

Univerzális programozás

Így neveld a programozód!

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>

KÖZREMÜKÖDTEK

	Cím :		
	Univerzális programozás		
HOZZÁJÁRULÁS	NÉV	DÁTUM	ALÁÍRÁS
ÍRTA	Bátfai Norbert, Bátfai Mátyás, Bátfai Nándor, Bátfai Margaréta, és Hosszú Gyula	2020. november 4.	

VERZIÓTÖRTÉNET

VERZIÓ	DÁTUM	LEÍRÁS	NÉV
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	A Brun tételes feladat kidolgozása.	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]

DRAFT

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket, előadásokat nézzek meg, könyveket olvassak el?	3
II. Tematikus feladatok	5
2. Helló, Turing!	7
2.1. Végtelen ciklus	7
2.2. Lefagyott, nem fagyott, akkor most mi van?	9
2.3. Változók értékének felcserélése	11
2.4. Labdapattogás	11
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	14
2.6. Helló, Google!	14
2.7. A Monty Hall probléma	16
2.8. 100 éves a Brun téTEL	17
2.9. Vörös Pipacs Pokol/csiga folytonos mozgási parancsokkal	21
3. Helló, Chomsky!	23
3.1. Decimálisból unárisba átváltó Turing gép	23
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	24
3.3. Hivatalos nyelv	25
3.4. Saját lexikális elemző	25
3.5. Leetspeak	26

3.6. A források olvasása	29
3.7. Logikus	30
3.8. Deklaráció	30
3.9. Vörös Pipacs Pokol/csiga diszkrét mozgási parancsokkal	34
4. Helló, Caesar!	37
4.1. double ** háromszögmátrix	37
4.2. C EXOR titkosító	38
4.3. Java EXOR titkosító	39
4.4. C EXOR törő	40
4.5. Neurális OR, AND és EXOR kapu	44
4.6. Hiba-visszaterjesztéses perceptron	51
4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve	52
5. Helló, Mandelbrot!	55
5.1. A Mandelbrot halmaz	55
5.2. A Mandelbrot halmaz a std::complex osztállyal	60
5.3. Biomorfok	64
5.4. A Mandelbrot halmaz CUDA megvalósítása	69
5.5. Mandelbrot nagyító és utazó C++ nyelven	73
5.6. Mandelbrot nagyító és utazó Java nyelven	75
5.7. Vörös Pipacs Pokol/fel a láváig és vissza	78
6. Helló, Welch!	80
6.1. Első osztályom	80
6.2. LZW	84
6.3. Fabejárás	85
6.4. Tag a gyökér	85
6.5. Mutató a gyökér	86
6.6. Mozgató szemantika	86
6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid	87
7. Helló, Conway!	90
7.1. Hangyaszimulációk	90
7.2. Java életjáték	91
7.3. Qt C++ életjáték	92
7.4. BrainB Benchmark	93
7.5. Vörös Pipacs Pokol/19 RF	94

8. Helló, Schwarzenegger!	104
8.1. Szoftmax Py MNIST	104
8.2. Mély MNIST	105
8.3. Minecraft-MALMÖ	111
8.4. Vörös Pipacs Pokol/javíts a 19 RF-en	113
9. Helló, Chaitin!	114
9.1. Iteratív és rekurzív faktoriális Lisp-ben	114
9.2. Gimp Scheme Script-fu: króm effekt	115
9.3. Gimp Scheme Script-fu: név mandala	116
9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-eden	117
10. Helló, Gutenberg!	120
10.1. Programozási alapfogalmak:	120
10.2. Programozás bevezetés	121
10.3. Szoftverfejlesztés C++ nyelven	121
10.4. Bevezetés a Pythonba	122
III. Második felvonás	123
11. Helló, Berner-Lee!	125
11.1. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven VS. Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II	125
11.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba.	127
12. Helló, Arroway!	129
12.1. A BPP algoritmus Java megvalósítása	129
12.2. Homokozó	132
12.3. "Gagyi"	139
12.4. Yoda	140
12.5. Kódolás from scratch	142
13. Helló, Liskov!	144
13.1. Liskov helyettesítés sértése	144
13.2. Hello, Android!	146
13.3. Szülő-gyerek	147
13.4. Ciklomatikus komplexitás	150
13.5. Anti OO	151

14. Helló, Mandelbrot!	156
14.1. Reverse engineering UML osztálydiagram	156
14.2. Forward engineering UML osztálydiagram	158
14.3. BPMN	160
14.4. EPAM: Neptun tantárgyfelvétel modellezése UML-ben	160
14.5. EPAM: Neptun tantárgyfelvétel UML diagram implementálása	161
15. Helló, Chomsky!	165
15.1. Encoding	165
15.2. l334d1c4	166
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	171
15.4. Perceptron osztály	172
16. Helló, Stroustrup!	175
16.1. JDK osztályok	175
16.2. Másoló-mozgató szemantika	177
16.3. Hibásan implementált RSA törése	177
16.4. Változó argumentumszámú ctor	178
17. Helló, Gödel!	182
17.1. Gengszterek	182
17.2. STL map érték szerinti rendezése	182
17.3. Alternatív Tabella rendezése	183
17.4. C++11 Custom Allocator	185
18. Helló, !	188
18.1. FUTURE tevékenység editor	188
18.2. OOCWC Boost ASIO hálózatkezelése	190
18.3. BrainB	190
18.4. Android "GPS tracker"	192
19. Helló, Lauda !	196
19.1. Port scan	196
19.2. AOP	197
19.3. JUnit teszt	199

IV. Irodalomjegyzék**201**

19.4. Általános	202
19.5. C	202
19.6. C++	202
19.7. Lisp	202

DRAFT

Ábrák jegyzéke

2.1. A B_2 konstans közelítése	21
4.1. A double ** háromszögmátrix a memóriában	38
5.1. A Mandelbrot halmaz a komplex síkon	56
12.1. BBP formula (forrás: bbp-alg.pdf)	142
15.1. Java l334d1c4	171

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 feladatsokrot. 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 gyerekeknek, 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 gyerekeknek, 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 mászt 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

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk?

Magam is ezeken gondolkozok. Szerintem a programozás lesz a jegünk egy másik világba..., hogy a galaxisunk közepén lévő fekete lyuk eseményhorizontjának felületével ez milyen relációban van, ha egyáltalán, hát az homályos...

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán 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
- C kapcsán a [**KERNIGHANRITCHIE**] könyv adott részei.
- C++ kapcsán a [**BMECPP**] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a [The GNU C Reference Manual](#), mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipete! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [?] könyv 25-49, kb. 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket, előadásokat nézzek meg, könyveket olvassak el?

A kurzus kultúrájának élvezéséhez érdekes lehet a következő elméletek megismerése, könyvek elolvasása, filmek megnézése.

Elméletek.

- Einstein: A speciális relativitás elmélete.
- Schrödinger: Mi az élet?
- Penrose-Hameroff: Orchestrated objective reduction.
- Julian Jaynes: Breakdown of the Bicameral Mind.

Könyvek.

- Carl Sagan, Kapcsolat.
- Roger Penrose, A császár új elméje.
- Asimov: Én, a robot.
- Arthur C. Clarke: A gyermekkor vége.

Előadások.

- Mariano Sigman: Your words may predict your future mental health, <https://youtu.be/uTL9tm7S1Io>, hihetetlen, de Julian Jaynes kétkamarás tudat elméletének legjobb bizonyítéka információtechnológiai...
- Daphne Bavelier: Your brain on video games, <https://youtu.be/FktsFc0oIG8>, az esporttal kapcsolatos sztereotípiák eloszlására („The video game players of tomorrow are older adults”: 0:40-1:20, „It is not true that Screen time make your eyesight worse”: 5:02).

Filmek.

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.
- Rain Man, <https://www.imdb.com/title/tt0095953/>, az [?] munkát ihlette, melyeket akár az **MNIST**-ek helyett lehet csinálni.
- Kódjátszma, <https://www.imdb.com/title/tt2084970/>, benne a **kódtörő feladat** élménye.
- Interstellar, <https://www.imdb.com/title/tt0816692/>.
- Middle Men, <https://www.imdb.com/title/tt1251757/>, mitől fejlődött az internetes fizetés?
- Pixels, <https://www.imdb.com/title/tt2120120/>, mitől fejlődött a PC?

- Gattaca, <https://www.imdb.com/title/tt0119177/>.
- Snowden, <https://www.imdb.com/title/tt3774114/>.
- The Social Network, <https://www.imdb.com/title/tt1285016/>.
- The Last Starfighter, <https://www.imdb.com/title/tt0087597/>.
- What the #\$\$*! Do We (K)now!?, <https://www.imdb.com/title/tt0399877/>.
- I, Robot, [https://www.imdb.com/title/tt0343818.](https://www.imdb.com/title/tt0343818/)

Sorozatok.

- Childhood's End, <https://www.imdb.com/title/tt4171822/>.
- Westworld, <https://www.imdb.com/title/tt0475784/>, Ford az első évad 3. részében konkrétan meg is nevezi Julian Jaynes kétkamarás tudat elméletét, mint a hosztok programozásának alapját...
- Chernobyl, <https://www.imdb.com/title/tt7366338/>.
- Stargate Universe, <https://www.imdb.com/title/tt1286039>, a Desteny célja a mikrohullámú háttér struktúrája mögötti rejtély feltárása...
- The 100, <https://www.imdb.com/title/tt2661044/>.
- Genius, <https://www.imdb.com/title/tt5673782>.

II. rész

Tematikus feladatok

DRAFT

**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.

DRAFT

2. fejezet

Helló, Turing!

Összefoglaló videó: <https://youtu.be/dix9zxoYh58>

2.1. Végtelen ciklus

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

Megoldás videó: <https://youtu.be/lvmi6tyz-nI>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Turing/infinity-for.c, bhax/thematic_tutorials/bhax_textbook_sajat/Turing/infinity-while.c.

Számos módon hozhatunk és hozunk létre végtelen ciklusokat. Vannak esetek, amikor ez a célunk, például egy szerverfolyamat fusson folyamatosan és van amikor egy bug, mert ott lesz végtelen ciklus, ahol nem akartunk. Saját péláinkban ilyen amikor a PageRank algoritmus rázza az 1 liter vizet az internetben, de az iteráció csak nem akar konvergálni...

Egy mag 100 százalékban:

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

    return 0;
}
```

vagy az olvashatóbb, de a programozók és fordítók (szabványok) között kevésbé hordozható

```
int
#include <stdbool.h>
main ()
{
    while(true);
```

```
    return 0;  
}
```

Azért érdemes a `for (;;)` hagyományos formát használni, mert ez minden C szabvánnyal lefordul, másrészről a többi programozó azonnal látja, hogy az a végtelen ciklus szándékunk szerint végtelen és nem szoftverhiba. Mert ugye, ha a `while`-al trükközünk egy nem triviális 1 vagy `true` feltétellel, akkor ott egy másik, a forrást olvasó programozó nem látja azonnal a szándékunkat.

Egyébként a fordító a `for`-os és `while`-os ciklusból ugyanazt az assembly kódot fordítja:

```
$ gcc -S -o infinity-for.S infinity-for.c  
$ gcc -S -o infinity-while.S infinity-while.c  
$ diff infinity-while.S infinity-for.S  
1c1  
<   .file "infinity-while.c"  
---  
>   .file "infinity-for.c"
```

Egy mag 0 százalékban:

```
#include <unistd.h>  
int  
main ()  
{  
    for (;;)()  
        sleep(1);  
  
    return 0;  
}
```

Minden mag 100 százalékban:

```
#include <omp.h>  
  
int main()  
{  
  
#pragma omp parallel  
{  
    for(;;);  
}  
    return 0;  
}
```

A `gcc infinity_all_core.c -o infinity_all_core -fopenmp` parancssorral készítve a futtathatót, majd futtatva, közben egy másik terminálban a `top` parancsot kiadva tanulmányozzuk, mennyi CPU-t használunk:

```
top - 18:55:20 up 1:06, 1 user, load average: 1,90, 0,96, 0,75  
Tasks: 229 total, 2 running, 172 sleeping, 0 stopped, 0 zombie
```

```
%Cpu0 : 98,7 us, 1,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, ↵
    0,0 st
%Cpu1 : 99,3 us, 0,7 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, ↵
    0,0 st
%Cpu2 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, ↵
    0,0 st
%Cpu3 : 98,0 us, 2,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, ↵
    0,0 st
KiB Mem : 7118404 total, 4456364 free, 1158216 used, 1503824 buff/cache
KiB Swap: 1219788 total, 1219788 free, 0 used. 5664008 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
4873 hosszu    20       35572    936     840 R 348,5  0,0   1:51.62 infinity
```



Werkfilm

- <https://youtu.be/lvmi6tyz-nl>

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

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

Megoldás videó:

Megoldás forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a `Lefagy` függvényt, amely tetszőleges programról el tudja dönteni, hogy van-e benne vlgtelen ciklus: [bhax/thematic_tutorials/bhax_textbook_sajat/Turing/T100.c](https://bhax.thematic-tutorials/bhax_textbook_sajat/Turing/T100.c).

```
Program T100
#include <stdbool.h>

bool lefagy (program)
{
    if()
        return true;
    else
        return false;
}

main(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.

Ez a feladat a megállási probléma bemutatására szolgál. A megállási probléma abból áll, hogy el lehet-e dönteni egy porgramról adott bemenet esetén, hogy végtelen ciklusba kerül-e. Alan Turing 1936-ban bizonyította be, hogy nem lehetséges olyan általános algoritmust írni, amely minden program-bemenet párról megmondja, hogy végtelen ciklusba kerül-e.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Turing/valcser.c

```
Változó csere
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=30, b=45;
    printf("Csere előtt a számok: a=%d b=%d", a, b);

    a=a*b;      //a=1350 (30*45)
    b=a/b;      //b=30 (1350/45)
    a=a/b;      //a=45 (1350/30)

    printf("\nCsere után a számok: a=%d b=%d", a, b);

    return 0;
}
```

A két szám kezdetben $a = 30$, $b = 45$ volt. Az " a "-t egyenlővé téve a * b -vel($30 * 45$ -el) 1350-et kapunk. A " b "-t egyenlővé téve a / b -vel ($1350 / 45$ -el) 30-at kapunk ami a kezdeti " a "-nak volt az értéke, tehát az első cserét végre is hajtottuk. Ezután " a "-t egyenlővé tettek a / b -vel ($1350 / 30$ -al). A " b " értéke az előző cserével 45-ről 30-ra változott, tehát így az " a " értéke 45 lesz. Ezzel végrehajtottuk a második cserét is. A végeredmény pedig : Csere előtt a számok : $a = 30$, $b = 45$. Csere után a számok : $a = 45$, $b = 30$.

2.4. Labdapattogás

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

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

Megoldás forrása:bhax/thematic_tutorials/bhax_textbook_sajat/Turing/labdapatt.c bhax/thematic_tutorials/bhax_textbook_sajat/Turing/labdapatt_if.c,

Pattogó labda program `if`-fel:

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

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

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ; ; ) {

        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, " (^o^) " );

        refresh ();
        usleep ( 100000 );

        //clear();

        x = x + xnov;
        y = y + ynov;

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

    }
}
```

```
    return 0;  
}
```

Pattogó labda program `if` nélkül:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <curses.h>  
#include <unistd.h>  
  
int main ( void )  
{  
    WINDOW *ablak;  
    ablak = initscr ();  
    noecho ();  
    cbreak ();  
    nodelay (ablak, true);  
  
    int xj = 0;  
    int xk = 0;  
    int yj = 0;  
    int yk = 0;  
  
    int mx = 79 * 2;  
    int my = 20 * 2;  
  
    for ( ; ; ) {  
  
        xj = (xj - 1) % mx;  
        xk = (xk + 1) % mx;  
  
        yj = (yj - 1) % my;  
        yk = (yk + 1) % my;  
  
        clear ();  
  
        mvprintw (0, 0,  
                  " ←  
-----  
                  ");  
        mvprintw (20, 0,  
                  " ←  
-----  
                  ");  
        mvprintw (abs ((yj + (my - yk)) / 2),  
                  abs ((xj + (mx - xk)) / 2), "○");  
  
        refresh ();  
        usleep ( 50000 );
```

```
    }

    return 0;
}
}
```

A programok fordítását a **gcc labdapatt_if.c -o labdapatt_if -fincs**, illetve a **gcc labdapatt.c -o labdapatt -fincs** parancssal végezzük. A program futtatását pedig a szokásos módon: **./labdapatt ./labdapatt_if**.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU, <https://youtu.be/KRZlt1ZJ3qk>, .

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Turing/bitem.cpp

Szóhossz bitben

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

int main ()
{
    int h = 0;
    int n = 0x01;
    do
        ++h;
    while (n <= 1);
    cout << "A szohossz ezen a gepen: " << h << " bites\n";
}
```

Tanulságok, tapasztalatok, magyarázat...

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Turing>Hello_google.c,

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

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

double tavolsag (double PR[], double PRv[], int n)
{
    double osszeg = 0;
    for (int i=0; i<n; ++i){
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);
    }

    return sqrt(osszeg);
}

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

    double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
    double PRv[4] = { 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};

    int i, j;

    for(;;){
        for(i = 0; i < 4; ++i)
        {
            PR[i] = 0.0;
            for (j = 0; j < 4; ++j)
            {
                PR[i] += (L[i][j] * PRv[j]);
            }
        }
        if (tavolsag (PR, PRv, 4) < 0.0000000001)
            break;
        for(i = 0; i < 4; ++i)
```

```
    PRv[i] = PR[i];  
  
}  
  
kiir(PR, 4);  
  
return 0;  
}  
}
```

A program fordítása a `gcc hello_google.c -o hello_google -lm` parancssal lehetséges. Futtatása pedig a `./hello_google` parancssal.

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

Megoldás forrása: bhax/t�ematic_tutorials/bhax_textbook_sajat/Turing/Montyhall.r

```
kiserletek_szama = 10000  
nyeremeny = sample(1:3, kiserletek_szama, replace = T)  
jatekos = sample(1:3, kiserletek_szama, replace = T)  
musorvezeto = vector(length = kiserletek_szama)  
  
#ESET ←  
1-----  
  
#Nem változtatunk a választásunkon  
for (i in 1:kiserletek_szama)  
{  
  if (nyeremeny[i] == jatekos[i])  
  {  
  
    mibol = setdiff(c(1, 2, 3), nyeremeny[i])  
  
  } else  
  {  
    mibol = setdiff(c(1, 2, 3), c(nyeremeny[i], jatekos[i]))  
  
  }  
  
  musorvezeto[i] = mibol[sample(1:length(mibol), 1)]
```

```
}
```

```
nemvaltoztatesnyer = which(nyeremeny == jatekos)
```



```
#ESET ←
```

```
2-----
```



```
#Változtatunk a választásunkon
```

```
jatekos_uj = vector(length = kiserletek_szama)
```

```
for (i in 1:kiserletek_szama)
```

```
{
```

```
    holvált = setdiff(c(1, 2, 3), c(musorvezeto[i], jatekos[i]))
```



```
    jatekos_uj[i] = holvált
```

```
}
```



```
valtoztatesnyer = which(nyeremeny == jatekos_uj)
```

```
print("Kísérletek szma:", quote = FALSE)
```

```
print(kiserletek_szama)
```

```
print("Hányszor nyernek változtatás nélkül:", quote = FALSE)
```

```
print(length(nemvaltoztatesnyer))
```

```
print("Hányszor nyerünk változtatással:", quote = FALSE)
```

```
print(length(valtoztatesnyer))
```

```
print("Nemváltoztat / Változtat:", quote = FALSE)
```

```
print(length(nemvaltoztatesnyer) / length(valtoztatesnyer))
```

```
print("Nemváltoztat + Változtat:", quote = FALSE)
```

```
print(length(nemvaltoztatesnyer) + length(valtoztatesnyer))
```



```
}
```

Tanulságok, tapasztalatok, magyarázat...

2.8. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

A természetes számok építőelemei a prímszámok. Abban az értelemben, hogy minden természetes szám előállítható prímszámok szorzataként. Például $12=2 \cdot 2 \cdot 3$, vagy például $33=3 \cdot 11$.

Prímszám az a természetes szám, amely csak önmagával és eggyel osztható. Eukleidész görög matematikus már Krisztus előtt tudta, hogy végtelen sok prímszám van, de ma sem tudja senki, hogy végtelen sok ikerprím van-e. Két prím ikerprím, ha különbségük 2.

Két egymást követő páratlan prím között a legkisebb távolság a 2, a legnagyobb távolság viszont bármilyen nagy lehet! Ez utóbbit könnyű bebizonyítani. Legyen n egy tetszőlegesen nagy szám. Akkor szorozzuk össze $n+1$ -ig a számokat, azaz számoljuk ki az $1*2*3*...*(n-1)*n*(n+1)$ szorzatot, aminek a neve $(n+1)$ faktoriális, jele $(n+1)!$.

Majd vizsgáljuk meg az a sorozatot:

$(n+1)!+2, (n+1)!+3, \dots, (n+1)!+n, (n+1)!+ (n+1)$ ez n db egymást követő azám, ezekre (a jól ismert bizonyítás szerint) rendre igaz, hogy

- $(n+1)!+2=1*2*3*...*(n-1)*n*(n+1)+2$, azaz $2*valamennyi+2$, 2 többszöröse, így ami osztható kettővel
- $(n+1)!+3=1*2*3*...*(n-1)*n*(n+1)+3$, azaz $3*valamennyi+3$, ami osztható hárommal
- ...
- $(n+1)!+(n-1)=1*2*3*...*(n-1)*n*(n+1)+(n-1)$, azaz $(n-1)*valamennyi+(n-1)$, ami osztható $(n-1)$ -el
- $(n+1)!+n=1*2*3*...*(n-1)*n*(n+1)+n$, azaz $n*valamennyi+n$, ami osztható n-el
- $(n+1)!+(n+1)=1*2*3*...*(n-1)*n*(n+1)+(n-1)$, azaz $(n+1)*valamennyi+(n+1)$, ami osztható $(n+1)$ -el

tehát ebben a sorozatban egy prim nincs, akkor a $(n+1)!+2$ -nél kisebb első prim és a $(n+1)!+ (n+1)$ -nél nagyobb első prim között a távolság legalább n.

Az ikerprímszám sejtés azzal foglalkozik, amikor a prímek közötti távolság 2. Azt mondja, hogy az egymástól 2 távolságra lévő prímek végtelen sokan vannak.

A Brun téTEL azt mondja, hogy az ikerprímszámok reciprokaiból képzett sor összege, azaz a $(1/3+1/5)+(1/5+1/7)+(1/11+1/13)+\dots$ véges vagy végtelen sor konvergens, ami azt jelenti, hogy ezek a törtek összeadva egy határt adnak ki pontosan vagy azt át nem lépve növekednek, ami határ számot B_2 Brun konstansnak neveznek. Tehát ez nem dönti el a több ezer éve nyitott kérdést, hogy az ikerprímszámok halmaza végtelen-e? Hiszen ha véges sok van és ezek reciprokait összeadjuk, akkor ugyanúgy nem lépjük át a B_2 Brun konstans értékét, mintha végtelen sok lenne, de ezek már csak olyan csökkenő mértékben járulnának hozzá a végtelen sor összegéhez, hogy így sem lépnék át a Brun konstans értékét.

Ebben a példában egy olyan programot készítetünk, amely közelíteni próbálja a Brun konstans értékét. A repó [bhax/attention_raising/Primek_R/stp.r](#) mevű állománya kiszámolja az ikerprímeket, összegzi a reciprokaikat és vizualizálja a kapott részeredményt.

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
```

```
# along with this program. If not, see <http://www.gnu.org/licenses/>

library(matlab)

stp <- function(x) {

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

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

Soronként értelemezzük ezt a programot:

```
primes = primes(13)
```

Kiszámolja a megadott számig a prímeket.

```
> primes=primes(13)
> primes
[1] 2 3 5 7 11 13
```

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

```
> diff = primes[2:length(primes)] - primes[1:length(primes)-1]
> diff
[1] 1 2 2 4 2
```

Az egymást követő prímek különbségét képzi, tehát 3-2, 5-3, 7-5, 11-7, 13-11.

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

> idx = which(diff==2)
> idx
[1] 2 3 5
```

Megnézi a diff-ben, hogy melyiknél lett kettő az eredmény, mert azok az ikerprím párok, ahol ez igaz. Ez a diff-ben lévő 3-2, 5-3, 7-5, 11-7, 13-11 különbségek közül ez a 2., 3. és 5. indexűre teljesül.

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

Kivette a primes-ból a párok első tagját.

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

Az $1/t1primes$ a $t1primes$ 3,5,11 értékéből az alábbi reciprokokat képzi:

```
> 1/t1primes  
[1] 0.33333333 0.20000000 0.09090909
```

Az $1/t2primes$ a $t2primes$ 5,7,13 értékéből az alábbi reciprokokat képzi:

```
> 1/t2primes  
[1] 0.20000000 0.14285714 0.07692308
```

Az $1/t1primes + 1/t2primes$ pedig ezeket a törteket rendre összeadja.

```
> 1/t1primes+1/t2primes  
[1] 0.5333333 0.3428571 0.1678322
```

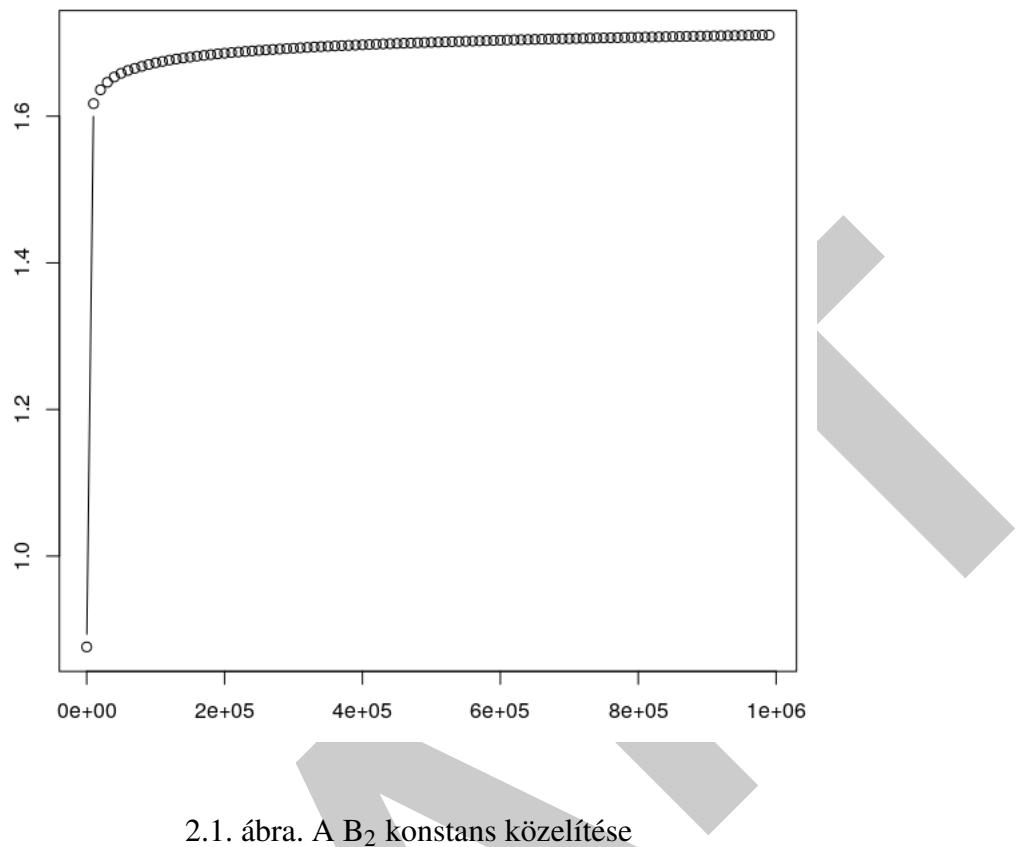
Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvényel.

```
sum(rt1plust2)
```

```
> sum(rt1plust2)  
[1] 1.044023
```

A következő ábra azt mutatja, hogy a szumma értéke, hogyan nő, egy határértékhez tart, a B_2 Brun konstanshoz. Ezt ezzel a csipettel rajzoltuk ki, ahol először a fenti számítást 13-ig végezzük, majd 10013, majd 20013-ig, egészen 990013-ig, azaz közel 1 millióig. Vegyük észre, hogy az ábra első köre, a 13 értékhez tartozó 1.044023.

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

**Werkfilm**

- <https://youtu.be/VkMFrgBhN1g>
- <https://youtu.be/aF4YK6mBwf4>

2.9. Vörös Pipacs Pokol/csiga folytonos mozgási parancsokkal

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Turing/csiga.py

Csiga mozgás

```
def run(self):  
    world_state = self.agent_host.getWorldState()  
  
    uthossz = 2  
  
    # Loop until mission ends:  
    while world_state.is_mission_running:  
        self.agent_host.sendCommand("move 1")  
        time.sleep(uthossz)  
        self.agent_host.sendCommand("jump 1")
```

```
        time.sleep(0.2)
        self.agent_host.sendCommand("jump 0")
        time.sleep(0.1)
        self.agent_host.sendCommand("turn 1")
        time.sleep(0.5)
        self.agent_host.sendCommand("turn 0")
        time.sleep(0.1)

        uthossz = uthossz + 1

        world_state = self.agent_host.getWorldState()

    }
```

A teljes fájlból azt a lényeges részt csippentettem ki, ami a csiga mozgáshoz szükséges. Én ezt úgy oldottam meg, hogy az előre mozgáshoz szükséges parancs futási idejének (vagyis a self.agent_host.sendCommand("move 1") time.sleep(adott idő)) egy uthossz nevezetű változót deklaráltam és az értéket egyenlővé tetteim 2-vel. A while ciklus végén látható, hogy "uthossz = uthossz + 1", vagyis minden egyes alkalommal, amikor a while ciklus lefut, az uthossz értéke nő 1-el. Ez teszi lehetővé, hogy minden egyes fordulás után Steve, egyre több utat tehessen meg, így előidézve a csigamozgást.

3. fejezet

Helló, Chomsky!

Összefoglaló videó: <https://youtu.be/pGngGsNb7EA>

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky/dectoun.c,

```
Decimálisból unárisba
#include <stdio.h>
```

```
int main(void)
{
    printf("Kérek egy számot decimálisan: ");
    int decnum = 0;
    scanf("%d", &decnum);

    printf("A szám unárisan:");

    for (int i = 0; i < decnum; ++i)
        (i % 5) ? printf("|") : printf(" |");

    printf("\n");
    return 0;
}
```

Lényege, hogy az decnum számot egy tetszőlegesen megválasztott, az 1 értékét jelölő szimbólum („számjegy”) decnum-szeri ismétlésével jelöli. Az ujjakon való számolás is az egyes

számrendszer használatának felel meg. A számok könnyebb olvasása érdekében elterjedt módszer az 1 értékét jelölő szimbólumok csoportosítása. Ez leginkább ötösével történik, ami visszavezethető az egy kézen lévő ujjak számára.

—Wikipedia

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

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

Megoldás videó:

Megoldás forrása:

S, X, Y "változók"
a, b, c "konstansok"
 $S \rightarrow abc, S \rightarrow aXbc, Xb \rightarrow bX, Xc \rightarrow Ybcc, bY \rightarrow Yb, aY \rightarrow aaX, aY \rightarrow aa$
S-ból indulunk ki

Első:

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

Második:

S (S \rightarrow aXbc)
aXbc (Xb \rightarrow bX)
abXc (Xc \rightarrow Ybcc)
abYbcc (bY \rightarrow Yb)
aYbbcc (aY \rightarrow aaX)
aaXbbcc (Xb \rightarrow bX)
aabXbcc (Xb \rightarrow bX)
aabbbXcc (Xc \rightarrow Ybcc)
aabbbYbcc (bY \rightarrow Yb)
aabYbbccc (bY \rightarrow Yb)
aaYbbbccc (aY \rightarrow aa)
aaabbccc

A generatív nyelvtan azoknak a szabályoknak az összessége, amelyekkel minden, a nyelvben lehetséges nyelvsorozat előállítható, azaz leírja, hogyan lehet előállítani egy átírási eljárással a kitüntetett kezdő szimbólumból a többi jelsorozatot a szabályokat egymás után alkalmazásával.

—Wikipedia

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky/c89_99.c,

Vannak olyan kódcsipetek, amelyeket ha egyes C verziókkal hibátlanul működnek, míg más C verzió esetében hibát jeleznek. Ilyen például a következő program:

```
#include <stdio.h>

int main()
{
    for (int i = 0; i < 10; i++) ;
    return 0;
}
```

Az adott programot, ha c89-es verzióval fordítjuk, akkor a következőt kapjuk válaszul:

```
hossszu@hossszu-X550JK:~/exor$ gcc -std=c89 c89_99.c
c89_99.c: In function `main':
c89_99.c:5:2: error: `for' loop initial declarations are only allowed in ←
      C99 or C11 mode
  for (int i = 0; i < 10; i++) ;
  ^~~
c89_99.c:5:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 ←
      to compile your code
```

Viszont, ha a rendszer által ajánlott c99-es verzóval fordítjuk, akkor tökéletesen működik.

3.4. Saját lexikális elemző

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

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky/realsnumber_1

```
% {
#include <stdio.h>
```

```
int realnumbers = 0;
%
digit [0-9]
%%
{digit}*(\.{digit}+) ? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

A program fordítása a **lex -o realnumber.c realnumber.l** parancssal, majd a **gcc realnumber.c -o realnumber -lfl** parancssal történik. A program futtatása pedig a **./realnumber** parancssal.

3.5. Leetspeak

Lexelj össze egy l33t cipher!

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky/l337d1c7.l](https://bhax.com/tutorials/bhax_textbook_sajat/Chomsky/l337d1c7.l)

```
%{
/*
Forditas:
$ lex -o l337d1c7.c l337d1c7.l

Futtatas:
$ gcc l337d1c7.c -o l337d1c7 -lfl
(kilépés az input vége, azaz Ctrl+D)
```

Copyright (C) 2019
Norbert Bátfai, batfai.norbert@inf.unideb.hu

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License

```
along with this program. If not, see <https://www.gnu.org/licenses/>.

*/
{

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>

#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))

struct cipher {
    char c;
    char *leet[4];
} l337d1c7 [] = {

{'a', {"4", "4", "@", "/-\\"}}, {'b', {"b", "8", "|3", "|{}"}}, {'c', {"c", "(", "<", "{}"}}, {'d', {"d", "|)", "[ ]", "|{}"}}, {'e', {"3", "3", "3", "3"}}, {'f', {"f", "|=", "ph", "|#"}}, {'g', {"g", "6", "[", "[+"}}, {'h', {"h", "4", "|-", "|[-"}}, {'i', {"1", "1", "|", "!"}}, {'j', {"j", "7", "_|", "_/"}} , {'k', {"k", "|<", "1<", "|{"}}, {'l', {"l", "1", "|", "|_"}}, {'m', {"m", "44", "(V)", "|\\//|"}}, {'n', {"n", "|\\|", "/\\/", "/V"}}, {'o', {"0", "0", "()", "[ ]"}}, {'p', {"p", "/o", "|D", "|o"}}, {'q', {"q", "9", "O_", "(, )"}}, {'r', {"r", "12", "12", "|2"}}, {'s', {"s", "5", "$", "$"}}, {'t', {"t", "7", "7", "'|'" }}, {'u', {"u", "|_|", "(_)", "[_]"}}, {'v', {"v", "\\\/", "\\\/", "\\\/" }}, {'w', {"w", "VV", "\\\/\\"}, {"(/\\)\", "(/\\)"}}, {'x', {"x", "%", ")("}, {"", ")("}}, {'Y', {"Y", "", "", ""}}, {'z', {"z", "2", "7_", ">_"}},


{'0', {"D", "0", "D", "0"}}, {'1', {"I", "I", "L", "L"}}, {'2', {"Z", "Z", "Z", "e"}}, {'3', {"E", "E", "E", "E"}}, {'4', {"h", "h", "A", "A"}}, {'5', {"S", "S", "S", "S"}}, {'6', {"b", "b", "G", "G"}}, {'7', {"T", "T", "j", "j"}},
```

```
{'8', {"X", "X", "X", "X"}},  
'9', {"g", "g", "j", "j"}}  
  
// https://simple.wikipedia.org/wiki/Leet  
};  
  
%}  
%%  
. {  
  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
    {  
  
        if(l337d1c7[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int) (100.0*rand() / (RAND_MAX+1.0));  
  
            if(r<91)  
                printf("%s", l337d1c7[i].leet[0]);  
            else if(r<95)  
                printf("%s", l337d1c7[i].leet[1]);  
            else if(r<98)  
                printf("%s", l337d1c7[i].leet[2]);  
            else  
                printf("%s", l337d1c7[i].leet[3]);  
  
            found = 1;  
            break;  
        }  
  
    }  
  
    if(!found)  
        printf("%c", *yytext);  
  
    }  
%%  
int  
main()  
{  
    srand(time(NULL)+getpid());  
    yylex();  
    return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

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

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

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

Bugok

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

i.

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

Ha a SIGNT jelkezelés nincs ignorálva, akkor a jelkezelő végezze a jelkezelést.

ii.

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

For ciklus, az i nulla, megnézzük hogy kisebb-e mint 5, minden iterációban növeljük 1-el.

iii.

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

Ugyanaz mint az előző

iv.

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

Ez érdekes, mivel az i-t már használjuk egyszer és hivatkozunk rá mint tomb i.-re.

v.

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

Az összehasonlító operátor helyett, értékadó operátort használunk, emiatt a && operátor jobb oldalán nem egy logikai operandus áll.

vi.

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

Ez is hibás kód, mivel az f függvény két int-et kap, de azok kiértékelésének sorrendje nincs meghatározva és a kiértékelési sorrendjük kérdéses.

vii.

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

Az f függvény a-ra való outputját és magát az a-t is irassuk ki.

viii.

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

Kérdéses, hogy az eredeti a-t vagy a módosított a-t fogja kiprintelni, mivel a kiértékelési sorrenddel gondok vannak.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

Megoldás forrása: [Chomksy/logikus.tex](#) [Chomksy/logikus.pdf](#)

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

A pandoc segédprogram segítségével betesszük a sorokat a logikus.tex fájlba majd a segédprogram segítségével lefordítjuk

```
$ pandoc -t latex logikus.tex -o logikus.pdf
```

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

Bármely számnál létezik nála nagyobb prím, azaz a prímek száma végtelen.

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

Bármely számnál létezik nála nagyobb prímszám, úgy hogy ennek a prímnak a rákövetkezőjének a rákövetkezője is prím legyen, vagyis az ikerprímek száma végtelen.

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

A prímek száma véges.

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

A prímek száma végtelen.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

Megoldás forrása: [Chomsky/declaration.cpp](#)

- egész
- egészre mutató mutató
- egész referencia
- egészek tömbje

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

```
#include <cstdlib>

int main(){
    int a; //egész

    int *b; // Egeszre mutato mutato

    int &r = a; //Egesz referenciaja

    int tomb[5]; //Egeszek tombje

    int (&d)[5] = tomb; //Egeszek tombjenek referenciaja

    int **e[5]; //Egeszre mutato mutatok tombje

    int *g(void); // Egeszre mutato mutatot visszaado fuggveny

    int *(*(*f)(void)) = f; //Egeszre mutato mutatot visszaado fuggvenyre ←
        mutato mutato

    int (*(*i)(int))(int, int); // Egeszet visszaado es ket egeszet kapo ←
        fuggvenyre mutato mutatot visszaado, egeszet kapo fuggveny

    int (*(*j)(int))(int, int) = i; // Fuggvenymutato egy egeszet visszaado es ←
        ket egeszet kapo fuggvenyre mutato mutatot visszaado, egeszet kapo ←
        fuggvenyre

    return 0;
}
```

Mit vezetnek be a programba a következő nevek?

- `int a;`

Egy egészet vezet be

- `int *b = &a;`

Egészre mutatót vezet be

- `int &r = a;`

Egy egész refenciáját vezeti be

- `int c[5];`

Egészek tömbjét vezeti be

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

Egészek tömbjének referenciáját vezeti be

- `int *d[5];`

Egészre mutató mutatók tömbjét vezeti be

- `int *h();`

Egészre mutató mutatót visszaadó függvényt vezeti be

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

Egészre mutató mutatót visszaadó függvényre mutató mutatót vezet be

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

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

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

Két egészet kapó, egy egészet visszaadó függvényre mutató mutató

Megoldás videó:

Megoldás forrása:

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összahasonlítása azt mutatja meg, hogy miért érdemes a **typedef** használata: [bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky/fptr.c](#), [bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky/fptr2.c](#).

```
#include <stdio.h>

int
sum (int a, int b)
{
    return a + b;
```

```
}

int
mul (int a, int b)
{
    return a * b;
}

int (*sumormul (int c)) (int a, int b)
{
    if (c)
        return mul;
    else
        return sum;

}

int
main ()
{
    int (*f) (int, int);

    f = sum;

    printf ("%d\n", f (2, 3));

    int (*(*g) (int)) (int, int);

    g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

```
#include <stdio.h>

typedef int (*F) (int, int);
typedef int (*(*G) (int)) (int, int);

int
sum (int a, int b)
{
    return a + b;
}

int
```

```
mul (int a, int b)
{
    return a * b;
}

F sumormul (int c)
{
    if (c)
        return mul;
    else
        return sum;
}

int
main ()
{
    F f = sum;

    printf ("%d\n", f (2, 3));

    G g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

3.9. Vörös Pipacs Pokol/csiga diszkrét mozgási parancsokkal

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky/csiga_d.py,

Csiga mozgás

```
def run(self):
    world_state = self.agent_host.getWorldState()
    # Loop until mission ends:
    while world_state.is_mission_running:
        if world_state.number_of_observations_since_last_state != 0:

            sensations = world_state.observations[-1].text
            #print("    sensations: ", sensations)
            observations = json.loads(sensations)
```

```
nbr7x7x7 = observations.get("nbr7x7", 0)
#print("    7x7x7 neighborhood of Steve: ", nbr7x7x7)

if "Yaw" in observations:
    self.yaw = int(observations["Yaw"])
if "Pitch" in observations:
    self.pitch = int(observations["Pitch"])
if "XPos" in observations:
    self.x = int(observations["XPos"])
if "ZPos" in observations:
    self.z = int(observations["ZPos"])
if "YPos" in observations:
    self.y = int(observations["YPos"])

#print("    Steve's Coords: ", self.x, self.y, self.z)
#print("    Steve's Yaw: ", self.yaw)
#print("    Steve's Pitch: ", self.pitch)

if "LineOfSight" in observations:
    lineOfSight = observations["LineOfSight"]
    self.lookingat = lineOfSight["type"]
#print("    Steve's <): ", self.lookingat)

if self.akadaly == self.uthossz:
    self.agent_host.sendCommand("turn 1")
    time.sleep(.2)
    self.agent_host.sendCommand("move 0")
    self.akadaly = 0
    self.akadaly2 = self.akadaly2 + 1

if self.akadaly2 > 0:
    self.agent_host.sendCommand("jumpstrafe -1")
    time.sleep(.1)
    self.agent_host.sendCommand("strafe -1")
    time.sleep(.1)
    self.akadaly2 = 0
    self.adakaly = 0
    self.uthossz = self.uthossz + 5

self.agent_host.sendCommand("move 1")
time.sleep(.1)
self.akadaly = self.akadaly + 1

world_state = self.agent_host.getWorldState()
}
```

Láthatjuk, hogy a folytonos mozgáshoz képest az úthossz ebben az esetben nem a `time.sleep(uthossz)`-on belül definiált érték, hanem annak az értéke, hogy hányszor `move`-oljon a karakterünk.

