

Univerzális programozás

Így neveld a programozód!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

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, Bátfai, Margaréta, Ács Nagy, Gergely	2020. április 3.	

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

Ajánlás

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

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

DRAFT

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	8
2.3. Változók értékének felcserélése	10
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	13
2.6. Helló, Google!	16
2.7. A Monty Hall probléma	18
2.8. 100 éves a Brun téTEL	20
2.9. MALMO Csiga	24
3. Helló, Chomsky!	25
3.1. Decimálisból unárisba átváltó Turing gép	25
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	26
3.3. Hivatalos nyelv	26
3.4. Saját lexikális elemző	27
3.5. Leetspeak	28

3.6. A források olvasása	30
3.7. Logikus	31
3.8. Deklaráció	32
3.9. MALMO Diszkret	35
4. Helló, Caesar!	36
4.1. double ** háromszögmátrix	36
4.2. C EXOR titkosító	38
4.3. Java EXOR titkosító	39
4.4. C EXOR törő	41
4.5. Neurális OR, AND és EXOR kapu	43
4.6. Hiba-visszaterjesztéses perceptron	45
4.7. MALMO	46
5. Helló, Mandelbrot!	47
5.1. A Mandelbrot halmaz	47
5.2. A Mandelbrot halmaz a std::complex osztállyal	48
5.3. Biomorfok	52
5.4. A Mandelbrot halmaz CUDA megvalósítása	57
5.5. Mandelbrot nagyító és utazó C++ nyelven	59
5.6. Mandelbrot nagyító és utazó Java nyelven	64
5.7. MALMO	68
6. Helló, Welch!	69
6.1. Első osztályom	69
6.2. LZW	69
6.3. Fabejárás	69
6.4. Tag a gyökér	69
6.5. Mutató a gyökér	70
6.6. Mozgató szemantika	70
7. Helló, Conway!	71
7.1. Hangyszimulációk	71
7.2. Java életjáték	71
7.3. Qt C++ életjáték	71
7.4. BrainB Benchmark	72

8. Helló, Schwarzenegger!	73
8.1. Szoftmax Py MNIST	73
8.2. Mély MNIST	73
8.3. Minecraft-MALMÖ	73
9. Helló, Chaitin!	74
9.1. Iteratív és rekurzív faktoriális Lisp-ben	74
9.2. Gimp Scheme Script-fu: króm effekt	74
9.3. Gimp Scheme Script-fu: név mandala	74
10. Helló, Gutenberg!	75
10.1. Programzási alapfogalmak (PICI)	75
10.2. Programozás bevezetés (Keringhan)	77
10.3. Programozás (BME C++)	77
10.3.1. Mobilprogramozás (Python könyv)	78
III. Második felvonás	80
11. Helló, Arroway!	82
11.1. A BPP algoritmus Java megvalósítása	82
11.2. Java osztályok a Pi-ben	82
IV. Irodalomjegyzék	83
11.3. Általános	84
11.4. C	84
11.5. C++	84
11.6. Lisp	84

Ábrák jegyzéke

2.1. BogoMIPS futtatásának eredménye a MacBookomon	15
2.2. BogoMIPS futtatásának eredménye a Windows PC-men	16
2.3. A B_2 konstans közelítése	24
4.1. A double ** háromszögmátrix a memóriában	38
5.1. A kirajzolt Mandelbrot halmaz	48
5.2. A kirajzolt complex Mandelbrot halmaz	52
5.3. A kirajzolt Biomorf	57

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

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

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 kódcsipek! 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 [**BMECPP**] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.
- Kódjátszma, <https://www.imdb.com/title/tt2084970>, benne a **kódtörő feladat** élménye.

- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

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: [YouTube](#)

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c, bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-w.c.

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

Egy mag 100 százalékban:

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

    return 0;
}
```

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

```
#include <stdbool.h>

int main ()
{
    while(true);

    return 0;
```

```
}
```

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

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

```
$ gcc -S -o infty-f.S infty-f.c
$ gcc -S -o infty-w.S infty-w.c
$ diff infty-w.S infty-f.S
1c1
<   .file "infty-w.c"
---
>   .file "infty-f.c"
```

Egy mag 0 százalékban:

```
#include <unistd.h>

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

    return 0;
}
```

Minden mag 100 százalékban:

```
#include <omp.h>

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

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

```
top - 20:09:06 up 3:35, 1 user, load average: 5,68, 2,91, ←
          1,38
Tasks: 329 total, 2 running, 256 sleeping, 0 stopped, 1 zombie
```

```
%Cpu0 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu6 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu7 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem :16373532 total,11701240 free, 2254256 used, 2418036 buff/cache
KiB Swap:16724988 total,16724988 free, 0 used. 13751608 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5850	batfai	20	0	68360	932	836	R	798,3	0,0	8:14.23	infty-f



Werkfilm

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

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

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

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

```
Program T100
{
    boolean Lefagy(Program P)
    {
        if (P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

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

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

T100 (t.c.pseudo)

true

akár önmagára

T100 (T100)

false

ezt a kimenetet adja.

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

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Nem lehet olyan programot írni, amely eldönti egy másik programról, hogy van-e benne végtelen ciklus. A programnak ami ellenőrzi, hogy egy másik programban van-e végtelen ciklus, meg kell várnia a ciklus végét (amely tegyük fel végtelen ciklus), aminek a következménye, hogy egy végtelen ciklusba kerül, így nem fogja tudni eldöntení róla, mert a végtelenségg fog futni ez az ellenőrző program is.

2.3. Változók értékének felcserélése

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

Megoldás videó: [YouTube](#)

Megoldás forrása: [GitHub](#)

Tanulságok, tapasztalatok, magyarázat...

Változók cseréje kivonással

```
#include <stdio.h>

int main()
{
    int a = 1;
    int b = 2;

    printf("Csere előtt a = %d, b = %d\n", a, b);

    a = a + b; // a = 1 + 2 = 3
    b = a - b; // b = 3 - 2 = 1
    a = a - b; // a = 3 - 1 = 2

    printf("Csere után a = %d, b = %d\n", a, b);

    return 0;
}
```

2.4. Labdapattogás

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

Megoldás videó: [YouTube](#)

Megoldás forrása: [if-ekkel C-ben , if nélkül C-ben](#)

Tanulságok, tapasztalatok, magyarázat...

Labdapattogtatás `if`-ekkel C-ben

```
#include <stdio.h>
// curses.h library is a terminal control library. We are using ncurses ( ←
// new curses) the third and latest version of it
#include <curses.h>
// unistd.h library contains system call functions, like usleep()
#include <unistd.h>

int main(void)
{
    WINDOW *screen;
    // initscr() function is belongs to curses.h library (ncurses)
    // We are using it to determine the terminal's type and to initialize ←
    // all curses data structures
    screen = initscr();

    // Starting x coordinate
    int x = 0;
    // Starting y coordinate
    int y = 0;

    // Step
    int xstep = 1;
    // Step
    int ystep = 1;

    // max-width
    int mx;
    // max-height
    int my;

    for (;;)
    {
        // getmaxyx defines the screen's maximum height(y) and width(x) by
        // passing 3 variables (the screen, the empty maxY variable and the ←
        // maxX variable)
        // and then set these variables to their value by the getmaxy() and ←
        // getmaxx() functions
        getmaxyx(screen, my, mx);

        // mvprintw() function is printing the "O" to the terminal's screen ←
        // by the x and y coordinates
        mvprintw(y, x, "O");

        // refresh() function is belongs to curses.h library (ncurses)
        // We are using it to get the actual output to the terminal
        refresh();
        // usleep() function is belongs to unistd.h library
        usleep(10000);
    }
}
```

```
// usleep(100000) is suspending the execution for 100000ms (=100s=1 ↔
// m40s) intervals
usleep(100000);

x = x + xstep;
y = y + ystep;

// if the "O" is at the right side
if (x >= mx - 1)
{
    xstep = xstep * -1;
}
// if the "O" is at the left side
if (x <= 0)
{
    xstep = xstep * -1;
}
// if the "O" is on top
if (y <= 0)
{
    ystep = ystep * -1;
}
// if the "O" is at the bottom
if (y >= my - 1)
{
    ystep = ystep * -1;
}
// clear() function is belongs to curses.h library (ncurses)
// We are using it to clear the terminal's screen
clear();
}

return 0;
}
```

Labdapattogtatás `if` nélkül C-ben

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

int main(void)
{
    WINDOW *screen;
    // initscr() function is belongs to curses.h library (ncurses)
    // We are using it to determine the terminal's type and to initialize ↔
    // all curses data structures
```

```
screen = initscr();
// noecho() function belongs to curses.h library
noecho();
// cbreak() function belongs to curses.h library
cbreak();
// nodelay() function belongs to curses.h library
nodelay(screen, true);

int xj = 0;
int xk = 0;
int yj = 0;
int yk = 0;

int mx = 160;
int my = 48;

for (;;)
{
    xj = (xj - 1) % mx;
    xk = (xk + 1) % mx;
    yj = (yj - 1) % my;
    yk = (yk + 1) % my;
    // clear() function is belongs to curses.h library (ncurses)
    // We are using it to clear the terminal's screen
    clear();
    mvprintw(abs((yj + (my - yk)) / 2), abs((xj + (mx - xk)) / 2), "O") ←
    ;
    // refresh() function is belongs to curses.h library (ncurses)
    // We are using it to get the actual output to the terminal
    refresh();
    // usleep() function is belongs to unistd.h libray
    // usleep(100000) is suspending the execution for 100000ms (=100s=1 ←
    m40s) intervals
    usleep(100000);
}

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!

Megoldás videó: [YouTube](#)

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bogomips.c

Tanulságok, tapasztalatok, magyarázat...

