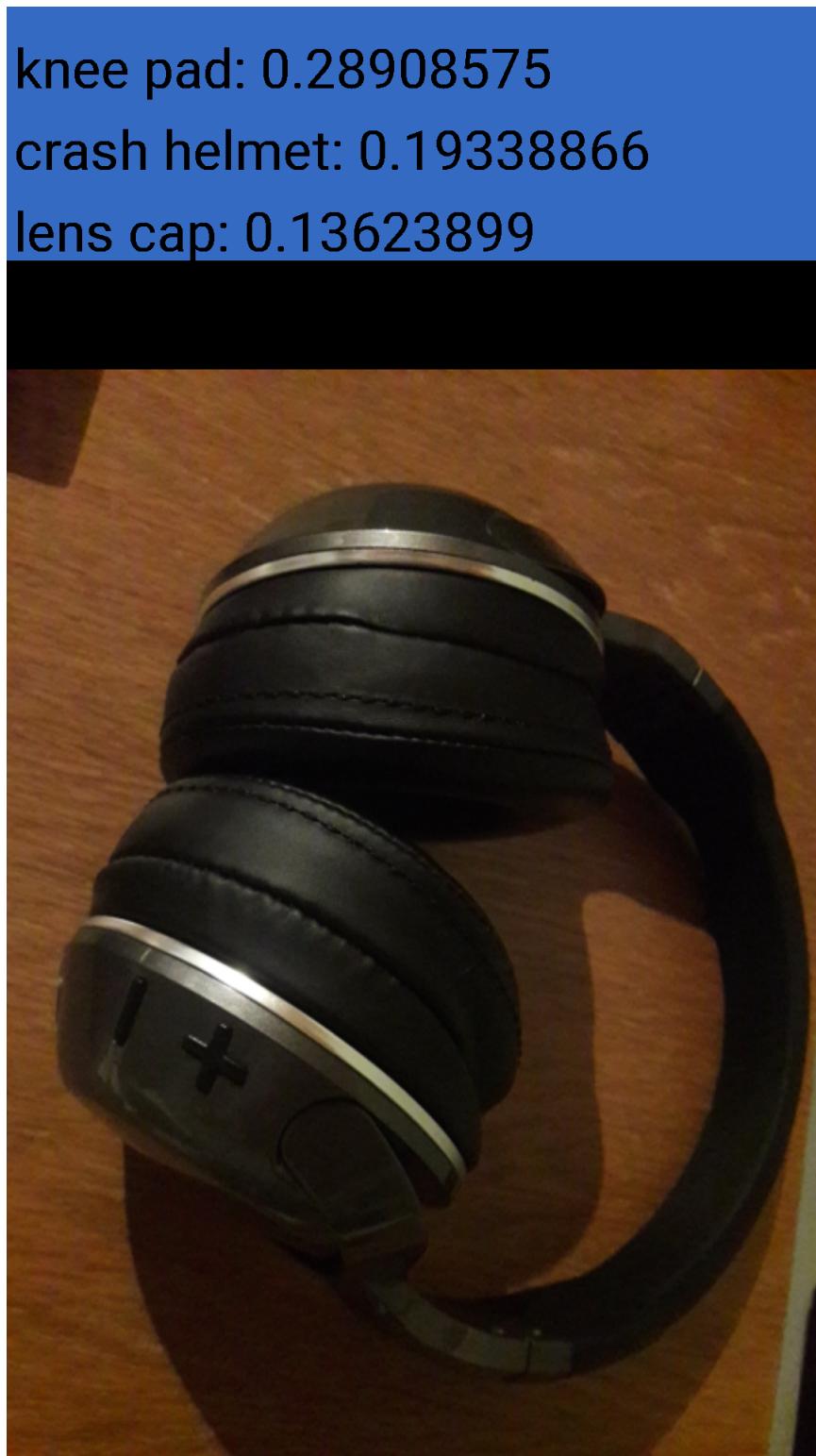


20.9. ábra. TF Detect 2

Na jó mondjuk ezen nem lepődtem meg hogy nem ismeri fel :DD

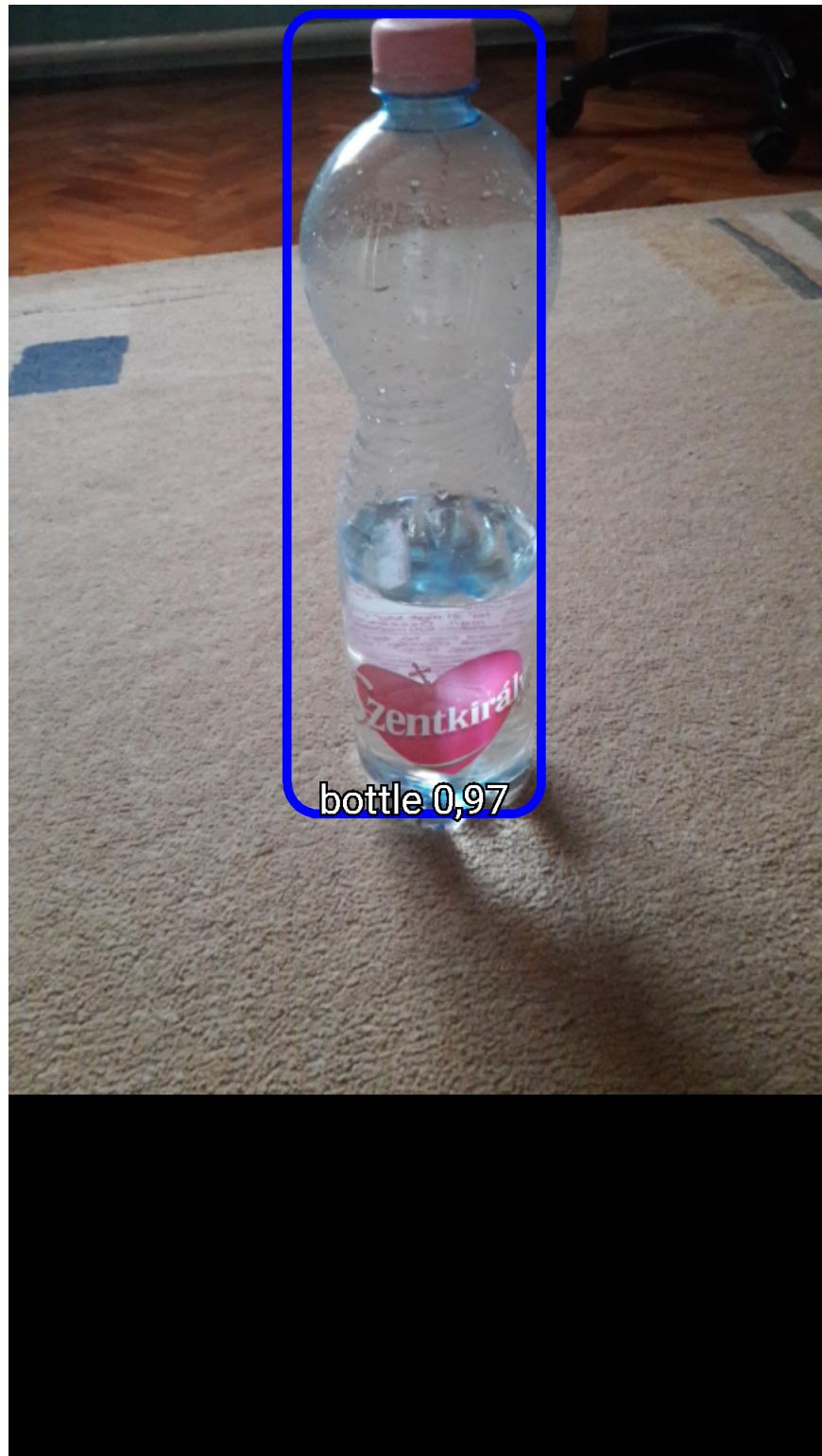