```
def __init__(self, agent_host):
    self.agent_host = agent_host
```

```
self.x = 0
self.y = 0
self.z = 0
self.yaw = 0
self.pitch = 0
self.akadaly = 0
self.akadaly2 = 0
self.uthossz = 9
```

Itt láthatjuk, hogy bevezettük az akadály, akadály2, és az úthossz változókat. Először 9-et move-ol előre a karakterünk, majd ennek az értéke növelődik minden szintlépéskor 5-tel a következő parancssal: **self.uthossz = self.uthossz + 5**. A fordulás részünk még nem hagyatkozik arra, hogy mit lát Steve, ezért ha a **self.akadaly == self.uthossz**-al, első esetben pl. 9-el, akkor jobbra fordul egyet. A ciklus végén az **akadaly**-t nullázuk, az **akadaly2**-öt pedig növeljük eggyel. Ez azért fontos, vagyis az **akadaly2**, mert ha már egyszer fordul karakterünk akkor szintet lép Steve, ezzel szemléltetve a csigamozgást.

4. fejezet

Helló, Caesar!

Összefoglaló videó: <https://youtu.be/ofHXEGjpeME>

4.1. double ** háromszögmátrix

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

Megoldás videó: <https://youtu.be/QWt28e78yCA>,

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Caesar/hm.c

Itt a C nyelv egyik nagy előnye mutatkozik meg, ami a dinamikus memóriakezelés. Ebben a példában létrehozunk egy duble ** háromszögmátrixot, ami lényegében egy két dimenziós tömb (C-ben a [] jelek használata a tömbök kezelésére csupán egy fordító adta kényelem, hogy ne kelljen minden mutató-, és cím aritmetikával foglalkozunk). A különbség itt az lesz, hogy nem mondjuk meg előre, hogy hányszám elemű tömbjeink lesznek, hanem azokat majd dinamikusan foglaljuk le.

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

int main()
{
    double **tm;

    tm = (double **) malloc(5*sizeof(double *)); //kasztolom, size_of-al ←
    //megadom a double * méretét
    for(int i = 0; i<5; ++i)
        tm[i] = malloc((i+1)*sizeof(double)); //helyet foglalok a tm-nek, (←
        //i+1) ciklusváltozó + 1

    for(int i = 0; i<5; ++i) //végig megyek a sorokon
        for(int j=0; j<i+1; ++j) //j0-tól végigmegyek i+1-ig
            tm[i][j] = i;
```

```

    *(*(tm+1)+1) = 42.0; //második sor második oszlopa, értékül adjuk neki ←
    pl. az 42-öt

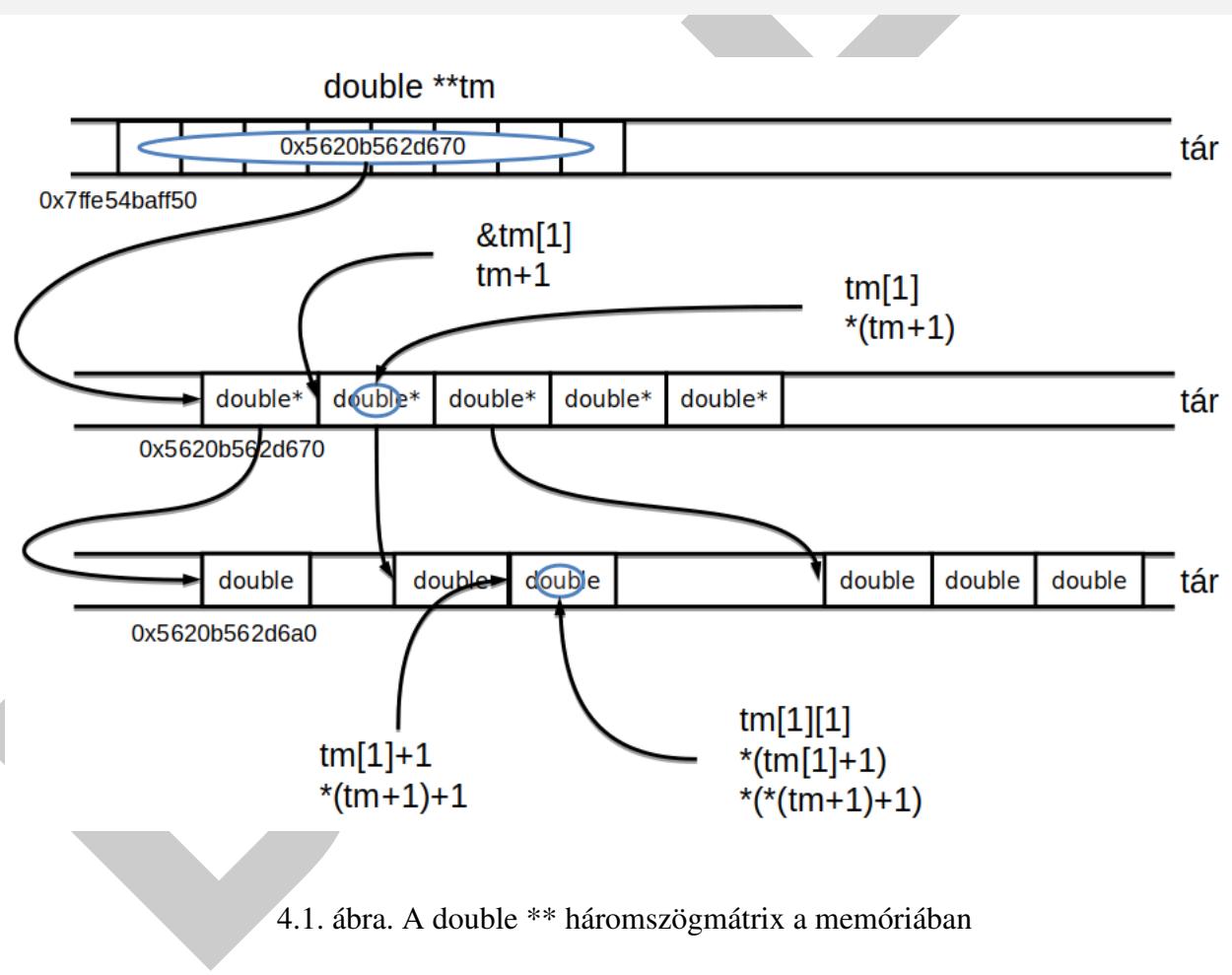
    for(int i = 0; i<5; ++i)
    {   printf("\n"); //soremelés mátrix szerű kiiráshoz
        for(int j=0; j<i+1; ++j)
            printf("%f ", tm[i][j]); //lebegőpontos számok ezért %f
    }

    for(int i = 0; i<5; ++i)
        free(tm[i]); //felszabadítjuk a tm[i]-t

    free(tm); //felszabadítjuk a tm[i]-t

    return 0;
}

```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:bhax/thematic_tutorials/bhax_textbook_sajat/Cesar/exor.c

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

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

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

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

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

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

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

A program fordítása: **gcc exor.c -o exor -std=c99** . Futtatása pedig: **./exor pla (ctrl+í) szoveg.txt >titkos.szoveg**

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Cesar/exor.java

```
public class exor {  
  
    public exor(String kulcsSzöveg,  
                java.io.InputStream bejövőCsatorna,  
                java.io.OutputStream kimenőCsatorna)  
        throws java.io.IOException {  
  
        byte [] kulcs = kulcsSzöveg.getBytes();  
        byte [] buffer = new byte[256];  
        int kulcsIndex = 0;  
        int olvasottBájt = 0;  
  
        while((olvasottBájt =  
               bejövőCsatorna.read(buffer)) != -1) {  
  
            for(int i=0; i<olvasottBájt; ++i) {  
  
                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);  
                kulcsIndex = (kulcsIndex+1) % kulcs.length;  
            }  
  
            kimenőCsatorna.write(buffer, 0, olvasottBájt);  
        }  
    }  
  
    public static void main(String[] args) {  
  
        try {  
  
            new exor(args[0], System.in, System.out);  
        } catch(java.io.IOException e) {  
  
            e.printStackTrace();  
        }  
    }  
}
```

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:bhax/thematic_tutorials/bhax_textbook_sajat/Cesar/t2.c

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 3 //
#define _GNU_SOURCE

#include <stdio.h>
#include <unistd.h>
#include <string.h>
//toro
double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
    int sz = 0;
    for (int i = 0; i < titkos_meret; ++i)
        if (titkos[i] == ' ')
            ++sz;

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

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

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

```
}

}

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

    return tiszta_lehet (titkos, titkos_meret);
}

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

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

    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\0';
    char str[3]={'p','l','a'};

    for (int ii = 0; ii <= 2; ++ii) //  

        for (int li = 0; li <= 2; ++li)
            for (int ki = 0; ki <= 2; ++ki)
            {
                kulcs[0] = str[ii];
                kulcs[1] = str[li];
                kulcs[2] = str[ki];

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

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

    return 0;
}
```

hosszu@hosszu-X550JK:~/exor\$ time ./t2 <titkos.szoveg

Kulcs: [pla]

Tiszta szöveg: [A neurális hálózat egyszerű egységekből áll, abban az ← értelemben, hogy belső állapotai leírhatók számokkal, ezek az aktivációs ← értékek. Mindegyik egység generál egy aktiválási értéktől függő ← kimeneti értéket (jelet). Az egységek csatlakoznak egymáshoz, minden egyik ← csatlakozás tartalmaz egy egyéni súlyt (szintén számokkal leírva, lásd ← súlyozás). minden egység kiküldi a kimeneti értékét az összes többi ← egységnek, amelyekkel kimenő kapcsolatban vannak. A "rendszer" bemenetei ← lehetnek érzékszervek vagy mesterséges szenzorok, érzékelők adatai, míg ← kimenetei lehet a viselkedés, jel egy kimeneti neuronon, esetleg ← bármilyen mesterségesen megjelenített válasz egy kérdésre (amik neuron- ← hálózatok esetében persze mintázatok). Ezen kapcsolatok miatt az egység ← kimenete hatással van a másik egység aktivációjára. A kapcsolat bemeneti ← oldalán álló egység fogadja az értékeket, és azok súlyozásával ← kiszámolja az aktivációs értékét (összeszorozza a bemeneti jelet a hozzá ← tartozó bemenet súlyával, és veszi ezek összegét) A kimenetet az ← aktivációtól függően az aktivációs függvény határozza meg (pl az egység ← kimenetet generál -"tüzel"- ha az aktivizáció egy határérték felett van) ← . Hogy ezek a jelek (értékek) elektrokémiai, elektromos, netán ← szimbolikus, ez a "megvalósítás" miként jétől (biológiai, hardver, ← szoftver) függ, de ez a működés alapelveit nem befolyásolja.

Fontos megjegyezni, hogy a neuronok bár számításokat végeznek ugyan, de ← mégsem processzorok. A fő különbség a kettő között az, hogy amíg a ← processzorokat programozzák (szekvenciális utasítássorozatot adnak meg ← neki), addig a neuronokat tanítják (a súlymátrix értékeinek ← beállításával). A hálózat tanulási technikája lehet ellenőrzött, ill. ← nem ellenőrzött típusú tanulás. Az ellenőrzött tanulású N-hálók esetében ← a rendszer nagyszámú, előre megadott példa alapján tanul: speciális ← algoritmusokkal addig változtatja a neuronok közötti kapcsolatokat, míg ← a megadott bemenetek minden a megadott kimeneteket "okozzák". Ilyenkor a ← hálózat a legtöbb esetben a csatlakozások súlyának módosításával tanul. ← A súlymódosítás során az ún. hibafüggvény eredményét veszi figyelembe. ← A hibafüggvény értékét sokféle módon lehet kiszámítani, a legegyszerűbb ← eset, amikor a kimeneti értékből kivonja a helyes kimeneti értéket.

A nem ellenőrzött hálóknál leginkább a Kohonen önszervező térképet ← használják, amely hálózat azon feltételezések alapján működik, hogy a ← hálózat képes a teljes bemeneti mintakészlet közös jellemzőinek ← azonosítására.

```
]  
  
real      0m0,009s  
user      0m0,009s  
sys   0m0,001s
```

4.5. Neurális OR, AND és EXOR kapu

R

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

Megoldás forrása:[bhax/thematic_tutorials/bhax_textbook_sajat/Cesar/neu.r](https://bhax-thematic-tutorials.bhax-textbook-sajat/Cesar/neu.r)

Neurális hálónak nevezzük azt a párhuzamos működésre képes információfeldolgozó eszközt, amely nagyszámú, hasonló típusú elem összekapcsolt rendszeréből áll. Továbbá jellemzője az is, hogy rendelkezik tanulási algoritmussal és képes előhívni a megtanult információt.

Itt az R programban egy neutrális hálót építünk fel, amely úgy működik, hogy meg adjuk milyen bemenetre, milyen kimenetet várunk, amit a program meg próbál mesterségesen utánozni.

a1 és a2 értékeit tartalmaz, az OR pedig a logikai VAGY műveletet jelöli. A program az általunk meghatározott szabályok alapján elkezd tanulni. A compute parancs segítségével tudjuk leellenőrizni, hogy a megfelelő eredményeket kaptuk-e vagy sem. A logikai ÉS művelet (AND) betanítása is hasonló módon történik. Az EXOR műveletnél azonban csak többrétegű neuronokkal lehetséges a tanítás (hidden = 2).

```
library(neuralnet)  
  
a1      <- c(0,1,0,1)  
a2      <- c(0,0,1,1)  
OR      <- c(0,1,1,1)  
  
or.data <- data.frame(a1, a2, OR)  
  
nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE, ←  
    stepmax = 1e+07, threshold = 0.000001)  
  
plot(nn.or)  
  
compute(nn.or, or.data[,1:2])  
  
a1      <- c(0,1,0,1)  
a2      <- c(0,0,1,1)  
AND    <- c(0,0,0,1)  
  
inand.data <- data.frame(a1, a2, AND)  
  
nn.inand <- neuralnet(AND~a1+a2, inand.data, hidden=0, linear.output=FALSE, ←  
    stepmax = 1e+07, threshold = 0.000001)
```

```
plot(nn.operand)

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

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

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)

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

```
hosszu@hosszu-X550JK:~/git_workspace/thematic_tutorials/bhax_textbook_sajat ←
/Caesar$ Rscript neu.r
$neurons
$neurons[[1]]
      a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1

$net.result
      [,1]
[1,] 0.001170313
[2,] 0.999109325
[3,] 0.999793549
[4,] 1.000000000
```

```
dev.new(): using pdf(file="Rplots1.pdf")
$neurons
$neurons[[1]]
  a1 a2
[1,] 1 0 0
[2,] 1 1 0
[3,] 1 0 1
[4,] 1 1 1

$net.result
[,1]
[1,] 2.233663e-09
[2,] 1.159100e-03
[3,] 1.278807e-03
[4,] 9.984990e-01

dev.new(): using pdf(file="Rplots2.pdf")
$neurons
$neurons[[1]]
  a1 a2
[1,] 1 0 0
[2,] 1 1 0
[3,] 1 0 1
[4,] 1 1 1

$net.result
[,1]
[1,] 0.5000036
[2,] 0.5000005
[3,] 0.5000009
[4,] 0.4999978

dev.new(): using pdf(file="Rplots3.pdf")
$neurons
$neurons[[1]]
  a1 a2
[1,] 1 0 0
[2,] 1 1 0
[3,] 1 0 1
[4,] 1 1 1

$neurons[[2]]
 [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      ←
 [,7]
[1,] 1 0.0288060 0.9799458 9.199682e-01 0.03558897 0.2907507186 ←
 0.02943174
[2,] 1 0.1476251 0.9017288 5.365905e-04 0.19099735 0.0003929578 ←
 0.99833333
```

```
[3,]      1 0.3820403 0.3803944 1.892449e-03 0.34949222 0.6783318100 ←
  0.99628773
[4,]      1 0.7830781 0.1033680 8.855493e-08 0.77463442 0.0020181408 ←
  0.99999981

$neurons[[3]]
 [,1]      [,2]      [,3]      [,4]      [,5]
[1,]      1 0.99611336 0.94805882 0.67380615 0.9981901250
[2,]      1 0.91528002 0.88208130 0.07473041 0.0020478591
[3,]      1 0.92366621 0.94113089 0.04950058 0.0003263360
[4,]      1 0.08004032 0.08133186 0.96256032 0.0001569666

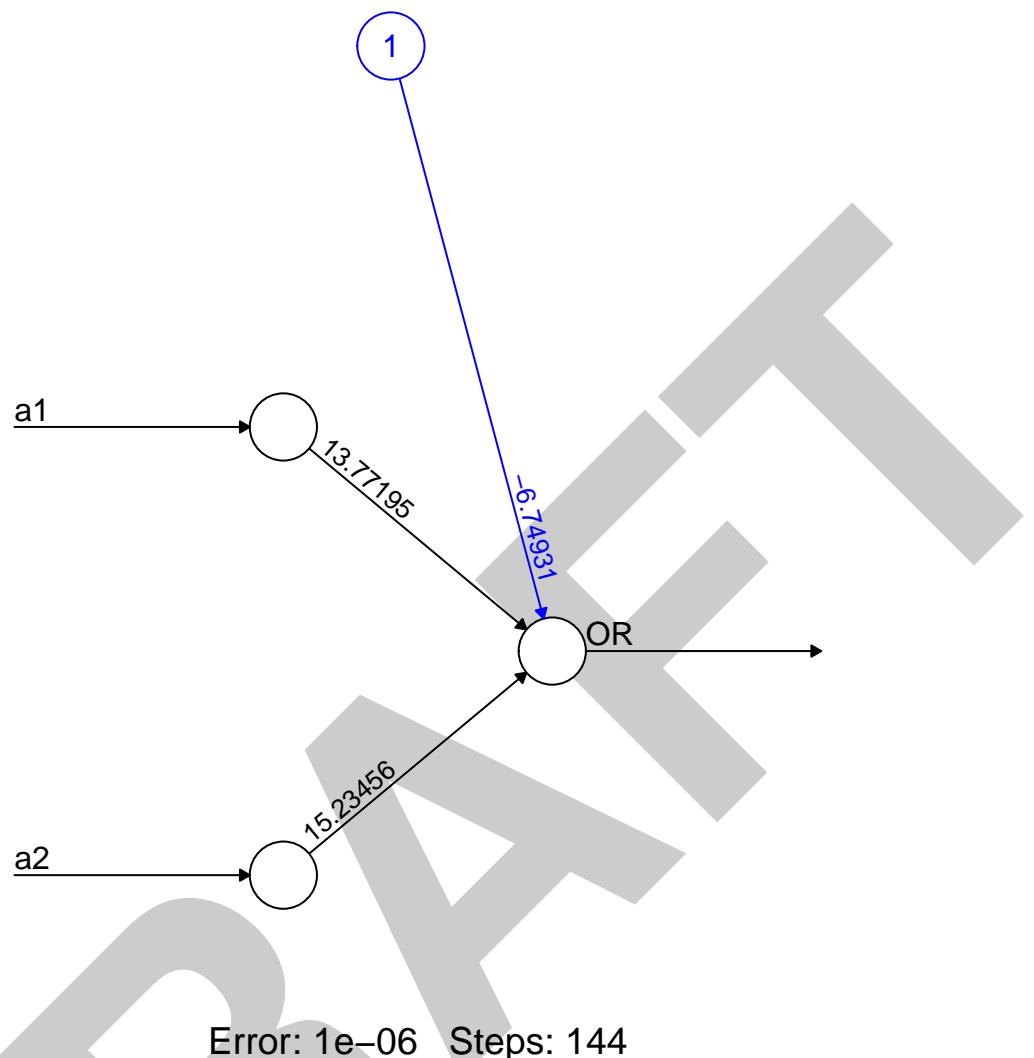
$neurons[[4]]
 [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,]      1 0.0009007891 0.99988861 0.0003044054 0.99831862 0.0001456069
[2,]      1 0.9514703536 0.09133801 0.8723371493 0.03207731 0.9715388434
[3,]      1 0.9669014323 0.06847480 0.8994491145 0.02248892 0.9808962827
[4,]      1 0.0079172003 0.99159691 0.0110317281 0.99619924 0.0012570094
 [,7]
[1,] 0.0007940185
[2,] 0.9888529539
[3,] 0.9925795567
[4,] 0.0011267832

$net.result
 [,1]
[1,] 0.0005579501
[2,] 0.9998548309
[3,] 0.9998841482
[4,] 0.0005971389
```

Ezután megkapjuk a szimuláció eredményeit a következő ábrákon!

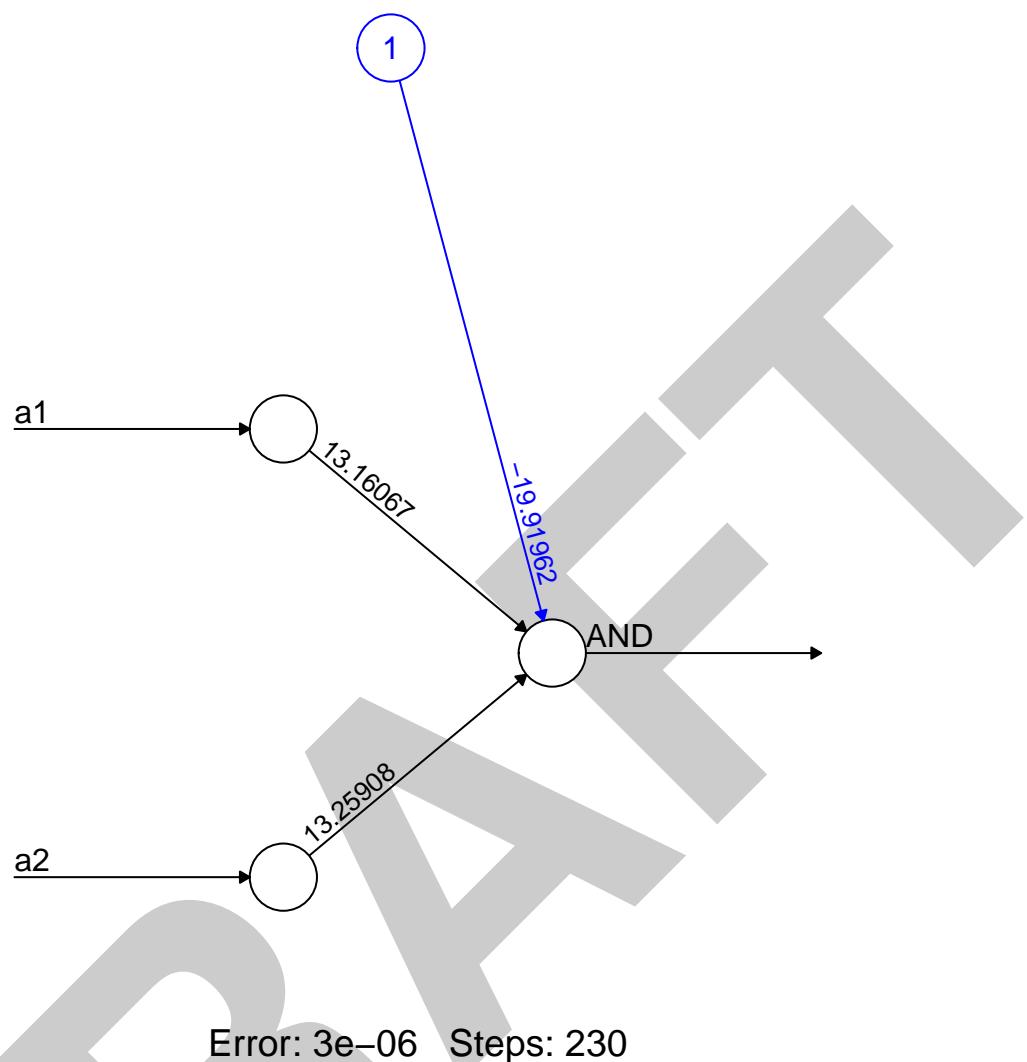
Látható, hogy megtanulta az "OR" kapcsolatot.

```
$net.result
 [,1]
[1,] 0.001447521
[2,] 0.999207152
[3,] 0.999211911
[4,] 0.999999999
```



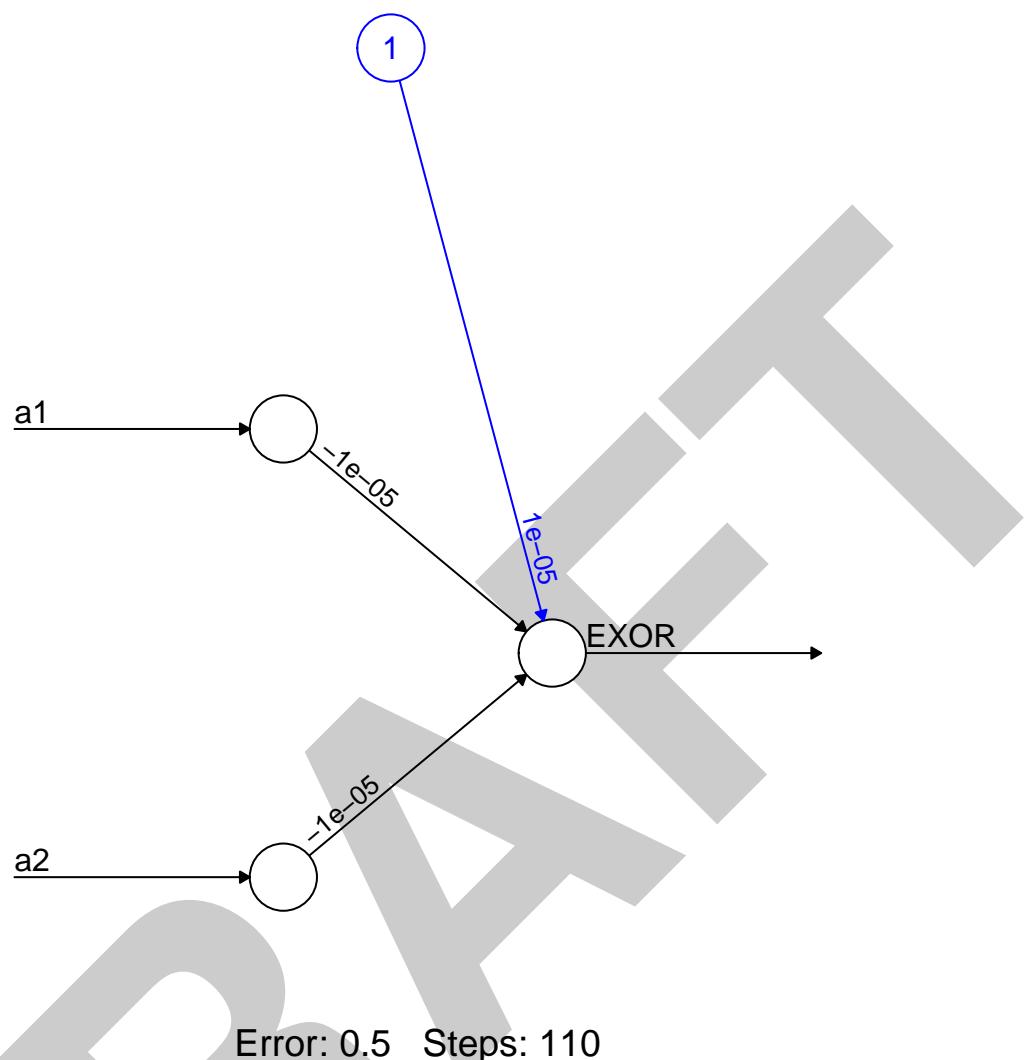
Látható, hogy megtanulta az "AND" kapcsolatot.

```
$net.result  
[,1]  
[1,] 2.233663e-09  
[2,] 1.159100e-03  
[3,] 1.278807e-03  
[4,] 9.984990e-01
```



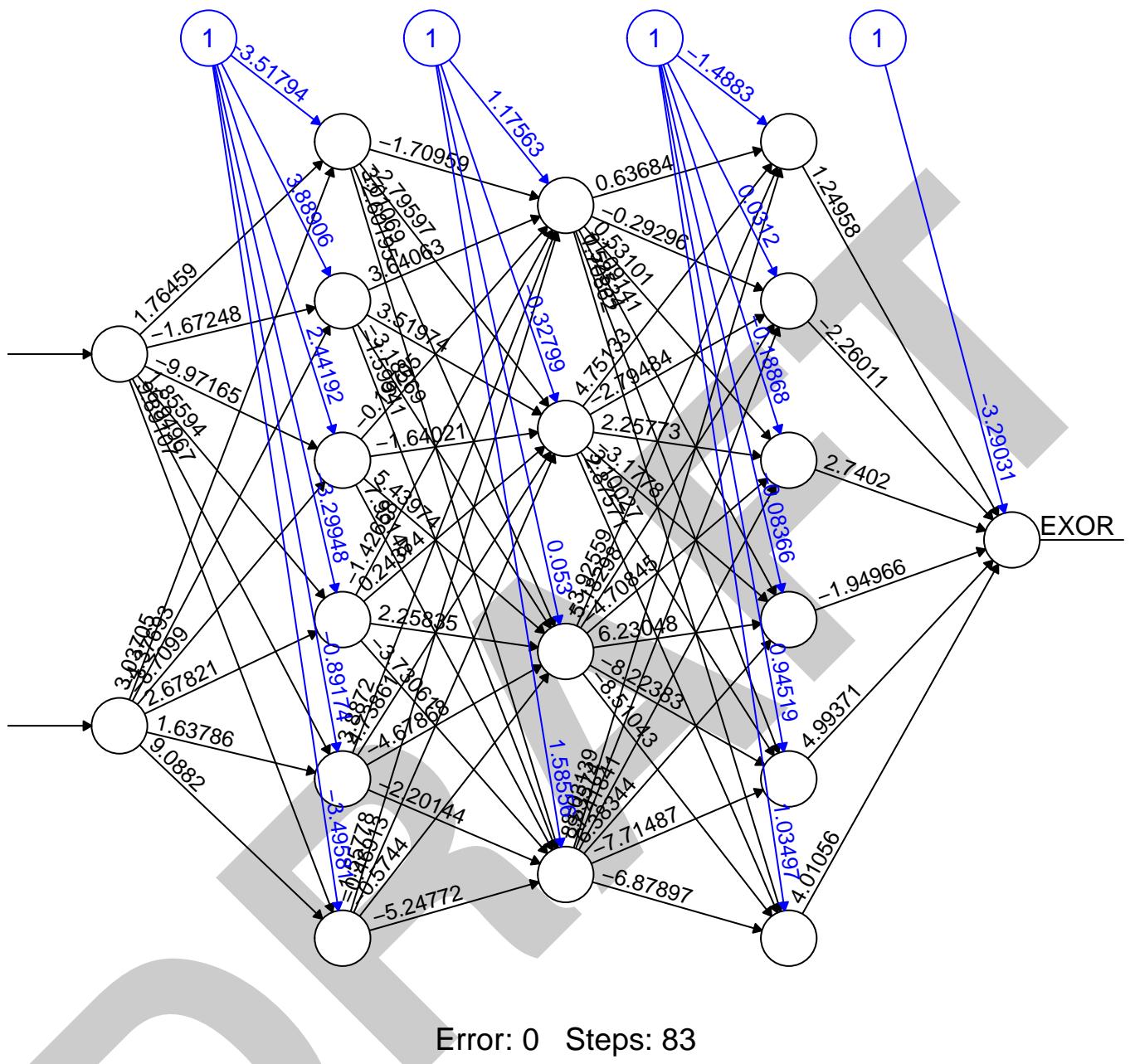
Láthatjuk, hogy a rejtett neuronok nélkül, nem ad megfelelő eredményt a program.

```
$net.result  
      [,1]  
[1,] 0.5000036  
[2,] 0.5000005  
[3,] 0.5000009  
[4,] 0.4999978
```



Viszont ha a rejtett neuronok számát, minimum kettőre állítjuk, megfelelő eredményt fogunk kapni.

```
$net.result  
      [,1]  
[1,] 0.0005579501  
[2,] 0.9998548309  
[3,] 0.9998841482  
[4,] 0.0005971389
```



4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Ceasar/prec.cpp

Ebben a feladatban a mandelbrot halmaz által generált kép rgb kódjait áttesszük a neurális háló inputjába, egy három rétegű hálót csinálunk és végül különböző számítások alapján kapunk a 3. rétegben egy számot.

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

int main (int argc, char **argv)
{
    png::image<png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width()*png_image.get_height();

    Perceptron* p = new Perceptron(3, size, 256, 1);

    double* image = new double[size];

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

    double value = (*p) (image);

    std::cout << value << std::endl;

    delete p;
    delete [] image;

}
```

4.7. Vörös Pipacs Pokolírd ki, mit lát Steve

Megoldás videó: <https://youtu.be/x9Kjn1GdycQ>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Cesar/whatcanisee.py

```
Mit lát Steve??
class Steve:
    def __init__(self, agent_host):
        self.agent_host = agent_host
        self.x = 0
        self.y = 0
        self.z = 0
        self.yaw = 0
        self.pitch = 0
        self.akadaly = 0
        self.akadaly2 = 0
        self.uthossz = 9
```

```
self.agent_host.sendCommand("look 1")

def run(self):
    world_state = self.agent_host.getWorldState()
    # Loop until mission ends:
    while world_state.is_mission_running:
        if world_state.number_of_observations_since_last_state != 0:

            sensations = world_state.observations[-1].text
            #print("    sensations: ", sensations)
            observations = json.loads(sensations)
            nbr7x7x7 = observations.get("nbr7x7", 0)
            print("    7x7x7 neighborhood of Steve: ", nbr7x7x7)

            if "Yaw" in observations:
                self.yaw = int(observations["Yaw"])
            if "Pitch" in observations:
                self.pitch = int(observations["Pitch"])
            if "XPos" in observations:
                self.x = int(observations["XPos"])
            if "ZPos" in observations:
                self.z = int(observations["ZPos"])
            if "YPos" in observations:
                self.y = int(observations["YPos"])

            #print("    Steve's Coords: ", self.x, self.y, self.z)
            #print("    Steve's Yaw: ", self.yaw)
            #print("    Steve's Pitch: ", self.pitch)

            if "LineOfSight" in observations:
                lineOfSight = observations["LineOfSight"]
                self.lookingat = lineOfSight["type"]
                #print("    Steve's <): ", self.lookingat)

            if self.lookingat == "red_flower":
                print ("Itt a pipacs, hol a pipacs?")
                self.agent_host.sendCommand("move 0")
                time.sleep(.5)
                self.agent_host.sendCommand("attack 1")
                time.sleep(.1)

            if self.akadaly == self.uthossz:
                self.agent_host.sendCommand("turn 1")
                time.sleep(.2)
                self.agent_host.sendCommand("move 0")
                self.akadaly = 0
                self.akadaly2 = self.akadaly2 + 1

            if self.akadaly2 > 4:
```

```
    self.agent_host.sendCommand("jumpstrafe -1")
    time.sleep(.1)
    self.agent_host.sendCommand("strafe -1")
    time.sleep(.1)
    self.akadaly2 = 0
    self.adakaly = 0
    self.uthossz = self.uthossz + 5

    self.agent_host.sendCommand("move 1")
    time.sleep(.1)
    self.akadaly = self.akadaly + 1

world_state = self.agent_host.getWorldState()
```

Itt látható a legfontosabb kódcsipet. Ezt fogom részletesebben magyarázni. Mint látható az előző feladatból vett megoldások is láthatóak benne a csigamozgást illetően.

```
def __init__(self, agent_host):
    self.agent_host = agent_host
    self.x = 0
    self.y = 0
    self.z = 0
    self.yaw = 0
    self.pitch = 0
    self.akadaly = 0
    self.akadaly2 = 0
    self.uthossz = 9
```

Itt deklarálnunk kell néhány változót: a pozíció koordináták a **self.x**, **self.y**, **self.z**. Az irányt a **self.yaw** változóban, a nézetet pedig a **self.pitch** változóban fogjuk tárolni.

```
sensations = world_state.observations[-1].text
#print("    sensations: ", sensations)
observations = json.loads(sensations)
nbr7x7x7 = observations.get("nbr7x7", 0)
print("    7x7x7 neighborhood of Steve: ", nbr7x7x7)
```

A json függvény segítségével információt szerzünk a Steve-et körülvevő blokkokról, ezeket természetesen ki is tudjuk iratni a következőképpen: **print("7x7x7 neighborhood of Steve: ", nbr7x7x7)**.

5. fejezet

Helló, Mandelbrot!

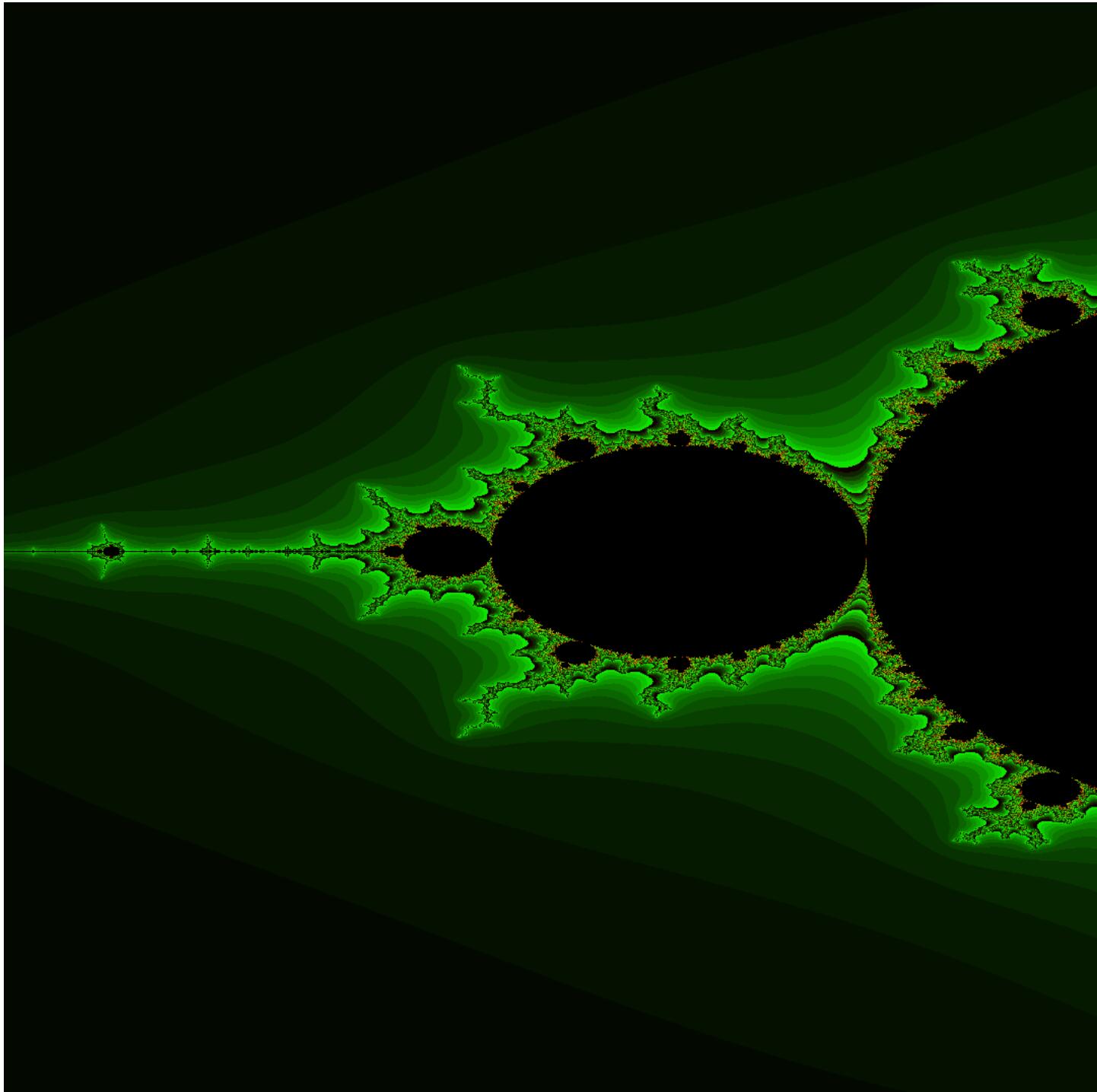
Összefoglaló videó: <https://youtu.be/p2l0p64kD58>

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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



5.1. ábra. A Mandelbrot halmaz a komplex síkon

A Mandelbrot halmazt 1980-ban találta meg Benoit Mandelbrot a komplex számsíkon. Komplex számok azok a számok, amelyek körében válaszolni lehet az olyan egyébként értelmezhetetlen kérdésekre, hogy melyik az a két szám, amelyet összeszorozva -9-et kapunk, mert ez a szám például a $3i$ komplex szám.