Szóhossz C++ -ban

```
#include <iostream>

using namespace std;

int main()
{
    int a = 1; // Az a változó bittolást vizsgálja
    int i = 0; // Az i változó pedig megszámolja, hogy hányszor tolunk/ ←
               // léptettük el

    // Amíg az a értéke nem lesz 0...
    while (a != 0)
    {
        a <<= 1; // Bitshifteljük
        ++i; // Számoljuk, a tolások/lépések számát
    }

    cout << "Lepesek szama: " << i << "\n";

    return 0;
}
```

BogoMIPS

A második alany egy MacBook Pro (Retina, 15-inch, Mid 2015)
Processor 2,2 GHz Quad-Core Intel Core i7
Memory 16 GB 1600 MHz DDR3
Graphics Intel Iris Pro 1536 MB
System macOS Catalina v10.15.1

```
mac@gergelyn:~/Documents/Skool/bhax-master-thematic_tutorials-bhax_textbook_IgyNeveldaProgramozod/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing$ gcc bogomips.c -o bogomips
mac@gergelyn:~/Documents/Skool/bhax-master-thematic_tutorials-bhax_textbook_IgyNeveldaProgramozod/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing$ ./bogomips
Calibrating delay loop...ok - 934.00 BogoMIPS
mac@gergelyn:~/Documents/Skool/bhax-master-thematic_tutorials-bhax_textbook_IgyNeveldaProgramozod/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing$
```

2.1. ábra. BogoMIPS futtatásának eredménye a MacBookon

A második alany egy CustomPC
Processor Ryzen 7 2700 Eight-Core Processor
Memory 16 GB 3000 MHz DDR4
Graphics MSI GeForce GTX 1660 Ti
System Windows 10 Pro v1903 build18362.657

The screenshot shows the Visual Studio Code interface. The left sidebar has icons for file operations like Open, Save, Find, and Refresh. The main editor window displays the content of the `bogomips.c` file. The terminal tab at the bottom shows the command-line output of running the program.

```
// BHAX BogoMIPS
// 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
//
// This program is based on
$ gcc bogomips.c -o bogomips
Nagy Gergely@DESKTOP-G98BGK5 MINGW64 /d
$ ./bogomips
Calibrating delay loop..ok - 925.63 BogoMIPS

Nagy Gergely@DESKTOP-G98BGK5 MINGW64 /d
$ ./bogomips
Calibrating delay loop..ok - 990.53 BogoMIPS
$
```

2.2. ábra. BogoMIPS futtatásának eredménye a Windows PC-men

2.6. Helló, Google!

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

Megoldás videó: [YouTube](#)

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Forrás: Bátfai Norbert

```
#include <stdio.h>
#include <math.h>
// A kiir függvény bekér egy lebegőpont típusú tömböt (tomb)
// és egy egész tipusú változó (db = darab)
void kiir(double tomb[], int db)
{
    // Deklarálunk egy i változót, a for ciklushoz
```

```
int i;
// A for ciklusban megadjuk, hogy addig fussen a loop
// míg el nem éri a db értékét
for (i = 0; i < db; i++)
{
    // minden egyes ciklusban kiíratjuk az adott i oldal rankját
    // az adatot a tömb i-edik tagja adja
    printf("PageRank[%d]: %f\n", i, tomb[i]);
}

// A tavolsag függvény lebegőpontos értéket fog vissza adni
// bekér egy PR és egy PRv nevű, lebegőpont tipusú tömböt
// és egy egész tipusú változót (db = darab)
double tavolsag(double PR[], double PRv[], int db)
{
    // Deklarálunk egy i változót, a for ciklushoz
    int i;
    // Deklarálunk és inicializáljuk 0.0-val a lebegőpontos tipusú osszeg ←
    // változót
    double osszeg = 0.0;
    // A for ciklusban megadjuk, hogy addig fussen a loop
    // míg el nem éri a db értékét
    for (i = 0; i < db; i++)
    {
        // minden egyes ciklusban az osszeg változó értékéhez adjuk
        // kétszer a PRv tömb i-edik tagjának értékéből kivont PR tömb i- ←
        // edik tagjának értékét
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);
    }
    // A for ciklus után (ha i elérte a db ot), a függvény az osszeg ←
    // változó értékének négyzetét adja vissza
    return sqrt(osszeg);
}

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

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

    for (;;)
    {
        // for ciklus i változóval, addig fut amíg i kisebb mint 4 és i ←
```

```
növekszik 1-gyel
for (i = 0; i < 4; i++)
{
    // a PR tömb i-edik eleme = 0.0-val
    PR[i] = 0.0;
    // Újabb for ciklus ugyanezen az elven csak j változóval
    for (j = 0; j < 4; j++)
    {
        // a PR tömb i-edik eleméhez (ami kezdetben mindig 0.0), ←
        // adja hozzá
        // az L tömb i-edik sor, j-edik oszlopának értéke szorozva ←
        // a PRv tömb j-edik elemével
        PR[i] += (L[i][j] * PRv[j]);
    }
}

// Hívjuk a tavolsag függvényt a PR, PRv tömbökkel és db = 4 -gyel
// Ezekkel az adatokkal, ha az osszeg negyzete kisebb mint ←
// 0.0000000001...
if (tavolsag(PR, PRv, 4) < 0.0000000001)
{
    // ... akkor álljon le a program
    break;
}

// for ciklus i változóval, addig fut amíg i kisebb mint 4 és i ←
// növekszik 1-gyel
for (i = 0; i < 4; i++)
{
    // a PRv tömb i-edik elemének értéke legyen egyenlő a PR tömb i ←
    // -edik eleméjével
    PRv[i] = PR[i];
}
}

// Hívjuk a kiir függvényt a PR tömbbel és 4 db-al
kiir(PR, 4);

return 0;
}
```

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [YouTube](#)

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

A Monti Hall probléma egy paradoxon, amely azt feltételezi, hogy az első választásnál 1/3 az esély, hogy jól választunk, majd az ajtó kinyílik és felkínálják a második választás lehetőségét, ahol már 2/3 esélye annak, hogy a kívánt nyeremény van az ajtó mögött. Az alábbi R program ezt a paradoxont próbálja szimulálni egy nagy terjedelmű, véletlen generált választások sorozatával, ahol a második körben változtattak és ahol nem változtattak. Ennek az eredménye az, hogy érdemes váltani. A programban megadjuk, hogy hány próbálkozást szeretnénk (minél nagyobb számot adunk, annál pontosabb lesz az eredmény). Létrehozunk benne egy játékost aki a döntéseket végzi és egy műsorvezetőt, aki megpróbálja elbizonytalánítani a játékosokat. Ezt követően megnézzük, hogy ha nem változtat hányszor nyer és ha változtat úgy hányszor nyer. Következmény képp kiderül, hogy ha elsőre eltaláljuk és nyerni akarunk, akkor nem kell változtatnunk. Viszont ennek az esélye 1/3. Azaz jobban járunk, ha minden változtatunk, mivel 2/3 az esélye annak, hogz nem találjuk el elsőre a kívánt ajtót, ami mögött a nyeremény van.

```
# An illustration written in R for the Monty Hall Problem
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or
# modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
# https://bhaxor.blog.hu/2019/01/03/erdos\_pal\_mit\_keresett\_a\_nagykonyvben\_a\_monty\_hall-paradoxon\_kapcsan
#
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musrorvezeto=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)
```

2.8. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

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

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

Két egymást követő páratlan prím között a legkisebb távolság a 2, a legnagyobb távolság viszont bármilyen nagy lehet! Ez utóbbit könnyű bebizonyítani. Legyen n egy tetszőlegesen nagy szám. Akkor szorozzuk

össze $n+1$ -ig a számokat, azaz számoljuk ki az $1 \cdot 2 \cdot 3 \cdots \cdot (n-1) \cdot n \cdot (n+1)$ szorzatot, aminek a neve $(n+1)$ faktoriális, jele $(n+1)!$.

Majd vizsgáljuk meg az a sorozatot:

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

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

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

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

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

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

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
```

```
library(matlab)

stp <- function(x) {

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

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

Soronként értelemezzük ezt a programot:

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

Kiszámolja a megadott számig a prímeket.

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

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

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

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

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

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

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

Kivette a primes-ból a párok első tagját.

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

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

```
> 1/t1primes
```

```
[1] 0.33333333 0.20000000 0.09090909
```

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

```
> 1/t2primes
```

```
[1] 0.20000000 0.14285714 0.07692308
```

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

```
> 1/t1primes+1/t2primes
```

```
[1] 0.53333333 0.3428571 0.1678322
```

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

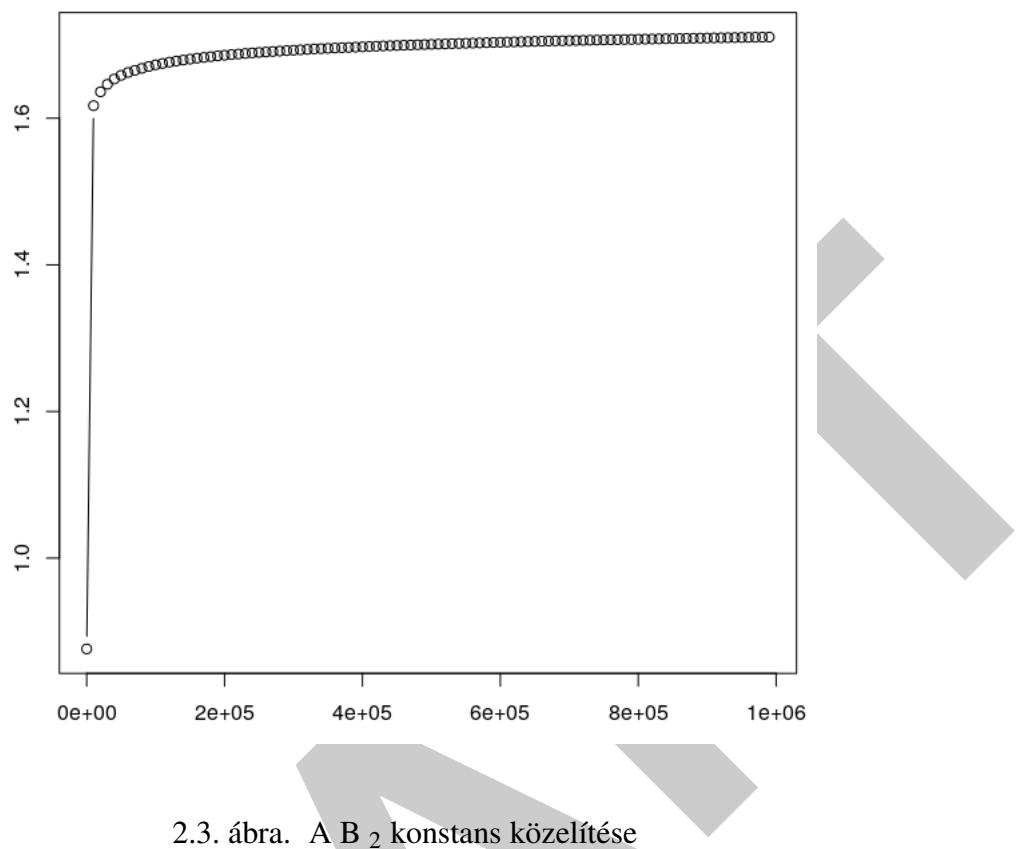
```
sum(rt1plust2)
```

```
> sum(rt1plust2)
```

```
[1] 1.044023
```

A következő ábra azt mutatja, hogy a szumma értéke, hogyan nő, egy határértékhez tart, a B₂ Brun konstanshoz. Ezt ezzel a csipettel rajzoltuk ki, ahol először a fenti számítást 13-ig végezzük, majd 10013, majd 20013-ig, egészen 990013-ig, azaz közel 1 millióig. Vegyük észre, hogy az ábra első köre, a 13 értékhez tartozó 1.044023.

