

# Univerzális programozás

---

**Írd meg a saját programozás tankönyvedet!**

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

**COLLABORATORS**

	<i>TITLE :</i>  Univerzális programozás	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY	Bátfai, Norbert Ã©s Koós, Zoltán	2019. március 31.

**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 <a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

# Tartalomjegyzék

<b>I. Bevezetés</b>	<b>1</b>
<b>1. Vízió</b>	<b>2</b>
1.1. Mi a programozás? . . . . .	2
1.2. Milyen doksikat olvassak el? . . . . .	2
1.3. Milyen filmeket nézzek meg? . . . . .	2
<b>II. Tematikus feladatok</b>	<b>3</b>
<b>2. Helló, Turing!</b>	<b>5</b>
2.1. Végtelen ciklus . . . . .	5
2.2. Lefagyott, nem fagyott, akkor most mi van? . . . . .	7
2.3. Változók értékének felcserélése . . . . .	8
2.4. Labdapattogás . . . . .	9
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . . .	12
2.6. Helló, Google! . . . . .	13
2.7. 100 éves a Brun téTEL . . . . .	15
2.8. A Monty Hall probléma . . . . .	16
<b>3. Helló, Chomsky!</b>	<b>18</b>
3.1. Decimálisból unárisba átváltó Turing gép . . . . .	18
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen . . . . .	19
3.3. Hivatalos nyelv . . . . .	19
3.4. Saját lexikális elemző . . . . .	21
3.5. l33t.l . . . . .	22
3.6. A források olvasása . . . . .	25
3.7. Logikus . . . . .	27
3.8. Deklaráció . . . . .	27

<b>4. Helló, Caesar!</b>	<b>30</b>
4.1. double ** háromszögmátrix . . . . .	30
4.2. C EXOR titkosító . . . . .	33
4.3. Java EXOR titkosító . . . . .	35
4.4. C EXOR törő . . . . .	36
4.5. Neurális OR, AND és EXOR kapu . . . . .	39
4.6. Hiba-visszaterjesztéses perceptron . . . . .	42
<b>5. Helló, Mandelbrot!</b>	<b>43</b>
5.1. A Mandelbrot halmaz . . . . .	43
5.2. A Mandelbrot halmaz a std::complex osztállyal . . . . .	47
5.3. Biomorfok . . . . .	51
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . . .	55
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . . .	59
5.6. Mandelbrot nagyító és utazó Java nyelven . . . . .	60
<b>6. Helló, Welch!</b>	<b>62</b>
6.1. Első osztályom . . . . .	62
6.2. LZW . . . . .	62
6.3. Fabejárás . . . . .	62
6.4. Tag a gyökér . . . . .	62
6.5. Mutató a gyökér . . . . .	63
6.6. Mozgató szemantika . . . . .	63
<b>7. Helló, Conway!</b>	<b>64</b>
7.1. Hangyaszimulációk . . . . .	64
7.2. Java életjáték . . . . .	64
7.3. Qt C++ életjáték . . . . .	64
7.4. BrainB Benchmark . . . . .	65
<b>8. Helló, Schwarzenegger!</b>	<b>66</b>
8.1. Ssoftmax Py MNIST . . . . .	66
8.2. Ssoftmax R MNIST . . . . .	66
8.3. Mély MNIST . . . . .	66
8.4. Deep dream . . . . .	66
8.5. Robotpszichológia . . . . .	67

<b>9. Helló, Chaitin!</b>	<b>68</b>
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .	68
9.2. Weizenbaum Eliza programja . . . . .	68
9.3. Gimp Scheme Script-fu: króm effekt . . . . .	68
9.4. Gimp Scheme Script-fu: név mandala . . . . .	68
9.5. Lambda . . . . .	69
9.6. Omega . . . . .	69
<b>10. Helló, Gutenberg!</b>	<b>70</b>
10.1. Juhász István - Magas szintű programozási nyelvek 1 . . . . .	70
10.2. Programozás bevezetés . . . . .	70
10.3. Programozás . . . . .	71
<b>III. Második felvonás</b>	<b>72</b>
<b>11. Helló, Arroway!</b>	<b>74</b>
11.1. A BPP algoritmus Java megvalósítása . . . . .	74
11.2. Java osztályok a Pi-ben . . . . .	74
<b>IV. Irodalomjegyzék</b>	<b>75</b>
11.3. Általános . . . . .	76
11.4. C . . . . .	76
11.5. C++ . . . . .	76
11.6. Lisp . . . . .	76

# Ábrák jegyzéke

3.1. Turing gép . . . . .	18
4.1. A double ** háromszögmátrix a memóriában . . . . .	32
4.2. AND OR . . . . .	40
4.3. exor . . . . .	41
5.1. A Mandelbrot halmaz a komplex síkon . . . . .	43

DRAFT

# 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

# Előszó

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

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

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

## Hogyan forgasd

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

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

## Milyen nyelven nyomjuk?

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

**Mi a programozás?**

**Milyen doksikat olvassak el?**

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

**Milyen filmeket nézzek meg?**

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

## 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!

### Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

A loop\_1core.c program futtatása 100 százalékban kihasznál egy magot.

```
int main()
{
    while(1) {}
    return 0;
}
```

Ha erről magunk is meg szeretnénk győződni akkor a **top -p `pgrep -u username loop1\_core`** parancsot kell beírnunk egy külön terminálba, hogy lássuk a várt eredményt.

```
Tasks:      1 total,      1 running,      0 sleeping,      0 stopped,      0 zombie
%Cpu0 : 100,0 us,   0,7 sy,   0,0 ni,  24,6 id,   0,0 wa,   0,0 hi,   0,3 si,   ←
          0,0 st
%Cpu1 :  8,3 us,   2,4 sy,   0,0 ni,  62,5 id,   0,0 wa,   0,0 hi,   0,0 si,   ←
          0,0 st
KiB Mem : 3942056 total,   216336 free,   2236064 used,   1489656 buff/cache
KiB Swap: 999420 total,   996336 free,      3084 used.  1123772 avail Mem

PID  USER      PR  NI      VIRT      RES      SHR S %CPU %MEM     TIME+  ←
COMMAND
32098  kewo      20     0      4220      628      560 R 100,0  0,0      4:02.84  ←
loop_1core
```

Ha azt szeretnénk elérni, hogy egy magot 0 százalékban terheljen, akkor meg kell hívunk a sleep(n) metódust ami lelassítja az iterálás sebességét.

```
int main()
{
    while(1)
    {
        sleep(1);
    }
    return 0;
}
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
32243	kewo	20	0	4220	632	564	S	0,0	0,0	0:00.01	loop_0core

Minden mag 100 százalékos kihasználását szálasítással lehet elérni az OpenMP segítségével.

```
#include <omp.h>
int main()
{
    #pragma omp parallel
    {
        while(1) {}
    }
    return 0;
}
```

**gcc loop\_all\_cores.c -o loop\_all\_cores -fopenmp** parancsal fordítjuk le, majd futtatjuk.

```
Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie
%Cpu0 : 100,0 us, 1,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,7 si, 0,0 st
%Cpu1 : 100,0 us, 2,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 3942056 total, 233304 free, 2368576 used, 1340176 buff/cache
KiB Swap: 999420 total, 995056 free, 4364 used. 943384 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
32728	kewo	20	0	18968	1532	1408	R	184,3	0,0	0:25.05	loop_all_c+

## Lefagyott, nem fagyott, akkor most mi van?

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

Megoldás videó:

Megoldás forrása:

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

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

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

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

```
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.

Mondjuk azt, hogy létezik olyan program T100, ami el tudja dönteni egy másik programról, hogy van-e benne végtelen ciklus. Ha van, akkor megáll a program. Ellenkező esetben pedig végtelen ciklusba kezd. Létrehozunk egy új programot T1000 az előzőt felhasználva és ha T100 megállt, akkor végtelen ciklusba kezd, ha pedig az T100 kezdett végtelen ciklusba, akkor megáll. Mi lesz a végeredménye annak, ha önmagára hívjuk meg ezt a programot? Csak akkor áll le a program, ha a második argumentumként kapott saját programunk megáll. Ez viszont ellentmondáshoz vezet. Ezért nem lehetséges ilyen programot írni.

## Változók értékének felcserélése

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

Megoldás videó: [https://bhaxor.blog.hu/2018/08/28/10\\_begin\\_goto\\_20\\_avagy\\_elindulunk](https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk)

Megoldás forrása:

Ezzel a programmal két változó értékét a különbségük kihasználásával segédváltozó nélkül cseréljük meg.