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak

felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1} = z_n^2 + c$, ($0 \leq n$) képlet alapján úgy, hogy a c az éppen vizsgált rácspont. A z_0 az origó. Alkalmazva a képletet a

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

Azaz kiindulunk az origóból (z_0) és elugrunk a rács első pontjába a $z_1 = c$ -be, aztán a c -től függően a további z -kbe. Ha ez az utazás kivezet a 2 sugarú körből, akkor azt mondjuk, hogy az a vizsgált rácspont nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végletesen sok z -t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácsponthoz. Ha eközben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácspont a halmaz része. (Színes meg úgy lesz a kép, hogy változatosan színezzük, például minél későbbi z -nél lép ki a körből, annál sötétebbre).

```
// mandelpngt.cpp
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
//
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>
```

```
#define MERET 800
#define ITER_HAT 64000

void
mandel (int kepadat[MERET] [MERET]) {

    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

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

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácson:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rácson csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértek, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
```

```
imZ = ujimZ;

++iteracio;

}

kepadat[j][k] = iteracio;
}

times (&tmsbuf2);
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

delta = clock () - delta;
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;

}

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

if (argc != 2)
{
    std::cout << "Hasznalat: ./mandelbrot fajlnev";
    return -1;
}

int kepadat[MERET][MERET];

mandel(kepadat);

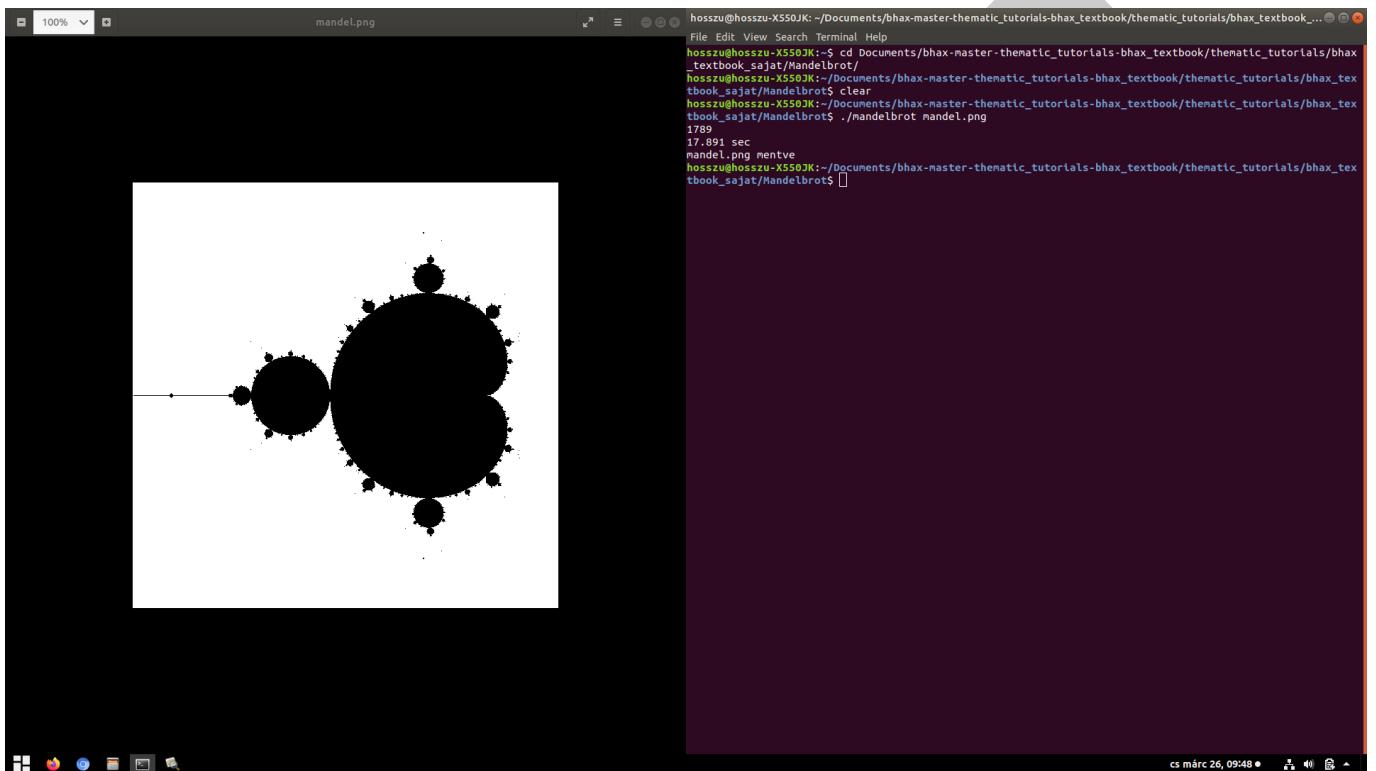
png::image<png::rgb_pixel> kep (MERET, MERET);

for (int j = 0; j < MERET; ++j)
{
    //sor = j;
    for (int k = 0; k < MERET; ++k)
    {
        kep.set_pixel (k, j,
                      png::rgb_pixel (255 -
                                      (255 * kepadat[j][k]) / ITER_HAT ←
                                      ,
                                      255 -
                                      (255 * kepadat[j][k]) / ITER_HAT ←
                                      ,
                                      255 -
                                      (255 * kepadat[j][k]) / ITER_HAT ←
                                      ));
```

```
    }

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

Az említett programot a **g++ mandelbrot.c++ -lpng16 -o mandelbrot**-al fordítjuk. Majd futtatjuk a következőképpen: **./mandelbrot mandel.png**. És a végeredmény a következő:



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

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

Megoldás videó:

Megoldás forrása:bhax/master-thematic_tutorials/bhax_textbook_sajat/Mandelbrot/3.1.2.cpp

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ←
// -0.01947381057309366392260585598705802112818 ←
// -0.0194738105725413418456426484226540196687 ←
// 0.7985057569338268601555341774655971676111 ←
// 0.798505756934379196110285192844457924366
```

```
// ./3.1.2 mandel.png 1920 1080 1020 ←
// 0.4127655418209589255340574709407519549131 ←
// 0.4127655418245818053080142817634623497725 ←
// 0.2135387051768746491386963270997512154281 ←
// 0.2135387051804975289126531379224616102874
// Nyomtatás:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --line-numbers=1 --left-footer="←
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
// color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.

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

int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double a = -1.9;
    double b = 0.7;
    double c = -1.3;
    double d = 1.3;

    if ( argc == 9 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
```

```
a = atof ( argv[5] );
b = atof ( argv[6] );
c = atof ( argv[7] );
d = atof ( argv[8] );
}
else
{
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ←
        " << std::endl;
    return -1;
}

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

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

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

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

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

        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam

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

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

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

            ++iteracio;
        }

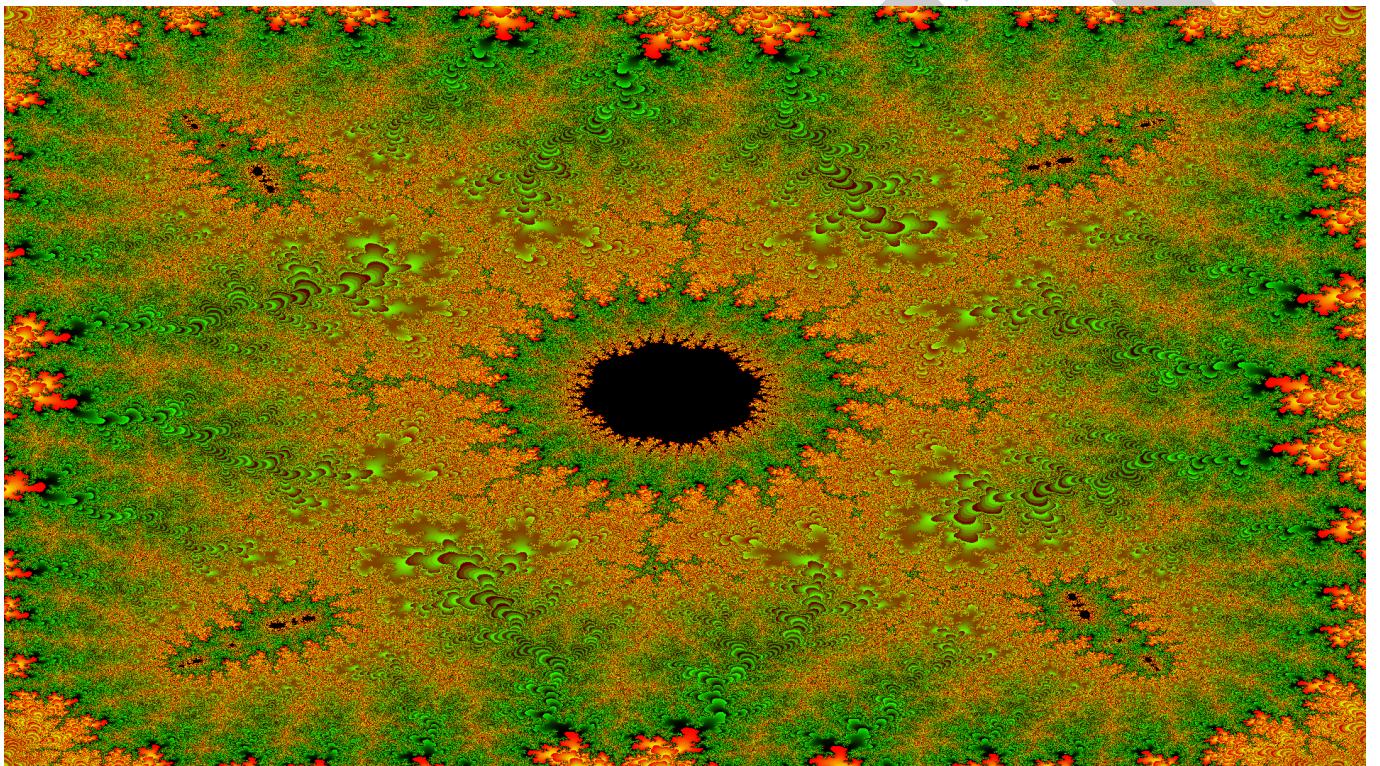
        kep.set_pixel ( k, j,
                        png::rgb_pixel ( iteracio%255, (iteracio*iteracio ←
                            )%255, 0 ) );
    }
}
```

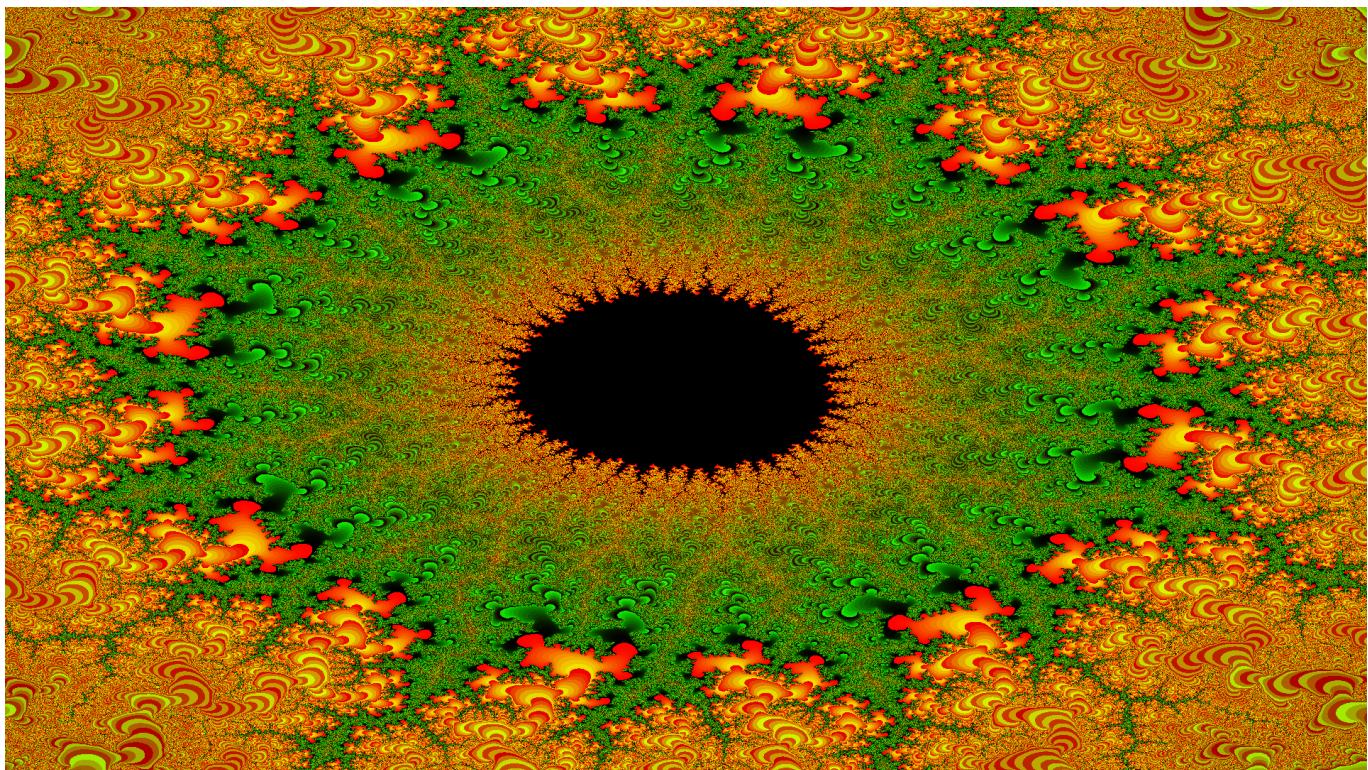
```
int szazalek = ( double ) j / ( double ) magassag * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
}

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

}
```

A program végkimenetele a következő :





5.3. Biomorfok

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_sajat/Mandelbrot/BIOMORF.cpp](https://bhax.thematic_tutorials/bhax_textbook_sajat/Mandelbrot/BIOMORF.cpp)

A biomorfokra (a Julia halmazokat rajzoló bug-os programjával) rátaláló Clifford Pickover azt hitte természeti törvényre bukkant: https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf (lásd a 2307. oldal aljától).

A különbség a **Mandelbrot halmaz** és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

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

        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam

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

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

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

        ++iteracio;
    }
```

Ezzel szemben a Julia halmazos csipetben a `cc` nem változik, hanem minden vizsgált z rácpontra ugyanaz.

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon
    for ( int k = 0; k < szelessseg; ++k )
    {
        double reZ = a + k * dx;
        double imZ = d - j * dy;
        std::complex<double> z_n ( reZ, imZ );

        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {
            z_n = std::pow(z_n, 3) + cc;
            if (std::real ( z_n ) > R || std::imag ( z_n ) > R)
            {
                iteracio = i;
                break;
            }
        }
    }
}
```

A bimorfos algoritmus pontos megismeréséhez ezt a cikket javasoljuk: https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf. Az is jó gyakorlat, ha magából ebből a cikkből from scratch kódoljuk be a sajátunkat, de mi a királyi úton járva a korábbi **Mandelbrot halmazt** kiszámoló forrásunkat módosítjuk. Viszont a program változónak elnevezését összhangba hozzuk a közlemény jelöléseivel:

```
// Forditas:
// g++ biomorf.cpp -lpng -O3 -o biomorf
// Futtatas:
// ./biomorf bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps biomorf.cpp -o biomorf.cpp.pdf -1 --line-numbers=1 --left-footer="←
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
// color
// ps2pdf biomorf.cpp.pdf biomorf.cpp.pdf
//
```

```
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9\_Iss5\_2305--2315\_Biomorphs\_via\_modified\_iterations.pdf
//

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
```

```

ymax = atof ( argv[8] );
reC = atof ( argv[9] );
imC = atof ( argv[10] );
R = atof ( argv[11] );

}

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

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

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

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

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

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

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

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

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

            z_n = std::pow(z_n, 3) + cc;
            //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
            if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
            {
                iteracio = i;
                break;
            }
        }

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

```

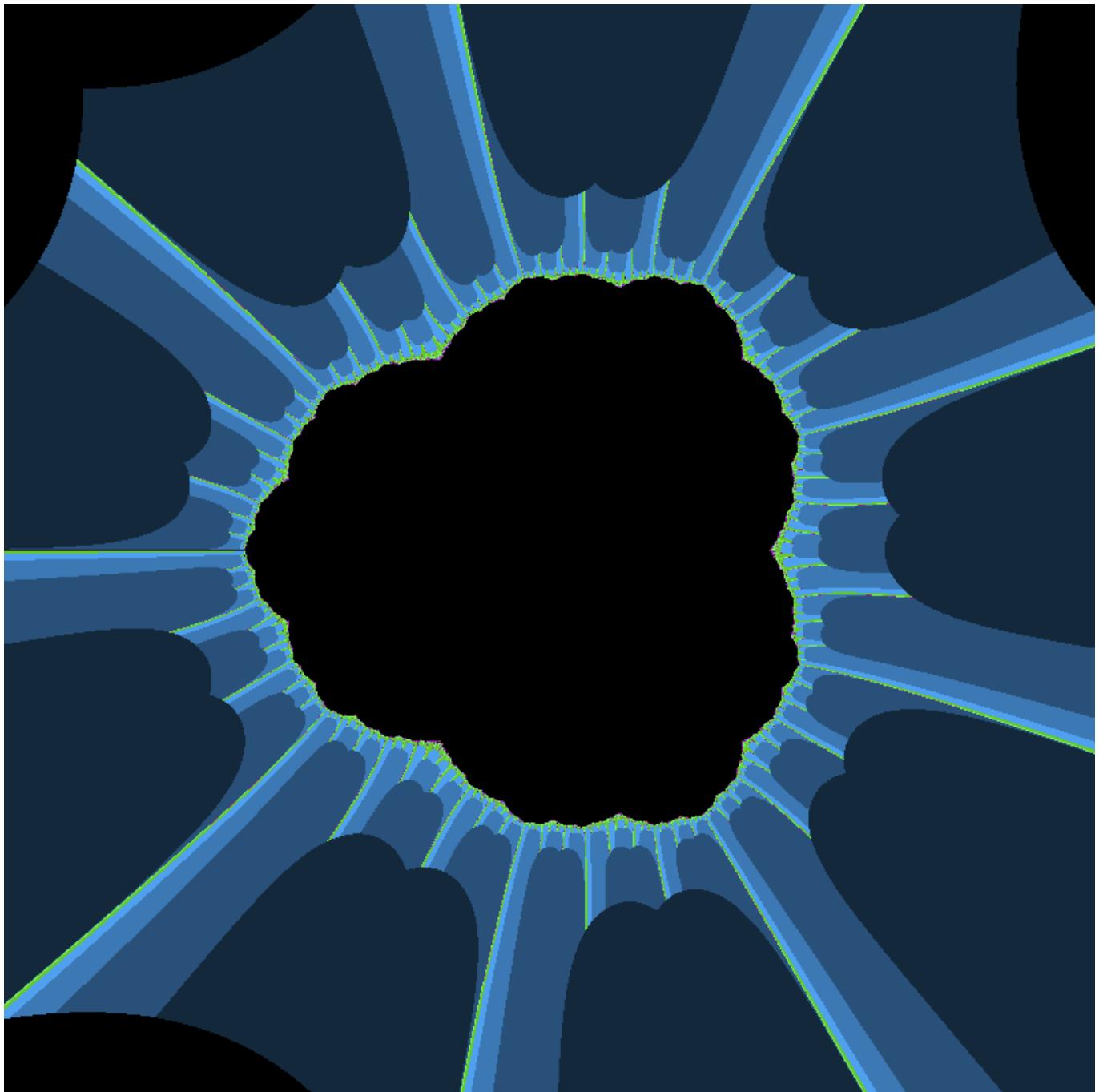
```
}

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

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

}
```

Ahogyan a Mandelbrot halmaznál, itt is egy komplex számsíkon ábrázolható függvényt nézünk meg. A végeredmény pedig a következő:



5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_sajat/Mandelbrot/mandelpngc_60x60_100.cu](https://bhax.thematic_tutorials/bhax_textbook_sajat/Mandelbrot/mandelpngc_60x60_100.cu)

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
```

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063\_01\_parhuzamos\_prog\_linux
//
// https://youtu.be/gvaqijHlRUs
//

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

#include <sys/times.h>
#include <iostream>

#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:
    // most eppen a j. sor k. oszlopban vagyunk

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

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
```

```
// c = (reC, imC) a rács csomópontjainak
// megfelelő komplex szám
reC = a + k * dx;
imC = d - j * dy;
// z_0 = 0 = (reZ, imZ)
reZ = 0.0;
imZ = 0.0;
iteracio = 0;
// z_{n+1} = z_n * z_n + c iterációk
// számítása, amíg |z_n| < 2 vagy még
// nem értük el a 255 iterációt, ha
// viszont elértek, akkor úgy vesszük,
// hogy a kiinduláci c komplex számra
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;
}

return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{
    int j = blockIdx.x;
    int k = blockIdx.y;

    kepadat[j + k * MERET] = mandel (j, k);
}

__global__ void
mandelkernel (int *kepadat)
{
    int tj = threadIdx.x;
    int tk = threadIdx.y;
```

```
int j = blockIdx.x * 10 + tj;
int k = blockIdx.y * 10 + tk;

kepadat[j + k * MERET] = mandel (j, k);

}

void
cudamandel (int kepadat [MERET] [MERET])
{

    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

    // dim3 grid (MERET, MERET);
    // mandelkernel <<< grid, 1 >>> (device_kepadat);

    dim3 grid (MERET / 10, MERET / 10);
    dim3 tgrid (10, 10);
    mandelkernel <<< grid, tgrid >>> (device_kepadat);

    cudaMemcpy (kepadat, device_kepadat,
               MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
    cudaFree (device_kepadat);

}

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

    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    if (argc != 2)
    {
        std::cout << "Használat: ./mandelpngc fajlnev";
        return -1;
    }

    int kepadat [MERET] [MERET];

    cudamandel (kepadat);

    png::image < png::rgb_pixel > kep (MERET, MERET);

    for (int j = 0; j < MERET; ++j)
```

```
{  
    //sor = j;  
    for (int k = 0; k < MERET; ++k)  
{  
        kep.set_pixel (k, j,  
            png::rgb_pixel (255 -  
                (255 * kepadat[j][k]) / ITER_HAT,  
                255 -  
                (255 * kepadat[j][k]) / ITER_HAT,  
                255 -  
                (255 * kepadat[j][k]) / ITER_HAT));  
    }  
}  
kep.write (argv[1]);  
  
std::cout << argv[1] << " mentve" << std::endl;  
  
times (&tmsbuf2);  
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime  
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;  
  
delta = clock () - delta;  
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;  
}
```

A CUDA egy párhuzamos számítástechnikai platform és programozási modell, amelyet az Nvidia fejlesztett ki a saját GPU-ján (grafikus feldolgozó egységek). A CUDA lehetővé teszi a fejlesztők számára, hogy gyorsítsák fel a számítási intenzív alkalmazásokat a GPU-k teljesítményének kihasználásával a számítás párhuzamosítható részében. 2003-ban az Ian Buck által vezetett kutatócsoport bemutatta a Brookot, amely az első széles körben elfogadott programozási modell, amely kiterjeszti a C-t az adat-párhuzamos konstrukciókkal. Buck később csatlakozott az Nvidia-hez, és 2006-ban vezette a CUDA elindítását, amely az első kereskedelmi megoldás az általános célú számítástechnikai eszközökön.

A program fordítását a **nvcc mandelpngc.cu -lpng16 -o mandelpngc** parancssal tudjuk megtenni. Futtatását pedig a **./mandelpngc cuda.png** parancssal.

5.5. Mandelbrot nagyító és utazó C++ nyelven

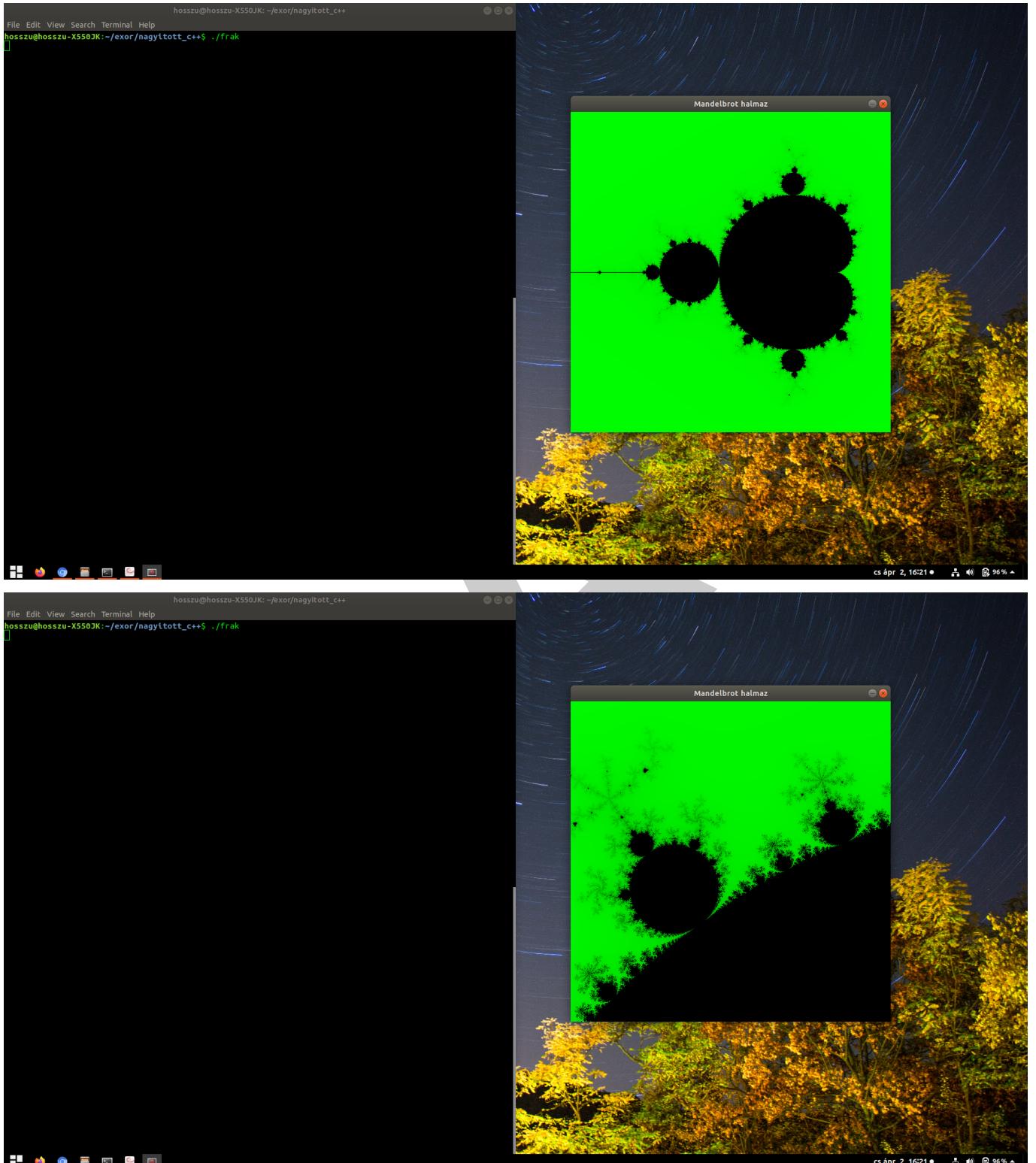
Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteráció bejárta z_n komplex számokat!

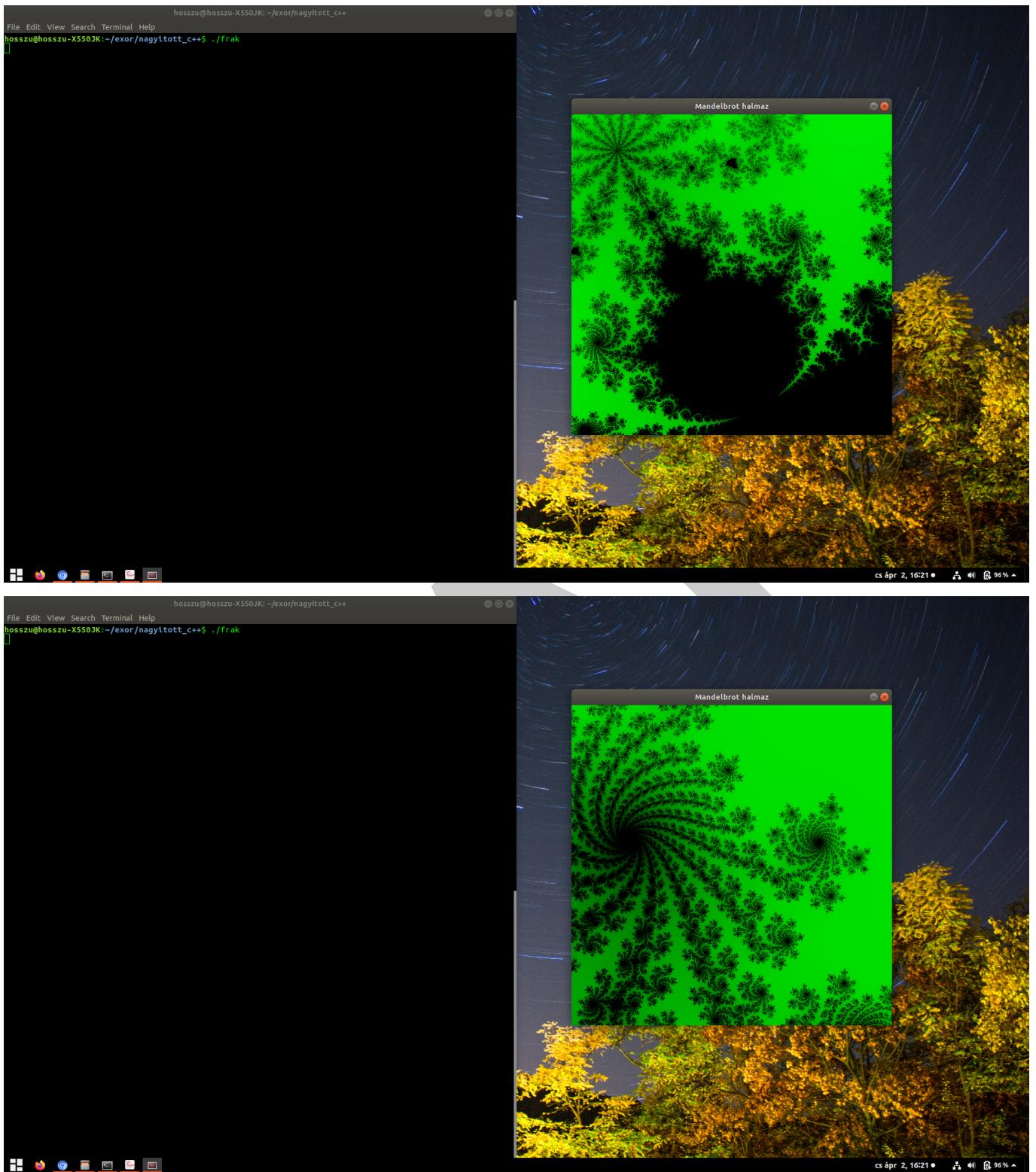
Megoldás videó:

Megoldás forrása:bhax/thematic_tutorials/bhax_textbook_sajat/Mandelbrot/frak

Először is telepítenünk kell a libqt könyvtárat, majd a **qmake** parancssal létrehozunk egy .pro fájlt. Ezek

után pedig elkészítjük a Makefile-t a `make` parancccsal. A program futattását pedig a `./frak` parancccsal tehetjük meg.





5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Mandelbrot/Mandelbrot_nagyitot.html

java

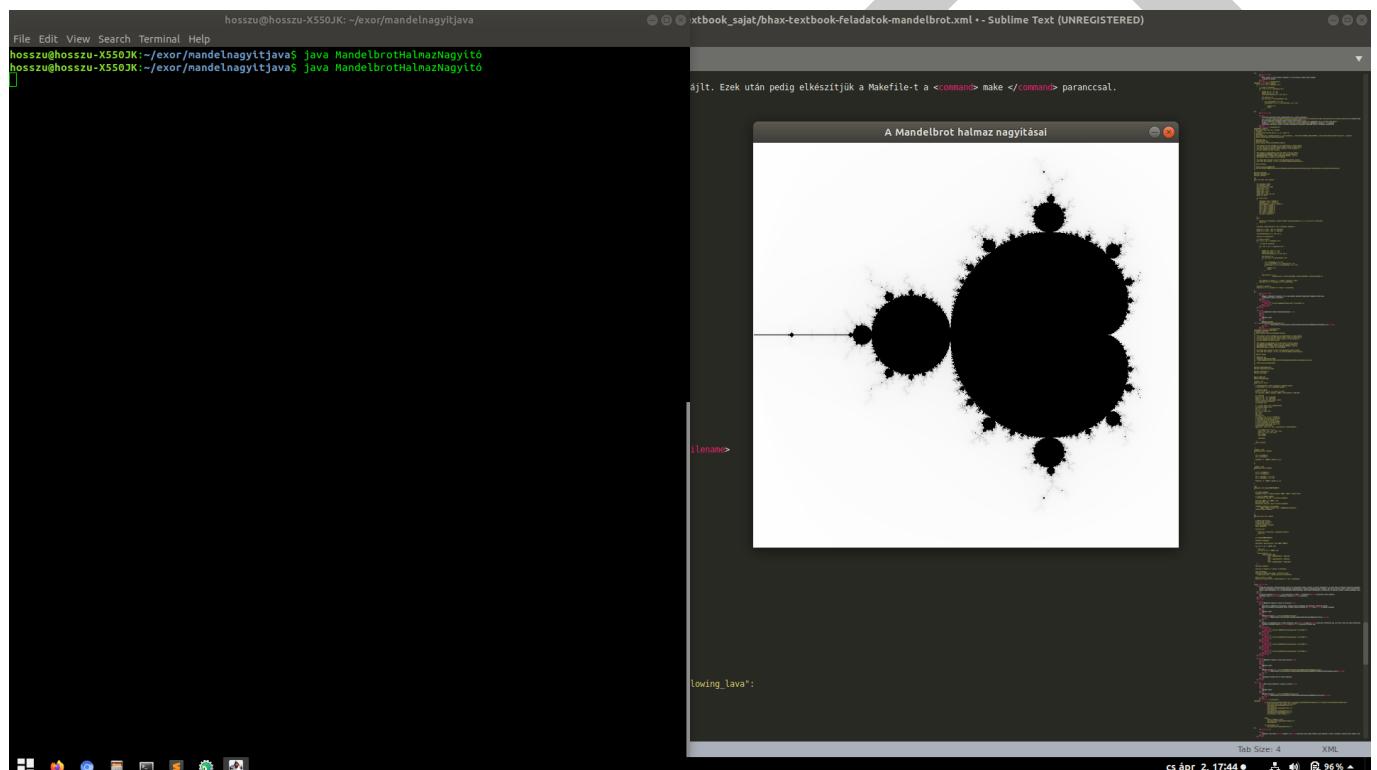
A programunk hasonló mint az előző programunk. A programot a **javac MandelbrotHalmazNagyító.java** parancssal fordítjuk, majd a futtatni a **java MandelbrotHalmazNagyító** parancssal tudjuk.

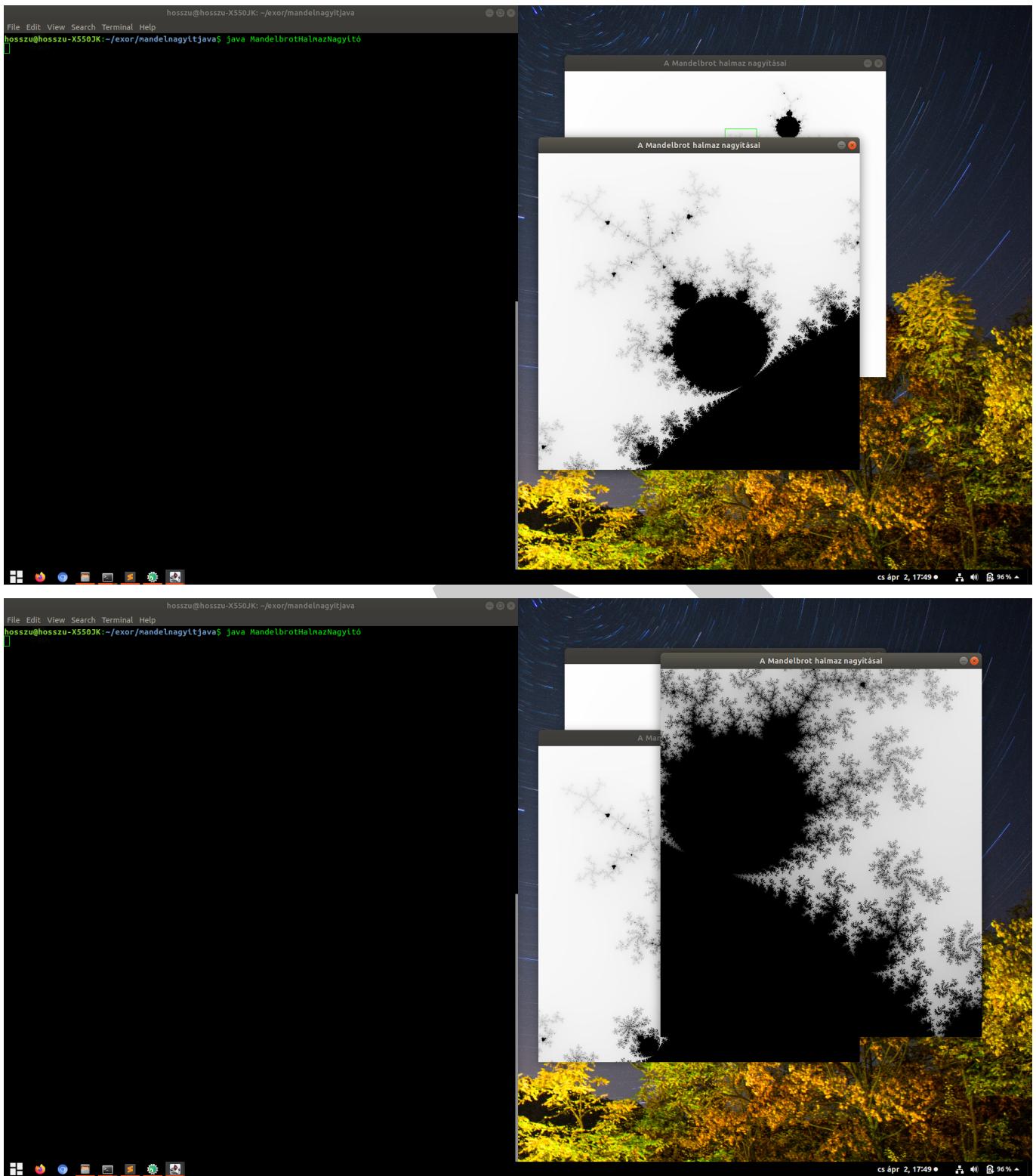
Az **s** billentyű lenyomásával egy felvétel kép készül az aktív ablakban számolt fraktálról.

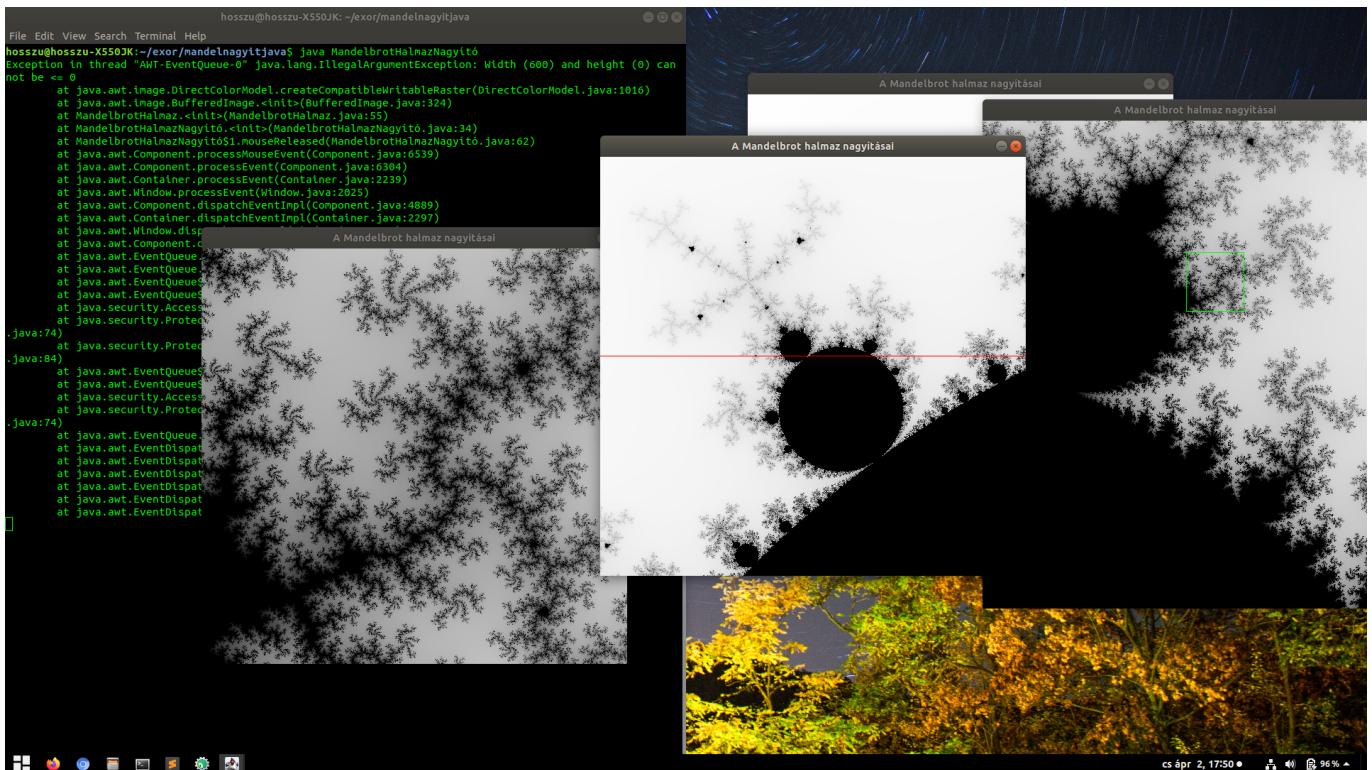
Az **n** billentyű lenyomásával növeljük a halmaz kiszámolásának pontosságát, 256-al megnöveljük az iterációk számát.

Az **m** billentyű lenyomásával nagyobb ugrással növeljük a halmaz kiszámolásának pontosságát, 10*256-al megnöveljük az iterációk számát.

Az **egérmutató bal gombjával** kijelöljük a nagyítandó terület bal felső sarkát, az egér vonszolásával megadjuk a terület nagyságát, az egérgomb felengedésére új ablakban megkezdődik a kijelölt terület nagyítása.







5.7. Vörös Pipacs Pokol/fel a láváig és vissza

Megoldás videó: <https://youtu.be/IzouIU5YFc0>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Mandelbrot/lava.py

```
if nbr[self.elotteidx+9]=="flowing_lava" or nbr[self. ←
elotteidxb+9]=="flowing_lava" or nbr[self.elotteidxj ←
+9]=="flowing_lava":
    print ("Na b*szki, jön a láva. Uccu lefele!")
    self.agent_host.sendCommand("move -1")
    time.sleep(.1)
    self.agent_host.sendCommand("move -1")
    time.sleep(.1)
    self.agent_host.sendCommand("turn 1")
    self.agent_host.sendCommand("turn 1")
    self.fordulas = self.fordulas + 1

else:
    #print ("Szabad az ut!")
    self.agent_host.sendCommand("jumpmove 1")
    time.sleep(.4)

if self.fordulas > 0:
```

```
self.agent_host.sendCommand("move 1")
```

Láthatjuk, hogy Steve **jumpmove 1** parancsal megy szépen felfele. Amint meglátja a lávát a közelében, gyorsan hátrál kettőt, majd 180°-os fordulatot véve megindul lefelé.



6. fejezet

Helló, Welch!

Összefoglaló videó: <https://youtu.be/As50VuMNS1o>

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_sajat/Welch](https://github.com/bhax/thematic_tutorials/tree/main/bhax_textbook_sajat/Welch),

A programunk C++-ban, ami három részből áll:

```
#ifndef POLARGEN__H
#define POLARGEN__H

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

class PolarGen
{
public:
    PolarGen ()
    {
        nincsTarolt = true;
        std::srand (std::time (NULL));
    }
    ~PolarGen ()
    {
    }
    double kovetkezo ();
```

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

Láthatjuk, hogy elsősorban egy header fájl-ra lesz szükségünk.

Véletlen szám sorsoláshoz szükség van bizonyos előre megírt kódokra, melyeket #include-olni kell ahhoz, hogy a feladatot megoldhassuk. A kódunk elején a sorsolás egyik részét végző rand() függvény használatához a következő sort kell beilleszteni a kódunk elejére: **#include cstdlib**.

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

A **polargen.cpp** fájlban valósítjuk meg a polártranszformációt. Ehhez **include**-olunk kell az előbbi **polargen.h** header fájlt. Ebben az esetben akkor fogunk új számot készíteni, ha nincs tárolt számunk a tárolt változóban.

```
#include <iostream>
#include "polargen.h"
#include "polargen.cpp"

int
main (int argc, char **argv)
{
    PolarGen pg;

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

    return 0;
}
```

Ezt a fájlt fogjuk fordítani és futtatni. A for ciklus miatt 10x fog lefutni a programunk, amely 10 szám kiiratását fogja eredményezni.

A futtatás után ez fogad minket:



```
File Edit View Search Terminal Help
hosszu@hosszu-X550JK:~/exor$ ./polargenteszt
-0.974983
-1.451013
1.5411
0.234262
-0.564868
-2.17969
0.0121469
-0.16991
1.16903
-1.74453
hosszu@hosszu-X550JK:~/exor$
```

És akkor jöjjön a Java programunk.

```
public class polargen {
    boolean nincsTarolt = true;
    double tarolt;
    public polargen() {
```

```
nincsTarolt = true;

}

public double kovetkezo() {
    if(nincsTarolt) {
        double u1, u2, v1, v2, w;
        do {
            u1 = Math.random();
            u2 = Math.random();

            v1 = 2*u1 - 1;
            v2 = 2*u2 - 1;

            w = v1*v1 + v2*v2;

        } while(w > 1);

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

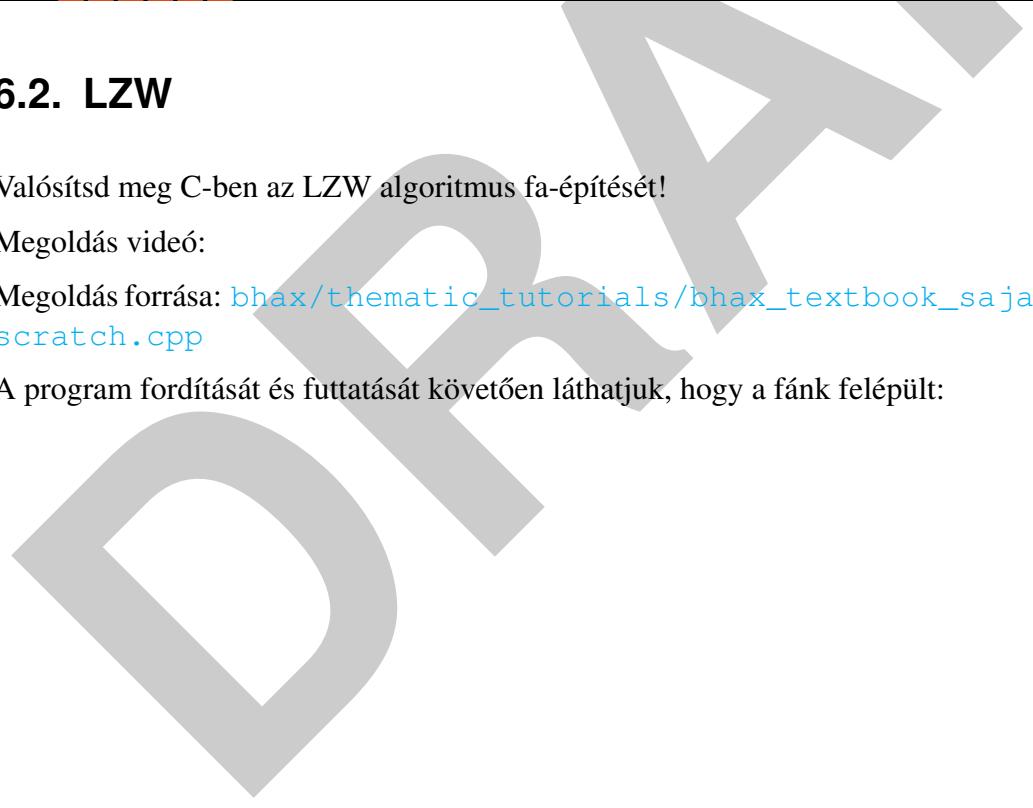
        tarolt = r*v2;
        nincsTarolt = !nincsTarolt;

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

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

A Sun programozói is hasonló képpen oldották meg a random függvényt. A polár metódussal főként az úgynevezett nextGaussian() függvény dolgozik, ami a Random.java fájlból származtatható.

A program futtatása után ez fogad minket:



```
File Edit View Search Terminal Help
hosszu@hosszu-X550JK:~/exor$ java polargen
-0.2124811331122822
-0.4249622613864816
0.2301196691349802
1.1399580754426655
0.8678575385751795
1.8050505323277528
-1.2956465849781216
-0.7053413715046083
-0.056433402855052153
-2.1856115356107813
hosszu@hosszu-X550JK:~/exor$
```

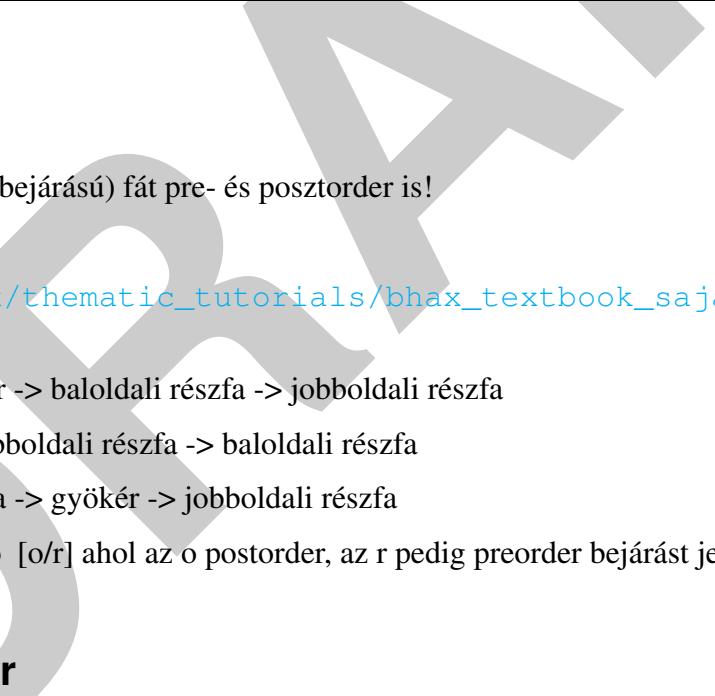
6.2. LZW

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Welch/z3a19a_from_scratch.cpp

A program fordítását és futtatását követően láthatjuk, hogy a fánk felépült:



```
File Edit View Search Terminal Help
hosszu@hosszu-X550JK:~/exor/lzw$ ./z3
BT ctor
-----2 2 0
---1 1 0
----2 2 2
-----7 4 1
-----5 6 0
-----6 5 0
-----5 3 0
BT copy ctor
-----2 2 0
---1 1 0
----2 2 2
-----7 4 1
-----5 6 0
-----6 5 0
-----5 3 0
BT dtor
BT dtor
hosszu@hosszu-X550JK:~/exor/lzw$ 
```

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Welch/z3a18a2_from_scratch.cpp

Preorder bejárás = gyökér -> baloldali részfa -> jobboldali részfa

Postorder = gyökér -> jobboldali részfa -> baloldali részfa

Inorder = bal oldali részfa -> gyökér -> jobboldali részfa

Program futtatása: **./z2 -o** [o/r] ahol az o postorder, az r pedig preorder bejárást jelent.

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Welch/z3a19a_from_scratch.cpp

A már megírt programban, a gyökér, már alápjáraton kompozícióban van a fával.

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Welch/z3a19a_from_scratch.cpp

Mi root-ot alapból pointerrel írtuk meg,

```
Node *root;
```

A gyoker és a fa mutatókat új tárterületre állítjuk, a foglalt területet fel szabadítjuk.