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

**Werkfilm**

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

2.9. MALMO Csiga

Megoldás forrása: Github

Megoldás videó: YouTube

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó: [YouTube](#)

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A turing gép bemenetként egy decimális számot kap, és a kimenete egyesekből fog állni. A decimális szám nagyságától függ az, hogy mennyi egyest fogunk kapni a kimenetben. Például a 2 (decimális szám) az unáris rendszerben 11, vagy az 5 az egyenlő 11111. Alapvetően a Turing gépeken végzett műveletek által balra vagy jobbra haladunk a memóriaszalagon és átmegyünk a következő állapotra. A gép részei még az író-olvasó fej, illetve a vezérlőegység.

Az alábbi kódban megkérjük a felhasználót, hogy adjon meg egy decimális számot, amit egy változóban fogunk eltárolni. Ezt a változót bementként megadjuk a decimálisból unárisba átváltó függvénybe, amely egy for ciklus segítségével egyeseket fog kiírni a konzolra, ezáltal megalkottuk a Turing gépet.

```
#include <iostream>

using namespace std;

int DecimalToUnary(int x) {

    for(int i = 0; i < x; i++) {
        cout << 1;
    }
}

int main() {

    int a;
    cout << "Decimális szám: ";
    cin >> a;
```

```
    DecimalToUnary(a);  
}
```

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

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

Megoldás videó: [YouTube](#)

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

1-es típusú (környezetfüggő), ha minden szabálya $\alpha A\gamma \rightarrow \alpha\beta\gamma$ alakú, ahol $A \in N, \alpha, \gamma \in (N \cup T)^*, \beta \in (N \cup T)^+$. Ezenkívül megengedhető az $S \rightarrow \epsilon$ szabály is, ha nem szerepel egyetlen szabály jobb oldalán sem.

Környezetfüggő (hossznemcsökkentő) $P1XP2 \rightarrow P1QP2, P1, P2$ eleme $(VN \cup VT)^*$, X VN beli, Q (VN \cup VT) $^+$ beli, kivéve $S \rightarrow \epsilon$ üres, de akkor S nem lehet jobb oldali egyetlen szabályban sem

- 0-s típusú: átlalános vagy mondatszerkezetű
- 1-es típusú: környezetfüggő
- 2-es típusú: környezetfüggetlen
- 3-as típusú: reguláris

3.3. Hivatkozási nyelv

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

Megoldás videó: [YouTube](#)

Megoldás forrása: [GitHub](#)

Tanulságok, tapasztalatok, magyarázat...

```
A C utasítás fogalma  
<utasítás> ::=  
  <címkezett_utasítás>  
    <azonosító> : <utasítás>  
    case (<állandó_kifejezés>) : <utasítás>  
    default: <utasítás>  
  <kifejezésutasítás>  
    <kifejezésopc>;  
  <összetett_utasítás>
```

```
{deklarációs_list้าopc utasítás_listाopc}
deklarációs_listा:
    deklaráció
    deklarációs_listा deklaráció
utasítás_listा:
    <utasítás>
    utasítás_listा <utasítás>
<kiválasztó_utasítás>
    if (<kifejezés>) <utasítás>
    if (<kifejezés>) <utasítás> else <utasítás>
    switch (<kifejezés>) <utasítás>
<iterációs_utasítás>
    while (<kifejezés>) <utasítás>
    do <utasítás> while (<kifejezés>);
    for (<kifejezésopc>; <kifejezésopc>; <kifejezésopc>) <utasítás>
<vezérlésátadó_utasítás>
    goto <azonosító>
    continue:
    break;
    return <kifejezésopc>;
```

For ciklus C-ben

```
#include <stdio.h>

int main()
{
    int n = 0;

    for (int i = 10; n <= i; n++) // Mivel az i változó deklarációja a for ↫
        // ciklus fejében történt, így C89-ben nem fog lefordulni, míg C99-ben ↫
        // igen
    {
        printf("%d\n", n);
    }

    return 0;
}
```

3.4. Saját lexikális elemző

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

Megoldás video: [YouTube](#)

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.l

```
%{  
#include <stdio.h>  
int realnumbers = 0; // a számláló  
%}  
digit [0-9] // 0-tól 9-ig keresse a számokat  
%%  
{digit}*(\.{digit}+)? {++realnumbers;  
    printf("[realnum=%s %f]", yytext, atof(yytext));} // printeljük a ←  
        számokat, és növeljük a számláló értékét  
%%  
int  
main ()  
{  
    yylex ();  
    printf("The number of real numbers is %d\n", realnumbers); // kiírjuk hány ←  
        valós számot talált  
    return 0;  
}
```

A lexer egy lexikális elemző. A program lényege, hogy tokeneket lehet megadni, ami alapján keres és előtönti, hogy mit csináljon az adott tokennel. Lehetséges karakterek felcserélése, összegzése stb. Az adott program meg szeretné számolni a valós számokat.

Az adott program egy .l kiterjesztésű file, amit fordítanunk kell .c -re a lex parancssal. Ezt követően a .c forráskódot kell fordítanunk az -ll kapcsolóval.

3.5. Leetspeak

Lexelj össze egy l33t cipher!

Megoldás videó: [YouTube](#)

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.l

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <ctype.h>  
  
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))  
  
struct cipher {  
    char c;  
    char *leet[4];
```

```
    } l337d1c7 [] = {  
  
    {'a', {"4", "4", "@", "/-\\"}},  
    {'b', {"b", "8", "13", "|{}}},  
    {'c', {"c", "(", "<", "{}"},  
    {'d', {"d", "|)", "[ ]", "|{}"}},  
    {'e', {"3", "3", "3", "3"}},  
    {'f', {"f", "|=", "ph", "|#"}},  
    {'g', {"g", "6", "[", "[+{}"}},  
    {'h', {"h", "4", "-|", "[-{}"}},  
    {'i', {"1", "1", "|", "!"}},  
    {'j', {"j", "7", "_|", "_/"}},  
    {'k', {"k", "|<", "1<", "|{"}},  
    {'l', {"l", "1", "|", "|_"}},  
    {'m', {"m", "44", "(V)", "|\\/|"}},  
    {'n', {"n", "|\\|", "/\\/", "/v"}},  
    {'o', {"0", "0", "()", "[]"}},  
    {'p', {"p", "/o", "|D", "|o"}},  
    {'q', {"q", "9", "O_", "(,)"}},  
    {'r', {"r", "12", "12", "|2"}},  
    {'s', {"s", "5", "$", "$"}},  
    {'t', {"t", "7", "7", "'|'"}}},  
    {'u', {"u", "|_|", "(_)", "[_]"}},  
    {'v', {"v", "\\/", "\\\/", "\\\/"}}},  
    {'w', {"w", "VV", "\\\/\\"}, "(/\\)"}}},  
    {'x', {"x", "%", ")("}},  
    {'y', {"y", "", "", ""}}},  
    {'z', {"z", "2", "7_", ">_"}},  
  
    {'0', {"D", "0", "D", "0"}},  
    {'1', {"I", "I", "L", "L"}},  
    {'2', {"Z", "Z", "Z", "e"}},  
    {'3', {"E", "E", "E", "E"}},  
    {'4', {"h", "h", "A", "A"}},  
    {'5', {"S", "S", "S", "S"}},  
    {'6', {"b", "b", "G", "G"}},  
    {'7', {"T", "T", "j", "j"}},  
    {'8', {"X", "X", "X", "X"}},  
    {'9', {"g", "g", "j", "j"}}}  
  
// https://simple.wikipedia.org/wiki/Leet  
};  
  
%}  
%%  
. {  
    // A kicserélés kezdete...  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
    {
```

```
if(l337d1c7[i].c == tolower(*yytext)) // kisbetűssé alakítjuk
{
    int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0)); // randomot adunk ←
    hozzá

    if(r<91) // random l33t karakterre cseréljük ki
        printf("%s", l337d1c7[i].leet[0]);
    else if(r<95)
        printf("%s", l337d1c7[i].leet[1]);
    else if(r<98)
        printf("%s", l337d1c7[i].leet[2]);
    else
        printf("%s", l337d1c7[i].leet[3]);

    found = 1;
    break;
}

if(!found) // ha nem talált...
    printf("%c", *yytext); // ...akkor az eredeti szöveget adjuk vissza

} // a kicserelelés vége...
%%

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

Az adott programban a megadott karaktereket (első oszlop) cseréljük ki az l33t párraira (az adott sor további karakterei).

3.6. A források olvasása

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

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

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

**Bugok**

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

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása:

Ha az átirányított SIGINT SIG_IGN -re nem egyenlő SIG_IGN -nel, akkor legyen átirányítva a jelkezelőre.

Pre-incremet for ciklus, amely kiírja a számokat 0-tól 4-ig.

Post-increment for ciklus, amely kiírja a számokat 0-tól 4-ig.

Post-increment for ciklus, amely a tomb i-edik elemét növeli 1-gel és kiírja 0-tól 4-ig a számokat.

Pre-increment for ciklus n-ig ÉS a két mutatónak ugyanaz legyen az értéke.

Printel két decimális számot, amelyeket az f függvénynek ad vissza.

Printel két decimális számot, amelyből az egyik a függvénytől kapott szám, másik az érték.

Printel két decimális számot, amelyből az egyik a függvénynek adott referencia, a másik az alap érték.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

Minden x, létezik y, x kisebb mint y vagy x prím

Minden x, létezik y, x kisebb mint y vagy prím vagy SSy prím

Létezik y, minden x, ahol x prím és y nagyobb

Létezik y, minden x y kisebb mint x és ebből nem következik, hogy x prím

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
int a; // egész
```

- ```
int *b = &a; // egészre mutató, ami megkaptá a ↔
memóriacímét
```
- ```
int &r = a; // egész referencia a -ra
```
- ```
int c[5]; // 5 tagú egészek tömbje
```
- ```
int (&tr)[5] = c; // 5 tagú egészek mutató tömbje
```
- ```
int *d[5]; // egész típusú függvény mutató
```
- ```
int *h();
```
- ```
int *(*l)();
```
- ```
int (*v(int c))(int a, int b)
```
- ```
int (*(*z)(int))(int, int);
```

Megoldás forrása:

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

```
#include <stdio.h>

int
sum (int a, int b) // 2 param osszege a return ertek ebbe a functionbe
{
    return a + b;
}

int
mul (int a, int b) // 2 param szorzata a return ertek ebbe a functionbe
{
    return a * b;
}

int (*sumormul (int c))(int a, int b) // egeszet kap, pointerrel ter ↔
// vissza es fuggvenyre mutat ez a function
{
    if (c)
        return mul;
```

```
else
    return sum;

}

int
main ()
{
    int (*f) (int, int); // function call
    f = sum; // f erteke
    printf ("%d\n", f (2, 3)); // print
    int (*(*g) (int)) (int, int); // function call
    g = sumormul; // g erteke
    f = *g (42); // f erteke
    printf ("%d\n", f (2, 3)); // print
    return 0;
}
```

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

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

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

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

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

```
int
main ()
{
    F f = sum;

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

    G g = sumormul;

    f = *g (42);

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

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

3.9. MALMO Diszkret

Megoldás forrása: [Github](#)

Megoldás videó: [YouTube](#)

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: [YouTube](#)

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c

A program egy alsó háromszögmátrixot fog létrehozni. Az nr (number of rows) a mátrix sorainak számát tároljuk, a tm pedig egy pointerre mutató pointer, aminek lefoglalunk 8 bájtot a memoriában.

