

Univerzális programozás

Így neveld a programozód!

Ed. BHAX, DEBRECEN,
2020. március 22, v. 0.0.7

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, 2020, 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>

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Ács Bátfai, Margaréta	2020. október 12.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2020-03-02	Az Chomsky/ $a^n b^n c^n$ és Caesar/EXOR csokor feladatok kiírásának aktualizálása (a heti előadás és laborgyakorlatok támogatására).	nbatfai

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.6	2020-03-21	A MALMÖ projekt feladatok átvezetése, minden csokor utolsó feladata Minecraft ágensprogramozás ezzel. Mottók aktualizálása. Prog1 feladatok aktualizálása. Javasolt (soft skill) filmek, elméletek, könyvek, előadások be.	nbatfai
0.0.7	2020-03-22	Javítások.	nbatfai

Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it.”

—Gregory Chaitin, *META MATH! The Quest for Omega*, [METAMATH]

„Csak kicsi hatást ért el a videójáték-ellenes kampány. A legtöbb iskolában kétműszakos üzemen dolgoznak a számítógépek, értő és áldozatos tanárok ellenőrzése mellett.”

„Minden számítógép-pedagógus tudja a világon, hogy játékokkal kell kezdeni. A játékot követi a játékprogramok írása, majd a tananyag egyes részeinek a feldolgozása.,,

—Marx György, *Magyar Tudomány*, 1987 (27) 12., [MARX]

„I can't give complete instructions on how to learn to program here — it's a complex skill. But I can tell you that books and courses won't do it — many, maybe most of the best hackers are self-taught. You can learn language features — bits of knowledge — from books, but the mind-set that makes that knowledge into living skill can be learned only by practice and apprenticeship. What will do it is (a) reading code and (b) writing code.”

—Eric S. Raymond, *How To Become A Hacker*, 2001.,
<http://www.catb.org/~esr/faqs/hacker-howto.html>

„I'm going to work on artificial general intelligence (AGI).”

I think it is possible, enormously valuable, and that I have a non-negligible chance of making a difference there, so by a Pascal's Mugging sort of logic, I should be working on it.

For the time being at least, I am going to be going about it “Victorian Gentleman Scientist” style, pursuing my inquiries from home, and drafting my son into the work.”

—John Carmack, *Facebook post*, 2019., [in his private Facebook post](#)

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?	8
2.3. Változók értékének felcserélése	10
2.4. Labdapattogás	11
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	14
2.6. Helló, Google!	16
2.7. A Monty Hall probléma	20
3. Helló, Chomsky!	22
3.1. Decimálisból unárisba átváltó Turing gép	22
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	23
3.3. Hivatkozási nyelv	23
3.4. Saját lexikális elemző	24
3.5. l33t.1	25
3.6. A források olvasása	25
3.7. Logikus	27
3.8. Deklaráció	28

4. Helló, Caesar!	30
4.1. double ** háromszögmátrix	30
4.2. C EXOR titkosító	33
4.3. Java EXOR titkosító	34
4.4. C EXOR törő	35
4.5. Neurális OR, AND és EXOR kapu	38
4.6. Hiba-visszaterjesztéses perceptron	41
4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve	42
5. Helló, Mandelbrot!	43
5.1. A Mandelbrot halmaz	43
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	46
5.3. Biomorfok	48
5.4. A Mandelbrot halmaz CUDA megvalósítása	52
5.5. Mandelbrot nagyító és utazó C++ nyelven	53
5.6. Mandelbrot nagyító és utazó Java nyelven	59
5.7. Vörös Pipacs Pokol/fel a láváig és vissza	60
6. Helló, Welch!	61
6.1. Első osztályom	61
6.2. LZW	63
6.3. Fabejárás	64
6.4. Tag a gyökér	65
6.5. Mutató a gyökér	65
6.6. Mozgató szemantika	65
6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid	66
7. Helló, Conway!	67
7.1. Hangyaszimulációk	67
7.2. Java életjáték	70
7.3. Qt C++ életjáték	73
7.4. BrainB Benchmark	75
7.5. Vörös Pipacs Pokol/19 RF	75

8. Helló, Schwarzenegger!	76
8.1. Szoftmax Py MNIST	76
8.2. Mély MNIST	77
8.3. Minecraft-MALMÖ	78
9. Helló, Chaitin!	81
9.1. Iteratív és rekurzív faktoriális Lisp-ben	81
9.2. Gimp Scheme Script-fu: króm effekt	81
9.3. Gimp Scheme Script-fu: név mandala	82
9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-edén	83
10. Helló, Gutenberg!	84
10.1. Programozási alapfogalmak	84
10.2. C programozás bevezetés	85
10.3. C++ programozás	86
10.4. Python nyelvi bevezetés	87
III. Második felvonás	88
11. Helló, Berners-Lee!	90
11.1. Olvasónapló:	90
12. Helló, Arroway!	91
12.1. OO szemlélet	91
12.2. Homokozó	91
12.3. Gagy	92
12.4. Yoda	92
13. Helló, Liskov!	94
13.1. Liskov helyettesítés sértése	94
13.2. Szülő-gyerek	95
13.3. Hello, Android!	96
13.4. Ciklomatikus komplexitás	97

IV. Irodalomjegyzék	98
13.5. Általános	99
13.6. C	99
13.7. C++	99
13.8. Python	99
13.9. Lisp	99

DRAFT

Ábrák jegyzéke

4.1. A double ** háromszögmátrix a memóriában	32
5.1. A Mandelbrot halmaz a komplex síkon	45
7.1. Qt c++ síklókilövő	74
7.2. Qt c++ síklókilövő egy kis idő	75
13.1. Ciklomatikus komplexitás	97

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 allokálni igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

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

Hogyan forgasd

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

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám felhasználjuk az egyetemi programozás oktatásban is: a reguláris programozás képzésben minden hallgató otthon elvégzendő labormérési jegyzőkönyvként, vagy kollokviumi jegymegajánló dolgozatként írja meg a saját változatát belőle. 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 (hogyan lássuk mást is) példával.

Hogyan nyomjuk?

Ránts 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 dlatex 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
dlatex 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 'dlatex' 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 találsz az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész

Bevezetés

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 jegyünk 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 kódcsipetek! 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 [**BMEPY**] 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/FktsFcooIG8>, az esporttal kapcsolatos sztereotípiák eloszlatására („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 [SMNIST] munkát ihlette, melyeket akár az MNIST-ek helyett lehet csinálni.
- Kódjátzsma, <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/>.

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

**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!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: Végtelen ciklus-t 2 féle képpen szokás írni

- while ciklussal :

```
int main()
{
    while(1)
    {

    }
}
```

- for ciklussal :

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

    }
}
```

Ha ezt futtatjuk 100%-on megy a processzor, viszont ha a gcc-be beépített OpenMP-s headert (omp.h) include-olva

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

párhuzamosan futtathatjuk végtelen ciklusunkat. A következő a párhuzamos futtatás szintaktikája :

```
#pragma omp parallel
{
    ide jön a fent megírt végtelen ciklus;
}
```

A feladat teljes forráskódja :

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

int main()
{
    #pragma omp parallel
    {
        while(1);
    }
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Fontos, hogy az fopen-t használva a fordítás "vegtelen.c" fájlnev esetén : gcc vegtelen.c -fopenmp -o vegtelen

Megjegyzés : "g++ vegtelen.c -fopenmp -o vegtelen"-t használva nem kell include-olni a omp.h headert.

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

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

Megoldás videó:

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

```
Program T100
{

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

    main(Input Q)
    {
```

```
    Lefagy(Q)
  }
}
```

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

    main(Input Q)
    {
        Lefagy2(Q)
    }
}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Két változó értékét megcserélni nem lenne nehéz,
swap függvény, segédváltozó, exor, stb.

Mi kell ahhoz, hogy mindezek nélkül cseréljük meg az értékeket?

- Valamilyen művelet aminek van ellentétje pl : osztás/szorzás összeadás/kivonás
- 2 változó deklarálva
- csere függvényben meghívva

Csere függvény :

```
void csere(int *a, int *b)
{
    printf("a = %d\nb = %d\n", *a, *b);
    *a = *a + *b; //a(17) = 12 + 5
    *b = *a - *b; //b(12) = 17 - 5
    *a = *a - *b; //a(5) = 17 - 12
    printf("\na = %d\nb = %d\n", *a, *b);
}
```

- Bekéri két változó memóriacímét : a,b.
- Kiírja a két változó nevét, majd az értékeket.
- Felcseréli őket összeadás/kivonás használatával.
- Kiírja a megcserélt változók értékeit.

```
#include <stdio.h>

void csere(int*, int*);

int main() {

    int a = 5;
```

```
int b = 12;

csere(&a, &b);

return 0;
}

void csere(int *a, int *b)
{
    printf("a = %d\nb = %d\n", *a, *b);
    *a = *a + *b; //a(17) = 12 + 5
    *b = *a - *b; //b(12) = 17 - 5
    *a = *a - *b; //a(5) = 17 - 12
    printf("\na = %d\nb = %d\n", *a, *b);
}
```

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés használata nélkül írd egy olyan programot, ami egy labdát pattogat a karakteres konzolon! (Hogy mit értek pattogatás alatt, alább láthatod a videón.)

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

Megoldás forrása: Iffel nem nehéz, csak feltételt kell írunk arra, hogy amikor a "labda" eléri a széleket visszapattanjon. Labdapattogatás if-el:

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

int
main ( void )
{
    WINDOW *ablak;
    ablak = initscr ();
    // noecho();
    // cbreak();
    // nodelay (ablak, true);

    int x = 0;
    int y = 0;

    int deltax = 1;
    int deltay = 1;
```

```
int keretx = 0;
int kerety = 0;

for ( ;; )
{

    getmaxyx( ablak, kerety,keretx);
    mvprintw ( y, x, "x:%dy:%d",x,y );
    refresh ();
    usleep ( 50000 );
    clear();

    x = x + deltax;
    y = y + deltay;

    if (x >= keretx - 1)
    {
        deltax = deltax * - 1;
    }
    if (y >= kerety - 1)
    {
        deltay = deltay * - 1;
    }
    if (y <= 0)
    {
        deltay = deltay * -1;
    }
    if (x <= 0)
    {
        deltax = deltax * -1;
    }

}

return 0;
}
```

If nélküli verzióhoz nagyjából az az ötlet, hogy csinálunk két int tömböt. Mind a két tömböt feltöltjük, 1-el (feltölt függvény), KIVÉVE azon koordináták helyére nem írunk 1-et, amelyek a szélén vannak. (pl.: adott egy 80x30-as ablakunk, akkor az 1. és 79. valamint az 1. és a 29. helyre -1-et írunk). Ez azért kell mert minden egyes "lépésnél" (lépés : egyszer lefut ami a for ciklusban van) azért mozog a "labda" mert 1-el növeljük az értékét mind a 2 tengelyen. Azzal, hogy feltöltöttük a két tömböt, -1 illetve 1-ekkel azért válik hasznunkra mert minden egyes "lépés"-nél megszorozzunk az adott koordináta 1, vagy -1-es értékével, ami azt eredményezi, hogy visszafelé fog elindulni a labdánk amikor a széleihez ér. Labdapattogtatás if nélkül :

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



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

//tombok feltöltése irányokkal (1,-1)
// az irányok értéke mindenhol 1, kivéve az első és az utolsó előtti pontot ↔
// ahol -1.
void feltolt(int tomb[],int max)
{
    for(int i=1; i<max; i++)
    {
        tomb[i]=1;
    }
    tomb[1]=-1;
    tomb[max-1]=-1;
}

int main() {

    //új ablak
    WINDOW *ablak;
    ablak = initscr();

    noecho();
    cbreak();
    nodelay (ablak,true);

    int mx = 0; //az ablak szélessége.
    int my = 0; //az ablak magassága.

    int x = 1; //y kezdő koordinátája.
    int y = 1; //y kezdő koordinátája.

    getmaxyx(ablak,my,mx);

    //mx = mx/2;//fele szélesség.
    //my = my/2;//fele magasság.

    int irány_x[mx];
    int irány_y[my];

    feltolt(irány_y,my);
    feltolt(irány_x,mx);

    // x és y mozgási sebessége/iránya.
    int deltax = 1;
    int deltay = 1;

    for (;;) //busy_loop
    {
        // x koordináta növelése 1-el.
```

```
x += deltax;
// y koordináta növelése 1-el.
y += deltay;

// x mozgási irányának beállítása.
deltax *= irany_x[x];
// y mozgási irányának beállítása.
deltay *= irany_y[y];

refresh();
clear(); // előzőleg rajzolt "labdák" törlése.
usleep(50000); // a "labda" sebessége. (minél kisebb annál gyorsabb) ↔
.
mvprintw(y ,x, "x:%dy:%d",x,y); // kiírja az aktuális koordinátákat.

}

return 0;
}
```

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!

Linus Torvalds a Linux atyja által kitalált BogoMipsel a régi Linux disztrók bootolása közben találkozhattunk, valami hasonló képpen:

```
rg@rg:~/Codes/Konyv/base/bhax/thematic_tutorials/Textbook_RG/textbook_new/bhax_textbook$ ../../Turing/BogoMIPS/bogo
1 2
1 4 <!--/programlisting-->
1 8
1 16 </section>
1 32
2 64 <section>
2 128 <title>Szóhossz és a Linus Torvalds féle BogoMIPS</title>
2 256 <para>
4 512 Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az <type>int</type> mérete.
6 1024 Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!
10 2048 </para>
20 4096 <para>
39 8192 Megoldás videó: <link xlink:href=""></link>
78 16384 </para>
159 32768 <para>
373 65536 Linus Torvalds a Linux atyja által kitalált BogoMipsel a régi Linux disztrók
625 131072 bootolása közben találkozhattunk, valami hasonló képpen:
1262 262144 <para>
1741 524288 <mediaobject>
1849 1048576 <imageobject>
4100 2097152 <imagedata fileref="files/bog.jpg" />
8355 4194304 </imageobject>
16278 8388608 </mediaobject>
33898 16777216
67341 33554432
137617 67108864 <prelisting language="c"><![CDATA[
265191 134217728 .h>
532227 268435456 .h>
1039886 536870912 .h>
ok - 1032, 00 BogoMips
```

A kód ami ezt eredményezte, a `BogoMips` mely arra szolgál, hogy a processzorunk gyorsaságát lemérje. Ha kicsit jobban belenézünk a kódba láthatjuk hogyan is működik. A `loops_per_sec`-et 1-re állítva, a `for` ciklus ciklusfejében, shiftelgetjük egy-el mindig. Tehát a `loops_per_sec` értékei a 2 hatványai lesznek lesznek :

1
2
4
8
16
32
64
128
...

Ami binárisan történni fog a `loops_per_sec`-el:

00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000

Ez arra lesz jó hogy a `delay()` függvényünk - amely azt csinálja, hogy 0-tól elmegy a paraméterként bekért értékig - betudja kérni paraméterként. Ahogy belépünk a `while` ciklusba egy szintén `unsigned long long` típusú `ticks`-be tároljuk a `clock()` értékét mely egy bizonyos tickszám amennyit a processzor tick-el egy adott idő alatt. Majd meghívjuk a `delay` függvényünket paraméterként a `loops_per_sec`-et átadva. Ezután ugyanugy a ticksben eltároljuk az eddig eltelt tickset a mostani `clock()` értékéből. Ezáltal megkapjuk a 0-`loops_per_sec`-ig elszámolás idejét.

Innentől már csak osztás/szorzás, hogy egy olvasható/mérhető értéket kapjunk.

Átvéve a forrás ciklusfejét hasonlóan használhatjuk számolásra is. Pl. egy szó hosszának megszámlálására hasonló képpen :

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

#define MAX_SZO 10

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

```
{
    char szo[10];

    int hossz = strlen(argv[1]);
    int szamlalo = 0;
    int bitek_szama = 0;

    unsigned long long int loops_per_sec = 1;

    while (loops_per_sec <= 1)
    {

        if (loops_per_sec % 8 == 0)
        {
            bitek_szama = hossz * 8;
            szamlalo++;
            if (szamlalo == hossz)
            {
                printf("szo hossza : %d karakter.\nszo merete : %d bit.\n", ←
                    hossz,bitek_szama);
                return 0;
            }
        }

    }

    // while(szoInt >= 1)
    // {
    //     printf("szo:%s\tszoInt:%d\n",szo,szoInt);
    // }

    return 0;
}
```

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: [PageRank.cpp](#) Az alábbi program a Google első, alap algoritmusának, egy 4 lapból álló weblaphálózat PageRank értékének kiszámolására szolgáló C++ implementációja. A PageRank algoritmust Larry Page és Sergey Brin feljesztette ki 1998-ban. Ez lehetővé teszi számunkra (és a Google számára) hogy meghatározza, egy lap mennyire "jó" a többi laphoz képest. Mindezt a rá mutató lapok száma és a rámutató lapok számának PageRank értéke befolyásolja. Jelenlegi 4 lapból álló hálózatunkra alkalmazva

az fentieket, gyakorlatilag egy mátrix szorzást kell elvégeznünk, majd az értékeket addig finomítani, míg meg nem közelítik az előzőleg megadott értéket, a pontosság érdekében.

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

void ki (double tomb[], int db)
{
    int i;
    for (i=0; i<db; i++)
    {
        std::cout<<"PageRank [" << i << "]" : " << tomb[i] << std::endl;
    }
}

double tavolsag(double PR[], double PR_v[], int db)
{
    double osszeg = 0.0;
    int i;
    for (i=0; i<db; i++)
    {
        osszeg+=abs (PR[i] - PR_v[i]);
    }
    return 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};
long int i=0, j=0;

    for (;;)
    {
        for(int i = 0; i < 4; ++i)
        {
            PR[i] = 0.0;
            for(int j = 0; j<4; ++j)
            {
                PR[i] += (L[i][j]*PRv[j]);
            }
        }
    }
}
```

```
    if (tavolsag(PR,PRv, 4) < 0.0000001)
        break;

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

ki (PR,4);
return 0;
}
```

Ugyanez vektorokkal.

```
#include <iostream>
#include <math.h>
#include <vector>
#include <iterator>

using namespace std;

void ki (vector<double> tomb)
{
    vector<double>::iterator it;
    for (it = tomb.begin(); it != tomb.end(); it++)
    {
        cout<<"PageRank ["<< distance(tomb.begin(),it)<<"] : " << *it << "↵"
        endl;
    }
}

double
tavolsag(vector<double> PR,vector<double> PR_v)
{
    vector<double>::iterator pri = PR.begin();
    vector<double>::iterator prvi = PR_v.begin();

    double tav = 0.0;
    for(pri = PR.begin();pri != PR.end();pri++)
    {
        tav += abs((*pri) - (*prvi));
        prvi++;
    }
    return tav;
}

int main(void) {
```

```
vector<vector<double>> L= {
    {0.0, 0.0, 1.0 / 3.0, 0.0},
    {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
    {0.0, 1.0 / 2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0 / 3.0, 0.0}
};

vector<vector<double>>::iterator sor;
vector<double>::iterator oszlop;
vector<double>::iterator Pi;
vector<double>::iterator PvI;

vector<double> PR;
    for(int i = 0; i < 4; i++)
        PR.push_back(0.0);
vector<double> PRv;
    for(int i = 0; i < 4; i++)
        PRv.push_back(1.0 / 4.0);

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

for (;;)
{
    PR = PRv;

    for(int i=0; i<PR.size(); i++)
    {
        double temp = 0.0;

        for(int j=0; j<PRv.size(); j++)
        {
            temp += L[i][j] * PR[j];
            PRv[i] = temp;
        }
    }

    if (tavolsag(PR,PRv) < 0.000001)
        break;
}

ki(PR);
return 0;
}

}
```

Tanulságok, tapasztalatok, magyarázat...