```
#include<stdio.h>
int main()
{
    int firstNumber = 2, secondNumber = 4;

    printf("1. number = %d\n2. number = %d\n", firstNumber, secondNumber);

    firstNumber = ( firstNumber - secondNumber );
    secondNumber = ( firstNumber + secondNumber );
    firstNumber = ( secondNumber - firstNumber );

    printf("1. number = %d\n2. number = %d\n", firstNumber, secondNumber ↔
          );

    return 0;
}
```

## Labdapattogás

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

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

Megoldás forrása:

A getmaxyx függvénytel meghatározzuk a terminál ablakunk méretét. Ezután kiíratjuk a labdát a kezdőpozícióra, majd addig növeljük a pozícióértéketet amíg el nem érjük az ablak széleit, tetejét vagy alját. Ekkor negáljuk azt az értéket amelyet hozzáadunk a hozzárendelt pozícióértékkel. Ez persze minden ciklusban van, tehát a folyamatosan fut a programunk.

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

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

    int x = 0;
    int y = 0;
```

```
int xnov = 1;
int ynov = 1;

int mx;
int my;

for ( ; ) {

    getmaxyx ( ablak, my , mx );

    mvprintw ( y, x, "O" );

    refresh ();
    usleep ( 100000 );

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

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

return 0;
}
```

If ek nélkül is megoldható ez a feladat. Először is létrehozzuk a gotoxy függvényt amelyel a kurzort fogjuk pozicionálni. Ezután a main függvényben megadjuk a labda kezdeti helyzetét és a pálya méretét. Meghatározzuk a pálya széleinek helyzetét és már indulhatunk is. Egy végtelen ciklusba kiíratjuk a labda pozícióját és meghívjuk a gotoxy-t ami kiírja nekünk a labdát. Folyamatosan töröljük a terminál ablakát és újra kiíratjuk a labda új helyzetét.

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

static void gotoxy(int x, int y) //kurzor pozicionálása
```

```
{  
    int i;  
    for(i=0; i<y; i++) printf("\n");           //lefelé tolás  
    for(i=0; i<x; i++) printf(" ");             //jobbra tolás  
    printf("o\n");     //labda ikonja  
  
}  
  
void usleep(int);  
int main(void)  
{  
  
    int egyx=1;  
    int egyy=-1;  
    int i;  
    int x=10;    //a labda kezdeti pozíciója  
    int y=20;  
    int ty[23]; //magasság      // a pálya mérete  
    int tx[80]; //szélesség  
  
    //pálya széleinek meghatározása  
  
    for(i=0; i<23; i++)  
        ty[i]=1;  
  
    ty[1]=-1;  
    ty[22]=-1;  
  
    for(i=0; i<79; i++)  
        tx[i]=1;  
  
    tx[1]=-1;  
    tx[79]=-1;  
  
  
    for(;;)  
    {  
  
        //címsor és pozíció kijelzése  
        for(i=0; i<36; i++)  
            printf("_");  
  
        printf("x=%2d", x);  
        printf("y=%2d", y);  
  
        for(i=0; i<=35; i++)  
            printf("_");
```

```
(void)gotoxy(x,y);
//printf("o\n"); Áthelyezve a gotoxy függvénybe

x+=egyx;
y+=egyy;

egyx*=tx[x];
egyy*=ty[y];

usleep (200000);
(void)system("clear");
}

}
```

## Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
int main()
{
    int word = 1;
    int length = 0;

    do
    {
        length++;
    }
    while (word<<=1);

    printf("A word %d bites\n", length);

    return 0;
}
```

Bitshifteléssel fogjuk meghatározni a típus méretét és azt, hogy hány bitet foglal. A while ciklusunkban folyamatosan shiftelünk egyet balra a biteken, eközben növeljük a length változó értékét. Ezt addig csinál-

jur amíg csupa 0 bitet nem fog tartalmazni a word változónk. Az int típusú változók 4 byte-on tárolódnak, ezért 32 bites lesz a word.

```
kewo@kewo-X540SA:~/Desktop/programs$ gcc lengthofword.c -o word  
kewo@kewo-X540SA:~/Desktop/programs$ ./word  
A word 32 bites
```

## Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Kezdéskor minden oldalnak van egy szavazati pontja és ha az egyik linkeli a másikat, akkor a linkelt oldal megkapja a linkelő pontját. Kereséskor annál előrébb lesz egy oldal, minél több oldal linkel rá és az is számít, hogy a rá linkelő oldalakra mennyi másik oldal linkel.

```
#include <stdio.h>  
#include <math.h>  
  
void  
kiir (double tomb[], int db) {  
  
    int i;  
  
    for (i=0; i<db; ++i){  
        printf("%f\n",tomb[i]);  
    }  
}  
  
double  
tavolsag (double PR[], double PRv[], int n) {  
  
    int i;  
    double osszeg=0;  
  
    for (i = 0; i < n; ++i)  
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);  
  
    return sqrt(osszeg);  
}  
  
void  
pagerank(double T[4][4]) {  
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény
```

```
double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0}; //ezzel szorzok

int i, j;

for(;;) {

    // ide jön a mátrix művelet

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

    if (tavolsag(PR, PRv, 4) < 0.0000000001)
        break;

    // ide meg az átpakolás PR-ből PRv-be

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

kiir (PR, 4);
}

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

    double L1[4][4] = {
        {0.0, 0.0, 1.0/3.0, 0.0},
        {1.0, 1.0/2.0, 1.0/3.0, 0.0},
        {0.0, 1.0/2.0, 0.0, 0.0},
        {0.0, 0.0, 1.0/3.0, 0.0}
    };

    double L2[4][4] = {
        {0.0, 0.0, 1.0/3.0, 0.0},
        {1.0, 1.0/2.0, 1.0/3.0, 0.0},
        {0.0, 1.0/2.0, 0.0, 0.0},
        {0.0, 0.0, 1.0/3.0, 1.0}
    };
}
```

```
printf("\nAz eredeti mátrix értékeivel történő futás:\n");
pagerank(L);

printf("\nAmikor az egyik oldal semmire sem mutat:\n");
pagerank(L1);

printf("\nAmikor az egyik oldal csak magára mutat:\n");
pagerank(L2);

printf("\n");

return 0;
}
```

Az L tömbben tároljuk a linkelésből szerzett adatokat, melyik oldalra melyik oldal linkel és mennyit. A végtelen ciklusban nullázzuk PR összes elemét, majd hozzáadjuk az L mátrix és PRv vektor szorzatainak értékét. Ezután a tavolság függvényel végigmegyünk a PR és PRv vektorokon és egy változóban eltároljuk ezek különbségének a négyzetét. Ezután gyököt vonva visszaadjuk az értéket, amely ha kisebb mint 0.00000001, akkor kiléünk a végtelen ciklusból, ellenkező esetben pedig PRv tömböt feltöljük PR elemeivel és kiiratjuk az értékeket.

## 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](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R)

A prímszám olyan természetes szám, ami csak önmagával és eggyel osztható és minden természetes szám előállítható prímszámok szorzataként. A prímszámok végtelen felé tartanak.

A ikerprímek olyan prímszámok, amelyek egymás után következnek és különbségük 2.

Brun téTEL: az ikerprímszámok reciprokából képzett összeg konvergál egy számhoz. Ez a határ a Brun konstans.