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

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

A malloc() függény lefoglal memóriát és egy pointert térít vissza. A tm-nek a malloc segítségével lefoglalunk 40 bájtot, ami úgy jön ki, hogy 5 az nr és a sizeof double * 8, tehát az 5*8 szorzat 40. Az if-ben megnézzük, hogy ha 0-val egyenlő a lefoglalt méret, akkor vagy egy null pointert térít vissza, vagy pedig egy olyan pointert térít vissza, amit át lehet adni a free függvénynek. A tm-nek a lefoglalásához használnunk kell egy double** típuskényszerítést, mert alapból a malloc void*-ot ad vissza. A program le fog állni, ha nincs több memória és null értéket kapunk vissza.

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

Egy for ciklusban végigmegyünk az összes soron. A malloc függvény arra szolgál, hogy az adott tm elemekbe betöltsük a hozzájuk tartozó memória foglalást, az első elem esetén 8 bájt. Itt szintén kell double* típuskényszerítést használni.

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

}
```

A dupla for ciklussal 0-tól 14-ig számokkal feltöljük a háromszögmátrixot, amit ki is íratunk.

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

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

Itt ebben a részben ha eltüntetjük a külső zárójelet, akkor sem fog változni a program kimenete futáskor.

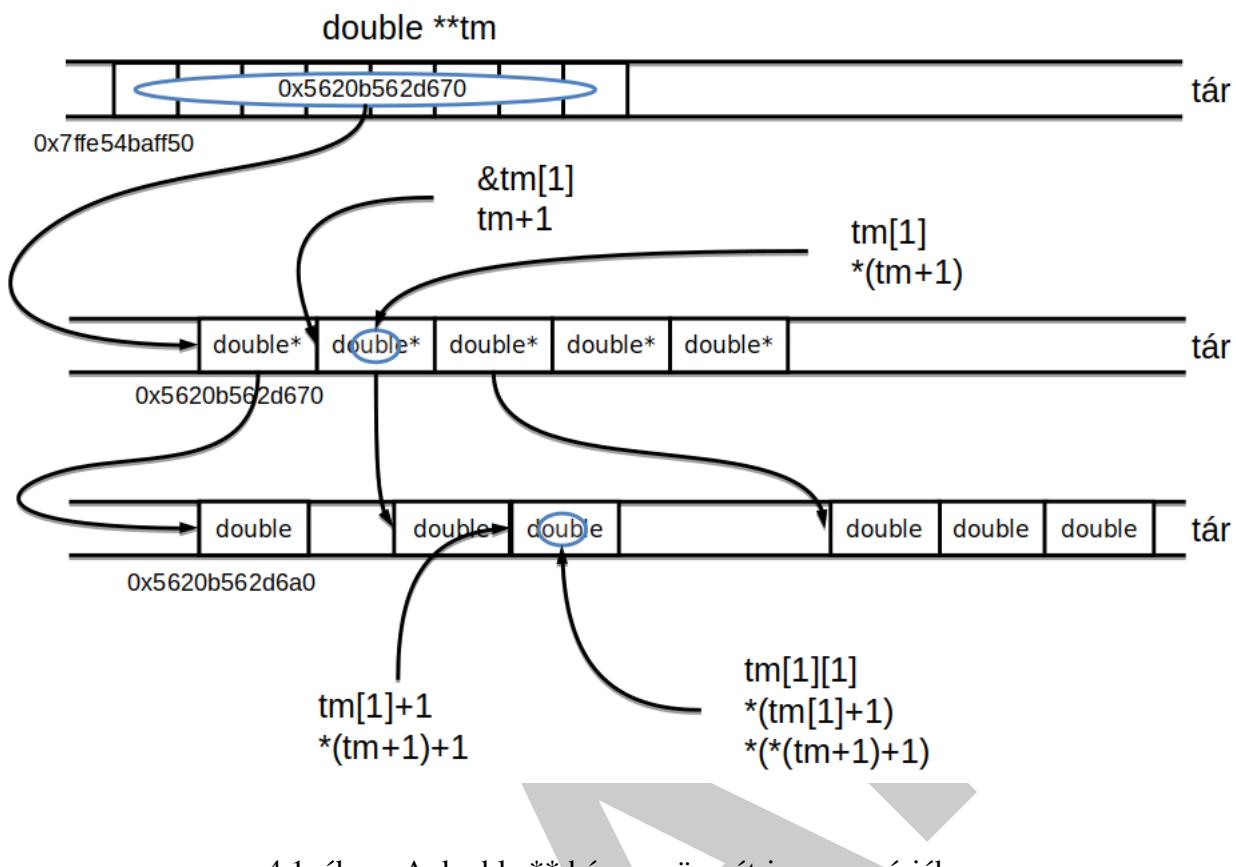
```
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;
```

A program végén a memóriában felszabadítjuk a pointerek által lefoglalt helyet a free függvény segítségével.

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

free (tm);

return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó: [YouTube](#)

Megoldás forrása: https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_parhuzamos_prog_linux/ch05s02.html?fbclid=IwAR0Ob3YIHBHHRixo0g2_qYixTp93FbqkCmIxXpFE38uxjFClzg

Tanulságok, tapasztalatok, magyarázat...

A kódcsípetben látható 3 függvénykönyvtárat beimportáljuk, majd definiálunk 2 konstansot, az egyik a maximális kulcs méretet határozza meg, a másik pedig a beolvasott bájtok maximális számát jelenti.

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

A mainben 2 char típusú tömböt deklarálunk, amelyeknek a mérete megegyezik a kulcs illetve buffer értékével.

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

int kulcs_index = 0;
int olvasott_bajtok = 0
```

A kulcs_meret változóban az strlen függvény arra szolgál, hogy lekérjük a kulcs hosszát. Az strcpy függvénnyel átmásoljuk a parancssori argumentumot bájtonként a kulcsba.

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

A while cikluson belül az olvasott_bajtok változónak értékül adjuk a bufferból beolvasott bajtok számát a read függvénnyel, ami a BUFFER_MÉRETnél nem lehet nagyobb.

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

```
{
```

A while ciklus után következik egy for ciklus amely 0-tól egészen elmegy az olvasott bajtokig, majd vesszük a buffer adott bajtjának illetve a kulcs adott bajtjának exorát. Annak érdekében, hogy jobb legyen a titkosítás, a kulcsindexet átírjuk az adott indexmérettel való maradékos osztásra minden bajt után.

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

```
{
```

```
    buffer[i] = buffer[i] ^ kulcs[kulcs_index];
```

```
    kulcs_index = (kulcs_index + 1) % kulcs_meret;
```

```
}
```

Végül a write függvény segítségével a titkosított szöveget beírjuk bájtonként a bufferba az olvasott bajtokig.

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

```
}
```

```
}
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó: [YouTube](#)

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

A Java és a C titkosító között nincs különbség. A program egy osztályba fog kerülni, ami az ExorTitkosító. A titkosítás pedig az ExorTitkosító függvénybe fog kerülni, aminek a paraméterei É string (a kulcs), és az input output csatornák. A getBytes függvény átalakítja byteok tömbjére a kulcsszöveget. A titkosított szöveget a kimenőCsatorna.write-tal írjuk a megadott output fileba. A read függvény pedig beolvassa a megadott input fileból az olvasott byteokat.

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

4.4. C EXOR törő

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

Megoldás videó: [YouTube](#)

Megoldás forrása: https://gitlab.com/misi1201/bhax/-/blob/master/thematic_tutorial/Caesar%20feladatok/t.c

Tanulságok, tapasztalatok, magyarázat...

A program elején beimportáljuk a megfelelő header fájlokat, illetve definiáljuk a szükséges konstansokat.

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE

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

Az atlagos_szohossz függvényben elosztjuk a szóközök számával a titkos szöveg méretét bájtokban, így megkapjuk, hogy egy szó hány bájtos átlagosan.

```
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;
}
```

A tiszta_lehet függvényben azt állapítjuk meg, hogy a szövegen szerepelnek-e a leggyakoribb magyar szavak, amik: hogy,nem,ha,az. Továbbá megnézzük azt is, hogy 6 illetve 9 között van-e az átlag szóhossz. Ezeket a lépéseket a returnben hajtjuk végre az && (és) operátorok segítségével. Így képesek vagyunk a potenciális törések számát csökkenteni.

```
int tiszta_lehet (const char *titkos, int titkos_meret)
{
    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

Az exor függvényben a for cikluson belül átvizsgáljuk a titkos szöveget és ennek a szövegnek a bájtjait exorozzuk a kulcsindexrel. A kulcsindexet növeljük minden 1-el, majd pedig maradékosan osztjuk a kulcs méretével. Ennek következtében a kules_index minden 0-ról fog indulni és akkor is ki fog nullázódni, ha eléri a kulcs_meret konstansot.

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

Az exor_tores függvényben az exor függvény meghívásra kerül , illetve a returnben a tiszta_lehet függvény. Ez a függvény azt nézi, hogy sikeres volt-e a törés.

```
int exor_tores (const char kulcs[], int kulcs_meret, char titkos[], int ←
    titkos_meret)
{
    exor (kulcs, kulcs_meret, titkos, titkos_meret);
    return tiszta_lehet (titkos, titkos_meret);
}
```

A main függvényben 2 tömböt hozunk létre, az egyik a kulcsokat, a másik a titkos szöveget fogja tárolni. Deklarálunk még egy p pointert , amely a titkos szöveg első karakterére mutat. A while ciklusban a read függvény segítségével az olvasott bájtok számát betöljtük az olvasott_bajtok változóba. A read függvények paraméterei a 0, vagyis hogy a standard inputról fogjuk beolvasni a szöveget , azután hogy hol tároljuk a szöveget amit beolvasunk és mekkora az a maximális méret amit beolvas. Abban az esetben, ha a titkos szöveg és p különbségének a buffer bájtszámával vett összege kisebb, mint a titkos szöveg maximális bájt száma, akkor a bufferig olvassa be a bajtakat. Viszont ha nagyobb, akkor a titkos szöveg + 4096 bájt - p -ig olvassa be a bajtakat. Ha kész van a beolvasás, akkor az utolsó beolvasott bájt memóriacímét értékül adjuk p-nek.