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

Tanulságok, tapasztalatok, magyarázat... A Monty Hall probléma szemléltetése a következő: - Adott 3 ajtó. - 2 ajtó mögött egy-egy kecske van, a harmadik mögött pedig egy új autó. - Választunk egy ajtót, ekkor egy a három közül, ami nem a választott és nem amelyik mögött az autó van kinyílik. - Ilyenkor élhetünk azzal a lehetőséggel, hogy válasszunk a 2 fennmaradó ajtó közül. - A kérdés, hogy megváltoztatjuk az választott ajtót, vagy maradunk az eredetnél. A helyes válasz minden esetben a változtatás. Ha választunk egy ajtót, annak 1/3-ad esélye van a nyerésre, a maradéknak 2/3. Miután kinyílik az az ajtó amelyik mögött nem az autó van, annak az esélye, hogy a "nem választott" másik ajtó mögött az autó van, ugyanúgy 2/3 lesz. Monty Hall probléma szimulációja R-ben :

```
# An illustration written in R for the Monty Hall Problem
# Copyright (C) 2019 Dr. Norbert Bاتفai, nbاتفai@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/>

kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

  if(kiserlet[i]==jatekos[i]){

    mibol=setdiff(c(1,2,3), kiserlet[i])

  }else{

    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))
```



```
    }

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

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

for (i in 1:kiserletek_szama) {

    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]

}

valtoztatesnyer = which(kiserlet==valtoztat)

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

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: Az Unáris talán a legegyszerűbb számrendszer. Annyi 'l' -t tartalmaz amennyi a szám decimálisan ötösével elszeparálva. Az alábbi program bekér egy számot decimális formában, amit egy egész változóban tárol, majd ezt kiírja unárisan.

```
#include <stdio.h>

int
main()
{
    int decimal = 0;
    scanf("%d",&decimal);
    decimal++;
    for(int i =1;i<decimal;i++)
    {
        printf("|");
        if (i %5 == 0)
        {
            printf(" ");
        }
    }
    return 0;
}
```

```
$ ./unary
14
|||| | ||| | |||
```

Tanulságok, tapasztalatok, magyarázat...

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

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

Változók $\in X, Y, Z$.

Konstansok $\in a, b, c$.

1. $X \rightarrow aXYZ$
 $aX \rightarrow aa aYZ$
 $Y \rightarrow bb aabb$
 $c \rightarrow Z aaYZ$
 $Z \rightarrow ccc aaYccc$
 $aaY \rightarrow aaabbb aaabbbccc$
2. $aXbc \rightarrow abXc$
 $ab \rightarrow aaa aaaXc$
 $X \rightarrow bYc aaabYc$
 $bYc \rightarrow bcY aaabcY$
 $Y \rightarrow c aaabcc$
 $bcc \rightarrow bbbccY aaabbbccY$
 $Y \rightarrow c aaabbbccc$

3.3. Hivatkozási nyelv

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

A legegyszerűbb példa olyan kódcsipetre, amely a c89-es szabvánnyal nem fordul le, a c99-essel pedig igen az nem más mint egy egyszerű for ciklusbeli deklaráció, valamint a `/*` kommentelés. A for ciklust akkor szoktuk használni amikor pontosan tudjuk (többnyire), hogy mettől meddig akarunk egy bizonyos műveletet elvégezni. pl: írassuk ki a számok négyzetét 1-10-ig:

```
#include <stdio.h>

int main()
{
    for(int i = 1;i<=10;i++) printf("%d\n",i*i );
    //ez egy komment lenne
    return 0;
}
```

Ha a fenti kódcsipetet futtatjuk `-std=c89` prefixel, a következő hibaüzeneteket kapjuk.

```
gcc temp.c -o temp -std=c89
temp.c: In function 'main':
temp.c:5:2: error: 'for' loop initial declarations are only allowed in C99  ↵
      or C11 mode
    for(int i = 1;i<=10;i++) printf("%d\n",i*i );
    ^~~
temp.c:5:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11  ↵
      to compile your code
temp.c:6:2: error: C++ style comments are not allowed in ISO C90
    //ez egy komment lenne
    ^
temp.c:6:2: error: (this will be reported only once per input file)
```

1. Probléma :

A for ciklusfejből nem engedélyezett a deklaráció, csak az értékadás.

2. Probléma :

A c89-es szabvány még nem tolerálja a `/**/` kommentet csak a `/* */` típusút.

C89-ES SZABVÁNYNAK MEGFELELŐ KÓD, UGYANAHHOZ A KÓDCSIPETHEZ:

```
#include <stdio.h>

int main()
{
    int i;
    for(i = 1;i<=10;i++) printf("%d\n",i*i );
    /* ez egy komment lenne */
    return 0;
}
```

3.4. Saját lexikális elemző

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

Ilyen féle feladatokhoz, amikor egy bemenetről, részletes "lexikai" elemzést akarunk készíteni, majd esetlegesen kiválogatni valamit, remekül alkalmazható a LeXer.

Lexer a következőt csinálja :

- A standard inputról érkező szimbólum sorozatokat, átkonvertálja tokenekké, amik valójában bizonyos stringek, különbözően azonosítva. Jelen esetben a lexxert a folyamatos bementről érkező szimbólumokat tokenizálja, majd kiselejtezi azokat, melynek racinális szám formája van.

```
%{
#include <stdio.h>
int realnumbers = 0;
}%
digit    [0-9]
%%
{digit}* (\. {digit}+)?    {++realnumbers;
    printf("[ felismert valos szam = %s %f ]\n", yytext, atof(yytext));}
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

[1337.1](#) A 1337 vagy leet cipher, egy úgymond számítógépes nyelv. Leggyakrabban előfordulása, ahol én találkoztam vele azok az online/multiplayer játékok. Gyakori, hogy valaki vagy csak poénból, vagy mert "hacker"-nek érzi magát.

A legérdekesebb előfordulása azonban a *DEAD COW CULT* hacker csoport által használt UDP port lásd : 31337, jelentése Elite ELEET, melyet a Window 95 feltöréséhez használtak. A következő program felismeri a leütött karaktereket, majd át"konvertálja" őket a kívánt 1337 verzióra:

Példa:

```
./l337
na lassuk mi lesz ennek a vege
n4 l4$su|< m1 l3sz 3nn3k @ v3g3
```

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 megváránzésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

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

Ha a SIGINT jel nincs figyelmen kívül hagyva, akkor innentől a jeleket a jelkezelő veszi irányításba.

ii.

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

0-4-ig megy a ciklus prefix inkrementálás.

iii.

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

0-4-ig megy a ciklus postfix inkrementálás.

iv.

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

0-4-ig megy és nem i-t hanem a tömb i-edik elemét növeli egyel.

v.

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

0-tól megy addig amíg vagy i el nem éri n-t vagy d nem lesz egyenlő s-el.

vi.

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

Függően attól, hogy mit csinál az f függvényünk, kiírja az f függvény által kiszámított értéket először úgy, hogy 1. paraméternek simán az a-t masodiknak a ++a-t keri be. Utána pedig fordított paraméterekkel.

vii.

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

Szintén nem lehet megmondani 100%-ra mit csinál a kódrészlet. Két számot írat ki az egyik f() függvény visszatérési értékét, úgy, hogy "a"-t adjuk paraméterként, a másik meg simán "a" lesz.

viii.

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

Ugyanazt csinálja mint az 1-el fentebbi kódcsipet, annyi különbséggel, hogy az f függvény "a" memóriacímét kapja.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\forall x \exists y ((x < y) \wedge (y \text{ prím})))$
$(\forall x \exists y ((x < y) \wedge (y \text{ prím})) \wedge (SSy \text{ prím})) \leftrightarrow$
  )$
$(\exists y \forall x (x \text{ prím}) \supset (x < y))$
$(\exists y \forall x (y < x) \supset \neg (x \text{ prím}))$
```

A fenti forrás egy latex formájú elsőrendű logika nyelven (Ar) íródott kifejezések. A Latex-et matematikai, logikai szövegek formázására használhatjuk.

Jelentések:

- \forall - BÁRMELY
- \exists - LÉZETIK
- \wedge - KONJUNKCIÓ
- \supset - IMPLIKÁCIÓ
- prím - SZÖVEG

Ha a feladat leírásából csinálunk egy pdf-et könnyebben rájöhethetünk mit is jelentenek. (na meg ha ismerjük az informatika logikai alapjait.)

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

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

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

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

- Végtelen prímszám létezik.
- Végtelen ikerprím létezik.
- Véges sok prímszám létezik.
- Véges sok prímszám létezik.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája
- 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 mutató mutatót visszaadó, egészet kapó függvény
- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

Mit vezetnek be a programba a következő nevek?

- `int a;`
egész
- `int *b = &a;`
egészre mutató pointer
- `int &r = a;`
egész referencia
- `int c[5];`
egész tömb
- `int (&tr)[5] = c;`
egész tömb referencia
- `int *d[5];`
egész tömbre mutató pointer

- `int *h ();`

egész re mutató pointert visszaadó függvény

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

egészre mutató, egész pointert visszaadó függvény pointer

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

egészre mutató mutatót visszaadó függvény

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

egészre mutató függvény pointer

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c)


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

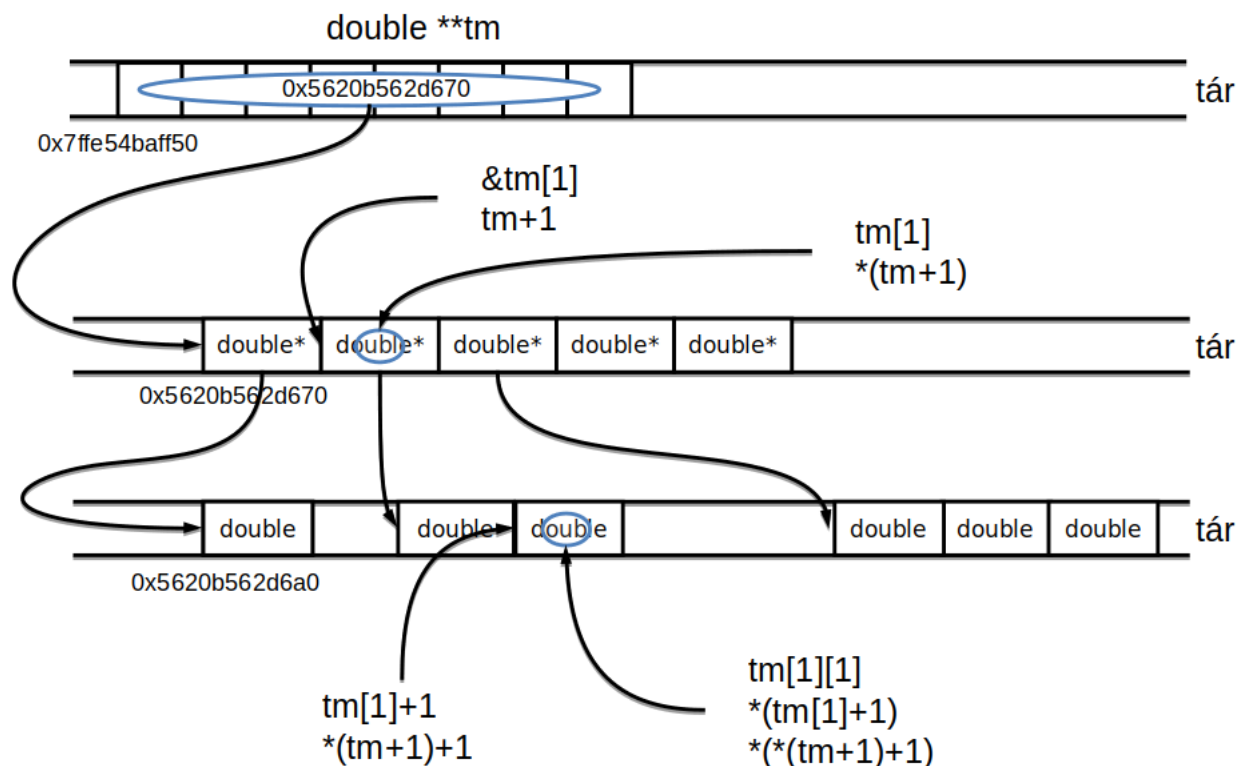
int main()
{
    double ** tm;
    int sorSzam = 5;

    if ((tm = (double**) malloc(sorSzam * sizeof(double*))) == NULL)
        return -1;

    for (int i = 0; i < sorSzam; i++) {
        if ((tm[i] = (double*) malloc ((i+1) * sizeof(double))) == NULL)
        {
            free(tm);
            return -1;
        }
    }
    int szamlalo = 0;
    // feltoltes ertekekkel
    for (int i = 0; i < sorSzam; i++)
    {
        for (int j = 0; j < i+1; j++) {
            szamlalo++;
            tm[i][j] = szamlalo;
        }
    }
}
```

```
    }  
}  
  
// kiiras formazva  
for (int i = 0; i < sorSzam; i++) {  
    for (int j = 0; j < i+1; j++) {  
        printf("%0.f ", tm[i][j]);  
        if (tm[i][j] <= 9) printf(" ");  
    }  
    printf("\n");  
}  
  
tm[3][0] = 42.0;  
(* (tm+3))[1] = 43.0; // ha nincsen zárójel akkor a [1]-nek magasabb a ←  
    precedenciája mind a dereference operatornak  
*(tm[3]+2) = 44.0;  
* (* (tm+3)+3) = 45.0;  
  
// kiiras formazva ujra  
for (int i = 0; i < sorSzam; i++) {  
    for (int j = 0; j < i+1; j++) {  
        printf("%0.f ", tm[i][j]);  
        if (tm[i][j] <= 9) printf(" ");  
    }  
    printf("\n");  
}  
  
// elemek felszabaditasa  
for (int i = 0; i < sorSzam; i++) {  
    free(tm[i]);  
}  
  
// tm felszab  
free(tm);  
  
return 0;  
}  
}
```





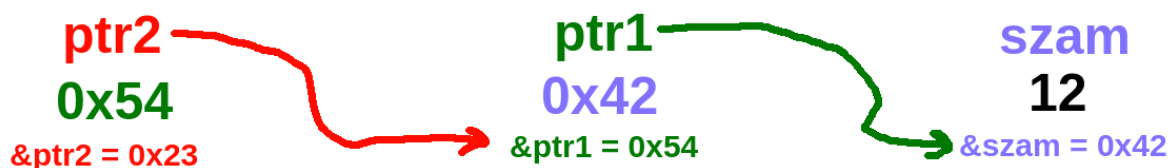
4.1. ábra. A `double **` háromszögmátrix a memóriában

A `double **` háromszögmátrix célja a mutatók, a memóriefoglalás, referencia használatának szemléltetése. Amit a fenti program csinál az az, hogy lefoglal a memóriában elegendő helyet egy alsó háromszög mátrixnak. A `sorSzam` változó meghatározza hogy hány sorunk lesz ezáltal egyben azt is, hogy hány elemből fog állni az utolsó sorunk. A foglalás/felszabadításra a `malloc/free` utasításokat használjuk, mivel C-ben vagyunk. C++-ban ezek a `new/delete` utasítások lennének.

A C, valamint a C++ nyelvek nagy előnye, alacsony szintű nyelvekhez méltóan, a dinamikus memóriefoglalási lehetőség. Ahelyett, hogy az ember - mint más programozási nyelvekben - előre definiált méretű helyet foglalna a memóriában, pl (30x3-as 2D tömbbel szemléltetve) :

```
int tomb[30][3];
```

Ez a példa egy fordító adta kényelem, egyszerűsítés, viszont néha a részletekben rejlik a lényeg és ebben az esetben, a részletekhez kicsit bonyolítanunk kell a dolgokat. Használjunk hát pointereket, magyarul mutatókat. Még pontosabban, mutatóra mutató mutatókat.



```
int **tomb;
```

Majd az stdlib.h header használatával kézzel memóriát lefoglalni neki.

```
#define MAX 3
tomb = (double **)malloc(MAX * sizeof(double));
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: egy részletes feldolgozása az [posztban](#), lásd az e.c forrást.

Az EXOR titkosításról legkevesbé sem lehet kijelenteni, hogy biztonságos, inkább egyfajta szemléltetése a XOR művelet egyik hasznosságának. A program úgy működik, hogy a titkosítandó szövegünk és a bekért kulcsunk alapján végig megy a titkosítandó szöveg bitein, melyet folytonosan összeexoroz a kul

Az exor titkosítás, ha csak nem használunk nagyon hosszú kulcsot, nem túl effektív titkosítási mód. Mint nevében is benne van, hogy ez a fajta titkosítás EXOR műveletet alkalmaz. Az EXOR vagy XOR kizáró vagy-ot jelent jelen esetben ezt bitekre értve, ha két bit különböző akkor 1 lesz a két bit EXOR-ja, ha a két bit megegyezik akkor pedig 0. A megértéshez kis maszkolásra lesz szükség. Legyen a és b esetén :

int x = 'a'; - ASCII kódja : 97

binárisan : 01100001

int y = 'b'; - ASCII kódja : 98

binárisan : 01100010 EXOR x ^ y = ?

'a':01100001

'b':01100010

x^y:00000011

x ^ y = 3

Most, hogy tudjuk mi az az EXOR titkosítás, lássuk a megvalósítást C-ben! A programunk így működik : Az 1. parancssori argumentumunkat bekérjük kulcsként. Olvasunk a std::in-ről és végigmenve a fájlunk bájtjain, exorozzuk őket folyamatosan a kulccsal.

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

#define MAX_KULCS 5
#define BUFF_SIZE 256

int main(int argc, char **argv)
{
    char buffer[BUFF_SIZE];
    char kulcs[MAX_KULCS];

    int olvasott_bajtok = 0;
    int kulcs_index = 0;

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

    while((olvasott_bajtok = read(0, (void *) buffer, BUFF_SIZE)) <= 0)
    {
        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);
    }

    return 0;
}
```

Amit ezek után a standard outputra megadott fájlban találunk pontosan az amire vártunk : bináris szemét.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

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

```
public class exor {
```

```
public exor(String kulcs,
            java.io.InputStream inc,
            java.io.OutputStream outg)
    throws java.io.IOException {

    byte [] kulcs = kulcs.getBytes();
    byte [] buffer = new byte[256];
    int kulcsIndex = 0;
    int olvasottByteok = 0;

    while((olvasottByteok =
        inc.read(buffer)) != -1) {

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

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

        }

        outg.write(buffer, 0, olvasottByteok);

    }

}

public static void main(String[] args) {

    try {

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

    } catch(java.io.IOException e) {

        e.printStackTrace();

    }

}
```

4.4. C EXOR törő

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

Megoldás forrása: egy részletes feldolgozása az [posztban](#), lásd az t.c forrást.

