

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Tóth, Donát	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

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?	5
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	7
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	8
2.6. Helló, Google!	8
2.7. 100 éves a Brun téTEL	8
2.8. A Monty Hall probléma	8
3. Helló, Chomsky!	10
3.1. Decimálisból unárisba átváltó Turing gép	10
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	10
3.3. Hivatalos nyelv	11
3.4. Saját lexikális elemző	12
3.5. l33t.l	12
3.6. A források olvasása	13
3.7. Logikus	14
3.8. Deklaráció	15

4. Helló, Caesar!	17
4.1. int *** háromszögmátrix	17
4.2. C EXOR titkosító	19
4.3. Java EXOR titkosító	19
4.4. C EXOR törő	20
4.5. Neurális OR, AND és EXOR kapu	20
4.6. Hiba-visszaterjesztéses perceptron	20
5. Helló, Mandelbrot!	22
5.1. A Mandelbrot halmaz	22
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	22
5.3. Biomorfok	22
5.4. A Mandelbrot halmaz CUDA megvalósítása	23
5.5. Mandelbrot nagyító és utazó C++ nyelven	23
5.6. Mandelbrot nagyító és utazó Java nyelven	23
6. Helló, Welch!	25
6.1. Első osztályom	25
6.2. LZW	26
6.3. Fabejárás	26
6.4. Tag a gyökér	27
6.5. Mutató a gyökér	27
6.6. Mozgató szemantika	28
7. Helló, Conway!	31
7.1. Hangyaszimulációk	31
7.2. Java életjáték	31
7.3. Qt C++ életjáték	32
7.4. BrainB Benchmark	32
8. Helló, Schwarzenegger!	33
8.1. Szoftmax Py MNIST	33
8.2. Mély MNIST	33
8.3. Minecraft-MALMÖ	33
9. Helló, Chaitin!	34
9.1. Iteratív és rekurzív faktoriális Lisp-ben	34

III. Második felvonás	35
10. Helló, Arroway!	37
10.1. A BPP algoritmus Java megvalósítása	37
10.2. Java osztályok a Pi-ben	37
11. Helló, Arroway!	38
11.1. OO szemlélet	38
11.2. „Gagyi”	39
11.3. Yoda	41
11.4. Kódolás from scratch	41
12. Helló, Berners-Lee!	42
12.1. Java 2 Útikalauz programozóknak I.	42
12.2. Java 2 Útikalauz programozóknak II.	42
12.3. Szoftverfejlesztés C++ nyelven	42
12.4. Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)	42
13. Helló, Liskov!	43
13.1. Liskov helyettesítés sértése	43
13.2. Szülő-gyerek	44
13.3. Hello, Android!	46
13.4. Ciklomatikus komplexitás	46
14. Helló, Mandelbrot!	48
14.1. Reverse engineering UML osztálydiagram	48
14.2. Forward engineering UML osztálydiagram	49
14.3. Egy esettan	50
14.4. BPMN	53
15. Helló, Chomsky!	55
15.1. Encoding	55
15.2. OOCWC lexer	57
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	60
15.4. Perceptron osztály	63

16. Helló, Stroustrup!	67
16.1. JDK osztályok	67
16.2. Másoló-mozgató szemantika	68
16.3. Hibásan implementált RSA törése	69
16.4. Változó argumentumszámú ctor	71
16.5. Összefoglaló	74
17. Helló, Gödel!	75
17.1. Gengszterek	75
17.2. C++11 Custom Allocator	76
17.3. STL map érték szerinti rendezése	76
17.4. Alternatív Tabella rendezése	77
17.5. Prolog családfa	80
18. Helló, !	82
18.1. FUTURE tevékenység editor	82
18.2. OOCWC Boost ASIO hálózatkezelése	84
18.3. SamuCam	84
18.4. BrainB	85
19. Helló, Lauda!	90
19.1. Port scan	90
19.2. AOP	91
19.3. Junit teszt	93
19.4. OSCI	95
20. Helló, Calvin!	99
20.1. MNIST	99
20.2. CIFAR-10	100
20.3. Android telefonra a TF objektum detektálója	102
20.4. Minecraft MALMO-s példa	104
IV. Irodalomjegyzék	110
20.5. Általános	111
20.6. C	111
20.7. C++	111
20.8. Lisp	111

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/v%C3%A9gtelen%20ciklus.c>

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/v%C3%A9gtelen%20ciklus%201%20mag.c>

Tutoráltam: Ujhazi Balázst

Tanulságok, tapasztalatok, magyarázat... Végtelen ciklust jelnlegi tudásom szerint két fajta ciklusnal tudom előállíthatni for ciklusnal vagy while cikusnal. A mind két megoldásom for ciklusnal készült. A különbség hogy sleep parancs amit mindenkorájával tisztázom. A pgrammot egy main függvényel indítom és ebbe írom bele a for ciklust. Utána a for ciklust végtelené teszem két db ;;-al. A printf parancsal ki kell iratni valami szöveget ,hogy lássuk az eredményt. Én a fut szót választtam. sleep parancsnak ha bele írjuk akkor tudjuk vele szabályozni ,hogy a proceszor a for ciklust egy magaon futtasa 100%-on.

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!

Tutoráltam: Ujhazi Balázst

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.

Tanulságok, tapasztalatok, magyarázat... Sajnos ez a mai világban még nem létezik T100 program .Ezért nem tudjuk eldönteni hogy van-e benne végtelen ciklus. Ezt mindenki tudja ezért nem tartom valami szuper feladatnak ezt a feladatot.

2.3. Változók értékének felcserélése

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/v%C3%A1ltoz%C3%B3%20%C3%A9rtékcser%C3%A9l%C3%A9s.cpp>

Tutorált engem: Ujhazi Balázs

Tanulságok, tapasztalatok, magyarázat... Létre kell hozni két int típusú változót amikben két értéket kell eltárolnunk. Programhoz csak egy kis matematikai tudás kell.Amit itt leírok legyen a=5 b=3 a=a+b behelyettesítve a=5+3 akkor a=8 számokkal b=a-b számokkal b=8-3 akkor b=5 b-t már sikeresen meg is cseréük Méghátra van a=a-b a=8-5 és akkor az a =5 Igy sikeresen megcserélik a két változót. Pythonban nagyon még egyszerü megírni.

2.4. Labdapattogás

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/labda.cpp>

Tutorált engem: Ujhazi Balázs

Tanulságok, tapasztalatok, magyarázat... A void fügvényel kezdjük a programot írni ami azt jelenti ,hogy nincs visszatérési tipusa. Eljárás működéshez hasonítanám amit egyáltalán nem lehet meghívni kifejezés részeként. A void fügvénnyel enyit érdemes tudni. Utána létrehoznunk két szám típusú változot és eltároljuk őket egy vátozóban, hogy könnyen tudjunk rá hivatkozni a késsőbbieken. A rendszer a következő sorban az összes értéket törölni fogja ennek a sornak köszönhetjük ,hogy a kijelzőn minden csak egy labdát láttunk. Létre kell hoznunk 2 db while ciklust ami soha nem fog meg állni , így folyamatosan patogni fog a labdánk. A ciklusokban ki iratni a és b értékeitet. Usleep egy beépitett parancs amivel a labda sebeségét tudjuk be állítani. Az emberi szem kb 500000 értéket érzékel normális labda patogásnak.A program végén felsméri a kijelző méretét max fügvény meg egy végtelen for és egy while segítségével méri fel ,hogy ki ne pattogjon a labda a kijelzről.

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 forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/sz%C3%B3hossz.cpp>

Tanulságok, tapasztalatok, magyarázat... main függvényt létrehozunk amiben bele tesszünk egy két db int típusú változót aminek értékei 1. Utána egy while ciklust hozok létre aminek a feltétele a Bogomips programban megtalálható azaz ha i nő egyel akkor a ciklus növelje meg az s változó értékét. Ha egy változó után ++ irunk akkor olyan mintha +1 egyet hozzá adnánk. A program végén ki íratjuk a megeoldást az s értéket meg egy kis plusz köritést, hogy megértse az is aki csak a porgram végeredményét lássa hogy mit is jelent a 32. Az én gépemen int most 32 bites.

2.6. Helló, Google!

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

Megoldás videó: <https://github.com/ttdoni99/Mag-Prog/blob/master/pagerank.cpp>

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/pagerank.cpp>

Tanulságok, tapasztalatok, magyarázat... Pár szót megemlítek pagerank algoritmusról mi előtt meg néznénk a foráskódöt. A google-nek a legfontosabb kerső motorja a pagerank. 1998-ban találat ki : Larry Page és Sergey Brin ahogya neve is mutatja a feltálról neveztek el a pagerankot. A PRv[] tömbben tároljuk az első iterációbeli értékeket, a PR-ben pedig mátrixszorzás eredményét tároljuk el. A mátrixszorzást a L és PRv tömb között megy végbe, pontosan ilyen sorrendben, mivel ez a művelet nem kommutatív. A szorzás eredménye lesz egy 4x1-es oszlopvektor lesz, amely hálózatunkban lévő 4 oldal PageRank-jét tartalmazza. A kiir()függvénykel pedig kiíratjuk az egyes oldalakhoz tartozó eredményeit.

2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/stp.r>

Tanulságok, tapasztalatok, magyarázat... Története: Viggo Brun bizonyította be ezta téTEL a feladata hogy ikerprimeek reciprokosszege véges értékhez konvergál. Fontos szerepe van szitaeljárások terén. Ez a függvény 1 paramétert kér be, és tovább adja a beépített matlab függvénynek. A függvény a megadott x-ig kiír minden prím számot egy vektorba. A diff változóban eltároljuk a primes vektorban lévő egymás melletti primek különbségét. Azután az idx vektorban A diff vektor elemeinek indexét tároljuk el. utána index alapján megnézzük, hogy melyek ezek párok, ahol a különbség kettő. Majd felhasználjuk a Brun-téTEL, tehát vesszük a primeknek a reciprokát, és azt adjuk össze.

2.8. A Monty Hall probléMA

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/mh.r>

Tanulságok, tapasztalatok, magyarázat... Monty Hall probléma nagyon egyszerü egy 228 IQ-jú hölgy találta ki. Egy műsorban vagyunk és tudunk három db ajtó közül egyet kiválasztani. Két db ajtó mögöt egy egy tragacs található. A harmadik ajtó mögöt pedig egy új vereseny autó. Én az első ajtót választom a műsorvezető pedig kinyit egy másik ajtót természetesen ő tudja, hogy melyik ajtó mögöt mi van amiben egy tragacs van. Ad egy lehetőséget hogy változtas a választot ajtódon. A legelején 3db ajtó közül lehet választani ezért ott csak 1:3 a valószínüség hogy azt a ajtót választom ami mögöt az új kocsi található. De mivel változik a válozonk száma azaz csak 2 db ajtó közül választunk meg a műsorvezető gondolkodása szerint nézzük emiat ha a másogik ajtót választjuk akkor meg kétszerezük a valószínüségünket azaz 2:3 a valószínüség, hogy az új autó lesz az ajtó mögöt.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/Turing%20g%C3%A9p.cpp>

Tanulságok, tapasztalatok, magyarázat... Turing gép fogalmát Alan Turing angol matematikus dolgozta ki 1936-ban jelent meg. Turing a számítógépek fejlesztésének máig tartó folyamatát indították el. A Turing-gépúgynevezett abszakt automata. Létrehozunk egy main fügvényt benne pedig két db int típusú váltózot az egyik neve b ebben fogom tárolni a számot. A másik a számláló aminek már értéke nulla. cin parancsal tudjuk eltárolni a b értéket amit a kijelzőn adja meg aki programot futtatja. Cout kiíratunk mondatokat. Lérte kell hozni egy forcikust ami anyiszor fog lefutni ahány értéket megadtunk a b váltózoban. cout kirtunk 1 -eseket és megszámlojuk b értéket és ha 5 egyes van akkor tesz egy szóközt a program. A program végén pedig ki íratjuk az egyeseket.

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

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

Tanulságok, tapasztalatok, magyarázat... Az informatikai életben fomális nyelvtan egy absztrakt struktúra. A fomális nyelvek egyik legnagyobb tűdosa Noam Chomsky volt. A munkálya hatott természetes és fomális nyelvek kutatására. A két legnagyobb formális nyevtanok: Az egyik generatív nyelvtan a másik pedig analitikus nyelvatan. Pár szót mindegyikről írok. Generatív nyelvtan : Egy szabály halmaz amelyben minden jelsorozat előállítható. A valóságban egy algoritmust formailáz ami generálja a jelsorozatot. Analitikus nyelvtan: Egy szabály hamz amelyben egy jelsorozat egymás utáni alkalmazása, a végen pedig egy bool értéket ad vissza. A valóságban egy nyelvezetjének fomúrmalizált leírása. Környezetfüggetlen nyelvtanok: bal oldalon egy önmagában álló terminális szinbólum lehet.

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

```
S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
aYbbcc (aY - aa)
aabbbcc
-----
```

```
S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
aYbbcc (aY -> aaX)
aaXbbcc (Xb -> bX)
aabXbcc (Xb -> bX)
aabbbXcc (Xc -> Ybcc)
aabbbYbcc (bY -> Yb)
aabYbbccc (bY -> Yb)
aaYbbbccc (aY -> aa)
aabbbbccc
```

A, B, C "változók"
a, b, c "konstansok"
 $A \rightarrow aAB, A \rightarrow aC, CB \rightarrow bCc, cB \rightarrow Bc, C \rightarrow bc$
S-ből indulunk ki

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

```
-----  
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 definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak

(például C89), mással (például C99) igen.

Megoldás forrása:

```
#include <stdio.h>

int main ()
{
    // Print string to screen.
    printf ("Hello World\n");
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat... A programozás nyelvén úgy lett megfogalmazva, mint egy imperativ programozási nyelv legkissebb eleme. A parancs több utasításból áll, sorban futnak le. A c program nyelv különbséget tesz az utasítások és a definíciók között, hogy az utasítás csak végrehajtható kódot táról, ezzel szemben a definíció azonosítót generál. Különbséget tehetünk egyszerű és összetett utasítások között. Az utóbbi tartalmazhat komponensként utasításokat. Ez a hello world program nem fordul le c89-es fordítóval mert nem szereti a komenteket. De már a c99-es fordítóval simmán lefordul.

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 forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/lexer.c>

Tanulságok, tapasztalatok, magyarázat... A lexikális elemző feladata, hogy a forásnyelvű program lexikális egységeit feismerje, azaz meghatározza a forásnyelvű kód szimbólumok szövegét és típusát. Az egységek definíciója reguláris kifejezésekkel adható meg. Fontos hogy nem adható meg Chomsky 3-as nyelvosztállyal mert külön válik. A lexer segítségével tudunk lexikális szabályokból elemző kódot csinálni. A kód a szamok_szama változóba számolja a számokat. Ahhoz, hogy kiírja a számok számát, a Ctrl+D billentyű-kombinációt kell használnunk. Miután kész a lex program, jöhet a futtatás és a fórdítás.

```
$ lex -o szamok.c szamok.l
$ gcc szamok.c -o szamok -lfl
$ ./szamok
```

3.5. l33t.l

Lexelj össze egy l33t cipher!

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/l33t.c>

Tanulságok, tapasztalatok, magyarázat... A leet nyelv lényege, hogy betüket kicserélünk számokra és közben végig lépkedünk input karakterein, ha véletlennel a vizsgált karakter megtalálható a szótárunkban, akkor

Létrehozunk 1 és 100 közti számot. A cipher egítségével létrehozunk egy struktúrát melyhez hozzá tartozik még egy 'char c' is. A programban minden betűt 4 lehetőségből választ ki egyet és átalakítja. Ebből a 4 lehetőségből az elsőre van a legnagyobb esélyünk. A leet története: Leet az 1980-as években a hirdetőtábla-rendszerekben keletkezet "elit" státusz lehetővé teszi a felhasználók számára, hogy hozzáférjenek a fájl-mappákhoz, játékokhoz és speciális csevegőszobákhoz.

3.6. A források olvasása

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

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

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



Bugok

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

i.

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

ii.

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

for ciklus ami 0-tól indul 5-ig megy és az i értékét növeli egyel az összes iterációban ezt a **++i** prefixnek nevezük.

iii.

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

for ciklus ami 0-tól indul 5-ig megy és az i értékét növeli egyel az összes iterációban ezt a **i++** postfixnek nevezük. Ez a különböző a két csipet között.

iv.

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

for ciklus ami 0-tól indul 5-ig tomb i-edik eleméhez hozzárendeli i-t majd növeli egyel az i értékét. De a **tomb[i] = i++** kifejezés hibát fog ki írni mert a végrehajtás sorrendje nem megfelelő, ha más fordítóprogramot használunk akkor kapunk más eredményeket.

v.

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

A for ciklus 0-tól n-ig fut, majd s-pointer hozzá rendeli magát d-pointerhez és utána az i értékét egyel növeli.

vi.

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

A printf kíratást jelent c nyelven. Itt konkrétan két db decimális számértéket iratunk ki. Ez a "%d %d" Utána f(a, ++a), f(++a, a) függvények adnak vissza, de a rossz sorrend miatt hibát kapunk.

vii.

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

Kiíratunk két db decimális számértéket az egyiket'f(a)' függvény adja vissza, a másik pedig az 'a' változo értéke.

viii.

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

Ez ugyanaz mint az előző csipet a különbség ,hogy az f függvénynek van referenciaja. Kiíratunk két db decimális számértéket az egyiket'f(a)' függvény adja vissza, a másik pedig az 'a' változo értéke.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
language="tex">>$(\forall x \exists y ((x < y) \wedge (y \text{ prim})) )$  
$(\forall x \exists y ((x < y) \wedge (y \text{ prim}) \wedge (\exists y \text{ prim})) \leftrightarrow  
)$  
$(\exists y \forall x (x \text{ prim}) \supset (x < y))$  
$(\exists y \forall x (y < x) \supset \neg (x \text{ prim}))$
```

Tanulságok, tapasztalatok, magyarázat... Az aritmetika nyelv az elsőrendűek közé tartozik. A logikai nyelv a legalapabb logikája a nulladrendű logika mert ez csak ítéletváltozókat tartalmaz, és 3 műveletet: a konjunkciót, diszjunkciót és implikációt. Erre épül rá az elsőrendű logika, kiegészülve függvényszimbólumokkal stb. Ha részletesebb tájékozódnál, akkor az interneten több könyv is található, ami a feladathoz szükséges ismereteket tartalmazza. A fentebb látható formulákat próbáltam lefordítotam emberi nyelvre: 1. minden x-hez létezik olyan y amelynél ha x kisebb akkor y prím. 2. minden x-hez létezik olyan y amelynél ha x kisebb akkor y prím, és ha y prím, akkor annak második rakkövetkezője is prím. 3. Létezik olyan y amelyhez minden x esetén az x prím és x kisebb, mint y. 4. Létezik olyan y amelyhez minden x esetén az x nagyobb és x nem prím.

Végtelen sok prímszám létezik.

Végtelen sok iker prímszám létezik.

Véges sok prímszám létezik.

Véges sok prímszám létezik.

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/matlog.tex>

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int b;
```

- egészre mutató mutató

```
"int *b;"
```

- egész referencia

```
int b=2; int *a=&b;
```

- egészek tömbje

```
"int tomb[5];"
```

- egészek tömbjének referencia (nem az első elemé)

```
int (&tomb)[5];
```

- egészre mutató mutatók tömbje

```
"int *tomb[5];"
```

- egészre mutató mutatót visszaadó függvény

```
"int *funkcio();"
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
"int *(*funkcio)();"
```

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

```
"int (*funkcio(int))(int,int);"
```

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

```
"int (*(*funkcio)(int))(int,int);"
```

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

Egy egész típusú változó amiben értéket tudunk eltárolni és aminek a hivatkozási neve egy a betű.

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

Egy egész típusú pointer ami a-ra mutat. A pointer magyarul mutatót jelent.

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

Egy egész típusú referencia ami a-ra hivatkozik. A pointer és a referencia közti különbözőség, hogy a referencia nem foglal külön memóriát le.

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

Egy egész típusú tömböt dekralál aminek a mérete öt.

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

Egy egész típusú tömbnek a referenciajá aminek mérete öt, és 'c' tömb nevére hivatkozik.

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

Egy egész típusú mutató pointert tartalmazó tömb aminek mérete 5.

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

Egy egész típusú mutató pointert visszaadó funkció ami konkrétan nem kér értéket.

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

Egy egész típusú mutató pointert visszaadó funkcióra mutató pointer.

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

Egészet visszaadó és két egészet bekérő funkcióra mutató mutatót visszaadó intet bekérő funkció.

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

Egészet visszaadó és két egészet bekérő funkcióra mutató pointert visszaadó intet bekérő funkcióra mutató pointer.

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/dekla.c>

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/haromszog.c>

Tanulságok, tapasztalatok, magyarázat... A háromszögmatrixokat más néven triangulámmátrixoknak is hívjuk. Mátrixoknak m db sorból és n db oszlopból állnak. A mátrixnak meg keresük a fő átlóját és ha az átló felet csupa nulla van akkor azt a mátrixszot alsó háromszögmaárixnak nevezzük. Felső háromszög mátrix is létezik itt pedig a főátló alatt van csupa nulla. Enyi ismeret elég is lesz a haromszögmátrixról. Nézzük a programot: Elsőnek létre kell hoznunk egy int típusú változót amiben eltároljuk a mátrix sorát mai jelen pilanatban 5. Utána létre hozok egy double típusú mutatóra mutatót tm néven. Ami 8 bitet fog lefoglalni. A kövezekző sor ki fogja írni a tm címét. A Maloc fügvény azt jelenti ,hogy az operációs rendszertől kér nemóriát ami jelenleg $8 \times 5 = 40$ bit. A következő sorban kírjuk a tm változó értékét. Létrehozunk egy for ciklust ami 0-tól nr fog tartani ami jelen esetben 5. A ciklus belsőben pedig tm[1] lefoglalja a megfelelő helyet. Után létrehozunk egy for ciklust és benne még egy for ciklust hogy tele tegyük a mátrixunkat értékekkel. Utána értéket rendelünk tm-hez pl 42 43 44 55 Free (tm) parancsal pedig felszabjuk a pointereket és vége is a programnak.

A program elemzése sorról sorra.

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

int
main ()
{
    int nr = 5;
    int **tm;
```

Most megismerkedünk az stdlib.h-val, mely a malloc utasítás működéséhez lesz szükséges. Az nr tartalmazza az oszlopok számát, ls itt deklaráljuk a **tm pointert is.

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

if ((tm = (int **) malloc (nr * sizeof (int *))) == NULL)
{
```

```

        return -1;
    }

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

```

Itt `printf` kiírja a `tm` memóriacímét, majd az `if` feltételén belül, a `tm`-et ráállítjuk a `malloc` által foglalt $5*8$ bájtnyi tárterületre. A `malloc` egy pointert ad vissza, ami a lefoglalt tárba mutat, `void *` típusút, tehát bármely típust vissza tud adni típuskényszerítéssel. Ebben az esetben ez `int**` nem kényszerítjük visszatérítésre attól még ez a viszatérített típusús de minden esetben `void*-ot` add vissza. Majd az `if` feltételeként megvizsgáljuk, hogy tudott-e lefoglalni területet a `malloc` akkor csak a program futása fejeződik be. Ha a tárfejlesztés sikerült, akkor kiírjuk a lefoglalt tár címét.

```

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

```

A `for` cikluson belül "létrehozzuk" az ábra szerinti második sor elemeit, melyek `int *` típusúak. A ciklusban 0-tól megyünk 4-ig, egyesével lépkedve. A `tm` mutatót itt úgy kezeljük, mint egy tömböt ettől még mutató marad, és a `tm` által mutatott mutatók és ahol foglalunk tárterületet, és ráállítjuk őket. Éredemes megfigyelni, hogy mindegyikhez $i+1$ -szer 4 bájtot foglalunk le, és a `malloc` `int *-ot` ad vissza. Itt is megvizsgáljuk, hogy sikerült-e a foglalás, hanem hibával térünk vissza. Most kész a második sor, és mindegyik `int *` egy harmadik sorban lévő `int`-ek csopoortjának első elemére mutat, mindegyik más csoportra.

```

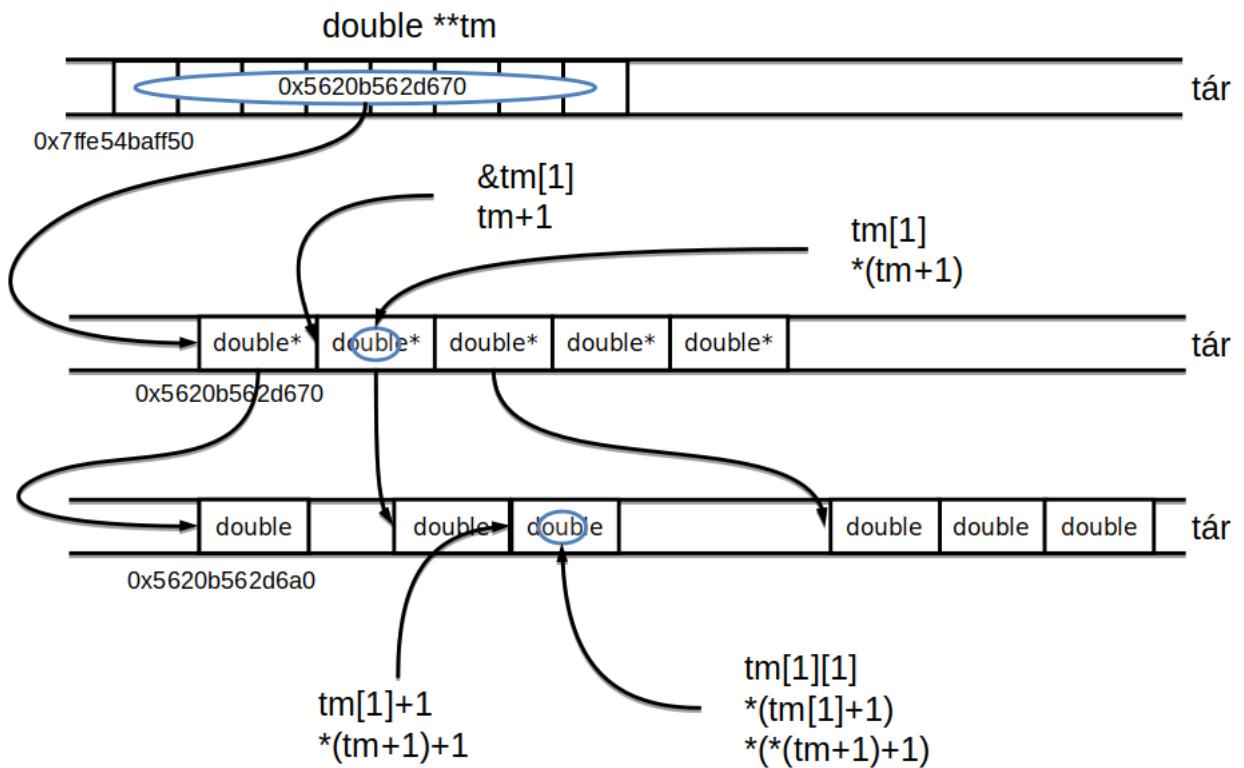
printf("%p\n", tm[0]);

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

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

```

Kiíratjuk a harmadik sor első `int` csoportjának első elemének a memóriacímét. Majd a `for` cikluson belül értéket adunk a harmadik sori `int`-eknek. Az `i`-vel megyünk a 4-ig, vagyis $nr-1$ -ig, `j`-vel pedig minden 0-tól i -ig. Az `i` jelöli a sorok számát, a `j` pedig az oszlopokat. Már a minden elemhez a sor-szám*(szorszám+1)/2+oszlopszám, és ezzel megkapjuk a feladat legelején felvázolt mátrixot, amit a következő `for`-ban már csak elemenként kiíratunk.



4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/exor.c>

Tanulságok, tapasztalatok, magyarázat... Exor magyarul kizárt vagy a logika nyelvén pedig Antivalenciának nevezzük. A igaz és B igaz akkor hamis lesz. A igaz és B hamis akkor igaz lesz. A hamis és B igaz akkor igaz lesz. A hamis és B hamis akkor hamis lesz. Ezt az igazság táblát felhasználva titkosit szövegeket a proram. Forás kód részletezése: konstansok MAX_KULCS mostantól kezdve 100, BUFFER_MERET értéke pedig 256. argc és **argv a parancssori argumentumok kezeléséhez szükséges. a Argc-ben tároljuk parancssorban megadott stringek számát, char **argv pedig a pointereket tartalmaza amik a meghatározott stringek amik első karaktere muattnak. A tömb kulcsnak a méretét íratjuk ki ami 100 és 256 lesz a puffer mérete. A következő sorban pedig kulcs indexen lépkedünk és természetesen folyamatosan növeljük ahol kell. Utána egy ciklust hozunk létre ami 0-tól tart és olvasott buffer értékéig. A ciklus belsejében összeszorozuk a puffert a kulcsal. A végén csak egy kíiratás történik és vége a programnak.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/exor.java>

Tanulságok, tapasztalatok, magyarázat... Az előző feladatot kell most java-ban megoldani. Mivel a Java a C és a C++ alapján készült, ezért az előző kód implementációja nem nehéz feldarab, de természetesen vannak különbségek, melyeket érdemes megemlíteni. Aki tud egy kicsit c++-ban programozni ezt a feladatot

nagyon könnyen meg csinálta mert a java egy objektum orientált nyelv. Különböző köztük hogy a javában classokat másnéven osztályokat kell létre hoznunk. Program leírás: A program elején argumentumok találhatóak. Utána a fügvény meg kaja ezeket az argumentumokat. A következő sorok ugyanazt jelenít mint a c kóban aki nem érti olvasa el az előző feladatot. A program végén a write funkcióval ki írja a titkosított szöveget.