```
template <typename ValueType>
void BinRandTree<ValueType>::deltree(Node *node) {
    if (node) {
        deltreenode->leftChild());
        deltreenode->rightChild());
        delete node;
    }
}
```

A gyökér mutató lett, tehát a gyökér álltal mutatott csomópont gyermekére kell meghívunk a szabadit függvényt. A gyokér által mutatott területet a delete ()-el szabadítjuk fel. Töröljük mindenhol a & címkepző operátort, azaz mostmár nem a gyökér memóriacímét kell átadni, hanem a gyökeret.

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Welch/z3a19a_from_scratch.cpp

A megírt programunkban a **move assign** -on belüli swap végett a mozgató konstruktor, a mozgató értékkadásra van alapozva.

```
BinRandTree(BinRandTree && old) {
    std::cout << "BT move ctor" << std::endl;

    root = nullptr;
    *this = std::move(old);
```

```
        }
BinRandTree & operator = (BinRandTree && old) {
    std::cout << "BT move assign" << std::endl;

    std::swap(old.root, root);
    std::swap(old.treep, treep);

    return *this;
}
```

6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid

Megoldás videó: <https://youtu.be/x9Kjn1GdycQ>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Welch/red_flower.py

```
<Grid name="nbr5x5">
    <min x="-2" y="-2" z="-2"/>
    <max x="2" y="2" z="2"/>
</Grid>
```

Első sorban módosítanunk kell az **nb4tf4i_d.xml** fájlon.

Ezután már mehet is a program és szedhetjük a pipacsokat 5x5x5-ös látásmódban.

```
        world_state = self.agent_host.getWorldState()
        # Loop until mission ends:
        while world_state.is_mission_running:
            if world_state.number_of_observations_since_last_state != 0:

                sensations = world_state.observations[-1].text
                #print("      sensations: ", sensations)
                observations = json.loads(sensations)
                nbr5x5x5 = observations.get("nbr5x5", 0)
                print("      5x5x5 neighborhood of Steve: ", nbr5x5x5)

                if "Yaw" in observations:
                    self.yaw = int(observations["Yaw"])
                if "Pitch" in observations:
                    self.pitch = int(observations["Pitch"])
                if "XPos" in observations:
                    self.x = int(observations["XPos"])
                if "ZPos" in observations:
                    self.z = int(observations["ZPos"])
                if "YPos" in observations:
                    self.y = int(observations["YPos"])
```

```
#print("    Steve's Coords: ", self.x, self.y, self.z)
#print("    Steve's Yaw: ", self.yaw)
#print("    Steve's Pitch: ", self.pitch)

if "LineOfSight" in observations:
    lineOfSight = observations["LineOfSight"]
    self.lookingat = lineOfSight["type"]
#print("    Steve's <): ", self.lookingat)

if self.lookingat == "red_flower":
    print ("Itt a pipacs, hol a pipacs?")
    self.agent_host.sendCommand("move 0")
    time.sleep(.5)
    self.agent_host.sendCommand("attack 1")
    time.sleep(.1)

if self.akadaly == self.uthossz:
    self.agent_host.sendCommand("turn 1")
    time.sleep(.2)
    self.agent_host.sendCommand("move 0")
    self.akadaly = 0
    self.akadaly2 = self.akadaly2 + 1

if self.akadaly2 > 4:
    self.agent_host.sendCommand("jumpstrafe -1")
    time.sleep(.1)
    self.agent_host.sendCommand("strafe -1")
    time.sleep(.1)
    self.akadaly2 = 0
    self.adakaly = 0
    self.uthossz = self.uthossz + 5

self.agent_host.sendCommand("move 1")
time.sleep(.1)
self.akadaly = self.akadaly + 1

world_state = self.agent_host.getWorldState()
```

```
def __init__(self, agent_host):
    self.agent_host = agent_host
    self.x = 0
    self.y = 0
    self.z = 0
    self.yaw = 0
    self.pitch = 0
    self.akadaly = 0
    self.akadaly2 = 0
    self.uthossz = 9
```

Itt deklarálnunk kell néhány változót: a pozíció koordináták a **self.x**, **self.y**, **self.z**. Az irányt a **self yaw** változóban, a nézetet pedig a **self.pitch** változóban fogjuk tárolni.

```
sensations = world_state.observations[-1].text
#print("    sensations: ", sensations)
observations = json.loads(sensations)
nbr5x5x5 = observations.get("nbr5x5", 0)
print("    5x5x5 neighborhood of Steve: ", nbr5x5x5)
```

A json függvény segítségével információt szerzünk a Steve-et körülvevő blokkokról, ezeket természetesen ki is tudjuk iratni a következőképpen: **print("5x5x5 neighborhood of Steve: ", nbr5x5x5)**.

DRAFT

7. fejezet

Helló, Conway!

Összefoglaló videó: https://youtu.be/Vp3_JGcW5Ls

7.1. Hangyszimulációk

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

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Conway/Hangyszim/,

Az alábbi programunk három darab header fájlból áll, és 3 "főfájlból".

A Hangyszim mappába belépve és onnan egy terminál segítségével, először ki kell adnunk a **qmake** parancsot. Ez elkészít nekünk a Makefile-t. Ezután kiadjuk a **make** parancsot, ami lebuildeli nekünk a programot.

Majd végül a **./myrmecologist** parancs kiadásával futtatjuk a programunkat és láthatjuk a "hangyszimulációt".

7.2. Java életjáték

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

Megoldás videó:

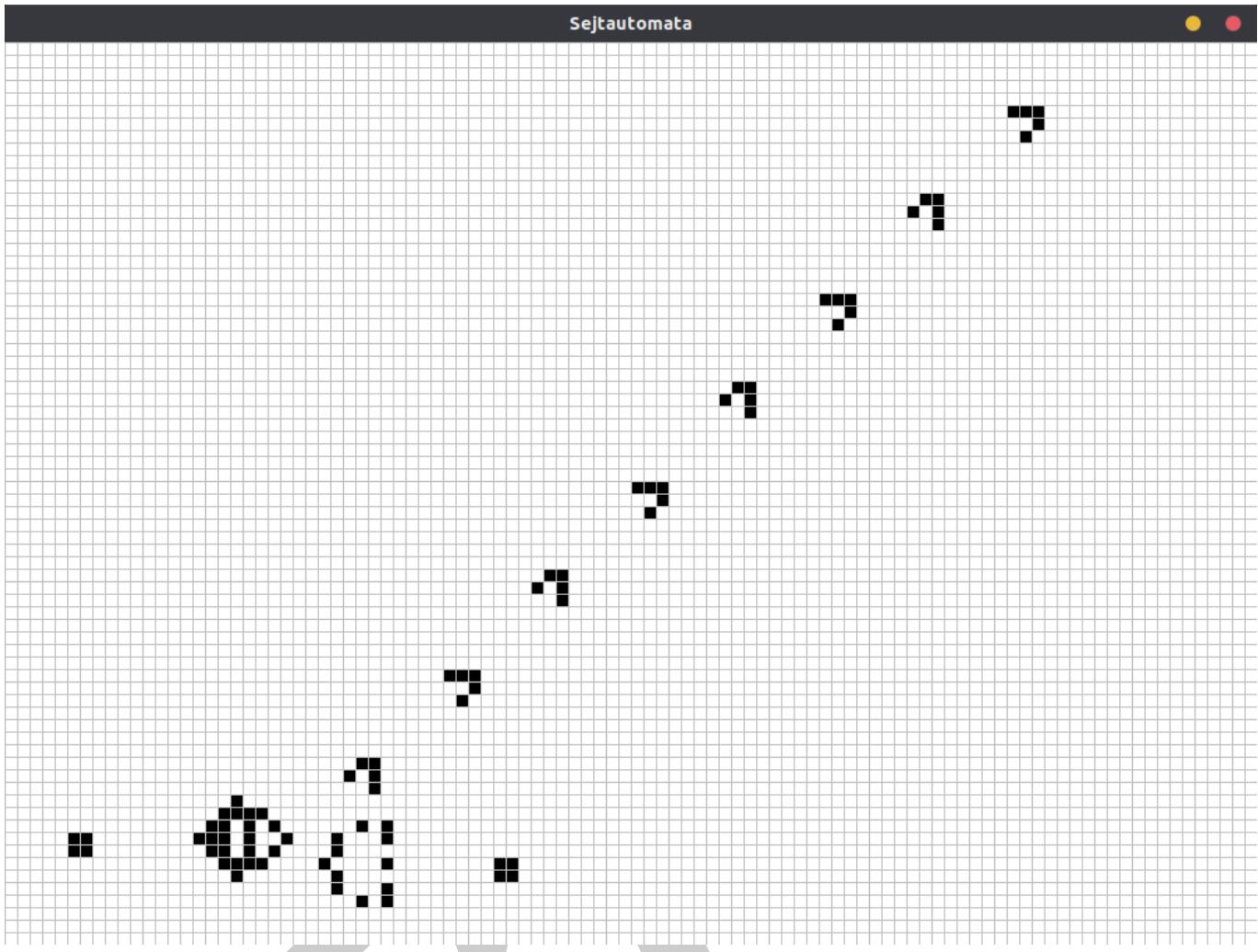
Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_sajat/Conways/Sejtautomatjava](https://github.com/bhax/thematic_tutorials/tree/main/bhax_textbook_sajat/Conways/Sejtautomatjava),

A programot a `javac Sejtautomata.java` parancssal fordítjuk.

Majd futtatni a **java Sejautomata** parancssal tudjuk.

A legismertebb sejtautomaták egyike a John Conway által kifejlesztett életjáték. Ennek a sejtmozaik háttere olyan, mint a kockás füzet: ebben a szerkezetben minden sejtnek nyolc sejtszomszédja van. Az egyes sejtek kétféle állapotban lehetnek: élő vagy halott állapotban. Az idő, ahogy minden egyszerűbb sejtautomatában, diszkrét időegységekben múlik. Időlépések-ként változtathatták állapotukat a sejtek a következő szabályok szerint: Egy olyan sejt helyére, amely halott, de három élő sejtszomszédja van, élő sejt „születik”. Egy olyan sejt, amely élő volt, és két vagy három szomszédja is élő volt, az ilyen sejt életben marad. Az összes többi, másmilyen környezetű sejt halott lesz a következő lépésben. Az életjáték sok érdekes szerkezet mozgását, gyarapodását, vagy elmúlását és sajátos alakzatok tartós fönnyelmaradását tudja szimulálni.

—Wikipedia



7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Conway/Sejtauto/,

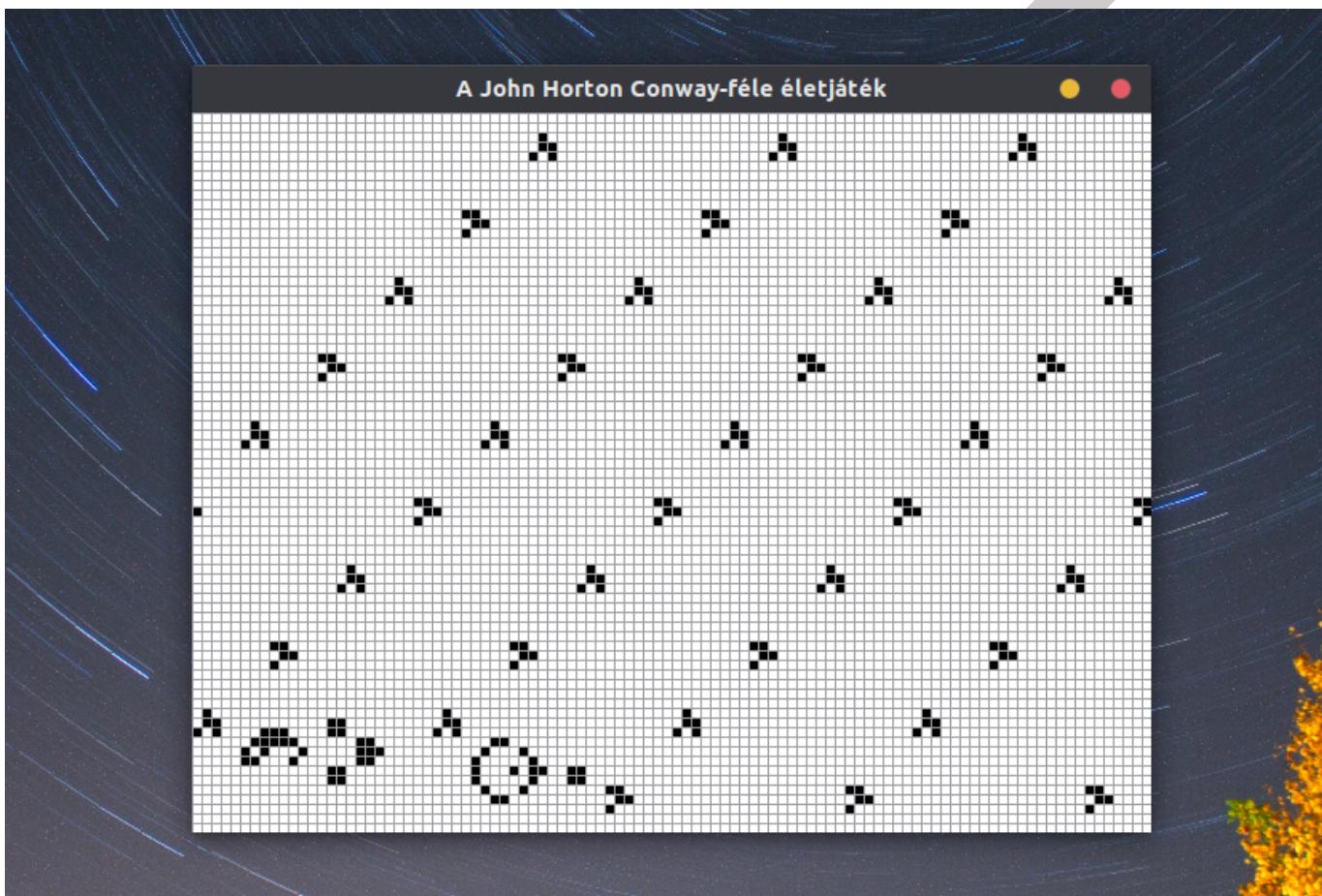
Az életjátékot (angolul: The game of life) John Horton Conway, a Cambridge-i Egyetem matematikusa találta ki. Játékként való megnevezése megtévesztő lehet, mivel „nullszemélyes” játék; és a „játékos” szerepe mindenkorra annyi, hogy megad egy kezdőalakzatot, és azután csak figyeli az eredményt. Matematikai szempontból az ún. sejtautomaták közé tartozik. A játék egyes lépéseinél eredményét számítógép számítja ki (ez elvileg nem szükséges, korlátozottabb mértékben lehet emberi erővel és négyzeteloszlás táblán is játszani, de ehhez türelem szükséges). A „játék”, a felfedezése utáni években, sokak hóbortos szabadidőtöltésévé vált Amerikában, s mint kiderült, komoly matematikai és filozófiai vonatkozásai vannak..

—Wikipedia

A program "elkészítését" a **qmake** parancs segítségével hajtjuk végre.

Először is a könyvtárban, ahol a két header fájl és a két .cpp fájl található, kiadunk egy **qmake-qt4 -project** parancsot. Ez létrehoz nekünk a mappa neve alapján egy .pro fájlt. Ezután a **qmake** parancsot ráküldjük erre a Valami.pro fájlra. Ezzel elkészül a Makefile-unk. Ezt követően a **make** parancssal elkészítjük a futtatható programunkat.

Végül a **./Sejtauto** parancssal futtatjuk a programot.



7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Conway/BrainB/,

BrainB Benchmark szoftver, összeállít egy elméleti profilt, hogy az adott játékos hogyan teljesít abban a szituációbanamikor "elveszti" a karakterét egy intenzívebb pillanatban, pld vizuális efektek ki takarják egymást, nagy a kavarodás stb stb. A feladatun a szimuláció alatt, hogy az egér kurzort a 'Samu Entropy' karakteren tartson, pontosabban a doboz közepében lévő körön.

Kezdjünk is neki az előkészületeknek:

```
2000 git clone https://github.com/nbatfai/esport-talent-search  
2001 cd esport-talent-search
```

```
2002 sudo apt-get install opencv-data
2003 sudo apt-get install libopencv-dev
2004 mkdir build && cd build
2005 qmake ..
2006 make
2007 ./BrainB
```

És akkor teszteljük magunkat!



7.5. Vörös Pipacs Pokol/19 RF

Megoldás videó: <https://www.youtube.com/watch?v=UbiBkkLx1Fk>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_sajat/Conway/Red_flower_mining5.py,

Tutorált: Zselenák Flórián

Ebben a megoldásban a saját RFH-3as döntőbejutó kódunkat mutatnám be:

A mi ágensünk 38 pipacsot szedett össze a legutóbbi fejlesztés alapján. Ennek működését írnám le tömören. Először is a játék meghívásakor meghívódik a run() függvény. Ez addig fut, amíg a mission véget nem ér. Innen hívódik meg az action() függvény, ami folyamatosan lekéri az Observation From Gridet. A körülvevő objektumok alapján eldöntjük mit csináljon, minden blokkot figyelemmel kísérünk, és ahol pipacs van, felszedi(még ha egy szinttel lejjebb van, és a 3x3x3-as látómezőbe szerepel) majd visszalép a helyére és folyamatosan halad előre, illetve sarkot fordul, amíg nem talál pipacsot. Ha talált, lejjebb megy egy szinttel.

```
from __future__ import print_function
```

```
# ←
-----
# Copyright (c) 2016 Microsoft Corporation
#
# Permission is hereby granted, free of charge, to any person obtaining a ←
# copy of this software and
# associated documentation files (the "Software"), to deal in the Software ←
# without restriction,
# including without limitation the rights to use, copy, modify, merge, ←
# publish, distribute,
# sublicense, and/or sell copies of the Software, and to permit persons to ←
# whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included ←
# in all copies or
# substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS ←
# OR IMPLIED, INCLUDING BUT
# NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A ←
# PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE ←
# LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR ←
# OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN ←
# THE SOFTWARE.
# ←
-----

# Tutorial sample #2: Run simple mission using raw XML

# Added modifications by Norbert Bátfa (nb4tf4i) batfai.norbert@inf.unideb ←
# .hu, mine.ly/nb4tf4i.1
# 2018.10.18, https://bhaxor.blog.hu/2018/10/18/malmo_minecraft
# 2020.02.02, NB4tf4i's Red Flowers, http://smartcity.inf.unideb.hu/~norbi/ ←
# NB4tf4iRedFlowerHell
# 2020.03.02, https://github.com/nbatfai/RedFlowerHell
# 2020.03.07, "_smartSteve": nof_turn (number of turns) is replaced by the ←
# dict self.collectedFlowers
# 2020.03.11, "_bu": bottom up, s4v3: https://youtu.be/VP0kfvRYD1Y

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

```
import random
import json
import math

if sys.version_info[0] == 2:
    sys.stdout = os.fdopen(sys.stdout.fileno(), 'w', 0) # flush print ←
        output immediately
else:
    import functools
    print = functools.partial(print, flush=True)

# Create default Malmo objects:

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

# -- set up the mission --
missionXML_file='nb4tf4i_d_RFHIII.xml'
with open(missionXML_file, 'r') as f:
    print("NB4tf4i's Red Flowers (Red Flower Hell) - DEAC-Hackers Battle ←
          Royale Arena\n")
    print("NB4tf4i vörös pipacsai (Vörös Pipacs Pokol) - DEAC-Hackers ←
          Battle Royale Arena\n")
    print("The aim of this first challenge, called nb4tf4i's red flowers, ←
          is to collect as many red flowers as possible before the lava flows ←
          down the hillside.\n")
    print("Ennek az első, az nb4tf4i vörös virágai nevű kihívásnak a célja ←
          összegyűjteni annyi piros virágot, amennyit csak lehet, mielőtt a ←
          láva lefolyik a hegyoldalon.\n")
    print("Norbert Bátfai, batfai.norbert@inf.unideb.hu, https://arato.inf. ←
          unideb.hu/batfai.norbert/\n")
    print("Loading mission from %s" % missionXML_file)
    mission_xml = f.read()
    my_mission = MalmoPython.MissionSpec(mission_xml, True)
    my_mission.drawBlock( 0, 0, 0, "lava")

class Hourglass:
    def __init__(self, charSet):
        self.charSet = charSet
        self.index = 0
    def cursor(self):
        self.index=(self.index+1)%len(self.charSet)
```

```
        return self.charSet[self.index]

hg = Hourglass('|/-\|')

class Steve:
    def __init__(self, agent_host):
        self.agent_host = agent_host

        for i in range(81):
            self.agent_host.sendCommand("jumpmove 1")

        self.agent_host.sendCommand("look 1")
        self.agent_host.sendCommand("look 1")

        self.x = 0
        self.y = 0
        self.z = 0
        self.yaw = 0
        self.pitch = 0

        self.front_of_me_idx = 0
        self.front_of_me_idxr = 0
        self.front_of_me_idxl = 0
        self.right_of_me_idx = 0
        self.left_of_me_idx = 0

        self.nof_red_flower = 0
        self.lookingat = ""
        self.attackLvl = 0
        self.trap = 0

        self.collectedFlowers = {}
        for i in range(100):
            self.collectedFlowers[i] = False

        self.collectedFlowers[1] = True
        self.collectedFlowers[2] = True

    def checkInventory(self, observations):
        for i in range(2):
            hotbari = 'Hotbar_'+str(i)+'_item'
            hotbars = 'Hotbar_'+str(i)+'_size'
            slot0_contents = observations.get(hotbari, "")
            if slot0_contents == "red_flower":
                slot0_size = observations.get(hotbars, "")
                if self.nof_red_flower < slot0_size :
                    self.nof_red_flower = slot0_size
                    #print("          A RED FLOWER IS MINED AND PICKED UP ←")
                    #
                    #print("Steve's lvl: ", self.y, "Flower lvl ←"

```

```
: ", self.attackLvl)
self.collectedFlowers[self.attackLvl] = True
if self_LVLUp(observations.get("nbr3x3", 0)):
    return True

def pickUp(self):
    print(" Felvesz")
    self.agent_host.sendCommand( "attack 1" )
    time.sleep(.1)
    self.agent_host.sendCommand( "move 0" )
    time.sleep(.5)
    self.agent_host.sendCommand("strafe -1")
    time.sleep(.1)

    self.attackLvl = self.y
    self.trap = 0

def lvlUp(self, nbr):
    if self.collectedFlowers[self.y]:
        self_LVLDown(nbr)
        return True
    else:
        return False

def idle(self, delay):
    #print("      SLEEPING for ", delay)
    time.sleep(delay)

def isInTrap(self, nbr):
    dc = 0
    nbri = [9,10,11,12,14,15,16,17]
    for i in range(1, len(nbri)):
        if nbr[nbri[i]]=="dirt" :
            dc = dc + 1
    return dc > 5

def turnFromWall(self, nbr):
    if (nbr[self.left_of_me_idx+9]=="air") and (nbr[self. ↵
    front_of_me_idx]=="dirt"):
        print(" Fordulás")
        self.agent_host.sendCommand( "turn -1" )
        time.sleep(.1)
        self.agent_host.sendCommand("strafe -1")
        time.sleep(.1)
    self.trap = self.trap+1

def lvlDown(self, nbr):
    if (nbr[self.left_of_me_idx+9]=="air" or (nbr[self.right_of_me_idx ↵
    ]=="dirt")):
        self.movement2()
```

```
def movement2(self):
    self.agent_host.sendCommand( "move 1")
    time.sleep(0.01)
    self.agent_host.sendCommand( "strafe -1" )
    time.sleep(.1)

def calcNbrIndex(self):
    if self.yaw >= 180-22.5 and self.yaw <= 180+22.5 :
        self.front_of_me_idx = 1
        self.front_of_me_idxr = 2
        self.front_of_me_idxl = 0
        self.right_of_me_idx = 5
        self.left_of_me_idx = 3
    elif self.yaw >= 180+22.5 and self.yaw <= 270-22.5 :
        self.front_of_me_idx = 2
        self.front_of_me_idxr = 5
        self.front_of_me_idxl = 1
        self.right_of_me_idx = 8
        self.left_of_me_idx = 0
    elif self.yaw >= 270-22.5 and self.yaw <= 270+22.5 :
        self.front_of_me_idx = 5
        self.front_of_me_idxr = 8
        self.front_of_me_idxl = 2
        self.right_of_me_idx = 7
        self.left_of_me_idx = 1
    elif self.yaw >= 270+22.5 and self.yaw <= 360-22.5 :
        self.front_of_me_idx = 8
        self.front_of_me_idxr = 7
        self.front_of_me_idxl = 5
        self.right_of_me_idx = 6
        self.left_of_me_idx = 2
    elif self.yaw >= 360-22.5 or self.yaw <= 0+22.5 :
        self.front_of_me_idx = 7
        self.front_of_me_idxr = 6
        self.front_of_me_idxl = 8
        self.right_of_me_idx = 3
        self.left_of_me_idx = 5
    elif self.yaw >= 0+22.5 and self.yaw <= 90-22.5 :
        self.front_of_me_idx = 6
        self.front_of_me_idxr = 3
        self.front_of_me_idxl = 7
        self.right_of_me_idx = 0
        self.left_of_me_idx = 8
    elif self.yaw >= 90-22.5 and self.yaw <= 90+22.5 :
        self.front_of_me_idx = 3
        self.front_of_me_idxr = 0
        self.front_of_me_idxl = 6
        self.right_of_me_idx = 1
        self.left_of_me_idx = 7
```

```
        elif self.yaw >= 90+22.5 and self.yaw <= 180-22.5 :
            self.front_of_me_idx = 0
            self.front_of_me_idxr = 1
            self.front_of_me_idxl = 3
            self.right_of_me_idx = 2
            self.left_of_me_idx = 6
        else:
            print("There is great disturbance in the Force...")

def whatISee(self, observations):
    self.lookingat = "NOTHING"
    if "LineOfSight" in observations:
        lineOfSight = observations["LineOfSight"]
        self.lookingat = lineOfSight["type"]

def whatMyPos(self, observations):
    if "Yaw" in observations:
        self.yaw = int(observations["Yaw"])
    if "Pitch" in observations:
        self.pitch = int(observations["Pitch"])
    if "XPos" in observations:
        self.x = int(observations["XPos"])
    if "ZPos" in observations:
        self.z = int(observations["ZPos"])
    if "YPos" in observations:
        self.y = int(observations["YPos"])

def run(self):
    world_state = self.agent_host.getWorldState()
    # Loop until mission ends:
    while world_state.is_mission_running:

        act = self.action(world_state)
        if not act:
            self.idle(.017)
        world_state = self.agent_host.getWorldState()

def action(self, world_state):
    for error in world_state.errors:
        print("Error:", error.text)

    if world_state.number_of_observations_since_last_state == 0:
        #print("    NO OBSERVATIONS NO ACTIONS")
        return False

    input = world_state.observations[-1].text
    observations = json.loads(input)
    nbr = observations.get("nbr3x3", 0)
    #print(observations)
```

```
self.whatMyPos(observations)
#print("\r      Steve's Coords: ", self.x, self.y, self.z, end=' ')
#print("      Steve's Yaw: ", self.yaw)
#print("      Steve's Pitch: ", self.pitch)

self.checkInventory(observations)
#print("Number of flowers: ", self.nof_red_flower)

self.whatISee(observations)
#print("      Steve's <): ", self.lookingat)

self.calcNbrIndex()

if self.isInTrap(nbr) :
    print(" Ugrás")
    self.agent_host.sendCommand( "jumpstrafe -1" )
    time.sleep(.2)
    return True

if self.trap > 5:
    print(" Ezen a szinten már nincs pipacs")
    self.lvlDown(nbr)
    self.agent_host.sendCommand("strafe -1")
    time.sleep(.1)
    self.trap = 0

for i in range(9):
    if nbr[i]=="red_flower" or nbr[i+9]=="red_flower":
        print(" Az ott egy pipacs?: ", i, " LEVEL ", self.y - 1)
        if i == self.front_of_me_idxl :
            print(" Balra lép ")
            self.agent_host.sendCommand("move -1")
            time.sleep(.1)
            self.agent_host.sendCommand( "strafe -1" )
            time.sleep(.1)
            self.pickUp()
            return True
        elif i == self.left_of_me_idx :
            print(" Balra lép")
            self.agent_host.sendCommand("move -1")
            time.sleep(.1)
            self.agent_host.sendCommand("move -1")
            time.sleep(.1)
            self.agent_host.sendCommand( "strafe -1" )
            time.sleep(.1)
            self.pickUp()
            return True
        elif i == 4 :
            self.red_flower_is_mining = True
```

```
        self.pickUp()
        return True

    if nbr[self.front_of_me_idx+9]!="air" and nbr[self.front_of_me_idx ↵
        +9]!="red_flower":
        #print("          THERE ARE OBSTACLES IN FRONT OF ME ",   nbr[self ↵
        .front_of_me_idx],  end=' ')
        self.turnFromWall(nbr)

    else:
        #print("          THERE IS NO OBSTACLE IN FRONT OF ME", end=' ')

        if nbr[self.front_of_me_idx]=="dirt":
            self.agent_host.sendCommand( "move 1" )
            time.sleep(0.013)
            self.agent_host.sendCommand( "move 1" )
            time.sleep(0.013)

    return True

num_repeats = 1
for ii in range(num_repeats):

    my_mission_record = MalmoPython.MissionRecordSpec()

    # Attempt to start a mission:
    max_retries = 6
    for retry in range(max_retries):
        try:
            agent_host.startMission( my_mission, my_mission_record )
            break
        except RuntimeError as e:
            if retry == max_retries - 1:
                print("Error starting mission:", e)
                exit(1)
            else:
                print("Attempting to start the mission:")
                time.sleep(2)

    # Loop until mission starts:
    print("  Waiting for the mission to start")
    world_state = agent_host.getWorldState()

    while not world_state.has_mission_begun:
        print("\r"+hg.cursor(), end="")
        time.sleep(0.15)
        world_state = agent_host.getWorldState()
```

```
for error in world_state.errors:  
    print("Error:",error.text)  
  
print("NB4tf4i Red Flower Hell running\n")  
steve = Steve(agent_host)  
steve.run()  
print("Number of flowers: "+ str(steve.nof_red_flower))  
time.sleep(3)  
  
print("Mission ended")  
# Mission has ended.
```

DRAFT

8. fejezet

Helló, Schwarzenegger!

Összefoglaló videó:<https://youtu.be/UbiBkkLx1Fk>

8.1. Szoftmax Py MNIST

Python

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

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa...
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Ma a Goggle által létrehozott Tensorflow-val fogunk megismerkedni, amit a tanulás fejlesztésének érdekkében hoztak létre. Tudni kell, hogy a Tensorflow a Google Brain's második generációs rendszere. A **Version 1.0.0** -t 2017. Február 11.-én jelentették be. A Tensorflow képes futtatni többmagos CPU-kon, illetve GPU-kon (persze az opcionális GPU-val, például CUDA kártyával). A tensorflow elérhető a 64 bites rendszereken (Linux, MacOs, Windows), és a mobil platformokkal is kompatibilis.

```
Import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)])

prediction = model(x_train[:1]).numpy()
prediction

tf.nn.softmax(prediction).numpy()
```

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

loss_fn(y_train[:1], prediction).numpy()

model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test, verbose=2)

probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])

probability_model(x_test[:5])
```

A program futtatását követően: **python3 mnist.py**. Az első eredményünk 5-ös **epochs**-nál:

```
Train on 60000 samples
Epoch 1/5
60000/60000 [=====] - 3s 50us/sample - loss: ←
    0.2945 - accuracy: 0.9142
Epoch 2/5
60000/60000 [=====] - 3s 47us/sample - loss: ←
    0.1434 - accuracy: 0.9578
Epoch 3/5
60000/60000 [=====] - 3s 47us/sample - loss: ←
    0.1081 - accuracy: 0.9672
Epoch 4/5
60000/60000 [=====] - 3s 51us/sample - loss: ←
    0.0884 - accuracy: 0.9733
Epoch 5/5
60000/60000 [=====] - 3s 56us/sample - loss: ←
    0.0740 - accuracy: 0.9766
10000/10000 - 0s - loss: 0.0723 - accuracy: 0.9791
```

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Itt olyan képeket készíthetünk a programunkkal, amit akkor látunk, amikor valamilyen pszichoaktív szer (kábítószer) hatása alatt látunk, például az LSD.

```
'''
#Deep Dreaming in Keras.
```

```
Run the script with:  
```python  
python deep_dream.py path_to_your_base_image.jpg prefix_for_results
```  
e.g.:  
```python  
python deep_dream.py img/mypic.jpg results/dream
```  
```  
from __future__ import print_function

from keras.preprocessing.image import load_img, save_img, img_to_array
import numpy as np
import scipy
import argparse

from keras.applications import inception_v3
from keras import backend as K

parser = argparse.ArgumentParser(description='Deep Dreams with Keras.')
parser.add_argument('base_image_path', metavar='base', type=str,
 help='Path to the image to transform.')
parser.add_argument('result_prefix', metavar='res_prefix', type=str,
 help='Prefix for the saved results.')

args = parser.parse_args()
base_image_path = args.base_image_path
result_prefix = args.result_prefix

These are the names of the layers
for which we try to maximize activation,
as well as their weight in the final loss
we try to maximize.
You can tweak these setting to obtain new visual effects.
settings = {
 'features': {
 'mixed2': 0.2,
 'mixed3': 0.5,
 'mixed4': 2.,
 'mixed5': 1.5,
 },
}

def preprocess_image(image_path):
 # Util function to open, resize and format pictures
 # into appropriate tensors.
 img = load_img(image_path)
 img = img_to_array(img)
 img = np.expand_dims(img, axis=0)
```

```
img = inception_v3.preprocess_input(img)
return img

def deprocess_image(x):
 # Util function to convert a tensor into a valid image.
 if K.image_data_format() == 'channels_first':
 x = x.reshape((3, x.shape[2], x.shape[3]))
 x = x.transpose((1, 2, 0))
 else:
 x = x.reshape((x.shape[1], x.shape[2], 3))
 x /= 2.
 x += 0.5
 x *= 255.
 x = np.clip(x, 0, 255).astype('uint8')
 return x

K.set_learning_phase(0)

Build the InceptionV3 network with our placeholder.
The model will be loaded with pre-trained ImageNet weights.
model = inception_v3.InceptionV3(weights='imagenet',
 include_top=False)
dream = model.input
print('Model loaded.')

Get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

Define the loss.
loss = K.variable(0.)
for layer_name in settings['features']:
 # Add the L2 norm of the features of a layer to the loss.
 if layer_name not in layer_dict:
 raise ValueError('Layer ' + layer_name + ' not found in model.')
 coeff = settings['features'][layer_name]
 x = layer_dict[layer_name].output
 # We avoid border artifacts by only involving non-border pixels in the ←
 # loss.
 scaling = K.prod(K.cast(K.shape(x), 'float32'))
 if K.image_data_format() == 'channels_first':
 loss += coeff * K.sum(K.square(x[:, :, 2:-2, 2:-2])) / scaling
 else:
 loss += coeff * K.sum(K.square(x[:, 2:-2, 2:-2, :])) / scaling

Compute the gradients of the dream wrt the loss.
grads = K.gradients(loss, dream)[0]
Normalize gradients.
grads /= K.maximum(K.mean(K.abs(grads)), K.epsilon())
```

```
Set up function to retrieve the value
of the loss and gradients given an input image.
outputs = [loss, grads]
fetch_loss_and_grads = K.function([dream], outputs)

def eval_loss_and_grads(x):
 outs = fetch_loss_and_grads([x])
 loss_value = outs[0]
 grad_values = outs[1]
 return loss_value, grad_values

def resize_img(img, size):
 img = np.copy(img)
 if K.image_data_format() == 'channels_first':
 factors = (1, 1,
 float(size[0]) / img.shape[2],
 float(size[1]) / img.shape[3])
 else:
 factors = (1,
 float(size[0]) / img.shape[1],
 float(size[1]) / img.shape[2],
 1)
 return scipy.ndimage.zoom(img, factors, order=1)
```

```
def gradient_ascent(x, iterations, step, max_loss=None):
 for i in range(iterations):
 loss_value, grad_values = eval_loss_and_grads(x)
 if max_loss is not None and loss_value > max_loss:
 break
 print('..Loss value at', i, ':', loss_value)
 x += step * grad_values
 return x
```

```
"""Process:
- Load the original image.
- Define a number of processing scales (i.e. image shapes),
 from smallest to largest.
- Resize the original image to the smallest scale.
- For every scale, starting with the smallest (i.e. current one):
 - Run gradient ascent
 - Upscale image to the next scale
 - Reinject the detail that was lost at upscaling time
- Stop when we are back to the original size.
To obtain the detail lost during upscaling, we simply
take the original image, shrink it down, upscale it,
and compare the result to the (resized) original image.
```

```
"""

Playing with these hyperparameters will also allow you to achieve new ↵
effects

step = 0.01 # Gradient ascent step size
num_octave = 3 # Number of scales at which to run gradient ascent
octave_scale = 1.4 # Size ratio between scales
iterations = 20 # Number of ascent steps per scale
max_loss = 10.

img = preprocess_image(base_image_path)
if K.image_data_format() == 'channels_first':
 original_shape = img.shape[2:]
else:
 original_shape = img.shape[1:3]
successive_shapes = [original_shape]
for i in range(1, num_octave):
 shape = tuple([int(dim / (octave_scale ** i)) for dim in original_shape ↵
])
 successive_shapes.append(shape)
successive_shapes = successive_shapes[::-1]
original_img = np.copy(img)
shrunk_original_img = resize_img(img, successive_shapes[0])

for shape in successive_shapes:
 print('Processing image shape', shape)
 img = resize_img(img, shape)
 img = gradient_ascent(img,
 iterations=iterations,
 step=step,
 max_loss=max_loss)
 upscaled_shrunk_original_img = resize_img(shrunk_original_img, shape)
 same_size_original = resize_img(original_img, shape)
 lost_detail = same_size_original - upscaled_shrunk_original_img

 img += lost_detail
 shrunk_original_img = resize_img(original_img, shape)

save_img(result_prefix + '.png', deprocess_image(np.copy(img)))
```

```
Model loaded.
Processing image shape (365, 619)
..Loss value at 0 : 0.9091206
..Loss value at 1 : 1.140517
..Loss value at 2 : 1.4529788
..Loss value at 3 : 1.79357
..Loss value at 4 : 2.143096
..Loss value at 5 : 2.4735835
```

```
..Loss value at 6 : 2.8045518
..Loss value at 7 : 3.1402962
..Loss value at 8 : 3.4778132
..Loss value at 9 : 3.795566
..Loss value at 10 : 4.1271343
..Loss value at 11 : 4.413039
..Loss value at 12 : 4.766541
..Loss value at 13 : 5.080829
..Loss value at 14 : 5.398209
..Loss value at 15 : 5.7061243
..Loss value at 16 : 6.0138664
..Loss value at 17 : 6.3064604
..Loss value at 18 : 6.5823793
..Loss value at 19 : 6.86627
Processing image shape (512, 867)
..Loss value at 0 : 1.552897
..Loss value at 1 : 2.3976157
..Loss value at 2 : 3.0669863
..Loss value at 3 : 3.6399107
..Loss value at 4 : 4.185704
..Loss value at 5 : 4.6712923
..Loss value at 6 : 5.139372
..Loss value at 7 : 5.5815964
..Loss value at 8 : 5.9831276
..Loss value at 9 : 6.368027
..Loss value at 10 : 6.753023
..Loss value at 11 : 7.114599
..Loss value at 12 : 7.45804
..Loss value at 13 : 7.8078575
..Loss value at 14 : 8.100944
..Loss value at 15 : 8.431373
..Loss value at 16 : 8.745302
..Loss value at 17 : 9.048229
..Loss value at 18 : 9.330298
..Loss value at 19 : 9.615529
Processing image shape (717, 1215)
..Loss value at 0 : 1.6899039
..Loss value at 1 : 2.5947077
..Loss value at 2 : 3.3479729
..Loss value at 3 : 3.9903874
..Loss value at 4 : 4.6247597
..Loss value at 5 : 5.2228127
..Loss value at 6 : 5.8321767
..Loss value at 7 : 6.4828396
..Loss value at 8 : 7.21245
..Loss value at 9 : 8.070387
..Loss value at 10 : 9.1532345
```



### 8.3. Minecraft-MALMÖ

Most, hogy már van némi ágensprogramozási gyakorlatod, adj egy rövid általános áttekintést a MALMÖ projektről!

Megoldás videó: initial hack: <https://youtu.be/bAPSu3Rndi8>. Red Flower Hell: <https://github.com/nbatfai/RedFlowerHell>.

Tutor: Bukovinszki Márk Tutoriált: Bukovinszki Márk

Megoldás forrása: a Red Flower Hell repójában.

MIT License Copyright (c) 2016, 2018 Microsoft Corporation

A Malmo projekt a Minecraft által épített mesterséges intelligencia kísérletezésének és kutatásának platformja. Arra törekszenek ezzel, hogy egy új kutatási generációt inspiráljanak az ezen egyedi környezet által bemutatott új problémák kihívásaira.

Letölteni a <https://github.com/Microsoft/malmo/releases> oldalról tudjuk. Szerencsénk van, mivel különböző platformokkal is kompatibilis a Minecraft. Legyen az Linux, Windows, MacOs, bármelyik platformon futtatni tudjuk játékot. Ubuntu pl. a Minecraft mappába belépve, onnan egy terminált megnyitva, a `./launchClient.sh` parancssal már indíthatjuk a Minecraft-ot.

Ezután ha indulásra kész a játék, neki kezdhetünk a saját kis karakterünk beprogramozásának. Lehetőségünk van akár C++, akár Python szövegkörnyezetben írni a forráskódunkat. Mozgási parancsokat adhatunk ki karakterünknek, legyen az előremozgás, ugrás, fordulás, ütés, amit csak szeretnénk (persze a megengedett kereteken belül).

Nálam, személy szerint az egész Malmö Project a Debreceni Egyetem második félévében kezdődött el. Bátfai Tanár Úr sokszor ösztönzött minket, hogy a Malmö-vel igencsak fejleszthetjük programozási tudásunkat, habár néha úgy gondoltuk, hogy a Delete gomb megnyomásával az egészet a kukába tesszük. Visszagondolva tényleg tanulhattunk a Minecraft-tól, hiszen néha matematikai számításokat, logikai dolgozást igényelt az adott feladatok beprogramozása. Eddig három verseny került megrendezésre. Az első versenyünkön bármilyen haxorkodás elfogadható volt. Teleportálhattunk a virágokért akár, ezzel összegyűjtve az összes virágot. A második versenytől lett csak igazán érdekes a verseny. Bevezettünk több szabályt, amely nehezebbé tette a dolgunkat a programozás terén. Majd a harmadik versenyen jött az igazi őrült. Finomítgattuk a programunkat, ahogy tudtuk, a bugokat megröbáltuk kijavítni, és a verseny indulása előtt kaptunk egy teljesen új pályát, amellyel ágensünk megmérkőzhetett. Sajnos igen, a miénk csunyán elbukott, de nem tántoradtunk meg, így napról napra tovább fejlesztettük azt. Na jó legyen vége a Story Time-nak és beszéljünk a lényegesebb dolgoról.

```
Loop until mission ends:
while world_state.is_mission_running:
 print("--- nb4tf4i arena -----\\n")
 self.agent_host.sendCommand("move 1")
 time.sleep(.5)
 self.agent_host.sendCommand("turn 1")
 time.sleep(.5)
 world_state = self.agent_host.getWorldState()
```

Itt láthatjuk a program egyik lényegi részét. Így adunk ki mozgási parancsokat.

```
if "Pitch" in observations:
 self.pitch = int(observations["Pitch"])
if "XPos" in observations:
 self.x = int(observations["XPos"])
if "ZPos" in observations:
 self.z = int(observations["ZPos"])
if "YPos" in observations:
 self.y = int(observations["YPos"])

print(" Steve's Coords: ", self.x, self.y, self.z)
print(" Steve's Yaw: ", self.yaw)
print(" Steve's Pitch: ", self.pitch)

if "LineOfSight" in observations:
 lineOfSight = observations["LineOfSight"]
 self.lookingat = lineOfSight["type"]
 print(" Steve's <): ", self.lookingat)
```

Itt amit láthatunk, nem más, mint az, hogy mit lát Steve. Magyarán megmondva, lekérdezzük a környezetet 3x3x3-as látásmódban.