C EXOR törő működési elve a következő: Elkezdjük előállítani a lehető összes kulcsot, majd minden egyes kulccsal vissza-exorozzuk a forrásunkat, melyet a standard iputról olvastunk be. Aztán, mivel valami

alapján ki lehetne deríteni, hogy melyik az értelmes szöveg, illetve melyik nem ezért számoljuk az átlag szóhosszt valamint azt is figyelembe vesszük, hogy hány db értelmes magyar szót tartalmaz a szöveg. Ha ezeknek a függvényeknek eleget tesz titkos.szövegünk, akkor megtaláltuk a kulcsot.

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 5
#define _GNU_SOURCE

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

double atlagos_szohossz(const char *titkos, int titkos_meret);
int tiszta_lehet(const char *titkos, int titkos_meret);
void exor(const char kulcs[], int kulcs_meret, char titkos[],
          int titkos_meret, char *buffer);
void exor_tores(const char kulcs[], int kulcs_meret, char titkos[],
                int titkos_meret);

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

    // titkos fajt berantasa
    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;

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

    char ii, ki, ji, li, mi;

#pragma omp parallel for private(kulcs, ii, ki, ji, li, mi) shared(p, ←
    titkos)
    // osszes kulcs eloallitasa
    for (ii = 'A'; ii <= 'Z'; ++ii)
        for (ji = 'A'; ji <= 'Z'; ++ji)
            for (ki = 'A'; ki <= 'Z'; ++ki)
                for (li = 'A'; li <= 'Z'; ++li)
```



```
        for (mi = 'A'; mi <= 'Z'; ++mi) {
            if ((kulcs =
                (char *)
                malloc(sizeof
                    (char) *
                    KULCS_MERET))
                == NULL) {
                printf
                    ("Memoria (kulcs) falióra\n");
                exit(-1);
            }

            kulcs[0] = ii;
            kulcs[1] = ji;
            kulcs[2] = ki;
            kulcs[3] = li;
            kulcs[4] = mi;

            exor_tores(kulcs,
                        KULCS_MERET,
                        titkos,
                        p -
                        titkos);
        }

    return 0;
}

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 valószínűleg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket

    double szohossz = atlagos_szohossz(titkos, titkos_meret);

    return szohossz > 3.0 && szohossz < 9.0
}
```

```
&& strstr(titkos, "hogy") && strstr(titkos, "nem")
&& strstr(titkos, "ne")
&& strstr(titkos, "az") && strstr(titkos, "ha");
}

void exor(const char kulcs[], int kulcs_meret, char titkos[],
         int titkos_meret, char *buffer)
{
    int kulcs_index = 0;
    for (int i = 0; i < titkos_meret; ++i) {
        buffer[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}

void exor_tores(const char kulcs[], int kulcs_meret, char titkos[],
               int titkos_meret)
{
    char *buffer;

    if ((buffer =
        (char *)malloc(sizeof(char) * titkos_meret)) == NULL) {
        printf("Memoria (buffer) falióra\n");
        exit(-1);
    }

    exor(kulcs, kulcs_meret, titkos, titkos_meret, buffer);

    if (tisztalehet(buffer, titkos_meret)) {
        printf
            ("Kulcs: [%c%c%c%c%c]\nTiszta szöveg: [%s]\n",
             kulcs[0], kulcs[1], kulcs[2], kulcs[3], kulcs[4],
             buffer);
    }

    free(buffer);
}
```

4.5. Neurális OR, AND és EXOR kapu

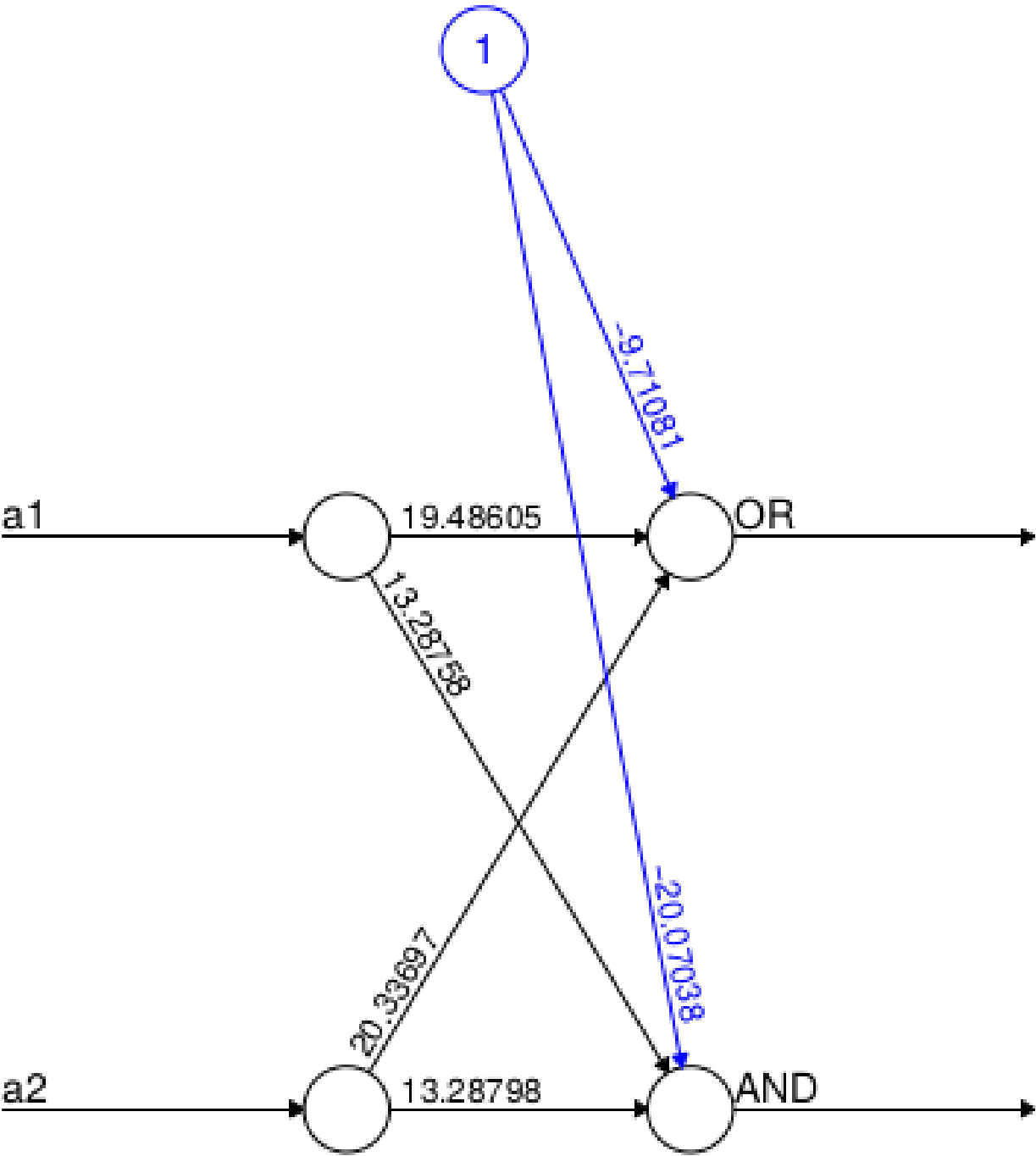
R

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

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

Az R egy szintaktailag sokkal egyszerűbb mint a magas szintű programozási nyelvek nagy rész. Főként statisztikai adatok szemléltetése, AI, ML, BigData, DataScience, Business Intelligence a felhasználásának nagy

része. Nekünk is hasonló a feladatunk. A Feladat, hogy felépítsünk egy egyszerű "neurális hálót" amelyet arra tanítunk, hogy meg tudja különböztetni a más programozási nyelvekben használt logikai operátorokat: or ,and,exor. OR: A vagy operátor, ha az egyik állítás igaz akkor az egész feltétel igaz.(0 1),(1 0) AND: Az és operátor, csak akkor igaz, ha mind a két állítás igaz.(1 1) EXOR: A kizáró vagy operátor, amely csak akkor igaz, ha vagy az egyik, vagy a másik állítás igaz. A különbség a vagy-hoz képest, hogy sima OR akkor is igaz, ha mind a két állítás igaz:(1,1), azonban az EXOR ebben az esetben hamis lesz. Ezekre a logika műveletekre fogjuk megtanítani a neurális hálónk segítségével a programunkat. Íme a neurális hálónk tanítási gráfjai:



Error: 2e-06 Steps: 203

Három double vár a a konstruktor, melyebe a perception memóriakezelés szemléltetése gyanánt a mandelbrot halmazunk rbg kódjait(RGB színek:red,green,blue a várt szín piros zöld és kék értékeinek megadásával érhetjük el). Ezeket pásztjuk bele a Perception osztályunk konstruktorába.

4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve

Megoldás videó: <https://youtu.be/-GX8dzGqTdM>

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

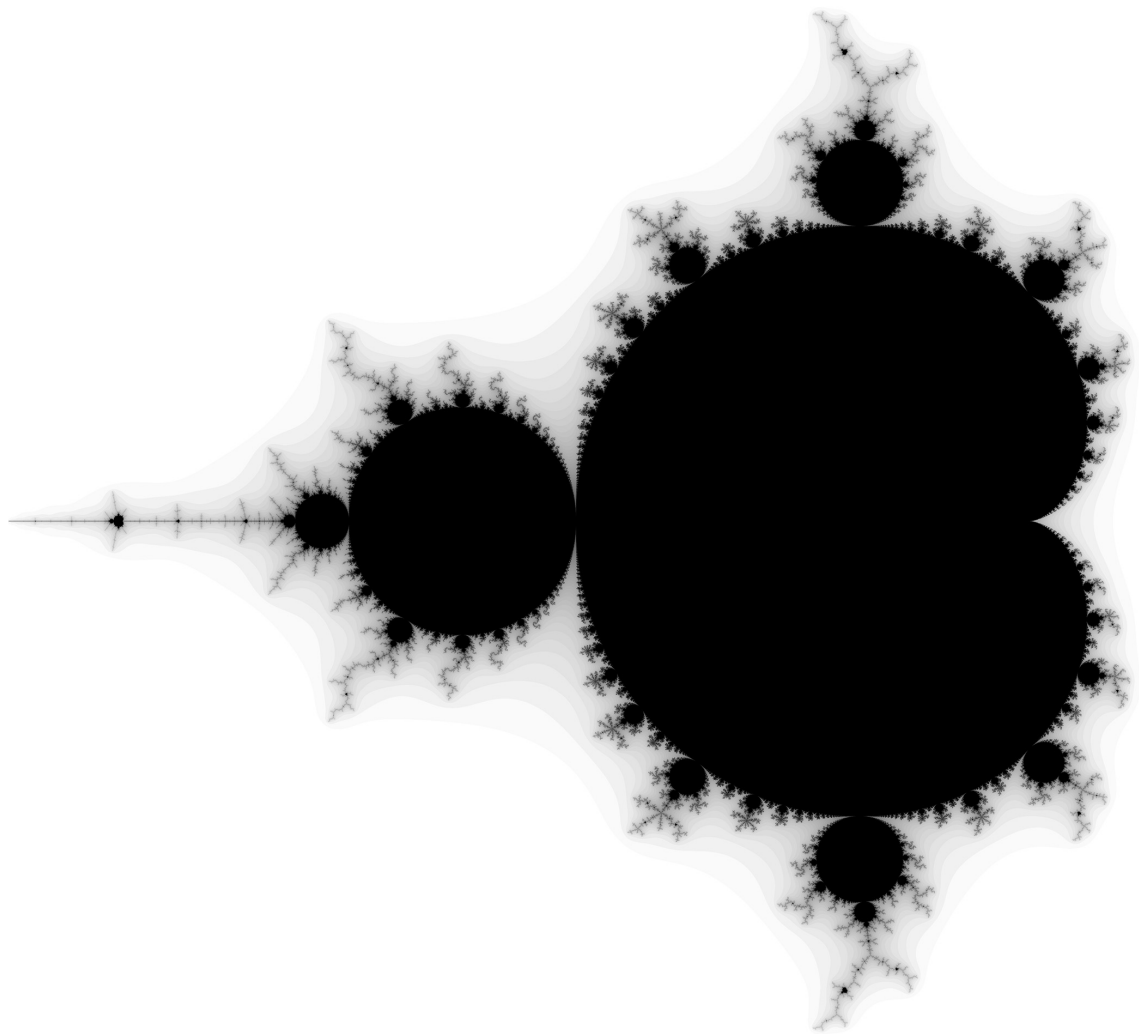
Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás forrása: A Mandelbrot halmaz segítségével létrehozhatunk egy úgy nevezett fraktál féle alakzatot. A mögötte lévő matematikáról, annyit, hogy az alábbi képlet segítségével dolgozik programunk: $f_c(z) = z^2 + c$ Érdekessége az, hogy ha kvázi egy végtelenített zoomot vezetünk be az alakzatra, amit legelőször látunk :



akkor ugyanúgy egy bizonyos iterációs szám után visszatér a fenti fraktál.

```
#ifndef POLARGEN__H
#define POLARGEN__H

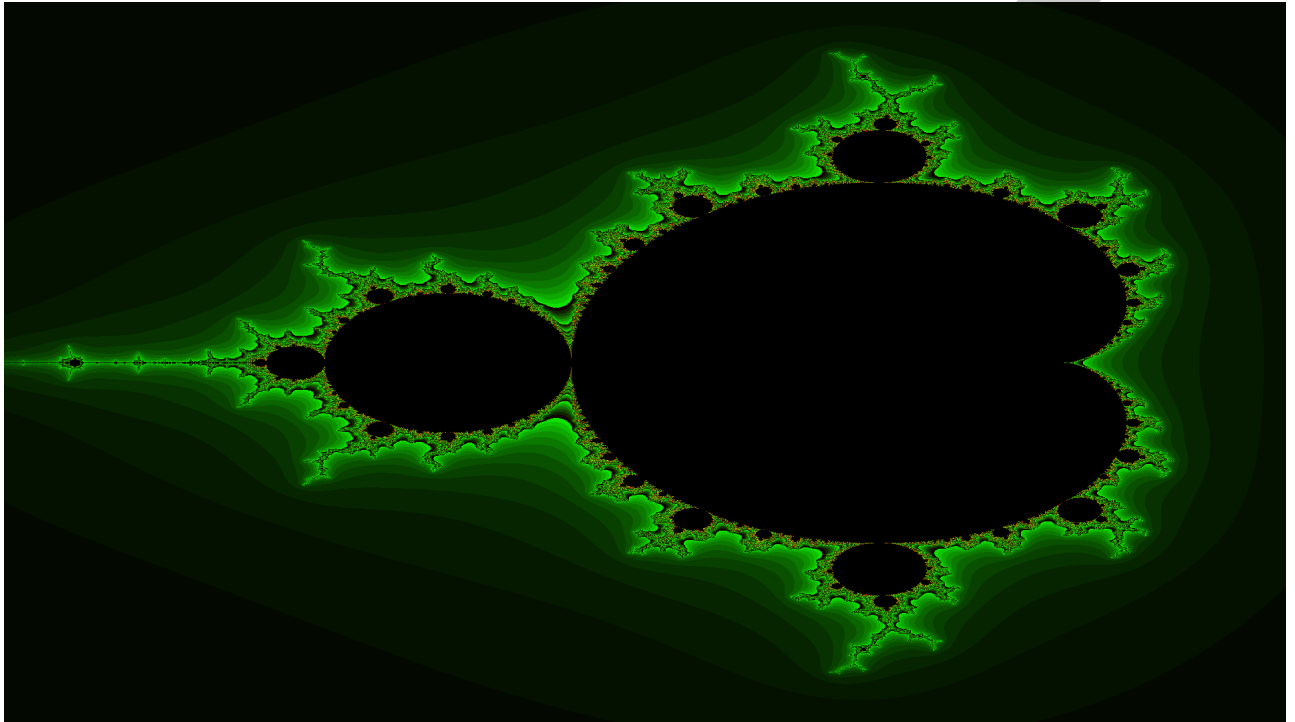
#include <cstdlib>
#include <cmath>
#include <ctime>

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

    double kovetkezo();
private:
```



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



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 800×800 -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 végtelen 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).

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

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

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

Megoldás forrása: BHAX repó, https://gitlab.com/nbatfai/bhax/-/blob/master/attention_raising/Mandelbrot/3.1.2.cpp

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhax/attention_raising/Mandelbrot/3.1.2.cpp](https://gitlab.com/nbatfai/bhax/-/blob/master/attention_raising/Mandelbrot/3.1.2.cpp) nevű állománya.

```
// 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
// Nyomtatas:
// 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 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/>.

#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;
}
```

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbRzY76E>

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

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 rácspontra ugyanaz.

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon
    for ( int k = 0; k < szelesseg; ++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 változóinak elnevezését összhangba hozzuk a közlemény jelöléseivel:

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatás:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ←
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
// color
//
// 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/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ←
// Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

int
```

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

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

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

    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ←  

            d reC imC R" << std::endl;
        return -1;
    }

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

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

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

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

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

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

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

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

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

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

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

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

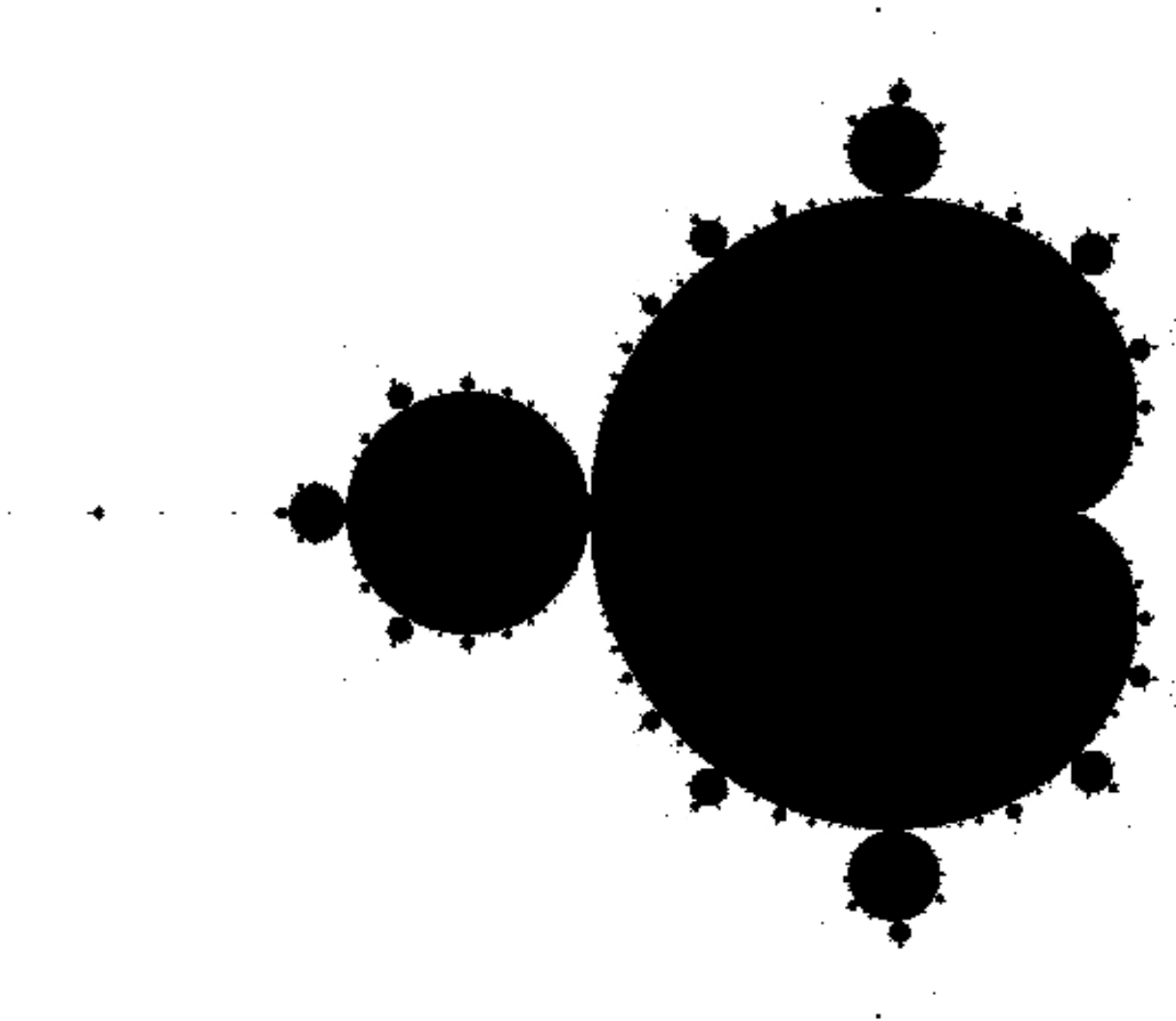
5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: [bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu](https://github.com/bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu) nevű állománya.