4.4. C EXOR törő

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/exort%C3%B6r%C3%B6.c>

Tanulságok, tapasztalatok, magyarázat... Az előbbi feladatokban láthattuk, hogy hogyan lehet titkosított szövegeket készíteni az EXOR titkosítás segítségével. Most ennek az ellentétét kell megcsinálnunk, ami egy kicsit trükkösebb, és talán nem is tökéletes, de az előző feladatban generált titkos szöveg feltörésére alkalmas lesz. Ez a program szerintem nagyon szuper. A program elején definiáljuk strcasestr funkciót ez nagyon fontos. Const char* kifogja számolni hogy hány bájtból áll a titkosított szöveg. Az sz változót létrehozzuk és benne fogjuk tárolni a szóhosz értékét. Létrehozunk egy for ciklust ami 0-tól titkos méret-ig fut és bele teszünk egy if-et ami azt jelenti magyarul, hogy ha feltele titkos[i] egyenlő a szóközel akkor szóköz változott nővelje meg egyel az az sz-t.

4.5. Neurális OR, AND és EXOR kapu

R

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/nn.r>

Tanulságok, tapasztalatok, magyarázat... Az R program nyelvel az előző feladatokban találkozhatunk. De azért nézük meg részletesen hogy mit is érdemes róluk tudni. Az R programozási nyelvet másnéven GNU-S ként is hívunk. Az első R programot Ross Ihaka és Robert gentleman készítette innen van az R név. And igazság táblája A hamis és B hamis akor hamis lesz. A hamis és B igaz akor hamis lesz. A igaz és B hamis akor hamis lesz. Ha A igaz és B igaz akor igaz lesz. Or műveletek ha A hamis és B hamis akor hamis lesz. A hamis és B igaz akkor igaz lesz. A igaz és B hamis akor igaz lesz. A igaz és B is igaz akkor igaz lesz. A logikai kapuk táblája megadja mely logikai érték fog megjeleni a kapu kimenetében.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/mlp.hpp>

Tanulságok, tapasztalatok, magyarázat... Perceptron magyara fordítva neurális hálózat ami mesterséges. A latin perceptioból ered és a észlelés szóból. Tehát igazából a perceptron szó fordítása az észlelésre használt eszközök. Nagyon biológiai ihletésű szimuláció. Gépi tanulás a alkalmazási területe melynek célja hogy magától tanuljon ilyen gyakorlati hálókat a program. Perceptron konkrétan egy algoritmus mely bináris osztályokat tanít a számítógépnek. A program vége eredménye nagyon jól néz ki csak elég bonyolult a

maga a program 1232 sor. A feladatban folytatjuk a gondolkodást a neuron hálókról, azon belül perceptronról beszélünk. Ez egy algoritmus amely a számítógépnek megtanítja a bináris osztályozást. A neuron egy bizonyos pontot elérve, ad egy kimenetet. Tehát van egy halmaz amiben vannak piros és fekete pontok, a fekete pontok a vonal felett találhatóak, a pirosak pedig alatta. Adok a perceptronnak bemenetként egyet-egyet, mind a kettőből, és a képes lesz megmondani a többöt, hogy a vonal felett van-e, vagy alatta. Ezért nevezzük bináris osztályozásnak.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/mandelbrot.cpp>

Tanulságok, tapasztalatok, magyarázat... A mandelbrot halmaz azon c komplex számokból áll amelyekben a rekurzív sorozat nem tart végtelenbe, és az abszolút értékben korlátos. Ha komplex számsíkon ábrázoljuk a halmazt akkor fraktrál alakzat jön létre. A Mandelbrot halmazt Benoit Mandelbrot fedezte fel aki farancia-amerikai matematikus volt, és Adrien és John nevezte el róla 1982-ben. A Mandelbrot elnevezést a latin fractus szó alapján adta Adrien és John, ami az alakzatok dimenziójára utal. mandel.cpp futtatásával fogjuk elkészíteni a Mandelbrot-halmaz kétdimenziós ábráját. For ciklust futatunk végig a magasságon,szélességen és kiszámítjuk a csomopontokhoz tartozó komplex számokat. A while ciklusban az iterációkat fogjuk számolni, míg az iteráció el nem éri a határokat. Ha eléri , akkor a c számra az iteráció konvergens lesz, tehát akkor a szám a Mandelbrot-halmaz eleme lett.

5.2. A Mandelbrot halmaz a std::complex osztályal

Megoldás forrása:<https://github.com/ttdoni99/Mag-Prog/blob/master/complex.cpp>

Tanulságok, tapasztalatok, magyarázat... Ez a feladat megegyezik az 1 feladataval csak itt complex osztályal kell dolgoznunk. Itt már a complex számokat nem két változóban fogjuk eltárolni. Mivel a programozók nagyon lusták ezért nem fogunk 2 db változót használni ha egyet is lehet. A komplex könyvtár segítségével fogjuk tudni kezelní a megadott számokat. Ha nem adunk meg semmit akkor a szoftver alapértelmezett értéket fogja használni. De ha még is meadjuk akkor a kép szélességét iterációs határát, és magaságát és a számítás adatit kell megadnunk. 2 fajta ciklust használ a program for-t és while-t. A for ciklus feladata ,hogy végig fuson a háló oszlóain és sorain. While ciklusban ha el távolodik kordináta rendszéről akkor a z_n a htár elérése miatt ki fog lépni a while ciklusból. Ekkor az iteráció konvergens abban a pontban.

5.3. Biomorfok

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/3.1.3.cpp>

Tanulságok, tapasztalatok, magyarázat... Ennek a feladatnak nem sok köze van a Mandelbrot halmazhoz. Júlia halmazhoz sokal közelebb áll. Mikor kiszámoljuk a madelbrot halmazban z értéket akkor egy újabb júlia halmazt számolunk ki vele Clifford Pickover amerikai író-kutató fedezte fel a biomorfokat. A konkrétan a kifejezés a görög bios és morphe szavakból származik. Hibás volt az egyik programja és akkor talált rá a júli halmaz tanulományozása közben Azt hitte hogy valami természeti csodára lelt közben csak bugos volt a programja. Emiatt a program konkréten meg fog egyezni a mandelbrot halamzos feladtal azzal a különbözővel, hogy a küszöbszámot és a c konstans értéket a felhasználótól fogjuk be kérni. A Mandelbrot-halmaz tartalmazza az összes Júlia-halmazt is ezt majdenm elfelejtetem leírni. A biomorfokról bővebben tudunk olvasni a következő pdfben https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/cuda.cu>

Tanulságok, tapasztalatok, magyarázat... A CUDÁT az NVIDIA fejlesztette ki rfejlesztői készlet , melyet GPU-képességek kihasználására hozott létre. Azzal hogy a GPU-t grafikai számítóegységgé használjuk bonyolultab grafikai számításokra, a cpu ez időben más feladatokkal tud foglalkozni. Ez akód az Nvidia cuda módszerét használva valósítja meg. Ezzátal jóban ki tudjuk használni a videó kártyánkat. Érdemes a madelbrot halmazt cudaval megvalósítani ,mert így gyorsabb lesz a kép generáció. De csak akkor kezdjünk neki ,ha a gépünkben Nvidia márkaú video kártya található mert ha nincs akkor nem fog futni a program, ezért ez a hátránya a gyorsabb verzónak.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/farablak.cpp>

Tanulságok, tapasztalatok, magyarázat... Ez kód ugyan az szinte mint az előző feladatokban plusz funkció az , hogy rá fogunk tudni nagyítani a madelbrot hamzazra. A kód a qt programot fogja igényelni, ennek köszönhetően tudjuk majd elkészíteni a Mandelbrot halmazt beutazó programunkat. Feltelepíteni ezzel a parancsal kell: sudo apt-get install libqt4-dev. N billentyüt kell nyomni hogy részletes képet kajunk a ránagyítót területre , és újabb számítást csinál és még kifogja számolni a kijelölt z-ket . Nagyon fontos mikor futatjuk a programot 5 db c++ fájl kell hogy legyen ugyan abban mapában. A végén a makefile megvalósítja a nagyító programunkat ami már használatba is vehetünk szinte azonál.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/mandelbrotnagyit%C3%B3.java>

Tanulságok, tapasztalatok, magyarázat... Az előző feladatban már meg írtuk a mandelbrot nagyítót csak c++ nyelven szerencsére nagyon hasonlít a c++ nyelv a java-ra ezért egy jó programozóknak ez könnyű feladat A forásokban könyen megtaláljuk mandelbrot-halmaz java környezetben való kidolgozását, erre

épül majd a nagyító is. A futtatáshoz elég lesz a mandelbrot halmaz nagyító.java fájlt leszedni, majd a szokásos parancsokat kell beírni a terminálba. Ezek után szuperül fog majd működni a javas mandelbrot halamz nagyítós verziója. A meg nagyobbított részeket új ablakokban tudjuk megvizsgálni.

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 forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/pol%C3%A1rgener%C3%A1tor.java>
C++ megvalósítás .

```
$ more polargen.cpp
#include "polargen.h"

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

        double r = std::sqrt ((-2*std::log (w)) / w);
        tarolt = r*v2;
        nincsTarolt = !nincsTarolt;
        return r*v1;
    }
    else
    {
```

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

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! Elsőnek nézük meg mi a külömlbség és mi közös benük c++ és java kód között. A c++ és a java is objektumorientált programozási nyelv. Java kód sokkal érhetőbb és letisztultabb ezért könyebb is vele dolgozni. Remek bevezetés szerintem a polártranszformáció. Mi jelent a polátraszformáció ezt nézzük meg közösen addig ne menyünk tovább. A polártarnszfromáció nagyon híres algoritmus amely véletlen szerű számokat tud generálni nekünk. c++ mindenkép ki kell tömbösenk két fajta létezik a public és private A public magyara fordítva nyiválnost jelent azaz mindenki hozzá férhet. A private magyara fordítva zártat jelent azaz nem látja felhasználó. Fontos mikor a programot futatjuk hogy a kódok azonos mappában legyenek!

6.2. LZW

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/z.c>

Tanulságok, tapasztalatok, magyarázat... Elsőnek nézzük meg mi is a LZW algoritmus: Egy veszteségmentes tömörítő algoritmus. Ezt az eljárást Terry Welch publikálta 1984-ben. A kódban eslönek létere kell hoznunk egy strukturát, ami pontosan 3 db részből áll. A struktúra fog nekünk segíteni ami definiálni fogja az új tipusokat és vele fogjuk kezelni az adtokat. Azokat a kodokat amiket ki írnak 12 bitesek ezért összesen 4096 bejegyzés fér bele. Lzw tuljadonképen újítást hozott a tárolásban ami nagyon szuper. Már csak byte csoportból az első byte kerül be, és a táblaban lévő csoport byte indexe. Ezért a beérkező adatokat fa strukturában jelenítí meg. A program végén adajuk meg a fa jobb és bal mutatóját . Ha 0 a bekért érték akkor biztosítunk neki egy tuti helyet hanem 0 akkor nullára állítjuk az értéket. .

6.3. Fabejárás

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

Megoldás forrása: https://github.com/ttdoni99/Mag-Prog/blob/master/z_post.c Megoldás forrása: https://github.com/ttdoni99/Mag-Prog/blob/master/z_pre.c

Tanulságok, tapasztalatok, magyarázat... Vizsgáljuk meg hogy mit is jelentenek a külömböző fa bejárási tipusok: 1. preorder megkeresük a gyökérelemt és hozzá viszonyítunk utána jön a bal érték utána meg a jobb a sorend gyökér ball jobb és ez a lényeg hogy ebben a sorendben járjuk be a fát ez volt preorder bejárás. Ezt anyira nem szeretem. 2. inorder itt a gyökér elem középen helyezkedik el azaz bal gyökér jobb a sorend ami szerint bejárjuk a fát. 3. Ez az utolsó itt a gyökér elme az utolsó lesz azaz bal jobb gyökér lesz a sorend . Ezeket a bejárásokat ugy meg kell tanulnia a programozónak, hogy ha álmából fölkeltik akkor is tudni kell anyira alap dolog. Fa bejárások előt mindig nézük meg,hogy nem üres a bejárandó fa. Ha üres akkor véget is ér az algoritmus.

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 forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/z3a7.cpp>

Tanulságok, tapasztalatok, magyarázat... Objektumorientált szemlélet a feldatunk az az a előző c kódót át kell írni c++-ra. Legnagyobb külömlések az osztályok melyek biztosítanak egyszerűbb kezelést. Az osztály minden esetben egy továbbfejlesztett struktúra. Az LZWBInFa osztályban átmásoljuk az LZW algoritmus bináris fa építését. Az osztályba be másoljuk a fa egy csomópontjának jellemzését, ez a csomopont osztály. Ezzel külön nem is foglalkozunk emiatt csak fa részeként fogunk számolni vele. Tagfüggvényként túl tudjuk terhelni a operátort, ekkor tudjuk bele tenni a fába az inputot: Ami nem más mint ha a fában benyomjuk a b betűt akkor a tagok lehet használni a függvényben. A Globális függvényt :Mindig binárisan olvasuk és a kimenő programokat karakteresen írjuk. A kód résletezése: A programban megjelenik az operátor túlterhelés , aminek a segítségével lehetőségünk van a saját típusaink kezelésére. Most az operator << segítségével tudjuk a bemenetként kapott elemeket beleshiftelni a fába. Ha 0-át kap paraméterként, akkor létrehozzuk a gyermeket, feltéve hogy még nincs 0-ás gyermek. A new-val foglalunk tárterületet, az újNullasGyermek () függvény segítségével tudjuk az új csomópontot belefűzni a fába. (természetesen a fa mutatót a gyökérre kell állítanunk) Az 1 esetén ugyanez történik. Ha már létezik a kérdéses csomópont akkor a fa mutatót az adott csomópontra kell állítanunk. A get * függvényekkel tudjuk a mélységet, az átlagot és a szórást elérni, mivel ezek az osztály private részében vannak, szóval csak függvényen belül tudjuk őket elérni. A Csompópont osztályt az LZWBInfa osztály private részében hoztuk létre, szóval Csomopont típusú objektum nem példányosítható ezen osztályon kívül. A konstruktorban a Csomopont alapértelmezett értékét '/'-re állítjuk, a gyermeket pedig kinullázzuk. A getBetu () segítségével tudjuk beolvasni a bemenetet. A void kiir és a void szabadit olyasmi mint a C-s programban. A fát itt is inorder bejárással járjuk be. Megjelenik az LZWBInFa osztály egy idáig "ismeretlen" része is, a protected rész. Ezt olyan osztályok el tudják érni, amik az LZWBInFa-ból vannak származtatva. Itt van az átlag, a szórás és a mélység kiszámításához szükséges függvények deklarációi is, de itt találjuk a gyoker változót is, ami tagja az osztálynak. Most érkeztünk el az LZWBInFa osztály végére. Azokat a függvényeket amiket az osztályon belül deklaráltunk, az osztályon kívül definiáljuk, ezzel rövidítjük magát az osztályt. Azon függvények amik a class-on belül vannak, az LZWBInFa előtaggal érhetők el.

6.5. Mutató a gyökér

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/z3a8.cpp>

Tanulságok, tapasztalatok, magyarázat... Az előző feladatban a gyökeret objektumként kelet kezelnünk. Számos hibát fog okozni ha nem írjuk át a gyökércsomópontot pintere. Másodsor akonstruktort is át kel írni. Mindenhol ahol gyökér pointert kér csak át kell egyszerűen írni gyökér pointere. A famutatót rá kel állítani a gyökére, amit utána alul fogunk léterhözni. Utána a gyökér muatová változik és referencia nélkül fogjuk átadni. A gyökeret példányositjuk és helyet foglalunk a memóriában és konkrétan ide állítjuk rá a fát. Ha mutató értékeit elérjük akkor felszabadítás történik.

6.6. Mozgató szemantika

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/z3a9.cpp>

Tanulságok, tapasztalatok, magyarázat... A régi C++-ban nem konstans referenciakat lehetett balértékekhez kötni, konstansokat pedig bal, és jobbértékekhez egyaránt. Viszont nem volt semmi, amit hozzá lehetett kapcsolni egy nem konstans jobbértékhez. Nyelvi eszköz a felesleges másolások csökkentésére. Számos esetben elkerülhető a másolás. Nem minden van szükség másolandó adat megörzésére. Mozgatás: könnyebb, olcsóbb művelet lehet, mint a másolás. A mozgató konstruktor egy konstruktor amely objektum értékeit át rakja egy másikba.

```
LZWBinFa ( LZWBinFa && regi ) {
    std::cout << "LZWBinFa move ctor" << std::endl;

    gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
    gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

    regi.gyoker.ujEgyesGyermek ( nullptr );
    regi.gyoker.ujNullasGyermek ( nullptr );

}

LZWBinFa& operator = (LZWBinFa && regi)
{
    if (this == &regi)
        return *this;

    gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
    gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

    regi.gyoker.ujEgyesGyermek ( nullptr );
    regi.gyoker.ujNullasGyermek ( nullptr );

    return *this;
}
```

Mozgatókonstruktor és értékkadás kell, a kettő nagyon hasonló, csak az értékkadás visszaadja az objektumot (*this). Mindkét esetben a fa gyökerének elemeit kapja az üres fa (uj), és a régit kinullázzuk.

```
LZWBinFa binFa2 = std::move(binFa);

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

A mainben kiírjuk az új(!) binfát. Az eredetit kinulláztuk a feladatban első programrészlet végén.

Nézzük most az elegánsabb megoldást, amikor a mozgató konstruktor a mozgató szemantikára van alapozva! A következő kódrészletben lesz minden, ami kell ilyen téren: a konstruktor, a destruktur, mozgató konstruktor, mozgató értékadás, másoló konstruktor, másoló értékadás:

```
LZWBInFa () :gyoker(new Csomopont)
{
    fa=gyoker;
}
```

A konstruktorban létrejön a gyökér, egy új csomópontként, a fa mutatót pedig ráállítjuk a gyökérre, ugye ez mutatja, hogy állunk épp a fában.

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

A destruktur csak hívja a szabadít függvényt.

```
LZWBInFa(LZWBInFa&& regi) {

    std::cout<<"Mozgató konstruktor\n";
    gyoker=nullptr;
    *this=std::move(regi);
}
```

Itt látható, hogy a mozgató konstruktor a mozgató szemantikát hívja, mégpedig a std::move segítségével.

```
LZWBInFa& operator=(LZWBInFa&& regi) {

    std::cout<<"Mozgató értékadás\n";
    std::swap(gyoker, regi.gyoker);
}
```

A mozgató értékadásban megcseréljük a gyökeret, illetve a régi gyökerét. A jobbérték referencia is mutatja, hogy mozgatásról van szó.

```
LZWBInFa ( const LZWBInFa & regi ) {

    std::cout << "Másoló konstruktor" << std::endl;
    gyoker=masol ( regi.gyoker , regi.fa ) ;
}
```

A másoló konstruktor is csak annyit tesz, hogy a másol függvényt hívja. Ezáltal áttekinthetőbb kódot kapunk.

```
LZWBInFa & operator= (const LZWBInFa & regi) {

    std::cout << "Másoló értékadás" << std::endl;
    Csomopont* ujgyoker= masol(regi.gyoker, regi.fa);
```

```
    szabadít(gyoker);  
    gyoker=ujgyoker;  
  
    return *this;  
  
}
```

Végül pedig a másoló értékkadás, ami kicsit összetettebb. először is létrehozunk egy csomópont mutatót újgyökér néven, ez a régi másolatára fog mutatni. ezután a régi gyökeret felszabadítjuk, majd a gyökeret az új gyökérrel tesszük egyenlővé, vagyis a gyökér mostmár az új objektumunkra mutat.

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 forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag-prog/Hangyszimul%C3%A1ci%C3%BA

Tanulságok, tapasztalatok, magyarázat... Hangyaélet élményét szimuláló program qt c++ környezetben írták jó programozok. Több fájllal is kell dolgozni, melyeket egy mapában tárolók. A program a hangyák feromonokkal történő kommunikációját ábrázolja. A képernyőt cellákra osztjuk, és hangyák megkeresik azt a szomszédjukat akinek a legerősebb a feromonja és arra mennek tovább. A cellák fermonon értékei állandóan csökkennek, ha valamelyik hangya bele lép az egyikbe, akkor megugrik a fermomon szint. Az ant.hheader fileban található a hangya koordinátái, illetve az irányt, ahova tart. A paintevent a hangyák színezéséért is felelős. A closeevent függvényel kezeljük az kikapcsolását és a gomblenyomásokat. A hangyszimuláció ablakának szélességét és magasságát a public részben tároljuk el.

7.2. Java életjáték

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

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/sejtautomata.java>

Tanulságok, tapasztalatok, magyarázat... Az életjátékot John Conway találta aki matematikus volt a Cambridge egyetemen. Matematikai szempontból a sejt automaták közé tartozik. A játék egyes lépései a számítógép számolja ki. Játék szabályok: A ngyzetrács mezőit celláknak a korongokat pedig sejteknek nevezzük. Egy cella környezete a hozzá legközelebb eső 8 mező. A játék körökre van osztva ha a kezdő állapotú cellába több sejtet teszünk akkor a játékosnak nincs beleszólása. Egy sejtel egy körben 3 dolog történhet: -A sejt túl éli a kört ha van 2 vagy 3 szomszédja. -A sejt meghal ha 2-nél keveseb vagy 3-nál több szomszédja van. -Új sejt akkor jön létre egy cella környezetben ha pont 3 sejt van. A változások csak a körvégén jönnek mert így nem akadájozzák a születést és a túlélést ez nagyon fontos. A Következő lépéseket célszerű ebben a sorrendben végrehajtani: Első az elhaló sejtek megjelölése. Második a születő sejtek elhelyezése. Harmadik a megjelölt sejtek eltávolítás.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag-prog/elet%20jatek%20c++

Tanulságok, tapasztalatok, magyarázat... Számomra a qt nagyon újdonság mivel még nem nagyon használtam. A telepítése nem egyszerű előre érdemes csinálni egy privát fiókot, és meg nézni egy youtube videót mi előtt elkezdenénk a telepítést. Az előző feladat fogom átültetni c++ környezetbe, méghozzá a qt rendszer segítségével, amely nem gui-s programok fejlesztését is lehetővé teszi. A java nagyon hasonlít a c++ ennek köszönhetően a Sejtautomata.java kódja köszön vissza, több fájlban. Ha ezeket a fájlokat egy mappába teszük akkor eletjatek név-vel ellátva, és telepítjük a qmake tool-t, akkor a megfelelő parancsok beírásával létre tudom hozni az eletjatek.pro fájlt és utolsó lépésben szükségünk van a fordításhoz szükséges Makefile-hoz. A fordítás után megkapjuk a futtatható állományt eletjatek néven, amit simán futtatunk a terminálban.

7.4. BrainB Benchmark

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag-prog/BrainB%20Benchmark

Tanulságok, tapasztalatok, magyarázat... A BrainB Benchmark program a jövő kiemelkedő e-sportolóinak korai felismerésében hivatott segíteni. A program arra a élményre jelenségre épül mely során a játékos elveszíti a karakterét. Az effektek takarják egymást, ezért idővel az egyszerű programokban is nehézkessé válhat a karakterünk irányítása. A programban a Samu Entropy dobozt kell figyelnünk: akkor eredményes a mérés, ha sikerül a doboz közepén lévő kék pöttyön tartanunk a kurzort. Az értékek növekedésével egyre több karakter jelenik meg a képernyőn, a dobozok pedig egyre gyorsabban kezdenek el mozogni. Ha a játékos a mérőprogram használata során egy adott pillanattól kezdve több mint 1 másodpercen keresztül a karakteren tudja tartani az egérmutatót, akkor a játékos a megtalálta a karakterjét. Ha több, mint 1 másodpercig nincs kapcsolata akkor a játékos elvesztette a karaktert sajnos. A szoftver figyeli, hogy az egyes karakterelvesztésekhez milyen bit/sec képernyőváltások jönnek létre. A fordításhoz szükségünk lesz egy qt keretrendszer-re. Fontos, hogy a forrásfájlok ugyan abban a mappában legyenek mert ennélkülfel nem fog működni a program! A benchmark 10 percig fog futni, a végén minél bonyolultabb ké jön létre annál jobb a játékos.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/rekurziv.scm> Megoldás forrása: <https://github.com/ttdoni99/Mag-Prog/blob/master/iterativ.scm>

Tanulságok, tapasztalatok, magyarázat... A lisp nylev család hosszú történetre tekint vissza. Eredetileg programnyelvek terveztek az 50-es évek végén de a mesterséges intelligencia nyelvét vált hamar. Ma lisp nyelvet több területen alkalmazák. pl számításelmélet oktatásban Lisp nyelvet John McCarthy alkotta meg 1958-ban a Stanford egyetemen. Eredményeit 1960-ban publikálta. A Lisp kifejezésorientált nyelv. Ez a legtöbb nyelvvel ellentétben a Lispben nincs különbség kifejezések és parancsok között, minden utasítást kifejezések formájában írunk le. A megoldások lényege ,hogy az algoritmus meghívja önmagát amíg meg nem kapjuk a helyes eredményt. Ha ismét 3 faktoriálist kérünk be, akkor nem fog teljesülni az első feltétel tehát a visszatérési érték nem 1 lesz. Ha nem a megadott szám meg lesz szorozva az 1-gyel kisebb számal, azaz 2 faktoriálisal. 2 még mindig nem egyenlő 1-gyel, tehát 2 meg lesz szorozva 1 faktoriálisal. Mivel 1 egyenlő egygyel ezért egy a visszatérési érték . Az iteratív megoldásban a bekért számal megegyező alkalmal hajtódik végre az algoritmus.

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.

10. fejezet

Helló, Arroway!

10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11. fejezet

Helló, Arroway!

11.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG> (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/PolarGen
Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/OOszem
Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/polargen

Tanulságok, tapasztalatok, magyarázat... A polártranszformációs generátor matematikai része bonyolult de szerencsére az nem is érdekel most. OOszelement.cpp futattásához szükségünk van a polargen.h fájlra mert ebben tároljuk el. A program annyit csinál, hogy generál 2 random számot. Az egyiket eltárolja, a másikat pedig kiírja terminálba. Elsőnek meg kell vizsgálni, hogy van-e tárolt elem. Ehhez szükségünk van a nincsTarolt logikai változóra, mely alapértelmezetten igaz. Majd a double függvény, abban az esetben ha ez a változó igaz értékű, akkor kiszámolja a két random számot. Eltárolja az egyiket, majd a nincsTarolt-t hamisra állítja. Így elkerülhetjük azt, hogy a programnak feleslegesen kelljen újra random számot generálnia.

Most, hogy láttuk, hogyan működik a saját Polár generátorunk, nézzük, hogy a OpenJDK Random.java forrásában nézzük mrg, hogyan oldották meg ezt.

```
private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

public synchronized 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;
```

```
        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;
}
}
```

Ahogy láthatjuk a hivatalos megoldás teljesen ugyanaz, mint a mi sajtunk. Ez a feladat arra ad példát, hogy az objektumorientált programozás egyáltalán nem annyira nehéz, mint azt sokan gondolják.

Itt a kód részlet ahol generál random:

```
u1 = std::rand () / (RAND_MAX + 1.0);
u2 = std::rand () / (RAND_MAX + 1.0);
```

Utána műveleteket végez a számokal szoroz kívon összead és el tárolja a második értéket a w változoban. A program w változoban lévő számból gyököt von stb Ezt folyamatosan végre hajtódik mert benne van egy for ciklusban a terminálba 10 db számnak kell megjelenni-e mert 10-szere fut le a for ciklus.

```
for ( int i=0;i<10;++i)
std::cout << pg.kovetkezo () << std::endl;
```

Utána le áll a program.

11.2. „Gagyi”

Az ismert formális2

```
„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 referencia) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására3 , hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/Gagyi2.java
Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/Gagyi3.java

Tanulságok, tapasztalatok, magyarázat...

A feladat arra épül, hogy a Java Integer osztálya a -128-tól 127-ig pool-ozza az egészeket. Ez abban nyilvánul meg, hogy ha példányosítunk egy Integer objektumot, ami a megadott tartományban lévő értéket kap, akkor az objektum referencia ugyanarra a memóriaterületre hivatkozik. Nézzük meg ezzel kapcsolatosan az Integer.java fájlban megtalálható IntegerCache osztály forrását:

```
private static class IntegerCache {
    static final int low = -128;
    static final int high;
```

```
static final Integer cache[];
static {
    // high value may be configured by property
    int h = 127;
    String integerCacheHighPropValue =
        VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
    if (integerCacheHighPropValue != null) {
        try {
            int i = parseInt(integerCacheHighPropValue);
            i = Math.max(i, 127);
            // Maximum array size is Integer.MAX_VALUE
            h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
        } catch( NumberFormatException nfe) {
            // If the property cannot be parsed into an int, ignore it.
        }
    }
    high = h;
    cache = new Integer[(high - low) + 1];
    int j = low;
    for(int k = 0; k < cache.length; k++)
        cache[k] = new Integer(j++);
    // range [-128, 127] must be interned (JLS7 5.1.7)
    assert IntegerCache.high >= 127;
}
private IntegerCache() {}
}
```

Térjünk rá a feladat megoldására: A Java feltételezi, hogy a programok sokat dolgoznak majd kis számokkal, ha Integer-re van szükség akkor nem készít új objektumot minden esetben hanem kivesz a pool-ból egy készet, mivel Javaban a !=,== operátorunk címeket fog összehasonlítani ezért ha a pool-ból vesszük ki a két Integer-t akkor azok egyenlők lesznek, ellenkező esetben pedig nem. A poolban 127 és -128 közötti értékek találhatóak.

```
public class helloworld{
    public static void main(String args[])
    {
        Integer t = 128;
        Integer x = 128;

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

Itt mivel a 128, kívül esik a pool-on ezért, a feltételek teljesülnek és végtelen ciklust kapunk.

11.3. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-lel leáll, ha nem követjük a Yoda conditions-t! https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/Yoda.java

Tanulságok, tapasztalatok, magyarázat... A Yoda feltételek, egy programozási stílus, melyben a kifejezés két részét megcseréljük a feltételes utasításokban, eltérően a megszokott rendtől, a Yoda feltételekben a konstans része fog balra kerülni a kifejezésnek. A Yoda név a Star Wars-ból származik, aki szintén a megszokotttól eltérő stílusban, sorrendben beszéli az angolt, nagyból tárgy-alany-ige stílusban. A konstanst a Yoda feltételben az összehasonlító operátor bal oldalára írjuk, és a változót aminek az értékét hasonlítjuk a konstansal jobra. A yoda stílus előnye:A konstans bal oldalra helyezése nem változtatja meg a program viselkedését, kivéve ha az értékek hamisra lesznek kiértékelve. A Yoda feltétel hátrányai:A Yoda feltételek kritikusai szerint így nehezebben olvasható a kód, ami tulszúlyozza a fent említett előnyeit. Néhány programozási nyelv például a Swift, nem engedélyez változó hozzárendeléseket feltételes kifejezésekben. Az előny amivel elkerüljük a null pointer hibát, hátránynak is tekinthető, mivel így a null pointer hibák rejtettek maradhatnak és később okozhatnak gondot a programban.

11.4. Kódolás from scratch

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

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/PiBBP.java

Tanulságok, tapasztalatok, magyarázat... A Bailey-Borwein-Plouffe algoritmut BBP-nek rövidítik. Maga az algoritmus a Pi Hexa krakttereit számolja ki. Az algoritmust David H. Bailey talaálta ki és ő is mutatja be. Most David H. Bailey jelenleg Senior nyugdíjas tudós. A PI hexa d+1 kifejtés van néhány különböző hexa jegy. $\{16^d \text{Pi}\} = \{4 * \{16^d S1\} - 2 * \{16^d S4\} - \{16^d S5\} - \{16^d S6\}\}$ A {}-ben a törtrészben számoljuk ki a Pi hexa jegyeket. Nézünk bele egy kicsit a kódba: Ez a két sor.

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

A sorozat jegyeinek a pontoságát növeli.