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

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

A szövegen ahol nincsen semmi, ott kinullázzuk azokat a helyeket a titkos szövegen a for ciklus segítségével.

```
for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';
```

A lehetséges kulcsokat a for ciklussal előállítjuk , majd ha az exor törés függvényel kipróbáljuk őket és ha igazat ad akkor készen vagyunk. Ha sikertelen volt a törés, akkor az eredeti szöveget visszafejtjük és próbálkozunk újra.

```
for (int ii = '0'; ii <= '9'; ++ii)
    for (int ji = '0'; ji <= '9'; ++ji)
        for (int ki = '0'; ki <= '9'; ++ki)
            for (int li = '0'; li <= '9'; ++li)
                for (int mi = '0'; mi <= '9'; ++mi)
                    for (int ni = '0'; ni <= '9'; ++ni)
                        for (int oi = '0'; oi <= '9'; ++oi)
                            for (int pi = '0'; pi <= '9'; ++pi)
                            {
                                kulcs[0] = ii;
                                kulcs[1] = ji;
                                kulcs[2] = ki;
                                kulcs[3] = li;
                                kulcs[4] = mi;
                               kulcs[5] = ni;
                                kulcs[6] = oi;
                                kulcs[7] = pi;

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

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

return 0;
}
```

4.5. Neurális OR, AND és EXOR kapu

R

Megoldás videó: [YouTube](#)

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

Tanulságok, tapasztalatok, magyarázat...

Ha a bemeneti értékek súlyozott összege (minden számnak van egy súlyértéke) egy bizonyos határértéket meghalad, akkor indul el a program.

A programban meghívjuk a neuralnet könyvtárat. A neuronok lehetséges értékeit a1 illetve a2-be betöljük, az OR-ba pedig ezek logikai 'vagy' műveletének értékét, majd elmondjuk az összes lehetséges esetet. A program képes lesz magát tanítani, és a súlyokat beállítja magának. Ugyanilyen módon megtanítjuk neki az 'és' illetve 'exor' műveleteket. Exor esetében minél több a rejtett neuronok száma, annál hatékonyabb a tanulási algoritmus.

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
# https://youtu.be/Koyw6IH5ScQ

library(neuralnet)

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

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

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

plot(nn.or)

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

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

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

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

```
plot(nn.operand)

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

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

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)

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

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: [YouTube](#)

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

Tanulságok, tapasztalatok, magyarázat...

Létrehozzuk a képet, aminek a mérete bekerül a size változóba, majd a magasságát összeszorozzuk a szélességével. A magasság illetve szélesség lekérésére a get_width illetve get_height függvényt használjuk.

Csinálunk egy perceptron objektumot memóriakezelésre, aminek 3 rétege van:

Az első rétegbe annzi neuron kerül, amekkora a kép mérete, a második rétegbe 256, majd a harmadik rétegbe 1 neuron kerül (utóbbi az eredménynek lesz lefoglalva)

A double* imageben a képnek helyetfoglalunk és feltöljük a beolvasott képpel egy dupla for ciklussal (egyik a szélesség, másik a magasság miatt). A betöltésnél csak a piros színkódú komponensek töltődnek be a képbe. Ezután a perceptron, mint függvényt használjuk az eredmény értékadásánál. A program végén a delete függvényekkel szabadítjuk fel a pointerek és a kép által lefoglalt memóriát.

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

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

    int size = png_image.get_width() * png_image.get_height();

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

    double* image = new double[size];

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

    double value = (*p)(image);

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

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

4.7. MALMO

Megoldás forrása: [Github](#)

Megoldás videó: [YouTube](#)

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás videó: [YouTube](#)

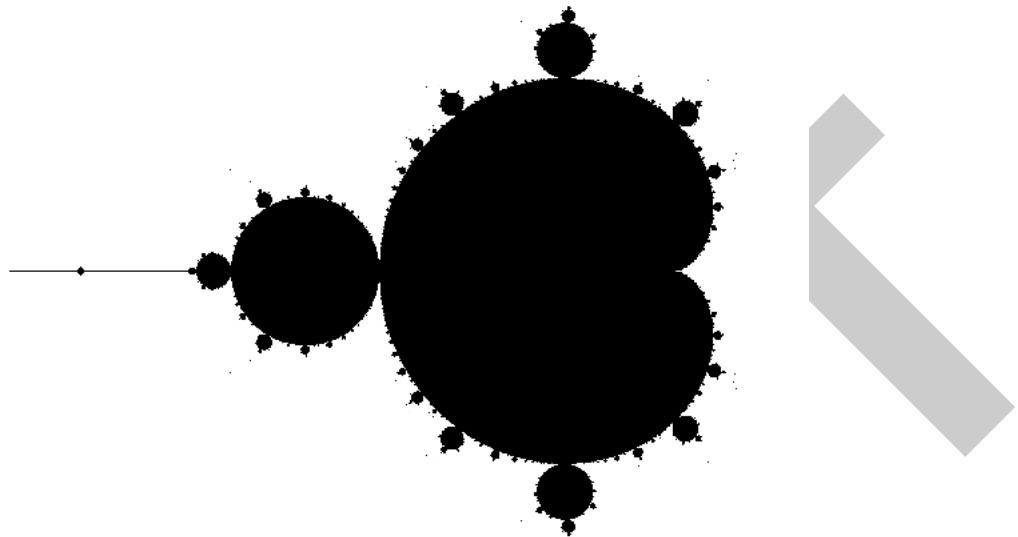
Megoldás forrása: [bhax/attention_raising/CUDA/mandelpngt.c++](#) nevű állománya.

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-öt kapunk, mert ez a szám például a 3i komplex szám.

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1} = z_n^2 + c$, ($0 \leq n$) képlet alapján úgy, hogy a c az éppen vizsgált rácspont. A z_0 az origó. Alkalmaszva 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).



5.1. ábra. A kirajzolt Mandelbrot halmaz

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

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

Megoldás videó: [YouTube](#)

Megoldás forrása:

A program az előző algoritmus alapján számítja ki és rajzolja meg a Mandelbrot halmazt. Szükségünk van a png++ és a complex könyvtárakra. Futtatásnál deklaráljuk a létrehozandó kép szélességét magasságát és iterációs határát.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ←
// -0.01947381057309366392260585598705802112818 ←
// -0.0194738105725413418456426484226540196687 ←
// 0.7985057569338268601555341774655971676111 ←
// 0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ←
// 0.4127655418209589255340574709407519549131 ←
// 0.4127655418245818053080142817634623497725 ←
```

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

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

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

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

```
d = atof ( argv[8] );
}

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

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

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

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

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

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

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

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

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

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

            ++iteracio;
        }

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

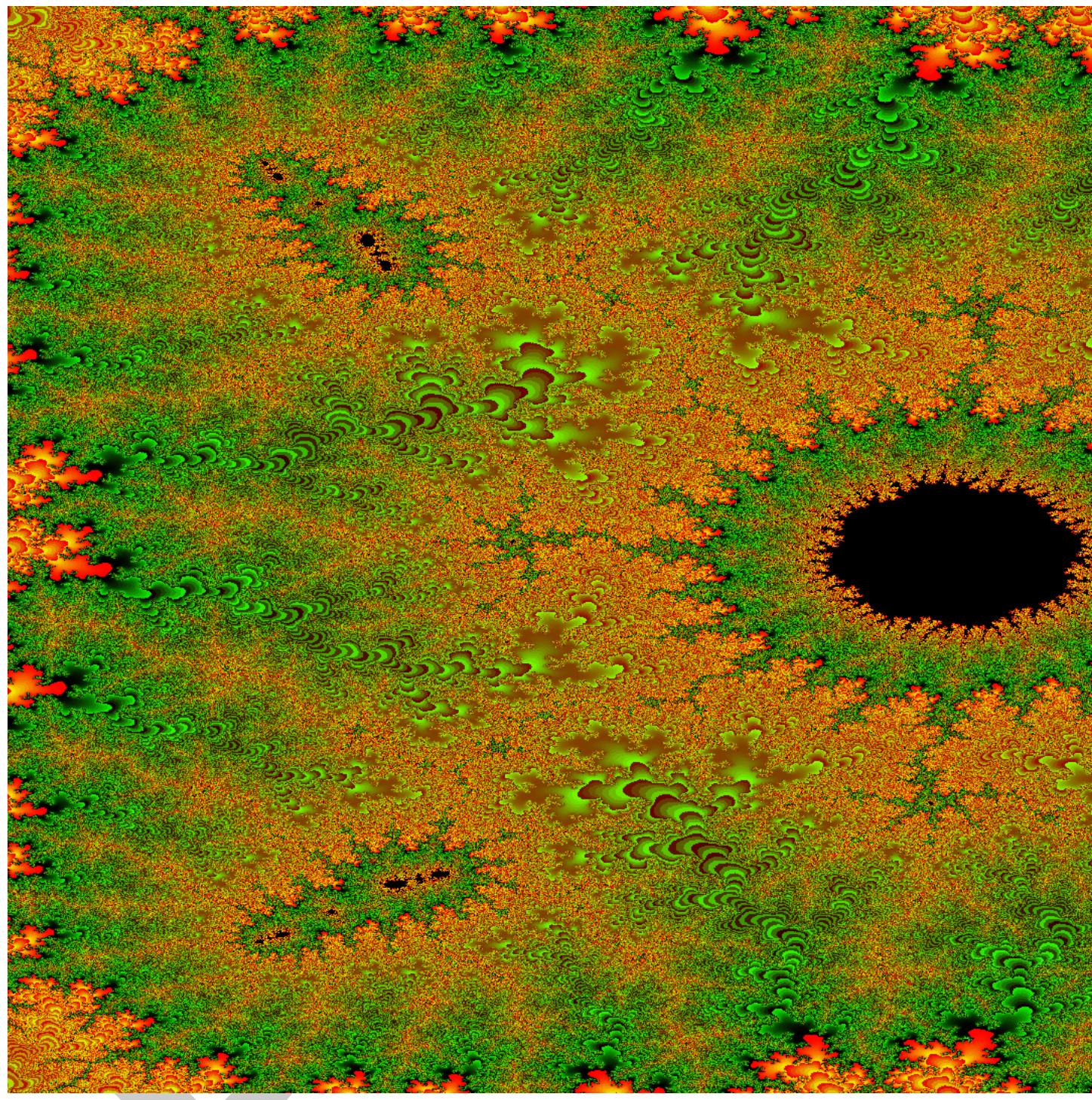
int szazalek = ( double ) j / ( double ) magassag * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
```

```
}

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

}
```

DRAFT



5.2. ábra. A kirajzolt complex Mandelbrot halmaz

5.3. Biomorfok

Megoldás videó: [YouTube](#)

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 természetű törvényre bukkant.