A Brun téTEL ezzel a programmal tudjuk bemutatni:

```
sumTwinPrimes <- 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 = sumTwinPrimes)
plot(x,y,type="b")
```

Létrehozunk egy függvényt aminek megadjuk, hogy meddig számolja az ikerprímszámokat. A primes(x) függvény álltal generáljuk a prímszámokat majd elmentjük őket primes változóba. Az egymásután következő prímszámok különbségét diff be tároljuk. Ahol ikerprímeket találunk, azaz a diff=2, elmentjük idx be.

A t1primes be kerülnek az ikerprímek első párja, a t2primes be pedig a második. Az rt1plust2 be tároljuk minden párnak a reciprokainak az összegét, majd ezeket összeadjuk és végül visszaadjuk ezt az értéket.

## A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/03/erdos\\_pal\\_mit\\_keresett\\_a\\_nagykonyvben\\_a\\_monty\\_hall-paradoxon\\_kapcsan](https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan)

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

A versenyzőnek ki kell választani a három ajtó közül egyet és a nyereménye az, amit az ajtó rejt. Az egyik ajtó mögött egy értékes nyeremény van, a másik két ajtó mögött pedig semmi. A versenyző választ egy ajtót, majd ezután a műsorvezető kinyit egy olyan ajtót, ami mögött semmi sincs. Ezek után a műsorvezető megkérdezi a játékosat, hogy kitart-e az eredeti választása mellett, vagy átvált a másik ajtóról.

Kezdetben a játékos 1 a 3-hoz valószínűsséggel választja ki a 3 ajtó közül azt, ami mögött az értékes nyeremény van és 2 a 3-hoz valószínűsséggel választ olyan ajtót, ami mögött semmi sincs. Ezután a műsorvezető egy olyan ajtót nyit ki, ami mögött tudja, hogy semmi sincs. A kérdés, hogy ezek után mekkora eséllyel van a játékos által kiválasztott ajtó mögött a nyeremény és mekkora eséllyel van a másik megmaradt ajtó mögött.

A következő programmal fogjuk demonstrálni a megoldást:

```
kiserletek_szama=1000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)
for (i in 1:kiserletek_szama) {
  if(kiserlet[i]==jatekos[i]) {

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

  } else{

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


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

```
}  
nemvaltoztatesnyer= which(kiserlet==jatekos)  
valtoztat=vector(length = kiserletek_szama)  
for (i in 1:kiserletek_szama) {  
    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
    valtoztat[i] = holvalt[sample(1:length(holvart),1)]  
  
}  
valtoztatesnyer = which(kiserlet==valtoztat)  
sprintf("Kiserletek szama: %i", kiserletek_szama)  
length(nemvaltoztatesnyer)  
length(valtoztatesnyer)  
length(nemvaltoztatesnyer)/length(valtoztatesnyer)  
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

DRAFT

### 3. fejezet

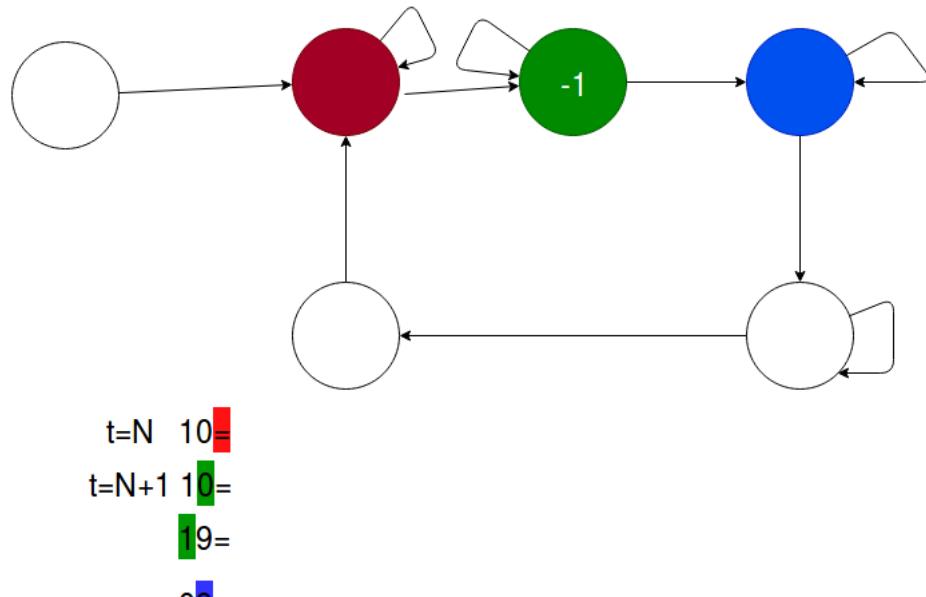
## Helló, Chomsky!

### Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:



3.1. ábra. Turing gép

Ez a turing gép végigmegy a beolvasandó adatokon, ha egyenlőség jelet talál , akkor kivon egyet az előtte lévő számból. Ez a folyamat addig ismétlődik amíg az előtte lévő szám le nem nullázódik. Mivel itt 0 van, de előtte 1 van, ezért a 0-ból 9 lesz és az 1-ből 0. A kivont egyeseket kiírja a tárolóra, ezért a végén egy egyeskből álló sorozatot kapunk. Az ebben a sorozatban lévő egyesek száma megegyezik a kiindulási szám értékével.

## Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

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

Megoldás videó:

Megoldás forrása:

változók: A, B, C

konstansok: a, b, c

képzési szabályok:

$A \rightarrow aAB$ ,  $A \rightarrow aC$ ,  $CB \rightarrow bCc$ ,  $cB \rightarrow Bc$ ,  $C \rightarrow bc$

A kezdő szimbólum az A.

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

1-es típusú (környezetfüggő) nyelvtan, 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 S nem szerepel egyetlen szabály jobb oldalán sem.

változók: S, X, Y

konstansok: a, b, c

képzési szabályok:

$S \rightarrow abc$ ,  $S \rightarrow aXbc$ ,  $Xb \rightarrow bX$ ,  $Xc \rightarrow Ybcc$ ,  $bY \rightarrow Yb$ ,  $aY \rightarrow aaX$ ,  $aY \rightarrow aa$

A kezdő szimbólum az S.

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

## Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

```
<utasítás> ::=  
    <összetett_utasítás> | <feltételes_utasítás> | <iterációs_utasítás> |  
    <vezérlés_átadó_utasítás> | <címkézett_utasítás> | <kifejezés_utasítás> |<nulla_utasítás>  
<összetett_utasítás> ::= <deklaráció_lista> | <utasításlista>  
<deklaráció_lista> ::= <deklaráció>  
<utasításlista> ::= <utasítás>  
<feltételes_utasítás> ::= if | if else | switch  
<iterációs_utasítás> ::= while | do while | for  
<vezérlés_átadó_utasítás> ::= break | return | goto | continue  
<címkézett_utasítás> ::= <azonosító>  
<kifejezés_utasítás> ::= <kifejezés>  
<nulla_utasítás> ::= ;
```

```
#include <stdio.h>  
int main ()  
{  
    // asd  
    printf ("Hello World\n");  
}
```

C99-el lefordul:

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$ gcc hello.c -o hello -std= ←  
c99  
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$
```

C89-el nem fordul le:

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$ gcc hello.c -o hello -std= ←  
c89  
hello.c: In function 'main':  
hello.c:4:7: error: C++ style comments are not allowed in ISO C90  
    // asd  
    ^  
hello.c:4:7: error: (this will be reported only once per input file)  
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$
```

## Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

Ezt a programot lexer segítségével fogjuk megoldani. A lexer egy általunk megadott minta alapján fogja legenerálni a lexikális elemzőket.

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

Először is létrehozunk egy int változót amiben fogjuk számolni a valós számok előfordulását. Ezután pedig definiáljuk digit néven 0-9 ig a számjegyeket.