Mindenkit csak bíztatni tudok, hogy próbálja ki ezt a programozási formát. Akik szeretik a Minecraft-ot és persze programozni is szeretnek/szeretnének, ajánlani tudom a következő Youtube csatornát, ahol megtanulhatjátok az alapokat, hogy hogyan kezdejetek neki a Malmö-Projectnek: [https://www.youtube.com/channel/UCKrJV21SFN\\_5YN6d1F\\_5UIw](https://www.youtube.com/channel/UCKrJV21SFN_5YN6d1F_5UIw)

## 8.4. Vörös Pipacs Pokol/javíts a 19 RF-en

Megoldás forrása: <https://youtu.be/UbiBkkLx1Fk>

Mivel az előző fejezetben is arról beszélünk, amivel a 19 RF-en javítottunk, ezért hivatkoznék az előző fejezet 5. feladatára, ahol meglehet nézni, hogyan javítottunk a kódon.

## 9. fejezet

# Helló, Chaitin!

Összefoglaló videó: <https://youtu.be/-1gtws7pZyc>

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

Megoldás videó: <https://youtu.be/-1gtws7pZyc>

Megoldás forrása:

A Lisp programozási nyelv (helyesebben nyelvcsalád) hosszú történetre tekint vissza. Eredetileg általános célú programnyelvnek terveztek az 50-es évek legvégén, de hamarosan a mesterséges intelligencia kutatás előszeretettel alkalmazott nyelvévé vált, amikor az 50-es, 60-as években ezen terület az első virágkorát élte. Ma a Lisp nyelveket számos területen alkalmazzák, és közkedvelt a számításelmélet oktatásában is.

—Wikipedia

A LISP használatához először is le kell telepítenünk a Lisp futtató programot, a clisp-t. Amit így futtatunk: clisp fájlnév.lisp

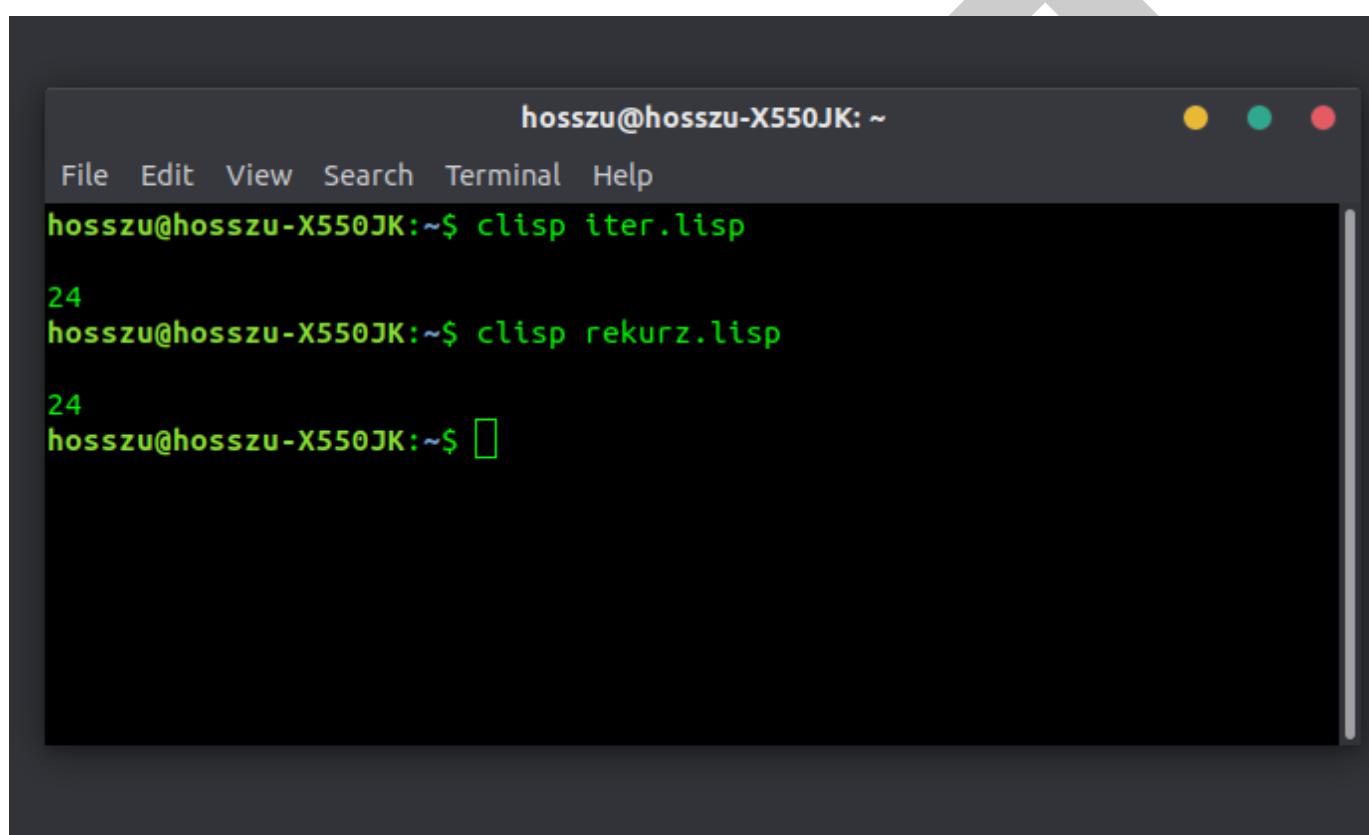
iterált -> az algoritmus annyiszor hajtódik végre, amennyi a megadott szám (4)

```
(defun factorial (N)
 (let ((R 1))
 (do ((i 1 (+ i 1))) ((> i N) R)
 (setf r (* r i)))
)
)
 (print (factorial 4))
)
```

rekurziv -> addig hívja meg önmagát, míg a beadott számból ki nem jön az 1 (4 -> 1)

```
(defun factorial (N)
 (if (= N 1)
 1
 (* N (factorial (- N 1))))
)
)

(print (factorial 4))
)
```



The screenshot shows a terminal window with a dark theme. The title bar says "hosszu@hosszu-X550JK: ~". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal window contains the following text:

```
hosszu@hosszu-X550JK:~$ clisp iter.lisp
24
hosszu@hosszu-X550JK:~$ clisp rekurz.lisp
24
hosszu@hosszu-X550JK:~$
```

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

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

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

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

A GIMP Chrome filterével krómszerű hatást adhatunk az általunk megadott szövegnek. A textünk a krómszegély bal felső sarkában fog elhelyezkedni. Mindezt az első feladatból megismert Scheme programozási nyelvben, a GIMP eljárásain keresztül megírni. A script fájl elhelyezése után (GIMP 2\share\gimp\2.0\scripts), a GIMP futtatásakor láthatjuk, hogy a Létrehozás fülnél megjelent a Chrome3-Border2.

```
(script-fu-register "script-fu-bhax-chrome-border"
```

```
"Chrome3-Border2"
"Creates a chrome effect on a given text."
"Norbert Bátfai"
"Copyright 2019, Norbert Bátfai"
"January 19, 2019"
""

SF-STRING "Text" "HosszúGyulaVideos"
SF-FONT "Font" "Sans"
SF-ADJUSTMENT "Font size" '(160 1 1000 1 10 0 1)
SF-VALUE "Width" "1920"
SF-VALUE "Height" "1080"
SF-VALUE "New width" "400"
SF-COLOR "Color" '(255 0 0)
SF-GRADIENT "Gradient" "Crown molding"
SF-VALUE "Border size" "7"

)
(script-fu-menu-register "script-fu-bhax-chrome-border"
 "<Image>/File/Create/BHAX"
)
```



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

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

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

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

A dolgunk hasonló mint az előbb. A script fájl elhelyezése után (GIMP 2\share\gimp\2.0\scripts), a GIMP futtatásakor láthatjuk, hogy a Létrehozás fülnél megjelent a Mandala9.

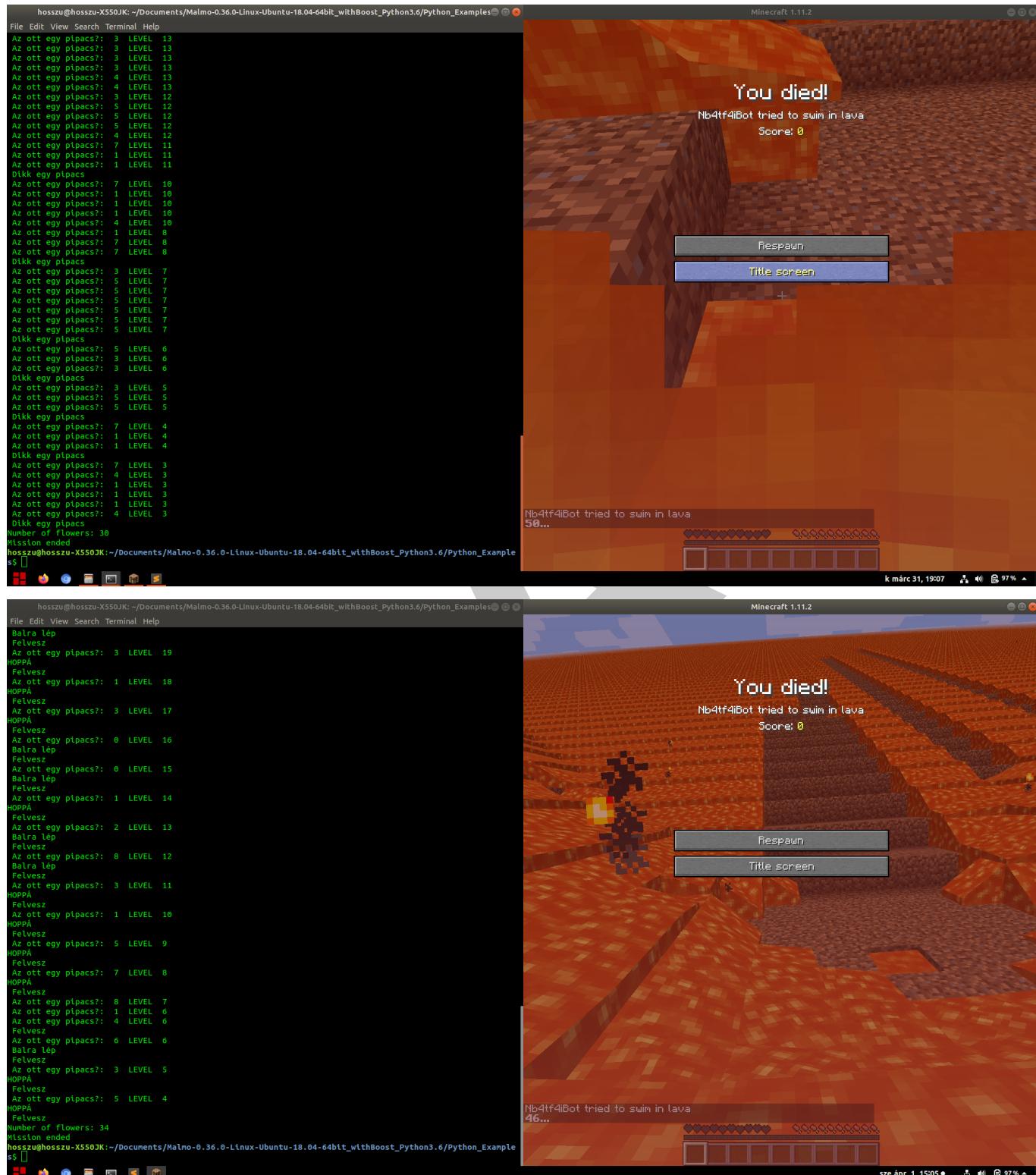
```
(script-fu-register "script-fu-bhax-mandala"
 "Mandala9"
 "Creates a mandala from a text box."
 "Norbert Bátfai"
 "Copyright 2019, Norbert Bátfai"
 "January 9, 2019"
 ""
 SF-STRING "Text" "BHAXOR"
 SF-STRING "Text2" "HAXOR"
 SF-FONT "Font" "Sans"
 SF-ADJUSTMENT "Font size" '(100 1 1000 1 10 0 1)
 SF-VALUE "Width" "1000"
 SF-VALUE "Height" "1000"
 SF-COLOR "Color" '(255 0 0)
 SF-GRADIENT "Gradient" "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
 "<Image>/File/Create/BHAX"
)
```

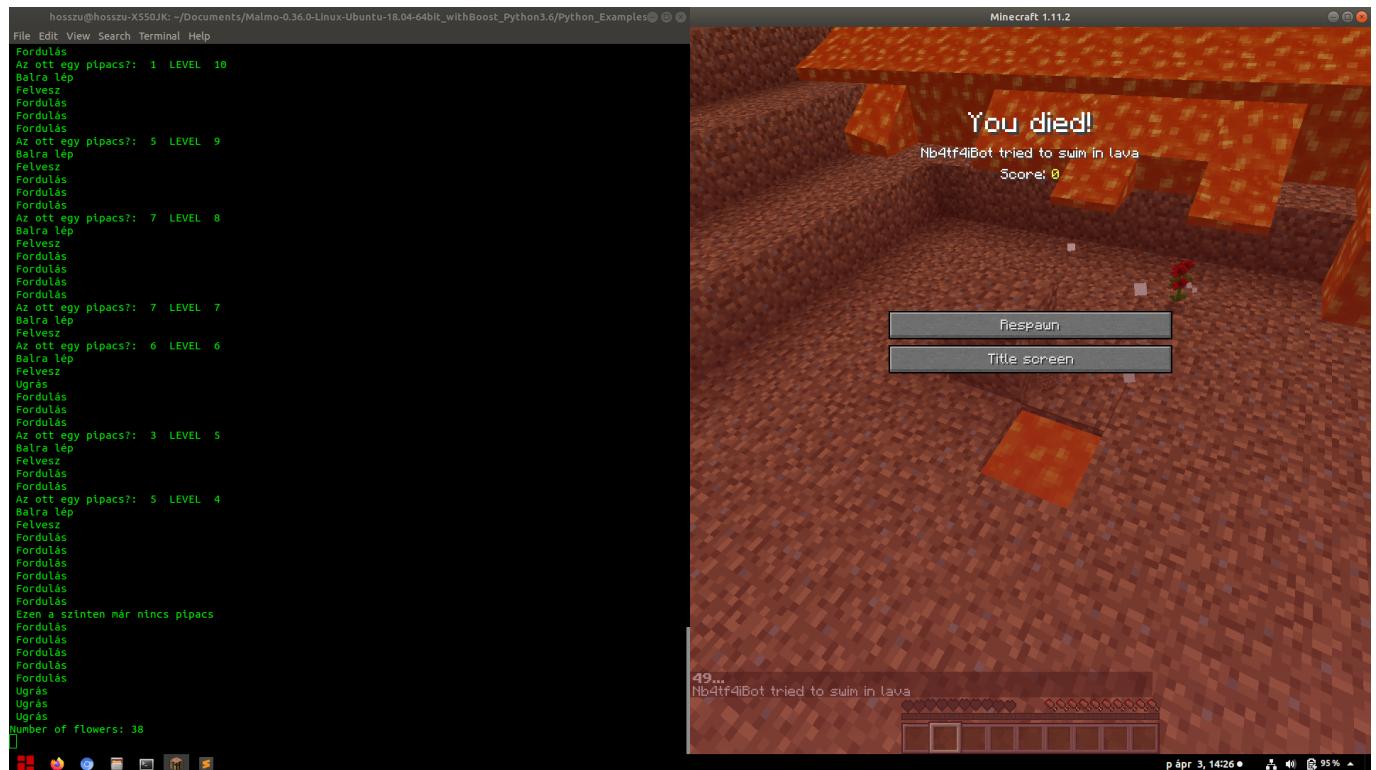


## 9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-eden

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Szokásos módon hivatkoznék az előző két fejezet Malmö-s feladatára. Hiszen folyamatosan javítottuk a kódunkat és nem minden forrásfájl van meg hozzá. De pár screenshot-tal bemutatom a fejlődést.





## 10. fejezet

# Helló, Gutenberg!

### 10.1. Programozási alapfogalmak:

A számítógépek programozási nyelvén belül három szintet különböztetettünk meg: van a gépi nyel; az assembly szintű nyelv; és a magas szintű nyelv. A magas szintű nyelveken írt programokat forrásprogramnak, vagy forrásszövegeknek nevezzük. A forrásszövegek felépítésére vonatkozó formai (nyelvtani) szabályok összességét szintaktikai szabályoknak hívjuk. A szemantikai szabályokat a tartalmi, értelmezési, jelentésbeli szabályok alkotják. Ezen szabályok összessége határozza meg a magas szintű programozási nyelveteket. A magas szintű nyelven megírt forrásszövegből fordítóprogramos vagy az interpreteres technikával el kell jutni a gépi nyelvű programokhoz, hogy azt a processzor, az adott gépi nyelven írt programot végre tudja hajtani. A fordítóprogram a magas szintű programozási nyelven megírt forrásszövegből, gépi kódú tárgyprogramot hoz létre. Általánosabb értelemben tetszőleges nyelvről tetszőleges nyelvre fordít. Ennek menete a lexikális elemzés, szintaktikai elemzés, szemantikai elemzés majd végső soron a kódgenerálás. Ilyen tárgyprogramot csak a szintaktikailag helyes forrásprogramból lehet előállítani. A C nyelv például egy olyan nyelv, amely egy előfordító segítségével elsősorban a forrásprogramból egy adott nyelvű forrásprogramot generál, ami már feldolgozható a nyelv fordítójával. Az interpreter technika első három lépése azonos a fordítóprogram első három lépéseihez, viszont nem készít tárgyprogramot. Egyes programnyelvek vagy egyiket vagy a másikat használják, viszont léteznek olyanok amely mindenkorral alkalmazzák. minden programnyelvnek megvannak a sajátosságai, amit hivatkozási nyelvnek nevezünk. Definiálva vannak a szintaktikai, szemantikai szabályok. A szintaktikát formálisan, a szemantikát általában természetes emberi nyelven adják meg. Ezek mellett léteznek implementációk. Ezekkel viszont az a probléma, hogy egymással és a hivatkozási nyelvvel nem kompatibilisek.

Karakterkészlet:

A programok forráskódjának legkisebb alkotóelemei a karakterek. A karakterkészlet alapvető a forráskód összeállításnál. Ezek a következők: lexikális egységek, szintaktikai egységek, utasítások, programegységek, fordítási egységek, program. minden nyelv rendelkezik saját karakterkészlettel. A karakterek kategóriái általában a következők: betűk, számjegyek, egyéb.. minden programnyelv az angol ABC betűit használja. "A lexikális egységek a program szövegének azon elemei, melyeket a fordító a lexikális elemzés során felismer és tokenizál." "A többkarakteres szimbólumok olyan karaktersorozatok, amelyeknek jelentést tulajdonít az adott nyelv és ezek csak ebben az értelemben használhatóak."

Lexikális egységek:

"A lexikális egységek a program szövegének azon elemei, melyeket a fordító a lexikális elemzés során

felismer és tokenizál." "A többkarakteres szimbólumok olyan karaktersorozatok, amelyeknek jelentést tulajdonít az adott nyelv és ezek csak ebben az értelemben használhatóak."

Szimbolikus nevek:

Azonosító: Arra használjuk, hogy programozás közben, saját eszközeinket megnevezzük velük, majd erre később tudunk hivatkozni a szövegben. Kulcsszó: Olyan karaktersorozat, amelynek a nyelv már alapjáraton tulajdonít jelentést, és ezt a programozó nem változtathatja meg. Ezek általában hétköznapi angol szavak. Standard azonosító: Olyan karaktersorozat, mint a kulcsszó, viszont a programozó ezt megváltoztathatja. Címke: "Az eljárásorientált nyelvekben a végrehajtható utasítások megjelölésére szolgál, azért, hogy a program egy másik pontjáról hivatkozni tudjunk rájuk. Bármely végrehajtható utasítás megcímkézhető." Megjegyzés: Olyan programozási eszköz, amely segítségével a programozó egy karaktersorozatot helyes el a kódban, ami a szöveg olvasójának szól. Egyszerűen egy magyarázó eszköz. Literálok (Konstansok): Segítségével fix, explicit értékek építhetők be a programba. Két komponensük van: típus és érték.

## 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

2.fejezet: Típusok, operátorok és kifejezések Ebben a fejezetben változókról, állandókról, deklarációkról, operátorokról, típuskonverzióról olvashatunk. A c-ben csak néhány alapvető adattípus van: a char, az int, a float és a double. Használat előtt minden változót deklarálni kell. Ha egy kifejezésben különböző típusú operandusok fordulnak elő, a kifejezés kiértékeléséhez az operandusokat azonos típusúakká kell alakítani. A C nyelv tartalmaz két szokatlan operátort, amelyekkel változók inkrementálhatók és dekrementálhatók. A ++ inkrementáló operátor operandusához 1 -et ad hozzá, a—decrementáló operátor pedig 1 -et von le belőle. A C nyelvben több bitmanipulációs operátor van; ezek a float és double típusú változókra nem alkalmazhatók.

3.fejezet: Vezérlési szerkezetek Ebben a fejezetben a C nyelv vezérlésátadó utasításairól olvashatunk. A kifejezések utasítássá válnak, ha a kifejezés végére pontosvesszőt teszünk. Részletesen olvashatunk az if-else utasításokról. A switch utasításról is olvashatunk, amely a programelágaztatás eszköze. A while-for utasításokról olvashatunk, melynek ha értéke nem nulla, akkor végrehajtja az utasítást és kiértékeli a kifejezést. vizsgálja. A következő C-beli ciklusfajta, a do-while a vizsgálatot a ciklus végén, a ciklustörzs végrehajtása után végzi el; a törzs tehát legalább egyszer mindenképpen végrehajtódik. A fejezet végén olvashatunk a break utasításról, a continue és a goto utasításokról.

4.fejezet: Függvények és programstruktúra C nyelvet úgy terveztek meg, hogy a függvények hatékonyak és könnyen használhatók legyenek. A C programok általában sok kis méretű függvényt tartalmaznak. A program lényegében egyedi függvénydefiníciók halmaza. A függvények közötti kommunikáció argumentumokkal és a függvények által visszaadott értékkal történik, de történhet külső változókon keresztül is. A meghívott függvények a return utasítással adhatnak visszatérési értéket. Olvashatunk továbbá Header álmányok vizsgálatáról, speciális változótípusokról, blokkstruktúráról, inicializálásról és a rekurzióról.

## 10.3. Szoftverfejlesztés C++ nyelven

Ebben a fejezetben a C++ programozási nyelv elődjének, a C nyelvnek egy továbbfejlesztéséről olvashatunk. A továbbfejlesztett C nyelvnek a célja, hogy a C nyelvben előforduló "veszélyesnek" mondható

elemeket lecserélik biztonságosabbra. Egy-két újítás csak a kényelem érdekében ment végbe (átláthatóbb programkód, stb...). Fontos tudni, hogy ebben a fejezetben előforduló programok csak a C++ fordítóval fordulnak le. Olvashatunk a függvényparaméterek és visszatérési értékekről. Például a C nyelvben, ha egy függvényt üres paraméterlistával definiálunk, akkor az tetszőleges számú paraméterrel hívható. A C++ nyelvben ez az "üresség" a void paraméter megadásával ekvivalens, vagyis a függvénynek nincs paramétere. A függvény visszatérési típusának meg nem adásakor a C nyelv és C++ nyelv eltérően viselkedik. A szabványos C++ nyelvben a main függvénynek két formája létezik: a paraméter nélküli, valamint a paraméteres(argc paraméter, argv paraméter). A C++ nyelvben bevezették a bool típust, amely logikai (igaz/hamis) értéket képes visszaadni. A C nyelvben ilyen értékek visszaadására az int vagy enum típusú kifejezések képesek. Ezen típus bevezetésével a forráskód átláthatóbb/olvashatóbbá vált. A C++ nyelvben egy változódeklaráció olyan helyen is szerepelhet, ahol utasítás állhat. Azt tudni kell, hogy egy C nyelvű megoldás használható C++ nyelvben is, mivel a C++ visszafelé kompatibilis a C-vel. A C++ nyelvben az operátorok a helyfoglalás mellett meghívják a konstruktort és a destruktort. C++ nyelvben a try-catch függvényt használjuk kivételkezeléskor, valamint makrók helyett konstansokat használunk.

## 10.4. Bevezetés a Pythonba

A Python egy magasszintű programozási nyelv, mégis a szkriptnyelvek családjába szoktál sorolni. Tudni kell róla, hogy a nyelv elsajátítása egyszerű, és komplex algoritmusok is könnyen leírhatóak vele. A C nyelvekkel szemben nagy előnye, hogy bő a standard könyvtára. Mivel interpretált használ, a terminálból rögtön tudjuk futtatni a megírt kódunkat.

Behúzásalapú a szintaktikája, nem szükséges ; a sorok végére, valamint egy kifejezés végét egy üres blokk jelöli. A kulcsszavak tipikusan angol szavak ebben a nyelvben. Kommentek elhelyezésére a # jelet használjuk.

Deklaráláskor megadnunk sem kell a típusokat, ezt az interpreter feldolgozza. Az adattípusok a következők: stringek, számok, listák, szótárak, ennesek.

Értékkadás a = jel segítségével megy végbe. Itt is léteznek lokális és globális változók. Ha egy változót egy függvényen belül hozunk létre lokális lesz, de ha alkalmazzuk a **global** előtagot akkor globálisként fogja tekinteni.

A **print** függvény a terminálunkra iratja ki a kifejezést. Az elágazásokkor az **if/else/elif** kulcsszavakat használjuk. és a kifejezés után :-ot használunk.

Címkeket is alkalmazhatunk a **label** kifejezéssel, majd a **goto** kulcsszóval ugorhatunk oda. A függvények deklarálásához a **def** kulcsszót használjuk.

**III. rész**

**Második felvonás**

**DRAFT**

**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.

DRAFT

## 11. fejezet

### Helló, Berner-Lee!

**11.1. C++: Benedek Zoltán, Levendovszky Tihámér Szoftverfejlesztés  
C++ nyelven VS. Java: Nyékyné Dr. Gaizler Judit et al. Java 2  
útikalauz programozóknak 5.0 I-II**

\*

A bevezetésben ki lett fejtve, hogy a Java sok minden át vett a C++ nyelvtől, sok figyelmet fordított a magbízhatóságra és felhívta a figyelmet a nyelvi szerkezetben felfedezhető újdonságokra, pld.: a mutatók helyett a referenciaiák használatára, futató környezet korlátozásai.

\*

A Java nylev teljesen objektum orientált (objektumok, és ezek mintáinak tekinthető osztályok összesége).

Egy osztály két dologból állhat: mezőkből, azaz változókból, valamint metódusokból, melyeket erőszerezzettel hívunk függvényeknek, holott nem minden van visszatérési értékük. Mezőkben tároljuk az a datokat, a metódusokkal pedig az adatokon végezhető műveletek kódját adjuk meg.

Megnéztük a HelloVilág programot, Javaban:

```
public class Hello {
 public static void main(String[] args) {
 System.out.println("Hello Világ");
 }
}
```

z objektumorientált paradigma alapfoglamai

És azt le is fordítottuk:

```
gyula@gyula-X550JK:~/Documents/prog2$ javac Hello.java
gyula@gyula-X550JK:~/Documents/prog2$ java Hello
Hello Világ
```

Megfigyelhettük, hogy a Java fordítóprogram egy bájtkódnak nevezett formátumra fordítja le a forráskódot, melyet a Java Virtuális Gép önálló interpretensként fog értelmezni. Biztonsági szempontból előnyös, de sebesség terén hátrányos, ez az eljárás.

Aztán kielemeztük minden részét a példa kódnak, `main`, a program fő metódusa, az azt követő `static` kulcs szó lehetővé teszi, hogy a Hello osztály futtatása során, nem jön létre egyetlen objektum sem, `public`, ha nem szerepelne a kódban, akkor a Java meggátolná a program futását, ez teszi lehetővé, hogy a metódus, a külvilág számára is látható legyen, `void`, nem-függvény jellegű metódusra, azaz a visszatrési érték elhagyására utal, `String []`, szövegtömbön keresztül kerülnek átadásra a parancssor argumentumai.

`System.out` objektumának `println` metódusa kerül meghívásra, mely a paraméterként kapott szöveget, kiírja a szabványos kimenetre, egy újsor karakterrel kiegészítve azt.

Java nyelv egy figyelemre méltó újdonsága, az ASCII karakterészletből hiányzó karaktereket, nemcsak a szövegliterátorokba, hanem a megjegyzésekben, sőt az azonosítókban is lehet használni.

\*

A Java nyelv teljesen objektumorientált. Egy osztály két dologból tevődik össze: mezők(adatok tárolása), metódusok(adatokon végezhető műveletek kódja).

Futtatáshoz szükségünk van egy Java nevű fordítóprogramra. Ez egy bájtkódnak nevezett formátumra fordítja le a forráskódot, amelyet majd a Java Virtuális Gép önálló interpretereként fog értelmezni. Ez lassabb futást eredményez. A legnagyobb különbség a Java illetve a C++ között itt mutatkozik. A C++ kézenfekvőbb, ha olyan programokat szeretnénk írni, amelyeknél a futás sebessége elsődleges szempont. A Java inkább a web-es téren elterjedt.

A Java nyelvnek vannak egyszerű típusai is, melyeket az adatok egyszerű reprezentálására lehet használni. Ezeknek értéket adni az '=' operátorral lehet. Ilyenkor ez valódi értékadást jelent, összetett típusok esetében csak egy referencia átmásolását jelenti.

Eltérés még a C++ és a Java között, hogy Java-ban már 16 bites Unicode karaktereket is lehet használni változók vagy konstansok deklarálásához, tehát használhatunk ékezetes karaktereket, görög ábécé betűit, stb.

Megjegyzéset ugyanúgy adhatunk hozzá a kódhoz, mint a C++ esetében.

A Java nyelvben semmilyen explicit eszköz nincs egy objektum megszüntetésére, egyszerűen nem kell hivatkozni rá és magától meg fog szűnni. Hivatkozás megszüntetéséhez az eddigi referencia helyett a null referenciát adjuk neki értékül.

A Java nyelvben tömböket a [] jelöléssel lehet megadni. A C++-tól eltérően ez egy igazi típus lesz és nem csak a mutató típus egy másik megjelenítési formája. A tömb típusok nem primitív típusok, a tömb típusú változók objektumhivatkozást tartalmaznak. Eltérés még a C++ és a Java között, hogy a Java-ban nincsenek többdimenziós tömbök, erre a tömb a tömbben megoldást használja.

További eltérés még a Goto utasítás teljes hiánya, ez a Java nyelvből kamaradt.

Java-ban egy objektumot a következő képen példányosítunk.

```
#Adott egy objektum
public class Alkalmazott {
 String nev; int fiztes;
 void fizetesemeles (int novekmeny) {
 fizetes += novekmeny
 }
}
```

```
}
```

#Példányosítás  
Alkalmazott a = new Alkalmazott();

A new operátor mögött adjuk meg, h melyik osztályt példányosítjuk. A zárójelek közé az adott esetben a konstruktornak szánt paramétereket írjuk. Csakúgy, mint a C++-ban, itt is vannak nyilvános (public) és privát (private) tagok. A private tagokhoz az adott osztályon kívül nem férhet hozzá semmi.

Java megkülönbözteti a referenciát és az imperatív programozási nyelvekben használt mutatót aképpen, hogy a kifejezésekben automatikusan a mutatott objektumot jelenti, nem pedig a címet.

## 11.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba.

A Python egy általános célú programozási nyelv. Alkotója Guido van Rossum. Leginkább prototípus készítésre és tesztelésre szokás alkalmazni. Képes együttműködni más nyelveken íródott modulokkal. Elterjedtségét főként az imént említett tulajdonságának, valamint a rövid tanulási ciklusnak köszönheti. A Python-ban megírt programok terjedelmükben jóval rövidebbek, mint a C, C-ben, C++-ban vagy Java-ban készült, velük ekvivalens társaik. A kódok tömörek és könnyen olvashatóak. A kódcsoporthoz új sort és tabuláltot használ (behúzásalapú), nincs szükség nyitó és zárójelekre, nincs szükség változó vagy argumentumdefiniálásra. Az utasítások a sorok végéig tartanak, nincs szükség ';' -re, ha egy adott utasítást a következő sorban szeretnénk folytatni, akkor ezt a '\' jellel lehetjük meg.

Példa:

```
if feltételi és feltétel2
 alapfeladat()
egyébfeladat()
```

Az értelmező a sorokat tokenekre bontja, amelyek között tetszőleges üres (whitespace) karakter lehet. A tokenek lehetnek: azonosító(változó, osztály, függvénymodul), kulccsó, operátor, delimiter, literál.

Lefoglalt kulcsszavak: and, del, for, is, raise, assert, elif, from, lambda, return, break, else, global, not, try, class, except, if, or, while, continue, exec, import, pass, yield, def, finally, in, print.

Típusok

A Pythonban minden adatot típusok reprezentálja. Az adatokon végezhető műveleteket az objektum típusa határozza meg. A változók típusait a rendszer futási időben "kitalálja".

Számok lehetnek: egészek (decimális, oktális, hexadecimális), lebegőpontosak és komplex számok.

Sztringeket aposztrófok közé írva adhatunk meg, illetve a 'u' betű használatával Unicode szövegeket is felvehetünk.

Példa:

```
print u"Hello %s, kedves %s!"
```

Az ennesek (tuples) objektumok gyűjteményei vesszővel elválasztva.

Példa:

```
('a','b','c') #három elemű ennes
tuple('abc') #tuple generálás kulcsszóval (ugyanaz, mint az előző)
() #üres ennes
(1, "szia", 3,) #három elemű ennes, az utolsó vessző elhagyható
```

A lista különböző típusú elemek rendezett szekvenciája, elemeit szögletes zárójelek közé írjuk, dinamikusan nyújtózkodik. Az elemeket az indexükkel azonosítjuk.

Példa:

```
[a','b','c'] #három elemű lista
list('abc') #lista generálás kulcsszóval (ugyanaz, mint az előző)
[] #üres ennes
[1, "szia", 3,] #három elemű lista, az utolsó vessző elhagyható
```

A szótár kulcsokkal azonosított elemek rendezetlen halmaza. Kulcs lehet: szám, sztring stb.

Példa:

```
{'a':1, 'b':5, 'e':1982}
{1:1, 2:1, } #az utolsó vessző elhagyható
{} #üres szótár
```

Változók Pythonban - egyes objektumokra mutató referenciai. Típusaik nincsenek. A hozzárendelést az '=' karakterrel végezzük. A del kulcsszóval törölhető a hozzárendelés, a mögöttes objektum törlését a garbage collector fogja elvégezni. Léteznek globális és lokális változók (akárcsak a C++-ban). A változók közötti típuskonverzió támogatott.

A szekvenciákon (sztringek, listák, ennesek) műveletek végezhetőek el (összefűzés, szélsőérték helyek meghatározása stb.).

Az elágazások és ciklusok működése megegyezik a többi programnyelvvel.

Függvényeket a def kulccsal definiálunk, tekinthetünk rájuk úgy, mint értékekre, mivel továbbadhatóak más függvényeknek illetve objektumkonstruktornak.

```
def hello();
 print "Hello world!"
 return
```

A Python támogatja a klasszikus, objektumorientált fejlesztési eljárásokat (osztályokat definiálunk, amelyek példányai az objektumok. Osztályoknak lehetnek attribútumaik(objektumok, függvények) illetve örökölhetnek más osztályokból.

A kivételkezelés:

Példa

```
try:
 utasítások
except [kifejezés]: #akkor fut le, ha az történik, amit a try után leírtunk
[else:
 utasítások]
```

## 12. fejezet

# Helló, Arroway!

### 12.1. A BPP algoritmus Java megvalósítása

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.!Lásd még fóliák!Ismétlés: [https://arato.inf.unideb.hu/~arato/inf100/22\\_folia.pdf](https://arato.inf.unideb.hu/~arato/inf100/22_folia.pdf)) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPORG repó: source/labor/polargen)

Megoldás video:

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Arroway/polargentes.cpp](https://github.com/bhax/thematic_tutorials/blob/main/bhax_textbook_sajat/Arroway/polargentes.cpp)

A programunk C++-ban, ami három részből áll:

```
#ifndef POLARGEN_H
#define POLARGEN_H

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

class PolarGen
{
public:
 PolarGen ()
 {
 nincsTarolt = true;
 std::srand (std::time (NULL));
 }
 ~PolarGen ()
 {
 }
 double kovetkezo ();

private:
```

```
bool nincsTarolt;
double tarolt;

};

#endif
```

Láthatjuk, hogy elsősorban egy header fájl-ra lesz szükségünk.

Véletlen szám sorsoláshoz szükség van bizonyos előre megírt kódokra, melyeket #include-olni kell ahhoz, hogy a feladatot megoldhassuk. A kódunk elején a sorsolás egyik részét végző rand() függvény használatához a következő sort kell beilleszteni a kódunk elejére: **#include cstdlib**.

```
#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;
 }
}
```

A **polargen.cpp** fájlban valósítjuk meg a polártranszformációt. Ehhez **include**-olunk kell az előbbi **polargen.h** header fájlt. Ebben az esetben akkor fogunk új számot készíteni, ha nincs tárolt számunk a tárolt változóban.

```
#include <iostream>
```

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

int
main (int argc, char **argv)
{
 PolarGen pg;

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

 return 0;
}
```

Ezt a fájlt fogjuk fordítani és futtatni. A for ciklus miatt 10x fog lefutni a programunk, amely 10 szám kiiratását fogja eredményezni.

A futtatás után ez fogad minket:

[bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Arroway/polargen.java](bhax/thematic_tutorials/bhax_textbook_sajat/Arroway/polargen.java)

És akkor jöjjön a Java programunk.

```
public class polargen {
 boolean nincsTarolt = true;
 double tarolt;
 public polargen() {

 nincsTarolt = true;

 }
 public double kovetkezo() {
 if(nincsTarolt) {
 double u1, u2, v1, v2, w;
 do {
 u1 = Math.random();
 u2 = Math.random();

 v1 = 2*u1 - 1;
 v2 = 2*u2 - 1;

 w = v1*v1 + v2*v2;

 } while(w > 1);

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

 tarolt = r*v2;
 nincsTarolt = !nincsTarolt;
 }
 }
}
```

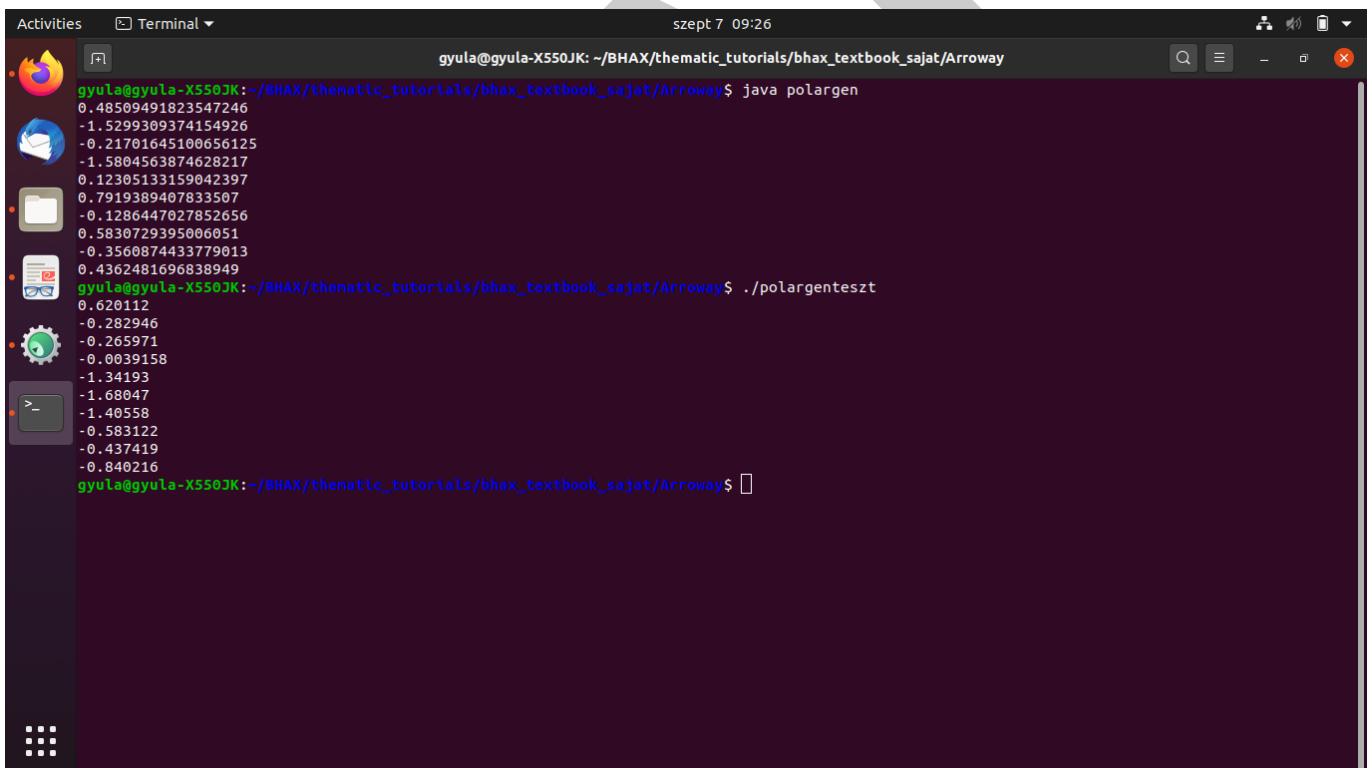
```
 return r*v1;

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

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

A Sun programozói is hasonló képpen oldották meg a random függvényt. A polár metódussal főként az úgynevezett nextGaussian() függvény dolgozik, ami a Random.java fájlból származtatható.

A program futtatása után ez fogad minket:



```
Activities Terminal Terminal szept 7 09:26
gyula@gyula-X550JK:~/BHAX/thematic_tutorials/bhax_textbook_sajat/Arroway$ java polargen
0.48509491823547246
-1.5299309374154926
-0.21701645100656125
-1.5804563874628217
0.12305133159642397
0.7919389407833507
-0.1286447027852656
0.5830729395006051
-0.3560874433779013
0.4362481696838949
gyula@gyula-X550JK:~/BHAX/thematic_tutorials/bhax_textbook_sajat/Arroway$./polargenteszt
0.620112
-0.282946
-0.265971
-0.0039158
-1.34193
-1.68047
-1.40558
-0.583122
-0.437419
-0.840216
gyula@gyula-X550JK:~/BHAX/thematic_tutorials/bhax_textbook_sajat/Arroway$
```

## 12.2. Homokozó

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

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

Megoldás forrása:

Megoldás videó:

```
import java.io.FileInputStream;
import java.io.File;
import java.io.PrintWriter;
import java.io.BufferedWriter;
import java.io.FileWriter;
```

A szükséges import sorok után létrehozzunk a fő Binfa osztályt.

Java-ban mivel nincsenek mutatók, kezdetben a fa egyenlő lesz a gyökérrel.

```
public class Binfa {

 private int id = 1;
 private int depth, max_depth, count = 0;
 private double avg_sum, disp_sum = 0;

 public Binfa() {
 tree = root;
 }
}
```

Az addItem() metódus attól függően, hogy 0 vagy 1 elemet kap, bal vagy jobb oldalról adja a fához azt (0:bal, 1:jobb).

```
void addItem(int i) {
 if(i == 1) {
 if(tree.getRightChild() == null) {
 Node newNode = new Node('1', tree.getChildDepth() + 1, id++);
 tree.newRightChild(newNode);

 tree = root;
 }
 else {
 tree = tree.getRightChild();
 depth = tree.getChildDepth();
 }
 }
 else {
 if(tree.getLeftChild() == null) {
 Node newNode = new Node('0', tree.getChildDepth() + 1, id++);
 tree.newLeftChild(newNode);
 }
 }
}
```

```
 tree = root;
 }
 else {
 tree = tree.getLeftChild();
 depth = tree.getChildDepth();
 }
}
}
```

A következő metódusok/függvények az inorder, mélység, átlag és szórás kiírásában segítenek.

```
void Inorder(PrintWriter outstream) {

 In(root, outstream);

 Depth(root);
 Avg(root);
 Disp(root);
}

public int depthOut() {
 max_depth = depth = 0;
 Depth(root);
 return max_depth;
}

public double avgOut() {
 count = 0;
 avg_sum = 0.0;
 Avg(root);
 return avg_sum/count;
}

public double dispOut() {
 disp_sum = 0.0;
 Disp(root);
 return Math.sqrt(disp_sum--count);
}

public int maxDepth() {
 return max_depth;
}

public double treeAvg() {
 return avg_sum/count;
}
```

A Node osztály beágyazottként van jelen a programban. Ez reprezentál egy csomópontot a bináris fában.