```
public long n16modk(int n, int k)
```

Ez a sor bináris hatványozást számol. Azaz kiszámolja menyi $16^n \text{ mod } k$. n a kitevő k pedig a modulusz. Program lényege hogy kiírja a pi-t hexa decimális számokba.

12. fejezet

Helló, Berners-Lee!

12.1. Java 2 Útikalauz programozóknak I.

Rengeteg C++ utasítás szintaktikailag helyes Javaban és sokszor hasonló jelentéssel is bír. Előfordulhat hogy a hasonló nyelvi elemek eltérő jelentése problémát okoz. A Java mint nyelv szűkebb a C++-nál, viszont szélesebb alkalmazási területet fed le. pl: nyelvi szinten támogatja a grafikus felhasználói felület programozását és a hálózati programozást, Ezeket a fogalmakat C++ nyelv is kezeli. A C++ lehetőséget ad forrásszinten hordozható programok írását. A szabványos módon megírt forrásból szép futó programot kapunk. A futtatható bináris kód azonban nem vagy nagyon ritka esetekben hordozható. A lefordított kód ugyanis tartalmaz speciális feltételezéseket. A Java célkitűzései között szerepel a platformok közötti bináris hordozhatóság .Ez nagyon jó. A bájtkodá lefordított programot átvihetjük más gépekre is. Ezek alapján joggal számíthatunk arra, hogy a Java sokkal szigorúbb előírásokat tesz a nyelvi elemekre.

12.2. Java 2 Útikalauz programozóknak II.

12.3. Szoftverfejlesztés C++ nyelven

12.4. Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)

Élmény olvasónapló kifejtése: A Python általános célú programozási nyelv, ami rengeteg pozitív tulajdonsággal rendelkezik. A magas szintű, objektum orientált és platform független. Prototípus készítésre, tesztelésre érdemes alkalmazni, de egyszerűbb alkalmazások is hatékonyan készíthetők vele. Más modulokkal is együtt tud működni egy Python komponens. A Python egy nagyon magas szintű programozási nyelv. A Python tulajdonképpen egy szkriptnyelv, de nagyon sok csomagot is és beépített eljárás is tartalmaz, ezért komolyabb alkalmazások megírására és komolyabb problémák megoldására is használható. Python esetén nincs szükség fordítás-ra. A Python interpreter elérhető számos platformon. A Python könyü hasznáni,megbízható és jelentős támogatást biztosít hibák javítására.

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

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

Megoldás forrása: <https://reiteristvan.wordpress.com/2011/07/05/s-o-l-i-d-objektum-orientált-tervezési-elvek-3-lsp/>

Tanulság, tapasztalatok... Liskov-helyettesítés az objektumorientált programozás öt fő tervezési elve közé tartozik. S osztály T osztály leszármazottja, akkor S minden helyre behelyesíthető ahol T típust kapunk. Madar osztály lesz a példámban a T osztály.

```
class Madar {  
};  
class Program {  
public:  
    void fgv ( Madar &madar ) {  
    }  
};  
class RepuloMadar : public Madar {  
public:  
    virtual void repul() {};  
};  
class Sas : public RepuloMadar  
{};  
class Pingvin : public Madar  
{};  
int main ( int argc, char **argv )  
{  
    Program program;  
    Madar madar;  
    program.fgv ( madar );  
    Sas sas;
```

```
        program.fgv ( sas );
        Pingvin pingvin;
        program.fgv ( pingvin );
    }
```

Az S osztály: RepuloMadar, Sas, Pingvin. Két osztály alkotja a programot az LPS-ben. A programban a Madar már nem tud repülni, hiába lesz a leszármazottja a típusoknak. Liskov-helyettesítés elvére odafigyeltem.

```
class Madar {
public:
    virtual void repul() {};
};

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

class Sas : public Madar
{};

class Pingvin : public Madar
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );
    Sas sas;
    program.fgv ( sas );
    Pingvin pingvin;
    program.fgv ( pingvin );
}
```

Ebből a kódcsipetből kiindulva megérhetjük a Lisakov elvet. A T osztály továbbra is megmaradt, viszont ezúttal nem a (S) osztályban jelenik meg a repülés. Emiatt így a programban is képes repülni a madár. Sérül a Liskov-helyettesítés elve mer t a pingvin repül a kódban.

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 (98. fólia)

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/szulo.cpp

Tanulság, tapasztalatok...

```
#include <iostream>
```

```
using namespace std;
// szulo osztály, ebből fogom származtatni a leszarmaztatott ←
// osztályt.
class szulo {
public:
    void uzenet() {cout<<"Az ōs üzenete\n";}
};

//ez a függvény kapja meg paraméterként a szulo&
void fgv ( szulo& os ) {
    os.uzenet();
}

// gyerek osztály szulo-ból származtatva
class gyerek : public szulo
{
public: void uzenet() {cout<<"Az gyerek üzenete\n";}
};

int main ( int argc, char **argv )
{
    szulo* szulol = new gyerek();
    szulol->uzenet(); //az ōs üzenete

    gyerek gyerek1;
    fgv(gyerek1); //az ōs üzenete
}
```

A szulo osztály, a gyerek a szulo-ból származtatva, szulo referenciát vár paraméterűl. C++ ban csak akkor dinamikus a kötés ha a viselkedés virtuálisa van állítva. A kód első részében foglalunk helyet, majd szulo pointer állunk, majd a meghívjük a szulol által mutatott üzenetet, ami a szulo üzenete lesz. A következő lépésekben pedig a szulo referenciát váró függvénynek adjuk a gyerek típusú objektumot. Emiatt a szulo üzenetét fogjuk elérni.

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/szarmaz.cpp

```
class szarmaz
{
    public class szulo
    {
        public void uzenet ()
        {
            System.out.println ("Az ōs üzenete");
        }
    };
    public void fgv (szulo szul)
    {
        szul.uzenet ();
    }
    public class gyerek extends szulo
    {
        public void uzenet ()
        {
            System.out.println ("A gyerek üzenete");
        }
    }
}
```

```
        }
    };
    public static void main (String args[])
    {
        szarmaz szarm = new szarmaz ();
        szulo szul = szarm.new gyerek ();
        szul.uzenet ();
        gyerek gyer = szarm.new gyerek ();
        szarm.fgv (gyer);
    }
}
```

Javában minden dinamikus a kötés, ezért a fenti kóddal, már a gyerek üzeneteit érjük el.

13.3. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNISTforHumansExp3/app/src/main/java/hu/blog/bhaxor/smnistforhumansexp3/SMNISTSurfaceView.java>
Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás forrása: <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNISTforHumansExp3/app/src/main>

Tanulság, tapasztalatok... Az SMNIST for Humans egy Android alkalmazás, ami fejlesztési célból készült. Ahhoz, hogy kizárálag a .java források alapján tudunk apró módosításokat eszközölni, elég módosítani a jelenlegi értékeket. A színvilág megváltoztatásához a SMNISTforHumansExp3/app/src/main/java/hu/blog/bhaxor/smnistforhumansexp3/SMNISTSurfaceView.java fájlt is használhatjuk. Az rgb paramétereket átírjuk a bgColor-ban, akkor az app háttérszíne megváltozik.

```
int [] bgColor =
{
    android.graphics.Color.rgb(11, 180, 250),
    android.graphics.Color.rgb(11, 250, 180)
};
```

13.4. Ciklomatikus komplexitás

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

Megoldás forrása:

Tanulság, tapasztalatok...

A ciklomatikus komplexitás egy szoftvermetrika, ami egy szoftver alapján határrozza meg annak komplexitását. Az egész gráfelméletre alapul. A forráskódban felépülő gráf pontjai között lévő élek alapján számítható ki. Röviden így:

$$M = E - N + 2P$$

Ahol:

- E: A gráf éleinek száma
- N: A gráfban lévő csúcsok száma
- P: Az összefüggő komponensek száma

A gráfot az adott függvényben lévő utasítások alkotják él csak 2 utasítás között van ha az egyik azonnal végrehajtódhat, így közvetlenül számolja a lineárisan független útvonalakat a forráskódon keresztül. Ciklomatikus komplexitást a Lizard nevű kód komplexitás analizálóval számolom ki. Használata egyszerű: másoljuk be a kódot az ablakra, ezután kattintsunk az Analyze gombra és megtörténik a csoda elkezd számolni.

14. fejezet

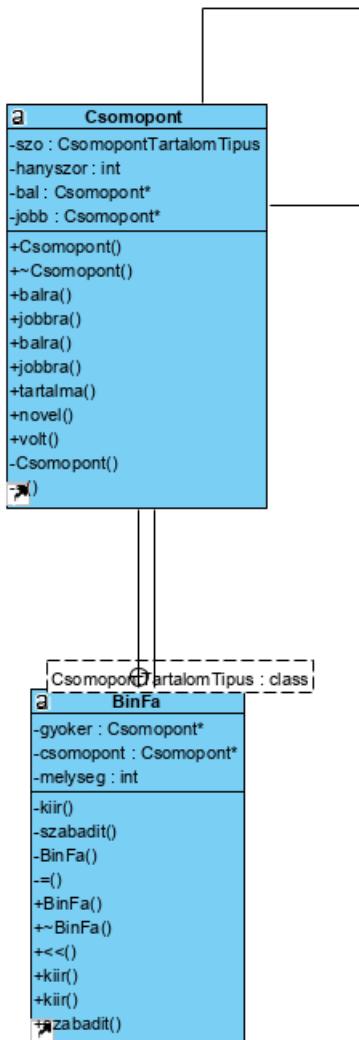
Helló, Mandelbrot!

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://arato.inf.unideb.hu/batfai.norbert/UD> (28-32 fólia)

Megoldás forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_6.pdf

Tanulság, tapasztalatok... Az UML Unified Modelling Language egy egységesített szabványosított modellező nyelv. Diagrammok készítésére jó, amik egy rendszer tervezését ábrázolják mint pl a nyers programkód. Számos szoftver elérhető, mellyel UML diagrammokat tervezhetünk vagy képezzük forráskódból illetve képezzük forráskódot belőle. Ezek a feladatleírásban megtalálhatóak és azok ingyenesek is én azonban a Visual Paradigm fizetős szoftver ingyenes próbaverzióját alkalmaztam. Ez támogatja a reverse engineeringet, tehát képes a legtöbb nyelv forráskódjából UML diagramot generálni. Ehhez először a "tools" fület kell kiválasztani, majd a "code" menüben az "instant reverse" opción, itt pedig a forrás nyelvét és elérési útját kell megadni, illetve hogy osztálydiagramot akarunk generálni. Ehhez az előző félévben már tárgyalt C++-os bináris fa programot használom. Ezután mintha varázsolunk volna itt is van egy gyönyörű szép osztálydiagram:



14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Tanulság, tapasztalatok... Az osztályok tervezése a Visual Paradigm-ban meglepően egyszerű mert teljesen vizuális felhasználói interface-el rendelkezik emiatt a kattintgatással bármit meg lehet csinálni. A generálás gyakorlatilag ugyanazon a módon történik, mint a másik irányba, csak itt nyilván az instant generatort kell alkalmaznunk az instant reverse helyett. Beszédes attribútumokat, műveleteket raktam egy osztályba, ami végül így nézett ki C++ kóddá generálva:

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

#ifndef __Class1_h__
#define __Class1_h__

class Class1;
```

```
class Class1
{
    private: string _attribute;
    private: string _attributeGetSet;

    public: void constructor();

    public: void getAttributeGetSet();

    public: void setAttributeGetSet(string aAttributeGetSet);

    public: void operation();
};

#endif
```

Az attribútumok egy attribute nevű és egy attributeGetSet nevű string a függvények pedig egy constructor függvény egy get és set függvény a második attribútumra, és egy operation függvény. Ezt importálva programunkban kifejthetjük a függvényeket amit a Visual Paradigm meg is tett magától:

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

#include "Class1.h"
void Class1::constructor() {
    throw "Not yet implemented";
}

void Class1::getAttributeGetSet() {
    return this->_attributeGetSet;
}

void Class1::setAttributeGetSet(string aAttributeGetSet) {
    this->_attributeGetSet = aAttributeGetSet;
}

void Class1::operation() {
    throw "Not yet implemented";
}
```

A konstruktor és az operation még nincs implementálva de mivel a get és a set függvény jelentése elég egyértelmű ezt megtette magától a második attribútumunkat kezelő függvény.

14.3. Egy esettan

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

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Esettan

Tanulság, tapasztalatok... Az esettanulmány az UML tervezés előnyeit próbálja megmutatni. A történet hogy van egy cég aki számítógépeket és alkatrészeket árul és az ő számukra kell egy alkalmazást készíteni. Ami nyilvántartja az alkatrészeket és a konfigurációkat. További cél hogy a jövőben minél egyszerűbben kiegészíthető koncepciót alkossunk. A megoldást esetünkben egy C++ könyvtár library segítségével oldották meg hiszen így felhasználható anélkül is hogy kiadnák a forráskódját. Ez a létrehozott keretrendszer képes kezelni elemi, vagy összetett terméket egyszerűen úgy hogy a terméket reprezentáló osztályban van egy termékeket tároló vektor. Ami tartalmazza az ő alkatrészeit amik szintén a termék osztály példányai.

UML-ben az osztályokat téglalapok ábrázolják, melyek 3 részből állnak. Az első részben szerepel az osztály neve, ezt követi az osztály attribútumai és végül tagváltozói. Mind a változók, mind a függvények előtt szerepel egy láthatóságot jelölő operátor. A "+" jelöli a publikus tagokat, a "-" a privát tagokat és a "#" a védetteket. Ezket követi az attribútum neve, típusa, multiplicitása és alapértelmezett értéke.

```
láthatóság név : típus multiplicitás = alapértelmezett ←  
érték {tulajdonságok}
```

Multiplicitásról akkor esik szó, ha az attribútum egy tömb, ez adja meg a lehetséges elemek számának a halmazát. Ez egy egész szám vagy egy egész számból álló intervallum. Szintaxisa a következő:

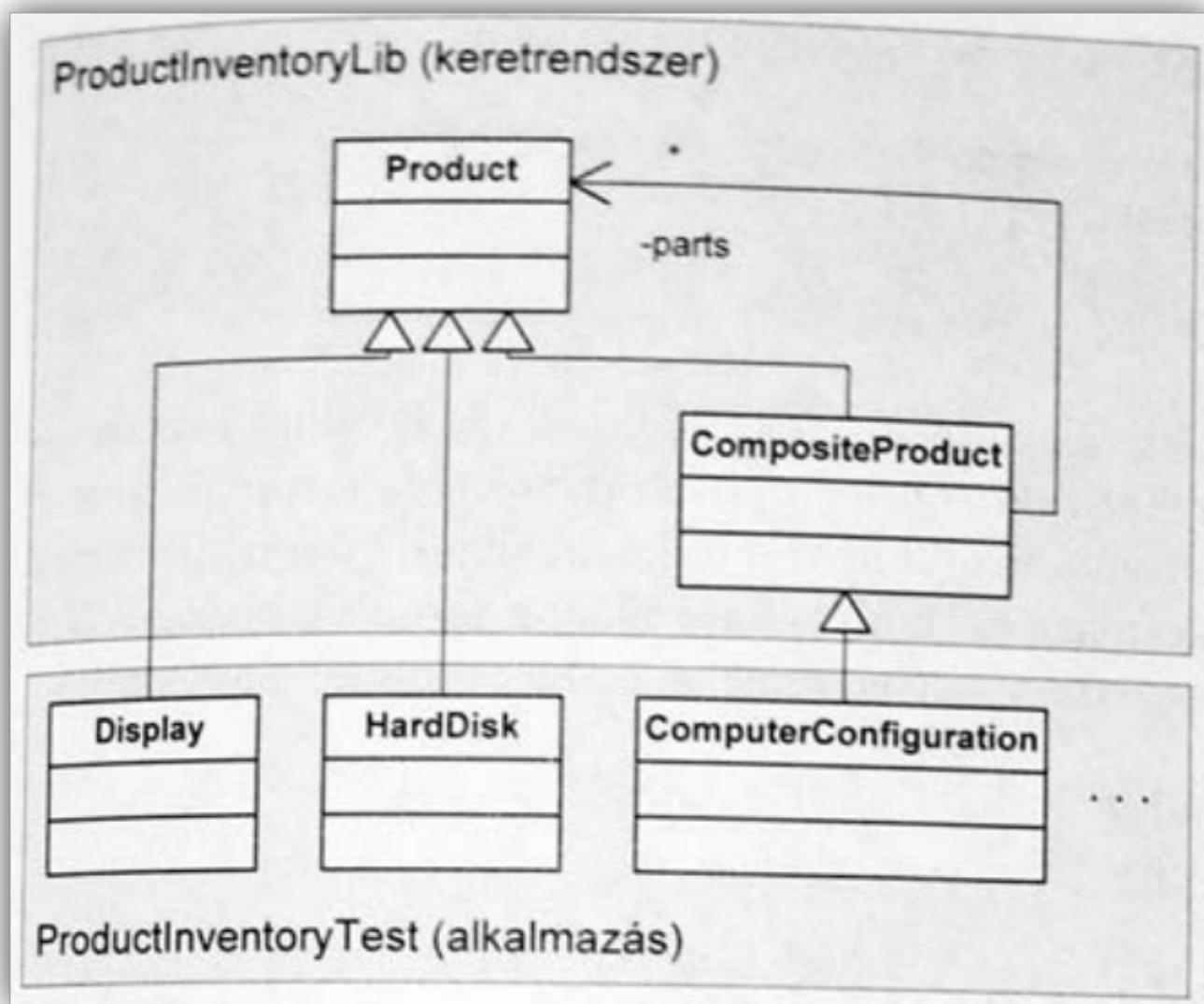
```
alsó_határ .. felső_határ  
például 0..1 (0 vagy 1)
```

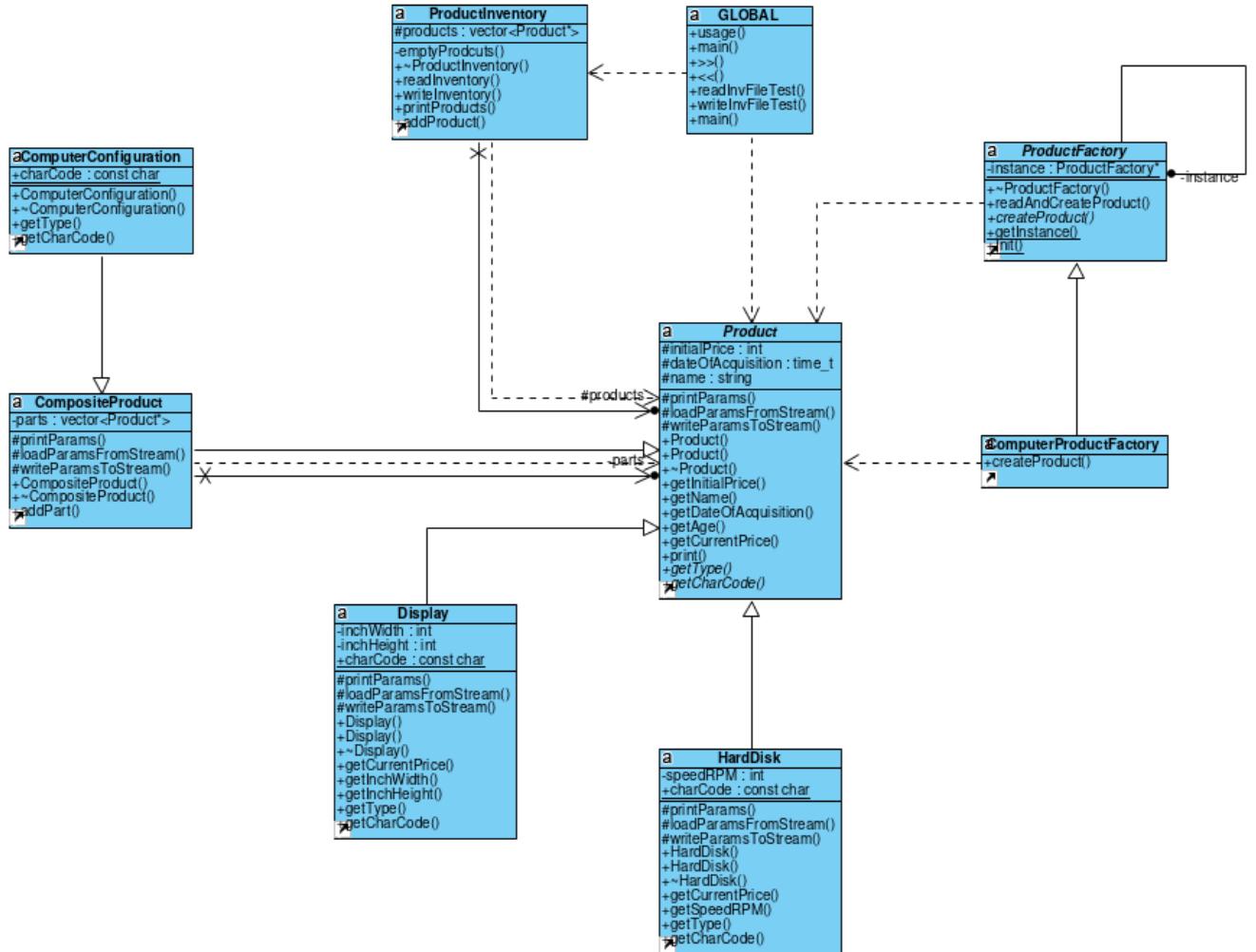
Kapcsos zárójelek között - a fentiek szerint - tulajdonságokat adhatunk meg. Ilyen például a `readOnly`, mely azt jelöli, hogy az attribútum konstans.

UML osztálydiagramok esetén a műveleteket a következő szintaxissal adhatjuk meg:

```
láthatóság név paraméterlista: visszatérési_érték { ←  
tulajdonságok}  
  
például:  
+lepesValidator(x:int, y:int):bool
```

Az osztályok UML osztálydiagrammja szerint a így néznek ki:



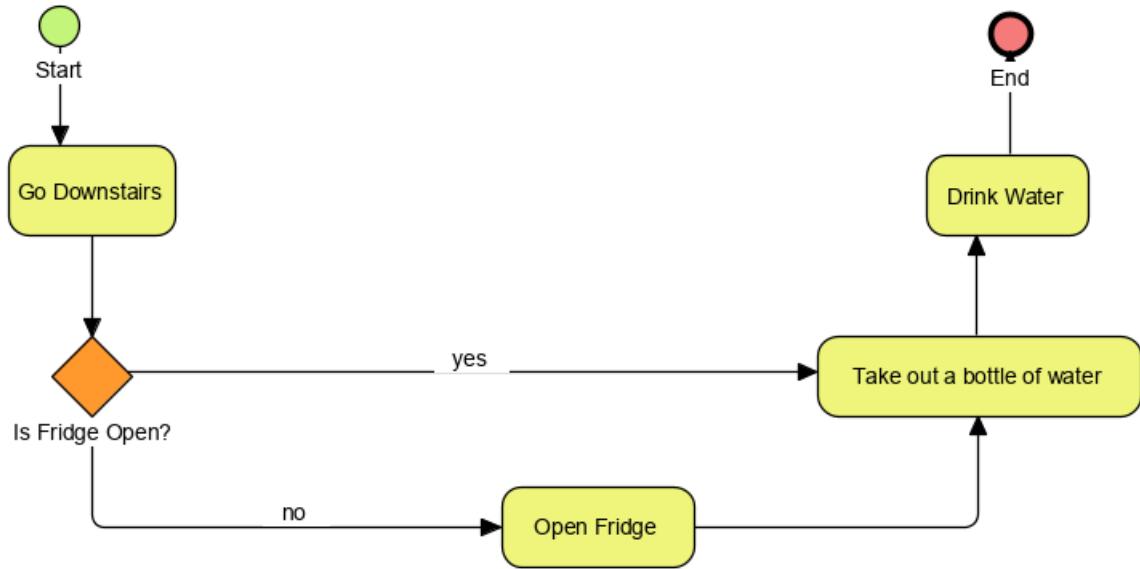


14.4. BPMN

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

Megoldás forrása:

Tanulság, tapasztalatok... A BPMN Business Process Model and Notation egy grafikai reprezentációs módja a vállalati folyamatoknak egy vállalati modellen belül. Mivel nincs sok céges tapasztalatom a minden napjaim egyik fontos elemét modelleztem le szintén a Visual Paradigm segítségével méghozzá a víz ivását mely nagyon fontos szervezetünk számára.



Ahogy láthatjuk az ábrán, 4 féle jelentősebb különböző elem van. A körök eseményeket szimbolizálnak ebben az esetben csak 2 esemény van a folyamat eleje és a folyamat vége. A kerekített téglalapok tevékenységek, azaz a munka atomi, bonthatatlan egységei. A forgatott négyzet alakú elem egy átjáró, ami a folyamat menetében villát vagy két út összeolvadását képes okozni. Az út haladási irányát megszabhatjuk feltételekkel, ahogy itt is látható. A nyilak a munkamenet irányát mutatják, amelyikre mutat egy elem az következik kronológiai utána.

15. fejezet

Helló, Chomsky!

15.1. Encoding

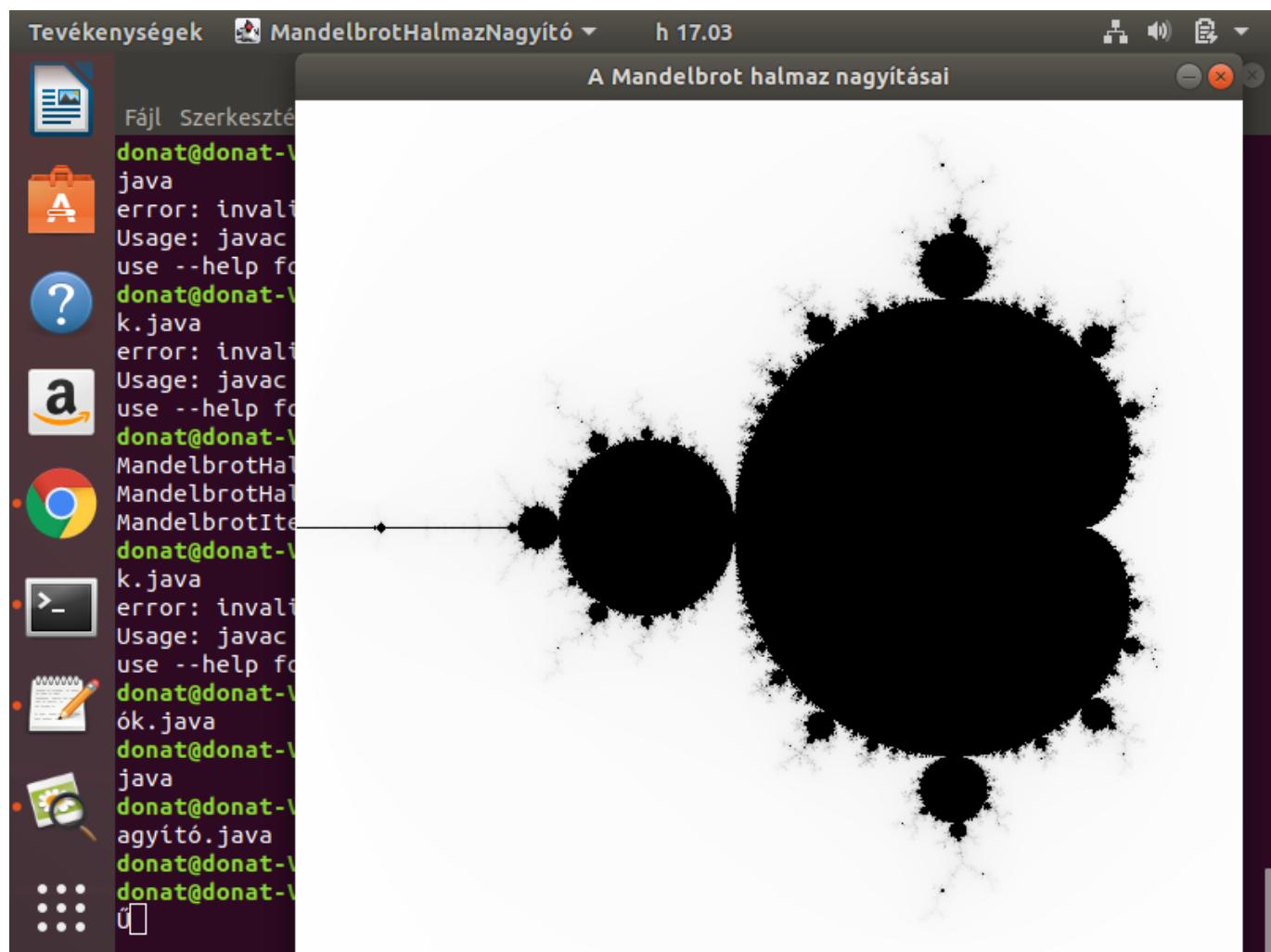
Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

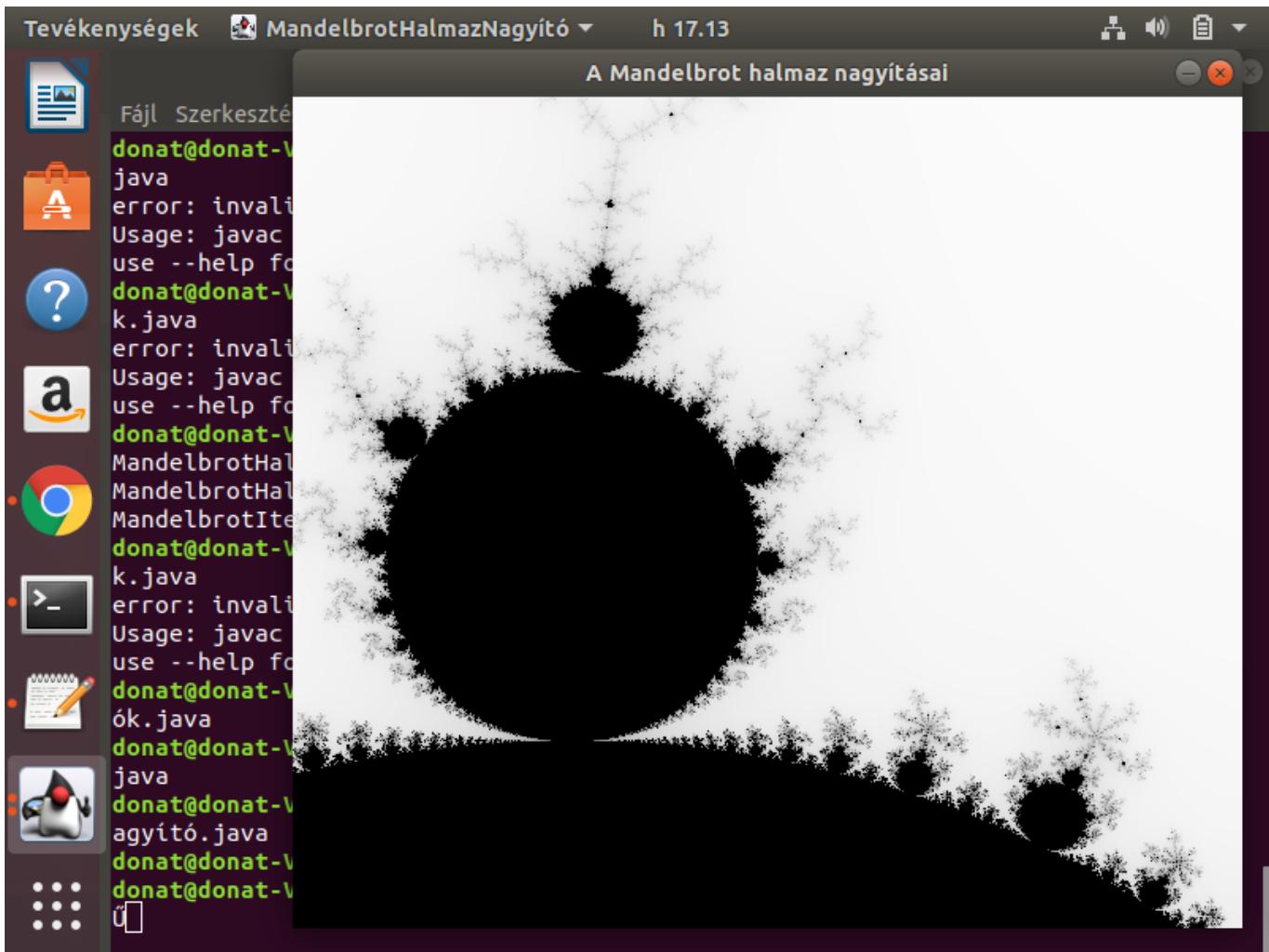
Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/MandelbrotHalmazNagyító.java

Tanulság, tapasztalatok... A MandelbrotHalmazNagyító osztály privát változói közül az x és y a nagyítandó az mx és my pedig a nagyítandó terület szélességét és magasságát tárolják. Az szülőosztály konstrukturát a super() függvény hívja meg. A setTitle függvény segítségével megadjuk az ablak címét. A mousePressed függvényben lévő x és y pozícióját tárolja el, az 1. egér gombbal pedig a nagyítandó terület kijelölését végezzük el, melyet if feltétel végez el. A pillanatfelvételek készítéséhez igénybe kell vennünk az Abstract Window Toolkit könyvtárat. A fájl nevébe beleveszük és png formátumú képet mentünk. A paint függvényben történik a Mandelbrot-halmaz kirajzolása. Hafut a számítás, akkor egy vörös vonall jelzi, hogy melyik sorban tart. A kiinduló halmazt a komplex sík tartományában keressük egy hálóval és az aktuális nagyítási pontossággal. A code futattása:

Encoding lényege a karakterkészlet helyre állítása. A következő képen tudjuk lefutatni a kódót:

```
donat@donat-VirtualBox:~/Asztal$ javac -encoding windows-1252 MandelbrotIteráció.java
donat@donat-VirtualBox:~/Asztal$ javac -encoding windows-1252 MandelbrotHalmaz.java
donat@donat-VirtualBox:~/Asztal$ javac -encoding windows-1252 MandelbrotHalmazNagyító.java
donat@donat-VirtualBox:~/Asztal$ java MandelbrotHalmazNagyító
donat@donat-VirtualBox:~/Asztal$ █
```





15.2. OOCWC lexer

Izzítsük be az OOCWC-t és vázoljuk a <https://github.com/nbatfai/robocaremulator/blob/master/justine/rcemu/src/> lexert és kapcsolását a programunk OO struktúrájába!

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/carlexer.l
Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/carlexer.l

Tanulság, tapasztalatok... A feladatban a nyelv lexikális egységeivel fogunk megismерkedni. A lexer egy olyan szoftver ami input stringeket dolgoz fel. Itt ez lesz a feladat. Előre meghatározott nevesített reguláris kifejezések vannak:

```
INIT "<init"
INITG "<init guided"
WS [ \t]*
WORD [^-:\n \t()]{2,}
INT [0123456789]+
FLOAT [-.0123456789]+
ROUTE "<route"
CAR "<car"
```

```
POS "<pos"
GANGSTERS "<gangsters"
STAT "<stat"
DISP "<disp">"
```

Ezután megfelelő blokkhoz ugrik a vezérlés és le fut a hozzá megfelelő kód részlet:

```
{DISP} {
    m_cmd = 0;
}

{POS} {WS} {INT} {WS} {INT} {WS} {INT} {

    std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
    m_cmd = 10001;
}

{CAR} {WS} {INT} {

    std::sscanf(yytext, "<car %d", &m_id);
    m_cmd = 1001;
}

{STAT} {WS} {INT} {

    std::sscanf(yytext, "<stat %d", &m_id);
    m_cmd = 1003;
}

{GANGSTERS} {WS} {INT} {

    std::sscanf(yytext, "<gangsters %d", &m_id);
    m_cmd = 1002;
}

{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;
}
```

```
{INIT}{WS}{WORD}{WS}("c"|"g") {  
  
    std::sscanf(yytext, "<init %s %c>", name, &role);  
    num = 1;  
    m_cmd = 0;  
}  
{INIT}{WS}{WORD}{WS}{INT}{WS}("c"|"g") {  
  
    std::sscanf(yytext, "<init %s %d %c>", name, &num, &role);  
    if(num >200)  
    {  
        m_errnumber = 1;  
        num = 200;  
    }  
    m_cmd = 1;  
}  
{INITG}{WS}{WORD}{WS}("c"|"g") {  
  
    std::sscanf(yytext, "<init guided %s %c>", name, &role);  
    num = 1;  
    m_guided = true;  
    m_cmd = 3;  
}  
{INITG}{WS}{WORD}{WS}{INT}{WS}("c"|"g") {  
  
    std::sscanf(yytext, "<init guided %s %d %c>", name, &num, &role);  
    if(num >200)  
    {  
        m_errnumber = 1;  
        num = 200;  
    }  
    m_guided = true;  
    m_cmd = 2;  
}  
. { ; }
```

Az fenti kódban bizonyos input stringek hatására a sscanf függvény más értékeket olvas be, minden csak annyit amennyi meg van adva, emiatt könyebb a beolvasás mert pontosan tudjuk, mit akarunk beolvasni.

A Car Lexer headerjére vessünk még egy pillantást, hogy a program struktúráját egy kicsit megértsük:

```
private:  
    int m_cmd {0};  
    char name[128];  
    int num {0};  
    char role;  
    int m_errnumber {0};  
    bool m_guided {false};  
    std::vector<unsigned int> route;  
    int m_id {0};  
    unsigned int from {0u};
```

```
unsigned int to {0u};
```

Ez az osztály a FlexLexer alosztálya, és getter függvények vannak benne, az előbbi változókhöz, illetve egy virtuális függvény. Ezzel az osztállyal egy helyes bemenetről sikeresen fogunk beolvasni, ezt a projectben használják. Az információk egy tcp porton érkeznek, és a lexer feladata az ō feldolgozásuk, a beérkező stringek az input stringek, ezeket dolgozza fel a lexer ahogy fentebb láthattuk. A beérkező adatok megfelelően továbbítodnak a programban.

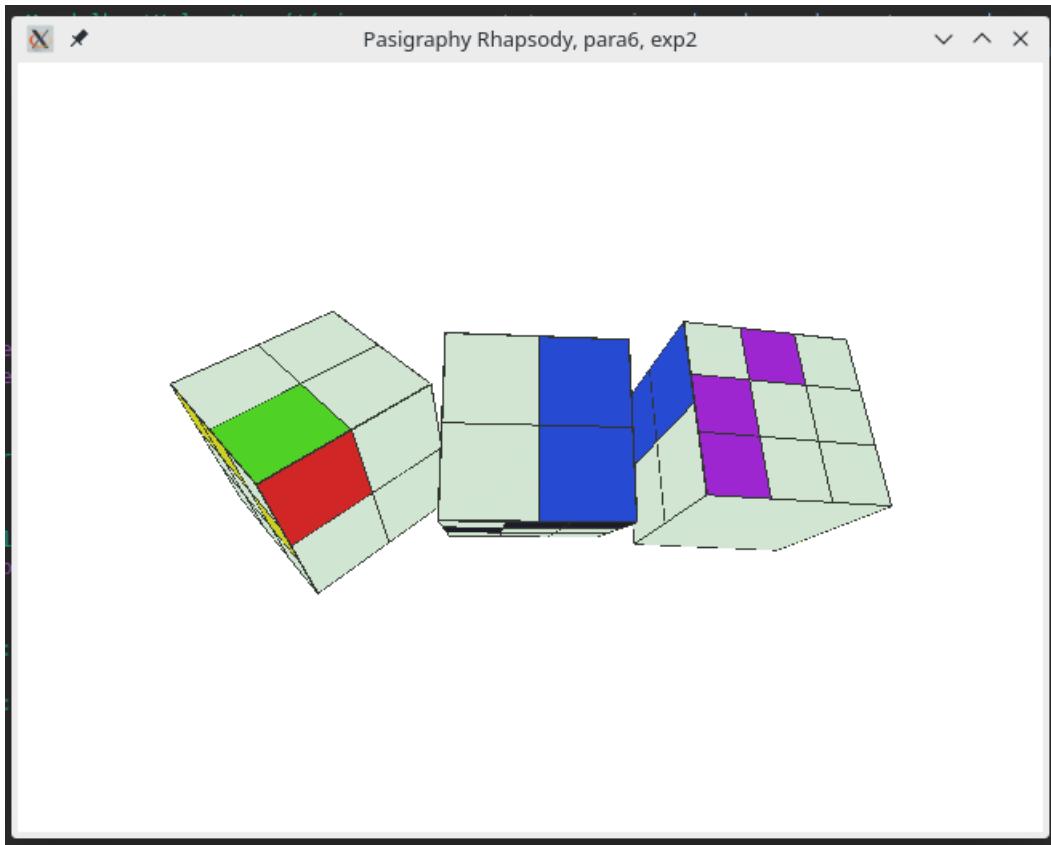
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/PR_Open

Tanulság, tapasztalatok... A Paszigráfia Rapszódia egy olyan mesterséges nyelv kialakítására törekvő kezdeményezés, mely lehetővé teszi a homunkulusz és a mesterséges homunkulusz közötti kommunikációt. A PaRa vizualizációjának alapját az első felvonásban már bemutatott SMNIST adja, azonban itt nem a pöttyök számossága, hanem pontos helyzete a meghatározó. A Paszigráfia Rapszódia OpenGL alapú vizualizációjában a kockákban négyzet betűk vannak, melyek minden irányban külön forgathatóak. A para6 vizualizációban háttér nélkül látható a három kocka, néhány színezett négyzettel. A kockák forgatásához a nyilakat kell használni. A kockák méretét növelni tudjuk a '+', csökkenteni '-' gomb segítségével tudjuk. A következő parancsokkal megvalósítani a fordítást és a futtatást:

```
$ g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut
$ ./para 3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:
0:1:0:1:0:1:0:2:2:0:1:1:1:3:2:1:0:2:0:2:1:1:1:2:3:0:
1:1:1:1:0:3:3:0:1:0:2:1:0:1:0:2:2:0:0:0:1:3:1:0:1:3:
2:1:0:2:0:3:3:0:1:0:2:1:0
```



A következő módosításokat kell végre hajtani: Az ablakunk méretét a w és h változók értékének átírásával tudjuk megváltoztatni.

```
int w = 1024;  
int h = 768;
```

A szín módosításához 3 argumentumra van szükségünk: a red és green és blue értékek 0 és 1 közé eshetnek. Ez a kód részlet lilára színezi be a kockánkat:

```
glBegin ( GL_QUADS );  
	glColor3f ( 0.5f, 0.0f, 1.0f );
```

A keyboard függvényben tudjuk módosítani a gombokat ha szeretnénk.

```
void keyboard ( unsigned char key, int x, int y )  
{  
    if ( key == 'q' ) {  
        index=0;  
    } else if ( key == 'w' ) {  
        index=1;  
    } else if ( key == 'e' ) {  
        index=2;  
    } else if ( key == 'r' ) {  
        index=3;  
    } else if ( key == 't' ) {  
        index=4;  
    } else if ( key == 'z' ) {  
        index=5;  
    }  
}
```

```
        index=5;
    } else if ( key == 'u' ) {
        index=6;
    } else if ( key == 'i' ) {
        transp = !transp;
    } else if ( key == 'm' ) {
        ++fovy;

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

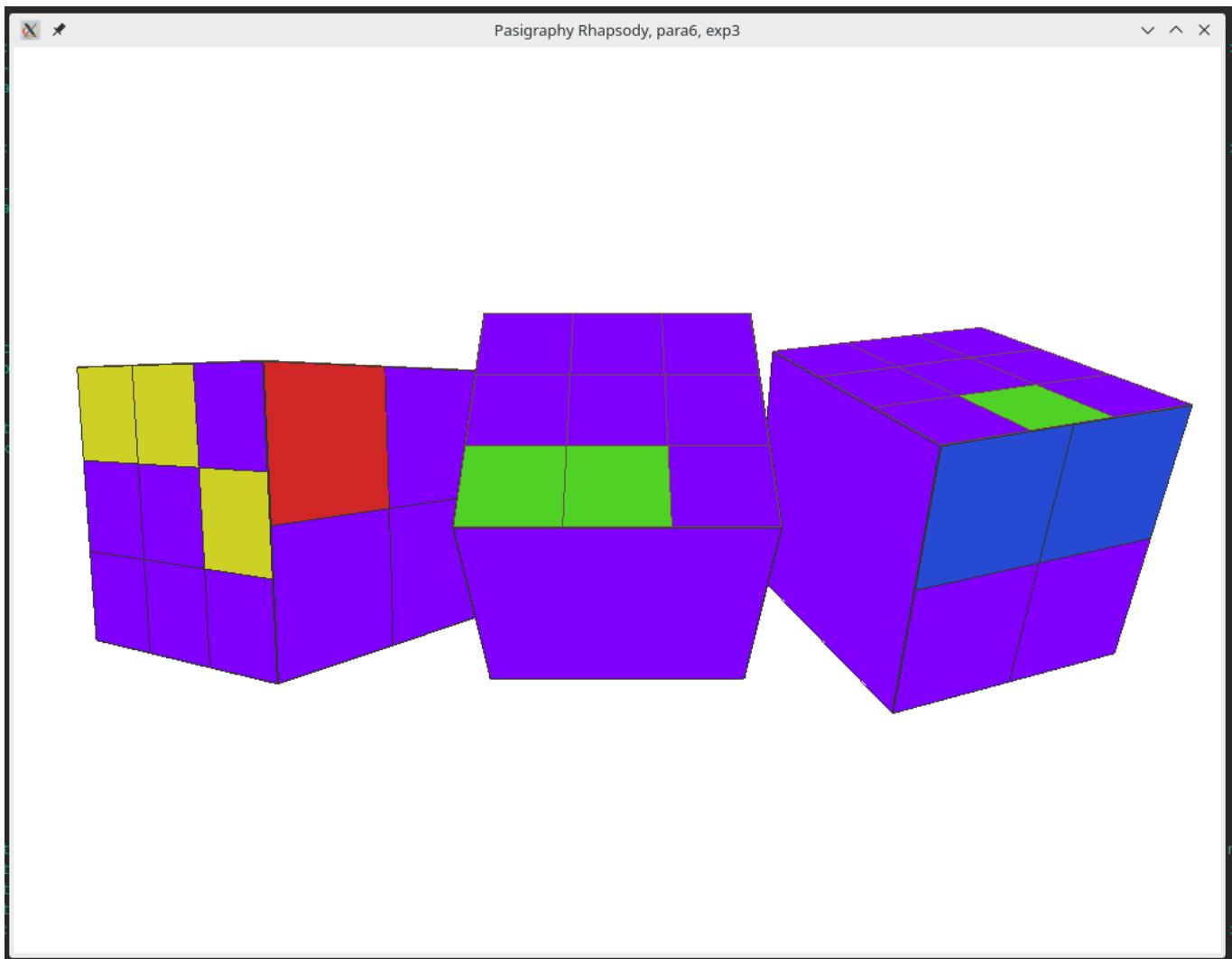
    } else if ( key == 'p' ) {
        --fovy;

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

    }

    glutPostRedisplay();

}
```



15.4. Perceptron osztály

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

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Perceptron

Tanulság, tapasztalatok... A Benoît Mandelbrot nevéhez köthető Mandelbrot halmaz a komplex számsíkon ábrázolva egy fraktálalakzatot jelent. A halmaz png ábráját fogjuk először létrehozni a `mandel.cpp` file segítségével. A sikeres fordításához és futtatásához szükségünk lesz a libpng++ könyvtárakra és a `png++/png.hpp` header fájlra. A kód fordításához ezeket a parancsokat kell beírni a terminálba:

```
$ sudo apt-get install libpng-dev
$ sudo apt-get install libpng++-dev
$ tar -zxf png++-0.2.9.tar.gz -C ~/Letöltések
$ cd ~/Letöltések/png++-0.2.9
$ make
```

A perceptron az egyik legegyszerűbb előrecsatolt neurális hálózat. A `main.cpp` segítségével fogjuk szimulálni a hiba-visszaterjesztéses módszert, mely a többrétegű perceptronok (MLP) egyik legfőbb tanítási

módszere. Ahhoz, hogy ezt fordítani és futtatni tudjuk később, szükségünk lesz az mlp.hpp header filera, mely már tartalmazza a Perceptron osztályt. A mandel.cppfuttatásával létrejött Mandelbrot png ábrát fogjuk beimportálni. A header filenak köszönhetően megadhatjuk a rétegek számát, illetve a neuronok darabszámát. A beolvasásra kerülő kép piros részeit a lefoglalt tárba másoljuk bele. A mandel.png alapján új képet állítunk elő, mely megkapja az eredeti kép magasságát és szélességét.

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

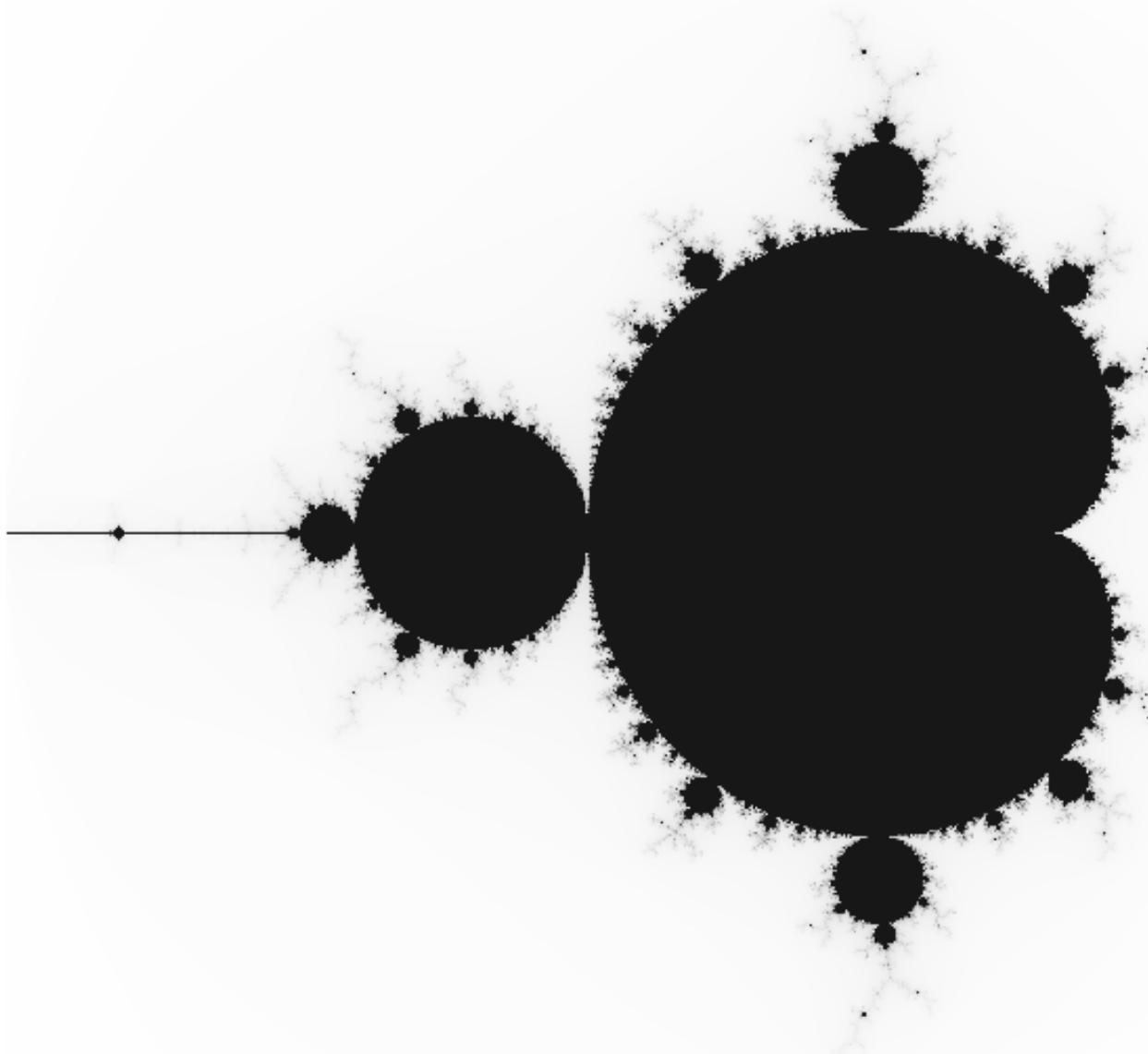
png::image <png::rgb_pixel> kep (szelesseg, magassag)
```

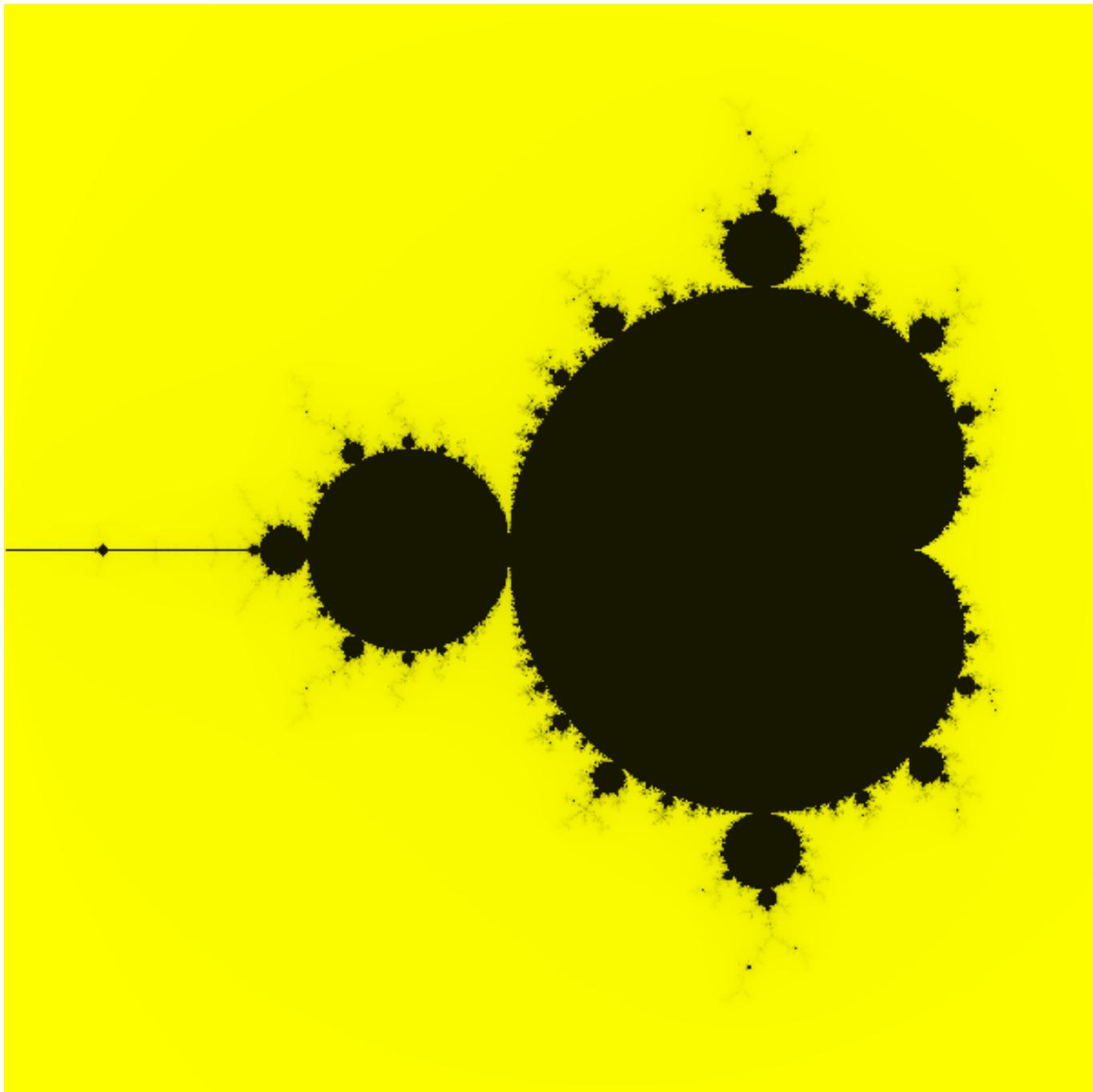
A visszakapott értékeket megfeleltetjük a blue értékeknek.

As Perceptron feladatához képest módosításokat kell végeznünk a header file-on is, ugyanis új képet akarunk előállítani. A double pointer () operátor már egy tömböt térít vissza, melynek segítségével bele tudunk nyúlni a képhez. Az utolsó units tömb értékei átkerülnek a paraméterként kapott tömbbe, az eredeti és az új kép egyforma méretű lesz.

```
srand(time(nullptr));
for(int i = 0; i < png_image.get_width(); ++i)
    for(int j = 0; j<png_image.get_height(); ++j) {
        png_image[i][j].green = rand()%256; //newPNG[j* ←
                                         png_image.get_height() + j];
    }
png_image.write("output.png");
```

A fordításhoz a C++11 szabványt használjuk.





16. fejezet

Helló, Stroustrup!

16.1. JDK osztályok

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

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/JDK

Tanulság, tapasztalatok... A JDK program elkészítéséhez mintául szolgálhat a `fénymond.cpp` fájl, melynek függvénye a fájlok kereséséért és listázásáért felelős. A `boost.cpp`-vel tehát a JDK-ban található `src.zip`-ben lévő java állományokat fogjuk kilistálni a könyvtárszerkezet kicsomagolása után, majd ezek alapján kiiratjuk a JDK osztályok számát. A Boost könyvtár segítségével rekurzívan megyünk véig a könyvtárszerkezeten, így a java fájlokban lévő osztályok kigyűjtése felgyorsul és leegyszerűsödik. Ha az `src.zip` tartalmát kicsomagoltuk, akkor következhet a `.cpp` fordítása. Ezután ha argumentumként megadjuk az `src` mappát utána futtatáskor megkapjuk a JDK osztályok számát. A kódok futatása:

```
$ g++ boost.cpp -o bejaro -lboost_system -lboost_filesystem -<br/>-lboost_program_options -std=c++14  
$ ./boost src
```

```
"src/org/omg/DynamicAny/NameValuePair.java"
"src/org/omg/DynamicAny/DynValueOperations.java"
"src/org/omg/DynamicAny/DynValueBox.java"
"src/org/omg/DynamicAny/_DynAnyStub.java"
"src/org/omg/DynamicAny/NameValuePairSeqHelper.java"
"src/org/omg/DynamicAny/DynArrayHelper.java"
"src/org/omg/DynamicAny/AnySeqHelper.java"
"src/org/omg/DynamicAny/DynValueCommon.java"
"src/org/omg/DynamicAny/DynValueHelper.java"
"src/org/ietf/jgss/Oid.java"
"src/org/ietf/jgss/GSSName.java"
"src/org/ietf/jgss/GSSException.java"
"src/org/ietf/jgss/GSSContext.java"
"src/org/ietf/jgss/MessageProp.java"
"src/org/ietf/jgss/GSSManager.java"
"src/org/ietf/jgss/GSSCredential.java"
"src/org/ietf/jgss/ChannelBinding.java"
7701
```

16.2. Másoló-mozgató szemantika

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

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Szemantika

Tanulság, tapasztalatok... Fel kell tünjön, hogy itt mély másolásról van szó. Először is létrehozzuk az Array nevű osztályunkat, mely tartalmazni fogja az elemszámláló konstruktorunkat és a másoló konstruktorunkat. A copy constructor segítségével beletölthük az eredeti tömb elemeit az új tömbbe. Az Array osztály részeként következhet a destruktur megírása, melynek feladata a memóriatár felszabadítása. Az array destructor kiírja a két tömb első elemének kezdőcímét, majd felszabadítja a memóriát a tömbök kiürítésével. Már csak a másoló értékadó operátor hiányzik. Túlterheljük az egyenlőségjel operátort, a paraméterét nem módosítjuk. Ha az eredeti és a copy ctor segítségével feltöltött tömb nem egyenlő értékekkel rendelkezik, akkor a két tömb elemszámát egyenlővé tesszük, majd kiürítjük az első tömböt. Ezután létrehozunk egy új tömböt megegyező elemszámmal, majd for ciklussal az első tömb aktuális elemét egyenlővé tesszük az új tömb aktuális elemével. Futtatáskor a következőket kapjuk:

```
$ g++ copy.cpp -o copy
$ ./copy
Copy Constructor
5
```

A mozgató szemantika lényege, hogy átruházza az erőforrásokat egyik objektumról a másikra, tehát az eredeti objektum elveszti a tartalmát.