A CUDA (Compute Unified Device Architecture) egy párhuzamos számítási platform és API, melyet az NVIDIA fejlesztett ki. A CUDA platform C,C++ és Fortran nyelvekkel való munkára lett kialakítva.

A Mandelbrot halmaz CUDA megvalósításához a mandelpngcu.cu nevű programot fogjuk használni. Usage : **nvcc mandelpngcu.cu -lpng16 -o mandel**, majd **./mandel img.png**. A img.png a következő képpen fog kinézni:



5.5. Mandelbrot nagyító és utazó C++ nyelven

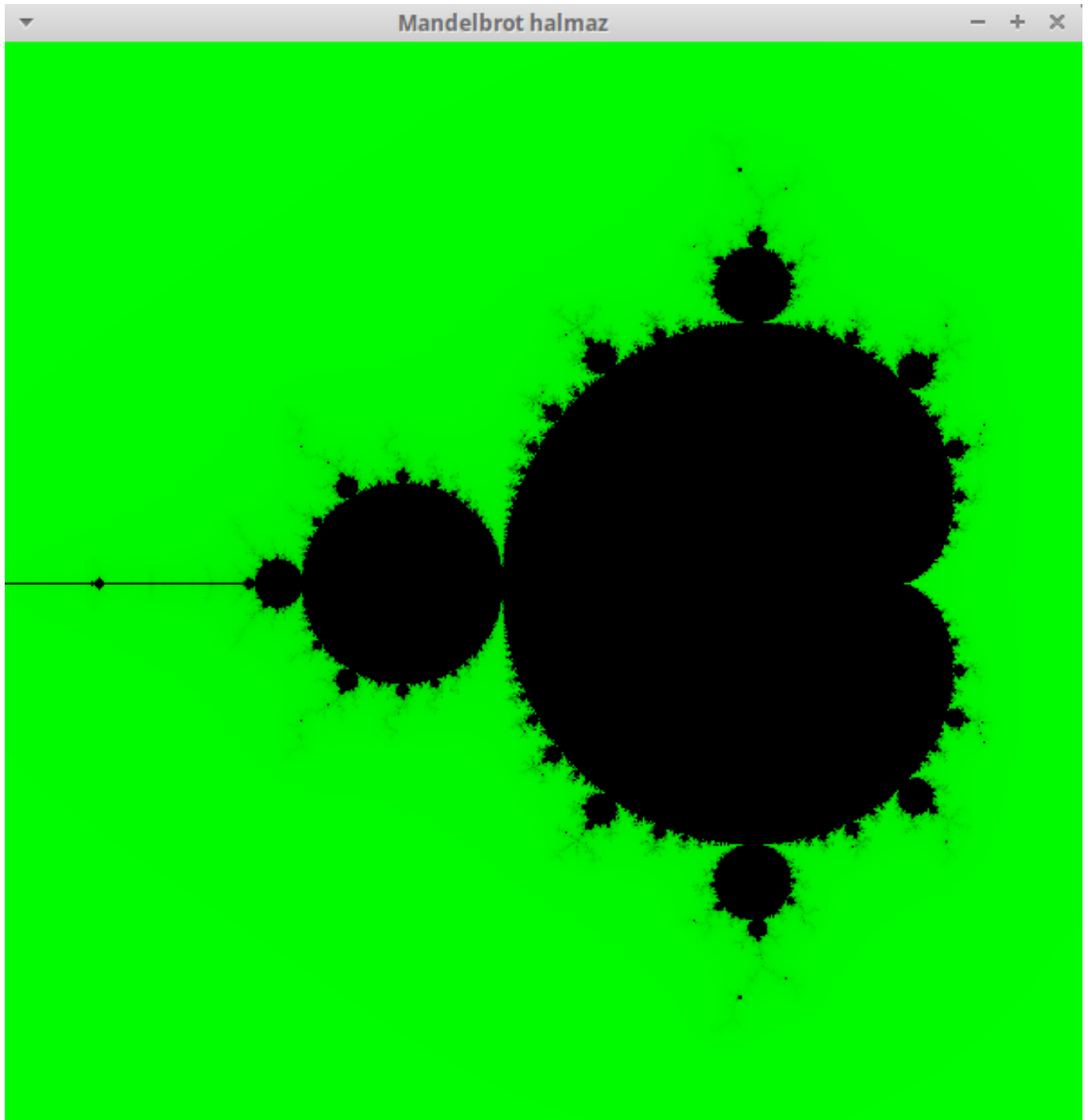
Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteráció bejárta z_n komplex számokat!

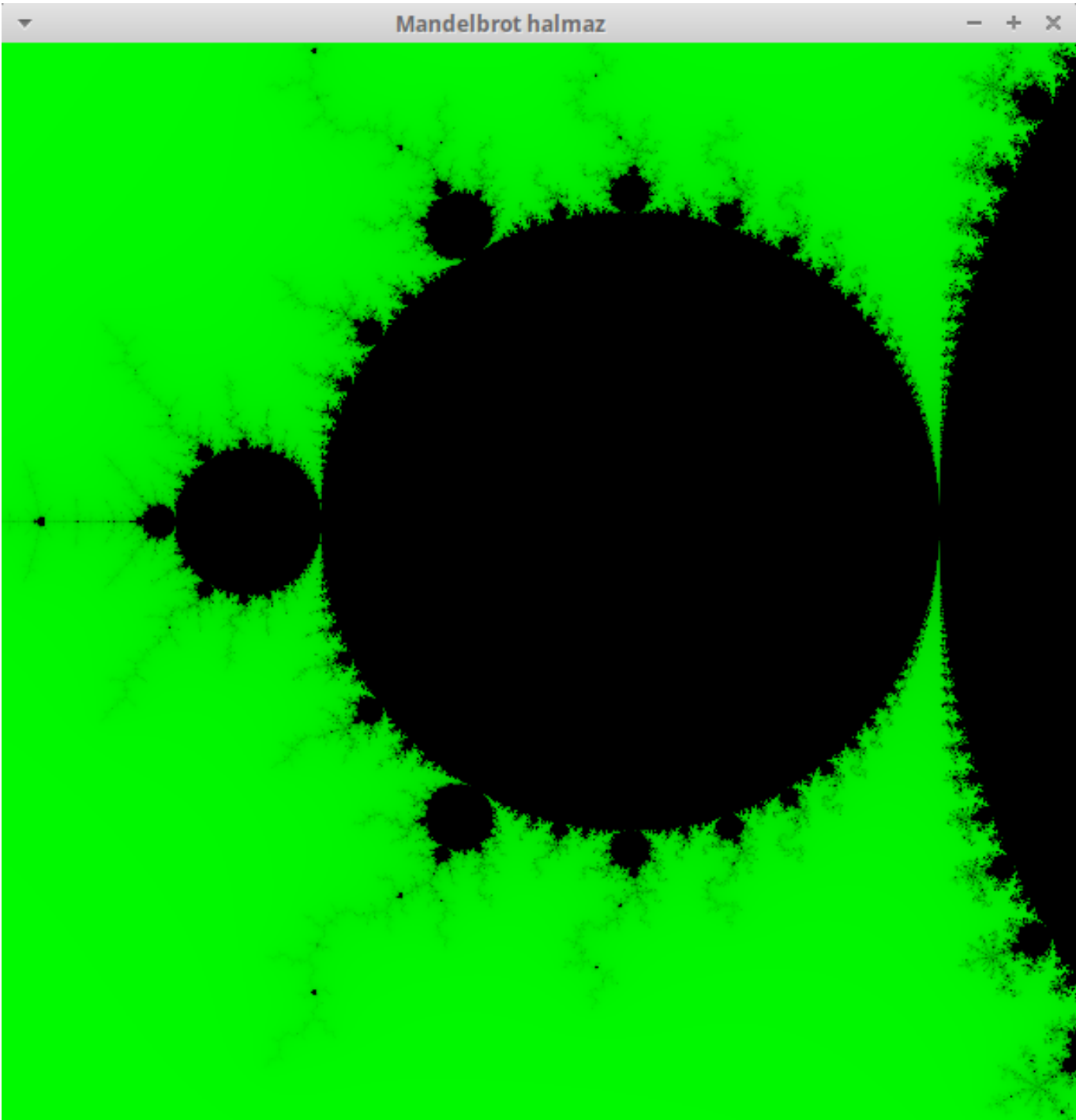
Megoldás videó: Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal.

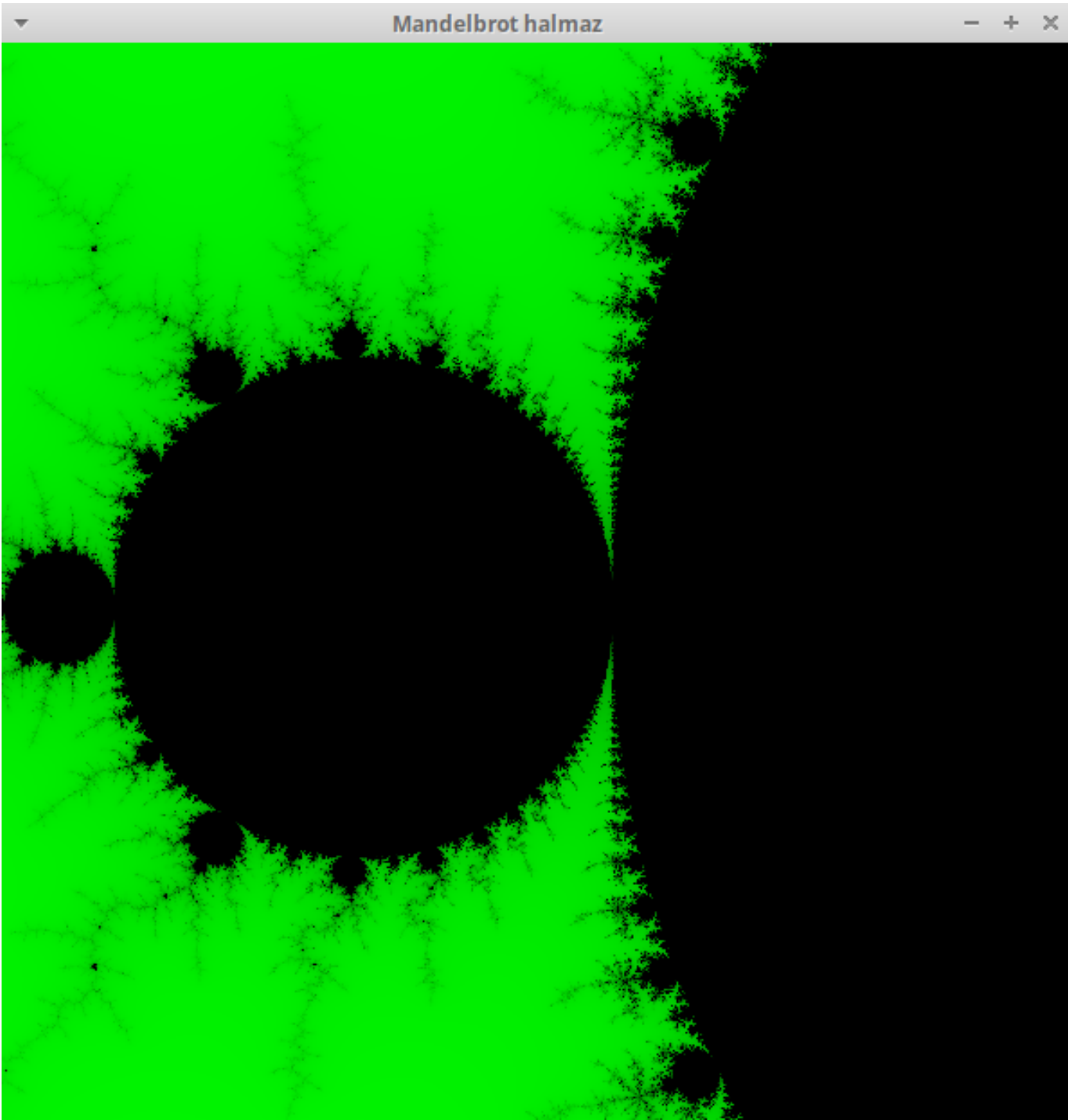
Megoldás forrása: az ötödik előadás 26-33 fólia, illetve <https://sourceforge.net/p/udprog/code/ci/master/-tree/source/binom/Batfai-Barki/frak/>

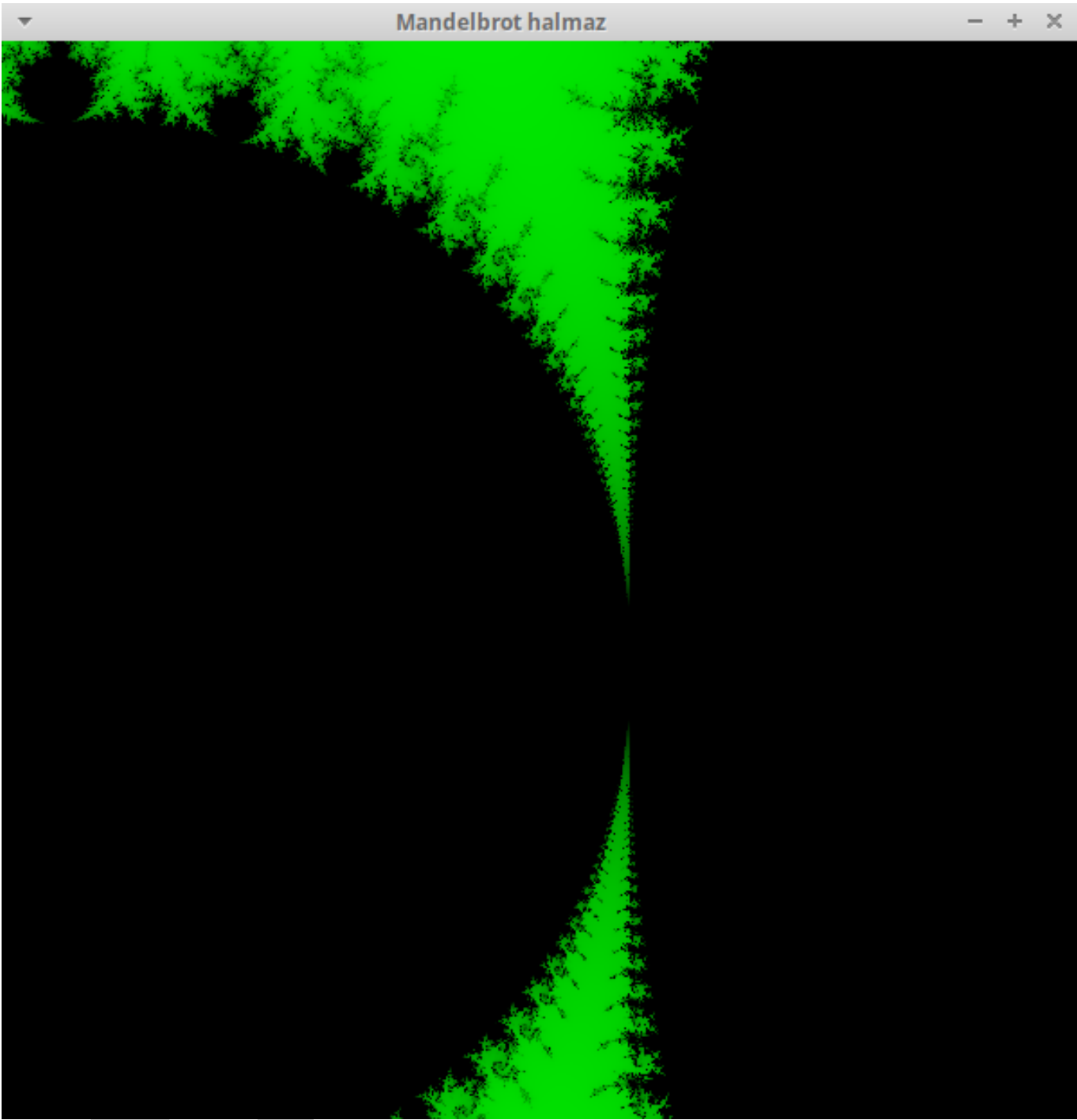
Mar a Prog1 előtt is néztem fél órás videókat Mandelbrot Zoomokrol. Valahogy az emberi szem, illetve