Az ezt követő részben adjuk meg a mintát. A digit után a csillag, azt jelenti, hogy digitből bármennyi lehet. A . azt jelenti, hogy bármilyen karakterre rá lehet illeszteni, azonban nekünk le kell védeni \ jellel és így a valós számoknál lévő pontot fogja értelmezni. Ezután megint jönnek a számjegyek, a + pedig azt jelzi, hogy legalább 1 számnak kell lennie a pont után. Ha van találat a mintára, akkor növeljük a realnumbers változót. Ezután kiiratjuk a felismert számot, illetve az atof al átkonvertált double verzióját is. Magában a programban meghívjuk a lexert majdpedig kiíratjuk a valós számok számát. Fordításkor hozzá kell csatolnunk a flex könyvtárat.

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$ lex -o valos_szam.c ←  
valos_szam.l  
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$ gcc valos_szam.c -o ←  
valos_szam -lfl
```

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/bhax_textbook/ ←
programs$ ./valos_szam
2324.342 23132 43333 342.232
[realnum=2324.342 2324.342000] [realnum=23132 23132.000000] [realnum=43333 ←
43333.000000] [realnum=342.232 342.232000]
The number of real numbers is 4
```

## I33t.I

Lexelj össze egy I33t ciphert!

Megoldás videó:

Megoldás forrása:

Az előző feladathoz hasonlóan a lexer egy általunk megadott minta alapján jogja létrehozni a lexikális elemzőket. Ebben a feldatban a begépelt szöveget leet nyelvre fogjuk konvertálni. A leet nyelv az online világ szlengje. Elsősorban üzenetek titkosítására szolgált, majd később a játékosok neveiben fordult leginkább elő.

```
/*
```

Fordítás:

```
$ lex -o 1337d1c7.c 1337d1c7.l
```

Futtatas:

```
$ gcc 1337d1c7.c -o 1337d1c7 -lfl
(kilépés az input vége, azaz Ctrl+D)
```

Copyright (C) 2019

Norbert Bátfai, batfai.norbert@inf.unideb.hu

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

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

```
You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>.
```

```
*/
```

```
% {
```

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

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

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

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

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

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

// https://simple.wikipedia.org/wiki/Leet
```

```
};

%}
%%

. {

    int found = 0;
    for(int i=0; i<L337SIZE; ++i)
    {

        if(l337d1c7[i].c == tolower(*yytext))
        {

            int r = 1+(int) (100.0*rand() / (RAND_MAX+1.0));

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

            found = 1;
            break;
        }
    }

    if(!found)
        printf("%c", *yytext);
}

%%

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

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$ ./l337d1c7
```

```
kewo
k3w0
Koos Zoltan
k00s >_0lt4n
```

## A források olvasása

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

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

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

### Bugok

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

i.

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

Ha a SIGINT jel kezelése nem volt figyelmen kívül hagyva, akkor a jelkezelő függvény kezelje, máskülönben figyelmen kívül legyen hagyva.

ii.

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

A for ciklussal 0-tól 4-ig megyünk, az i-t növelve. Az ++i megnöveli az i értékét majd visszaadja az 1-el megnövelt értéket.

iii.

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

A for ciklussal 0-tól 4-ig megyünk, az i-t növelve. Az i++ először visszaadja az eredeti értéket majd megnöveli az i értékét.

iv.

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

A for ciklussal 0-tól 4-ig megyünk, eközben tomb i edik elemét megváltoztatjuk arra az i-re ami megnövelés előtt volt.

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/programs$ splint wat4.c
Splint 3.1.2 --- 03 May 2009
```

```
wat4.c: (in function main)
wat4.c:12:28: Expression has undefined behavior (left operand uses i, ←
modified
                                by right operand): tomb[i] = i++
Code has unspecified behavior. Order of evaluation of function ←
parameters or
subexpressions is not defined, so if a value is used and modified in
```

```
different places not separated by a sequence point constraining ←  
evaluation  
order, then the result of the expression is unspecified. (Use - ←  
evalorder to  
inhibit warning)
```

Finished checking --- 1 code warning

v.

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

A for ciklussal 0-tól megyünk, addig amíg i kisebb mint n és a d mutató által mutatott tömb eleme egyenlő az s mutató által mutatott tömb elemével.

```
wat4.c:13:19: Right operand of && is non-boolean ( ←  
int): i < n && (*d++ = *s++)  
The operand of a boolean operator is not a boolean.
```

vi.

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

Kíratunk két egész számot. Mindkettőt az f függvény adja vissza. Először az f függvénynek átadjuk az a változót és az a változó 1-el megnövelt értékét. Másodszor pedig az f-nek, az a változó 1-el megnövelt értékét és a-t adjuk.

```
wat4.c:13:19: Argument 2 modifies a, used by argument 3 (order of ←  
evaluation of  
actual parameters is undefined): printf("%d %d", f(a, ++a), f(++a, ←  
a))  
Code has unspecified behavior. Order of evaluation of function ←  
parameters or  
subexpressions is not defined, so if a value is used and modified in  
different places not separated by a sequence point constraining ←  
evaluation  
order, then the result of the expression is unspecified. (Use - ←  
evalorder to  
inhibit warning)
```

vii.

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

Két egész számot iratunk ki, az egyik az f függvény a változó felhasználása által visszaadott szám, a másik pedig az a változó.

viii.

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

Két egész számot iratunk ki, az egyik az f függvény a változó memóriacímének felhasználásával visszaadott szám, a másik pedig az a változó.

Megoldás forrása:

Megoldás videó:

## Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/MatLog\\_LaTeX](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX)

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, [https://youtu.be/AJSXOQFF\\_wk](https://youtu.be/AJSXOQFF_wk)

A feladatban szereplő kifejezések jelentése:

```
-forall: univerzális kvantor  
-exist: egzisztenciális kvantor  
-wedge: implikáció  
-supset: konjukció  
-neg: negáció  
-S: rákövetkező
```

Ezeket a kifejezéseket felhasználva az alábbi mondatokat állíthatjuk elő:

1. minden x-hez létezik olyan y, amelynél ha x kisebb, akkor y prím.
2. minden x-hez létezik olyan y, amelynél ha x kisebb, akkor y prím, és  $\leftrightarrow$  ha y prím, akkor annak második rákövetkezője is prím.
3. létezik olyan y, amelyhez minden x esetén az x prím, és x kisebb, mint  $\leftrightarrow$  y.
4. létezik olyan y, amelyhez minden x esetén az x nagyobb, és x nem prím.

## Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referencia

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

```
#include <iostream>

int i = 0;
int a = 5;
int *b = &a;
int &r = a;
int c[5] = {0,1,2,3,4};
int (&tr)[5] = c;
int *d[5];
int *h();
int *( *l)();
int (*v( int c))(int a, int b);
int (*(*z)(int))(int, int);

int main()
{
    return 0;
}
```

Mit vezetnek be a programba a következő nevek?

- `int a;`  
Egy int típusú a nevű változó.
- `int *b = &a;`  
`b` egészre mutató mutató a-ra mutat.
- `int &r = a;`  
`r` egészre mutató mutató, ami a címet tartalmazza.

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

Öt elemű egészkből álló tömb.

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

Öt elemű egészkből álló tombre mutató mutató, ami a c tömbre mutat.

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

5 elemű egészre mutató mutatókból álló tömb.

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

Egy egéssel visszatérő paraméter nélküli függvényre mutató mutató.

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

Egy egészre mutató mutatóval visszatérő, paraméter nélküli függvényre mutató mutató.

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

Egy egéssel visszatérő 2 egészet váró függvényre mutató mutatóval visszatérő 2 egészet váró függvény.

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

Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre mutató mutató.