16.3. Hibásan implementált RSA törése

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

Megoldás forrása: https://gitlab.com/bedote/bhx/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/RSA

Tanulság, tapasztalatok...

A Java a JDK 1.1 verziótól tartalmaz két új osztályt, a BigInteger-t és a BigDecimal-t. Az osztályokat azonban a szándékkal fejlesztették ki, hogy megkönnyítsék azoknak a munkáját, akiknek a Long nem elég nagy, illetve a Float osztály nem elég pontos.

A BigDecimal osztály tetszőleges pontosságú előjeles valós szám ábrázolására alkalmas, amely már a Java 1.1 verziótól a java.math csomag része és amelyet a J2SE 5.0 verziótól valódi lebegőpontos műveletekkel is kiegészítettek. A típus egy tetszőlegesen nagy egész értékből és egy nem negatív 32-bites egészből, a skálázó faktorból áll.

```
//helyes megoldás:  
BigDecimal a = new BigDecimal("123.456");  
//a keletkezett változó értéke 123.456  
  
//helytelen megoldás:  
BigDecimal a = new BigDecimal(" 123.456");  
//a hiba típusa NumberFormatException
```

Az eltérő skálázó értékkel rendelkező számaink nem egyenlőek:

```
class Main {  
    public static void main(String[] args) {  
        BigDecimal a = new BigDecimal("1.23");  
        BigDecimal b = new BigDecimal("1.230");  
        System.out.println(a.equals(b)); //false
```

```
System.out.println(a.compareTo(b)); //0, stringként ←  
megegyeznek
```

A BigInteger osztály egy érdekes használata a számelmélet egy speciális területe: a nagy értékű kongruenciák vizsgálata, a hozzákapcsolódó moduló műveletek, valamint a nagy prímszámok előállítása, illetve megbízható prímtesztek. A jelentőséget az adja ennek a matematikai területnek, hogy különös módon összefonódott a kriptografiával a biztonságos üzenetküldés problematikájával.

A nyilvános kulcsú rejtjelezés alapötlete, hogy a kódolás, titkosítás folyamatát elválasztja a dekódolástól, és olyan algoritmust használ, ahol a kódoláshoz használt paraméter nem azonos a dekódoláshoz használt paraméterrel, és a kódoláshoz használt paraméterből nem határozható meg a dekódoláshoz szükséges paraméter. A nyilvános kulcsú rejtjelezés legelterjedtebb módszere az 1978-ban publikált RSA algoritmus, amely nevét alkotóinak kezdőbetűjéből kapta.

Az RSA (Ronald L. Rivest, Adi Shamir, Leon Adleman) aszimmetrikus algoritmus egy matematikai tételel (Fermat-tétel) alapszik, amely kimondja, hogy ha p prímszám és nem osztója egy a egész számnak, akkor $a^{(p-1)-1}$ osztható a p számmal. Vegyük először két véletlenszerű nagy prímszámot, $p-t$ és $q-t$, $n = pq$. Amennyiben p értéke 5 és q értéke 7, úgy n értéke 35 lesz értelemszerűen. Ezután vegyük egy olyan kis páratlan e számot, amely relatív prím $f(n) = (p-1)(q-1)$ -hez.

Amennyiben maradunk a korábbi példaértékeknél, úgy 24-et kapunk $f(n)$ -re, e értéke pedig 7 lesz (relatív prím 24-hez). Keressünk egy olyan d számot, amely $ed = 1 \bmod f(n)$. Mivel e értéke 7, ezért $e*d = 7*31 = 1 \bmod 24$ lesz. Az RSA nyilvános kulcs a $P = (e,n)$ pár lesz ($P = (7,35)$), az RSA titkos kulcs az $S = (d,n)$ pár lesz ($S = (31,35)$).

Az RSA titkosításra vonatkozó ellenőrzési eljárások közül talán a legismertebb a Solovay-Strassen teszt, illetve a Miller-Rabin teszt. Utóbbi a következő tényeken alapszik: legyen n egy páratlan prím, és $n-1 = 2^r * r$, ahol r egy páratlan szám. Legyen a egy tetszőleges egész úgy, hogy $\gcd(a,n) = 1$. Ekkor minden az $a^r \equiv 1 \pmod{n}$, minden az $a^{2^r} \equiv -1 \pmod{n}$ igaz, bármely j értékre.

A Miller-Rabin prímteszt pedig a következő: MILLER-RABIN(n,t). A bemenő érték egy páratlan 3-nál nagyobb vagy egyenlő egész és egy t egynél nagyobb biztonsági paraméter. A kimenő paraméter egy válasz, "prím" vagy "összetett" az n számunk. Az RSA összefoglalója után következhet a "Hibásan implementált RSA törése".

Az RSA összefoglalója után következhet az rsa algoritmus:

- i. Választani kell tetszőleges nagy prímet, **p-t** és **q-t**.
- ii. Ki kell számolni az **N = p*q-t**, melynél **N** jelöli a modulusát a nyílt és a titkos kulcsnak.
- iii. Euler-féle **f** függvényérték kiszámítása, melynek képlete a következő: **f(N) = (p-1)(q-1)**
- iv. Kell találni egy olyan **e** számot, amely 1-nél nagyobb és **f(N)**-nél kisebb. Emellett fontos, hogy **e**nek és **f(N)**-nek a legnagyobb közös osztólya 1 legyen. Az **e** lesz a nyilvános kucs kitevője, ezért nyilvánossá tesszük.
- v. A privát kulcshoz is kell találni egy **d** kitevőt, mely esetén teljesül, hogy **d*e f(N)** osztva 1 maradékot ad. Mivel ez a titkos kulcsnak a kitevője, ezért ezt is titkosítjuk.

Az Rsa fordítása után megkapjuk a hibás implementáció alapján létrejövő kulcsokat.

```
donat@donat-VirtualBox:~/Letöltések$ javac RSA.java
donat@donat-VirtualBox:~/Letöltések$ java RSA
Méret bitekben:
2019
p_i
4592788309323904878162894561476599491548330742108949861166681019368784575647030518819263328757585473319169818317566809017080689539664
8121060772985498777944142699173657966199353940968876983665724858284537818313193384123111416682739709980854688334498325998253479629
5462177633045288790372098260103481315308305292815959479147864681779127012745070815015923394847719802168771907026268726914521404399871
00652760466563208899882286750429943455939294953814445288720757025692628960552680243577987445539615251051019200562969274434365932664
3158776809565861975467454856184777641976486173495909895980565396654347876133
p_i hexa
61a9720834e58997e7f1a8619c9fea4b1319f1c85ab648ff79ce677264a04492ccfd8f46ce1b719355154831b0178fd3dc4d577c335338db08f0952314b0137ae77
0a5814db6fc31fc88bf61ffc2b46c880e68ce5b098a86582cf58ee0c84ef5e12178c967ba5b120c87ae1e54a015882004921efdf4c1ff30273746f99baefcb3ef1814
7888885840cd9e4b40a48a049d829f9deb54873190d40c9ae80ecf88254bbf2be004e3da257b3d641e30d60fd3eb4854a8e1b50c2b5b779f53c80cb88920ddf6c1ab
f60ce69bffd9bc085a7e99a6ae82ad044528649383072fac8583ce2b0a5a6387ea0f5cccd2b0a9fc7c375a1385b83e11794c97725
q_i
5622954297684830468681738712435456285669109962403004214522831698385668244697103761214909487113935955065789894406120878773829016472661
964665408833955366271969398054071195543891632763819488332610138684069122584388661664779964865412954351068519025842067943223382325807
555253588843318314897404824125672964862318138140146434524375299777526285170356952817396202051108045064070596656142375697430002773166
135550108286965098827475351152359316238661131480728989688753227367862618215578847582822184028380257668860014547468817923793408985158
7842900102822523873953022245643117480728022538579465878611474004938015200003
m_i
258250387622694974699156095701996488792123634878120525806252334051613215520918349960203228897747447939465916143289394317707399594916
1411251070536704179655918388454491656700310977407175594015868804867197004273794957461856831388778626306010543258392099042036247891245
049051192391896499330359376388558941633690917650181290317232842292343034891222766354760715303406630146051633182966153813007285984409
895712932397938974838269311782548905023775712864344357415853728251423100352046922947367391350816854162956113304436647761216512985425
6391223190178597288266551536058430647821387424542343963090817183643119517591760060196577498861055589693012057942816599760232184666359
75459065284320305706618017772570344108719416424021731018004515100130987998324453576724289168064026532796940564576353485575949090955
3465573118228868436746864583661628510299920473182317489019973938423546343152096419860191016359678855339858549734658600399238299769133
62344792891402883819929684167095066003688777614639587848773184109349143295693447858953899557793485715208283329087393472137888660363
7435403341980315999802343066384400340142613963934254143784158983705360286303359916024398723780513146747838826083937614817160336594569
0696622434265228399
z_i
```

Az RSA titkosításhoz egy titkos és egy nyilvános kulcsot használunk. Először létrehozzuk a modulust, a kitevőt és annak inverzét a KulcsPar osztályban. A p és a q lesz a két nagy prímszámunk, melyeket összeszorozva kapjuk meg a modulus értékét.

Visszafejtéskor ismét igénybe vesszük egy függvényt, ezúttal viszont az első paraméter a d lesz, így újra ASCII kód tölthető a bufferbe ezt betöltve tiszta szöveget kapunk.

```
donat@donat-VirtualBox:~/Letöltések$ javac RSA2.java
donat@donat-VirtualBox:~/Letöltések$ java RSA2
Utra kelunk. Megyunk az Oszbe, Vijjogva, sirva, kergetözve, Ket lankadt szarnyu heja-madar.
donat@donat-VirtualBox:~/Letöltések$ █
```

16.4. Változó argumentumszámú ctor

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

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Ctor

Tanulság, tapasztalatok... A kétdimenziós halmazt png ábráját fogjuk először létrehozni a mandel.cpp file segítségével. hogy látjuk, a mandel.cpp sikeres fordításához és futtatásához szükségünk lesz a libpng, libpng++ könyvtárakra és a png++/png.hpp fájlra. A könyvtárakat a sudo parancs segítségével tudjuk telepíteni, viszont a headerhez le kell töltenünk a png++ hivatalos oldaláról a becsomagolt telepítő állományokat: png++-0.2.9.tar.gz

```
//A terminálba leadott parancsok:
$ sudo apt-get install libpng-dev
$ sudo apt-get install libpng++-dev
$ tar -zxf png++-0.2.9.tar.gz -C ~/Letöltések
$ cd ~/Letöltések/png++-0.2.9
$ make
```

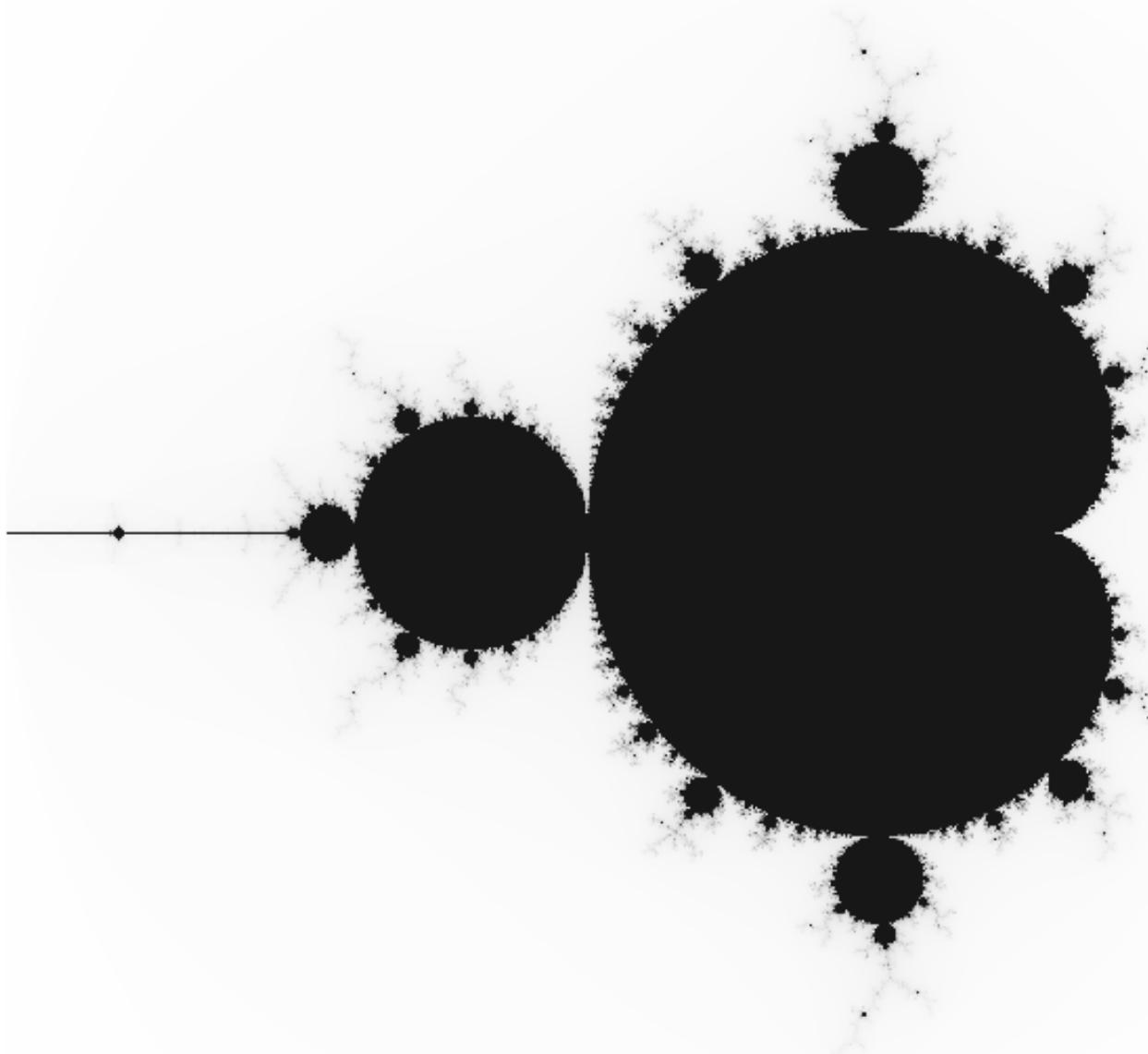
Ha minden jól megy, mostmár tudjuk fordítani és futtatni a png elkészítéséhez szükséges programot.

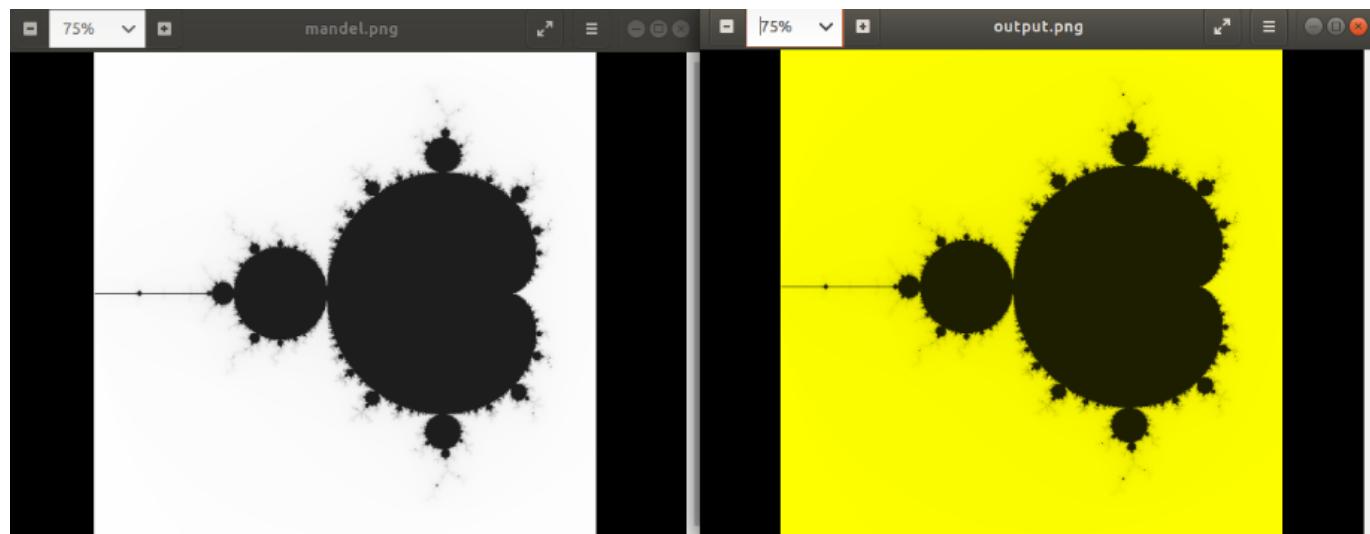
```
$ g++ mandel.cpp `libpng-config --ldflags` -o mandel
$ ./mandel mandel.png
Szamitas.....mandel.png mentve
```

A perceptron az egyik legegyszerűbb előrecsatolt neurális hálózat. A main.cpp segítségével fogjuk szimulálni a hiba-visszaterjesztéses módszert, mely a többrétegű perceptronok egyik legfőbb tanítási módszere. Ahhoz, hogy ezt fordítani és futtatni tudjuk később, szükségünk lesz az mlp.hpp fájlra, mely már tartalmazza a Perceptron osztályt.

Az előző program futtatásával létrejött Mandelbrot png ábrát fogjuk beimportálni. A fájloknak köszönhetően megadhatjuk a neuronok darabszámát. A mandel.png alapján új képet állítunk elő. A visszakapott értékeket megfeleltetjük a blue értékeknek.

Az első felvonás Perceptron feladatához képest módosításokat kell végeznünk a header file-on is, ugyanis új képet akarunk előállítani. Az operátor már egy tömböt térít vissza, melynek segítségével bele tudunk nyúlni a képhez.





16.5. Összefoglaló

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

Megoldás forrása:

Tanulság, tapasztalatok... Hibásan implementált RSA törése feladtot választottam.

17. fejezet

Helló, Gödel!

17.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világajánokságban <https://youtu.be/DL6iQwPx1Yw> (8:05-től)

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/myshmcl

Tanulság, tapasztalatok...

Az OOCWC egy párhuzamos platform, melynek célja volt a forgalomirányítási algoritmusok kutatása és a robotautók terjedésének vizsgálata. A Robocar City Emulator lehetővé tette volna fejlesztők számára új modellek és elméletek tesztelését. A debreceni Justine prototípus gengsztereinek rendezése volt ezúttal a feladat: ehhez először le kell rántanunk a [GitHub projektet](#).

A C++11-ben megjelenő lambda kifejezések lehetővé teszik, hogy egy vagy többsoros névtelen függvényeket definiálunk a forráskódban. Szerkezetük nem kötött.

Általános formája:

```
[] (int x, int y) { return x + y; }
```

A szögletes zárójelpár jelzi, hogy lambda kifejezés következik. A kerek zárójelpár a függvényhívást jelent. Utána jön a függvény törzse, mely return utasításából a fordító meghatározza a függvény értékét és típusát.

A myshmclient.cpp-ben a gangsters vektort rendezzük lambda kifejezéssel. Ha x gengszer közelebb van az elkapott cop objektumhoz, mint y gengszer, akkor igaz értéket ad vissza. Rendezés után így a vektor elejére a rendőrhöz legközelebb álló gengszterek kerülnek.

```
std::sort (gangsters.begin(), gangsters.end(), [this, cop] (←
    Gangster x, Gangster y)
{
    return dst (cop, x.to) < dst (cop, y.to);
});

//void sort (RandomAccessIterator first, RandomAccessIterator ←
//last, Compare comp);
```

17.2. C++11 Custom Allocator

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

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/fenykard

Tanulság, tapasztalatok... C++-ban az allokátorok szerepe, mint ahogyan a nevük is mutatja, hogy memóriát allokáljanak az adatszerkezeteink számára. Az allokátornak van alapértelmezett értéke de ezt felül írhatjuk. Az allokátorok mögötti ötlet véleményem szerint a kiss alapelvein nyugszik. A cél, hogy egy osztály minél kevesebb dologról feleljön. A memória foglalás mindenkor veszélyes vizeken nyugszik.

A CustomAlloc osztály végzi az allokáló munkát, valamint végez némi követést. minden alkalommal, mikor foglalunk, láthatjuk a sztender kimenetén is.

A kód nem olyan nehéz megemészteni, amit nem érhetünk benne, az például a `abi::__cxa_demangle(t 0, 0, &s);` kifejezés.

Ez a demangle függvény azt tudja, hogy egy megcsónkított azonosítót megpróbál kibogozni és megmondani az eredeti formáját. Fontos, hogy mivel ez a függvény egy char pointert ad vissza, a neki foglalt memóriát kézileg kell felszabadítani a `free()` használatával.

Tehát a cél az, hogy generalizáljuk a kódot, és ezáltal minél kevesebbet kelljen írni. Ha nem használnánk ezt a templatet, akkor meg kéne írjuk ezt az allokátort a létező összes típusra.

A program kimenete a következőképp néz ki:

```
g++ --std=c++17 CustomAlloc.cc && ./a.out
    Allocating 1 object(s) of 4 bytes. i=int
    Allocating 1 object(s) of 8 bytes. l=long
    Allocating 1 object(s) of 32 bytes. ←
        NST7__cxx11basic_stringIcSt11char_traitsIcESaIcEEE=std:: ←
        __cxx11::basic_string<char, std::char_traits<char>, std:: ←
        allocator<char> >
```

17.3. STL map érték szerinti rendezése

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

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/fenykard

Tanulság, tapasztalatok...

Ebben a példában megismérjük a C++ STL map adatszerkezetét. Ez az adatszerkezet úgynevezett párok tárolására képes, általában a párok első elemét hívjuk kulcsnak, a másodikat pedig értéknek. Új értékeket az `insert()` metódus meghívásával lehet hozzáadni, mely paramétereként elfogad egy `std::pair` típusú objektumot.

A mapoknak két tagjuk van (most string és int), ezeket rendeljük hozzá egy pair vektorhoz, és ezt követően rendezünk. Érték szerint csökkenő sorrendben történik a rendezés (p1.second és p2.second használatával). Az előző feladathoz hasonlóan lambda kifejezés segítségével rendezünk: a nagyobb elemek lesznek elől, ugyanis ha az első pair érték nagyobb, akkor igazzal tér vissza a függvény.

```
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;
}
```

17.4. Alternatív Tabella rendezése

Mutassuk be a https://progpter.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang

Interface Comparable<T> szerepét!

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Alternativ
Tanulság, tapasztalatok...

Az Alternatív Tabella a magyar labarúgó bajnoság csapatainak alternatív rangsorát tartalmazza. Különlegessége abban rejlik, hogy nem csak azt vizsgálja, hogy nyert, döntetlen vagy vereséget ért el egy csapat, hanem azt is, hogy ki ellen éri el az eredményt. Tehát célja, hogy hitelesebb képet kapjunk a csapatok valós teljesítményéről.

A feladat célja, hogy megismerkedjünk a csapatok közti összehasonlítás mechanizmusát. Az csapatok kezeléséhez létrehozunk egy Csapat osztályt, mely implementálja a Comparable interfést. Lássuk az osztályt.

```
class Csapat implements Comparable<Csapat> {
    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
```

```
    }

    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Tehát egy `Csapat` típusú objektum rendelkezik egy névvel és egy értékkel. Mivel implementáltuk a `Comparable` interfészét, ezért definiálnunk kell a `compareTo` függvényt. Ennek segítségével tudjuk meghatározni, hogyan legyen összehasonlítva a két objektum.

Most hogy láttuk, hogyan mi a feladata a `Comparable` interfésznek. Nézzük meg az AlternatívTabellát.

Az AlternatívTabella a Google algoritmusán, a PageRank-en alapszik. A PageRank ötlete, hogy azok a weblapok jobb minőségűek, amelyekre jobb minőségű lapok mutatnak. Jelen esetben az foglal előkelőbb helyet a labdarúgó bajnokságon, aki előkelőbb helyen lévő csapatuktól szerez pontot.

A táblázatban lévő eredményeket a kereszt nevű kétdimenziós tömbbe dobuk bele a Wiki2Matrix osztályban. Vegyük például a 2010-11-es magyar labdarúgó bajnokság első osztályának eredményeit. Az üres legyen 0, a zöld 1, a sárga 2, a piros pedig 3. A példa alapján a mátrixba tehát a következő értékek kerülnek:

```
int[][] kereszt = {
    {0, 0, 0, 1, 0, 3, 2, 3, 3, 2, 0, 0, 0, 0, 2, 2, 3},
    {3, 0, 2, 1, 3, 2, 0, 3, 3, 3, 0, 0, 0, 0, 0, 0, 1},
    {1, 1, 0, 0, 3, 1, 3, 0, 0, 0, 3, 1, 1, 0, 2, 0},
    {0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 2, 1, 1},
    {3, 3, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {1, 0, 3, 1, 0, 0, 0, 1, 3, 2, 0, 0, 0, 1, 2, 3},
    {0, 2, 0, 0, 1, 0, 1, 1, 0, 3, 0, 1, 3, 3, 1},
    {0, 0, 1, 1, 3, 0, 0, 0, 1, 3, 1, 1, 1, 3, 0},
    {0, 0, 1, 2, 3, 0, 0, 1, 0, 0, 0, 2, 1, 1, 3, 1},
    {0, 1, 1, 2, 0, 0, 3, 1, 1, 0, 0, 0, 1, 3, 3},
    {2, 3, 0, 2, 1, 1, 0, 0, 1, 2, 0, 1, 0, 0, 0, 2},
    {3, 3, 0, 0, 0, 3, 3, 0, 2, 1, 1, 0, 2, 0, 0, 0},
    {1, 2, 0, 1, 0, 2, 1, 0, 0, 1, 3, 1, 0, 0, 0, 2},
    {2, 1, 2, 0, 1, 2, 1, 0, 0, 0, 3, 1, 1, 0, 0, 0},
    {1, 3, 1, 0, 2, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 1, 0, 1, 0, 3, 1, 1, 0, 0, 1, 2, 1, 3, 0}
};
```

A fordítás és futtatás után kapott új értékeket az AlternatívTabella osztályba kell belehelyezni, a `double[][] Lnk = {}` rész kapcsos zárójelei közé. Az eredetinek tömbbe kerülnek a csapatok nevei a táblázatban való megjelenésük alapján. Az eredeti pont tömbbe az eredeti tabella pontjait írjuk be. Készítünk egy újnev tömböt is, melybe a Wiki2Matrix kereszt nevű mátrixa alapján létrejött sorrend szerint kerülnek be a csapatok.

```
double[][] Lnk = {
    { 0.0, 0.0833333333333333, 0.0, 0.1111111111111111,
        0.14285714285714285, 0.0, 0.1111111111111111,
        ...,
        0.15384615384615385, 0.0833333333333333, 0.0, 0.0, } ←
    };

String[] eredetinev = { "Videoton", "Ferencvaros", "Paks",
    "Debreceni VSC", "Zalaegerszegi TE", "Kaposvari Rakoczi",
    "Lombard Papa", "Kecskemeti TE", "Ujpest", "Gyori ETO",
    "Budapest Honved", "MTK Budapest", "Vasas",
    "Szombathelyi Haladas", "BFC Siofok", "Szolnoki MAV" };

int[] eredetipont = { 40, 34, 31, 31, 30, 29, 27, 24, 23, 23, ←
    22, 22,
    21, 20, 18, 9 };

String[] ujnev = { "BFC Siofok", "Budapest Honved", "Vasas",
    "Debreceni VSC", "Ferencvaros", "Gyori ETO",
    "Kaposvari Rakoczi", "Kecskemeti TE", "Lombard Papa",
    "MTK Budapest", "Paksi FC", "Szolnoki MAV FC",
    "Szombathelyi Haladas", "Ujpest", "Videoton",
    "Zalaegerszegi TE" };
```

Az alternatív tabella értékeinek összehasonlításához, rendezéséhez igénybe vesszük a java.lang package Comparable interfészét. Az interface lehetővé teszi, hogy képések legyünk alkalmazni a listákra és tömbökre definiált függvényeket. A compareTo metódus a paraméterként megadott objektumot hasonlíta össze az aktuális objektummal. Háromféle értéket adhat vissza: pozitív, negatív számot vagy nullát.

```
class Csapat implements Comparable<Csapat> {

    private String nev = "Nincs";
    private double ertek = '0';

    public Csapat(String nev, double ertek) {
        this.setNev(nev);
        this.setErtek(ertek);
    }

    public int compareTo(Csapat a) {
        if (this.getErtek() < a.getErtek()) {
            return 1;
        } else if (this.getErtek() > a.getErtek()) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

17.5. Prolog családfa

Ágyazd be a Prolog családfa programot C++ vagy Java programba! Lásd para_prog_guide.pdf!