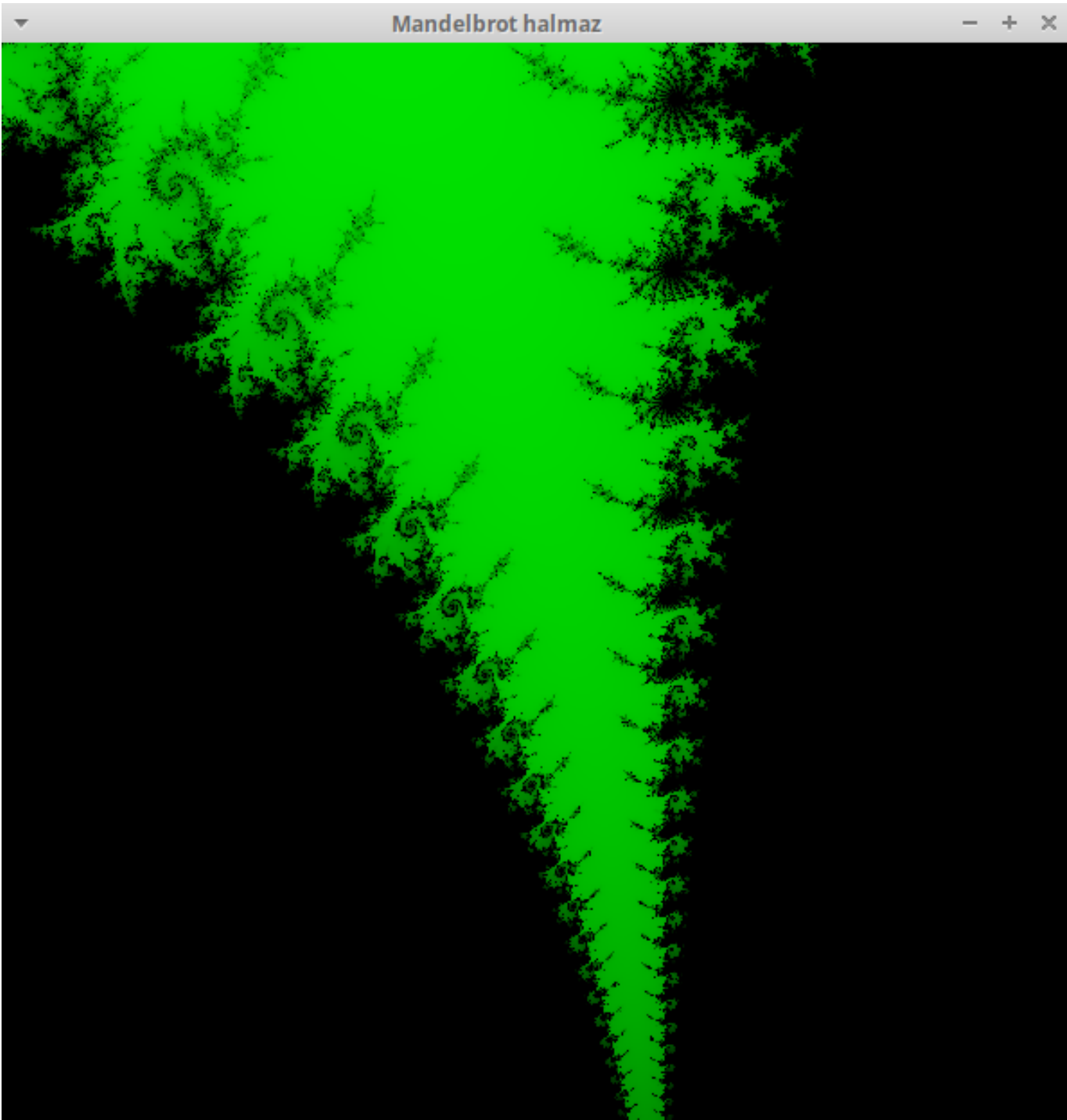
agy nehezen tudja felfogni, hogy mindig van tovább, talán az egyik legjobb módszer a "végtelen" szemléltetésére. Itt is ezt a hatást tudjuk elérni azonban egy bizonyos iterációs szám után elpixelesedik, tehát nem lehet a végtelenbe zoomolni, azonban egész sokáig, tovább mint az agy számítana rá. Az érdekesség az, hogy gyakran új alakzatok úgy nevezett fraktálok rajzolódnak ki azonban a fraktáloknak van egy bizonyos patternje, ami folyamatosan ismétlődik.

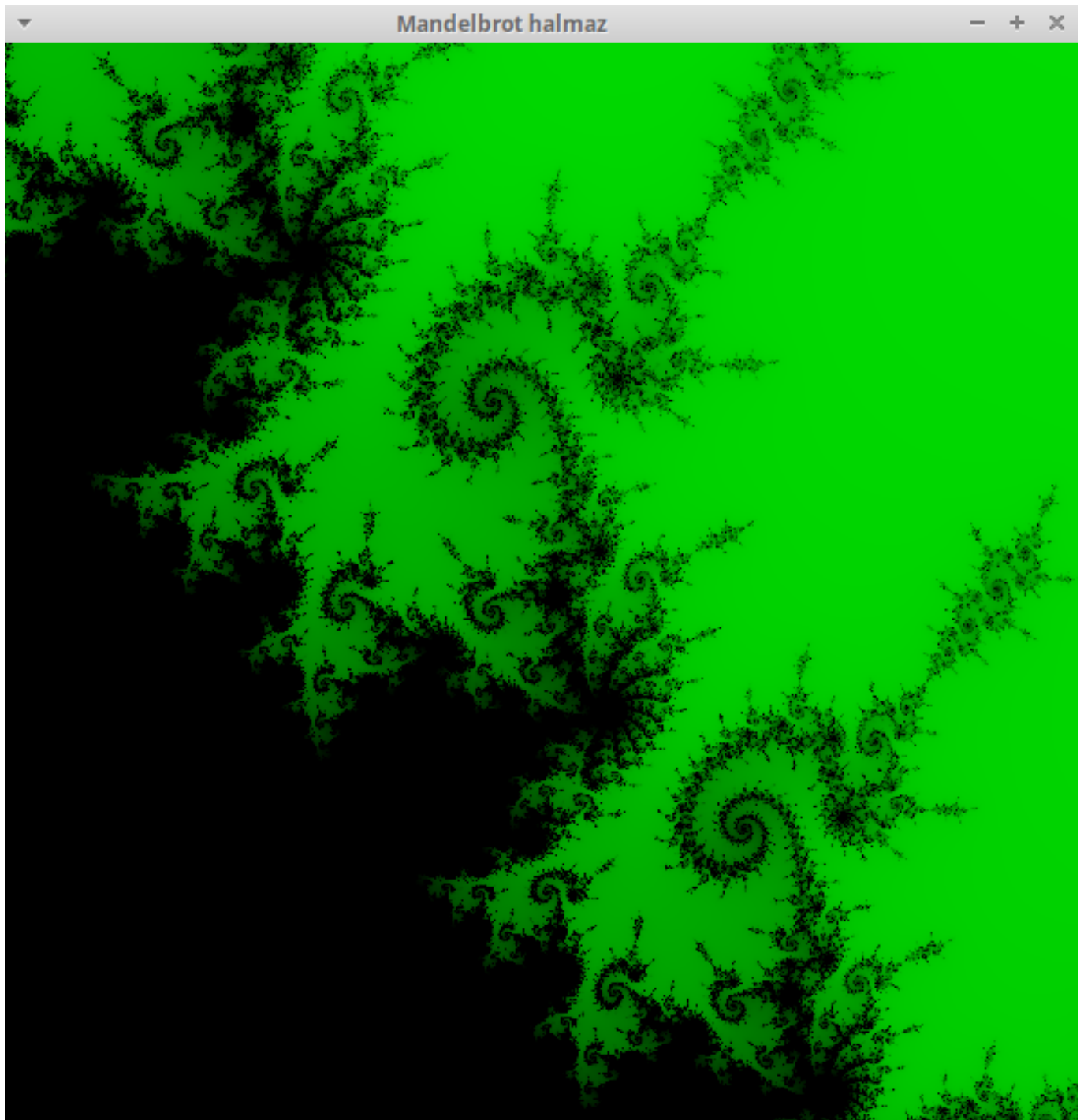












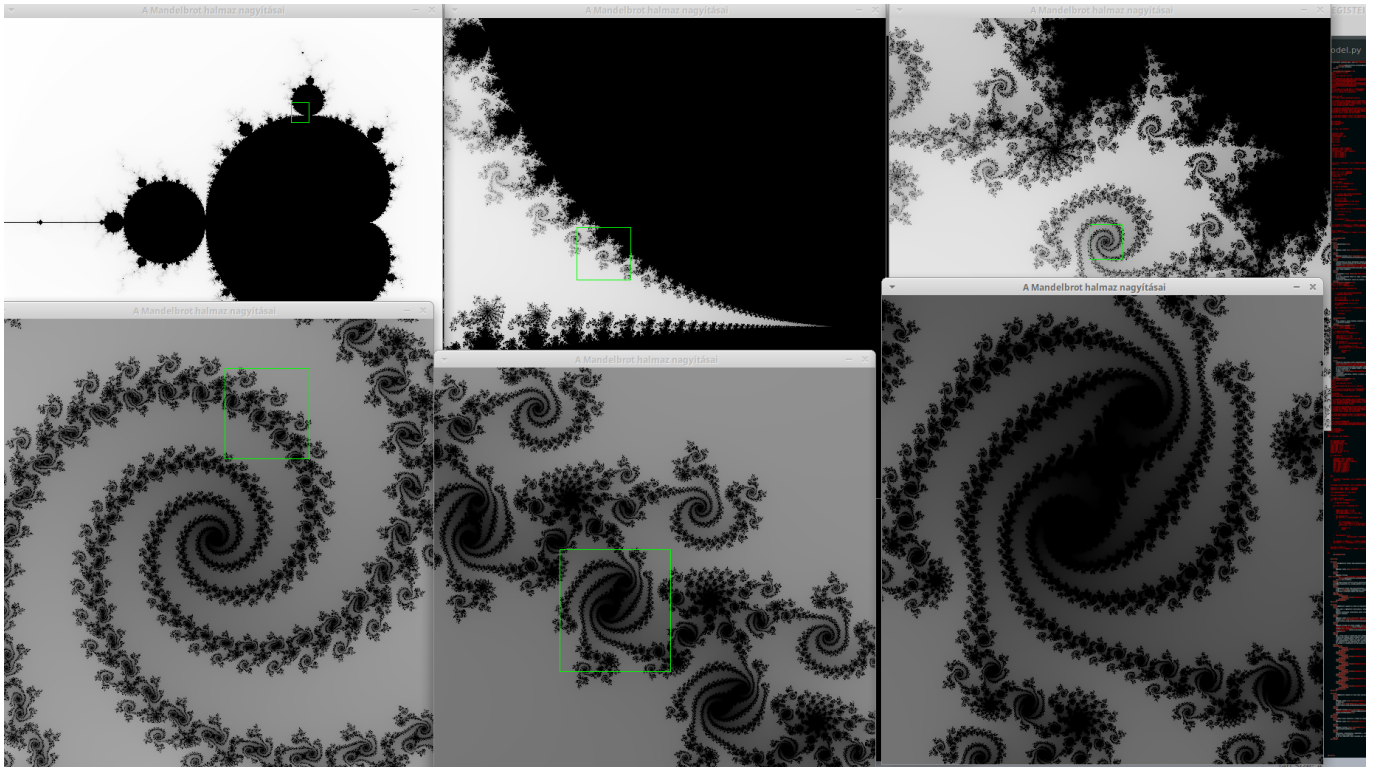
5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJnssY>, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a

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

Az eljárás hasonló azonban, egy különbség, hogy amikor C++-ban zoomoltunk, abba az ablakba töltöttük be a zoomolt részt, amiben volt az eredeti halmazunk vizualizációja. Jelen esetben, viszont minden egyes zoomolásnál új ablakot hozunk létre. Ráadás az, hogy a zoomolt terület, ahol kijelöltük a kis négyzet,

megmarad az előző ablakban így nyomon lehet követni, hogy hova zoomoltunk is pontosan melyik képen:



5.7. Vörös Pipacs Pokol/fel a láváig és vissza

Megoldás videó: <https://youtu.be/I6n8acZoyoo>

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

6. fejezet

Helló, Welch!

6.1. Első osztályom

Valósítsd meg C++-ban és Java-ban az módosított polártranszformációs algoritmust! A matek háttér teljesen irreleváns, csak annyiban érdekes, hogy az algoritmus egy számítása során két normálist számol ki, az egyiket elspájzolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltérő kiszámolt szám. A programot 3 részre osztjuk, egy headerre, ami az osztály váza. Egy osztály implementációra. Végül pedig egy main-re ami példányosítja a PolarGen osztályt és meghívja a példány `kovetkezo()` függvényét. Header:

```
#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
```

Implementáció:

```
#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;
    }
}
```

Main :

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

int main(int argc, char**argv){
    PolarGen pg;
    for (int i{0}; i<10; ++i)
        std::cout<<pg.kovetkezo()<<std::endl;
    return 0;
}
```

A javas verzió:

```
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());
}
}

```

Megoldás forrása: a második előadás [17-22 fólia](#).

Az OOP szemlélet a mai programozói világ esszenciális eszköze. Lehetővé teszi, hogy osztályokban gondolkozzunk. Ezáltal lehetségessé válik, hogy új példányt hozzunk létre az általunk deklarált osztályból.

6.2. LZW

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

Megoldás forrása: https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_p

A fenti program az LZW algoritmus fa építési eljárása. Készítünk egy LZW-fát, melynek a Bináris fa az alosztálya, melynek alosztálya a Node osztály amely a fa egy-egy csomópontjának értékét valamint a bal és jobb oldali gyermekére mutató mutatókat tartalmaz. A BinFa és az LZWBInFa template osztályok ami azt jelenti, hogy nincs előre deklarált bemeneti típusuk, tehát az osztály példányosításánál meg kell adnunk a shift operátor által bevitt érték típusát. Az alábbi kódban int típusú binfát és char típusú LZW fát hozunk példányosítunk.

```

BinTree<int> MyBinaryTree;
MyBinaryTree << 7 << 1 << 9 << 6 << 6 << 10 << 7 << 9 << 9 << 3 <<
    << 4 <<5 <<1;
LZWTree<char> MyLZWTree;
MyLZWTree <<'0'<<'1'<<'1'<<'1'<<'1'<<'0'<<'0'<<'1'<<'0'<<'0'<<'1'<<
    '0'<<'0'<<'1'<<'0'<<'0'<<'0'<<'1'<<'1'<<'1';

```

6.3. Fabejárás

Járd be az előző (inorder bejárású) fát pre- és posztorder is! A bejárások típusai a bal, jobb, és a gyökér csomópont kiírási sorrendjén múlik, pre order módon járjuk be a fát, ha az sorrend a következő: - Root - Left - Right

```
template <typename Vtype>
void BinTree<Vtype>::print(Node*node, std::ostream & os )
{
    if (node)
    {
        ++depth;
        for (int i=0; i < depth; ++i)
        {
            os<<"---";
        }
        os << node->getValue()<<std::endl; //<< " ("<<depth<<") "<<node-> ←
            getCount()<<". "<<std::endl;
        print(node->leftChild(), os);

        print(node->rightChild(), os);
        --depth;
    }
}
```

Postorder: - Left - Right - Root

```
template <typename Vtype>
void BinTree<Vtype>::print(Node*node, std::ostream & os )
{
    if (node)
    {
        ++depth;
        print(node->leftChild(), os);
        print(node->rightChild(), os);
        for (int i=0; i < depth; ++i)
        {
            os<<"---";
        }
        os << node->getValue()<<std::endl; //<< " ("<<depth<<") "<<node-> ←
            getCount()<<". "<<std::endl;
        --depth;
    }
}
```

Megoldás forrása:

6.4. Tag a gyökér

Az LZW algoritmust ültess át egy C++ osztályba, legyen egy Tree és egy beágyazott Node osztálya. A gyökér csomópont legyen kompozícióban a fával! [z3a7.cpp](#) A z3a7.cpp forrásunkat használva bemutathatjuk, hogy a gyökér jelen esetben kompozícióban van a fánkkal.

Megoldás videó: https://youtu.be/_mu54BDkqiQ

6.5. Mutató a gyökér

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

Megoldás videó: https://youtu.be/_mu54BDkqiQ

A [Binfa](#) programunk tökéletesen szemlélteti milyen, ha a gyökér csak aggregációban van a fával, hiszen a BinTree osztályunk változói között a gyökeret a csomópont(Node) alosztály mutatójaként deklaráljuk.

```
Node *root;  
}
```

6.6. Mozgató szemantika

Írj az előző programhoz másoló/mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva, a másoló értékadás pedig a másoló konstruktorra! A [Binfa](#) programunk BinTree osztályának Másoló konstruktora:

```
BinTree(const BinTree & old){  
    std::cout<<"BT copy ctor"<<std::endl;  
    root = cp(old.root,old.currentNode);  
}
```

Ehez szükségünk lesz egy rekurzív cp függvényre, hogy másoljuk a régi fánk csomópontjait az új fánkba, melyet az első sorban nullptr-re állítunk.

```
template <typename Vtype>  
Node * BinTree<Vtype>::cp(Node* node ,Node * currentNode)  
{  
    Node * newNode = nullptr;  
  
    if (node)  
    {  
        newNode = new Node(node->getValue());  
        newNode.setleftChild(cp(node->leftChild(),currentNode));  
        newNode.setrightChild(cp(node->rightChild(),currentNode));  
  
        if (node == currentNode){
```

```
        this->currentNode = newNode;
    }
}

return newNode;
}
```

Másoló értékadás: Felhasználjuk a mozgató konstruktort egy ideiglenes tmp fa létrehozására, majd kicseréljük az új fának mutatóját a tmp-re.

```
BinTree &operator=(const BinTree &){
    std::cout<<"BT copy assignment"<<std::endl;
    BinTree tmp{old};
    std::swap(*this,tmp);
    return *this;
}
```

Mozgató konstruktor: Felhasználjuk hozzá a mozgató értékadást. Majd a régi fát átmozgatjuk az újba.

```
BinTree(BinTree && old){
    std::cout<<"BT move ctor"<<std::endl;
    root = nullptr
    *this = std::move(old);
}
```

Mozgató értékadás: A standard névtérben található swap függvényt felhasználva a régi fának a gyökerét, valamint a régi fának az aktuális csomópontra mutató mutatóját kicseréljük az új fáéval.