Megoldás videó:

Megoldás forrása:

## 4. fejezet

# Helló, Caesar!

### double \*\* háromszögmátrix

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

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

Megoldás forrása: [bhaxx/thematic\\_tutorials/bhaxx\\_textbook\\_IgyNeveldaProgramozod/Caesar/tm.c](https://bhaxx/thematic_tutorials/bhaxx_textbook_IgyNeveldaProgramozod/Caesar/tm.c)

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

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

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

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

    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");
}

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;

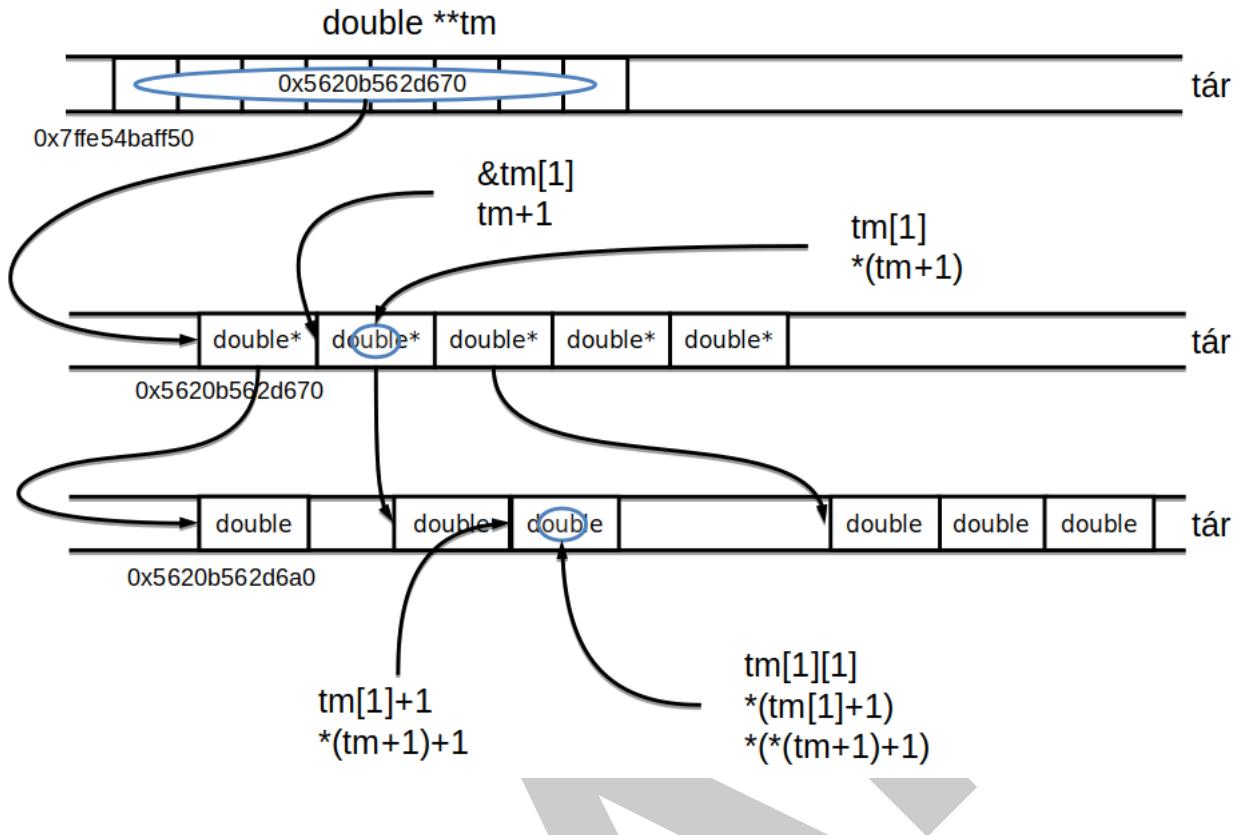
for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
}

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

free (tm);

return 0;
}
```





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

A előbbi kép a double \*\* háromszögmátrix memória foglalását ábrázolja. Az nr változóval adjuk meg a háromszögmátrix magasságát, ami itt öt lesz. A double \*\*tm egy mutatóra mutató mutató. A malloc-al memóriát foglalunk és egy mutatót kapunk vissza, ami a lefoglalt területre mutat (double \*). Ezt double \*\* mutatóvá típuskényszerítjük és a tm mutatókat ráállítjuk. A malloc megkapja a double \* mutató méretét ami 8 bájt, de ezt még megsorozzuk nr-el. Így végül 40 bájtot foglalunk a memóriában.

Ezután egy for ciklussal végigmegyünk ezen a területen 1-től 5-ig. minden double \* mutatót ráállítunk egy double típusú területre, aminek a darabszámát mindig egyet növeljük. Az első mutató 1db double-re mutat, a második 2-re és így tovább. Így kapjuk meg a háromszögmátrixot. Ezután értékeket adunk a változóinknak és kiiratjuk őket. Ha mindenkel készvagyunk akkor felszabadítjuk a lefoglalt területeket.

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/ ←
bhax_textbook/programs$ ./tm
```

```
0x7ffe2b00ea50
0xd20420
0xd20450
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
```

10.000000, 11.000000, 12.000000, 13.000000, 14.000000,

## C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

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

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

    while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
    {
        for (int i = 0; i < olvasott_bajtok; ++i)
        {

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

        }

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

}

Először definiáljuk a kulcs és a buffer méretét. Ezek konstansok tehát nem módosíthatóak. A forrás további részében a program ezen konstansok helyén , az előbb definiált értékeket fogja felhasználni. A main()-ben az argc-vel adjuk át az argumentumok számát, és az argumentumokra mutató mutatókat pedig az argv tömbben tároljuk el.

Létrehozunk két tömböt, az elsőbe a kulcsot a másikba a beolvasott karaktereket tesszük. A kulcs index változó mutatja a kulcs aktuális elemét, az olvasott\_bajtok pedig a beolvasott bajtok számát tárolja. Az strlen függvényel megnézzük a megadott kulcs méretét, majd elmentjük ezt a kulcs\_meret változóba. Ezután a strncpy függvényel átmásoljuk az argv[1]-ben tárolt sztringet a kulcs tömbbe, a MAX\_KULCS határozza meg, hogy mennyi karaktert másoljon át.

A while ciklus addig fut, amíg tudunk olvasni a bemenetről. A beolvasott adatokat a bufferbe tesszük. Végigmegyünk a beolvasott bajtokon és végrehajtjuk a kizárá vagy műveletet, majd kiírjuk az eredményt. Amint a beolvasott szöveg végéhez érünk, a read() függvény 0 értéket ad vissza és ekkor a ciklusunk véget ér.

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/ ←
bhax_textbook/programs$ ./e kulcs <tiszta.txt >titkos.txt
```

A hagymát megtisztítjuk és kockára vágjuk. Kevés olajat forróítunk egy lábasban, beletesszük a hagymát, üvegesre pirítjuk, hozzáadjuk a felkarikázott kolbászt, majd megpirítjuk. Lehúzzuk a lábast a tűzhelyről, megszórjuk a piros paprikával. Átkeverjük, majd hozzáadjuk a megmosott, megtisztított és kockára vágott burgonyát, és felöntjük annyi vízzel, ami majdnem ellepi. Magsózzuk, paradicsompürét teszünk bele, és addig főzünk, amíg a burgonya megpuhul. Friss kenyérrel, savanyú uborkával tálaljuk.