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

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

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

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

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

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

            ++iteracio;
        }
}
```

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

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

```
}
```

```
// 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 -l --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/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ←
// Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
```

```
double ymin = -1.3;
double ymax = 1.3;
double reC = .285, imC = 0;
double R = 10.0;

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

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

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

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

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

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

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

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

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

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

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

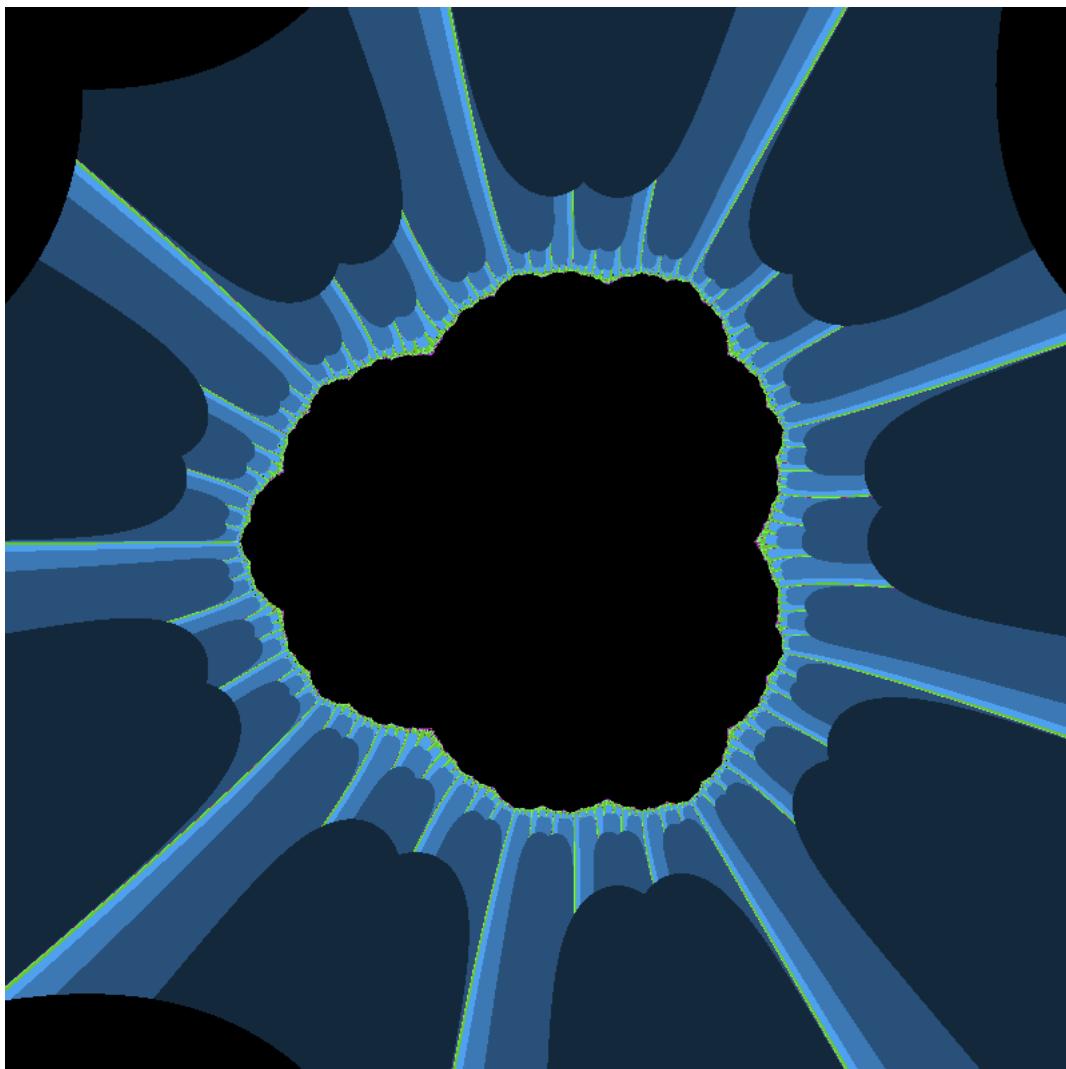
kep.set_pixel ( x, y,
                png::rgb_pixel ( (iteracio*20)%255, (iteracio*40)%255, (iteracio*60)%255 ) );
}

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

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

}
```

DRY



5.3. ábra. A kirajzolt Biomorf

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu nevű állománya.

A CUDA technológiával való megvalósítás csökkenti a program futási idejét, mivel CPU szál helyett a GPU szálait használja. Létrehozunk egy 60x60-as blokkot, ahol blokkonként 100 szál fog futni, tehát 600x600-as Mandelbrot halmaz lesz. 50-70x lesz így rövidebb a számítási idő. Létrehozunk egy kernelt (mandelkernel) és minden szál, ami ezt használja, külön azonosítóval fog rendelkezni, ezek a tj és tk-k. Létrehozzuk a j és k blokkokat, amelyekbe 10x annyi szálat helyezünk el. `cudaMalloc()` függvényteljesen lefoglaljuk a szükséges memóriát majd létrehozzuk a 3 dimenziós hálót (`dim3`). A `cudaMemcpy` függvényteljesen átmásoljuk az adatokat a device memóriába a host memóriából, majd a `cudaFree()` függvényteljesen felszabadítjuk a memóriát.

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↔
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
_global__ void
mandelkernel (int *kepadat)
{
    int tj = threadIdx.x;
    int tk = threadIdx.y;

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

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

void
cudamandel (int kepadat[MERET] [MERET])
{
    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

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

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

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

}
```

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:

A GUI QT-ban van megírva, hogy Cross-Platformos legyen.

A header fileokban a szükséges osztályokat és a függvényeket tároljuk.

A frakablak az ablak elkészítését végzi. Itt állítjuk be az ablak nevét, paramétereit és itt számítjuk újra a képet minden nagításnál és a kép színeit is itt állítjuk.

```
// frakablak.cpp
//
// Mandelbrot halmaz rajzoló
// Programozó Páternoszter
//
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, ←
// nbatfai@gmail.com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses>.
//
```

```
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) ←
// későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY ←
// CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1    Bár a Nokia Qt SDK éppen tartalmaz egy Mandelbrotos példát, ←
//           de
// ezt nem tartottam megfelelőnek első Qt programként ajánlani, mert ←
// elég
// bonyolult: használ kölcsönös kizárást stb. Ezért "from scratch" ←
// megírtam
// egy sajátot a Javát tanítokhoz írt dallamára:
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
//
```



```
#include "frakablak.h"

FrakAblak::FrakAblak(double a, double b, double c, double d,
                      int szelesseg, int iteraciosHatar, QWidget *parent ←
)
: QMainWindow(parent)

{
    setWindowTitle("Mandelbrot halmaz");

    int magassag = (int)(szelesseg * ((d-c) / (b-a)));

    setFixedSize(QSize(szelesseg, magassag));
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);

    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
                           iteraciosHatar, this);
    mandelbrot->start();

}

FrakAblak::~FrakAblak()
```

```
{  
    delete fraktal;  
    delete mandelbrot;  
}  
  
void FrakAblak::paintEvent(QPaintEvent*) {  
    QPainter qpainter(this);  
    qpainter.drawImage(0, 0, *fraktal);  
    qpainter.end();  
}  
  
void FrakAblak::vissza(int magassag, int *sor, int meret, int hatar)  
{  
    for(int i=0; i<meret; ++i) {  
        //           QRgb szin = qRgb(0, 255-sor[i], 0);  
        QRgb szin;  
        if(sor[i] == hatar)  
            szin = qRgb(0, 0, 0);  
        else  
            szin = qRgb(  
                255-sor[i],  
                255-sor[i]%64,  
                255-sor[i]%16 );  
  
        fraktal->setPixel(i, magassag, szin);  
    }  
    update();  
}
```

A frakszalban a szükséges változókat állítjuk és itt zajlik a halmazt létrehozó számítás.

```
// frakszal.cpp  
//  
// Mandelbrot halmaz rajzoló  
// Programozó Páternoszter  
//  
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, ←  
// nbatfai@gmail.com  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.
```

```
//  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <http://www.gnu.org/licenses/>.  
//  
// Ez a program szabad szoftver; terjeszthető illetve módosítható a  
// Free Software Foundation által kiadott GNU General Public License  
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi  
// változata szerint.  
//  
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,  
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA  
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.  
// További részleteket a GNU General Public License tartalmaz.  
//  
// A felhasználónak a programmal együtt meg kell kapnia a GNU General  
// Public License egy példányát; ha mégsem kapta meg, akkor  
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.  
//  
//  
// Version history:  
//  
// 0.0.1 Bár a Nokia Qt SDK éppen tartalmaz egy Mandelbrotos példát, de  
// ezt nem tartottam megfelelőnek első Qt programként ajánlani, mert elég  
// bonyolult: használ kölcsönös kizárást stb. Ezért "from scratch" megírtam  
// egy sajátot a Javát tanítokhoz írt dallamára:  
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1  
//  
  
#include "frakszal.h"  
  
FrakSzal::FrakSzal(double a, double b, double c, double d,  
                    int szelesseg, int magassag, int iteraciosHatar, FrakAblak *frakAblak)  
{  
    this->a = a;  
    this->b = b;  
    this->c = c;  
    this->d = d;  
    this->szelesseg = szelesseg;  
    this->iteraciosHatar = iteraciosHatar;  
    this->frakAblak = frakAblak;  
    this->magassag = magassag;  
  
    egySor = new int[szelesseg];
```

```
}

FrakSzal::~FrakSzal()
{
    delete[] egySor;
}

// A szál kódját a Javát tanítokhoz írt Java kódomból vettetem át
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
// mivel itt az algoritmust is leírtam/lerajzoltam, így meghagytam
// a kommenteket, hogy a hallgató könnyen hozzáolvashassa az "elméletet ←
",
// ha érdekli.

void FrakSzal::run()
{
    // A [a,b]x[c,d] tartományon milyen sűrű a
    // megadott szélesség, magasság háló:
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság hálót:
    for(int j=0; j<magassag; ++j) {
        //sor = j;
        for(int k=0; k<szelesseg; ++k) {
            // c = (reC, imC) a háló rácspontjainak
            // megfelelő komplex szám
            reC = a+k*dx;
            imC = d-j*dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while(reZ*reZ + imZ*imZ < 4 && iteracio < iteraciosHatar) {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ*reZ - imZ*imZ + reC;
                ujimZ = 2*reZ*imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;

                ++iteracio;
            }
        }
    }
}
```

```
        }
        // ha a < 4 feltétel nem teljesült és a
        // iteráció < iterációsHatár sérülésével lépett ki, azaz
        // feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c
        // sorozat konvergens, azaz iteráció = iterációsHatár
        // ekkor az iteráció %= 256 egyenlő 255, mert az esetleges
        // nagyítások során az iteráció = valahány * 256 + 255

        //a színezést viszont már majd a FrakAblak osztályban lesz
        egySor[k] = iteracio;
    }
    // Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.
    frakAblak->vissza(j, egySor, szelesseg, iteraciosHatar);
}
}
```