Egy csomópontnak két gyermeké van: bal és jobb oldali, ezek szintén csomópontok. Ezeknek is lesz egy azonosítója, konkrét értéke és mélysége.

```
class Node {

 Node left;
 Node right;

 private int id;
 private char value;
 private int depth;

 public Node(char value, int depth, int id) {
 this.value = value;
 this.depth = depth;
 this.id = id;

 left = null;
 right = null;
 }
}
```

Azt osztály további függvényeivel érhetjük el a bal vagy jobb oldali gyermeket, hozhatunk létre újakat, kérhetjük el az értéküket vagy éppen az azonosítójukat.

```
public Node getLeftChild() {
 return left;
}

public Node getRightChild() {
 return right;
}

public void newLeftChild(Node leftNode) {
 left = leftNode;
}

public void newRightChild(Node rightNode) {
 right = rightNode;
}

public char getChildValue() {
 return value;
}

public int getChildDepth() {
 return depth;
}

public int getChildId() {
}
```

```
 return id;
 }

}
```

Létrehozzuk a fát null értékkel, majd külön csomópontként a gyökeret, ami '/' értékkel, 0 mélységgel és azonosítóval rendelkezik.

```
private Node tree = null;
private Node root = new Node('/', 0, 0);
```

A Depth metódussal számoljuk ki a fa mélységét.

```
public void Depth(Node n) {
 if(n != null) {
 Depth(n.getRightChild());
 if(n.getChildDepth() > max_depth)
 max_depth = n.getChildDepth();
 Depth(n.getLeftChild());
 }
}
```

Az In metódust az Inorder() hívja. Itt írjuk a fa értékeit a fájlba inorder módon (ezért szükséges a PrintWriter példány átadása is) rekurzívan.

```
public void In(Node n, PrintWriter outstream) {
 if(n != null) {
 int i;

 In(n.getRightChild(), outstream);

 for(i = 0; i < n.getChildDepth() + 1; i++)
 outstream.print("---");

 outstream.print(n.getChildValue());
 outstream.print("(");
 outstream.print(n.getChildDepth());
 outstream.println(")");
 In(n.getLeftChild(), outstream);
 }
}
```

Az Avg() az átlagot, a Disp() a szórást számolja ki.

```
public void Avg(Node n) {
 if(n != null) {
```

```
 if(n.getRightChild() == null && n.getLeftChild() == null) {
 count++;
 avg_sum += n.getChildDepth();
 }
 Avg(n.getRightChild());
 Avg(n.getLeftChild());
 }
}

public void Disp(Node n) {
 if(n != null) {
 if(n.getRightChild() == null && n.getLeftChild() == null) {
 disp_sum += (double) ((n.getChildDepth() - treeAvg()) * (n.
 getChildDepth() - treeAvg()));
 }
 Disp(n.getRightChild());
 Disp(n.getLeftChild());
 }
}
```

A main függvényben először ellenőrizzük, hogy a felhasználó megfelelő számú argumentumot adott-e meg és azt, hogy a -o kapcsoló jó helyen van-e.

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

 if(args.length != 3) {
 System.out.println("Használat:\nJava Binfa bemeneti_fajl -o ↪
 kimeneti_fajl");
 System.exit(-1);
 }

 if(!"-o".equals(args[1])) {
 System.out.println("Használat:\nJava Binfa bemeneti_fajl -o ↪
 kimeneti_fajl");
 System.exit(-1);
 }
}
```

Az IO műveletek miatt a main további részét try-catch blokkokba kell írni, mert a nem elérhető fájl vagy IO hiba kivételt generál. Létrehozzuk az íráshoz/olvasáshoz szükséges Stream-eket, példányosítjuk az osztályt, majd beolvassuk a bemeneti fájl tartalmát. Ha a következő sor > karakter (vagy azzal kezdődik) vagy új üres sor, akkor kihagyjuk. Az adott karakter bitmaszkolása után átadjuk az addItem() metódusnak az 1-est vagy a 0-t. Kiírjuk az inorder fát, annak 3 tulajdonságát, zárjuk a Stream-eket, és kezeljük az esetleges kivételeket.

```
try {
 FileInputStream in = new FileInputStream(new File(args[0]));
 PrintWriter out = new PrintWriter(new BufferedWriter(new ↪
 FileWriter(args[2])));
 byte[] b = new byte[1];
```

```
Binfa bintree = new Binfa();

while(in.read(b) != -1) {
 if(b[0] == 0x0a) {
 break;
 }
}

boolean comment = false;

while(in.read(b) != -1) {
 if(b[0] == 0x3e) { // > char
 comment = true;
 continue;
 }

 if(b[0] == 0x0a) { // new row
 comment = false;
 continue;
 }

 if(comment)
 continue;

 if(b[0] == 0x4e) // N char
 continue;

 for(int i = 0; i < 8; ++i) {
 if((b[0] & 0x80) != 0)
 bintree.addItem(1);
 else
 bintree.addItem(0);

 b[0] <<= 1;
 }
}

bintree.Inorder(out);
out.println("depth = " + bintree.maxDepth());
out.println("avg = " + bintree.avgOut());
out.println("var = " + bintree.dispOut());
out.close();
in.close();

} catch(java.io.FileNotFoundException fNFE) {
 fNFE.printStackTrace();
} catch(java.io.IOException ioE) {
 ioE.printStackTrace();
}
```

}

### 12.3. "Gagyí"

Az ismert formális „while ( $x \leq t \&& x \geq t \&\& t \neq x$ );” 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 forrására, hogy a 128-nál inkluzív objektum példányokat poolozza!

## Megoldás forrása:

```
public class Gagy2
{
 public static void main (String[] args)
 {
 Integer x = -128;
 Integer t = -128;
 System.out.println (x);
 System.out.println (t);
 while (x <= t && x >= t && t != x);
 }
}
```

Akkor vizsgáljuk meg ezt a programot. Tudjuk, hogy ez le fog futni hiszen ugyanarra a számra (-128), ugyanazt az objectet adja, mivel csak kiveszi a kész integert a poolból, vagyis a két cím equal lesz.

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

```
Integer t = -129;
System.out.println (x);
System.out.println (t);
while (x <= t && x >= t && t != x);
}
```

Na most, nézzük meg ezt az esetet, amikor `-129` -el dolgozunk. Ez a program lefog fagyni, és végtelen ciklus eredményez. Nyissuk ki az `src.zip` fájlt, azon belül is az `integer.java` forrást.

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

Mivel láthatjuk, hogy ebben az esetben az if nem teljesül, így folyamatosan a `return new Integer(i);` fog lefutni. Tehát a két integernél kettő új objektum fog kreálódni, más címmel és máshol a tárban, így az összehasonlításuk mindenkor hamis lesz.

## 12.4. Yoda

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

```
public class Yoda {

 public static void teszt1() {

 String yoda = "yodahola";

 if (yoda.equals("yodahola")){
 System.out.println(yoda);
 }
 }

 public static void teszt2(){

 String yodaNull = null;

 if(yodaNull.equals("yodahola")){
 System.out.println(yodaNull);
 }
 }

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

```
teszt1();
teszt2();
}
}
```

Az első programrészről `teszt1`, semmi sem fog különösen történni, a programunk lefordul majd lefut és kiírja amit akarunk.

The screenshot shows the KDevelop IDE interface. On the left is the code editor with a Java file named `Yoda.java`. The code contains two methods, `teszt1` and `teszt2`, and a `main` method. The `teszt1` method prints "yodahola". The `teszt2` method attempts to print from a null variable `yodaNull`. The terminal window on the right shows the command `javac Yoda.java` followed by `java Yoda`, which outputs "yodahola".

```
teszt1();
teszt2();
}
}

Az első programrészről teszt1, semmi sem fog különösen történni, a programunk lefordul majd lefut és kiírja amit akarunk.

A második programrészről teszt2, a program lefordul, viszont futtatáskor a következőt kapjuk: Except i
in thread "main" java.lang.NullPointerException
```

A második programrészről `teszt2`, a program lefordul, viszont futtatáskor a következőt kapjuk: `Exception in thread "main" java.lang.NullPointerException ...`.

```

Activities Terminal Sept 15 15:20
/home/gyula/Documents/prog2/Yoda.java -- KDevelop
Session Project Run Navigation File Edit View Bookmarks Tools Code Window Settings Help Build Stop All Debug Execute Quick Open... Line: 25 Col: 17
Scratchpad Yoda.java
public class Yoda{
 // public static void teszt1(){
 // String yoda = "yodahola";
 // if (yoda.equals("yodahola")){
 // System.out.println(yoda);
 // }
 // }

 public static void teszt2(){
 String yodaNull = null;
 if(yodaNull.equals("yodahola")){
 System.out.println(yodaNull);
 }
 }

 public static void main(String[] args){
 // teszt1();
 teszt2();
 }
}

```

```

gyula@gyula-X550JK:~/Documents/prog2$ javac Yoda.java
gyula@gyula-X550JK:~/Documents/prog2$ java Yoda
Exception in thread "main" java.lang.NullPointerException
 at Yoda.teszt2(Yoda.java:16)
 at Yoda.main(Yoda.java:26)
gyula@gyula-X550JK:~/Documents/prog2$

```

## 12.5. Kódolás from scratch

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

Megoldás forrása:

Megoldás videó:

A pi kiszámítására adott a BBP formula:

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

12.1. ábra. BBP formula (forrás: [bbp-alg.pdf](http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf))

A “Bailey-Borwein-Plouffe” (BBP) algoritmust 1995-ben fedezték fel, majd 1996-ban publikálták. A BBP formulán alapszik, a következő példa ezt szemlélteti. Az main-ben létrehozunk egy  $y$  változót, ezt fogjuk majd kiíratni a végén, vagyis a Pi-t. Az main konstruktora kap egy számot, ez határozza majd meg, hogy milyen pontossággal kell dolgoznia a programnak, azaz a for ciklus hányszor fusson le. Ha egy adott  $x$  értékre kiszámoltuk a sort, akkor hozzáadjuk az összeg változóhoz, ezt  $i=x$ -szor ismételjük.

```
import java.util.Scanner;
public class Pi {

 public static void main(String[] args) {
 System.out.println("Add meg milyen pontossággal akarod megadni a Pi értékét?");
 String line = System.console().readLine();
 double x = Double.parseDouble(line);
 double y = 0.0;

 for(double i = 0; i <= x; i++) {
 y = y + (1.0/Math.pow(16.0,i))*((4.0/(8.0*i + 1.0))-(2.0/(8.0*i + 4.0)) -
 (1.0/(8.0*i + 5.0)) - (1.0/(8.0*i + 6.0)));
 }
 System.out.println("Pontosság: " + y);
 }
}
```

The screenshot shows the KDevelop IDE interface. On the left, the project tree displays a single file named 'Pi.java'. The code itself is a Java program that calculates the value of Pi using a series expansion. It prompts the user for the desired precision (number of digits after the decimal point), reads the input, and then iterates to calculate the value of Pi. The right side of the interface features a terminal window where the program is being run. The terminal output shows the user's command to run the Java application, followed by the calculated value of Pi (3.141592653589793) repeated multiple times, indicating the precision requested.

```
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
1
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
2
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
3
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
4
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
5
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
6
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
7
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
8
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
9
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
10
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$ java Pi
Add meg milyen pontossággal akarod megadni a Pi értékét?
11
Pontosság: 3.141592653589793
gyula@gyula-X550JK: ~/Documents/prog$
```

# 13. fejezet

## Helló, Liskov!

### 13.1. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. [https://arato.inf.unideb.hu/batfai.norbert/PROG2/Prog2\\_1.pdf](https://arato.inf.unideb.hu/batfai.norbert/PROG2/Prog2_1.pdf) (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPORG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Megoldás videó:

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/liskov/liskovsert.cpp](bhax/thematic_tutorials/bhax_textbook_sajat/liskov/liskovsert.cpp)

Láthatjuk, hogy ez így hibás.

```
// ez a T az LSP-ben
class Szamitogep {};
public:
 void teleprolmukodo() {};

// ez a két osztály alkotja a "P programot" az LPS-ben
class Program {
public:
 void fgv (Szamitogep &szamitogep) {}
 szamitogep.teleprolmukodo();

};

// itt jönnek az LSP-s S osztályok
class TeleprolMukodoSzm : public Szamitogep {
public:
};

class Laptop : public Szamitogep // ezt úgy is lehet olvasni, hogy a laptop ←
 tud telep nélkül működni
{};


```

```
int main (int argc, char **argv)
{
 Program program;
 Szamitogep szamitogep;
 program.fgv (szamitogep);

 Asztali asztali;
 program.fgv (asztali);

 Laptop laptop;
 program.fgv (laptop); // sérül az LSP, mert a P::fgv csak telep ←
 // nélkül működtetné a laptopot, ami ugye lehetetlen.
}
```

Viszont ha írjuk meg akkor már sokkal elfogadhatóbb.

```
// ez a T az LSP-ben
class Szamitogep {};
//public:
//void teleprolmukodo() {};

// ez a két osztály alkotja a "P programot" az LPS-ben
class Program {
public:
 void fgv (Szamitogep &szamitogep) {}

};

// itt jönnek az LSP-s S osztályok
class TeleprolMukodoSzm : public Szamitogep {
public:
 virtual void teleprolmukodo() {};

};

class Asztali: public TeleprolMukodoSzm
{};

class Laptop : public Szamitogep // ezt úgy is lehet olvasni, hogy a laptop ←
 // tud telep nélkül működni
{};

int main (int argc, char **argv)
{
 Program program;
 Szamitogep szamitogep;
 program.fgv (szamitogep);

 Asztali asztali;
```

```
 program.fgv (asztali);

 Laptop laptop;
 program.fgv (laptop);
}
```

## 13.2. Hello, Android!

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

Megoldás videó:

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/liskov/SMNISTFORHUM](https://bhax/thematic_tutorials/bhax_textbook_sajat/liskov/SMNISTFORHUM)

```
int[] bgColor =
{
 android.graphics.Color.rgb(54, 49, 49), //Itt ←
 változtattam a színeket
 android.graphics.Color.rgb(220, 0, 0) //
};
```

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

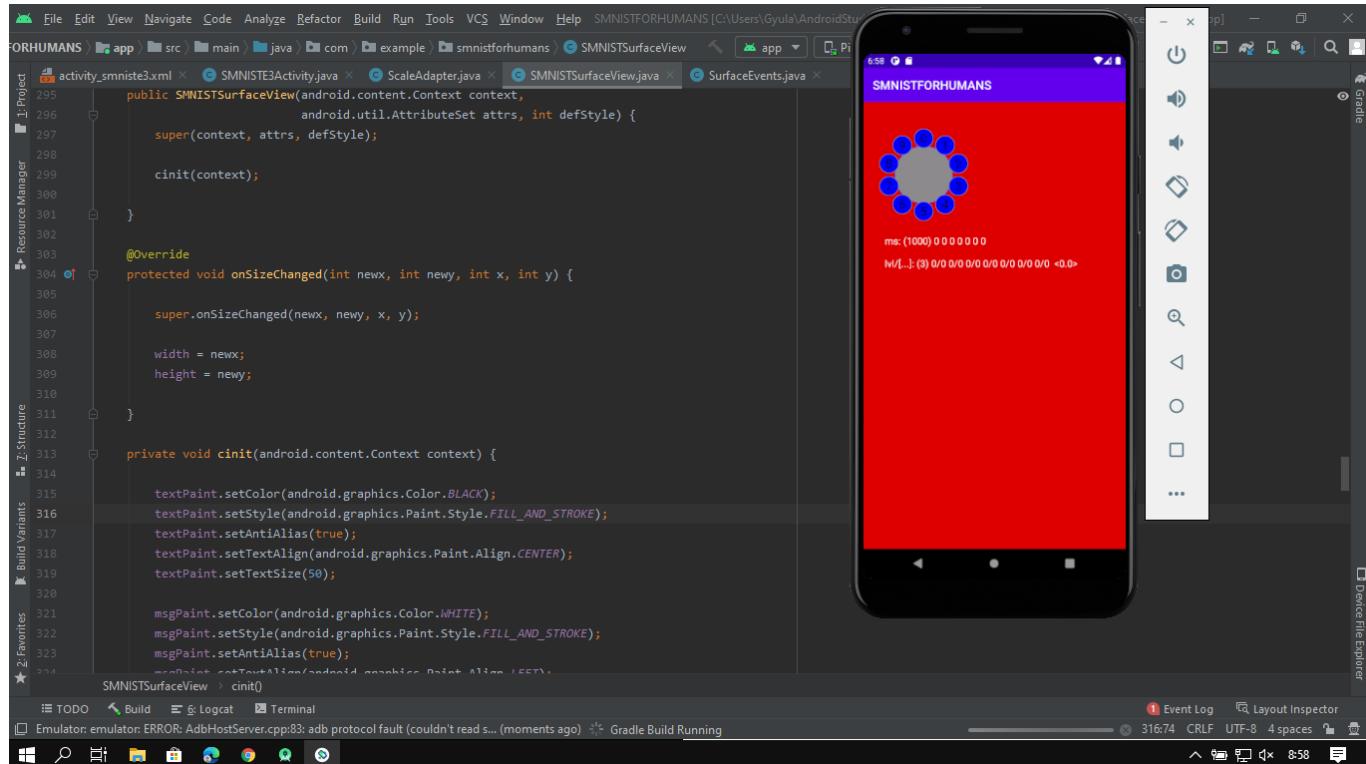
 textPaint.setColor(android.graphics.Color.BLACK); //Szürkéről ←
 feketére változtattam
 textPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
 textPaint.setAntiAlias(true);
 textPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
 textPaint.setTextSize(50);

 msgPaint.setColor(android.graphics.Color.WHITE); //Szürkéről ←
 fehérre változtattam
 msgPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
 msgPaint.setAntiAlias(true);
 msgPaint.setTextAlign(android.graphics.Paint.Align.LEFT);
 msgPaint.setTextSize(40);

 dotPaint.setColor(android.graphics.Color.BLACK);
 dotPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
 dotPaint.setAntiAlias(true);
 dotPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
 dotPaint.setTextSize(50);

 borderPaint.setStrokeWidth(2);
 borderPaint.setColor(android.graphics.Color.GRAY);
```

```
fillPaint.setStyle(android.graphics.Paint.Style.FILL);
fillPaint.setColor(android.graphics.Color.BLUE); //Citromsárgáról ←
kékre változtattam
```



### 13.3. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek! Lásd fóliák! [Prog2\\_1.pdf](#) (98. fólia)

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/liskov/szulo.java](https://bhax/thematic_tutorials/bhax_textbook_sajat/liskov/szulo.java)

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/liskov/szulo.cpp](https://bhax/thematic_tutorials/bhax_textbook_sajat/liskov/szulo.cpp)

Megoldás videó:

Az objektumorientált programozásban azért használjuk az öröklést, mert ezáltal egy létező osztály tulajdonságait alapul véve tudunk új osztályt létrehozni. Az új osztály (azaz a gyerekosztály) használhatja a szülőosztály tulajdonságait és metódusait, de felülbírálhatja azokat, illetve létre is hozhat újakat.

Java:

Minden objektum referencia, minden dinamikus a kötés. De ezzel nem küldhetjük a gyerekosztály által hozott új üzeneteket.

```
class szulo
{
```

```
protected int m_kor;
protected String m_nev;
public void setKor(int kor) {
 m_kor = kor;
}

public void setNev(String nev) {
 m_nev = nev;
}

public int getNev() {
 return m_kor;
}

class gyerek extends szulo
{
 public String getNev() {
 return m_nev;
 }
}

class szgy
{
 public static void main (String args[])
 {
 szulo s = new gyerek();
 s.setNev("Apa");
 s.setKor(60);

 gyerek gy = new gyerek();
 s.setNev("Geza");
 s.setKor(15);

 System.out.println(gy.getNev() + " " + s.getNev());
 }
}
```

A Java programunknál a következő hibába ütközünk :

The screenshot shows the KDevelop IDE interface. On the left, the Projects view lists a 'Scratchpad' project containing 'szulo.java' and 'szulo.cpp'. The code editor shows Java code with syntax highlighting. The terminal window on the right displays the command 'javac szulo.java' followed by an error message: 'szulo.java:20: error: getNev() in gyerek cannot override getNev() in szulo public String getNev() { ^ return type String is not compatible with int 1 error'. The terminal prompt is 'gyula@gyula-X550JK: ~/Documents/prog2\$'.

C++ :

Csak akkor van dinamikus kötés, ha a viselkedés virtuálisra van deklarálva. Ugyanúgy igaz, hogy ősosztály referenciaján vagy pointeren keresztül, csak az ős üzenetei küldhetőek.

```
#include <iostream>

class Teglalap
{
public:
 int m_szelesseg;
 int m_magassag;

 Teglalap(int szelesseg, int magassag) {
 m_szelesseg = szelesseg;
 m_magassag = magassag;
 }

 int getSzelesseg() {
 return m_szelesseg;
 }

 int getMagassag() {
 return m_magassag;
 }
};

class Negyzet: public Teglalap
{
public:
```

```
Negyzet(int oldal): Teglalap(oldal, oldal)
{
 m_szelesség = oldal;
 m_magasság = oldal;
}

int getTerulet(){
 return m_szelesség * m_magasság;
}

int main ()
{
 Teglalap* r = new Negyzet(10);
 Negyzet* s = new Negyzet(10);

 std::cout << s->getTerulet() << r->getTerulet() << std::endl;
}
```

A fordítás sikertelen lesz, ugyanis az altípus új függvényei nem lesznek elérhetőek.

The screenshot shows the KDevelop IDE interface. On the left, the code editor displays the `szulo.cpp` file with the following content:

```
#include <iostream>

class Teglalap
{
public:
 int m_szelesség;
 int m_magasság;

 Teglalap(int szélesség, int magasság) {
 m_szelesség = szélesség;
 m_magasság = magasság;
 }

 int getSzelesség(){
 return m_szelesség;
 }

 int getMagasság(){
 return m_magasság;
 }
};

class Negyzet: public Teglalap
{
public:
 Negyzet(int oldal): Teglalap(oldal, oldal) {
 m_szelesség = oldal;
 m_magasság = oldal;
 }

 int getTerulet(){
 return m_szelesség * m_magasság;
 }

 int main ()
 {
 Teglalap* r = new Negyzet(10);
 Negyzet* s = new Negyzet(10);

 std::cout << s->getTerulet() << r->getTerulet() << std::endl;
 }
}
```

On the right, a terminal window shows the compilation command and its error output:

```
gyula@gyula-X550JK: ~/Documents/prog2$ g++ szulo.cpp -o szulo
szulo.cpp: In function 'int main()':
szulo.cpp:42:40: error: 'class Teglalap' has no member named 'getTerulet'
 42 | std::cout << s->getTerulet() << r->getTerulet() << std::endl;
 ^~~~~~
```

## 13.4. Ciklomatikus komplexitás

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

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/liskov/](https://bhax/thematic_tutorials/bhax_textbook_sajat/liskov/)

Gráfelméletben a ciklomatikus komplexitás a vezérlési gráfban megtalálható független utak maximális számát jelenti. Két út független, ha minden kettőben létezik olyan pont vagy él, amelyik nem eleme a másik útnak. Képlete:  $M = E - N + 2P$ , ahol E a gráf éleinek száma, N a gráfban lévő csúcsok száma és P az összefüggő komponensek száma.

A ciklomatikus komplexitás egy metrika is, amely forráskód alapján, számértékben határozza meg a szoftverünk komplexitását. A korábban megírt programok függvényeinek vizsgálatára én a lizard alkalmazást használtam.

Elsőnek egy egyszerű programot néztem meg, a liskovos feladatot :

Function Name	NLOC	Complexity	Token #	Parameter #
Program::fgv	1	1	8	
TelepolMukodoSzm::telepolmukodo	1	1	5	
main	10	1	44	

Másodjára már egy komplex programot, a Binfát java változatban :

Function Name	NLOC	Complexity	Token #	Parameter #
Binfa::Binfa	3	1	9	
Binfa::addItem	24	4	142	
Binfa::Inorder	6	1	29	
Binfa::depthOut	5	1	19	
Binfa::avgOut	6	1	25	
Binfa::dispOut	5	1	27	
Binfa::maxDepth	3	1	8	
Binfa::treeAvg	3	1	10	
Binfa::Node::Node	7	1	39	
Binfa::Node::getLeftChild	3	1	8	
Binfa::Node::getRightChild	3	1	8	
Binfa::Node::newLeftChild	3	1	11	
Binfa::Node::newRightChild	3	1	11	
Binfa::Node::getChildValue	3	1	8	
Binfa::Node::getChildDepth	3	1	8	
Binfa::Node::getChildId	3	1	8	
Binfa::Depth	8	3	51	
Binfa::In	13	3	105	
Binfa::Avg	10	4	64	
Binfa::Disp	9	4	84	
Binfa::main	53	14	367	

### 13.5. Anti OO

A BBP algoritmussal 5 a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10 6, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket!

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apas03.html#id561066>

Ahhoz, hogy a futási időket megkapjuk terminálban, az eredeti kódban a d értékét kell állítanunk, nyelvtől függetlenül. Ha a 0. pozíciótól számított  $10^6$  darab jegyet szeretnénk meghatározni, akkor a for ciklusban kezdésképp a d-nek 1000000-t kell beírnunk.

```
for(int d=1000000; d<1000001; ++d) {

 d16Pi = 0.0d;

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

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

 d16Pi = d16Pi - Math.floor(d16Pi);

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

}
```

```
$ gyula@gyula-X550JK:~/Documents/prog2$./PiBBPBench1
6
1.777196
gyula@gyula-X550JK:~/Documents/prog2$ java PiBBPBench
6
1.549
```

The screenshot shows a web browser window with the title "Online Csharp Compiler - Online Csharp Editor - Online Csharp IDE - Csharp Coding Online - Practice Csharp Online - Execute Csharp Online - Compile Csharp Online - Run Csharp Online - Chromium". The URL is "tutorialspoint.com/compile\_csharp\_online.php". The page contains a code editor with C# code for calculating pi using the BBP algorithm, and a results panel showing the output of the compilation and execution.

```

63- if(n >= t) {
64- r = (10*r) % k;
65- n = n - t;
66- }
67-
68- t = t/2;
69-
70- if(t < 1)
71- break;
72-
73- r = (r+r) % k;
74-
75-}
76-
77- return r;
78-}
79-
80-/// A.CBBP ALGORITHM] David H. Bailey: The
81-/// BBP Algorithm for PI, alapján a
82-/// 4π = 4*[16^d * 54] - 2*[16^d * 55] + [16^d * 56]
83-/// kiszámítás a [1] a törzsezi jeléll. A PI hexa kifejezésében a
84-/// d41. hexa jegytel.
85-/// //rendszer
86-
87-public static void Main(System.String[]args) {
88-
89- double d16Pi = 0.0d;
90-
91- double d16S1t = 0.0d;
92- double d16S4t = 0.0d;
93- double d16S5t = 0.0d;
94- double d16S6t = 0.0d;
95-
96- int jegy = 0;
97-
98- System.DateTime kezd = System.DateTime.Now;
99- for(int d=10000000; d<10000001; ++d) {
100-
101- d16Pi = 0.0d;
102-
103- d16S1t = d16Sj(d, 1);
104- d16S4t = d16Sj(d, 4);
105- d16S5t = d16Sj(d, 5);
106- d16S6t = d16Sj(d, 6);
107-
108- d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;
109-
110- d16Pi = d16Pi - System.Math.Floor(d16Pi);
111-
112- jegy = (int)System.Math.Floor(16.0d*d16Pi);
113-
114- }
115-
116- System.Console.WriteLine(jegy);
117- System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
118- System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
119-}
120-
```

Látható, hogy ennél a "kis" értéknél még - a rendszerünk teljesítményétől függetlenül - rövid idő alatt megtörténik a futtatás. Nálam szinte egy időben, körülbelül 1.5-1.7 másodpercig tartott a bench program C, C++ és Java átiratának futtatása is. A C# átirat fordításához és futtatásához azonban online compiler használatához kellett folyamodnom, így az érték kb. 2,5 másodpercre nőtt.

Át is térhettünk a  $10^7$  darab jegy meghatározásához. Ahhoz, hogy az új értékhez tartozó futási időket kapjuk, a d változóban tárolt nullák számát meg kell növelnünk eggyel.

```

for(int d=10000000; d<10000001; ++d) {

 d16Pi = 0.0d;

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

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

 d16Pi = d16Pi - Math.floor(d16Pi);

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

}
```

Ha ez megvan, akkor az előbb látott módon indulhat a fordítás, majd a futtatás. Az eredeti értékeinkhez képest most jelentősen több időt vett igénybe a kódok lefutása. Nálam 3 és fél másodperc helyett ezúttal 18-20 másodpercret vett igénybe a folyamat a Java, C és C++ átiratoknál, míg a C# átirat 53 másodpercet.

```
$ gyula@gyula-X550JK:~/Documents/prog2$./PiBBPBench1
7
21.293616
gyula@gyula-X550JK:~/Documents/prog2$ java PiBBPBench1
7
18.576
```

The screenshot shows the Repl.it C# Online Compiler interface. On the left, there's a sidebar with various icons for different tools like a file manager, terminal, and help. The main area has tabs for 'Files' and 'Terminal'. The 'Files' tab shows a directory structure with 'main.cs' as the active file. The 'Terminal' tab shows the command to build and run the application: 'mcS -out:main.exe main.cs' and 'mono main.exe'. The code editor displays the following C# code:

```
public static void Main(System.String[] args) {
 double d16P1 = 0.0d;
 double d16S1t = 0.0d;
 double d16S4t = 0.0d;
 double d16S5t = 0.0d;
 double d16S6t = 0.0d;
 int jegy = 0;
 System.DateTime kezd = System.DateTime.Now;
 for(int d=10000000; d<10000001; ++d) {
 d16P1 = 0.0d;
 d16S1t = d16Sj(d, 1);
 d16S4t = d16Sj(d, 4);
 d16S5t = d16Sj(d, 5);
 d16S6t = d16Sj(d, 6);
 d16P1 = 4.0d*d16S1t + 2.0d*d16S4t + d16S5t - d16S6t;
 d16P1 = d16P1 - System.Math.Floor(d16P1);
 jegy = (int)System.Math.Floor(16.0d*d16P1);
 }
 System.Console.WriteLine(jegy);
 System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
 System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
}
```

A  $10^8$  darab jegy meghatározásához is csak a nullák számán kell változtatni az új értékek megszerzéséhez.

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

 d16Pi = 0.0d;

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

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

 d16Pi = d16Pi - Math.floor(d16Pi);

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

}
```

A korábbi, egy perc alatti értékeimhez képest most a C, C++ és a Java változat körülbelül 3-5 perc alatt

futott le. A C# átitrat online compiler használatával több, mint 10.5 perc alatt futott csak le.

```
$ gyula@gyula-X550JK:~/Documents/prog2$./PiBBPBench1
12
245.961797
gyula@gyula-X550JK:~/Documents/prog2$ java PiBBPBench
12
219.316
```

The screenshot shows the Repl.it C# Online Compiler IDE interface. On the left, there's a sidebar with various icons for file management, collaboration, and tools. The main area has tabs for 'Repl.it - C# Online Compiler' and 'replit/languages/csharp'. The code editor window contains 'main.cs' with the following C# code:

```
public static void Main(System.String[] args) {
 double d16Pi = 0.0d;
 double d16S1t = 0.0d;
 double d16S4t = 0.0d;
 double d16S5t = 0.0d;
 double d16S6t = 0.0d;
 int jegy = 0;
 System.DateTime kezd = System.DateTime.Now;
 for(int d=100000000; d<100000001; ++d) {
 d16Pi = 0.0d;
 d16S1t = d16Sj(d, 1);
 d16S4t = d16Sj(d, 4);
 d16S5t = d16Sj(d, 5);
 d16S6t = d16Sj(d, 6);
 d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;
 d16Pi = d16Pi - System.Math.Floor(d16Pi);
 jegy = (int)System.Math.Floor(16.0d*d16Pi);
 }
 System.Console.WriteLine(jegy);
 System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
 System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
}
```

To the right of the code editor is a terminal window showing the command-line output:

```
> ncs -out:main.exe main.cs
> mono main.exe
12
635.31079
```

## 14. fejezet

# Helló, Mandelbrot!

### 14.1. Reverse engineering UML osztálydiagram

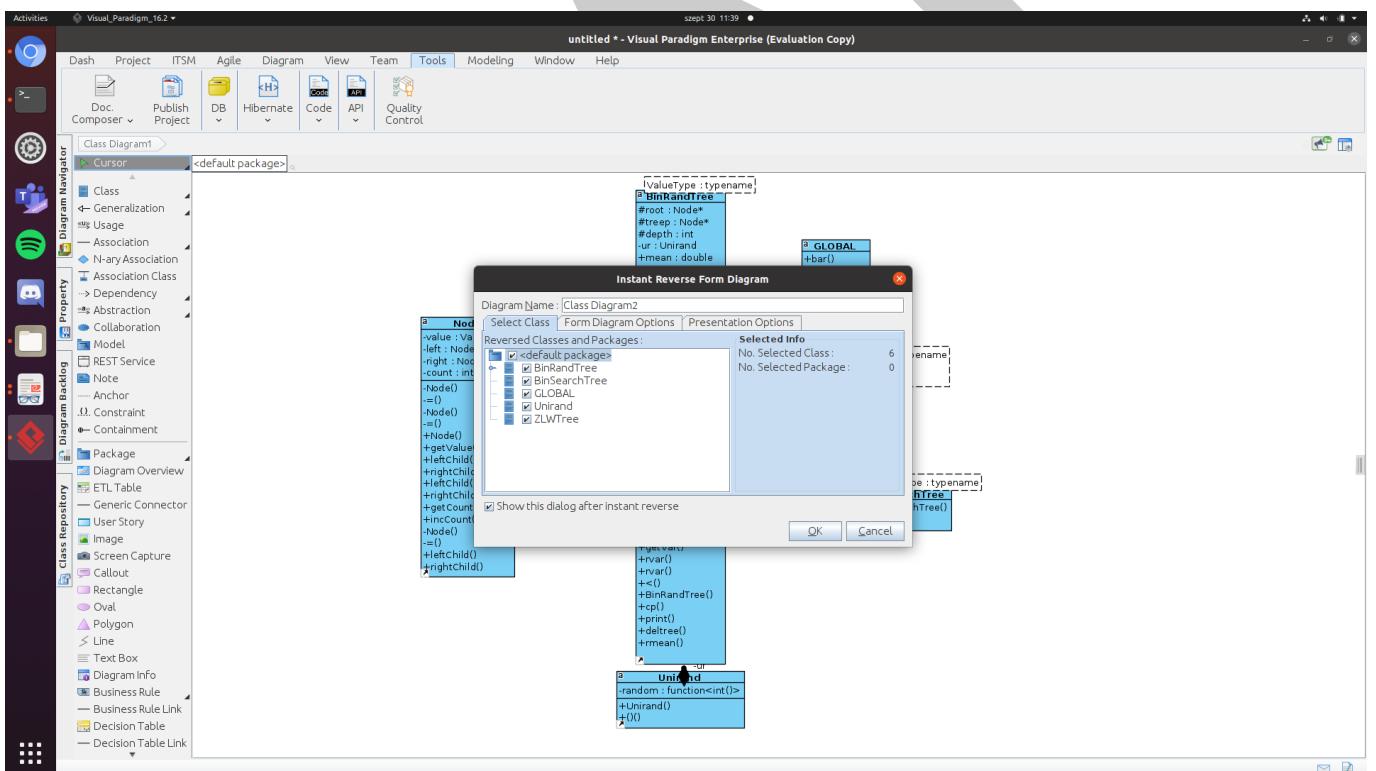
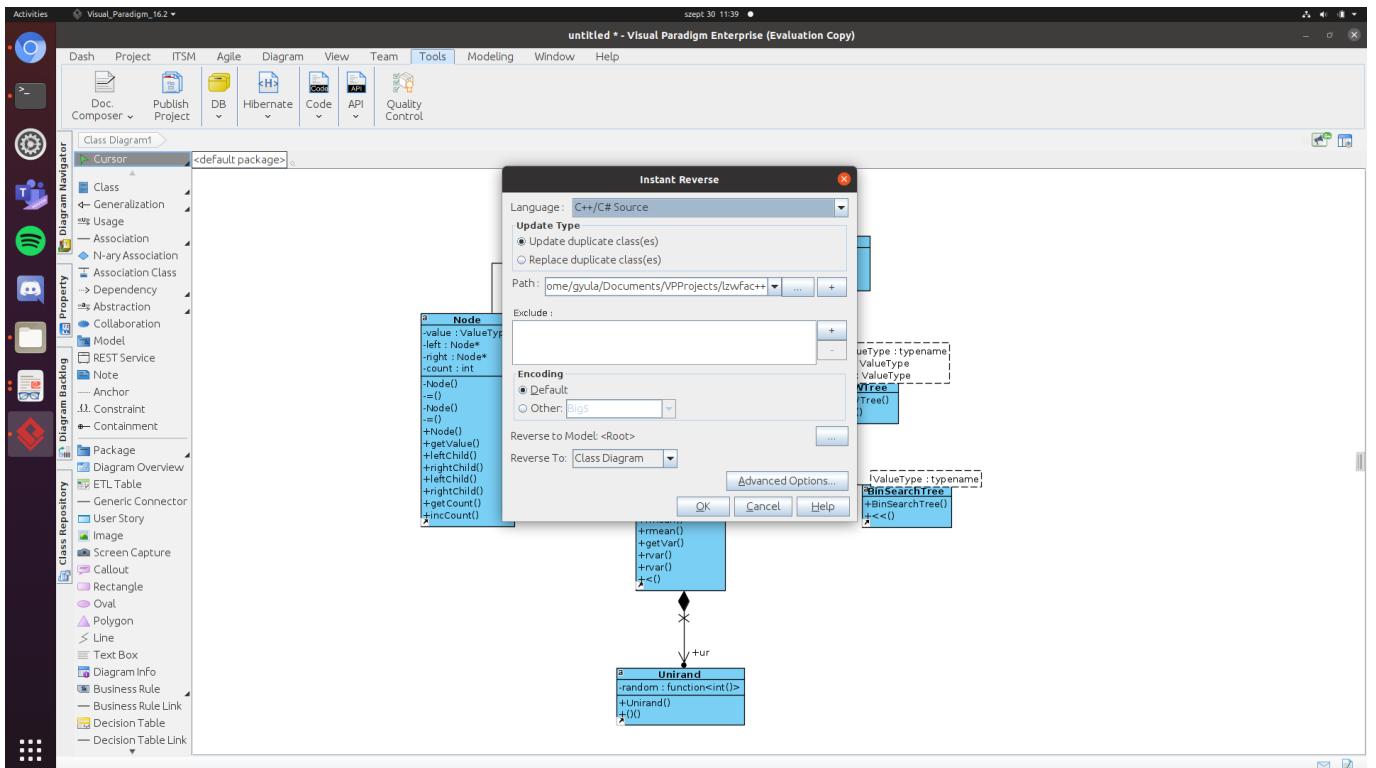
UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskóban és a diagramon, lásd még: [https://youtu.be/Td\\_nlERIEOs](https://youtu.be/Td_nlERIEOs)

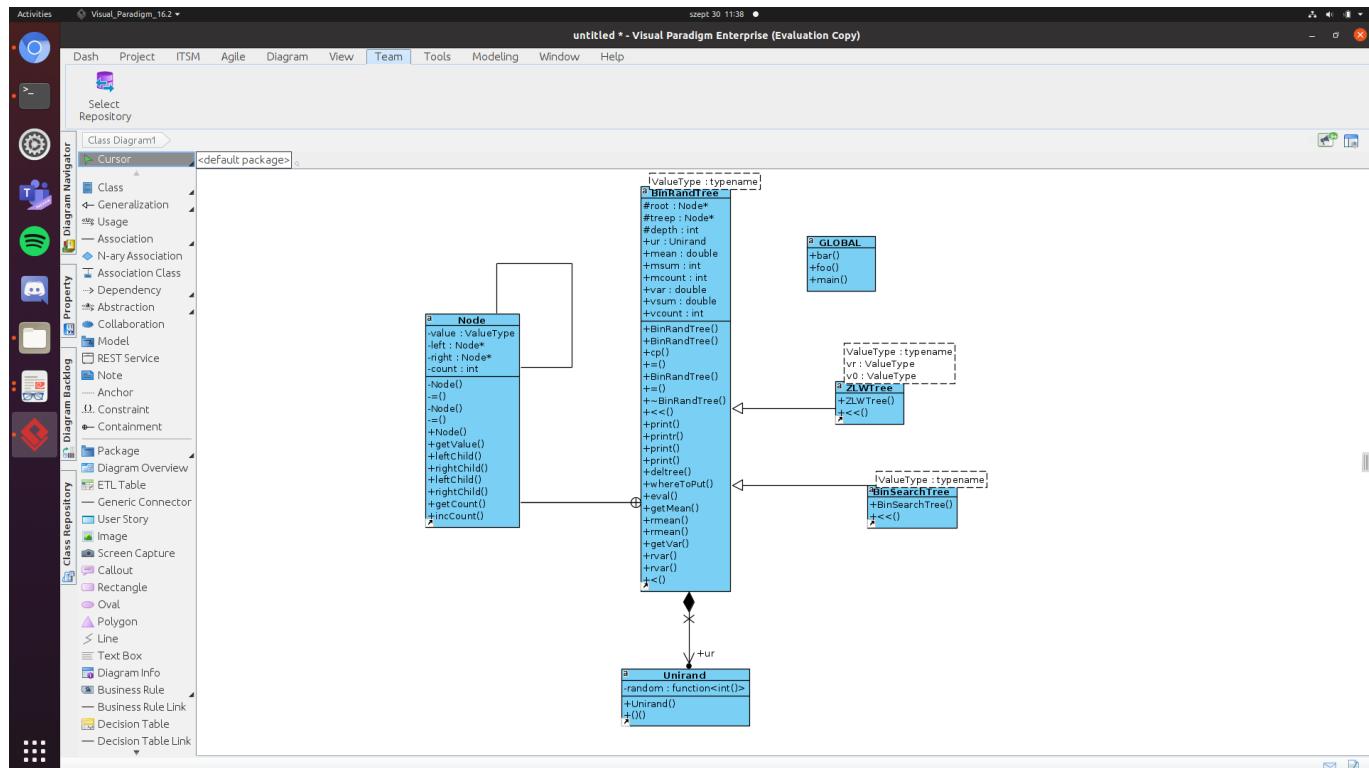
Forrás: [https://www.tankonyvtar.hu/hu/tartalom/tamop425/0027\\_RSZ4/ch01s02.html](https://www.tankonyvtar.hu/hu/tartalom/tamop425/0027_RSZ4/ch01s02.html)

Az UML (Unified Modeling Language) szabványos, általános célú modellező nyelv, üzleti elemzők, rendszertervezők, szoftvermérnökök számára. Grady Booch, Ivar Jacobson és James Rumbaugh egyesített munkájának terméke. Az objektum orientált (OO) modellezés módszerét alkalmazni lehet a való világ bonyolultságának leírására. Az UML egy gyakorlati, objektum orientált modellező megoldás, nagy méretű programrendserek modelljeinek vizuális dokumentálására alkalmas eszköz. Az UML módszer és leíró nyelv segítségével különböző nézőpontú szöveges és grafikus modellek készíthetők

—Wikipedia

Az általam választott program a Visual Paradigm volt. A Tools-ban a Code-ra kattintva elérhetjük az Instant Reverse funkciót, mely lehetővé teszi a C++ forráskódok azonnali átalakítását UML osztálydiagrammá.





## 14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/mandelbrot2/umltoco](https://bhax/thematic_tutorials/bhax_textbook_sajat/mandelbrot2/umltoco)

A Visual Paradigm segítségével az előző feladatban C++ kódóból UML osztálydiagram készült. Ez azonban fordítva is elvégezhető, tehát a diagram alapján új, egyedi C++ kódot kapunk.

A forward engineeringhez ismét a Tools-ban kell keresgélünk. A Code-ra kattintva megkapjuk az Instant Generator funkciót, amely az általunk megadott mappába generálja a diagram alapján kódot.

A header fájlok meglepően pontosra sikeredtek, szinte ugyanazt a kódot kaptuk a diagramból, mint amit korábban megírtunk C++-ban. Az UML forward engineering nagy hátránya viszont, hogy a függvények törvse üresen marad, hiszen a Visual Paradigm kizárolag a diagram alapján dolgozta ki a kódot.