```
2e19 07a0 da18 0faf ce07 a8dc 1f69 794b
554c 4353 4b55 4c22 5303 140b 1a1e a8d4
1843 1e0e 1218 0a00 1101 afce 0701 0007
43b0 c206 4c08 1c08 1leaf c201 0a55 1aa0
d20c 1f19 085d 4b3e 0915 b0c2 064c 0c1f
0a1f 4c02 074b 1303 1101 a8c6 1fa0 de1f
0002 0853 0e12 1543 1fa8 d40e 0200 0914
024f 5309 1000 0607 0e06 1f19 b0d7 1e4c
0253 0314 0bla 1ea8 d418 4f53 a8c9 1a06
140e 061e 0653 1b1c 1ea0 de1f 1f19 085f
4b1d 0319 09a8 d40d 0719 1e1e 4c02 530d
1000 0812 191c 07a0 d211 1a18 1753 001a
0001 b0ca 0616 175f 4b18 0d09 174b 1809
0403 0207 afce 0701 0007 4d53 6155 4c43
534b 554c 433f 0e1d afd9 0911 0007 4312
```

```
4b19 afc2 110a 0618 4312 4b01 a9d2 0903
1000 1a01 aee4 004f 5306 100b 1009 a8c6
1e09 0600 550d 4303 0207 0310 030a 051e
0a18 a8d4 1a02 1f45 55af e207 0010 1a06
0101 b6d0 085f 4b18 0d09 174b 1d03 1909
a8d4 0d07 191e 1e4c 0253 0610 0b0e 1c18
1a18 175f 4b18 0904 0702 0616 17b0 c601
0317 074b b6c5 1053 001a 0f08 b0ca 070d
4305 a8d4 0b0c 071f 550e 1601 0c1a 021a
b0ca 0140 43b0 c206 4c05 1607 b6da 0d07
01b6 d008 530a 1b02 1a1a 4b03 afce 0911
1000 4f53 0a18 0543 1e0a 1f08 0d16 0655
090f 1f0e 0505 4d53 2610 0b10 b0d8 0f16
1618 4755 1c02 010a 1105 0000 0418 1ca0
cf19 b6c5 1753 1f10 1f19 b0d7 1b07 4311
0e19 094f 53a8 dc1f 4312 0f11 0504 530d
b0fd 1909 a8c9 074f 530a 18af ce14 4b14
4c01 0619 1203 0d0a 0a55 0106 141b 0004
161f 457f 4c43 534b 554c 4353 2d07 0510
004b 1e09 0d0a a8dc 1e11 1607 594c 1012
1d14 021a b0d1 5519 011c 191e afc2 050a
194c 17b0 ca19 0d0f 191e 1e42
```

## Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: [https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#exor\\_titkosito](https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#exor_titkosito)

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcsSzöveg,
                          java.io.InputStream bejövőCsatorna,
                          java.io.OutputStream kimenőCsatorna)
                          throws java.io.IOException {

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

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

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

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

    }

    kimenőCsatorna.write(buffer, 0, olvasottBájtok);

}

public static void main(String[] args) {

    try {

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

    } catch(java.io.IOException e) {

        e.printStackTrace();

    }

}

}
```

Létrehozunk egy ExorTitkosító nevű publikus osztályt. A paraméterként átadott kulcsot tároljuk egy stringben és létrehozzuk a bejövő és kimenő csatornákat. Tömbökben tároljuk a megadott kulcs bájtjait és a buffer méretét. A while ciklusban addig olvasunk, amíg -1-et nem kapunk. Alkalmazzuk a kizárá vagy műveletet, pont mint az előző feladatban és végül kiíratjuk a titkosított szöveget.

A try-ban tárhelyet foglalunk az ExorTitkosító függvénynek, melynek átadjuk a kulcsot, a bemenetet és a kimenetet. Ha nem adunk meg kulcsot, akkor hibát kapunk.

## C EXOR törő

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

Megoldás videó:

Megoldás forrása:

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

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

double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
    int sz = 0;
    for (int i = 0; i < titkos_meret; ++i)
        if (titkos[i] == ' ')
            ++sz;

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

    int kulcs_index = 0;

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

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

    }

}

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

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

return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

char kulcs[KULCS_MERET];
char titkos[MAX_TITKOS];
char *p = titkos;
int olvasott_bajtok;

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

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

// osszes kulcs eloallitasa
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);
}
```

```
// ujra EXOR-ozunk, így nem kell egy második buffer  
exor (kulcs, KULCS_MERET, titkos, p - titkos);  
}  
  
return 0;  
}
```

A t.c exor törő 0-tól 9-ig számokat tartalmazó 8 számjegyű kulccsal kódolt szöveget tör fel. A while ciklusban beolvassuk a titkos szöveget, majd a titkos bufferben nullázzuk a maradék helyet. Ezután az összes lehetséges kulcsot előállítjuk for ciklusokkal 0-tól 9-ig. Eközben alkalmazzuk a kulcsokon az exor műveletet a titkos szöveggel. Amint van találat kiíratjuk a kulcsot és a feltört szöveget. Ezután újra exorozunk, így nem kell egy második buffer.

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/bhax_textbook/ ←  
programs$ ./e 10000001 <tiszta.txt > titkos.txt  
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/bhax_textbook/ ←  
programs$ ./t <titkos.txt  
Kulcs: [10000001]  
Tiszta szöveg: A hagymát megtisztítjuk és kockára vágjuk. Kevés olajat ←  
forrósítunk egy lábasban, beletesszük a hagymát, üvegesre pirítjuk, ←  
hozzáadjuk a felkarikázott kolbászt, majd megpirítjuk.  
Lehúzzuk a lábast a tűzhelyről, megszörjük a piros paprikával. Átkeverjük, ←  
majd hozzáadjuk a megmosott, megtisztított és kockára vágott burgonyát, ←  
és felöntjük annyi vízzel, ami majdnem ellepi. Magsózzuk, ←  
paradicsompürét teszünk bele, és addig főzzük, amíg a burgonya megpuhul.  
Friss kenyérrel, savanyú uborkával tálaljuk.
```

## Neurális OR, AND és EXOR kapu

R

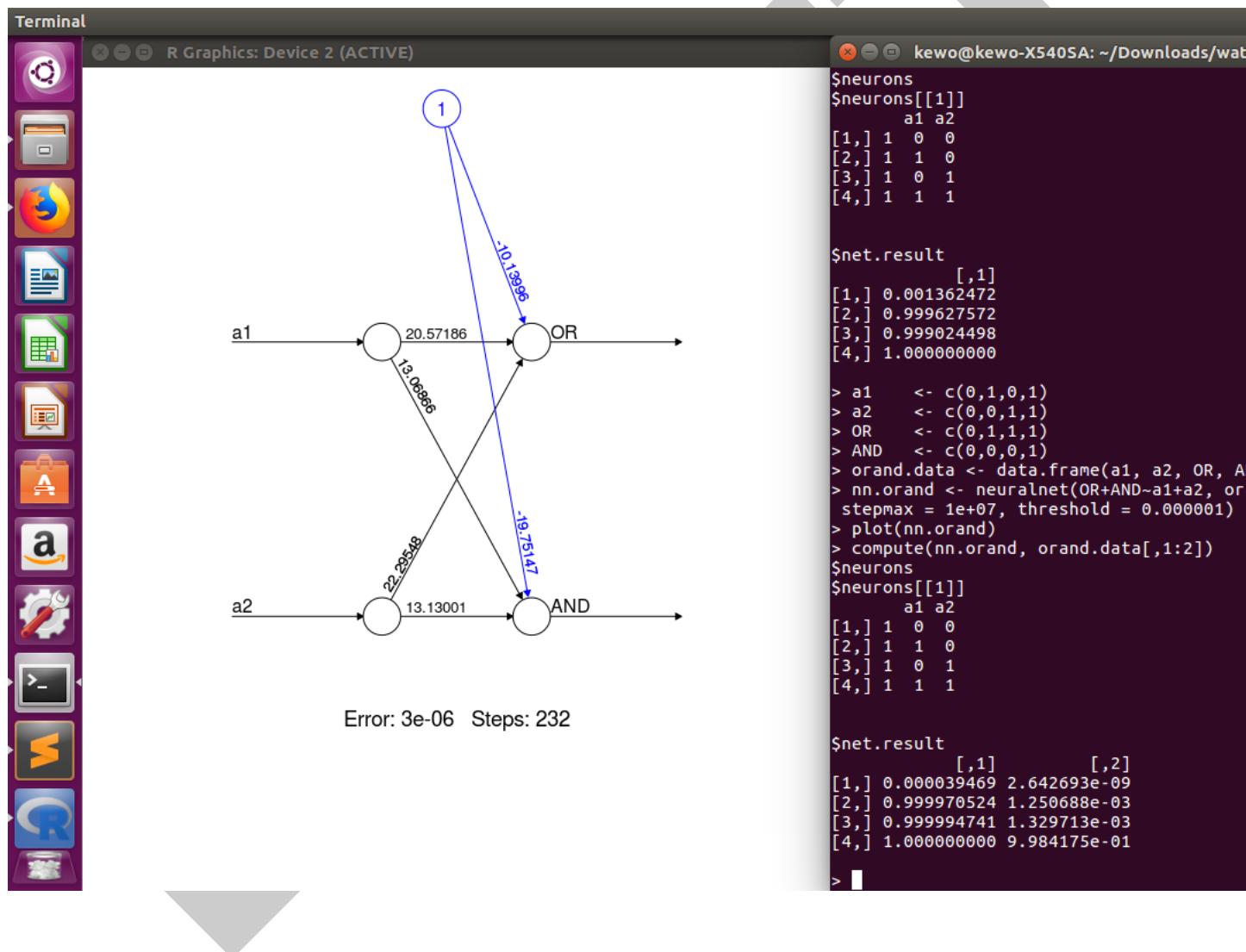
Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