20.10. ábra. TF Classify 1

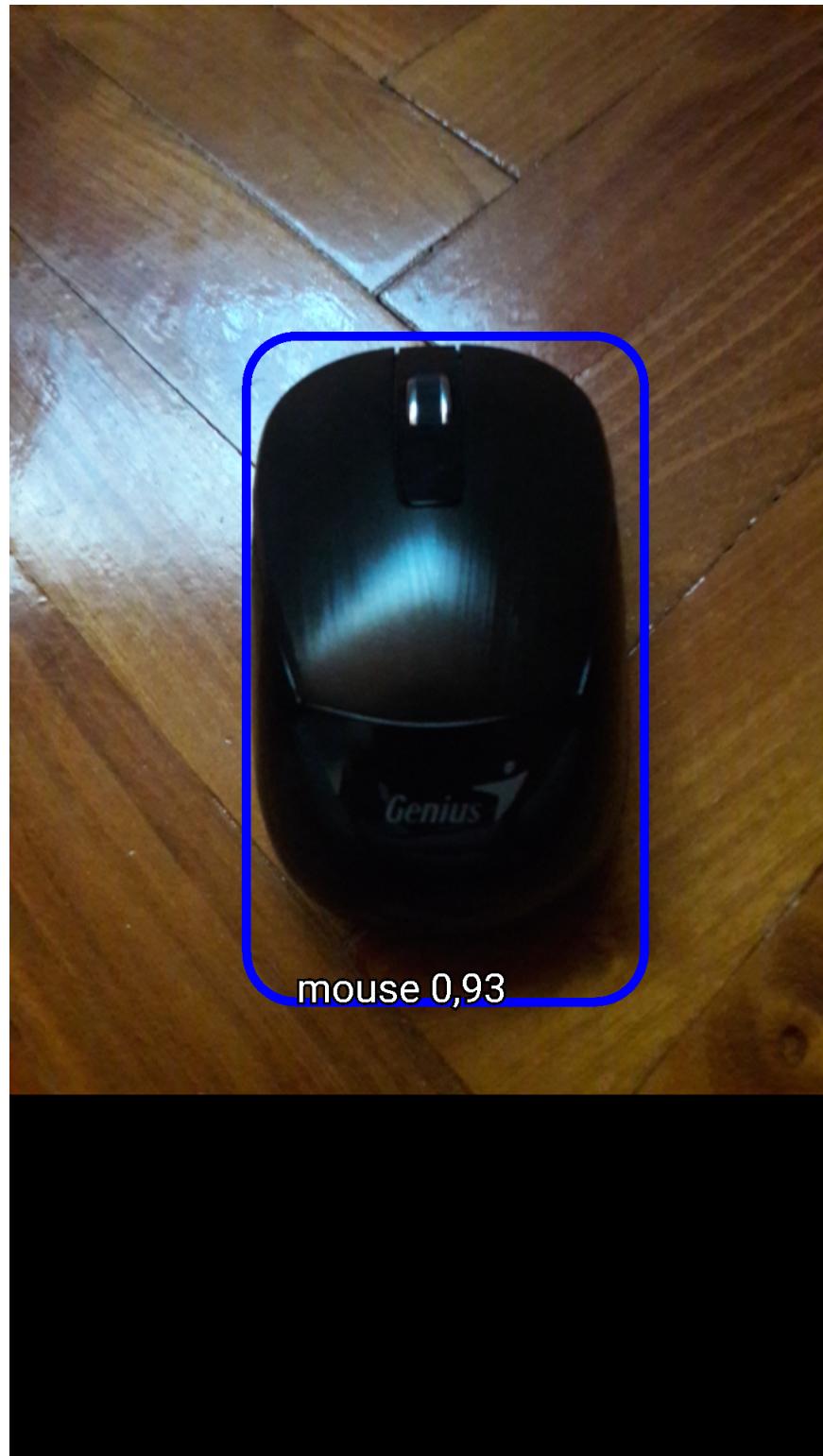


20.11. ábra. TF Classify 2

Ezeket (fenti 4 kép) lámpafénynél készítettem és nem igazán ismerte fel őket valamint az ekkor készülteket nagyon egyiket se, ezért másnap újra próbáltam délután természetes fény mellett!