5.6. Mandelbrot nagyító és utazó Java nyelven

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

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

```
/*
 * MandelbrotHalmazNagyito.java
 *
 * DIGIT 2005, Javat tanitok
 * Batfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Batfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.2
 */
public class MandelbrotHalmazNagyito extends MandelbrotHalmaz {
    /**
     * A nagyítando kijelolt teruletet bal felső sarka.
     */
    private int x, y;
    /**
     * A nagyítando kijelolt terulet szelessége és magassága.
     */
    private int mx, my;
    /**
     * Letrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszamolo és nyitani tudo
     * <code>MandelbrotHalmazNagyito</code> objektumot.
}
```

```
* @param      a          a [a,b]x[c,d] tartomany a koordinataja.
* @param      b          a [a,b]x[c,d] tartomany b koordinataja.
* @param      c          a [a,b]x[c,d] tartomany c koordinataja.
* @param      d          a [a,b]x[c,d] tartomany d koordinataja.
* @param      szelesseg   a halmazt tartalmazo tomb szelessege.
* @param      iteraciosHatar a szamitas pontossaga.
*/
public MandelbrotHalmazNagyito(double a, double b, double c, double d,
                                 int szelesseg, int iteraciosHatar) {
    // Az os osztaly konstruktoranak hivasa
    super(a, b, c, d, szelesseg, iteraciosHatar);
    setTitle("A Mandelbrot halmaz nagyitasai");
```

A a nagyítandó területet a bal egérgombbal tudjuk kijelölni. Az egér kattintásait egy listener segítségével dolgozzuk fel. Majd megnézzük hol van a mutató pozíciója. A kijelölő téglalap bal felső sarkának a koordinátái az x és y, az mx és my a téglalap magassága és szélessége. A mousepressed() függvényben ezeket az adatokat kérjük le. Az mx és my-t 0-ra állítjuk és majd akkor fogjuk növelni ha a téglalapot pakoljuk. A pakolás mértékét egy egér mozgását érzékelő listener-rel kapjuk meg. Ha felengedjük az egeret újrakezdi a számolást és csinál egy új objektumot. Ha kivonjuk téglalap x és y koordinátáiból az egérmutató aktuális helyzetét akkor megkapjuk a kirajzolandó téglalap méreteit.

```
addMouseListener(new java.awt.event.MouseAdapter() {
    // Eger kattintassal jeloljuk ki a nagyitando teruletet
    // bal felső sarkat vagy ugyancsak eger kattintassal
    // vizsgaljuk egy adott pont iteracioit:
public void mousePressed(java.awt.event.MouseEvent m) {
    // Az egermutato pozicioja
    x = m.getX();
    y = m.getY();
    // Az 1. eger gombbal a nagyitando terulet kijeloleset
    // vegezzuk:
    if(m.getButton() == java.awt.event.MouseEvent.BUTTON1 ) {
        // A nagyitando kijelolt teruletet bal felső sarka: (x, ←
        y)
        // es szelessege (majd a vonszolas noveli)
        mx = 0;
        my = 0;
        repaint();
    } else {
        // Nem az 1. eger gombbal az egermutato mutatta c
        // komplex szambol inditott iteraciokat vizsgalhatjuk
        MandelbrotIteraciok iteraciok =
            new MandelbrotIteraciok(
                MandelbrotHalmazNagyito.this, 50);
        new Thread(iteraciok).start();
    }
}
```

```
// Vonszolva kijelölünk egy teruletet...
// Ha felengedjük, akkor a kijelölt terulet
// ujraszamítása indul:
public void mouseReleased(java.awt.event.MouseEvent m) {
    if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
        double dx = (MandelbrotHalmazNagyito.this.b
                    - MandelbrotHalmazNagyito.this.a)
                    /MandelbrotHalmazNagyito.this.szelesseg;
        double dy = (MandelbrotHalmazNagyito.this.d
                    - MandelbrotHalmazNagyito.this.c)
                    /MandelbrotHalmazNagyito.this.magassag;
        // Az új Mandelbrot nagyító objektum elkeszítése:
        new MandelbrotHalmazNagyito(
            MandelbrotHalmazNagyito.this.a+x*dx,
            MandelbrotHalmazNagyito.this.a+x*dx+mx*dx,
            MandelbrotHalmazNagyito.this.d-y*dy-my*dy,
            MandelbrotHalmazNagyito.this.d-y*dy,
            600,
            MandelbrotHalmazNagyito.this.iteraciosHatar)
        )
    }
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // A nagyítando kijelölt terulet szelessége és magassága:
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
}
```