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

#ifndef __BinRandTree_h__
#define __BinRandTree_h__

// #include "Node.h"
#include "Unirand.h"

class Node;
class Unirand;
template <typename ValueType> class BinRandTree;
```

```
template <typename ValueType> class BinRandTree
{
protected: Node* _root;
protected: Node* _treep;
protected: int _depth;
public: Unirand _ur;
public: double _mean;
public: int _msum;
public: int _mcount;
public: double _var;
public: double _vsum;
public: int _vcount;
public: Node* _unnamed_Node_;

public: BinRandTree(Node* aRoot = nullptr, Node* aTreep = nullptr);

public: BinRandTree(const BinRandTree& aOld);

public: Node* cp(Node* aNode, Node* aTreep);

public: BinRandTree& _(const BinRandTree& aOld);

public: BinRandTree(BinRandTree&& aOld);

public: BinRandTree& _(BinRandTree&& aOld);

public: void _BinRandTree();

public: BinRandTree& _<(ValueType aValue);

public: void print();

public: void printr();

public: void print(Node* aNode, std::ostream& aOs);

public: void print(const Node& aNode, std::ostream& aOs);

public: void deltree(Node* aNode);

public: int whereToPut();

public: void eval();

public: double getMean();

public: void rmean();

public: void rmean(Node* aUnnamed_1);
```

```

public: double getVar();

public: void rvar();

public: void rvar(Node* aUnnamed_1);

```

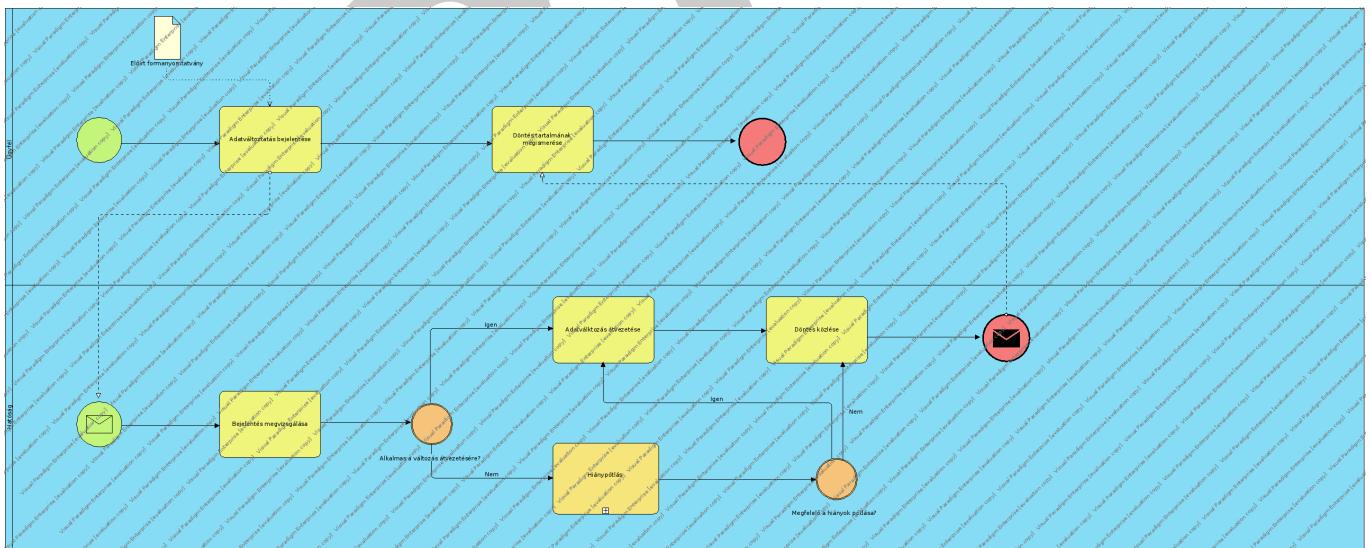
## 14.3. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! [https://arato.inf.unideb.hu/batfai.norbert/PROG2/Prog2\\_7.pdf](https://arato.inf.unideb.hu/batfai.norbert/PROG2/Prog2_7.pdf) (34-47 fólia)

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/mandelbrot2/](bhax/thematic_tutorials/bhax_textbook_sajat/mandelbrot2/)

A BPMN (Business Process Model and Notification) egy egységes folyamatábra alapú jelölés üzleti folyamatok modellezéséhez. Célja az egységes grafikus jelölés az üzleti folyamat ábrázolásához. Nem terjed ki stratégiák, üzleti szabályzatok modellezésére. Ilyen folyamatábrát tudunk a Visual Paradigm segítségével is készíteni, én is ezt használtam, ugyanis a reverse és forward engineering feladatok esetében jól bevált.

Az ábrámon két sáv látható: az egyik az ügyfél helyzetét mutatja, a másik pedig a hatóságét, aki várja az adatváltozás bejelentését. Ha érkezik egy új bejelentés, egy hatósági személy személy először megvizsgálja a bejelentést, majd eldönti hogy alkalmas-e változtatásra. A két lehetséges végkifejlet közül a pozitívabb az, ha minden rendben van, az adatváltozás megtörténik, közlik a döntést, majd az ügyfél ezután megismeri a hatóság által hozott döntést. Ellenkező esetben viszont elutasítják, ekkor a hatóság hiányok pótlására kéri meg az ügyfélét. Ha ez megfelelő akkor elfogadják és közlik a döntést, valamit az ügyfél megismeri a döntést. Ha ez nem megfelelő, akkor közlik a döntést az ügyféllel, majd az ügyfél megismeri ezt a döntést.

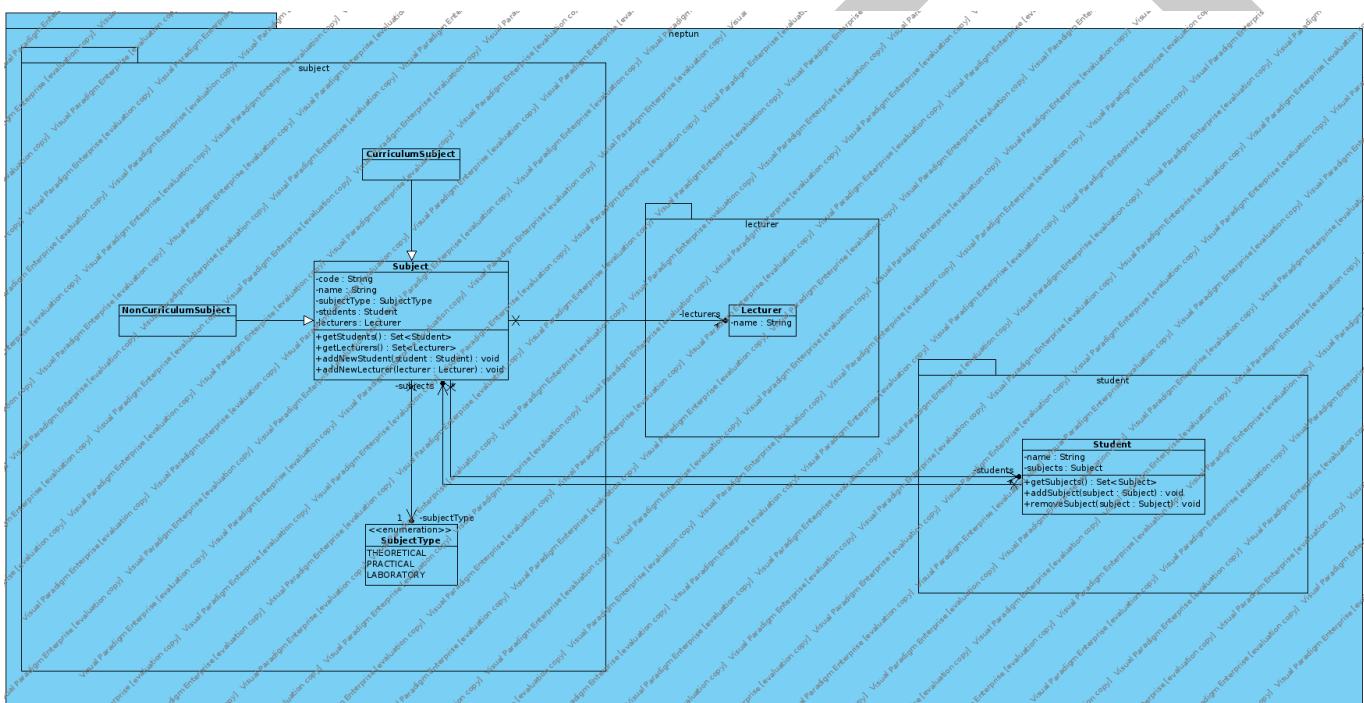


Az UML (Unified Modeling Language) szabványos, általános célú modellező nyelv, üzleti elemzők, rendszertervezők, szoftvermérnökök számára. Grady Booch, Ivar Jacobson és James Rumbaugh egyesített munkájának terméke. Az objektum orientált (OO) modellezés módszerét alkalmazni lehet a való világ bonyolultságának leírására. Az UML egy gyakorlati, objektum orientált modellező megoldás, nagy méretű programrendserek modelljeinek vizuális dokumentálására alkalmas eszköz. Az UML módszer és leíró nyelv segítségével különböző nézőpontú szöveges és grafikus modellek készíthetők

—Wikipedia

Ugyebár van három fő class-unk. A lecturer, a student, és a subject. A subject-nek vannak gyerekei: a curriculum, és a noncuriculum.

A következő feladatban pedig java nyelven implementáljuk a alábbi diagrammot.



## 14.5. EPAM: Neptun tantárgyfelvétel UML diagram implementálása

Implementáld le az előző feladatban létrehozott diagrammot egy tetszőleges nyelven.

Megoldás forrása: <https://github.com/epam-deik-cooperation.epam-deik-prog2/tree/master/week-3/uml-modelling>

Ugyebár van három fő class-unk. A lecturer, a student, és a subject. A subject-nek vannak gyerekei: a curriculum, és a noncuriculum.

```

package neptun.subject;

import neptun.lecturer.Lecturer;
import neptun.student.Student;
import java.util.Set;

```

```
public class Subject {

 private String code;
 private String name;
 private SubjectType subjectType;
 private Set<Student> students;
 private Set<Lecturer> lecturers;

 public String getCode() {
 return code;
 }

 public String getName() {
 return name;
 }

 public SubjectType getSubjectType() {
 return subjectType;
 }

 public Set<Student> getStudents() {
 return students;
 }

 public Set<Lecturer> getLecturers() {
 return lecturers;
 }

 public void addNewStudent(Student student) {
 this.students.add(student);
 }

 public void addNewLecturer(Lecturer lecturer) {
 this.lecturers.add(lecturer);
 }
}
```

Láthatjuk, hogy a subject class-unk kap egy code-ot, egy nevet, a subjecttype speciális osztályból(enum: amely egy olyan speciális class amely a contansok egy csoportját reprezentálja). Megkapja a student-ükből a student-et, a lecturer-ből a lecturers-t, valami hozzáad új student-eket, és új lecturer-eket.

```
package neptun.student;

import neptun.subject.Subject;
import java.util.Set;

public class Student {
```

```
private String name;
private Set<Subject> subjects;

public String getName() {
 return name;
}

public Set<Subject> getSubjects() {
 return subjects;
}

public void addSubject(Subject subject) {
 this.subjects.add(subject);
}

public void removeSubject(Subject subject) {
 this.subjects.remove(subject);
}

}
```

Láthatjuk, hogy a student class-unk kap egy egy nevet, valamint a subject classból átveszi a subject-eket. Továbbá a student hozzá tud adni subject-et, valamint azt el is tudja távolítani.

```
package neptun.lecturer;

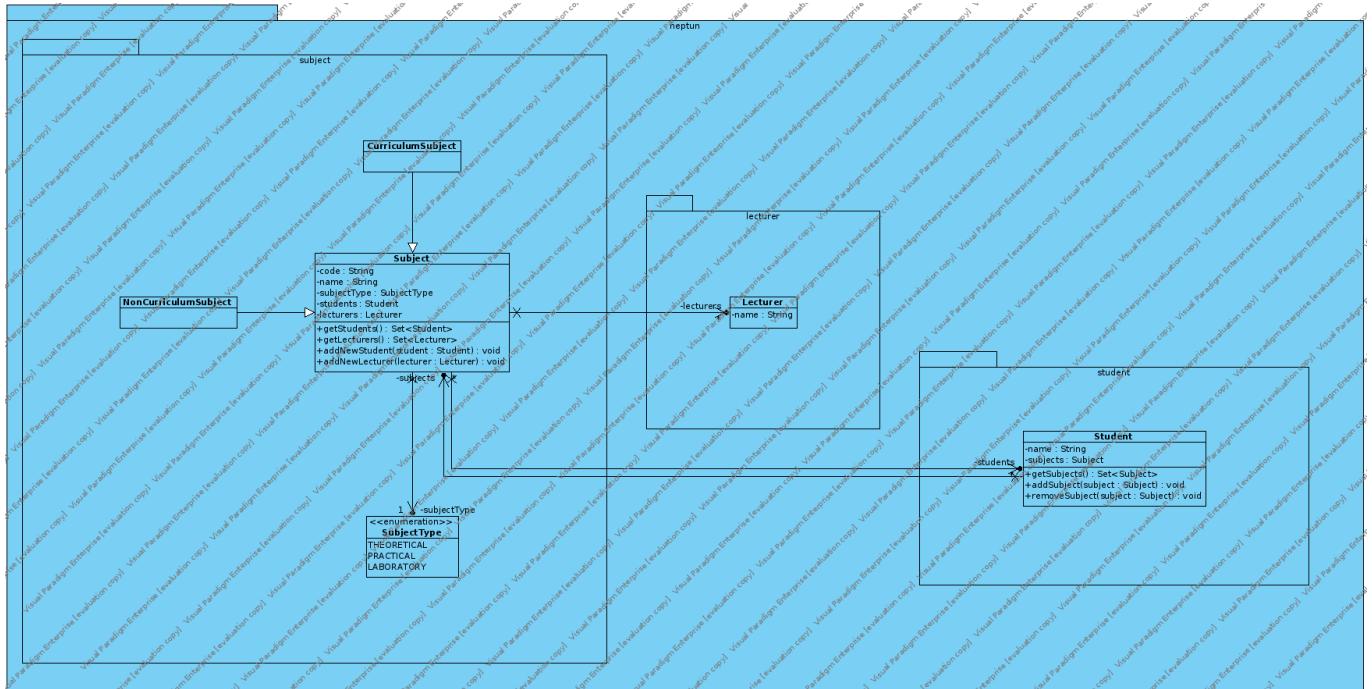
public class Lecturer {

 private String name;

 public String getName() {
 return name;
 }

}
```

A lecturer class-t nem részletezném, mert az triviális.



## 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 ékezetes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Megoldás videó:

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Chomsky2/Mandelbrot.java](bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky2/Mandelbrot.java)

A programot a **javac MandelbrotHalmazNagyító.java** parancsal fordítjuk, majd a futtatni a **java MandelbrotHalmazNagyító** parancsal tudjuk.

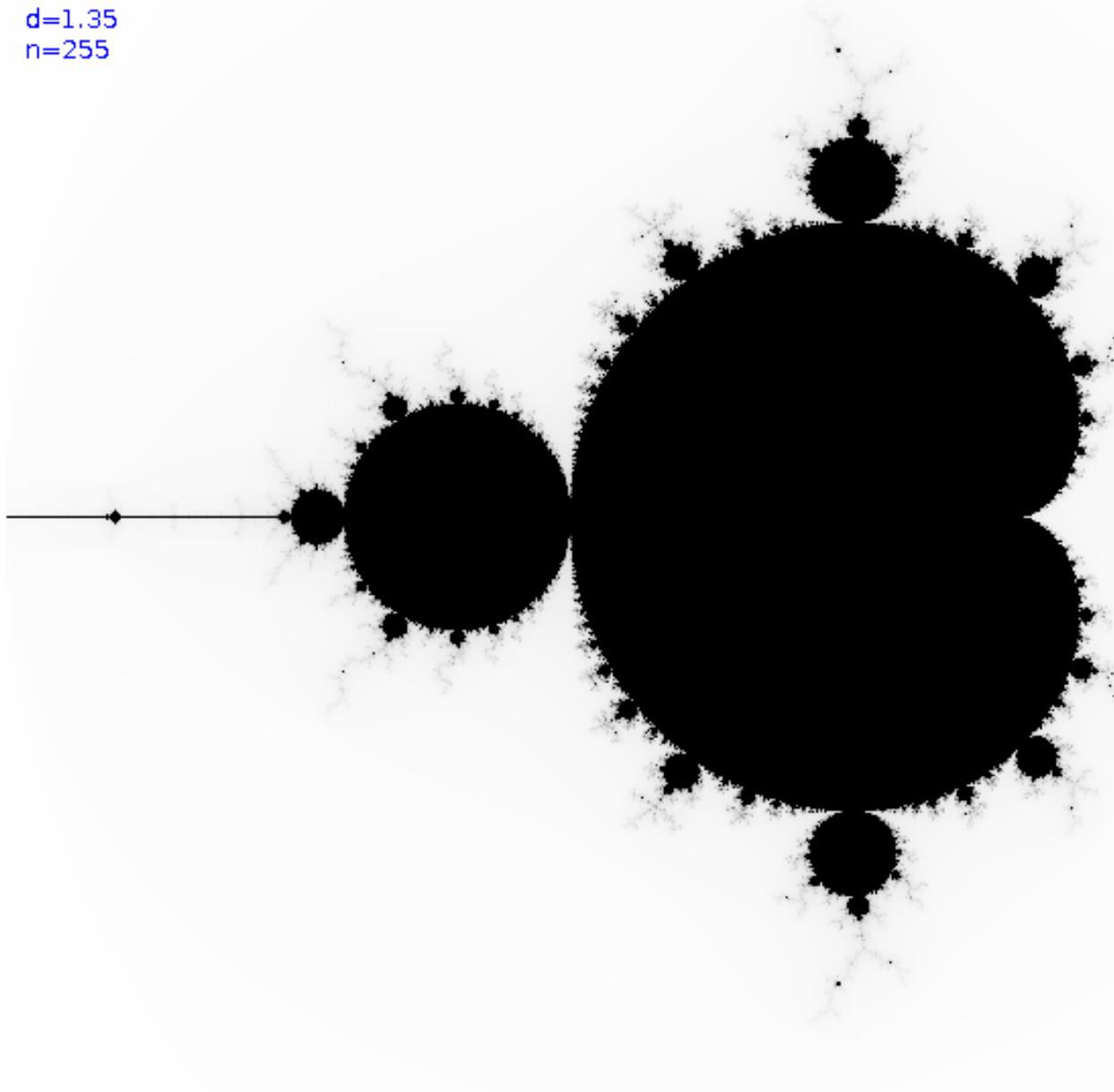
Az **s** billentyű lenyomásával egy felvétel kép készül az aktív ablakban számolt fraktálról.

Az **n** billentyű lenyomásával növeljük a halmaz kiszámolásának pontosságát, 256-al megnöveljük az iterációk számát.

Az **m** billentyű lenyomásával nagyobb ugrással növeljük a halmaz kiszámolásának pontosságát, 10\*256-al megnöveljük az iterációk számát.

Az **egérmutató bal gombjával** kijelöljük a nagyítandó terület bal felső sarkát, az egér vonszolásával megadjuk a terület nagyságát, az egérgomb felengedésére új ablakban megkezdődik a kijelölt terület nagyítása.

a=-2.0  
b=0.7  
c=-1.35  
d=1.35  
n=255



## 15.2. I334d1c4

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

Megoldás forrása:[bhaxx/thematic\\_tutorials/bhaxx\\_textbook\\_sajat/Chomsky2/1334d1c4.java](https://bhaxx.github.io/thematic_tutorials/bhaxx_textbook_sajat/Chomsky2/1334d1c4.java)

Megoldás videó:

A programhoz a beolvasás és a random számok generálása miatt kellenki fog 4 import sor:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Random;
```

A LexerData osztály az adott karakterekhez tartozó lehetséges megfelelőket tartalmazza tömbökben.

```
class LexerData {
 String[] a = {"4", "4", "@", "/-\\"};
 String[] b = {"b", "8", "|3", "|"};
 String[] c = {"c", "(", "<", "{"};
 String[] d = {"d", "|)", "|]", "|)"};
 String[] e = {"3", "3", "3", "3"};
 String[] f = {"f", "|=", "pf", "|#"};
 String[] g = {"g", "6", "[", "[+"};
 String[] h = {"h", "4", "|-", "|-", "-"};
 String[] i = {"1", "1", "|", "!"};
 String[] j = {"j", "7", "|-", "|/"};
 String[] k = {"k", "|<", "1<", "|{"};
 String[] l = {"l", "1", "|", "|_"};
 String[] m = {"m", "44", "(V)", "|\\|/|"};
 String[] n = {"n", "|\\|", "/\\/", "/V"};
 String[] o = {"0", "0", "()", "["};
 String[] p = {"p", "/o", "|D", "|o"};
 String[] q = {"q", "9", "O_", "(,)"};
 String[] r = {"r", "12", "12", "|2"};
 String[] s = {"s", "5", "$", "$"};
 String[] t = {"t", "7", "7", "'|'"};
 String[] u = {"u", "|_|", "(_)", "[_]"};
 String[] v = {"v", "\\|/", "\\\/", "\\|/"};
 String[] w = {"w", "VV", "\\\|\\\|/", "(/\\\\")"};
 String[] x = {"x", "%", ")(","(");
 String[] y = {"y", "y", "y", "y"};
 String[] z = {"z", "2", "7_",">_"};
 String[] a0 = {"D", "0", "D", "0"};
 String[] a1 = {"I", "I", "L", "L"};
 String[] a2 = {"Z", "Z", "Z", "e"};
 String[] a3 = {"E", "E", "E", "E"};
 String[] a4 = {"h", "h", "A", "A"};
 String[] a5 = {"S", "S", "S", "S"};
 String[] a6 = {"b", "b", "G", "G"};
 String[] a7 = {"T", "T", "j", "j"};
 String[] a8 = {"X", "X", "X", "X"};
 String[] a9 = {"g", "g", "j", "j"};
}
```

A LexerJava osztályban példányosítjuk az előzőt, majd egy Random-ot is. Beolvassuk a sort, ami ha üres, akkor vége is a programnak, ha nem, akkor viszont kisbetűssé alakítjuk. A kesz String-be épül fel majd az új szó. Végigmegyünk a beolvasott soron, és sorra megvizsgáljuk a karaktereit: ha pl. a, akkor egy neki meg-

feleltetett karaktert választunk véletlenszerűen. Ha az olvasott karakter olyan, aminek nincs megfeleltetés (pl. space), akkor azt átalakítás nélkül írjuk hozzá a szóhoz és a kész szót kiírjuk.

```
public class 1334d1c4 {

 LexerData ld = new LexerData();
 Random rand = new Random();

 public 1334d1c4() {

 while (true) {
 String sor = readLine();
 if(sor.isEmpty())
 break;
 sor = sor.toLowerCase();
 String kesz = "";
 for(int i = 0; i < sor.length(); ++i) {
 char c = sor.charAt(i);
 if(c == 'a') {
 kesz += ld.a[getRandInRange(0, 4)];
 } else if (c == 'b') {
 kesz += ld.b[getRandInRange(0, 4)];
 } else if (c == 'c') {
 kesz += ld.c[getRandInRange(0, 4)];
 } else if (c == 'd') {
 kesz += ld.d[getRandInRange(0, 4)];
 } else if (c == 'e') {
 kesz += ld.e[getRandInRange(0, 4)];
 } else if (c == 'f') {
 kesz += ld.f[getRandInRange(0, 4)];
 } else if (c == 'g') {
 kesz += ld.g[getRandInRange(0, 4)];
 } else if (c == 'h') {
 kesz += ld.h[getRandInRange(0, 4)];
 } else if (c == 'i') {
 kesz += ld.i[getRandInRange(0, 4)];
 } else if (c == 'j') {
 kesz += ld.j[getRandInRange(0, 4)];
 } else if (c == 'k') {
 kesz += ld.k[getRandInRange(0, 4)];
 } else if (c == 'l') {
 kesz += ld.l[getRandInRange(0, 4)];
 } else if (c == 'm') {
 kesz += ld.m[getRandInRange(0, 4)];
 } else if (c == 'n') {
 kesz += ld.n[getRandInRange(0, 4)];
 } else if (c == 'o') {
 kesz += ld.o[getRandInRange(0, 4)];
 } else if (c == 'p') {
 kesz += ld.p[getRandInRange(0, 4)];
 }
 }
 System.out.println(kesz);
 }
 }

 private int getRandInRange(int min, int max) {
 return rand.nextInt(max - min + 1) + min;
 }
}
```

```
 } else if (c == 'q') {
 kesz += ld.q[getRandInRange(0, 4)];
 } else if (c == 'r') {
 kesz += ld.r[getRandInRange(0, 4)];
 } else if (c == 's') {
 kesz += ld.s[getRandInRange(0, 4)];
 } else if (c == 't') {
 kesz += ld.t[getRandInRange(0, 4)];
 } else if (c == 'u') {
 kesz += ld.u[getRandInRange(0, 4)];
 } else if (c == 'v') {
 kesz += ld.v[getRandInRange(0, 4)];
 } else if (c == 'w') {
 kesz += ld.w[getRandInRange(0, 4)];
 } else if (c == 'x') {
 kesz += ld.x[getRandInRange(0, 4)];
 } else if (c == 'y') {
 kesz += ld.y[getRandInRange(0, 4)];
 } else if (c == 'z') {
 kesz += ld.z[getRandInRange(0, 4)];
 } else if (c == '0') {
 kesz += ld.a0[getRandInRange(0, 4)];
 } else if (c == '1') {
 kesz += ld.a1[getRandInRange(0, 4)];
 } else if (c == '2') {
 kesz += ld.a2[getRandInRange(0, 4)];
 } else if (c == '3') {
 kesz += ld.a3[getRandInRange(0, 4)];
 } else if (c == '4') {
 kesz += ld.a4[getRandInRange(0, 4)];
 } else if (c == '5') {
 kesz += ld.a5[getRandInRange(0, 4)];
 } else if (c == '6') {
 kesz += ld.a6[getRandInRange(0, 4)];
 } else if (c == '7') {
 kesz += ld.a7[getRandInRange(0, 4)];
 } else if (c == '8') {
 kesz += ld.a8[getRandInRange(0, 4)];
 } else if (c == '9') {
 kesz += ld.a9[getRandInRange(0, 4)];
 } else {
 kesz += c;
 }
 }

 System.out.println(kesz);
}
```

A getRandInRange() függvény a megadott határok között generál egészet, a mi esetünkben a 4 elemű tömbök és indexeik miatt ezek 0 és 4.

```
public int getRandInRange(int min, int max) {
 return rand.ints(min, max).findAny().getAsInt();
}
```

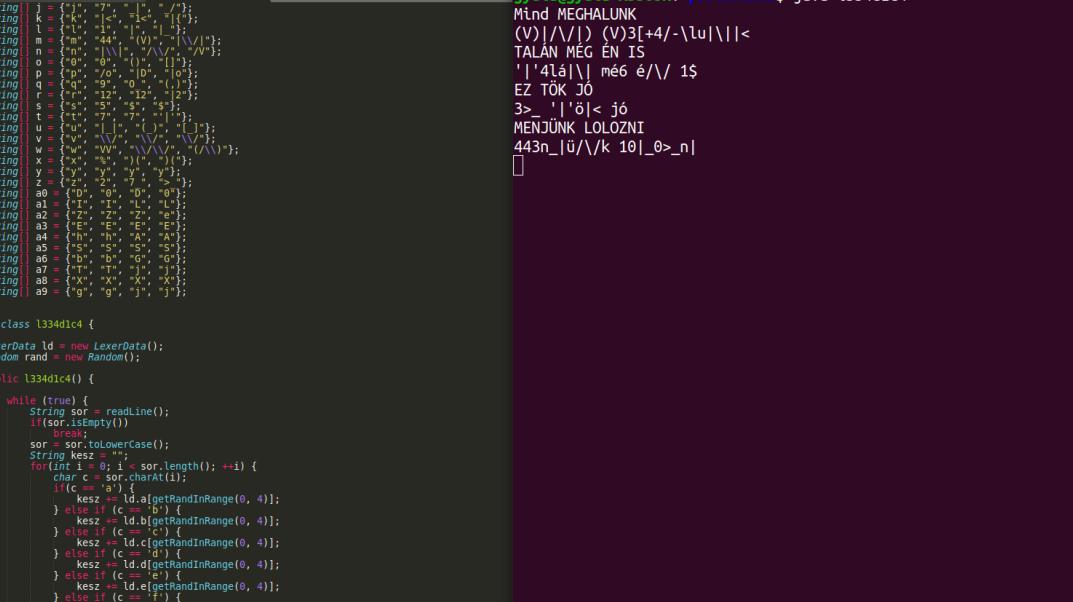
A readLine() függvény olvassa be az adott sort, és adja vissza Stringként.

```
public String readLine() {
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 String line = "";
 try {
 line = br.readLine();
 } catch (IOException e) {
 e.printStackTrace();
 }
 return line;
}
```

A main-ben csak az osztály konstruktörét hívjuk.

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

 new L337d1c4();
}
```



The screenshot shows a terminal window with the following session:

```
gyula@gyula-X550JK:~/Documents$ javac l334d1c4.java
gyula@gyula-X550JK:~/Documents$ java l334d1c4
Mind MEGHALUNK
(V) / (V) (V) 3 [+ - \] lu \ | | <
TALÁN MÉG ÉN IS
' ' 4 lá | | m é 6 é / \ / 1 $
EZ TÖK JÓ
3 > _ ' | ' | < jó
MENJÜNK LOLOZNI
443n_ü / \ k 10 | _ > n
```

15.1. ábra. Java l334d1c4

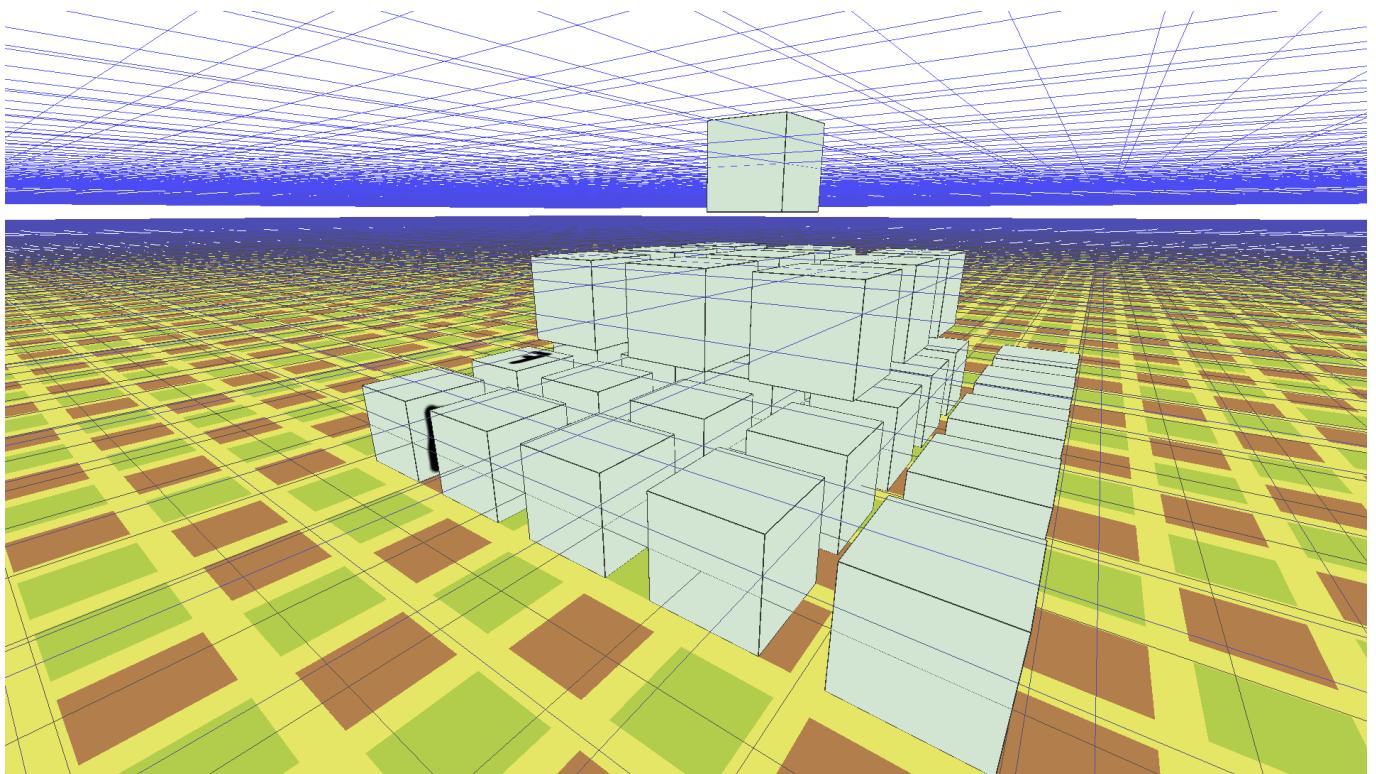
### **15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció**

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

## Megoldás videó:

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Chomsky2/para12.cpp](https://bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky2/para12.cpp)

A program fordítását a **g++ para12.cpp -o para -lboost\_system -lGL -lGLU -lglut -I/usr/include/SDL2 -lSDL2 -lSDL2\_image** parancssal hajtjuk végre.



## 15.4. Perceptron osztály

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

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Chomsky2/perceptron.cpp](https://bhax/thematic_tutorials/bhax_textbook_sajat/Chomsky2/perceptron.cpp)

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

—Wikipedia

Perceptron felépítése három fő részből áll: 1. Retina - bemeneti jeleket fogadó cellákat tartalmazza. 2. Asszociatív cellák - összegzik a hozzájuk érkező jeleket, impulzusokat. 3. Perceptronok kimenetele - összegzi a hozzájuk érkező jeleket, impulzusokat.

Feladatunk pontosan az lesz, hogy adjunk egy képet bemenetként, MandelbrotHalmazos kép kézre esik, és ennek a képnak fogja venni az egyik színkódját ami majd a többrétegű perceptronunk bemenete lesz.

Szükségünk lesz a többrétegűségre ezért kell az mlp (Multi Layer Perceptron) könyvtár, és PNG képpel fogunk dolgozni ezért a png könyvtárra is szükség lesz:

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

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

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

 Perceptron* p = new Perceptron(3, size, 256, 1);
```

Létrehouunk egy double típusú változót, valamint a for ciklusok végig mennek a kép szélesség és magassági pontokon. A képpontokat image tárolni fogja a képállomány vörös színkomponensét, ahogyan azt a kódban megadtuk, a value pedig azt a double típusú számot amit majd kiíratunk a végén.

```
double* image = new double[size];

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

double value = (*p) (image);

std::cout << value << std::endl;
```

Memóriában felszabadítjuk a szükséges helyet:

```
delete p;
delete [] image;

}
```

A program fordítása és futtatása pedig így tehető meg:

```
gyula@gyula-X550JK:~/Documents/prog2/chomsky$ g++ mlp.hpp perceptron.cpp -o ←
 perc -lpng
gyula@gyula-X550JK:~/Documents/prog2/chomsky$./perc mandel.png
0.5
gyula@gyula-X550JK:~/Documents/prog2/chomsky$./perc mandel.png
0.504763
```



A screenshot of a Linux desktop environment showing a terminal window, a code editor, and a file viewer.

The terminal window (top right) shows the command:

```
gyula@gyula-X550JK:~/Documents/prog2/chomsky$ g++ mlp.hpp perceptron.cpp -o perc -lpng
gyula@gyula-X550JK:~/Documents/prog2/chomsky$./perc mandel.png
0.5
gyula@gyula-X550JK:~/Documents/prog2/chomsky$ 6.564763
gyula@gyula-X550JK:~/Documents/prog2/chomsky$
```

The code editor (left) displays the C++ source code for `perceptron.cpp`:

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

int main (int argc, char **argv)
{
 png::image<rgb_pixel> png_image (argv[1]);
 int size = png_image.get_width() * png_image.get_height();
 Perceptron* p = new Perceptron(3, size, 256, 1);
 double* image = new double [size];
 for(int i {0}; i < png_image.get_width(); ++i)
 for(int j {0}; j < png_image.get_height(); ++j)
 image[i * png_image.get_width() + j] = png_image[i][j].red;
 double value = (*p) (image);
 std::cout << value << std::endl;
 delete p;
 delete [] image;
}
```

The file viewer (bottom center) displays the generated image `mandel.png`, which is a fractal pattern resembling the Mandelbrot set.

# 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: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Stroustrup/jdk.cpp](https://bhax/thematic_tutorials/bhax_textbook_sajat/Stroustrup/jdk.cpp)

A mainben, először is megadjuk a mappa nevét, azaz srczip, a path metódusnak(p), majd azt vizsgáljuk, a p-t, hogy létezik e egyáltalán, és könyvtár-e, (negált formában végezzük az ellenőrzést). Amennyiben minden rendben van, jön a read\_acts metódus meghívása a p-re, melyet a fénykard feladatból szereztünk. Egy kis magyarázat a readacts-hez, a readacts metódus argumentumként, egy path változót várt, lényegében a metódus annyit csinál, hogy megnézi az elérési útból származó file kiterjesztését, hogy .java-e? A vizsgálatot a !ext.compare(extension(path))) itt azért van a negálás, hogy az if-nek a true ága fut le ugyanis a compare egyezésnél 0-t adna vissza. Aztán mivel a path mappa, rekurzívan bejárjuk, ameniben a file .java kiterjesztésű, beteszi a vectorba, a végén, a vectort bejárjuk, és az eltárolt file neveket kiírjuk + kiírjuk, mennyi JDK osztályt találtunk.

```
#include <iostream>
#include <vector>
#include <boost/filesystem.hpp>
using namespace std;
using namespace boost::filesystem;
int i;
vector<path> v;
void read_acts (path p)
{
 if (is_regular_file(p)) {
 string ext ("." + java);
 if (!ext.compare(extension(p)))
 {
 i++;
 v.push_back(p);
 }
 }
}
```

```
 }

 } else if (is_directory (p))
 for (directory_entry & entry:directory_iterator(p))
 read_acts (entry.path());
 }

int main(int argc, char *argv[])
{
 /* cout << argc << "\n";
 cout << *argv << "\n"; */
 path p("srczip");
 if(!exists(p) || !is_directory(p))
 {
 cout<<p<<"Valami nem oké a könyvtárral, próbáld újra!\n";
 return 1;
 }
 read_acts(p);
 for(auto&& x : v)
 cout << x.filename()<< endl;
 cout << i << " JDK osztályt találtunk.\n";
}
```

A program fordítása a következőképpen történik: **g++ jdk.cpp -o jdk -std=c++11 -lboost\_system -lboost\_filesystem**.

A futtatás követően pedig meg is listázza, valamint számolja, hogy mennyi JDK osztályt találtunk-

The screenshot shows a Linux desktop environment with a terminal window and a code editor window.

The terminal window (top right) displays the command: `g++ jdk.cpp -o jdk -std=c++11 -lboost_system -lboost_filesystem`. Below it, the output of the compilation and execution is shown:

```
okt 7 20:36 ~/Doc gyula@gyula-X550JK: ~/Documents/prog2
"XmlOneWay.java"
"XmlWebMethod.java"
"JavaParam.java"
"XmlFaultAction.java"
"XmlServiceMode.java"
"XmlWebEndpoint.java"
"SoapBindingStyle.java"
"XmlWebFault.java"
"SoapBindingParameterStyle.java"
"XmlWebServiceRef.java"
"JavaMethod.java"
"XmlWebServiceClient.java"
"XmlWebService.java"
"package-info.java"
"ObjectFactory.java"
"XmlRequestWrapper.java"
"XmlResponseWrapper.java"
"XmlAddressing.java"
"XmlSOAPBinding.java"
"XmlWebServiceProvider.java"
"ExistingAnnotationsType.java"
"XmlWebResult.java"
"XmlHandlerChain.java"
"XmlAction.java"
"WebParamMode.java"
"XmlMTOM.java"
"SoapBindingUse.java"
"Util.java"
"XmlBindingType.java"
"XmlWebParam.java"
15761 JDK osztályt találtunk.
gyula@gyula-X550JK: ~/Documents/prog2$
```

The code editor window (bottom left) shows the C++ source code for `jdk.cpp`. The code implements a function `read_acts` that traverses a directory tree and prints out Java class names found. It also counts the number of classes found and prints that count at the end.

## 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!

Túlterheljük az egyenlőség operátort ( $\|$  típus operator op(paraméterlista); a függvény automatikusan meg-hívódik, amikor az operátort egy meghatározott szövegkörnyezetben használjuk). A mozgató értékkadás:

```
BinRandTree & operator = (BinRandTree && old) {
 std::cout << "BT move assign" << std::endl;

 std::swap(old.root, root);
 std::swap(old.treep, treep);

 return *this;
}
```

Ez a binfának az a változata, ahol mutatóként tartalmazza a gyökeret. Ebben a függvényben a két gyökérmutatót a swap függvényt, majd az aktuális objektumot téritjük vissza a this segítségével hivatkozva rá. Mozgató konstruktort, az új objektum megkapja a régi binfa gyökérpointerét, majd a régi objektum pointerét nullpointerre alakítjuk, ezért nevezhetjük mozgatásnak, mert az új binfa megkapja a régi objektum elemeit a jobbértekreferencia segítségével. (jobb/bal érték tisztázása - minden olyan kifejezés, amelynek érték adható, balérték, minden olyan kifejezést, amit értékül adhatunk, jobbértek)

Annak érdekében, hogy a mozgató konstruktor a mozgató értékkadást használja, alkalmaznunk kell a move függvényt:

```
BinRandTree(BinRandTree && old) {
 std::cout << "BT move ctor" << std::endl;

 root = nullptr;
 *this = std::move(old);
```

Itt azt kell elérnünk, az aktuális fa gyökerének kinullázása, move segítségével, ezáltal a mozgató értékkadás érvényesül. Ugyanis ekkor a swappel a mozgató értékkadás megcseréli a két fa gyökérmutatóját, így a az aktuális fa gyökere megkapja a régi fáét, a régi pedig a nullpointerre lesz állítva. Így ez a változata a mozgató konstruktornak ugyanazt az eredményt éri el, mint az eredeti változat.

Mozgató szemantika, move függvénnyel, az argumentumként megadott értékből jobbértekreferenciát készít, ami megfelelő paraméter a mozgató konstruktornak.

## 16.3. Hibásan implementált RSA törése

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

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Stroustrup/RSA.java](bhax/thematic_tutorials/bhax_textbook_sajat/Stroustrup/RSA.java) [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Stroustrup/RSApelda.java](bhax/thematic_tutorials/bhax_textbook_sajat/Stroustrup/RSApelda.java)

Az RSA-eljárás nyílt kulcsú (vagyis „aszimmetrikus”) titkosító algoritmus, melyet 1976-ban Ron Rivest, Adi Shamir és Len Adleman fejlesztett ki (és az elnevezést nevük kezdőbetűiből kapta). Ez napjaink egyik leggyakrabban használt titkosítási eljárása.<sup>[1]</sup> Az eljárás elméleti alapjait a moduláris számelmélet és a prímszámelmélet egyes tételei jelentik.

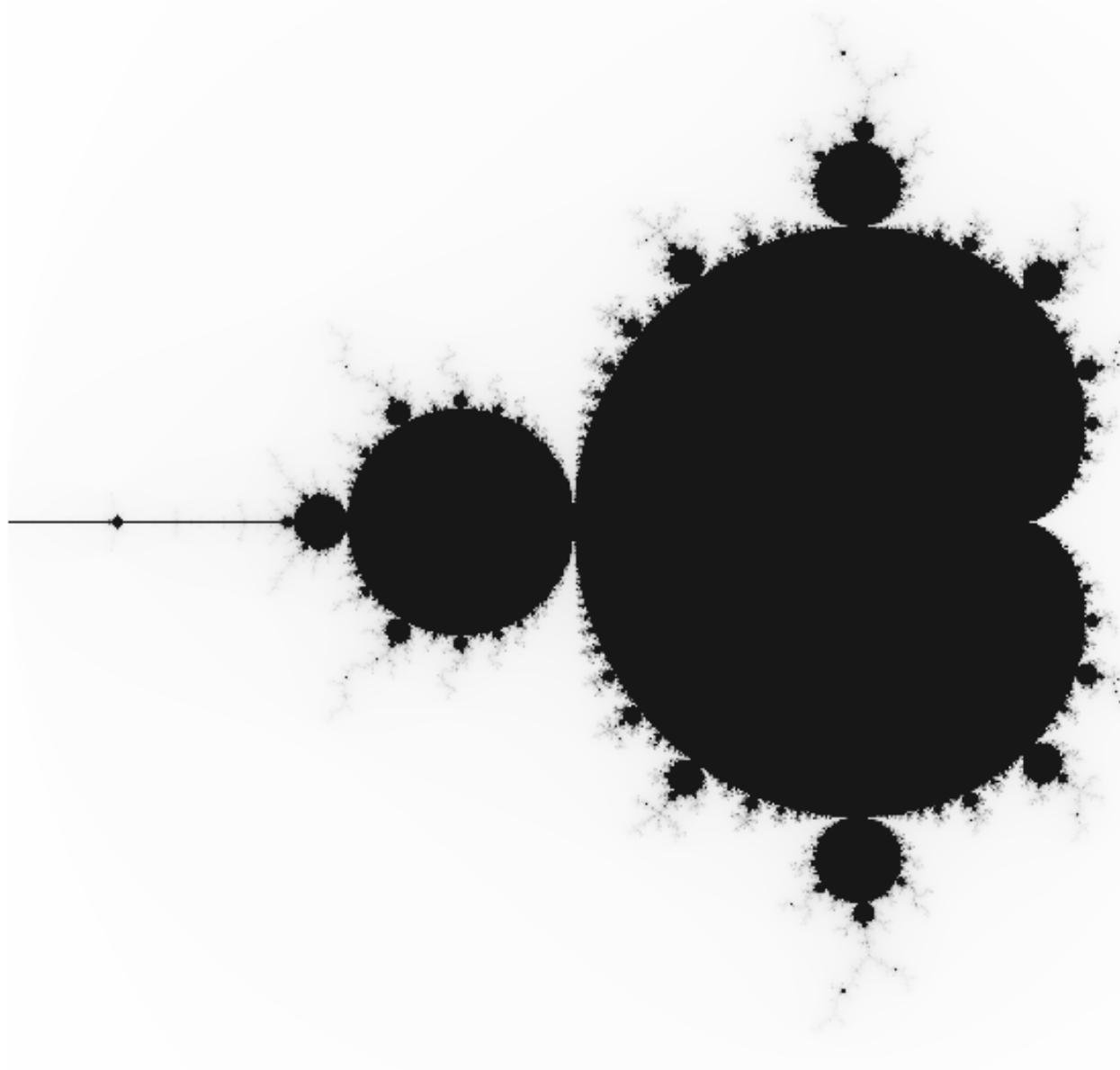
—Wikipedia

## 16.4. Változó argumentumszámú címke

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: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Stroustrup/preceptor.cpp](https://bhax/thematic_tutorials/bhax_textbook_sajat/Stroustrup/preceptor.cpp)

Először is a Benoît Mandelbrot nevéhez fűződő kétdimenziós halmaz png ábráját fogjuk létrehozni a mandel.cpp segítségével.



A perceptron az egyik legegyszerűbb előrecsatolt neurális hálózat. A `perceptron.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 filéra, mely már tartalmazza a Perceptron osztályt.

Az előző program futtatásával 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. A visszakapott értékeket megfeleltetjük a blue értékeknek.

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

```
#include <fstream>

int main (int argc, char **argv)
{
 png::image<png::rgb_pixel> png_image (argv[1]);
 int size = png_image.get_width() *png_image.get_height();

 Perceptron* p = new Perceptron (3, size, 256, size);

 double* image = new double[size];

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

 double* newimage = (*p) (image);

 for (int i = 0; i<png_image.get_width(); ++i)
 for (int j = 0; j<png_image.get_height(); ++j)
 png_image[i][j].blue = newimage[i*png_image.get_width() +j];

 png_image.write("output.png");

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

Az első felvonás Perceptron feladatához képest módosításokat kell végezniünk 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 képbe. Az utolsó units tömb értékei átkerülnek a paraméterként kapott tömbbe, az eredeti és az új kép egyforma méretű lesz.

```
double* operator() (double image [])
{
 units[0] = image;

 for (int i {1}; i < n_layers; ++i)
 {

#ifndef CUDA_PRCPS

 cuda_layer (i, n_units, units, weights);

#else

 #pragma omp parallel for
 for (int j = 0; j < n_units[i]; ++j)
 {
 units[i][j] = 0.0;
 }
 }
}
```

```
for (int k = 0; k < n_units[i-1]; ++k)
{
 units[i][j] += weights[i-1][j][k] * units[i-1][k];
}

units[i][j] = sigmoid (units[i][j]);

}

#endif

}

for (int i = 0; i < n_units[n_layers - 1]; i++)
image[i] = units[n_layers - 1][i];

return image;

}
```

A fordításhoz a C++14 szabványt használjuk, futtatáskor pedig megadjuk annak a képfájlnak a nevét és kiterjesztését, amelyet be kívánunk olvasni (a mandel.cpp-ből kapott mandel.png ábra).