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

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

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

Az a1-be és a2-be berakjuk az értékeket. Az OR-ba megadjuk mi lenne az eredménye, ha logikai vagy műveletet alkalmazunk az előbb megadott értékekre. Az AND-be pedig a logikai és művelet eredménye kerül. Ezzel két új szabályt vezetünk be. Ezek alapján elkezdi önmagát tanítani a program. Beállítjuk a határokat és a compute parancssal leellenőrizzük, hogy megfelelő eredményt adott vissza a program. Akkor sikerült a programnak megtanulnia a szabályt, ha pontos vagy nagyon közel a értékre jut az általunk megadott eredményhez képest.



4.2. ábra. AND|OR

```
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)

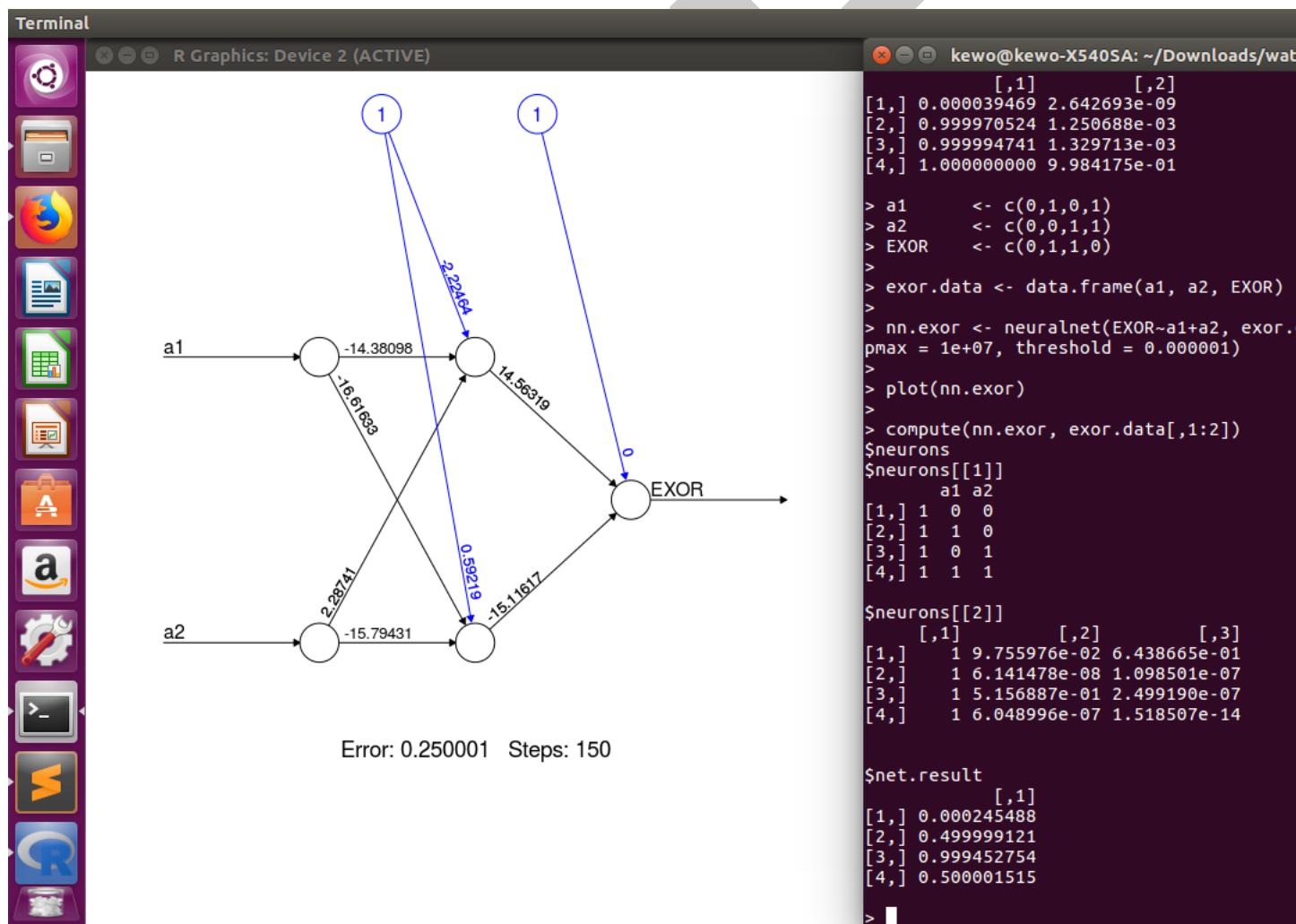
nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=2, linear.output=FALSE, ←
  stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

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

```

Az exor művelet esetében rejtett neuronokra lesz szükségünk, amik segítik a tanulást. Ha a neuralnet hidden értékét átállítjuk 2-re, akkor többrétegű neuronokkal tudjuk tanítani a programunkat. Ha segédneurononk nélkül tanítunk, akkor az eredmény és az általunk megadott kimenet közötti különbség a határértékeken kívül fognak esni.



4.3. ábra. exor

## Hiba-visszaterjesztéses perceptron

C++

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

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

A perceptron egy olyan algoritmus, amely képes betanítani a bináris osztályozást. Hasonlóan végzi a tanítást az előző feladathoz.

```
#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;
}
```

A mandel.cpp álltal generált kép rgb kódját betesszük a neurális háló inputjába. Létrehozunk egy három rétegű hálót és megadjuk, hogy hány darab neuront akarunk rakni az egyes rétegekbe. Definiálunk egy double\* pointert, amit ráállítunk egy size-zal megyegyező memóriaterületre. Ide bemásoljuk a beolvasott kép piros pixeleit. Meghívjuk a Perceptron class () operátortát, amivel megkapjuk az eredményt és ez a value-be kerül. Ezt az értéket pedig kiíratjuk.

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/bhax_textbook/ ←
programs$ g++ mlp.hpp main.cpp -o perc -lpng -std=c++11
```

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/bhax_textbook/ ←
programs$ ./perc mandel.png
0.508287
```

## 5. fejezet

# Helló, Mandelbrot!

### A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása: [bhax/attention\\_raising/CUDA/mandelpngt.cpp](bhax/attention_raising/CUDA/mandelpngt.cpp) nevű állománya.

5.1. ábra. A Mandelbrot halmaz a komplex síkon

A Mandelbrot halmazt 1980-ban találta meg Benoit Mandelbrot a komplex számsíkon. Komplex számok

azok a számok, amelyek körében válaszolni lehet az olyan egyébként értelmezhetetlen kérdésekre, hogy melyik az a két szám, amelyet összeszorozva -9-et kapunk, mert ez a szám például a 3i komplex szám.