Megoldás forrása: https://github.com/fupn26/BHAX/tree/master/attention_raising/Source/PrologFamily

Tanulság, tapasztalatok...

A Prolog egy olyan programozási nyelv, melynek segítségével matematikai logikai formulákat tudunk vizsgálni. A matematikai logikáról ez a feladat nem fog részletesebb tájékoztatást nyújtani, a fentebb lévő pdf-ben találhattok példákat a mélyebb megértéshez. Annyit fontos megemlíteni, hogy a program az előrendű logikán alapszik. Tehát itt az atomi formulák mellett megjelennek a függvényszimbólumok és a kvantorok, mint a logikai formulák építőelemei.

SWI-Prolog használata:

```
sudo apt install swi-prolog-jav
```

Elsőnek létre kell hozni egy új osztályt.

A programunk Java programban íródott, mely az SWI-Prolog könyvtárat használja. Alapvetően a program úgy működik, hogy a elsőnek betöljük a Prolog fájlt, majd arról készítünk lekérdezéseket.

A family.pl fájl tartalmazza a pdf-ben megadott Prolog programot.

```
férfi(nándi).  
férfi(matyi).  
férfi(norbi).  
férfi(dodi).  
férfi(joska).  
nő(gréta).  
nő(erika).  
nő(kitti).  
nő(marica).  
gyereke(nándi, norbi).  
gyereke(matyi, norbi).  
gyereke(gréta, norbi).  
gyereke(nándi, erika).  
gyereke(matyi, erika).  
gyereke(gréta, erika).  
gyereke(norbi, dodi).  
gyereke(norbi, kitti).  
gyereke(erika, joska).  
gyereke(erika, marica).  
  
    . . .
```

Ezt a programunkba a következő módon fogjuk beolvasni.

```
String s = "consult('family.pl');  
Query q = new Query(s);  
System.out.println(q.hasSolution());
```

Itt jól látható, hogy hogyan fog működni maga az alkalmazás. Elsőnek megadunk egy stringet, mely kiértékelni kívánt formulát tartalmazza. Majd létrehozunk egy Query osztályú objektumot hozunk létre. Ez az osztály teszi lehetővé, hogy kiértékeljük a formula igazságértékét, vagy az egyes változók lehetséges értékeit. Ha csak az igazságértékre vagyunk kíváncsiak, akkor a hasSolution() függvényt kell használni. Ha itt hamis eredményt ad, akkor a fájt nem találja.

```
String t2 = "apa(gréta)";  
System.out.println(t2 + " is " + (Query.hasSolution(t2) ? " ↵  
provable" : "not provable"));
```

Az első példa tehát azt mutatja be, hogy hogyan lehet megtudni a formula igazságértékét. Jelen esetben az vizsgáljuk, hogy Gréta apának tekinthető-e. Nyilván nem, vagyis a terminálban a "not provable" kifejezés jelenik meg.

```
String t3 = "nagyapja(X, matyi)";  
System.out.println("each solution of " + t3);  
Query q3 = new Query(t3);  
while (q3.hasMoreSolutions()) {  
    Map<String, Term> s3 = q3.nextSolution();  
    System.out.println("X = " + s3.get("X"));  
}
```

A változók értékének lehetséges értékeit kétféleképpen lehet felsorolni. A fenti példában azokat az X-eket keressük, akik Matyi nagyapjai lehetnek. Ezt úgy tudjuk megfogalmazni matematikai logikával, hogy azokat az X-eket keressük, akik Matyi apjának vagy anyjának az apja. Tehát végig iterálunk a lehetséges X-eken.

18. fejezet

Helló, !

18.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>
Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Future
Tanulság, tapasztalatok...

A future projekt eredeti célja egy város alternatív jövőinek legenerálása, a legenerált jövők elemzése volt.
A future6-hoz készült egy tulajdonság editor, melyben a hallgatók az adott nap tevékenységeit tudták feljegyezni.

Az a fájl futtatásához a következőt kell megtennünk a terminálban:

```
$ javac ActivityEditor.java  
$ java ActivityEditor --city=Debrecen --props=me.props,gaming.props ↵  
,programming.props
```

Az editor megjelenéséhez feldobható egy CSS stíluslap . Ahhoz, hogy a stíluslapban lévő tulajdonságok és értékek megjelennek az editoron. Ki kell bővítenünk a kódot:

```
public class ActivityEditor extends javafx.application.Application ↵  
{  
    private static String path;  
    public static void main(String[] args) {  
  
        path="style.css";  
        javafx.application.Application.launch(args);  
    }  
}
```

Az editor dobozainak fő háttérszíne világoszöld lesz, melyhez jól passzol az, ha a tulajdonságok szövege erős, sötétkék színt kap. Ha rákattintunk egy dobozra, akkor annak szegélye is sötétkékké válik, de ennek hex kódja némi képp különbözik a korábban említett text fill értéktől. A címek kapnak egy halványszürke hátteret, így egy kicsit jobban kiemelkednek a címekben olvasható szövegek.

```
.tree-cell{
    -fx-background-color: #8ed49d;
    -fx-text-fill: #2200ff;
}

.vbox{
    -fx-background-color: #2f00ff;
}

.label{
    -fx-text-fill: #2f00ff;
    -fx-background-color: #eee9e1;
}

.content{
    -fx-background-color: #8ed49d;
}
```



18.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocaremulator/blob/master/justine/robo.h>

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/Carlexer.cpp

Tanulság, tapasztalatok...

Az OOCWC (RoboCar World Championship) egy páréves platform, melynek célja volt a forgalomirányítási algoritmusok kutatása és a robotautók terjedésének vizsgálata. A RoboCar City Emulator lehetővé tette volna fejlesztők számára új modellek és elméletek tesztelését. A debreceni Justine prototípus része a CarLexer, melynek sscanf függvényét kell feldolgoznunk.

Az sscanf függvényt a bemenet feldolgozására használjuk. A yytext char* tartalmazza az éppen feldolgozásra váró sztringet.

Az sscanf használata egyszerű. Meg kell adnunk neki a vizsgálandó szöveget, majd egy sztringet, amiben különleges jelentéssel bíró karaktereket helyezhetünk el, ezekről tájékozódhatunk a man scanf parancs kiadásával. Ezek a különleges jelentéssel bíró karakterek fogják megadni, hogy pontosan milyen jellegű bemenetet is várunk. Ezután meg kell adni annyi darab referenciait, ahány ilyen különleges jelentéssel bíró karaktert használtunk a szabályok leírása során. A különleges jelentéssel bíró karaktereket % jel után kell megadnunk.

A következő sor:

```
std::sscanf(yytext, "betu: %c, szam: %d", &c, &i);
```

Jelentése, hogy vizsgáljuk meg a yytext sztringet, és amennyiben betu: BETŰ, szam: SZÁM alakú, a BETŰ-t tegyük be a c változóban, a SZÁM-ot pedig az i változóba.

18.3. SamuCam

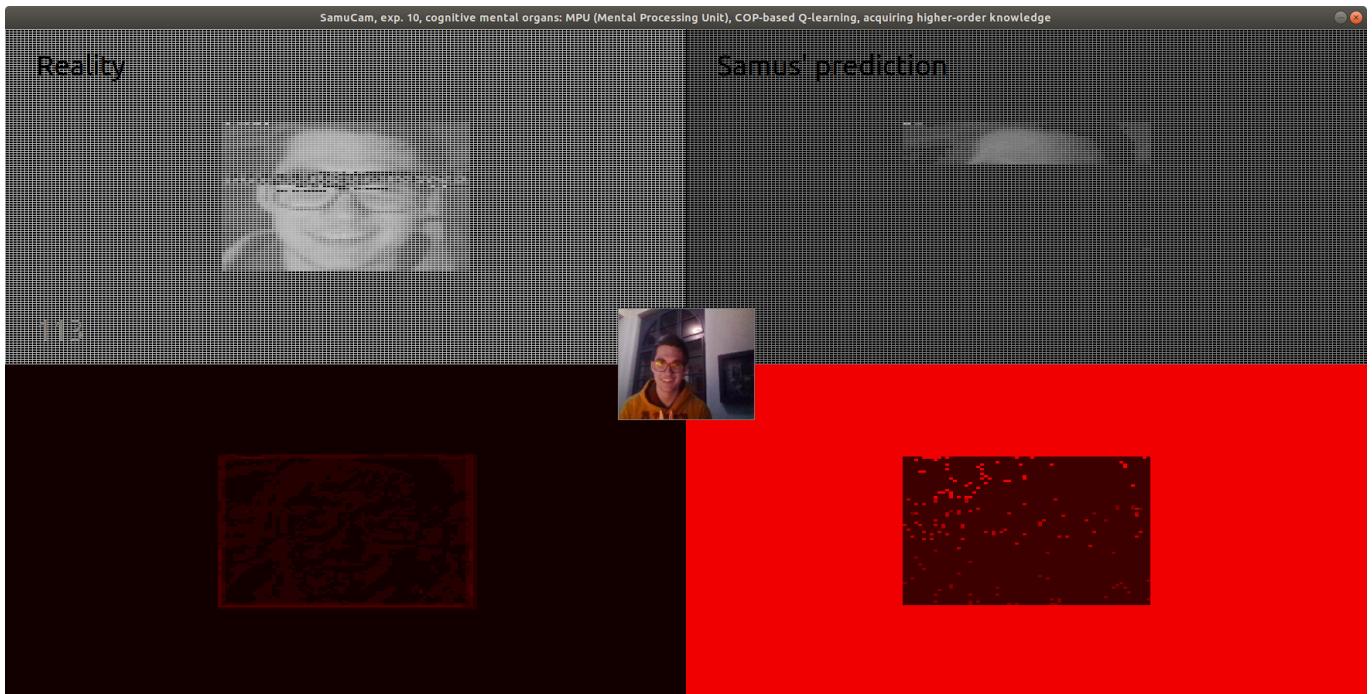
Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>

Megoldás forrása:

Tanulság, tapasztalatok...

A SamuCam.cpp-ben látható, hogy a videoCapture open függvénye nyitja meg a VideoStream-et. Mivel alapból telefonos használatra íródott a program, ezért a VideoStream helyett 0-t kell írnunk, ha webkamerával szeretnénk dolgozni. Ezután beállítjuk a kamerakép szélességét, magasságát és FPS értékét. A CascadeClassifier alapvetően tárgyak felismerését segíti, jelen esetben pedig ez teszi lehetővé a kameraképen látható arcok feldolgozását. A helyes működés érdekében le fogunk rántani a GitHub-ról egy frontal face XML-t, melyben az értékek a kamerával szemben elhelyezett arcképek felismerését biztosítja. Képkockánként kerül beolvasásra a kamerakép a read függvényben. Ha kap egy képkockát, akkor első lépésként átméri, majd szürkére átszínezi. Az equalizeHist függvény kiegyenlíti a szürke képkocka hisztogramját. A detectMultiScale függvény segítségével történik a képkockán lévő arcok keresése. Ha talált egy arcot, akkor az alapján létrehozunk egy QImage-t. A faceChanged egy signal, bekövetkezte után az arc köré rajzolunk egy keretet, és egy újabb QImage-t készítünk. Ha pedig a webcamChanged signal bekövetkezett, akkor 80 ms-t követően következhet egy újabb képkocka beolvasása. A SamuCam futtatásához

először le kell szednünk a GitHub projektet, majd pedig a korábban említett XML fájlt. A Qt keretrendszer segítségével létrehozzuk a Makefile-t, ezután pedig indíthatjuk is a programot.



18.4. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esporttalent-search>

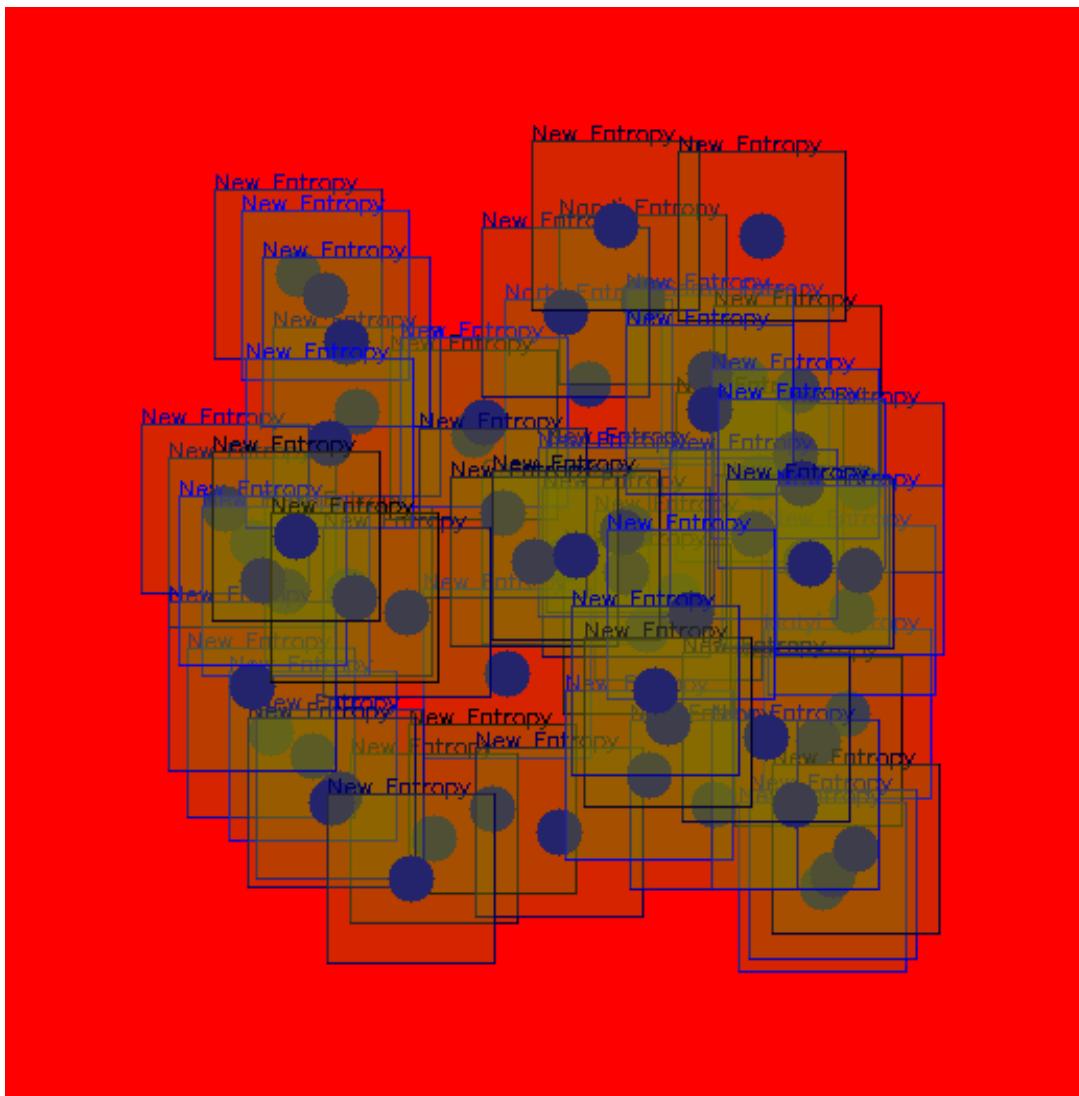
Megoldás forrása:

Tanulság, tapasztalatok...

BrainB Benchmark szoftver a jövő kiemelkedő e-sportolóinak korai felismerésében, felkutatásában hivatott segíteni. A mérőprogram arra a játékélményre, jelenségre épül, mely során a játékos átmenetileg elveszíti a karakterét az intenzív, komplex pillanatokban. A vizuális effektek kitakarják egymást, így idővel az egyszerű szoftverekben is nehézkessé válhat a karakterünk irányítása.

Ebben a programban a 'Samu Entropy' karaktert azaz dobozt kell figyelnünk: akkor eredményes a mérés, ha sikerül a doboz közepén lévő kék pöttyön tartanunk a kurzort. Az értékek növekedésével egyre több doboz jelenik meg a képernyőn, a dobozok pedig egyre gyorsabban kezdenek el mozogni.

Ha a játékos a mérőprogram használata során egy adott pillanattól kezdve 1200 ms-on több mint 1 másodpercen keresztül a karakteren tudja tartani az egérmutatót, akkor a játékos a benchmark szerint megtalálta a karaktert. Ha több, mint 1 másodpercig nincs kapcsolata akkor a játékos elvesztette a karaktert.

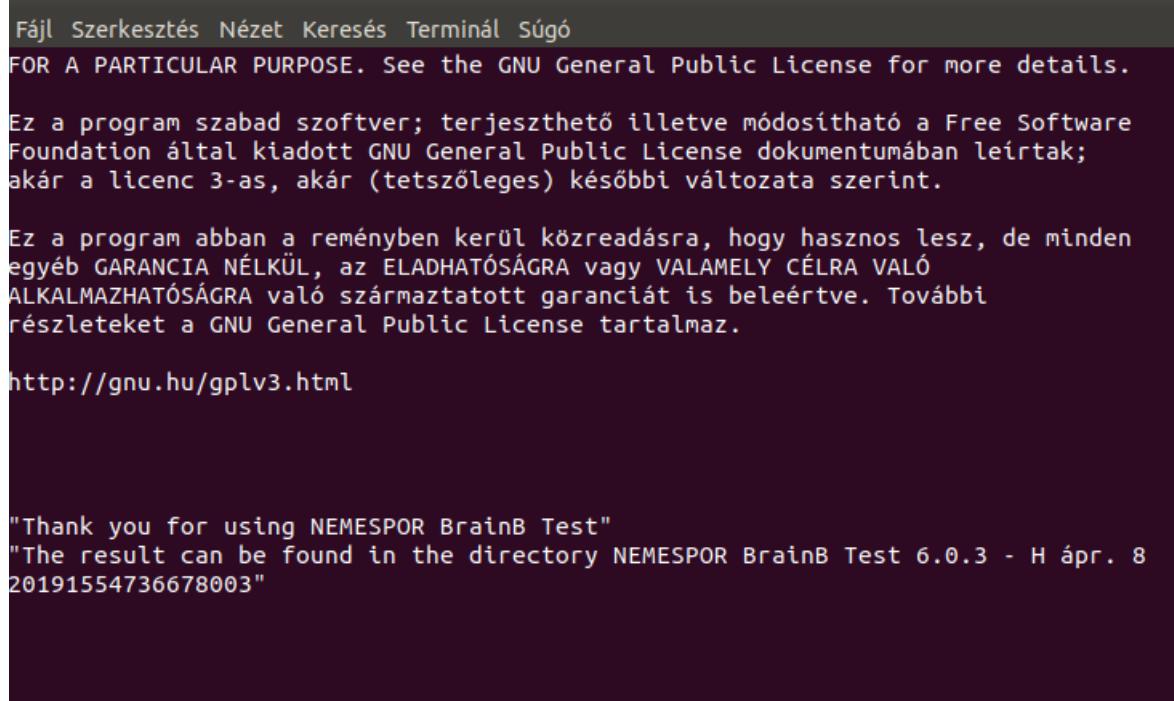


A helyes fordításhoz és futtatáshoz szükségünk lesz az OpenCV könyvtárra, illetve a Qt keretrendszerre. Fontos, hogy a forrásfájlok azonos mappában szerepeljenek!

```
$ sudo apt-get install opencv-data  
$ sudo apt-get install libopencv-dev  
$ qmake BrainB.pro  
$ make  
$ ./BrainB
```



A benchmark 10 percig figyeli a teljesítményünket, ezután pedig a statisztikákat megkapjuk egy txt fájl formájában.



```
$ more Test-6000-stats.txt
NEMESPOR BrainB Test 6.0.3
time      : 6000
bps       : 33890
noc       : 53
nop       : 0
lost      :
46270 54570 62080 40990 56610 43630 30630 26480 15440 22740 ←
    20190 47680 29930 28400 47280 43750 44030 37380 24640
mean      : 38037
var       : 13210.3
found     :
8420 19300 25390 30100 33490 33470 48620 53970 ←
    47480 73410 17420 27980 27930 34140 47550 35060 23870 40610 ←
    42360 41260 45220 55600 29580 36080 19290 35660 10200 10030 ←
    15930 30610 23230 20520 16110 14870 20060 30940 9760 22740 ←
    30520 20900 23000 22140 27110 21300 29510 36750 34340 31890 ←
    28420 22350 28010 33270 42420 20120 44620 39880 29150 29670 ←
    45280 11100 10010 19630 18100 23930 33240 15700 31540
mean      : 29285
var       : 12497.9
lost2found: 30100 17420 23870 29580 15930 30610 22740 27110 ←
    34340 28420 44620 10010 18100
mean      : 25603
var       : 9039.09
found2lost: 54570 56610 43630 30630 15440 20190 47680 28400 ←
    47280 44030 37380 24640
mean      : 37540
var       : 13543.7
mean(lost2found) < mean(found2lost)
time      : 10:0
```

U R about 3.85394 Kilobytes

19. fejezet

Helló, Lauda!

19.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! <https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/ch01.html#id527287>

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/KapuSzkenner.java

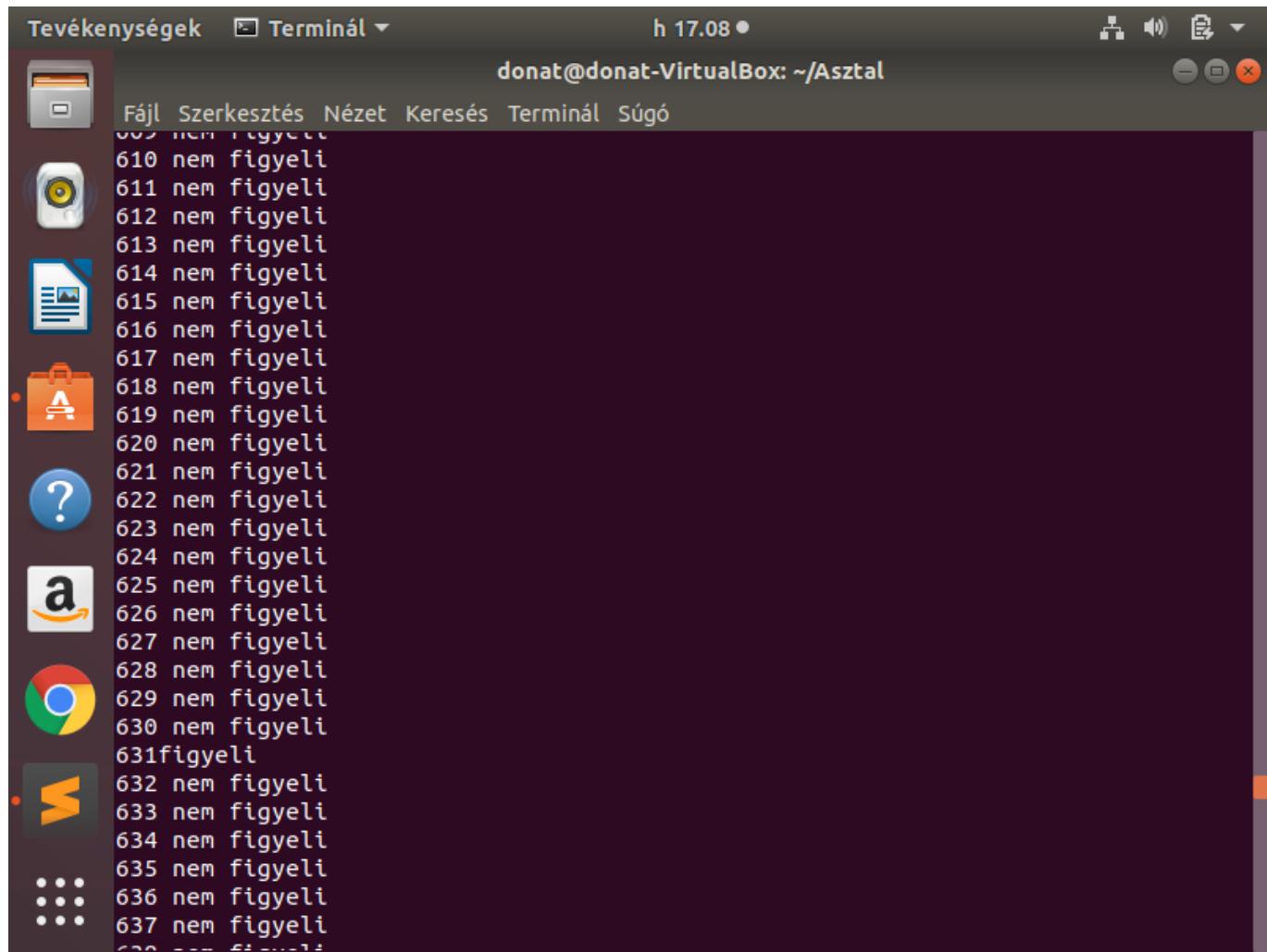
Tanulság, tapasztalatok... A KapuSzkenner.java segítségével megnézhetjük, hogy gépünk éppen milyen portokat figyel. A kivételkezelés try blokkjában az argumentumként megkapott gép 1024 alatti portjain próbálunk meg TCP kapcsolatot létesíteni. Ha az adott gépen egy szerverfolyamat figyeli az adott portot, akkor ezt ki is iratjuk.

```
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 (IOException e) {  
                System.out.println(i + " nem elérhető");  
            }  
    }  
}
```

Portszkennelést páldul akkor használunk, mikor szeretnénk meggyőződni arról, hogy egy általunk üzemeltetett szerver nem fed fel túl sokat (tehát nem rendelkezik feleslegesen nyitva levő portokkal).

Ezen program működési elve, hogy 0-tól 1024-ig végigmegy az összes porton, és megpróbál egy socketet nyitni a java.net.Socket osztály segítségével. Amennyiben elolvassuk az ehhez az osztályhoz tartozó dokumentációt, azt láthatjuk, hogy amennyiben nem tudunk csatlakozni, úgy SecurityException hibát fog dobni a meghívás.

Ennek köszönhetően egyszerűen, amennyiben nem tudunk socketet nyitni az adott porton, tudjuk, hogy nem elérhető.



```
donat@donat-VirtualBox: ~/Asztal
h 17.08 •
Fájl Szerkesztés Nézet Keresés Terminál Súgó
610 nem figyeli
611 nem figyeli
612 nem figyeli
613 nem figyeli
614 nem figyeli
615 nem figyeli
616 nem figyeli
617 nem figyeli
618 nem figyeli
619 nem figyeli
620 nem figyeli
621 nem figyeli
622 nem figyeli
623 nem figyeli
624 nem figyeli
625 nem figyeli
626 nem figyeli
627 nem figyeli
628 nem figyeli
629 nem figyeli
630 nem figyeli
631figyeli
632 nem figyeli
633 nem figyeli
634 nem figyeli
635 nem figyeli
636 nem figyeli
637 nem figyeli
638 nem figyeli
```

19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Tanulság, tapasztalatok...

A feladatban az átszövés orientált programozással fogunk megismerkedni, melyhez az AspectJ programozási nyelvet fogjuk használni. Az átszövés lényegében annyit tesz, hogy a régebbi programunk működését úgy tudjuk módosítani, hogy nem kell módosítani a forráskódon. Helyette írunk egy szöveget aspect-et, mellyel megváltoztathatjuk például egy függvény működését, vagy esetleg megadhatjuk, hogy a függvény végrehajtása előtt és után mit csináljon a program. Mi most az AspectJ nyelv around függvényével fogunk foglalkozni, melynek utasításait egy adott függvény helyett hajtjuk végre. Aspektusok felépítés: Az aspektusok önállóan nem használhatóak, minden kell egy programhoz, AspectJ esetén Java. A lényeg 3 fogalom amit szépen le írok. Az első a kapcsolódási pont ami az eredeti program egyik függvénye. Második vágási pont ami az aspektus része, és a csatlakozási pontokat jelöli. Harmadik tanács mely lényegében azt tartalmazza, hogy hogyan szeretnénk módosítani az eredeti program működését. Hogyan lehet megoldani azt, hogy a programunk a fát preorder módon járja be. Ezt már korábbi feladatokban megcsináltuk, viszont akkor a forráskódot közvetlenül módosítottuk. Lássuk is az Aspect .aj fájl tartalmát.

```
privileged aspect Aspect{
    void around(LZWBInFa fa, LZWBInFa.Csomopont elem, java.io.BufferedReader os):
        call(public void LZWBInFa.kiir(LZWBInFa.Csomopont, java.io.BufferedReader))
            && target(fa) && args(elem, os){
    if (elem != null)
    {
        try{
            ++fa.melyseg;
            for (int i = 0; i < fa.melyseg; ++i)
                os.write(" --- ");
            os.write(elem.getBetu () + " (" + (fa.melyseg - 1) + ") \n");
            fa.kiir(elem.egyesGyermek (), os);
            fa.kiir(elem.nullasGyermek (), os);
            --fa.melyseg;
        }
        catch(java.io.IOException e){
            System.out.println("Csomópont írása nem sikerült.");
        }
    }
}
}
```

Az első kulcszsó privileged emiatt az aspektusunk hozzá tudnak férni az osztályok privát tagjaihoz. Az aspect szó jelöli, hogy most aspektust írunk, nem pedig hagyomásnyos osztályt. Az aspektusunk egy függvényből áll, ez a around. Ahhoz, hogy a bejárás módját módosítani tudjuk először a kiir (Csomopont, BufferedReader) függvényre van szükségünk. Mivel ez a függvény olyan tagváltozókat és tagfüggvényeket tartalmaz, melyek nem statikusak, ezért az around függvény paraméterének meg kell adnunk egy LZWBInFa objektumot, melyen keresztül ezeket a tagokat el tudjuk érni. Majd megadjuk, hogy az around -t hogy függvény helyett hívódjon meg. Ehhez meg kell adnunk a teljes paraméterlistáját, mivel enélkül a fordító nem tudná egyértelműen azonosítani, melyikre gondolunk. Ezután && elválasztva megadjuk, hogy az around paraméterei közül melyiket szeretnénk paraméterként átadni a kiir függvénynek, és azt is, hogy melyik LZWBInFa objektumra szeretnénk végrehajtani a függvényt. Az előbbihez a args, utóbbitohoz target szót használjuk. Ezután következik a around függvény törzse, mely a már ismert kód részletet tartalmaza.