The screenshot shows a Linux desktop environment with several windows open. On the left, there's a dock with icons for various applications like a file manager, terminal, and browser. In the center, there's a terminal window titled 'Sublime Text (INHERITED)' showing command-line output:

```
okt 8 17:39
~/Documents/prog2/perceptron.cpp ~ Sublime Text (INHERITED)
gyula@gyula-X550JK:~/Documents/prog2$ g++ mlp.hpp perceptron.cpp -o perc -lpng -std=c++14
gyula@gyula-X550JK:~/Documents/prog2$./perc mandel.png
mandel.png->output.png mentve
gyula@gyula-X550JK:~/Documents/prog2$
```

Below the terminal, there are two image viewer windows. The left one is labeled 'mandel.png' and shows the classic black-and-white fractal pattern of the Mandelbrot set. The right one is labeled 'output.png' and shows the same fractal pattern but with a solid yellow background. This demonstrates how the C++ program processes the image.

# 17. fejezet

## Helló, Gödel!

### 17.1. Gengszterek

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Gödel/myshmclient.cpp](https://bhax/thematic_tutorials/bhax_textbook_sajat/Gödel/myshmclient.cpp)

Az OOCWC (Robocar World Championship) 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 alakja:

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

A szögletes zárójelpár jelzi, hogy lambda kifejezés következik. A kerek zárójelpár a függvényhívást jelenti. Ezután 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. STL map érték szerinti rendezése

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

A **FUTURE projekt** fenykard.cpp fájljában a sort függvényt használjuk a map érték szerinti rendezésére.

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). 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;
}
```

### 17.3. Alternatív Tabella rendezése

Mutassuk be a [https://progpater.blog.hu/2011/03/11/alternativ\\_tabella](https://progpater.blog.hu/2011/03/11/alternativ_tabella) a programban a java.lang Interface Comparable T szerepét!

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Gödel/AltTabella.cpp](bhax/thematic_tutorials/bhax_textbook_sajat/Gödel/AltTabella.cpp) [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Gödel/Wiki2Matrix.cpp](bhax/thematic_tutorials/bhax_textbook_sajat/Gödel/Wiki2Matrix.cpp)

Az alternatív tabella a Google algoritmusán, a PageRank-en alapszik. A PageRank alapötlete, 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, 2, 2, 3},
 {3, 0, 2, 1, 3, 2, 0, 3, 3, 3, 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, 1, 1, 0, 1, 1, 0, 2, 1, 1},
```

```

{3, 3, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0},
{1, 0, 3, 1, 0, 0, 0, 1, 3, 2, 0, 0, 0, 1, 2, 3},
{0, 2, 0, 0, 0, 1, 0, 1, 1, 0, 3, 0, 1, 3, 3, 1},
{0, 0, 1, 1, 3, 0, 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, 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, 0, 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 eredetinev tömbbe kerülnek a csapatok nevei a táblázatban való megjelenésük alapján. Az eredetipont tömbbe az eredeti tabella pontjait írjuk be. Készítünk egy ujnev 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 (attól függően, hogy a jelenlegi objektum nagyobb vagy kisebb-e) vagy nullát (egyenlőség esetén).

```

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;
 }
 }
}

```

```

norma = 0.007535279795238463
oszeg =1.0
Iteracio...
norma = 0.007535279795238463
oszeg =1.0
*** 0.0698112943387984
0.0698112943387984
0.0540414942606666
0.0540414942606666
0.0231477718117064
0.06892915574158383
0.06496516770876724
0.05702294374983677
0.04393368667699879
0.05608830969430056
0.05958830154689826
0.07251615915920115
0.03295035734097101
0.05608830969430056
0.0636413924822027
0.0941168277078394
0.06360412832329729
*** 1 Vldeoton 0.08411682770703394
1 Vldeoton 40.0
2 Debreceni VSC 0.08281477718117394
2 Ferencvaros 34.0
3 Paks FC 0.07251615915920115
3 Paks 31.0
4 B Stiropak 0.0698112943387984
4 Debreceni VSC 31.0
5 Budapest Honvid 0.06975083672427473
5 Zalaegerszegi TE 30.0
6 Ferencvaros 0.06892915574158383
6 Kapovari Rakoczi 29.0
7 Gyuri ETO 0.06496516770876724
7 Lombard Popa 27.0
8 Ijpest 0.06364130248286927
8 Kecskemeti TE 24.0
9 Zalaegerszegi TE 0.06360412832329729
9 Ijpest 23.0
10 MTK Budapest 0.0595830154689826
10 Gyuri ETO 23.0
11 Kapovari Rakoczi 0.05702294374983677
11 Budapest Honvid 22.0
12 Lombard Popa 0.05608830969430056
12 MTK Budapest 22.0
13 Szombathelyi Haladás 0.05594660883459867
13 Vasas 21.0
14 Vasas 0.05434414942609066
14 Szombathelyi Haladás 20.0
15 BFC Siófok 18.0
16 Szolnoki MAV FC 0.03295635734097101
16 Szolnoki MAV 9.0
gyula@gyula-X550JK:~/Documents/prog2$ []

```

## 17.4. C++11 Custom Allocator

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

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Gödel/customalloc.cpp](bhax/thematic_tutorials/bhax_textbook_sajat/Gödel/customalloc.cpp)

Első sorban #include-oljuk a megfelelő header-öket.

```
#include <iostream>
#include <string>
#include <vector>
```

Ezután létrehozunk egy template osztályt és index-eljük a következő dolgokat.

```
template<typename T>
class CustomAlloc{

public:
 using size_type = size_t;
 using pointer = T*;
 using const_pointer = const T*;
 using value_type = T;
```

Lefoglalunk n-darabot a size\_type-nak. Ezután kiiratjuk, hogy foglalunk n darab objektumot, valamint hogy mennyi byte-ot az n\*sizeof(value\_type)-nak. Végül pedig visszaadjuk a lefoglalt memóriát, char-al foglalok n\*sizeof(value\_type) byte-ot

```
pointer allocate(size_type n){

 std::cout <<"Allocating " << n << " object of " << n*sizeof(←
 value_type) << std::endl;

 return reinterpret_cast<pointer>(new char [n*sizeof(value_type)]);
}
```

Elsősorban megadjuk neki a pointert és hogy mennyit szabadítson fel. Mivel tömbösen foglaltam, ezért tömbösen is törlöm a p pointert.

```
void deallocate(pointer p, size_type n){

 std::cout <<"Deallocating " << n << " object of " << n*sizeof(←
 value_type) << std::endl;

 delete [] reinterpret_cast<char *>(p);
}
```

Ezután létrehozzuk a main függvényünket. Inteket rakunk bele a vektorunkba, megadjuk neki, hogy használja ami CustomAlloc-unkat. Végül pedig v-be bele push\_back-elem a számunkat.

```
int main(){

 std::string s;
 std::allocator<int> a;

 std::vector<int, CustomAlloc<int>> v;

 v.push_back(42);
 v.push_back(42);
```

```
v.push_back(46);

return 0;
}
```

A program fordítását és futtatását követően látjuk hogy alloce-álja elsőnek a 4-et, majd a 8-at. A program végén pedig dealloce-álja a 4-et, majd a 8-at.

The screenshot shows the KDevelop IDE interface. On the left is the code editor with 'customalloc.cpp' open, containing the provided C++ code. On the right is a terminal window showing the execution of the program:

```
gyula@gyula-X550JK:~/Documents/prog$ g++ customalloc.cpp -o custom
gyula@gyula-X550JK:~/Documents/prog$./custom
Allocating 1 object of 4
Allocating 2 object of 8
Deallocating 1 object of 4
Allocating 4 object of 16
Deallocating 2 object of 8
Deallocating 4 object of 16
gyula@gyula-X550JK:~/Documents/prog$
```

## 18. fejezet

# Helló, !

### 18.1. FUTURE tevékenység editor

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/hello/ActivityEditor.java](https://github.com/nbatfai/future.git)

A [FUTURE projekt](#) eredeti célja az egyes játékosok és ezáltal egy város alternatív jövőinek legenerálása, a legenerált jövők elemzése volt. A FUTURE6-hoz készült egy tevékenység/tulajdonság editor, melyben a hallgatók az adott nap tevékenységeit tudták összegezni, feljegyezni.

Az ActivityEditor futtatásához a következő lépéseket kell megtennünk a terminálban:

```
$ git clone https://github.com/nbatfai/future.git
$ cd future/cs/F6
$ javac ActivityEditor.java
$ java ActivityEditor --city=Debrecen --props=me.props,gaming.props, ↵
 programming.props
```

Az editor külső megjelenése könnyen feldobható egy CSS stíluslap készítésével. Ahhoz, hogy a stíluslapban található 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 véleményem szerint jól passzol az, ha a tevékenységek és tulajdonságok szövege erős, sötétkék színt kap. Ha rákattintunk egy dobozra, akkor annak szegélye is sötétkékké válik, de ennek hex kódja némiképp különbözik a korábban említett text fill értéktől. A címek kapnak egy halványszürke háttérrel, így egy kicsit jobban kiemelkednek a címekben olvasható szövegek.

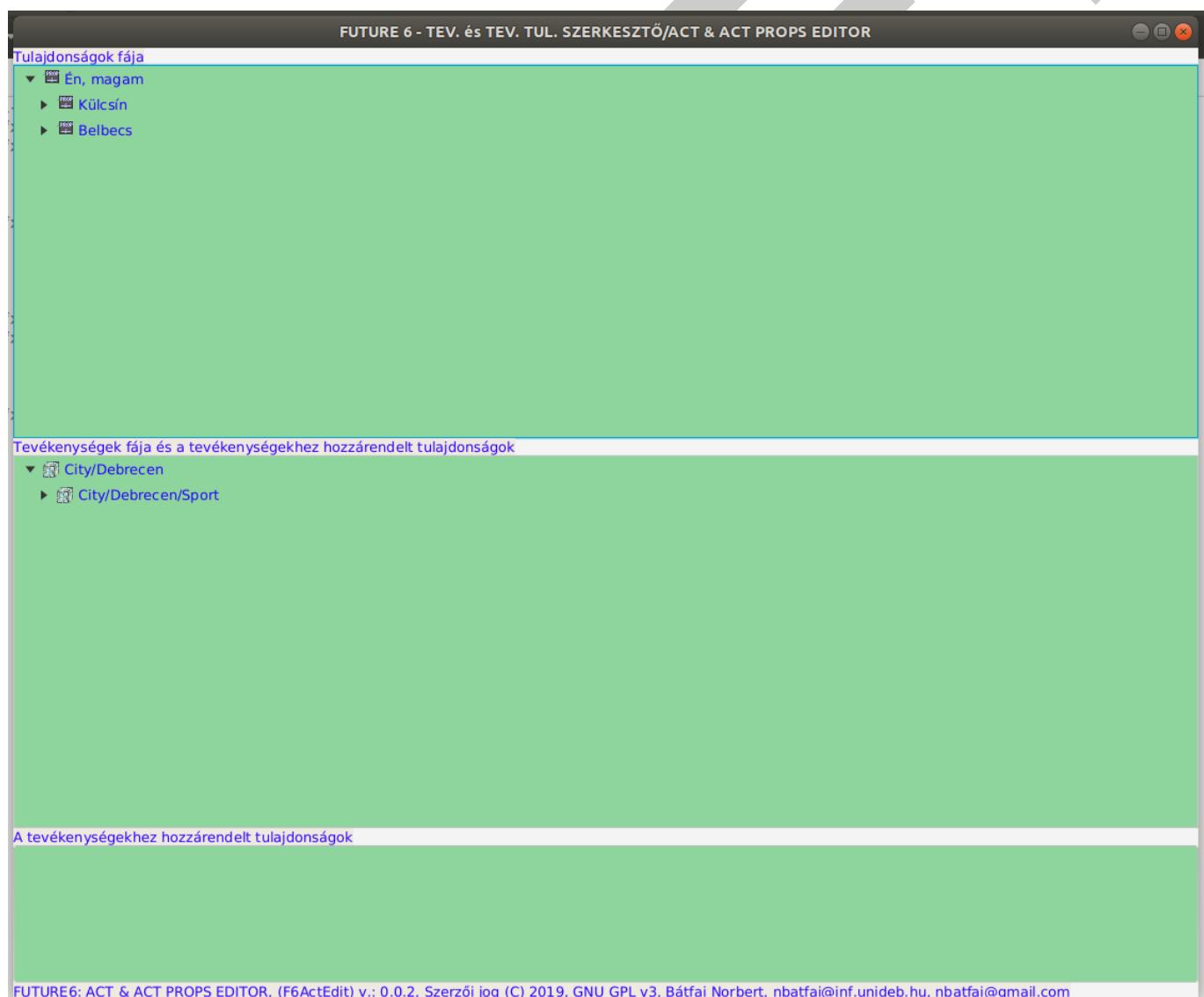
```
.tree-cell{
 -fx-background-color: #8ed49d;
```

```
-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/robocar-emulator/blob/master/-justine/rcemu/src/carlexer.ll>

Ez a feladat egy inputstream adatainak feldolgozását mutatja be, ezáltal megismerkedhetünk a streamekkel.

```
while (std::sscanf (data+nn, "<OK %d %u %u %u>%n", &idd, &f, &t, ←
 &s, &n) == 4)
{
 nn += n;
 gangsters.push_back (Gangster { idd, f, t, s });
}
```

Az sscanf formázott stringből olvas be, itt azért van szükség erre a whilera mert nem tudjuk pontosan hány gengster adatot tudunk beolvasni, ezért ellenőrini kell az adatok érvényességét. A whileban a sscanf visszatérési értékét (beolvasott paraméterek száma), és a négyet hasonlítjuk össze, azaz ha ez az egyenlőség igaz, minden gangster adatát sikeresen beolvastuk. Létrehozunk egy új gangster objektumot, amit bele rakunk a gangster vectorba, az nn változó az eddig beolvasott adatok számát tárolja, erre a data karakter tömb miatt van szükség, hogy ne olvassuk be belölle kétszer ugyan azt az adatot.

## 18.3. BrainB

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

A BrainBvel már foglalkoztunk, ez egy olyan program ami azt szimulálja mikor egy játék közben nagy a katyvasz, és szinte eltűnik a karakterünk, a szimuláció 10 percnyi időn belül teszteli mennyire tudjuk követni a Samu Entropy négyzetet, a futás végén kapunk egy statisztikát.

Használata:

```
sudo apt-get install libqt4-dev
sudo apt-get install opencv-data
sudo apt-get install libopencv-dev

qmake BrainB.pro
make
./BrainB
```

Most tekintsük meg a slot-signal mechanizmust, Qt-ban a slot-signal mechanizmust, objektumok közötti kommunikációra lehet használni. Egy slot akkor hívódik meg amikor egy signal bekövetkezik, a signal pedig akkor következik be, ha egy bizonyos esemény bekövetkezik, ezzel elkerülődik a cross reference, mivel nem kell egy osztályon belül egy másik osztályt meghívni. A slotok a signalokhoz, connect függvényel kapcsolódnak (szignaturáiknak meg kell egyezniük), itt két connectet figyelhetünk meg.

```
connect (brainBThread, SIGNAL (heroesChanged (QImage, int, int)),
```

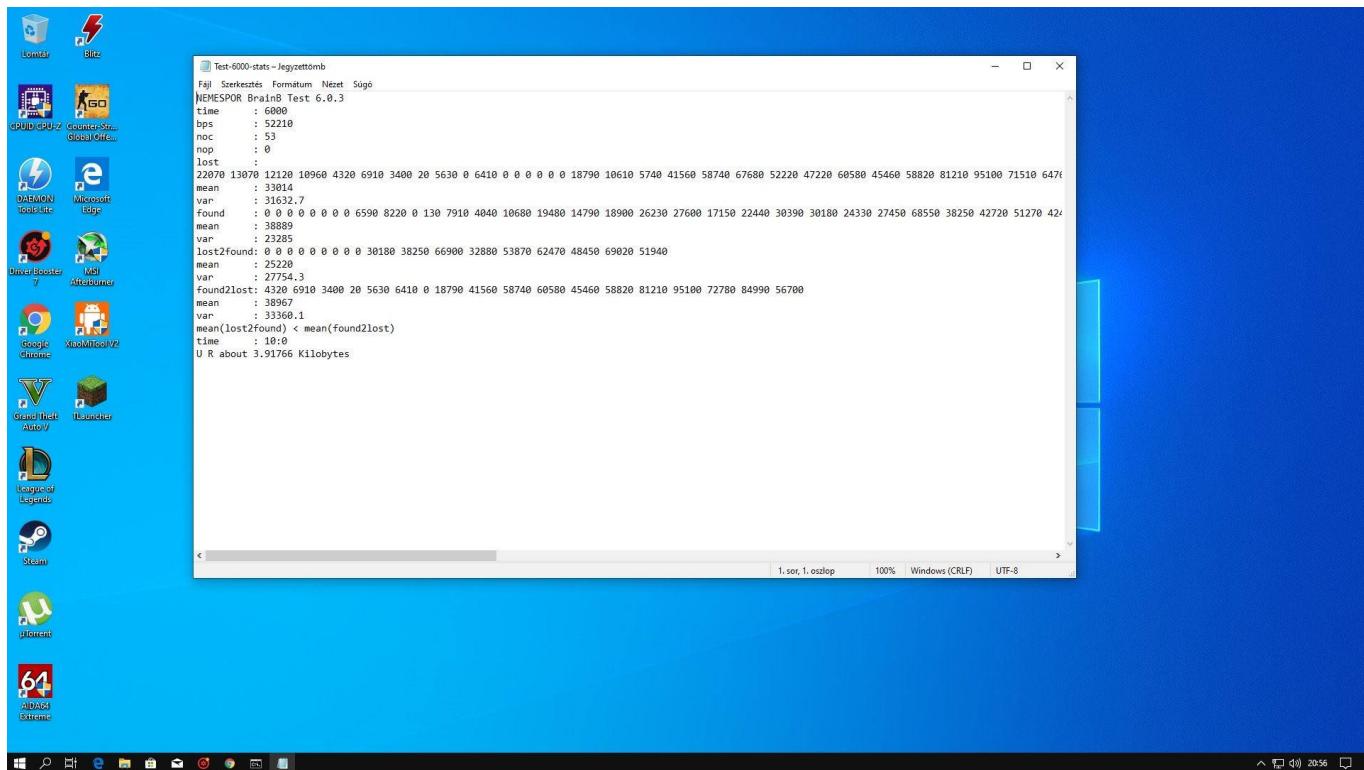
```
this, SLOT (updateHeroes (QImage, int, int)));
connect (brainBThread, SIGNAL (endAndStats (int)),
this, SLOT (endAndStats (int)));
```

Azt jelenti, hogya brainBThread-ben a heroesChanged signal bekövetkezik akkor lefut a BrainBWin updateHeroes függvénye, endAndStats esetén ugyanígy, ami a futási idő lejártakor következik be, aminek köszönhetően, kiíródik a debug üzenet, hogy vége a játéknak:

```
void BrainBThread::run()
{
 while (time < endTime) {
 QThread::msleep (delay);
 if (!paused) {
 ++time;
 devel();
 }
 draw();
 }
 emit endAndStats (endTime);
}
```

```
void BrainBWin::endAndStats (const int &t)
{
 qDebug() << "\n\n\n";
 qDebug() << "Thank you for using " + appName;
 qDebug() << "The result can be found in the directory " + ←
 statDir;
 qDebug() << "\n\n\n";

 save (t);
 close();
}
```

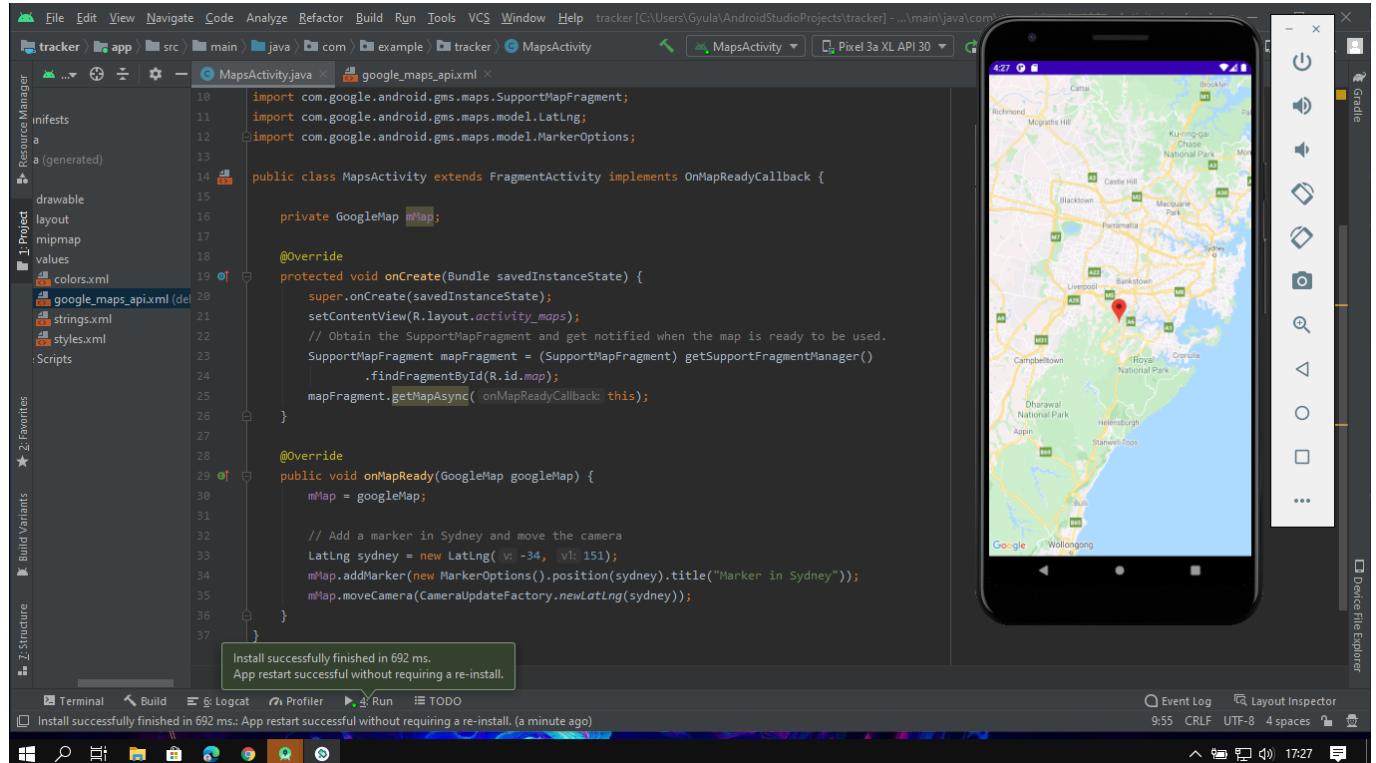


## 18.4. Android "GPS tracker"

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/hello/MapsActivity.java](https://bhax/thematic_tutorials/bhax_textbook_sajat/hello/MapsActivity.java)

Forrás: [ProgrammerWorld](https://ProgrammerWorld)

Az OSM térképre rajzolása helyett én a Google Maps alapú Android "GPS tracker" elkészítését választottam. Amennyiben Android Studio-val dolgozunk, érdemes Google Maps Activity projektet készítenünk, így azonnal kapunk egy működőképes, használható térképet.



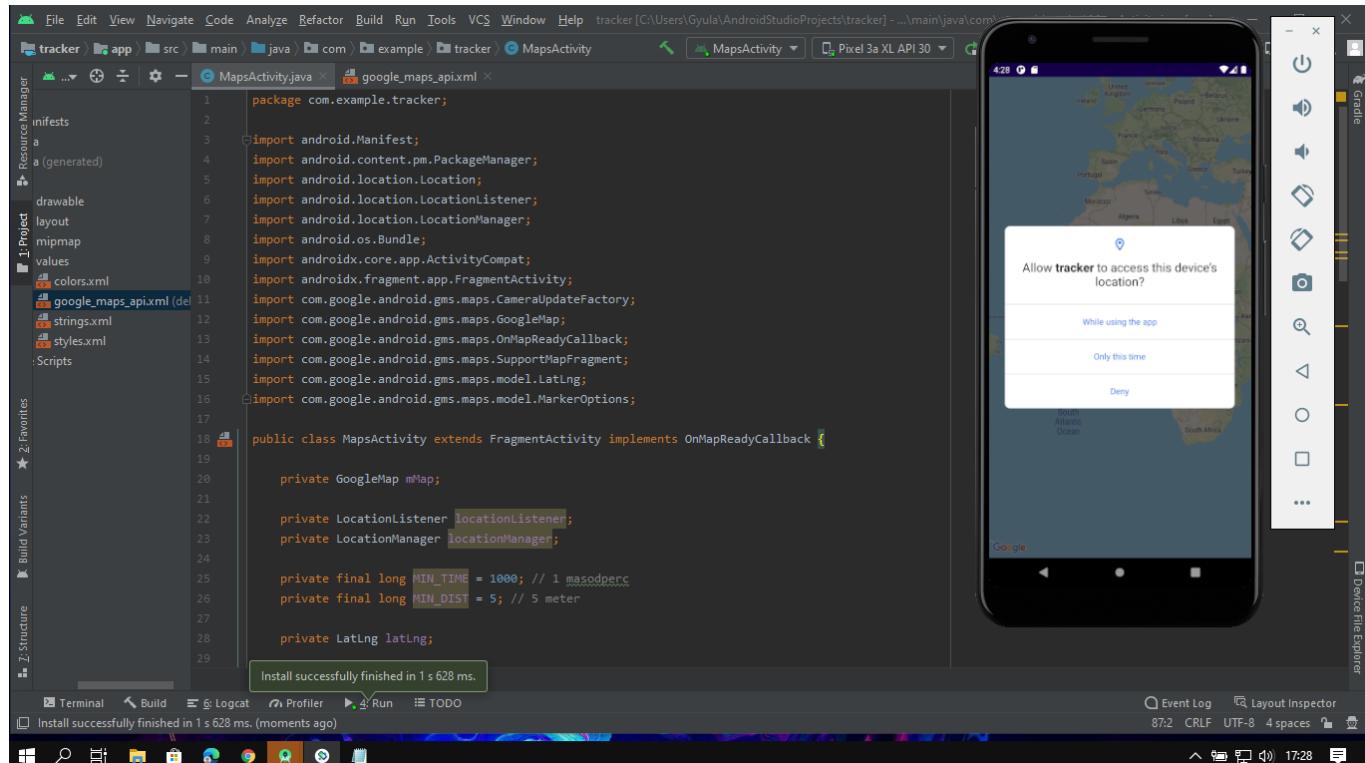
Ahoz, hogy tudjunk a Google Maps API-val dolgozni, kénytelenek leszünk a `google_maps_api.xml`-ben található linkre ellátogatva egy API kulcsot generálni. Amennyiben ez megvan, másoljuk be a következő sorba:

```
<string name="google_maps_key" templateMergeStrategy="preserve" ↩
 translatable="false">YOUR_KEY_HERE</string>
```

Ha a Google Play szolgáltatások megfelelően vannak telepítve készüléinkre, akkor az app futtatása sikeres lesz. A `MapsActivity.java` fájlban láthatjuk, hogy jelenleg a térképünkre még csak egy darab marker van, amely Sydneyre mutat. A térkép bejárásától, illetve zoomolástól függetlenül is megmarad a meghatározott helyzetben a marker.

```
@Override
public void onMapReady(GoogleMap googleMap) {
 mMap = googleMap;

 LatLng sydney = new LatLng(-34, 151);
 mMap.addMarker(new MarkerOptions().position(sydney).title("Marker ↩
 in Sydney"));
 mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
}
```



Ebből kiindulva alakítsuk át az appot úgy, hogy felismerje a tartózkodási helyünket, bizonyos időközönként frissítse azt, és jelezze újabb markerek elhelyezésével!

A frissítések közötti minimális időköz nálam 1 másodperc, míg a minimális távolság 5 méter.

```
private final long MIN_TIME = 1000;
private final long MIN_DIST = 5;
```

Ahhoz, hogy ne csak a .java fájlban megadott, fix szélességi/hosszúsági fokok alapján tudjunk markereket készíteni, engedélyt kell kérnünk a felhasználótól.

```
ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.ACCESS_FINE_LOCATION}, PackageManager.PERMISSION_GRANTED);
ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.ACCESS_COARSE_LOCATION}, PackageManager.PERMISSION_GRANTED);
```

A tartózkodási helyünkből kapott szélességi és hosszúsági fokot kihasználva hozunk létre új markereket, a korábban meghatározott frissítési gyakoriság alapján.

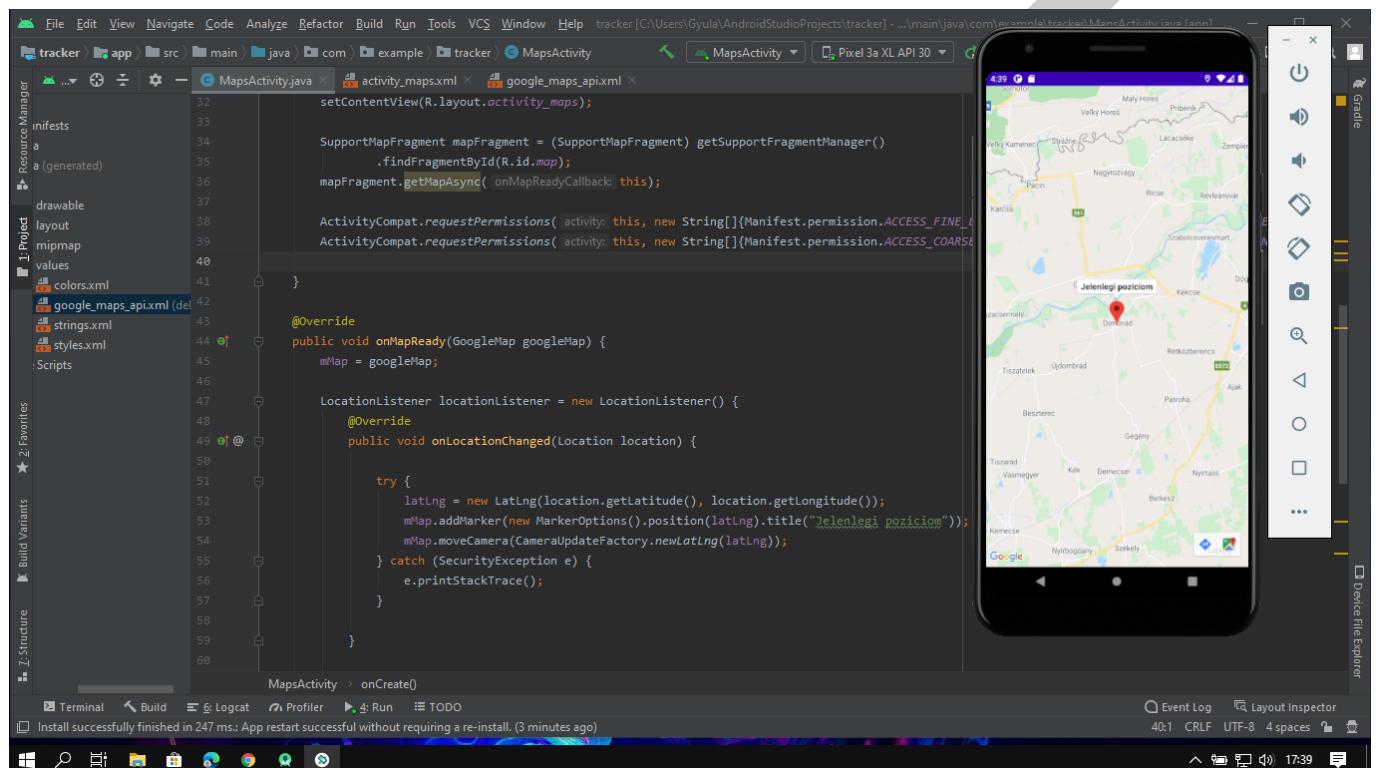
```
@Override
public void onLocationChanged(Location location) {

 try {
 latLng = new LatLng(location.getLatitude(), location.getLongitude());
 mMap.addMarker(new MarkerOptions().position(latLng).title("Jelenlegi poziciom"));
 mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
 }
 catch (SecurityException e) {
```

```
 e.printStackTrace();
 }

}
```

Amennyiben feltelepítettük a megfelelő **OEM USB Drivet** a telefonunkra, csatlakoztatás esetén tudjuk is futtatni készülékünkön a GPS trackert. Kis várakozást követően sikeresen felismernie tartózkodási helyemet:



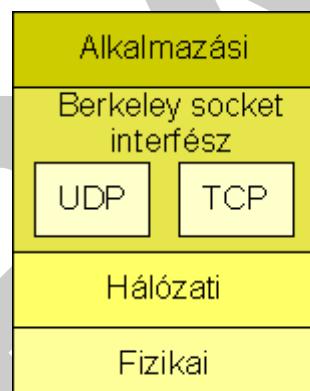
## 19. fejezet

# Helló, Lauda !

### 19.1. Port scan

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

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Lauda/KapuSzkenner.java](bhax/thematic_tutorials/bhax_textbook_sajat/Lauda/KapuSzkenner.java)



```
public class KapuSzkenner {

 public static void main(String[] args) {

 for(int i=0; i<1024; ++i)

 try {

 java.net.Socket socket = new java.net.Socket(args[0], i);

 System.out.println(i + " figyeli");

 socket.close();

 } catch (Exception e) {
 }
```

```
 System.out.println(i + " nem figyeli");

 }
}

}
```

A program végigzongorázza a parancssorában megkapott nevű gép 1024 alatti számú TCP kapuit: megpróbál egy TCP kapcsolatot

```
java.net.Socket socket = new java.net.Socket(args[0], i);
```

létrehozni, ha sikerül, akkor a célporton ül egy szerver folyamat, ha nem, azaz ha kivétel keletkezik, akkor nem. Egyébként siker esetén sem csinálunk semmit, hanem csak bezárjuk az éppen elkészített kliensoldali kommunikációs végpontot reprezentáló socket objektumot.

The screenshot shows a Linux desktop environment with a dark theme. On the left, there's a dock with icons for various applications like a browser, file manager, and system tools. In the center, there's a terminal window titled 'Terminal' with the command 'java KapuSzkenner | more'. The output of the program is visible in the terminal, showing a series of lines from 0 to 26, each containing the text 'nem figyeli'. To the right of the terminal is a Sublime Text editor window with an open Java file named 'KapuSzkenner.java'. The code in the editor matches the one shown in the terminal, with a 'try' block attempting to create a socket and print its index, followed by a 'catch' block for exceptions and a closing brace. The status bar at the bottom of the Sublime window shows 'Line 23, Column 25' and other file details.

## 19.2. AOP

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Lauda/HellóVilág.java](https://bhax/thematic_tutorials/bhax_textbook_sajat/Lauda/HellóVilág.java)

Forrás: [Prognyelvek portál](#)

Az AOP (aspektusorientált programozás) az AspectJ nyelv segítségével valósul meg. Ez a programozási nyelv lehetővé teszi már megírt Java kódok használatát, egységesítését.

Az AspectJ vezette be a join point (csatlakozási pont) fogalmát is, mely a kereszthivatkozások helyeinek definiálásáért felelős. A pontok segítségével meg tudjuk tehát határozni, hogy mely sorok nincsenek fiz-

kailag elszeparálva saját modulba. Dinamikus join pointok lehetnek pl. metódus vagy konstruktor hívások, osztály vagy objektum inicializálása.

A pointcut (vágási pont) a csatlakozási pontok és a hozzájuk tartozó értékek kiválogatásáért felel. Vágási pontból operátorok segítségével (pl. `||`) tudunk újakat is létrehozni.

Az egyes vágási pontokhoz kódot is rendelhetünk adviceok (tanácsok) által. AspectJ-ben a before advice a csatlakozási pont végrehajtása előtt fut le, az after advice pedig a csatlakozási pont után. Az around advice akkor hajtódik végre, ha a join pointot eléri a végrehajtás.

Hozzunk létre egy `helló` függvényt, melyben a köszönő line előtt szerepel egy alma, utána pedig egy körte string. Ha ezt futtatjuk, akkor minden sor kiiratásra kerül.

```
public class HellóVilág {
 public void helló() {
 System.out.println("alma");
 System.out.println("HelloVilág> Hello!");
 System.out.println("korte");
 }

 public static void main(String[] args) {
 new HellóVilág().helló();
 }
}
```

Biztonság kedvéért futassuk le:

```
$ javac HellóVilág.java
$ java HellóVilág
alma
HelloVilág> Hello!
korte
```

A csatlakozási pontunk tehát az eredeti Java kódban már szerepel. A vágási pontunk az AspectJ kódban lesz, ez fogja meghívni a `helló` függvényünket. A before advice által az alma, az after advice pedig a körte kerülhet kiiratásra.

```
public aspect ElótteUtána {
 public pointcut fgvHívás(): call(public void HellóVilág.helló());

 before(): fgvHívás() {
 System.out.println("ElötteUtána> Alma");
 }

 after(): fgvHívás() {
 System.out.println("ElötteUtána> Korte");
 }
}
```

Fordítás és futtatás után:

```
$ ajc HellóVilág.java ElótteUtána.aj
$ java HellóVilág
```

```
ElotteUtana> Alma
HelloVilag> Hello!
ElotteUtana> Korte
```

Az eredeti Java kódban található csatlakozási pontok sor szerinti elhelyezkedését is megkaphatjuk a getSourceLocation() metódus segítségével. A getKind()-nak köszönhetően a join point fajtáját is kiirathatjuk (pl. metódushívás). A getSignature() pedig visszaadja a sorban található értéket.

```
public aspect Nagytesó {
 before(): call(* *(..)) && !cflow(adviceexecution()) {
 System.out.println("Nagyteso> " + thisJoinPointStaticPart.←
 getSourceLocation());
 System.out.println("Nagyteso> " + thisJoinPointStaticPart.getKind() ←
);
 System.out.println("Nagyteso> " + thisJoinPointStaticPart.←
 getSignature());
 }
}
```

A HellóVilág.java alapján tehát a következőket kapjuk:

```
$ ajc HellóVilág.java Nagytesó.aj
$ java HellóVilág
Nagyteso> HellóVilág.java:13
Nagyteso> method-call
Nagyteso> void HellóVilág.helló()
Nagyteso> HellóVilág.java:5
Nagyteso> method-call
Nagyteso> void java.io.PrintStream.println(String)
alma
Nagyteso> HellóVilág.java:7
Nagyteso> method-call
Nagyteso> void java.io.PrintStream.println(String)
HelloVilag> Hello!
Nagyteso> HellóVilág.java:8
Nagyteso> method-call
Nagyteso> void java.io.PrintStream.println(String)
korte
```

### 19.3. JUnit teszt

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_sajat/Lauda/BinfaTest.java](https://bhax/thematic_tutorials/bhax_textbook_sajat/Lauda/BinfaTest.java)

Forrás: [Programozási technológiák](#)

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;

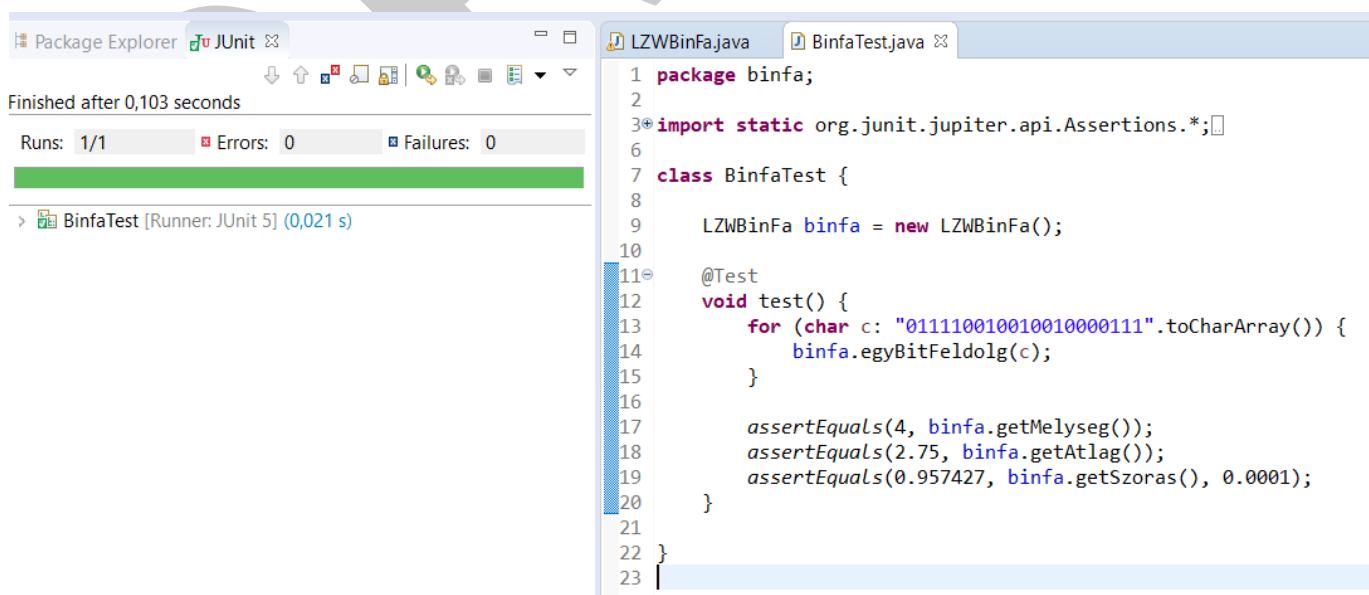
class BinfaTest {

 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:



## IV. rész

### Irodalomjegyzék

DRAFT

## 19.4. Általános

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

## 19.5. C

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

## 19.6. C++

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

## 19.7. Lisp

[METAMATH] Chaitin Gregory, *META MATH! The Quest for Omega*, [http://arxiv.org/PS\\_cache/math/pdf/0404/0404335v7.pdf](http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf) , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPORG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (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.