```
BinTree &operator=(BinTree && old){
    std::cout<<"BT move assignment"<<std::endl;
    std::swap(old.root,root);
    std::swap(old.currentNode,root);

    return *this;
}
}
```

Megoldás videó: <https://youtu.be/QBD3zh5OJ0Y>

Megoldás forrása: ugyanott.

6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7. fejezet

Helló, Conway!

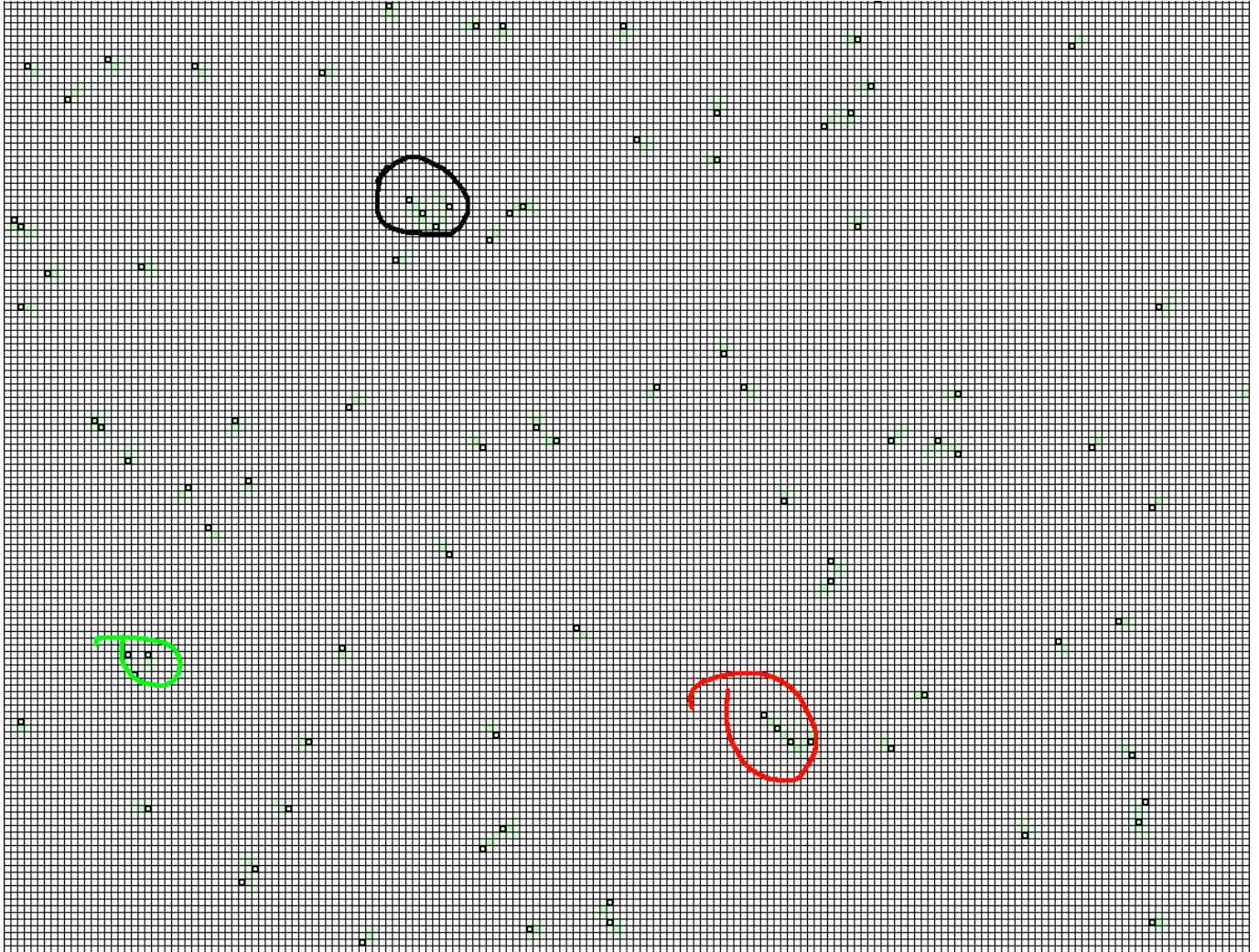
7.1. Hangyaszimulációk

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

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

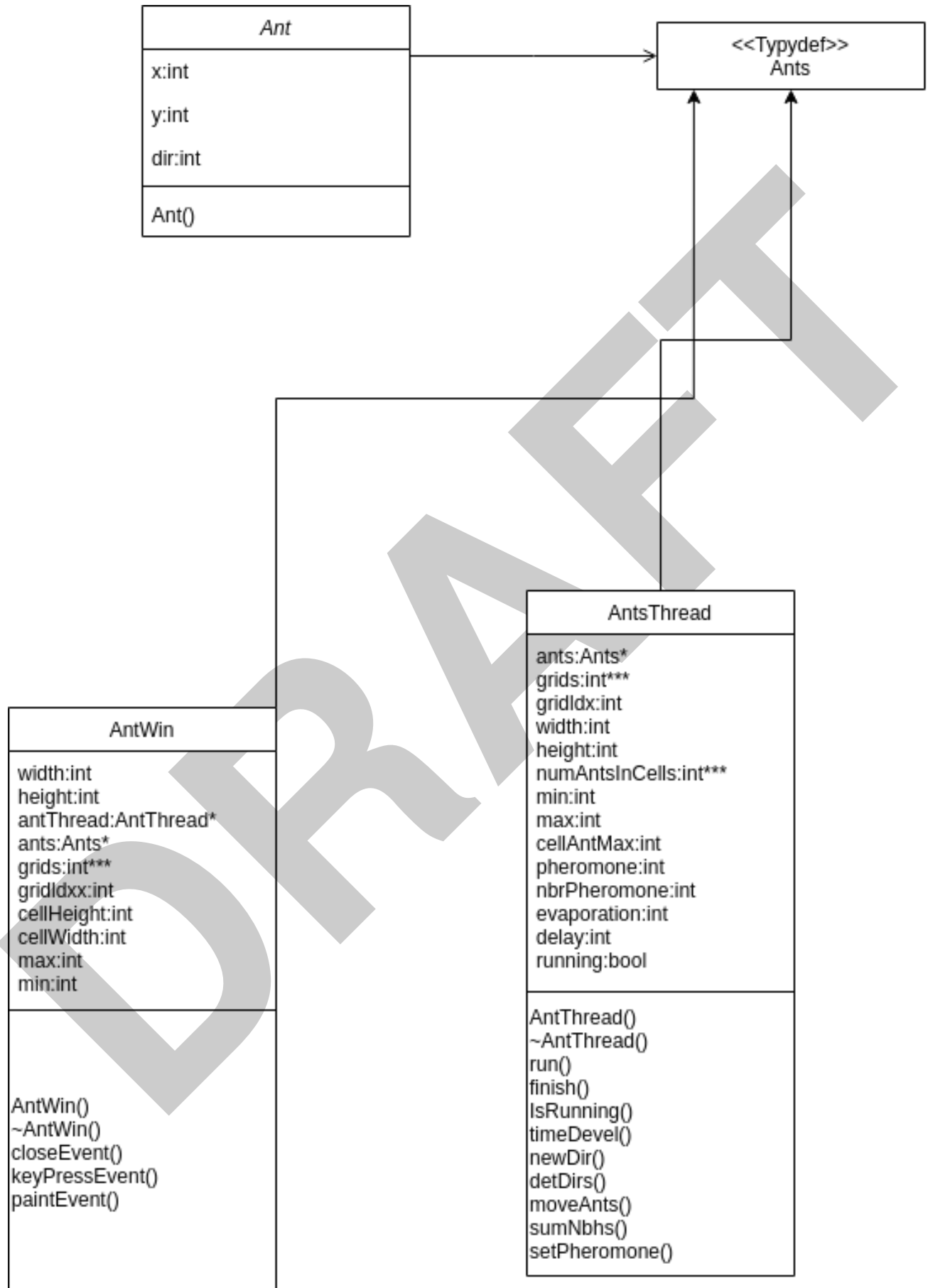
Megoldás forrása: https://gitlab.com/nbatfai/bhax/-/tree/master/attention_raising%2FMyrmecologist

A program működése hangyák mozgásának szimulálásán alapul. Mint tudhatjuk a hangyák feromonokat bocsájtanak ki magukból. Ezen a koncepción elindulva készült el a Hangyaszimuláció is. Azokon az útvonalakon, ahol a hangyák feromont bocsájtottak ki úgymond egy súlyozott útvonal alakul ki.



Láthatjuk, hogy ahol a bekarikázott részek vannak, ott a hangyák a feromon hatására követni kezdték egymást.

[Reverse engineering](#) módszerrel visszafejtve elkészítjük az UML osztálydiagramot.



7.2. Java életjáték

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

Megoldás forrása: [Sejtautomata](#)

A John Horton Conway féle Életjáték szabályai:

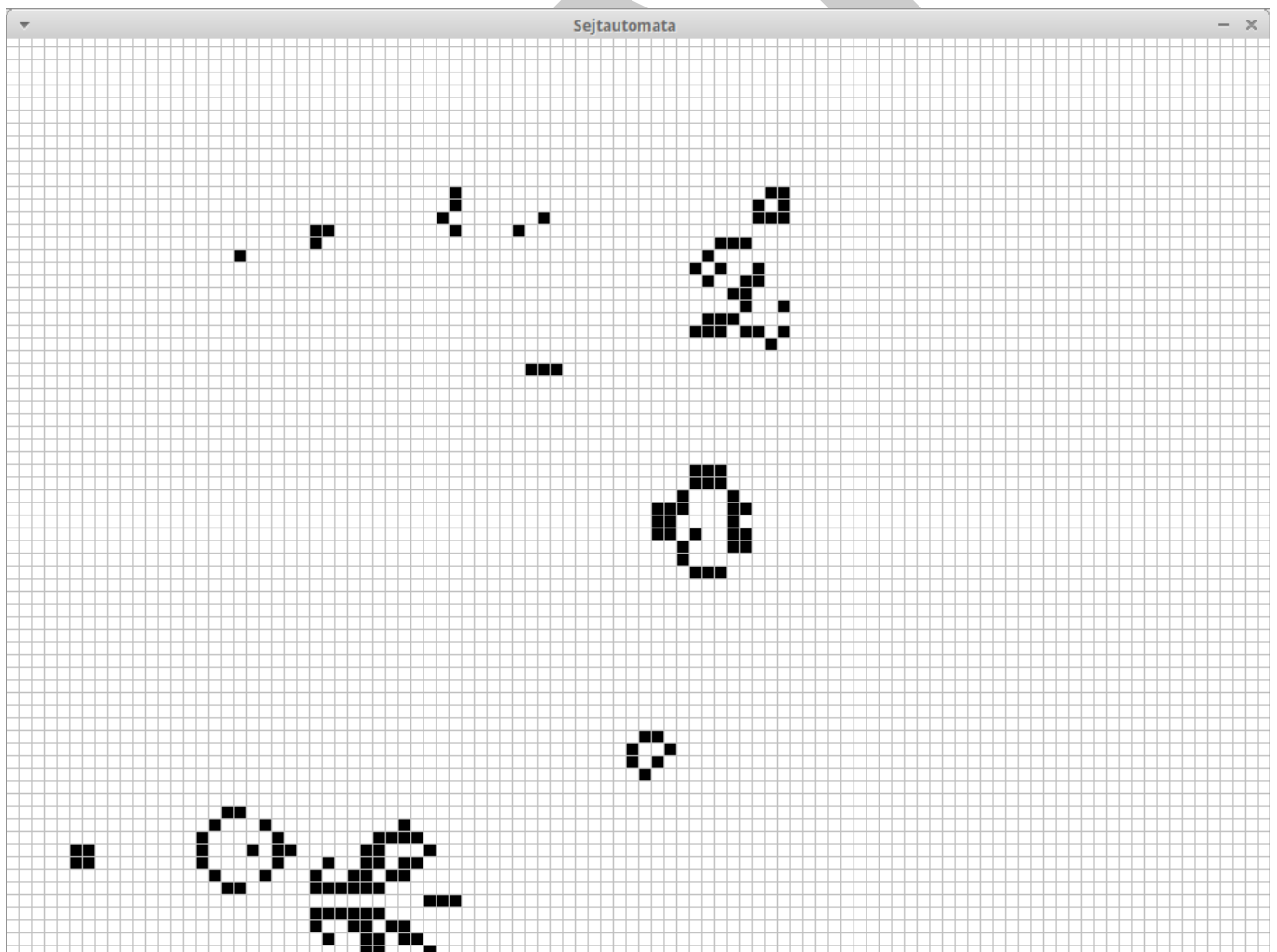
A sejt meghal, ha:

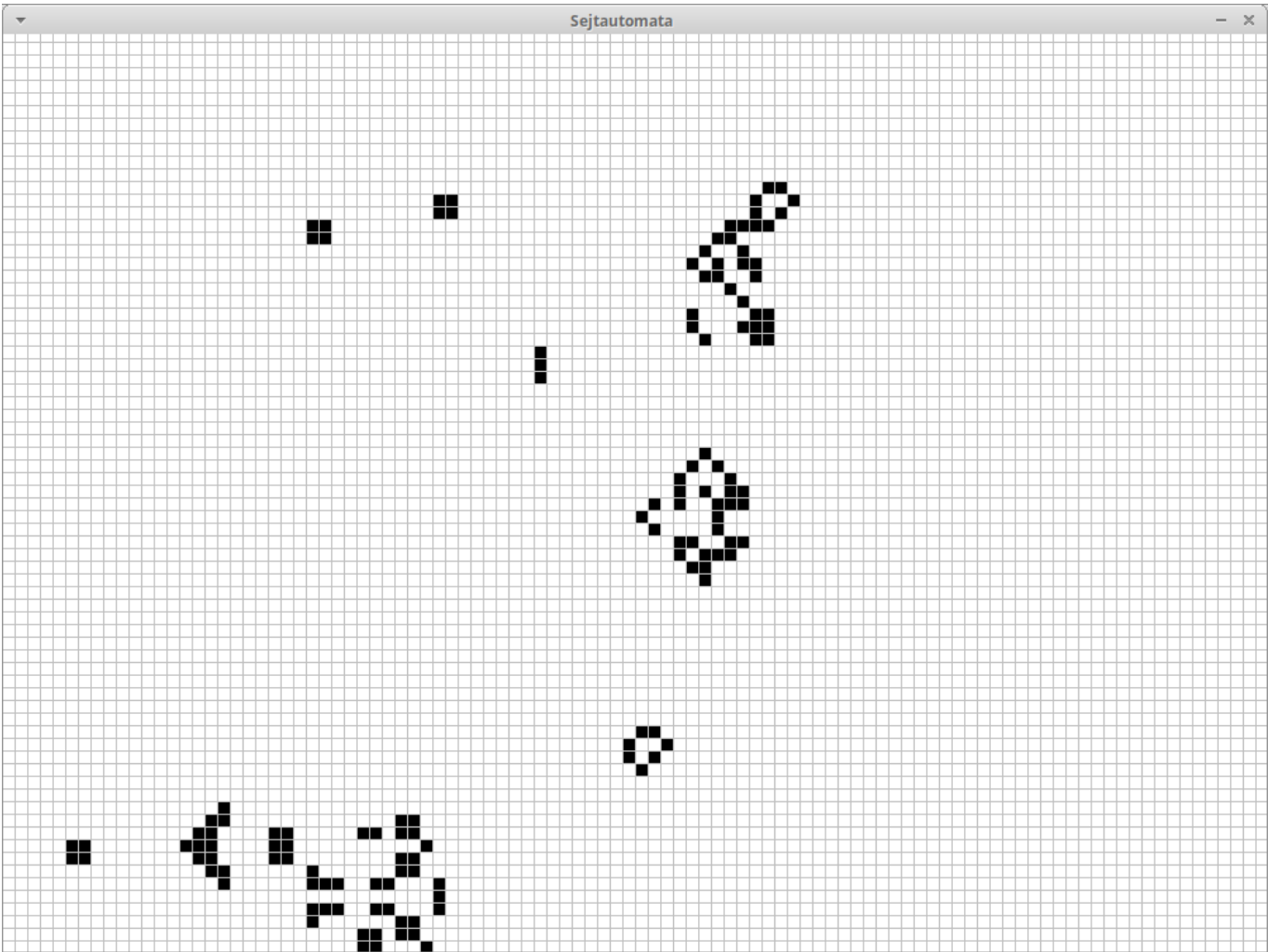
- 3nál több szomszédja van.
- 2nél kevesebb szomszédja van.

A sejt túléli, ha:

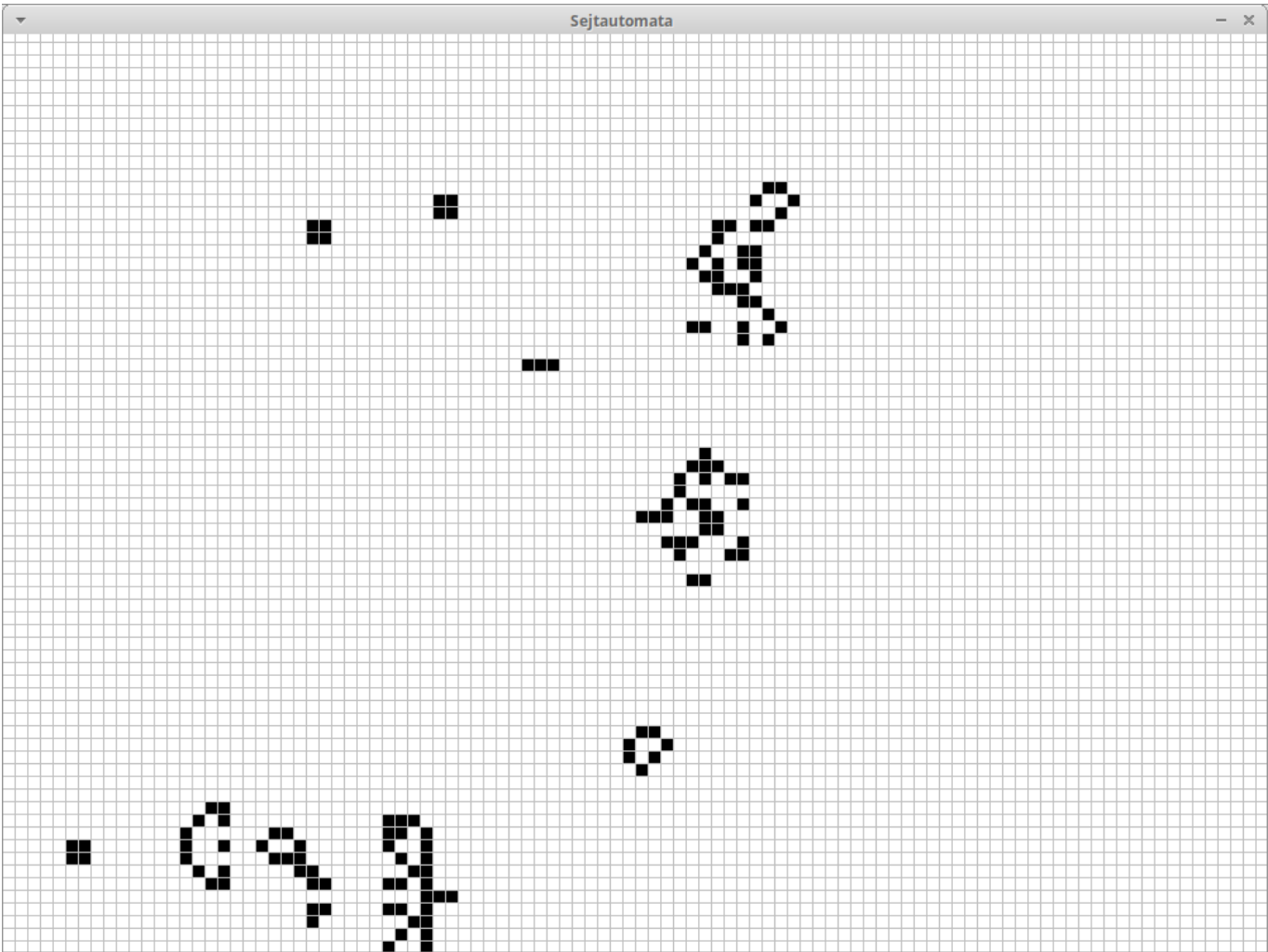
- 2 vagy 3 szomszédja van.

Új sejt születik, ha pontosan 3 sejt veszi körül. Képek a játékról :

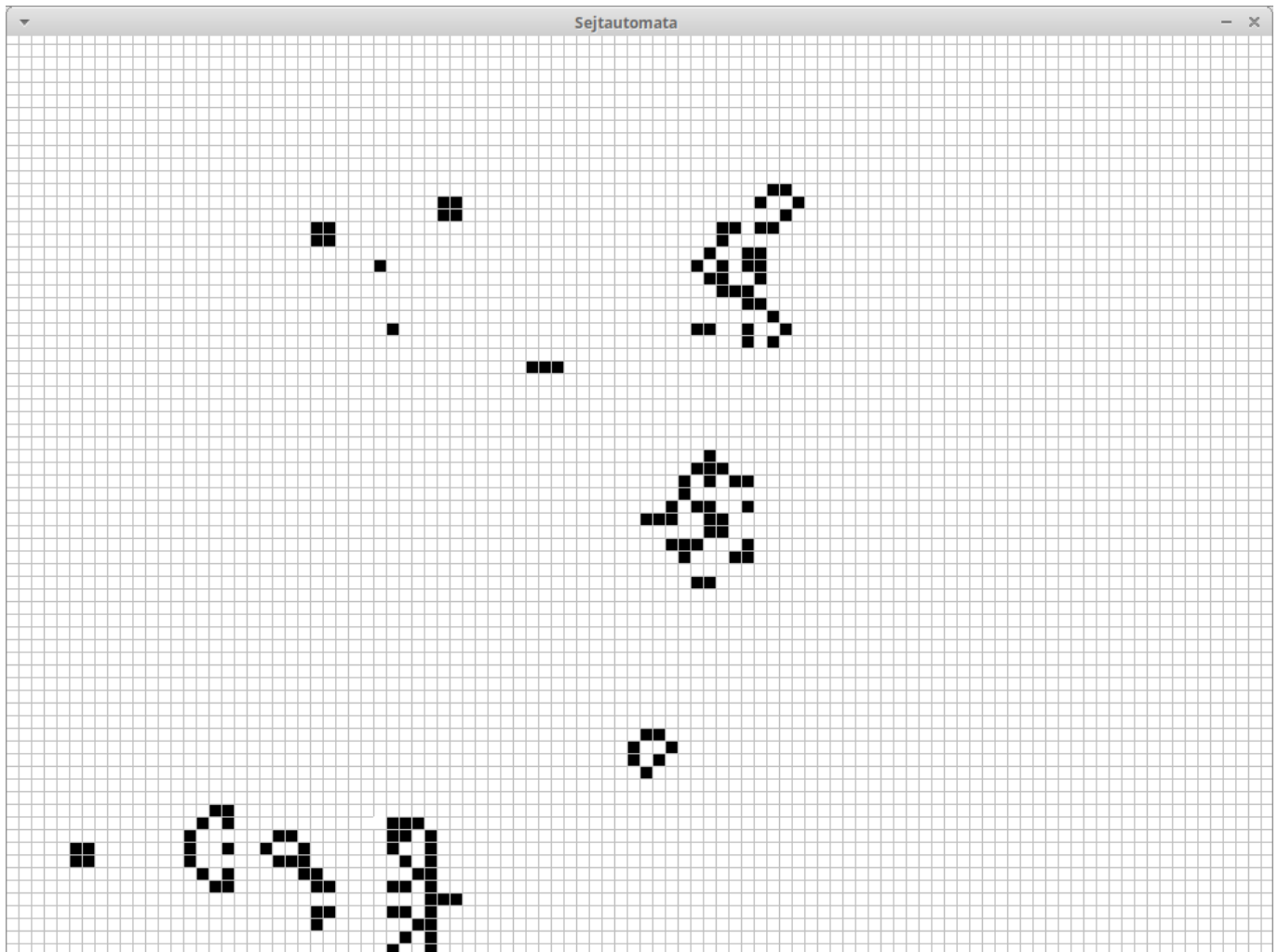




DR



DR



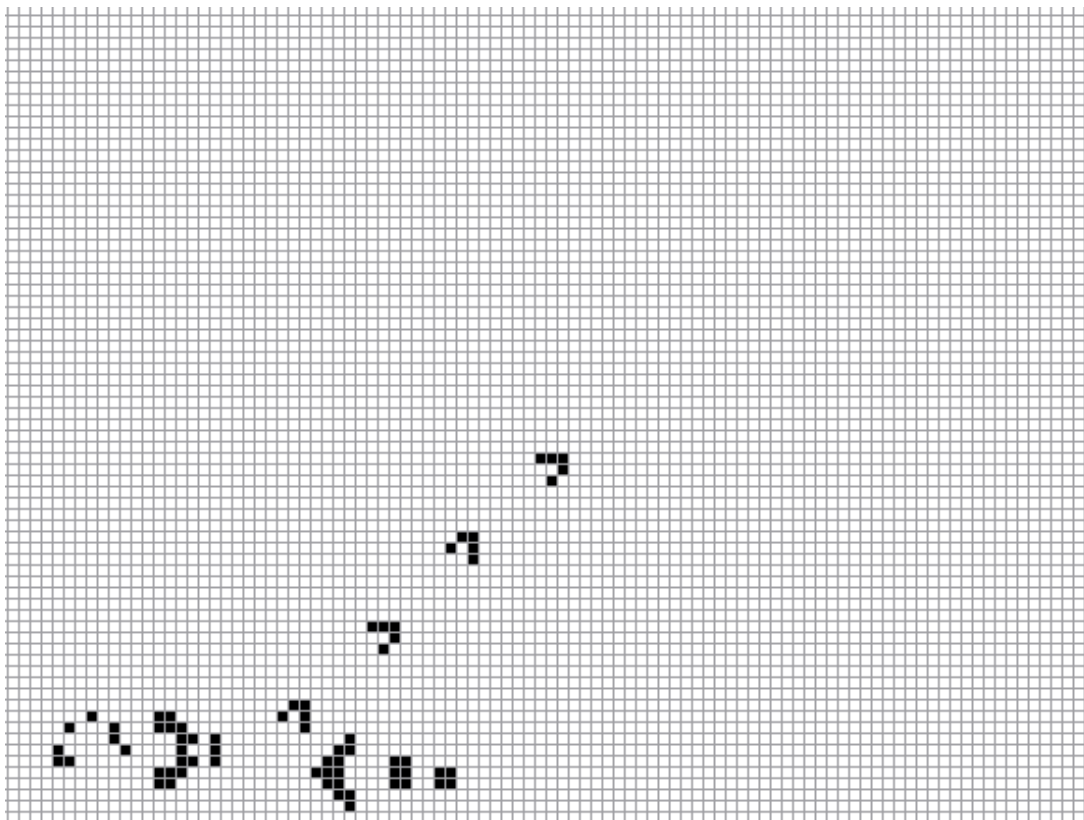
7.3. Qt C++ életjáték

Most Qt C++-ban!

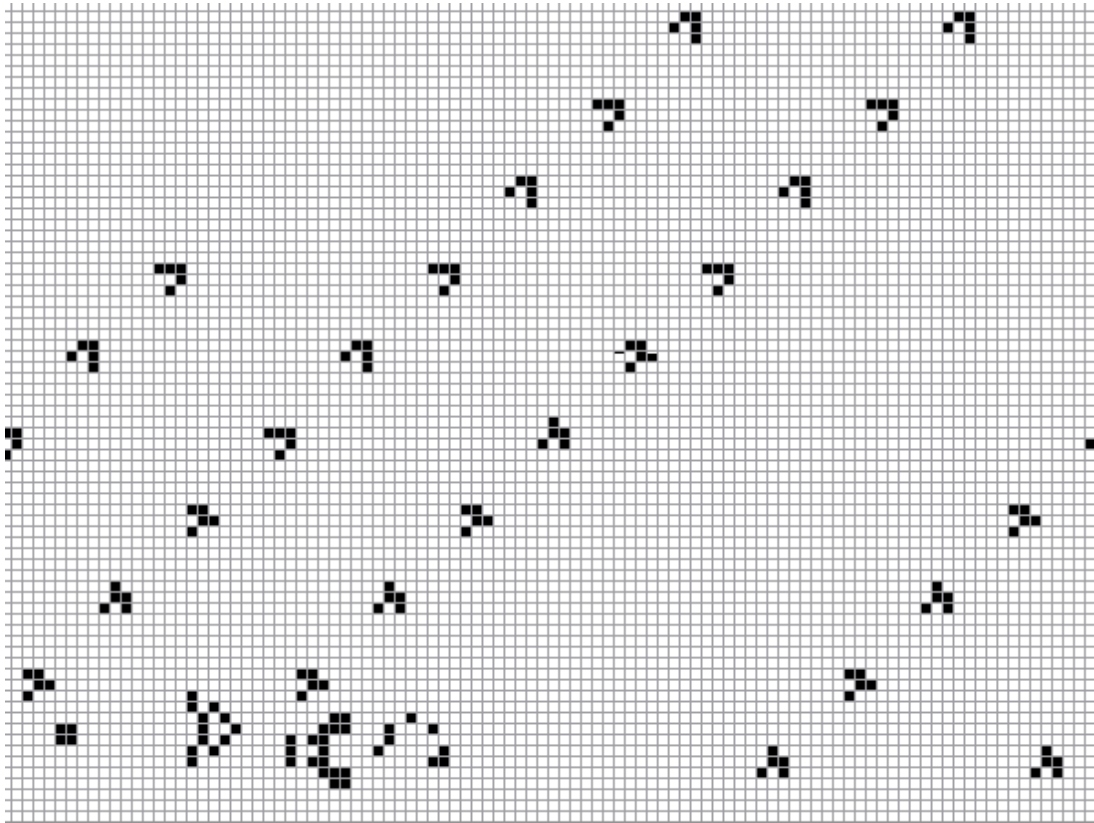
Megoldás forrása: [Sejtautomata_cpp](#)

Megoldás videó: a hivatkozott blogba ágyazva.

Ez a verzió a Java megfelelőjének tökéletes másolata c++-ban. A sikló kilövőről egy pár kép működés közben:



7.1. ábra. Qt c++ szíklókilövő



7.2. ábra. Qt c++ sıklókilövő egy kis idő

7.4. BrainB Benchmark

Megoldás videó: initial hack: <https://www.twitch.tv/videos/139186614>

Megoldás forrása: [BrainB](#)

A BrainB egy Benchmark program amivel bárki lemérheti mekkora potenciál van benne az e-sport terén, illetve, hogy mennyire ügyes bizonyos játékelemekben. További a program készít egy e-sportoló profil az felhasználó képességei alapján.

7.5. Vörös Pipacs Pokol/19 RF

Megoldás videó: <https://youtu.be/VP0kfvRYD1Y>

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás forrása: [progpáter](#)

A TensorFlow használata, egy mai programozó eszköztárában kötelezően benne kell, hogy legyen. Rengeteg olyan mindennapi problémára nyerhetünk megoldást amelyre, sokkal komplexebb programozást igényelne. Ha valaki jól megtanulja az MI programozást, Machine Learninget, tisztába van a regressziós modellekkel, az aktivációs függvényekkel és mindazzal, hogy hogyan működik ez az egész, az valószínűsíthetően jóval értékesebb programozói ismeretekkel fog rendelkezni, mint az aki ezzel nem rendelkezik.

Hogyan is működik ? Neurális hálók segítségével, egy vagy több bemenetet adunk meg, és többnyire 1 kimenetet várunk. Az, hogy a két végpont között mi történik, egy másik kérdés.

C-hez viszonyítva, a TensorFlow HelloWorld-je az MNIST. Azaz kézzel írt számok felismerése. Azonban ez előtt nézzünk valami igazán egyszerűt.

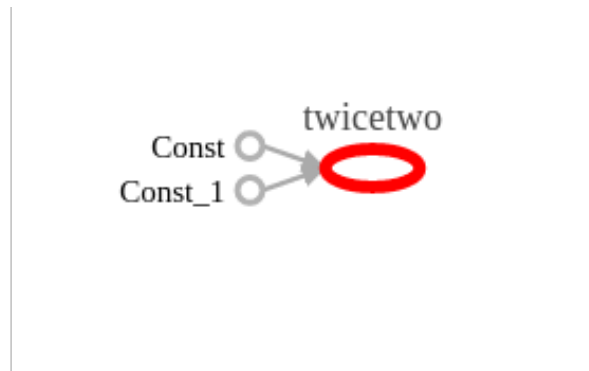
Alábbi programunk, MI segítségével, valamint a tensorflow biztosította eszközökkel számolja ki nekünk $2*2$ értékét.