Kód futtatása:

```
sudo apt install aspectj
ajc LZWBInFa.java Aspect.aj
java -cp /usr/share/java/aspectjrt.jar:. LZWBInFa input.txt -o output.txt
```

A alap programunk a következő kimenetet adja a input.txt-ben található szövegre:

```
-----1 (2)
-----0 (3)
```

```
-->-----1 (5)
-->-----0 (4)
---->1 (1)
---->0 (2)
---->0 (3)
---->0 (4)
--->/ (0)
-----1 (2)
---->0 (1)
---->0 (2)
---->0 (3)
depth = 5
mean = 3.5
var = 1.2909944487358056
```

Aspektus használatával pedig a következő:

```
-->/ (0)
---->1 (1)
-----1 (2)
-----0 (3)
-----0 (4)
----->1 (5)
-----0 (2)
-----0 (3)
-----0 (4)
-----0 (1)
-----1 (2)
-----0 (2)
-----0 (3)
depth = 5
mean = 3.5
var = 1.2909944487358056
```

Összegezve, az aspektusok legnagyobb előnye, hogy nem szükséges komolyabban belenyúlnunk a régi kódunkba, ahhoz, hogy módosítsunk a működésen.

19.3. Junit teszt

A https://propater.blog.hu/2011/03/05/labormeres_othton_avagy_hogyan_dolgozok_fel_egy_pedat_poszt_kézzel_számított_mélységét_és_szórását_dolgozd_be_egy_Junit_tesztbe (sztenderd védési feladat volt korábban).

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Junit_Tanulság_tapasztalatok...

A JUnit egy egységesztelő keretrendszer, amellyel ellenőrizhetjük, hogy az általunk leprogramozott kód az elvárások szerint működik-e. A tényleges és elvárt eredmény közötti összehasonlítás állítások (assertions) segítségével történik. Az assertEquals az equals metódus alapján megvizsgálja, hogy a két eredmény megegyezik-e vagy sem.

A [ProgPater blogbejegyzés](#) tesztsorozata alapján végezzük el a JUnit tesztet, ez a string bitenként kerül feldolgozásra a BinfaTest for ciklusában (az LZWBinFa Java átirata alapján). Az assertEquals metódus első paramétere a várt érték lesz, tehát ide a papíron kapott mélység-, átlag- és szórásértékeket kell megadnunk. Ezután meghívjuk a Java átirat getMelyseg, getAtlag és getSzoras függvényeit, melyek így szintén a tesztsorozat bitjeivel fognak dolgozni. Mivel a szórásra pontosabb eredményt adhat a függvény, ezért megadjuk a lehetséges deltányi eltérést is.

```
package binfa;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class BinfaTest {

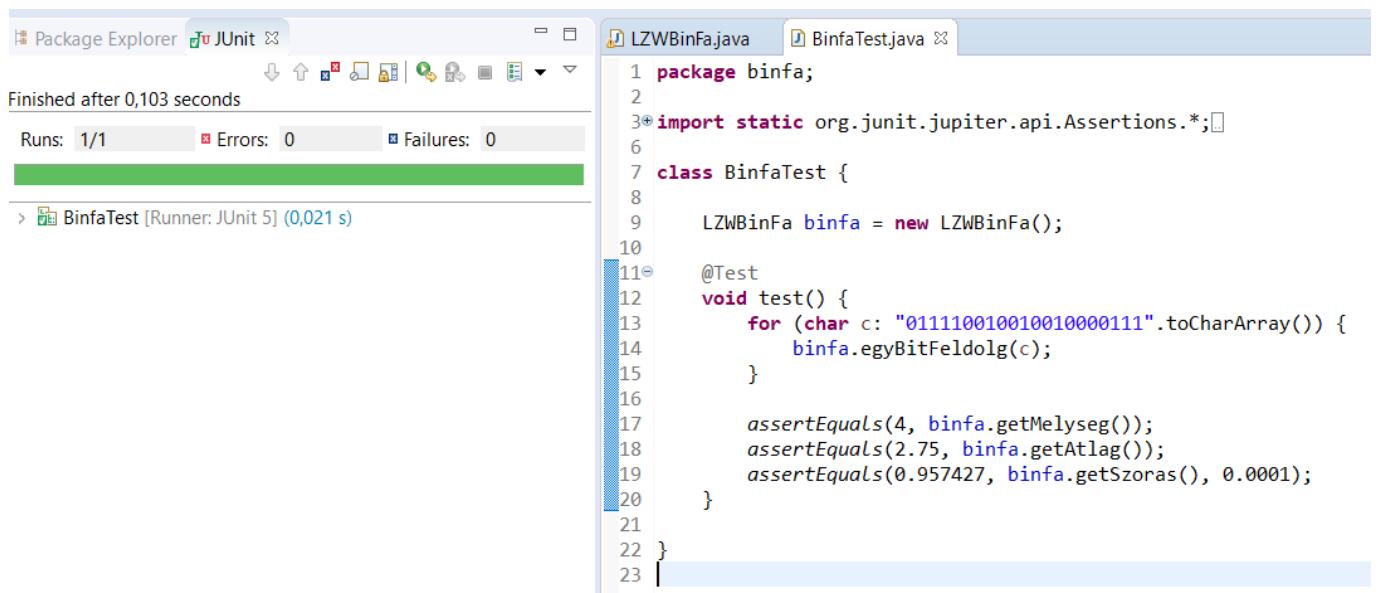
    LZWBinFa binfa = new LZWBinFa();

    @Test
    void test() {
        for (char c: "011110010010010000111".toCharArray()) {
            binfa.egyBitFeldolg(c);
        }

        assertEquals(4, binfa.getMelyseg());
        assertEquals(2.75, binfa.getAtlag());
        assertEquals(0.957427, binfa.getSzoras(), 0.0001);
    }

}
```

A JUnit teszt elvégzéséhez én az Eclipse Marketplace-ről letölthető JUnit-Tools plugint használtam. Mivel megegyeztek az értékek, ezért sikeresen le is futott:



19.4. OSCI

Készíts egyszerű C++/OpenGL-es megjelenítőt, amiben egy kocsit irányítasz az úton.

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/3dcar.cpp

Tanulság, tapasztalatok...

Az OpenGL (Open Graphics Library) egy platformfüggetlen API, ami lehetővé teszi valós idejű 2D és 3D grafikák létrehozását. Az OpenGL Utility Toolkit C++ környezetben történő használatához először telepítenünk kell a legfontosabb OpenGL könyvtárakat.

Ubuntu operációs rendszer esetén a terminálba a következő parancsokat kell beírnunk:

```
$ sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev  
$ sudo apt-get install libglew-dev libsdl2-dev libsdl2-image-dev libglm-dev libfreetype6-dev  
$ glxinfo | grep OpenGL
```

Az OpenGL könyvtárral dolgozó C++ kódok fordításához és futtatásához bizonyos kapcsolókat is igénybe kell vennünk:

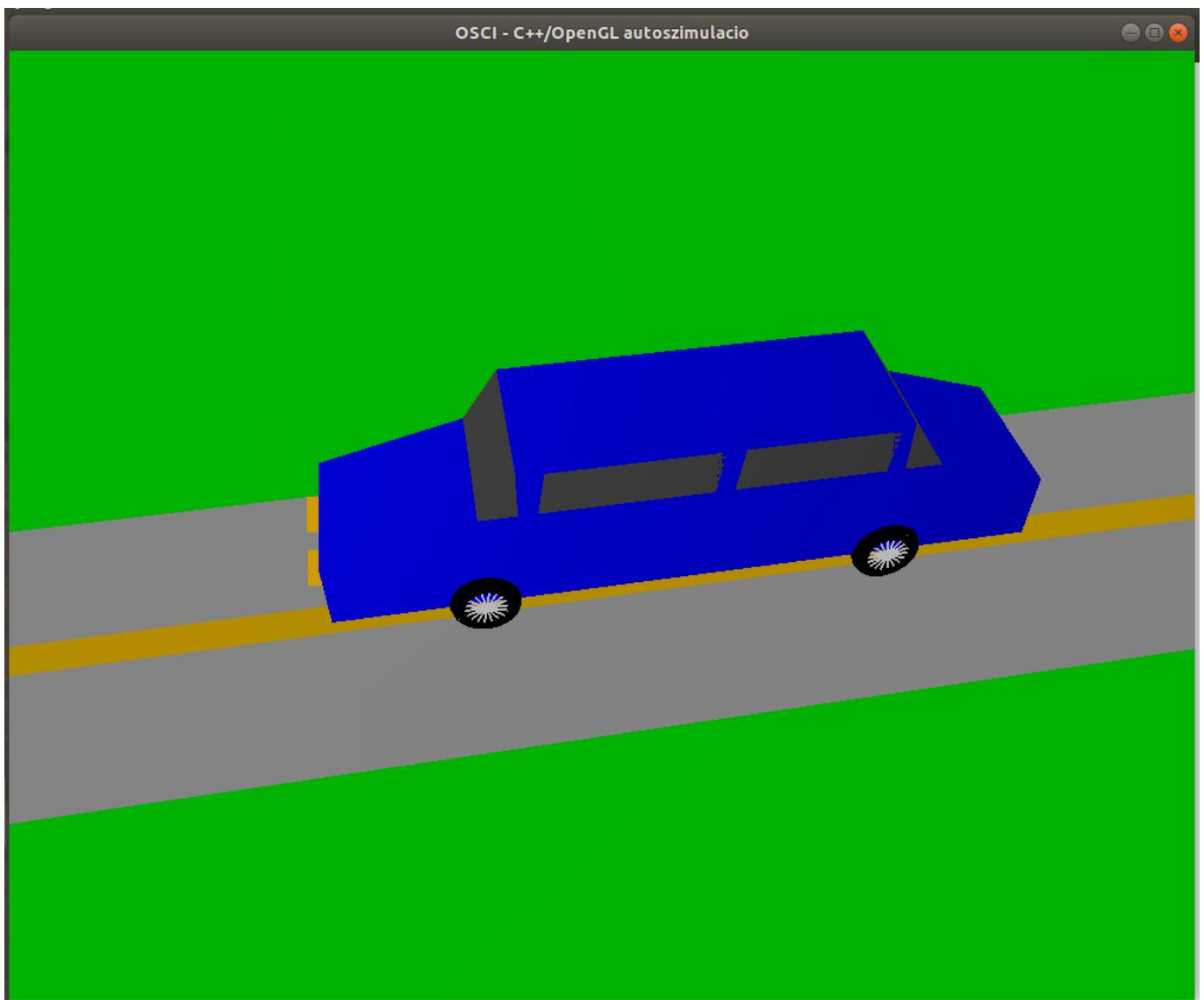
```
$ g++ 3dcar.cpp -o 3dcar -lGL -lGLU -lGLEW -lglut
```

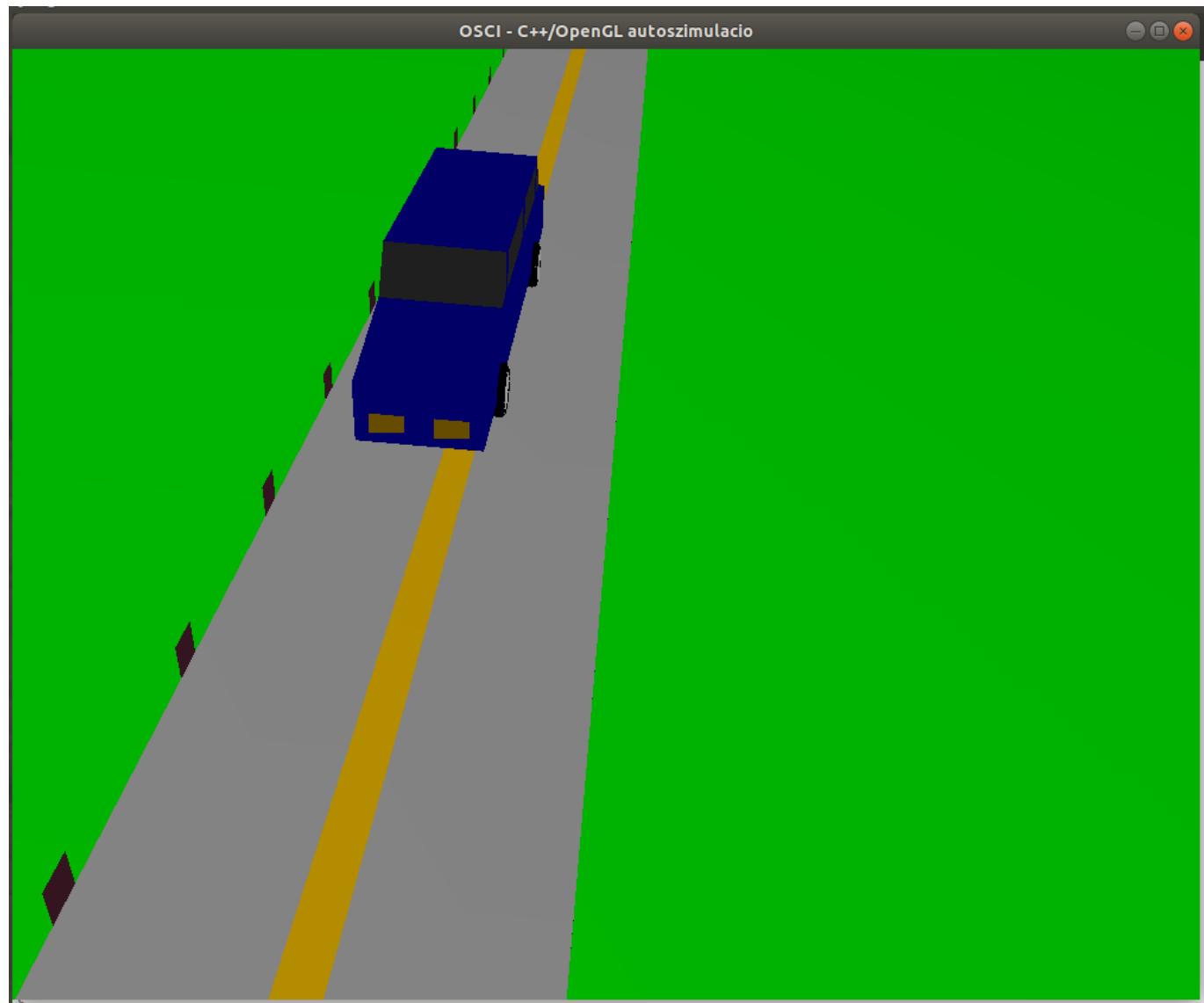
Futtatáskor láthatjuk a szimuláció irányításához, módosításához használható gombokat, billentyűket. Az X, Y és Z tengely forgatásához természetesen az X, Y és Z billentyűket használhatjuk:

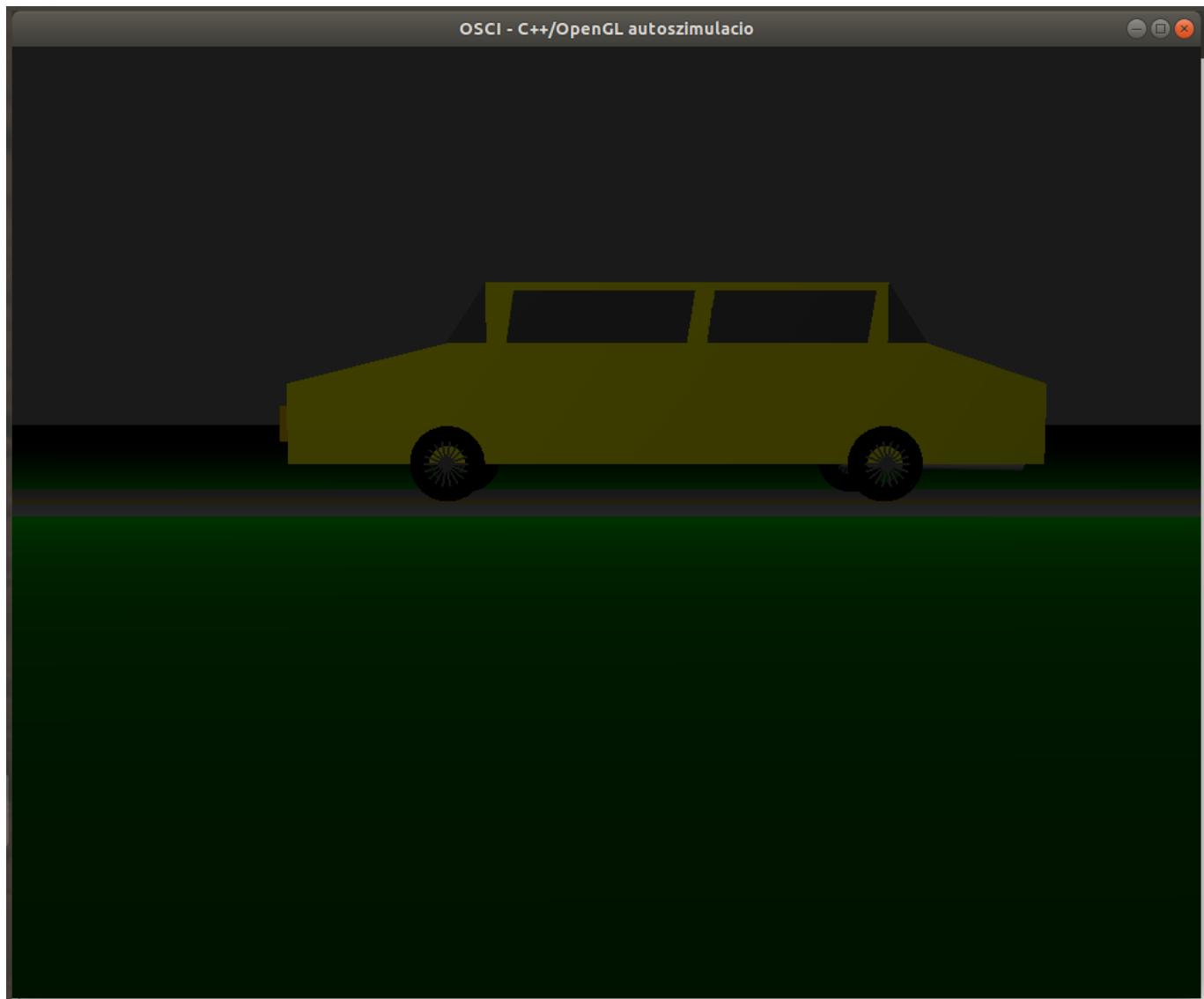
```
case 'x': xangle += 5.0;  
            glutPostRedisplay();  
            break;  
  
case 'X': xangle -= 5.0;  
            glutPostRedisplay();  
            break;  
  
case 'y':  
            yangle += 5.0;  
            glutPostRedisplay();  
            break;  
  
case 'Y':  
            yangle -= 5.0;  
            glutPostRedisplay();  
            break;  
  
case 'z':  
            zangle += 5.0;  
            glutPostRedisplay();  
            break;  
  
case 'Z':  
            zangle -= 5.0;
```

```
glutPostRedisplay();  
break;
```

Közélítéshez és távolításhoz az f és F billentyűket, a szimuláció elemeinek fel-, és lefele történő mozgatásához pedig az u és U gombokat használhatjuk. Az autó méreteinek változtatásához pedig a Q, A és S billentyűket vehetjük igénybe. Az autó az úton előre és hátra tud mozogni a bal-, és jobb nyilak segítségével. Jobb klikkel hozhatjuk elő a menüsor, amely többek között egy ködszerű hatást idéző effektet tartalmaz, melyet egy kattintással bekapcsolhatunk. A ködszerű effekten kívül tudunk egy erős fényeffektet is generálni az OpenGL-nek köszönhetően. Természetesen az autó színét is át lehet állítani az alapértelmezett kékről pirosra, zöldre, feketére, sárgára és szürkére is.







20. fejezet

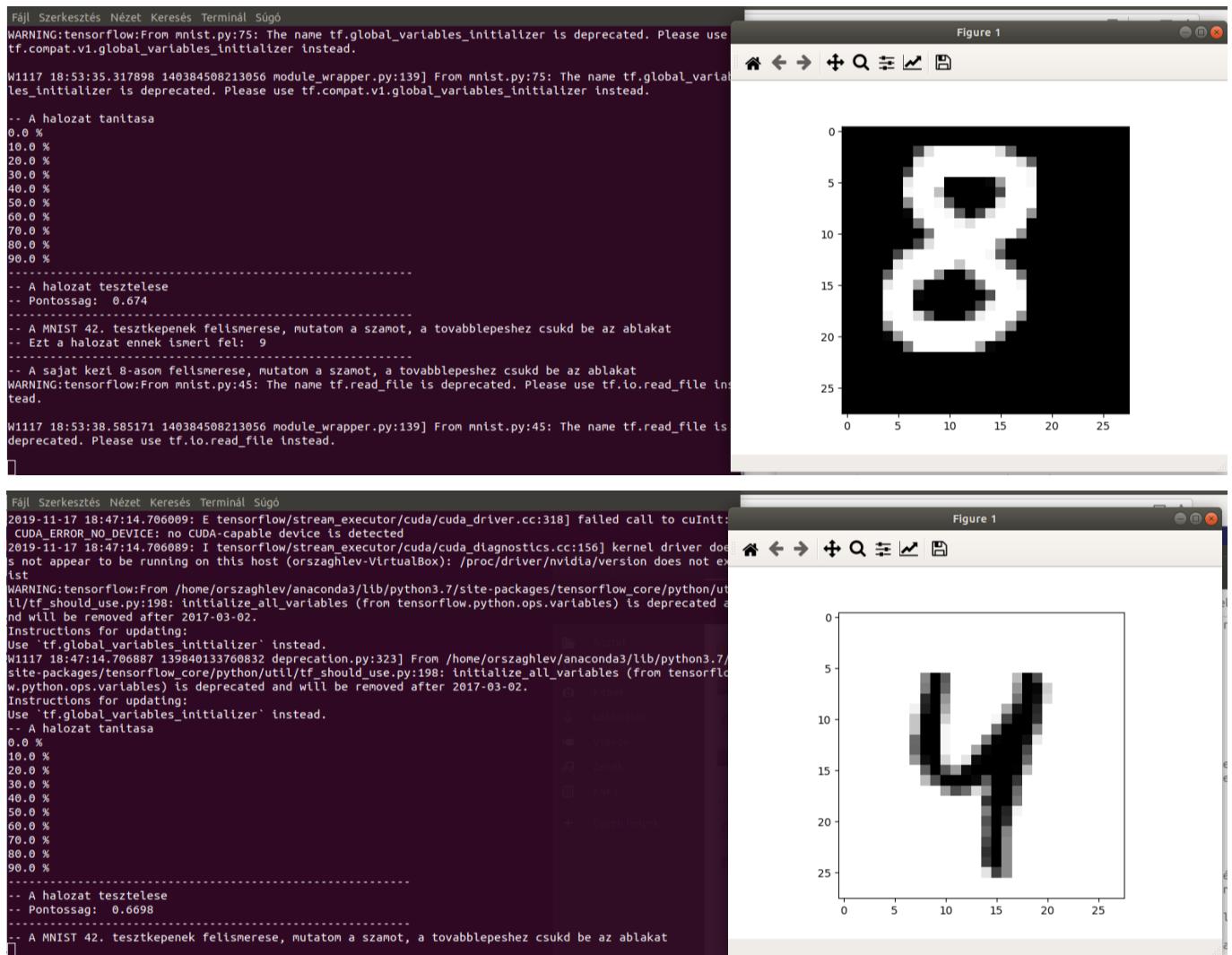
Helló, Calvin!

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 -bol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Mnist_Tanulság_tapasztalatok...

A TensorFlow egy szoftverkönyvtár, gépi tanulási algoritmusok leírására és végrehajtására. Roppant flexibilis, nagyon széles körű algoritmusok megvalósítására alkalmas, például a beszédfelismerésben, a robotikában, az információ kinyerésben, a számítógépek elleni támadások felderítésében, és az agykutatásban. A TensorFlow számítást egy irányított gráf írja le. Az adatáramlás a gráf élei mentén történik. A gráfban minden csúcs egy műveletet reprezentálhat és minden csúcsnak lehet nulla vagy több inputja. A gráf normál élei mentén áramló értékek tensorok, tetszőleges dimenziójú vektorok. A feladatban a TensorFlow segítségével készítünk fel egy modellt készírású számjegyek felismerésére. Ehhez az MNIST adatállományt is felhasználjuk, ami készírással írt számjegyek képeit tartalmazza. A Softmax egy olyan függvény, ami valószínűségeloszlást ad meg. Egy x inputhoz kiszámítja az egyes osztályokba tartozás súlyait, azután megadja az osztályokba tartozási valószínűségeket. A következő képletben W a súlyokat jelenti, b pedig a torzítási (bias) érték: $y = \text{softmax}(Wx + b)$. Az `mnist_softmax_UDPROG61.py` fájl segítségével tehát a 28x28 pixeles png képeken lévő, kézzel rajzolt számjegyek felismerésére tanítjuk a gépi modellt. A használat előtt importálnunk kell a TensorFlow-t, illetve a korábban már említett Matplotlib könyvtárat. A modell létrehozásához definiáljuk x és y változókat. A modellünkhöz szükség lesz súlyokra és bias értékre is (W és b változókban tároljuk ezeket). A keresztrétrónia implementálásához szükségünk van egy $y_{placeholder}$ is a helyes válasz bevitelére. A tanítás során a hiba csökkentésére használhatjuk a gradiens módszeren alapuló optimalizáló eljárást. Ezután a modellünket interaktív session-be tehetjük. Megkezdődik a hálózat tesztelése: kiiratjuk a pontosság értékét, amelyet az határoz meg, hogy milyen arányban képes a gépi modell észrevenni azt, hogy milyen szám van írva a képre. Kiiratjuk a tesztképet, így mi magunk is meggyőződhetünk arról, hogy milyen értéket kéne visszaadnia. A kép megvizsgálása és az ablak bezárása után pedig kiiratásra kerül az, hogy a hálózat milyen értéket észlelt.



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

Megoldás forrása: https://gitlab.com/bedote/bhax/tree/master/thematic_tutorials/Mag2/K%C3%B3dok/Cifar

Tanulság, tapasztalatok...

A CIFAR-10 60000 db 32x32-es színes kép gyűjteménye, melyet gépi tanulásra használnak. A képek 10 különböző osztályba vannak besorolva, annak alapján, hogy mit is tartalmaznak: repülőgép, autó, madár, macska, szarvas, kutya, béka, macska, hajó vagy kamion. A CIFAR-10 képek betanításához a TensorFlow mellett további packageket is le kell töltenünk, ehhez érdemes használni a Python package managerét, a pip-et. A CIFAR-10 osztályok letöltéséhez a torchvision-t fogjuk igénybe venni. A programunk futtatásakor megvizsgálja, hogy le lettek-e rántva a mappánkba a képek korábban, ha nem, akkor elindul a letöltés és a kicsomagolás. Ellenőrzés céljából a numpy és a matplotlib segítségével jelenítsük meg néhány képet az adattárból.

Figure 1

```

listing language="Python"><![CDATA[
t
=
or Fájl Szerkesztés Nézet Keresés Terminál Súgó
r python cifar10.py
Files already downloaded and verified
Files already downloaded and verified
total = torch.max(outputs.data, 1)
total = labels.size(0)
correct := (predicted == labels).sum().item()

'Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total)

correct = list(0, for i in range(10))
total = list(0, for i in range(10))
torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c += (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

    if __name__ == '__main__':
        int('Accuracy of %ss : %2d %%' % (
            classes[i], 100 * class_correct[i] / class_total[i]))]]>
am

```

A képek megvannak, az osztályuk is kiiratásra került, így következhet a neurális háló létrehozása a torch segítségével. Vegyük figyelembe, hogy kisméretű, színes képeink háromcsatornásak (RGB, nem grayscale). A képek és osztályok betanításához for ciklussal megyünk végig az adattáron. Kiiratjuk a lehetséges adatveszteséget három tizedesjegy pontossággal. Sikeres tréning esetén le is mentjük a torch segítségével az új modellt. A betanítás elméletileg megtörtént, ideje tehát tesztelni a neurális hálót. Ehhez először kiválasztunk a torchvision-nel 4 tesztfotót, és kiiratjuk ezeknek a tényleges osztályát az outputra. Kis idő után a számítás megtörténik, és megkapjuk a tippeket: az eddigi tesztjeim alapján valószínűleg legalább 2 osztályt el is fog találni.