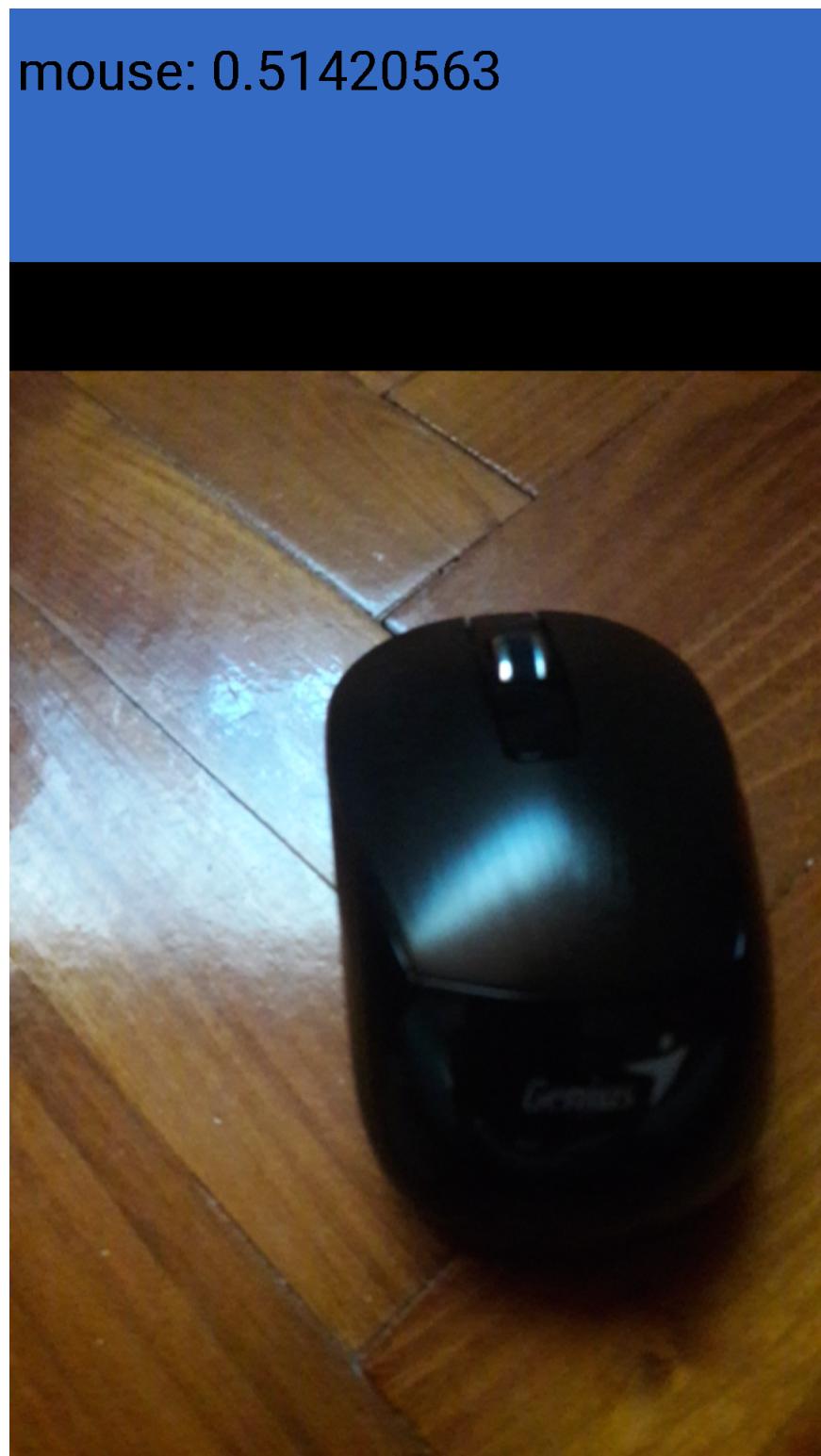
20.12. ábra. TF Detect 3



20.13. ábra. TF Detect 4



20.14. ábra. TF Classify 3



20.15. ábra. TF Classify 4

Itt már elég pontos és elfogadható megoldásokat kaptam.

## **IV. rész**

### **Irodalomjegyzék**

## 20.4. Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

## 20.5. C

[KERNIGHANRITCHIE] Kernighan, Brian W. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

## 20.6. C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

## 20.7. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, [http://arxiv.org/PS\\_cache/math/pdf/0404/0404335v7.pdf](http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf) , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPORG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEAHCackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésekért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPORG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségen született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.