```
#  
# TensorFlow Hello World 1!  
# twicetwo.py  
#  
import tensorflow  
  
node1 = tensorflow.constant(2)  
node2 = tensorflow.constant(2)  
  
node_twicetwo = tensorflow.mul(node1, node2, name="twicetwo")  
  
sess = tensorflow.Session()  
print sess.run(node_twicetwo)  
  
writer = tensorflow.train.SummaryWriter("/tmp/twicetwo", sess.graph)
```



```
# nbatfai@robopsy:~/Robopsychology/repos/tensorflow/tensorflow/tensorboard$ ↵  
python tensorboard.py --logdir=/tmp/twicetwo  
  
tensorflow.train.write_graph(sess.graph_def, "models/", "twicetwo.pb", ↵  
as_text
```

A számítási gráfja ennek a programnak :



8.2. Mély MNIST

Most már valóban megérkeztünk a Mesterséges Intelligencia programozásának VALÓDI Hello World-jéhez, ami nem más mint az MNIST. Az MNIST egy Keras dataset, mely kézzel írt számokat tartalmaz. Érdekessége, hogy felcímkézett számokat tartalmaz, ezek alapján tudjuk majd azonosítani, majd esetlegesen korrigálni a hibákat, melyeket rosszul ismert fel a gép. A TensorFlow oldalán minden megtalálható. <https://www.tensorflow.org/tutorials>.

```
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(512, activation=tf.nn.relu),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
)  
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])  
model.fit(x_train, y_train, epochs=5)  
model.evaluate(x_test, y_test)
```

Ha futtatjuk a konkrétan TensorFlowrol első tutorialját a következőt látjuk :

```
$ ./mi.py
Epoch 1/5
60000/60000 [=====] - 13s 224us/sample - loss: ↩
0.2020 - acc: 0.8740
Epoch 2/5
60000/60000 [=====] - 13s 224us/sample - loss: ↩
0.0987 - acc: 0.8903
Epoch 3/5
60000/60000 [=====] - 14s 226us/sample - loss: ↩
0.0647 - acc: 0.9031
Epoch 4/5
60000/60000 [=====] - 13s 224us/sample - loss: ↩
0.0598 - acc: 0.9702
Epoch 5/5
60000/60000 [=====] - 13s 224us/sample - loss: ↩
0.0482 - acc: 0.9853
10000/10000 [=====] - 1s 74us/sample - loss: ↩
0.0475 - acc: 0.9844
```

Az 5. Epoch-unk végére a majdnem 99%-os pontossággal meg tudja mondani az általunk tanított mesterséges intelligencia, hogy milyen számot lát a képen. Látható, hogy 60000 képet használunk a tanításhoz, 10000-et pedig az ellenőrzéshez. A modellünk 4 rétegű neurális hálóból áll. Az első réteg amely a 2D-s numpy tömbjeinkből 1D-set csinál az az $28 \times 28 = 784$ px. A következő "Dense" megnevezésű rétegek, "dense" azaz sűrűn kötött rétegek. Az első ilyen rétegünk 128 "neuron"-ból áll. Az utolsó dense réteg pedig egy softmax réteg alacsony szintű, kezdő TF-es projektekhez szokták használni főként. A Softmax nem csinál mást mint, visszaad egy 10 valószínűségi értékből álló tömböt. Amelyiknél a legnagyobb érték szerepel, az lesz a választott szám.

8.3. Minecraft-MALMÖ

A Minecraft-MALMÖ egy programozható ágens kiegészítő a Minecraft játékhoz. A projekt GitHub oldala a következő címen érhető el: <https://github.com/Microsoft/malmo>. Ezen a címen találunk példaprogramokat is, valamint remek dokumentációt.

A Microsoft Minecraft-Malmö nevű projektje, lehetővé teszi, hogy a jól ismert Steve figuránkat most ne billentyűzet/egér segítségével irányítsuk, hanem előre megírt kóddal.

A következőket tudjuk irányítani az `agent.sendCommand("")` paranccsal:

- ugrás
- mozgás, előre hátra
- fordulás
- strafelés
- kamera vertikális szöge
- guggolás

- attack

Ami hatalmas segítséget nyújthat, az a Grid, melyet a következőképpen hozunk létre :

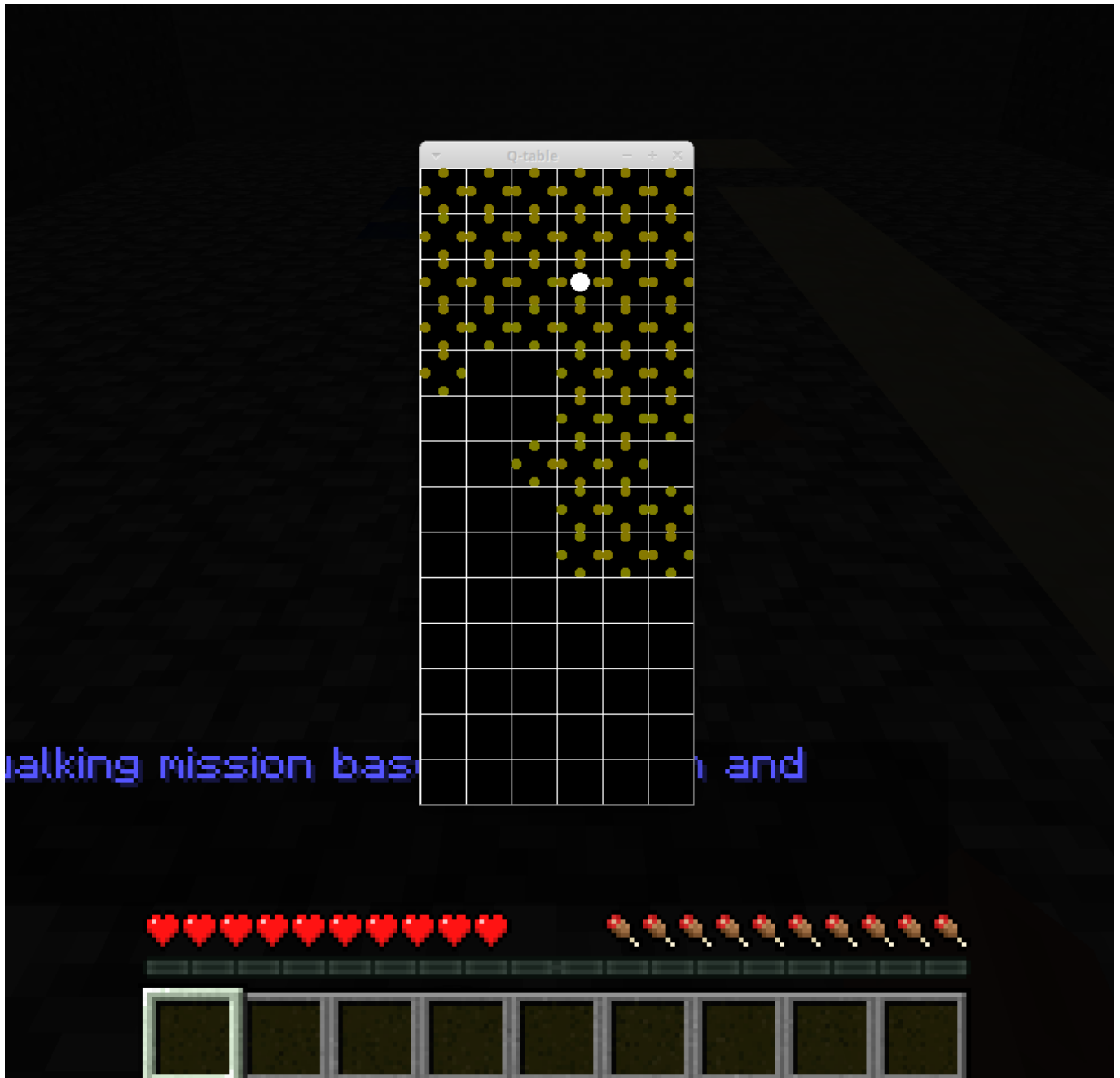
```
<ObservationFromGrid>
<Grid name="floor3x3">
<min x="-1" y="-1" z="-1"/>
<max x="1" y="1" z="1"/>
</Grid>
</ObservationFromGrid>
```

Ez a grid egy 3*3-as blokkötényt figyel a karakter lába alatt. Ezzel a megfigyeléssel kiküszöbölhetjük, hogy pl. a karakterünk lábába essen, vagy esetleg kimásszon a vízből.