Az alábbi kódcsípet mutatja be a pillanatkép készítő funkciót. Kirajzoljuk a területet jelző téglalapot és visszaadjuk az egérmutató akutális helyzetét. A main függvényben példányosítunk egy MandelBrotHalmazNagyito objektumot és kész is.

```
/**
 * Pillanatfelvetelek készítése.
 */
public void pillanatfelvetel() {
    // Az elmentendő kép elkeszítése:
    java.awt.image.BufferedImage mentKep =
        new java.awt.image.BufferedImage(szelesseg, magassag,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKep.getGraphics();
    g.drawImage(kep, 0, 0, this);
```

```
g.setColor(java.awt.Color.BLACK);
g.drawString("a=" + a, 10, 15);
g.drawString("b=" + b, 10, 30);
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iteraciosHatar, 10, 75);
if(szamitasFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
// A pillanatfelvetel kepfajl nevenek kepzese:
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyitas_");
sb.append(++pillanatfelvetelszamlalo);
sb.append("_");
// A fajl nevebe belevesszuk, hogy melyik tartomanyban
// talaltuk a halmazt:
sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");
// png formatumu kepet mentunk
try {
    javax.imageio.ImageIO.write(mentKep, "png",
        new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
    e.printStackTrace();
}
}
/***
 * A nagyitando kijelolt teruletet jelzo negyzet kirajzolasa.
 */
public void paint(java.awt.Graphics g) {
    // A Mandelbrot halmaz kirajzolasa
    g.drawImage(kep, 0, 0, this);
    // Ha eppen fut a szamitas, akkor egy voros
    // vonallal jeloljuk, hogy melyik sorban tart:
    if(szamitasFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    // A jelzo negyzet kirajzolasa:
```

```
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
    }
    /**
     * Hol all az egermutato?
     * @return int a kijelolt pont oszlop pozicioja.
     */
    public int getX() {
        return x;
    }
    /**
     * Hol all az egermutato?
     * @return int a kijelolt pont sor pozicioja.
     */
    public int getY() {
        return y;
    }
    /**
     * Peldanyosit egy Mandelbrot halmazt nagyito obektumot.
     */
    public static void main(String[] args) {
        // A kiindulo halmazt a komplex sik [-2.0, .7]x[-1.35, 1.35]
        // tartomanyaban keressuk egy 600x600-as haloval es az
        // aktualis nagyitasi pontossaggal:
        new MandelbrothalmazNagyito(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```

5.7. MALMO

Megoldás forrása: [Github](#)

Megoldás videó: [YouTube](#)

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

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

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndi8>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

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

Tanulságok, tapasztalatok, magyarázat...

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak (PICI)

A programozási nyelvek 3 szintjét különböztetjük meg: gépi nyelv->assembly szintű nyelv->magas szintű nyelvek.

A forráskódot magas szintű nyelvek (amelyeket szemantikai és szintaktikai szabályok összessége határoz meg) egyikével írjuk. Ezt a kódöt gépi kóddá kell alakítani, amit fordítóprogramos (pl.: C, C++) vagy interpreteres (pl.: Python) megoldással vagy minden kettővel (pl.: Java) végzünk.

A fordítóprogram a gépi kóddá alakítás mellett lexikális, szintaktikai és szemantikai elemzést végez.

Az interpreteres technika nem készít tárgyprogramot, hanem sorról sorra olvassa és hajtja végre a kódöt.

A programnyelvek szabványát hivatkozási nyelvnek nevezzük (ebben van definiálva a szemantikai és szintaktikai szabályok).

A szintaktika leírásához formalizmust míg a szemantika meghatározásához emberi nyelvet használnak.

Manapság IDE-k állnak rendelkezésünkre amelyek fejlesztői környezetek, ezekbe bele vannak építve például a szövegszerkesztők illetve fordítók is, például a Visual Studio Code (VSCode), PyCharm, PHPStorm.

Az imperatív programnyelvek közé tartoznak az algoritmikus nyelvek, amelyek működtetni fogják a processzort. A program utasítások sorozatából áll, legfontosabb eszközei a változók. Alcsoportjai az eljárás-orientált és az objektumorientált nyelvek.

A deklaratív programnyelvek nem kötődnek annyira a Neumann architektúrához, alcsoportjai a funkcionális nyelvek, illetve a logikai nyelvek.

A programok forrásszövegének a legkisebb alkotóelemei a karakterek.

Az adattípusok absztrakt programozási eszközök, amiket valamilyen konkrét programozási eszköz részeként használunk. Egy nyelvben általában előre vannak deklarálva a használható adattípusok (int, string, boolean...stb.), de néhány esetben lehetőség van saját létrehozására is pl.: C-ben. Az adattípusokat két nagy csoportra oszthatjuk: egyszerű (pl.: egész, valós, logikai) és összetett típusúak (pl.: tömb, rekord, mutató).

A konstansokat akkor használjuk, ha egy értéket többször használunk egy program során és nem szeretnénk, hogy megváltozzon az adott értéke.

A kifejezésekkel az adott értékeknek új értéket adunk a programban. Ezek lehetnek operandusok, változók, konstansok és függvényhívások, illetve operátorok. Szükség esetén zárüjeleket is használhatunk.

Az utasítások a programozási nyelvek nélkülözhetetlen elemei. Két csoportjuk a deklarációs és végrehajtandó utasítás.

A deklarációs utasításokkal változókat, konstansokat...stb. deklarálunk.

A végrehajtandó utasításoknak több csoportja létezik, ilyen a vezérlési szerkezetet megvalósító utasítások, pl.: if, switch, goto

Másik nagyobb csoportja a ciklus utasítások, ilyen például a for, a while és a do-while ciklus is.

A program tagolása nyelvenként változik. Vannak nyelvek, amelyekben külön-külön fordítható önálló részeket írhatunk meg, míg más nyelvekben egységekkel kell fordítanunk a kódot, illetve ezek kombinációja is létezik.

Az eljárásorientált nyelvekben megjelennek az alprogramok, amelyekben megírjuk, hogy minnek kell történnie és ezt a főprogramban hívjuk, ahányszor használni szeretnénk. Legtöbbször függvényeket és eljárásoakt írunk az alprogramokba. Függvény deklarálásnál meghatározzuk a típusát, nevét és a kért paramétereit. Az itt deklarált változók lokálisak tehát csak a függvényben használhatóak. A rekurzió azt jelenti, hogy egy programrészben egy másik programrész hívunk meg.

A paraméterátadás a programrészek közötti kommunikáció egy módja. Két résztvevője a hívó (akkármilyen programrész lehet) és a hívott (csak alprogram lehet). A C-ben az érték szerinti átadást használjuk. Az érték szerinti átadáskor a formális karakter megkapja az adott értéket, ilyenkor értékmásolás történik. Ez csak akkor lehetséges, ha az alprogramban a formális paraméternek van címe. Az érték átadódik és ezzel az alprogram a saját területén dolgozik, tehát az információ áramlás csak egyirányú. Hátránya, hogy lassú lehet nagyobb adatcsoporthoz másolásánál. C++-ban már megjelent a referencia szerinti átadás is.

A blokk egy program belsejében elhelyezkedő egység, amely tetszőleges utasításokat tartalmaz. A kezdetét és végét valamilyen speciális karakter jelöli. (pl.: {} vagy {}...stb.) A változóknak, konstansoknak, adatszerkezeteknek stb. hatásköre van (lokális vagy globális), ami azt jelenti, hogy a program mely részein értelmezhető, végezhető vele műveletek. Ennek beállítása statikusan és dinamikusan történhet. Statisztikus beállítás esetében a fordítóprogram dolgozik, végigmegy a kódon és egyes programrészekre megnézi lokális-e az adott név. Ezt addig csinálja amíg el nem érjük a legkülső részét a programnak. Ilyenkor kaphatunk hibát vagy deklarálódhat automatikusan a név. Az ilyen módú beállításnál az adott programrészben deklarált változók lokálisak. Ha nem ott deklaráltuk őket, de elérhetőek akkor pedig globálisak. A dinamikus megoldásban úgynevezett hívási láncot használunk. Az deklarált változók hatásköre a deklarálásuk helyének programrészre és minden olyan programrész, ami ezzel hívási láncban van. Az ilyen hatáskörkezelést a futtató rendszer végzi.

Az input-output kezelés a legtöbb programozási nyelvnél más, azonban a legtöbbre jellemző, hogy középpontjában az állomány áll. Ez lehet egy fizikai és logikai rész, amelyeket funkciója szerint csoportosítjuk. Az input állományból csak olvasni lehet és már létezik. Az output állomány nem létezik a létrehozása előtt és ebbe csak írni lehet. Az input-output állományok ennek keveréke, lehet olvasni-írni és létezik feldolgozás előtt, de a tartalmát megváltoztathatjuk. Az állományokból való munka több lépésből áll: a logikai részt deklarálni kell, majd megfeleltetni a fizikai állománnyal. Ezt követően tudjuk megnyitni, feldolgozni, majd lezárni. A C-ben az I/O műveletek alapból nem részei a nyelvnek, standard könyvtári függvényeket használunk hozzájuk.

A kivételkezelés arra szolgál, hogy a megszakításoknak a kezelését a program szintjén tudjuk végrehajtani. Egy megadott esemény után lép életbe, arra az eseményre reagálva. A bizonyos kivételek kezelése letiltható és engedélyezhető.

A programok szekvenciálisak tehát sorrendben hajták végre az utasításokat. Az erőforráshasználatnak megfelelően a programokat csoportosíthatjuk szálakra és folyamatokra. A folyamatok csak külön-külön míg a

szálak közösen birtokolhatják az adott erőforrásokat. A szálak egymással kommunikálnak, időközönként találkoznak és adatcserét hajthatnak végre és egymással versenyben vannak az erőforrásokért. Biztosítanunk kell, hogy ne használjanak egy időben adatokat pl. az egyik változtatja a másik felveszi értékül.

10.2. Programozás bevezetés (Keringhan)

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

C nyelvben a konzolra íratás a printf() függvényvel lehetséges.

Minden változót a használata előtt deklarálni kell, ami végrehajtandó utasítások előtt történik. A deklaráció leírja a változók tulajdonságait egy típus megadásával és az adott típusú változók felsorolásával.

Az int (integer) típus az egész értékeket jelöli, pl.: 3. A float a lebegőpontos értékeket, pl.: 3.5

További adattípusok a C nyelvben, a char (character), ami 1 byton tárol, vagy a double, ami kétszeres pontosságú float.

Ha while ciklus zárójelben lévő feltétel igaz, akkor a kapcsos zárójelben lévő kódblokk utasításai hajtódnak végre, majd ismét ellenőrzi a feltételt. Ez a ciklus addig kering, míg a feltétel hamis nem lesz.

A for ciklusra is elmondható ez a folyamat, viszont a változódeklarálás helye megváltozik.

Ezek mellett a do-while egy ellentét, mivel elsőnek lefut az itasítás, majd ellenőrzi a feltételt.

Az operátoroknak több csoportja van, például az aritmetikai operátorok (+,-,/,*,%), relációs operátorok (<,>,=,<=,>=), valamint logikai operátorok az és && illetve a vagy || operátorok. Fontos még megemlíteni a bitshift <<= operátort amelyet általában ciklusokon belül alkalmazunk.

A függvények rendelkeznek visszatérési értékkel, névvel, paraméterlistával és törzzsel. Emellett tartalmazznak egy return utasítást, amely befejezi a függvény működését, ezzel az értékkel tér vissza. Meg lehet adni visszatérés nélküli függvényt is, ami a void function.

10.3. Programozás (BME C++)

A C++ nyelv a C nyelv továbbfejlesztett változata. Tervezője Bjarne Stroustrup. Fő célja a C nyelvben bizonytalanságokat okozó megoldások helyett egy biztonságosabb alternatívát nyújtson.

Ha a C nyelvben egy üres paraméterlistájú függvényt deklarálunk, akkor C++-ban a függvényben egy void utasítást megadva, ekvivalens függvényt deklarálunk.

Amennyiben elhagyjuk a visszatérési típusát a függvénynek, akkor a C-ben az alapértelmezett visszatérési érték az int térne vissza, míg C++-ban hibaüzenetet dobna, hiszen kötelező megadni típust.

A main()-ben nem kötelező return utasítást adni, alapértelmezett, hogy 0-val tér vissza.

A C++-ban jelent meg a bool (boolean) típus, amely igaz vagy hamis lehet. C-ben ezt int vagy enum típusú kifejezésekkel lehetne reprezentálni, de ez a technika nehezebben olvashatóbb.

C++-ban ahol utasítás állhat, ott állhat változódeklaráció is.

A függvényeket C-ben a függvények nevei azonosítják, tehát 2 ugyanolyan nevű függvény nem deklarálható ugyanabban a hatáskörben. C++ esetén a függvényeket a neük illetve az argumentum listájuk együttesen

azonosítja a függvényeket, tehát létrehozhatunk több ugyanolyan nevű függvényt amíg az argumentumoknak különbözik. Fontos továbbá megjegyezni azt is, hogy két függvényt nem különböztethetünk meg csak visszatérési érték alapján.

A C-ről C++-ra való áttérést a következő könyvtábeli elemek segíthetik, amelyek olvashatóbbak, karbantartthatóbb kódokat illetve kevesebb hibalehetőséget hordoznak magukkal:

```
struct -> class  
malloc/free -> new/delete  
#define MAX 300 -> const int max = 300
```

10.3.1. Mobilprogramozás (Python könyv)

A Python egy interpretes nyelv, amely az elmúlt évek során emelkedett ki, népszerűsége vitathatatlan. Atyja Guido van Russom. A nyelv előnyei között van az, hogy nagyon gyorsan tanulható nyelv, mivel könnyen olvasható, hamar lehet vele komolyabb eredményeket elérni, a komplexebb problémák megoldására is képes, nagy a standard könyvtára, illetve kompatibilis más nyelveken megírt modulokkal. A Python akkor lehet jó választás, ha egy prototípus alkalmazást szeretnénk készíteni valamilyen algoritmus vagy elmélet tesztelésére. Számos területen kiemelkedik, ilyen az automatizálás, a scrapelés, a machine learning vagy akár szerveroldalon is.

Az utasítások végét nem jelöl ; hanem a sor végéig tart.

Kommentelni a sor elejére elhelyezett # után lehet.

A változó típusát itt nem kell megadnunk, ugyanis a program futásakor kitalálja a típust a hozzárendelt érték alapján.

Az adattípusok a következők lehetnek: int, float, complex, string, list, tuple, range, dict, set, frozenset, bool, bytes, bytearray, memoryview.

Változók értékkadására az = jelet használjuk, változókhöz hozzárendelhetünk egy objektumot, függvényt vagy akár típust. Ha nem vagyunk biztos egy már meglévő változónk típusában, a type(variable) függvényel kiírathatjuk.

A del kulcsszó törölhetünk egy változó hozzárendelést és a garbage collector fogja elvégezni a mögöttes objektum törlését.

A print metódus használatával írathatunk ki változót a konzolra, a tokenek elválasztására vesszőket alkalmazunk.

Más nyelvekhez hasonlóan itt is támogatja a Python az if elágazást, valamint jelen van a for illetve while ciklus használata.

Címkeket a label kulcsszóval készíthetünk a kód egyes részeiben, a goto paranccsal pedig a címkekhez tudunk ugrani a címke nevének megadásával.

A függvények deklarálására a def kulcsszót alkalmazzuk. A függvényeknek van visszatérési értékük, nevük és paraméterjeik.

A Python támogatja a megszokott objektumorientált eljárásokat, tehát deklarálhatunk osztályokat is, melyek példányai az objektumok valamint a függvények. Az osztályok öröklődése is megtalálható a nyelvben.. Az osztályoknak lehet egy speciális konstruktor tulajdonságú metódusa, az __init__. Ennek első paramétere a

self, vagyis maga a létrehozandó objektum. Ezentúl pedig további paramétereket is várhat a konstruktor. Tehát def __init__(self, param1, param2,...):

A Python is támogatja a kivételkezeléseket. Ebben az esetben a try kulcsszó után szerepel a kódblokk amelyben a kivételes helyzet előállhat, majd ezt az except blokk követi, erre csak hiba esetén kerül a vezérlés.

DRAFT

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPORG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

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