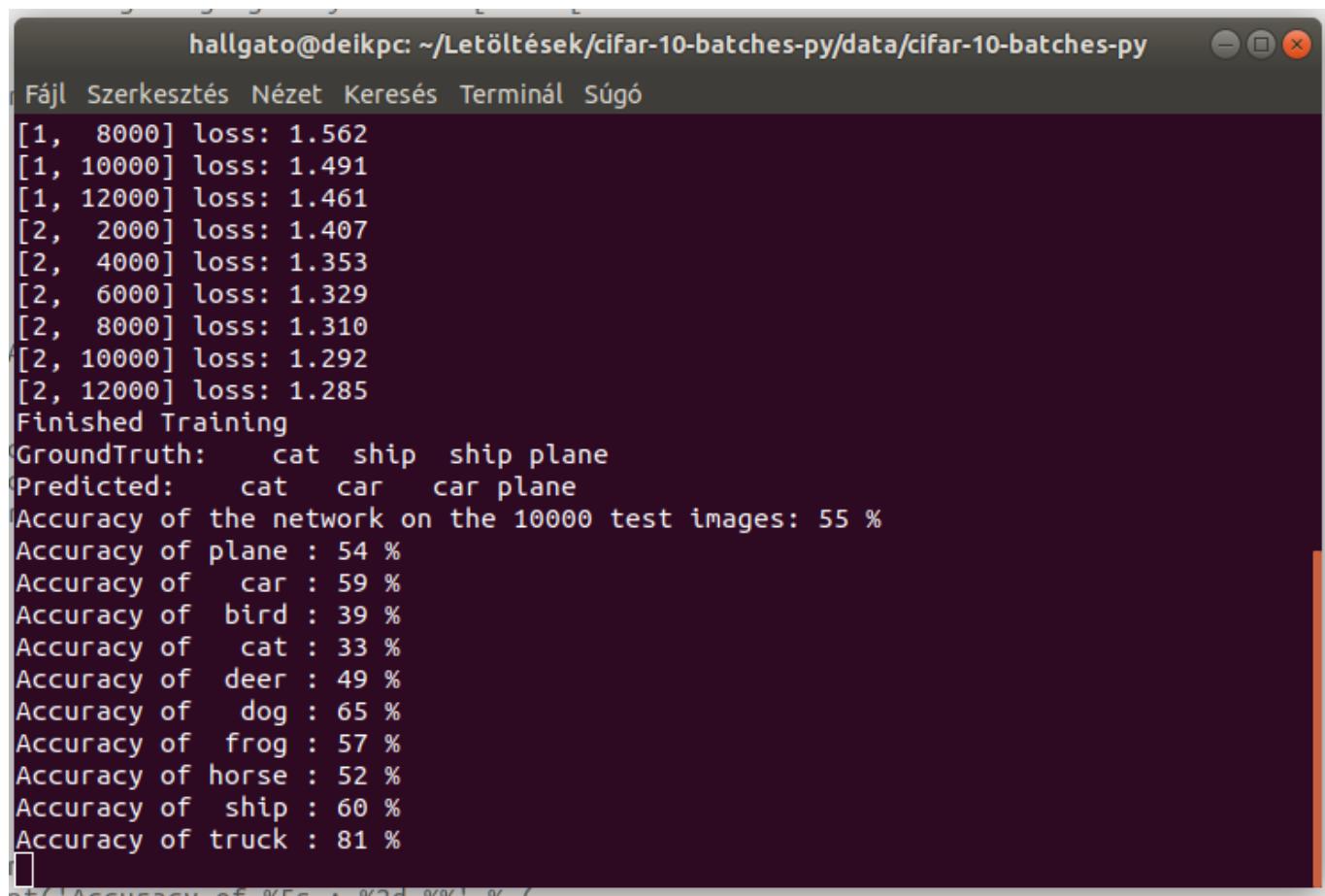
Figure 1

```

listing language="Python"><![CDATA[
hallgato@deikpc: ~/Letöltések/cifar-10-batches-py/data/cifar-10-batches-py
Fájl Szerkesztés Nézet Keresés Terminál Súgó
python cifar10.py
Files already downloaded and verified
Files already downloaded and verified
bird deer bird plane
[1, 2000] loss: 2.171
[1, 4000] loss: 1.842
[1, 6000] loss: 1.664
[1, 8000] loss: 1.562
[1, 10000] loss: 1.491
[1, 12000] loss: 1.461
[2, 2000] loss: 1.407
[2, 4000] loss: 1.353
[2, 6000] loss: 1.329
[2, 8000] loss: 1.310
[2, 10000] loss: 1.292
[2, 12000] loss: 1.285
Finished Training
[1, 2000] loss: 1.407
[1, 4000] loss: 1.353
[1, 6000] loss: 1.329
[1, 8000] loss: 1.310
[1, 10000] loss: 1.292
[1, 12000] loss: 1.285
[2, 2000] loss: 1.407
[2, 4000] loss: 1.353
[2, 6000] loss: 1.329
[2, 8000] loss: 1.310
[2, 10000] loss: 1.292
[2, 12000] loss: 1.285
int('Accuracy of %ss : %2d %%' % (
    classes[i], 100 * class_correct[i] / class_total[i]))]]>
am

```

A 10000 tesztfotó átvizsgálása után kapott pontosságról úgy kapunk eredményt, hogy a predicted képek számát elosztjuk az osztályok tényleges méretével. Ennek megfelelően osztályonként is megkaphatjuk, hogy a képek hány százalékát sikerült helyesen felismernie a modellnek.



The screenshot shows a terminal window with the following text output:

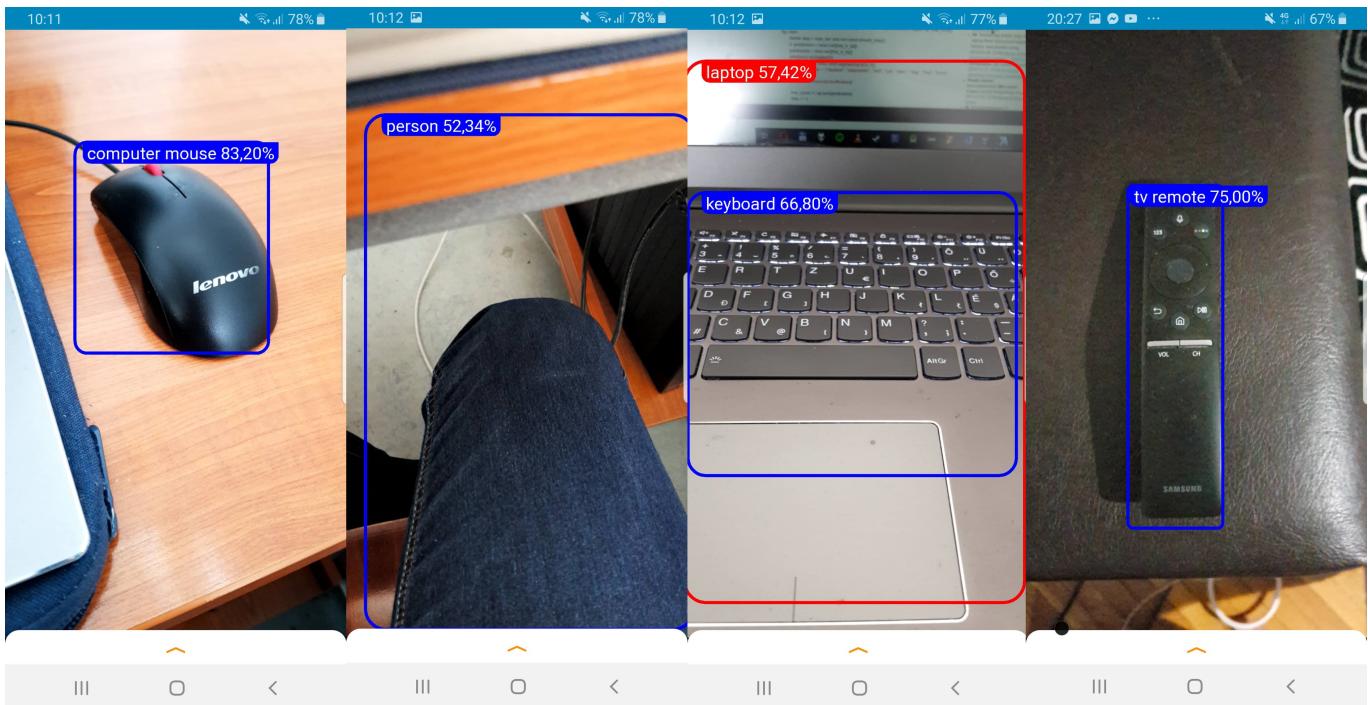
```
hallgato@deikpc: ~/Letöltések/cifar-10-batches-py/data/cifar-10-batches-py
Fájl Szerkesztés Nézet Keresés Terminál Súgó
[1, 8000] loss: 1.562
[1, 10000] loss: 1.491
[1, 12000] loss: 1.461
[2, 2000] loss: 1.407
[2, 4000] loss: 1.353
[2, 6000] loss: 1.329
[2, 8000] loss: 1.310
[2, 10000] loss: 1.292
[2, 12000] loss: 1.285
Finished Training
GroundTruth: cat ship ship plane
Predicted: cat car car plane
Accuracy of the network on the 10000 test images: 55 %
Accuracy of plane : 54 %
Accuracy of car : 59 %
Accuracy of bird : 39 %
Accuracy of cat : 33 %
Accuracy of deer : 49 %
Accuracy of dog : 65 %
Accuracy of frog : 57 %
Accuracy of horse : 52 %
Accuracy of ship : 60 %
Accuracy of truck : 81 %
```

20.3. Android telefonra a TF objektum detektálója

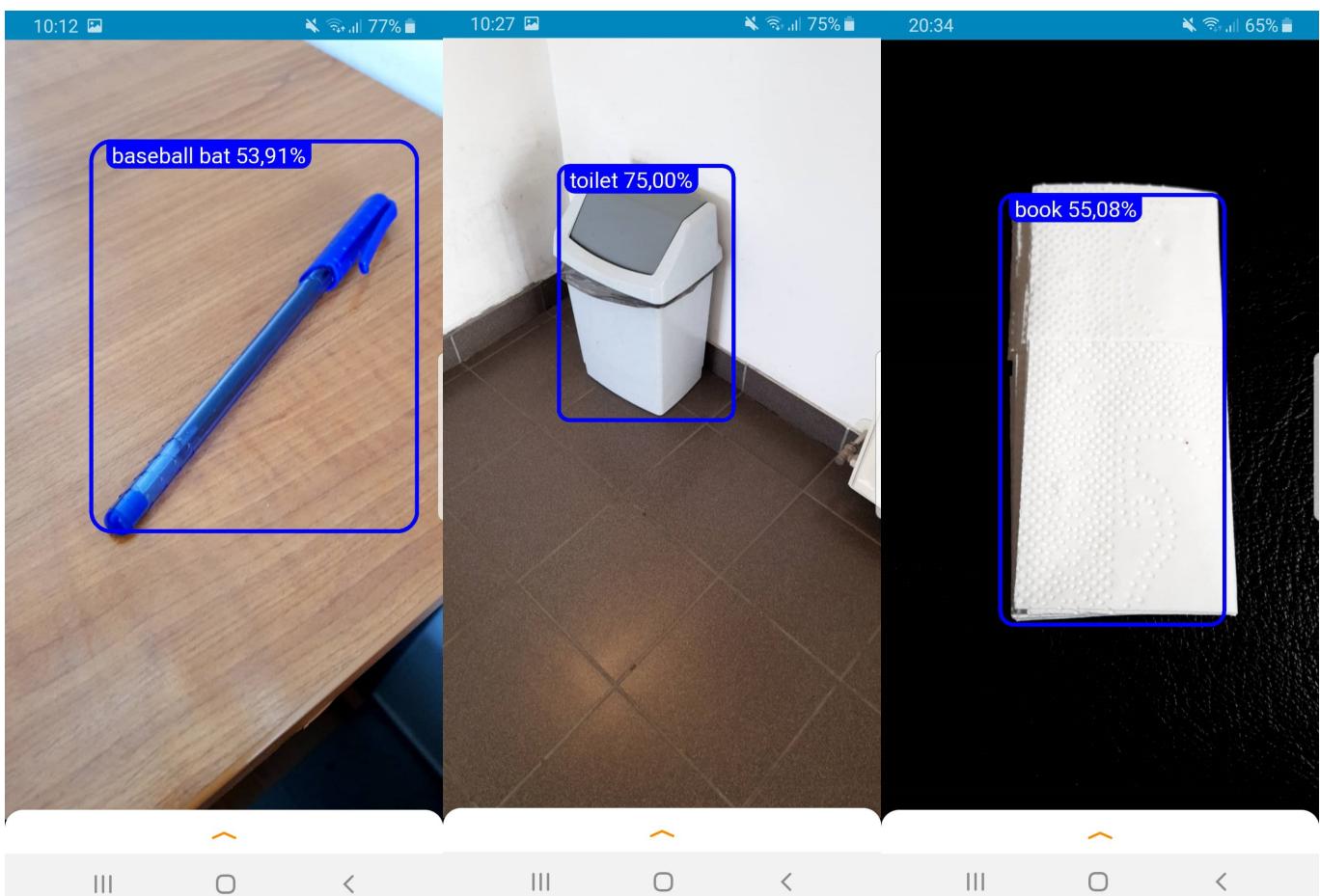
Telepítsük fel, próbáljuk ki!

Tanulság, tapasztalatok...

A TensorFlow egy szoftverkönyvtár, mely lehetővé teszi gépi tanulási algoritmusok végrehajtását. Ezek a számítások nemcsak számítógéprendszereken, de akár mobiltelefonokon is végrehajthatóak. A TensorFlow számítást egy irányított gráf írja le, melynek élein keresztül adatok (tenzorok) áramlása történik. A gráf csúcsai műveleteket reprezentálnak, a csúcsok inputjainak és outputjainak száma lehet nulla vagy több. A gráfok szerkezetének áttekintését a TensorBoard nevű vizualizációs eszközzel érhetjük el. A TensorFlow segítségével tehát képesek vagyunk betanítani Androidos készüléinket különféle objektumok detektálására. A teszteléshez én az [Object Detector and Classifier - TensorFlow](#) nevű appot töltöttem le a Google Play Áruházból. Ez az objektumfelismerő alkalmazás a Google TensorFlow gépi tanulás modelljét használja fel. A hátsó kamera segítségével valós időben próbálja érzékelni a képen látható tárgyakat. A tárgy nevén kívül egy 2 tizedesjegyű számértéket (százalékot) is kapunk, amely azt érzékelte, hogy mennyire biztos a felismerésben. Az alkalmazás viszonylag sok nyelvet ismer, magyar nyelv hiányában én a tárgyak angol megfelelőit kaptam. Az informatikai eszközök felismerésével nem volt különösebb problémám. Jó értékeket és megnevezést kaptam többek között egérre, billentyűzetre, távirányítóra is. Az emberi test felismerése is gyorsan és pontosan működött.



A probléma ott kezdődött, hogy az app leírásában szereplő képeken is hasonlóan egyértelmű példák vannak, mégghozzá nem véletlenül. Könnyen félrevezethető, ráadásul ilyenkor a pontatlan megnevezés mellé még a számérték is viszonylag magas lesz. A tollamat például a viszonylag vastag kupak miatt baseball ütőnek ismerte fel, de a kuka és a papírzsebkendő felismerése is sikertelen volt.



Sajnos az általam kipróbált alkalmazás nem tanítható, tehát kizárolag előre meghatározott, betanított tártyakat tud csak felismerni, így a szűkös tudástára miatt többszöri használatra szerintem nem kifejezetten alkalmas.

20.4. Minecraft MALMO-s példa

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lássd pl.: <https://youtu.be/bAPSu3Rndl> https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innen_tol_mi https://bhaxor.blog.hu/2018/10/28/minecraft_

Megoldás forrása: https://gitlab.com/bedote/bhax/blob/master/thematic_tutorials/Mag2/K%C3%B3dok/malmo.py

Tanulság, tapasztalatok...

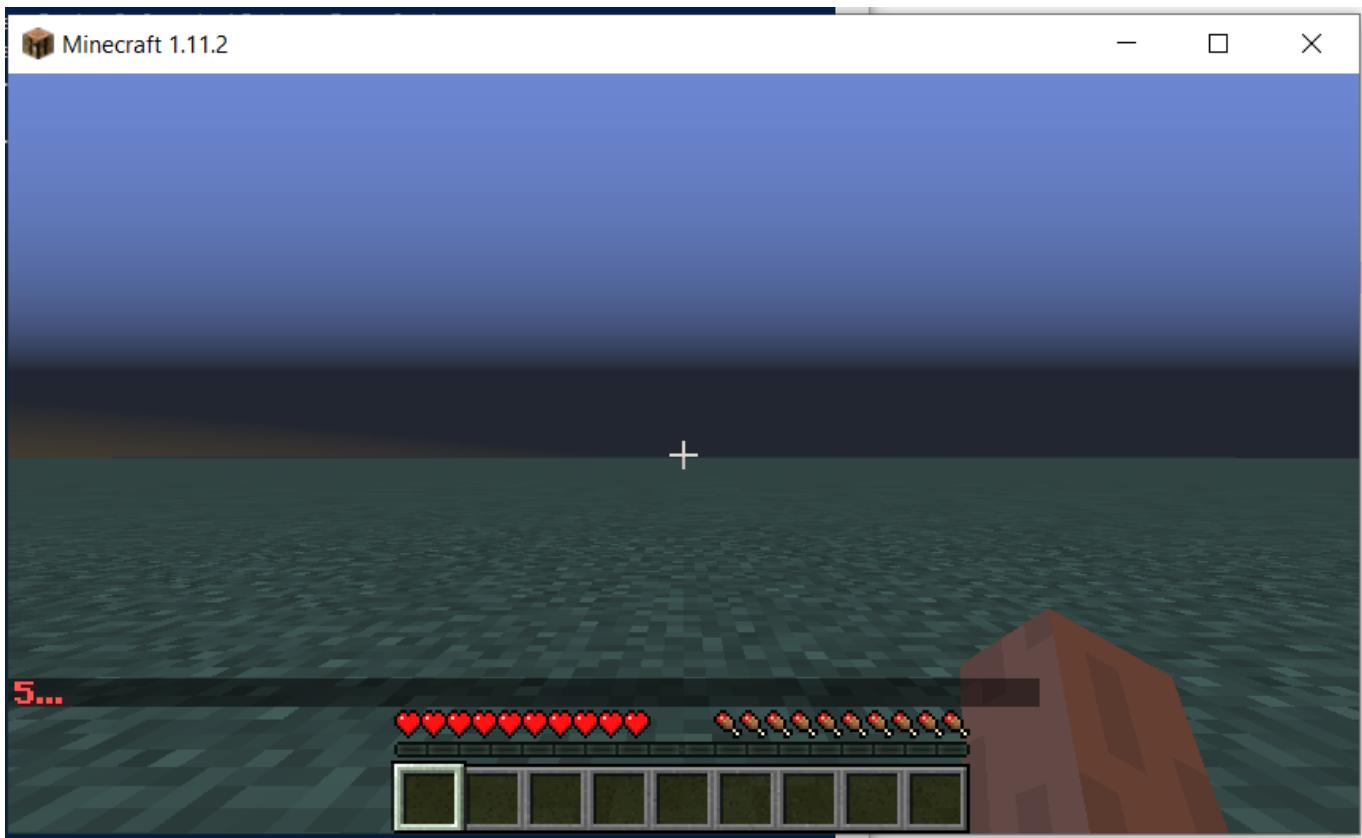
A Project Malmo egy olyan platform, amely segítségével kutatásokat végezhetünk a mesterséges intelligencia világában, méghozzá a népszerű játék, a Minecraft használatával. Olyan Python ágenst készítünk, amely egy alapértelmezett biomban gazdag világban az akadályokat legyőzve, jobbra-balra tartva felfedezi a világot. Először meg kell látogatnunk a [hivatalos GitHub repót](#), ahol láthatjuk, hogy a telepítés elvégezhető akár a Python Pip Package segítségével, de dolgozhatunk pre-built verzióval is, oprendszeről függetlenül. Én ezúttal Windows oprendszeren dolgoztam, és a pre-built fájlok közül töltöttem le egy stabil [Windows release-t](#). Ha megvolt az [OpenJDK Java 8](#), és a [CodeSynthesis XSD](#) telepítése, illetve az [FFmpeg](#) kicsomagolása, akkor jöhetnek a speciális rendszerbeállítások között megtalálható környezeti változók. A JAVA_HOME rendszerváltozó értékének meg kell adnunk az OpenJDK 8.0 főkönyvtárát, a MALMO_XSD_PATH-nek pedig a kicsomagolt pre-built zipben található MALMO mappa Schemas könyvtárát. Végezetül a Path változóhoz hozzárendeljük az FFmpeg bin almapját.

Ezután indíthatjuk is a Windows PowerShell-t, melyben a következő parancsokat kell leadnunk:

```
PS D:\> cd .\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\  
PS D:\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6> cd Minecraft  
PS D:\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\Minecraft> .\←  
launchClient.bat
```

Mostmár fut a játék, nyithatunk egy másik PowerShell terminált, amely segítségével az ágenseket fogjuk működésre bírni. Nézzük meg egyfajta tesztként az első tutorialt. Itt még az ágens nem mozdul, tíz másodperc után le is áll ez a küldetés.

```
PS D:\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6> cd ←  
Python_Examples  
PS D:\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\←  
Python_Examples> python tutorial_1.py
```



Az ágensnek természetesen tudunk adni utasításokat is, melyekkel mozgásra tudjuk bírni:

```
//turn [-1, 1] pl.:  
agent_host.sendCommand("turn -1") //balra fordul, teljes ←  
    sebességgel  
//move [-1, 1] pl.:  
agent_host.sendCommand("move 1") //előre halad, teljes sebességgel  
//pitch [-1, 1] pl.:  
agent_host.sendCommand("pitch 1") //felfele tekintés, teljes ←  
    sebességgel  
//strafe [-1, 1] pl.:  
agent_host.sendCommand("strafe -1") //bal oldalra mozdul, teljes ←  
    sebességgel  
//jump 1/0 pl.:  
agent_host.sendCommand("jump 1") //folyamatos ugrálás  
//crouch 1/0 pl.:  
agent_host.sendCommand("crouch 1") //folyamatos guggolás  
//attack 1/0 pl.:  
agent_host.sendCommand("attack 1") //folyamatos támadás  
//use 1/0 pl.:  
agent_host.sendCommand("use 1") //itemek folyamatos használata
```

A my_mission = MalmoPython.MissionSpec() parancs létrehoz egy XML stringet, amelyet a Minecraft képes értelmezni, így el is tud indulni a küldetésünk. A missionXML részben röviden összegezhetjük a mission-t, időlimitet állíthatunk (ms-ben megadva), illetve játékmódot is itt tudunk beállítani. A mission alapértelmezett élővilágát, biomját is meg tudjuk meghatározni a FlatWorldGenerator generatorString segítségével. Ahhoz, hogy olyan paramétereket adjunk meg, amelyeket értelmezni tud a játék, használhatjuk

pl. a Superflat Preset Generator-t.



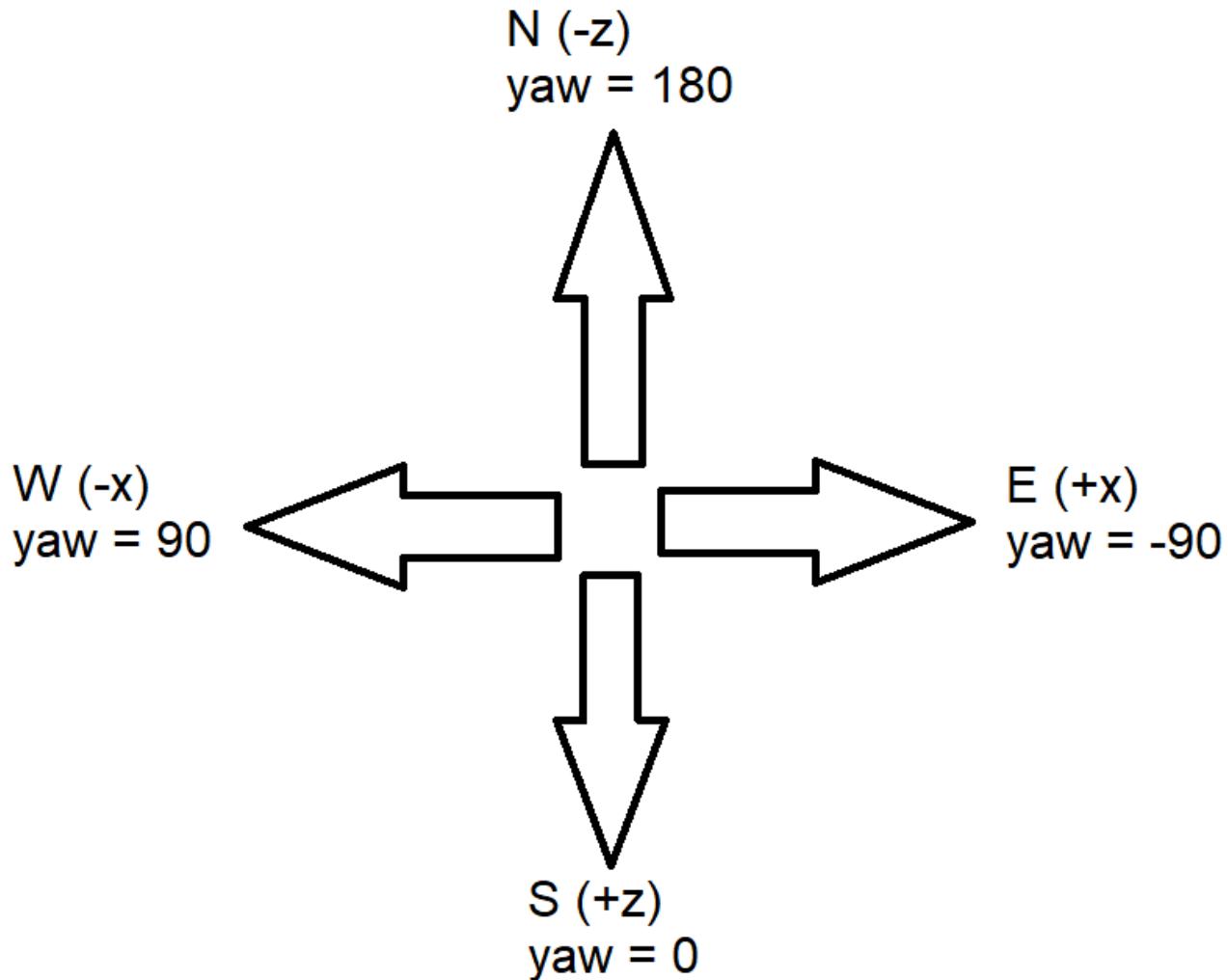
```
<FlatWorldGenerator generatorString="3;1*minecraft:bedrock,7*minecraft:dirt,1*minecraft:grass;4;village,decoration,lake"/>
```

Mielőtt a küldetés végéig tartó while ciklussal foglalkoznánk, utasítanunk kell az ágenst arra, hogy haladjon előre teljes sebességgel. Továbbá deklarálnunk kell néhány változót: a pozíció koordinátákat a stevex, stevey és stevez változók, az irányt a steveyaw változó, a nézetet pedig a stevepitch változó fogja tárolni.

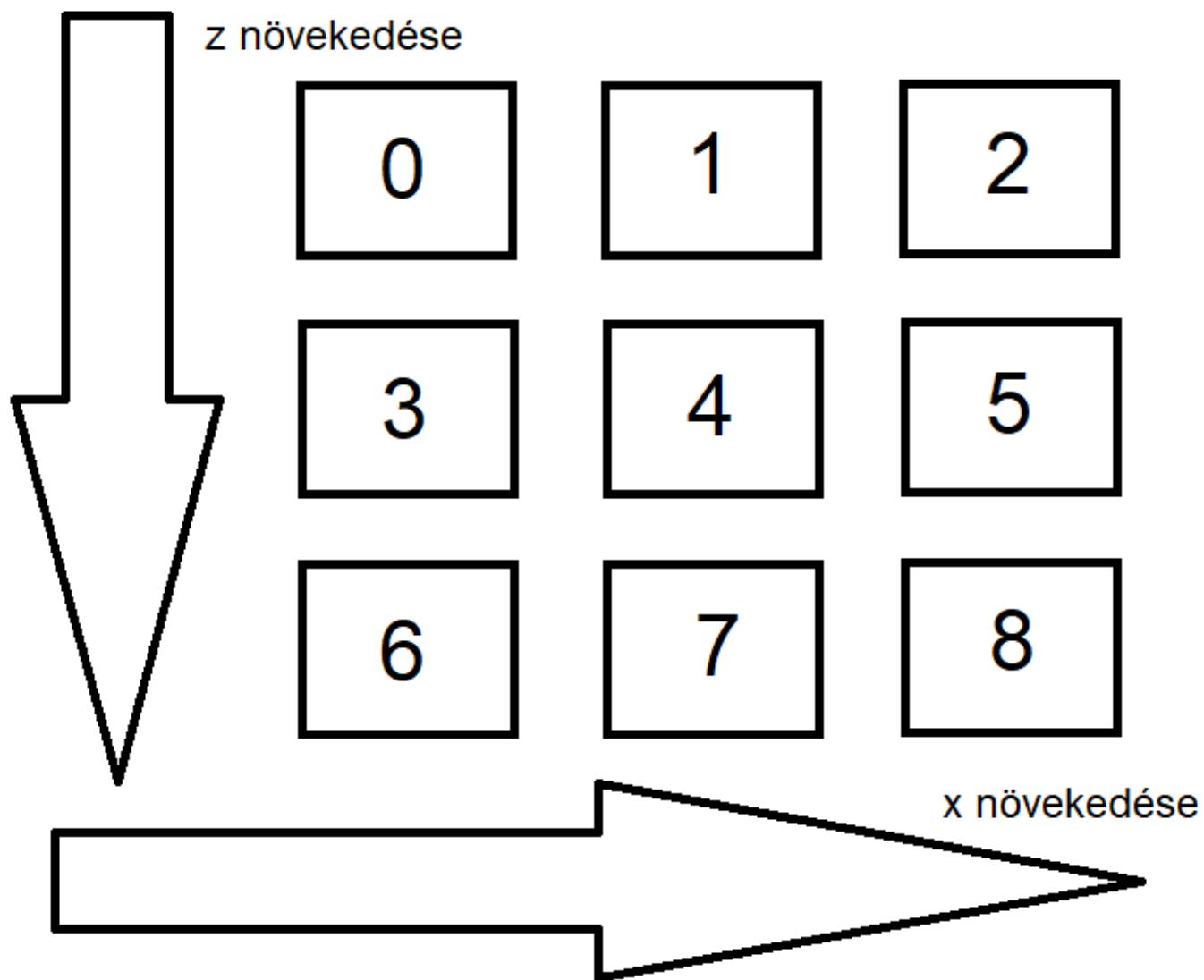
```
# Loop until mission starts:
print("Waiting for the mission to start ", end=' ')
world_state = agent_host.getWorldState()
while not world_state.has_mission_begun:
    print(".", end="")
    time.sleep(0.1)
    world_state = agent_host.getWorldState()
    for error in world_state.errors:
        print("Error:",error.text)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

A yaw értéke a Minecraft koordinátarendszeréhez igazodik: észak 180, dél 0, nyugat 90, kelet pedig -90. Ezt hivatott szemléltetni a következő ábra is:



Miközben Steve mozog, blokkok veszik körül, előfordulhat tehát, hogy akadályokkal kell szembenéznie és pl. vízbe esik vagy fának ütközik. Hogy ezt elkerüljük, a json függvény segítségével információt szerünk az őt körülvevő blokkokról, ezeknek típusát ki is tudjuk iratni a terminálra (pl. air, grass, tallgrass). 3x3-as, blokkokból álló griddel dolgozunk, ahol egy nyugat felé tekintő ágenssel szemben lévő négyzet pl. a 3-as pozíción lenne:





IV. rész

Irodalomjegyzék

20.5. Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

20.6. C

[KERNIGHANRITCHIE] Kernighan, Brian W. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

20.7. C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

20.8. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf , 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.