```
if world_state.number_of_observations_since_last_state > 0:
msg = world_state.observations[-1].text
observations = json.loads(msg)
grid = observations.get(u'floor3x3', 0)
print(grid)
if grid[13] != "air":
print("valami utban van")
agent_host.sendCommand("jump 1")
time.sleep(1)
agent_host.sendCommand("jump 0")
```

A fenti kódcsipetben látható a floor3x3-nevű gridünk, ami már nem csak a Steve lába alatti területet, hanem a körülötte lévő blokkokat reprezentálja. Így vizsgálható, hogy a karakter beakadt-e valamibe.

A Malmö nem véletlenül került az AI/ML-es témakörbe. Az api lehetővé teszi, hogy az ágensünket megjutalmazzuk, ha eljut bizonyos pontokra, akár egy pályán keresztül, melynek szélei lava blokkok. A jutalmazás lényege, hogy ha Steve lava-ba lép -200 at vonunk le tőle, ha a lapis blokkra lép +200-at kap, valamint minden egyes lépés -1-nek számít.



A Q-table az eddig megtett lépéseket mutatja, melyik blokkra lépett eddig Steve. Az eredeti tutorialban a lava-ba esés-t a blokk oldalán lévő pont pirosra váltásával szemléltetjük. Ha eléri a lapis blokkot, a blokk azon oldalán lévő pont ahol a kék blokk volt, zöldre változik, indikálva, hogy az errefele történő mozgás jó ötlet.

Felfedezhető a kódban az alábbi részlet:

```
self.epsilon = 0.01
```

Ez azt jelenti, hogy 1%-ban inkább random útvonalat választ, mint, hogy a legjobbat választaná. Ezt úgy nevezzük *Epsilon-Greedy policy*.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása: ugyanott.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

A GIMP nevű szofver a PS helyettesítője sok esetben. Szerencsénkre a GIMP lehetővé teszi, hogy a saját scriptünket használjuk a programon belül. Ehez a Scheme formátumot az az .scm -et fogjuk használni. A scriptünket bemásoljuk a GIMP scripteket tároló mappájába. Íme a kívánt eredmény:



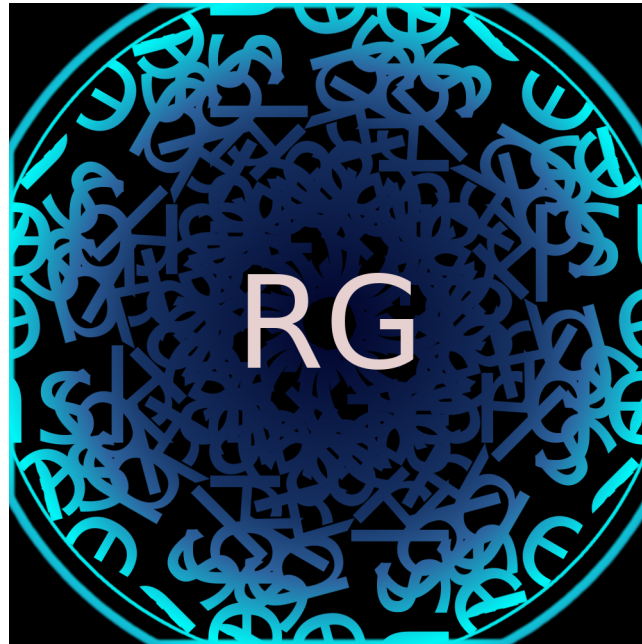
9.3. Gimp Scheme Script-fu: név mandala

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

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

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

Hasonlóan a fenti Króm-effekthez szintén egy scriptet fogunk használni ahhoz, hogy 2 beírt szövegből egy mandalát készítsünk. A gimp megnyitása után a Create>BHAX>Mandala úton csinálhatjuk meg az alábbi mandalát:



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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

Rövid bevezető: A számítógépek programozásának alvető 3 nyelv típusa:

- gépi nyelv
- assembly szintű nyelv
- magas szintű nyelv Ezen a kurzuson a magas szintű nyelvekkel foglalkozunk. A magass szintű nyelveket forrásszövegnek is nevezzük. A forrássövegek alapvető szabálytípusai:
 - szintaktikai
 - szemantikai Ezen szabályok együttese határoz meg egy magas szintű programozási nyelvet.

Ahoz, hogy a számítógép, valamint főként a processzor értelmezni tudja az általunk írt nyelvet, úgy mond le kell fordítanunk gépi nyelvre. Erre két fajta technika létezik:

- fordítóprogramos
- interpreteres A fordítóprogram egy magas szintű nyelvben megírt forrásprogramból, úgy nevezett tárgy-programot állít elő, amely a következő lépéseket hajtja végre:
 - lexikális elemzés
 - szintaktikai elemzés
 - szemantikai elemzés
 - kódgenerálás

Karakterkészlet:

A programok forráskódjának legkisebb alkotóelemei a karakterek. A karakterkészlet minden nyelvnél alapvető és nyelvenként különböző lehet. Ezekből összeállíthatunk bonyolultabb nyelvi elemeket (eljárásorientált nyelveknél):

- lexikális egységek
- szintaktikai egységek

- utasítások
- programegységek
- fordítási egységek
- program

Minden nyelvben hasonló a karakterek kategorizálása:

- betűk (az angol ABC 26 betűje)
- számjegyek
- egyéb karakterek (elhatároló jelek: (pl. [,], ., :, { , }, ' , " , ;), műveleti jelek : (+-*/), írásjelek : (?!) stb.)

Lexikális egységek:

A program szövegének azon elemei, melyeket a fordító a lexikális elemzés során felismer, tokenizál, fajtái a következők:

- többkarakteres szimbólum
- szimbolikus név
- címke
- megjegyzés
- literál

10.2. C programozás bevezetés

Rövid olvasónapló a [KERNIGHANRITCHIE] könyvről.

2. fejezet:

A 2. fejezetben a típusokról, változókról, állandókról olvashatunk. Adattípusok: char, int, float, double. Az int típus long, long long, short előtagjáról is szó esik. Megemlíti az unsigned változók használatát is. Olvashatunk továbbá a '\ ' operátor felhasználásával alkotott állandókról. A deklarációk pontos szintaktikájáról folyamatáról, aritmetikai operátorok (+-*/%) sajátosságairól, logikai operátorok használatáról, típuskonverzióról (castolás), in-dekrementáló operátorokról, maszkoláshoz használatos bitművelet operátorokról, értékadó operátorokról, feltételekről, precedenciákról is többet tudhatunk meg.

3. fejezet:

Tartalma a vezérlési szerkezetek:

- Utasítások (; jelentősége, { })
- if, else utasítás

- if,else if használata
- switch utasítás
- ciklusok (for,while,do-while)
- break, continue utasítások

4. fejezet:

Tartalma a függvényes és a program szerkezete:

- Függvények alapfogalmai:paraméter,visszatérési érték..
- Különböző, nem egészen visszatérő függvények
- Külső változók
- Érvényességi tartomány
- Header file-ok
- static használata(függvény saját érték esetén)
- register használata(gyakran használt változó)
- Rekurzió függvényekben
- C előfeldolgozás(#include,#define)
- Makrók,feltételes fordítás

10.3. C++ programozás

Ez a fejezet főként a C++, C-hez mért változásairól és úgy amblokk a C++-alap fogalmairól és újdonságairól. A fejezet kódcsipeteit csak c++ fordítóval tudjuk fordítani. Első különbség a C-től eltérően a c++-ban egy függvény üres paraméterlistáját (void)-al jelöljük, míg a C-ben csak üresen kellett hagyni. Ha tetszőleges számú paramétert akarunk megadni akkor a (...) szintaktikát kell alkalmaznunk. Visszatérési típusoknál a következő a helyzet a C-t és a C++-t illetően: az f(){ //függvénytörzs } függvény C-ben int típusal térne vissza, azonban C++-ban hibát kapunk, mert itt már nincs lehetőségünk visszatérési típus nélküli függvény deklarálunk. C++ main függvénye kétféle képpen deklarálható : argc,argv argumentumokkal, vagy azok nélkül. Érdekeség, hogy nincs szükség a return 0;-ra, mert sikeres futás esetén a main függvényünk alapve-tően 0-val tér vissza. Meg kell említenünk mint ultimate feature-t : bool típus! A C-ben nem használhatunk bool típust, azonban a C++-ban már megtehetjük ezt. A bool azaz boolean típusú változó, igaz, vagy hamis értékkel inicializálható. Létezik automatikus konverzió az int és a bool között : 0-false 1-true. Változódek-laráció, mint utasítás : ami azt jelenti, hogy bárhol állhat változó, ahol állhat utasítás. Ez lehetővé teszi, hogy akár ideiglenes változót akkor deklaráljunk amikor szükség van rá, így az általa lefoglalt memória nem lesz felhasználva, ameddig arra feltétlen szükségünk nincsen.

10.4. Python nyelvi bevezetés

A Python nyelvi sajátosságai, további saját kiegészítések :

- Deklaráció során nincs típus deklaráció.
- A for ciklus szintaktikája teljesen más. A C# foreach-hez hasonló.
- Szintaktia, kódcsopontosítás egyszerű tagolással: (" { }", ";" nincsenek kódcsopontosításra. Helyette space, kettőspontus, tab.)
- A Python interpreter használ.
- Az állításokat azonos szintű behúzásokkal tudjuk csoportosítani.
- Komment : '#'
- Az if-else, ciklusok, függvények deklarációja után közvetlen : -ot használunk
- Nincs típus -> nincs függvény visszatérési érték deklaráció sem.

III. rész

Második felvonás

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Berners-Lee!

11.1. Olvasónapló:

DRAFT

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

[Polártranszformációs normális szám generátor Java nyelven.](#) . Ezen algoritmus megvalósításához szükségünk lesz egy logikai változóra amely megmondja, hogy páros, vagy páratlanadik lépésben hívtuk-e meg a következő() függvényt. Ha igaz akkor a "tárolt" nevezetű double változóban eltároltuk a számot. A megoldás mondható jónak, hiszen a hivatalos JDK osztályok között a Random.java-val összevetve kísértetes hasonlóságot vélünk felfedezni.

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

Megoldás forrása: <https://regi.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apas03.html>

12.2. Homokozó

Ebben a feladatban a már meglévő LZW binfát írjuk át Java nyelvre, úgy, hogy funkcionalitásában ne változzon. Amint már a gyakorlati órán is tárgyaltunk, a pointerek/referenciák nélkül gyakorlatilag működik

is.

`./Arroway/LZWBinFa.java`

12.3. Gagyi

Az ismert formális `while (x <= t && x >= t && t != x);` tesztkérdéstípusra adj a szokásosnál (miszerint `x`, `t` az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) "mélyebb" választ, írd 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!

Ha belegondolunk, hogy mi lehet a vége ennek a programnak, tévesen azt gondolhatjuk, hogy nincs olyan szám amit `x` és `t` helyére beírva ne végtelen ciklust kapnánk. Miért?

A magyarázat a jelenségre, hogy a Java nyelvben az `Integer` osztály `equals` függvénye nem egyformán dolgozik a `==` operátorral. Míg az operátor azt nézi, hogy a jobb és bal oldalak referenciája egyenlő-e, az `equals` függvény az értékeket hasonlítja össze, az osztály `valueOf` függvényének segítségével.

A lényeg az, hogy bármilyen 128-nál kisebb értékre a program megáll, bármilyen 127-nél nagyobb értékre pedig végtelen ciklusba kerül.

Ennek az oka pedig az, hogy a Java ezeket a nullához relatíve közel lévő számokat egy afféle gyorsítótárban tárolja, poolozza.

A következő kódcsipet a JDK7 forrásából származik.

```
public static Integer valueOf(int i) {  
    assert IntegerCache.high >= 127;  
  
    if (i >= IntegerCache.low && i <= IntegerCache.high)  
        return IntegerCache.cache[i + (-IntegerCache.low)];  
  
    return new Integer(i);  
}
```

A Java, a jobb teljesítmény és alacsonyabb futásidő érdekében ezekhez a kicsi számokhoz nem új objektumokat hoz létre a Java, hanem az `IntegerCache`-ből szed ki egy megfelelő számot.

`./Arroway/Gagyi.java`

12.4. Yoda

Írjunk olyan Java programot, ami `java.lang.nullpointerex`-el leáll, ha nem követjük a yoda conditions-t!
https://en.wikipedia.org/wiki/yoda_conditions

Főként kezdő programozók hibája, vagy esetleg typoból származó szintaktikai hiba, az if fejében való értékadás =, az összehasonlítás == helyett. A Yoda conditions erre a gyakori hibára ad megoldást, ugyanis az így írt kódunk:

```
class Yoda {  
    public static void main(String[] args) {  
        int num = 42;  
        if(42 = num) { /* NullPointerException */ }  
    }  
}
```

fordítási időben java.lang.NullPointerException-el áll le.

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

A Liskov helyettesítési elv nem mást mond ki, mint, hogy minden osztály legyen helyettesíthető a gyerek osztályával anélkül, hogy bármilyen működésbeli differencia lépne fel. A Liskov elv része a clean code egyik szabályrendszerének is a S.O.L.I.D elveknek.

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

// ez a két osztály alkotja a "P programot" az LPS-ben
class Program {
public:
    void fgv ( Madar &madar ) {
        madar.repul();
    }
};

// itt jönnek az LSP-s S osztályok
class Sas : public Madar
{};

class Pingvin : public Madar // ezt úgy is lehet/kell olvasni, ↔
    hogy a pingvin tud repülni
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Sas sas;
```

```
program.fgv ( sas );

Pingvin pingvin;
program.fgv ( pingvin ); // sérül az LSP, mert a P::fgv ←
    röptetné a Pingvint, ami ugye lehetetlen.

}
```

Az alábbi [programcsipet](#) a Liskov elvet sértő példa. A példából jól látszik, hogy a Pingvin ugyanúgy a Madár alosztálya, melynek létezik repül metódusa. Nyilvánvalóan a Pingvin nem tud repülni, így az öröklött `repul()` függvénynek nem kellene meghívhatónak lennie a Pingvin alosztályból.

Erre egy valid megoldás, ha a Madár főosztálynak készítünk egy RepülőMadár alosztályt, és a repül metódust, majd abba inicializáljuk, így a Pingvin tud majd a Madár alosztálya lenni, ezáltal nem fog tudni repülni, azonban mégis lesz olyan madár amely képes lesz erre.

13.2. Szülő-gyerek

```
#include <iostream>
#include <string>

class Szulo {
public:
    void print(std::string s) {
        std::cout << s << std::endl;
    }
};

class Gyermek : public Szulo {
public:
    void kiir(std::string s) {
        std::cout << s << s << std::endl;
    }
};

int main(int argc, char **argv) {
    Szulo* szulo = new Szulo();
    Szulo* szulo2 = new Gyermek();

    // Ez az, ami nem fog mukodni:
    szulo2->kiir("Uzenet");

    return 0;
}
```

Amennyiben megpróbáljuk fordítani a fenti C++ kódot, a fordító a következő hibával kilép:

```
$ g++ ParentChild.cc -o /dev/null
      ParentChild.cc: In function 'int main(int, char**)':
ParentChild.cc:23:13: error: 'class Szulo' has no member named 'kiir'
   23 |         szulo2->kiir("Uzenet");
      |                   ^~~~
```

Hasonló a helyzet Javában is:

```
class Szulo {
public void print(String s) {
    System.out.println(s);
}

class Gyermeke extends Szulo {
    public void kiir(String s) {
        System.out.println(s);
        System.out.println(s);
    }
}

public class ParentChild {
    public static void main(String[] args) {
        Szulo sz = new Szulo();
        Szulo sz2 = new Gyermeke();

        // Ez az, ami nem megy
        sz2.kiir("Uzenet");
    }
}
```

```
$ java ParentChild.java
      ParentChild.java:20: error: cannot find symbol          sz2.kiir(" ←
      Uzenet");
      ^
      symbol:   method kiir(String)
      location: variable sz2 of type Szulo1 error
error: compilation failed
```

13.3. Hello, Android!

Élesszük fel az [SMNIST for Humans](#) projektet!

Érdekes módon a mai natív androidos appok nagy része, áll sok XML fájlból ami a UI-ért felelős és áll Kotlin/Java osztályokból melyek a UI működésért felelősek. Az SMNIST-ben azonban a UI elemek nagy részét egy Java osztályban hozunk létre, számolunk ki.

A forráskódot saját kitalált feladatként kicsit updateltem, átírtam Kotlin nyelvre, valamint kijavítottam a deprecated függvényeket és updateltem a verziót.

13.4. Ciklomatikus komplexitás

A Ciklomatikus komplexitás egy program gráfelméletre alapuló, a forráskódban az elágazó gráfok pontjai és a köztük lévő élek száma alapján, számolandó. A végeredmény egy pozitív egész szám lesz. Én a fenti Szülő-Gyermek programot választottam. A Ciklomatikus komplexitás kiszámolására számos fejlesztői környezet plug-in van, azonban én az egyszerűség kedvéért egy online-t használtam.

Try Lizard in Your Browser

.java

Analyse

```
public class ParentChild {
    public static void main(String[] args) {
        Szulo sz = new Szulo();
        Szulo sz2 = new Gyermek();

        // Ez az, ami nem megy
        sz2.kiir("Uzenet");
    }
}
```

Code analyzed successfully.

File Type

.java

Token Count

95

NLOC

18

Function Name	NLOC	Complexity	Token #	Parameter #
Szulo::print	3	1	16	
Gyermek::kiir	4	1	25	
ParentChild::main	5	1	32	

13.1. ábra. Ciklomatikus komplexitás

IV. rész

Irodalomjegyzék

DRAFT

13.5. Általános

- [MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.
- [PICI] Juhász, István, *Magas szintű programozási nyelvek I.*
- [SMNIST] Norbert Bátfai, Dávid Papp, Gergő Bogacsovics, Máté Szabó, Viktor Szilárd Simkó, Mórió Bersenszki, Gergely Szabó, Lajos Kovács, Ferencz Kovács, and Erik Szilveszter Varga, *Object file system software experiments about the notion of number in humans and machines*, Cognition, Brain, Behavior. An Interdisciplinary Journal , DOI 10.24193/cbb.2019.23.15 , 2019.

13.6. C

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

13.7. C++

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

13.8. Python

- [BMEPY] Ekler, Péter, Forstner, Bertalan, És Kelényi, Imre, *Bevezetés a mobilprogramozásba - Gyors prototípusfejlesztés Python és Java nyelven*, Bp., Szak Kiadó, 2008.

13.9. Lisp

- [METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemepor>, az UDPROG 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 érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPROG 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 érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.