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

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

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

```
// mandelpngt.c++
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063\_01\_parhuzamos\_prog\_linux
```

```
//  
//  https://youtu.be/gvaqijHlRUs  
//  
#include <iostream>  
#include "png++/png.hpp"  
#include <sys/times.h>  
  
#define MERET 600  
#define ITER_HAT 32000  
  
void  
mandel (int kepadat[MERET] [MERET]) {  
  
    // Mérünk időt (PP 64)  
    clock_t delta = clock ();  
    // Mérünk időt (PP 66)  
    struct tms tmsbuf1, tmsbuf2;  
    times (&tmsbuf1);  
  
    // számítás adatai  
    float a = -2.0, b = .7, c = -1.35, d = 1.35;  
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;  
  
    // a számítás  
    float dx = (b - a) / szelesseg;  
    float dy = (d - c) / magassag;  
    float reC, imC, reZ, imZ, ujreZ, ujimZ;  
    // Hány iterációt csináltunk?  
    int iteracio = 0;  
    // Végigzongorázzuk a szélesség x magasság rácsot:  
    for (int j = 0; j < magassag; ++j)  
    {  
        //sor = j;  
        for (int k = 0; k < szelesseg; ++k)  
        {  
            // c = (reC, imC) a rács csomópontjainak  
            // megfelelő komplex szám  
            reC = a + k * dx;  
            imC = d - j * dy;  
            // z_0 = 0 = (reZ, imZ)  
            reZ = 0;  
            imZ = 0;  
            iteracio = 0;  
            // z_{n+1} = z_n * z_n + c iterációk  
            // számítása, amíg |z_n| < 2 vagy még  
            // nem értük el a 255 iterációt, ha  
            // viszont 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;
}

kepadat[j][k] = iteracio;
}
}

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

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

}

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

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

int kepadat[MERET][MERET];

mandel(kepadat);

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

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

```
    255 -  
    (255 * kepadat[j][k]) / ITER_HAT ←  
    ,  
    255 -  
    (255 * kepadat[j][k]) / ITER_HAT ←  
    );  
}  
}  
  
kep.write (argv[1]);  
std::cout << argv[1] << " mentve" << std::endl;  
}
```

Elsőként kiszámoljuk a mandel metódusban az adatokat a mandelbrot halmaz képi megjelenítéséhez. Megmérjük mennyi idő alatt történik a számolás, majd megadjuk a számításhoz szükséges adatokat. Végigmegyünk a magasságon és szélességen, közben pedig kiszámítjuk minden rács csomóponthoz tartozó komplex számot. A while ciklusban addig amíg a  $|z_n|$  kisebb mint 2 vagy az iterációk száma nem érte el a 255-t, a  $z_{n+1} = z_n^2 + c$  iterációkat számoljuk. A ciklus lezárásakor a kiindulási  $c$  komplex számra az iteráció konvergens, tehát a  $c$  a mandelbrot halmaz eleme. Ezután kiíratjuk az eltelt időt és összeállítjuk a képet.

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/attention_raising/CUDA$ g++ ←  
mandelpngt.c++ -lpng -o mandelpngt  
kewo@kewo-X540SA:~/Downloads/wat/bhax/attention_raising/CUDA$ ./mandelpngt ←  
mandel.png  
4588  
45.8869 sec  
mandel.png mentve
```

## A Mandelbrot halmaz a `std::complex` osztályal

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

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

Megoldás forrása:

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó `bhax/attention_raising/Mandelbrot/3.1.2.cpp` nevű állománya.

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

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

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

std::cout << "Számítás\n";

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

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

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

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

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

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

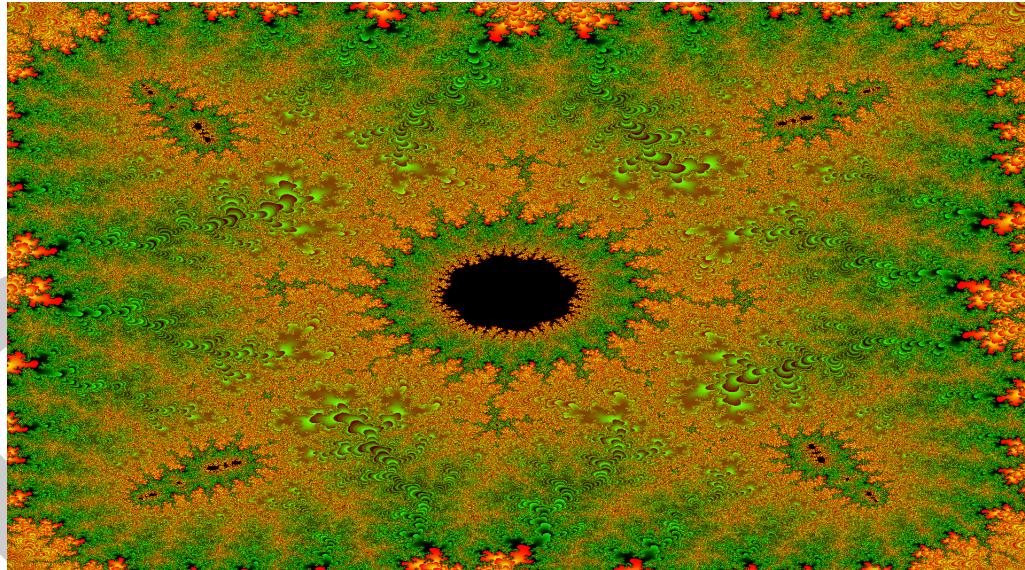
```
    }
```

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

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

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

C++-ban a complex osztály teszi lehetővé, hogy komplex számokkal tudjunk dolgozni. Pont mint az előző feladatban, itt is megadjuk a kép méretét, az iterációs határt és a számítás adatait. Ezután pedig az argumentumként kapott adatokkal fogunk dolgozni. Végigmegyünk a háló sorain és oszlopain, közben a complex osztály segítségével kiszámítjuk a háló rácspontokhoz (reC,imC) tartozó komplex számokat. Kiszámoljuk minden c esetén a z\_n-eket és amint elérjük az iterációs határt kilépünk a while ciklusból. A ciklus lezárá sakor a kiindulási c komplex számra az iteráció konvergens, tehát a c a mandelbrot halmaz eleme. Ezután ezt a pontot feketére színezzük és összeállítjuk a képet.



```
kewo@kewo-X540SA:~/Downloads/wat/bhax/attention_raising/Mandelbrot$ g++ ←
3.1.2.cpp -lpng -O3 -o 3.1.2
kewo@kewo-X540SA:~/Downloads/wat/bhax/attention_raising/Mandelbrot$ ./3.1.2 ←
mandel.png 1920 1080 2040 -0.01947381057309366392260585598705802112818 ←
-0.0194738105725413418456426484226540196687 ←
0.7985057569338268601555341774655971676111 ←
0.798505756934379196110285192844457924366
```

Szamitas

mandel.png mentve.

## Biomorfok

Megoldás videó: <https://youtu.be/IJMbgRzY76E>

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

A biomorfokra (a Julia halmazokat rajzoló bug-os programjával) rátaláló Clifford Pickover azt hitte természeti törvényre bukkant: [https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9\\_Iss5\\_2305--2315\\_Biomorphs\\_via\\_modified\\_iterations.pdf](https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf) (lásd a 2307. oldal aljától).

A különbség a **Mandelbrot halma**z és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a c változik, 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 < iteracionsHatar; ++i)
{
    z_n = std::pow(z_n, 3) + cc;
    if (std::real ( z_n ) > R || std::imag ( z_n ) > R)
    {
        iteracio = i;
        break;
    }
}
```

A bimorfos algoritmus pontos megismeréséhez ezt a cikket javasoljuk: [https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9\\_Iss5\\_2305-2315\\_Biomorphs\\_via\\_modified\\_iterations.pdf](https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305-2315_Biomorphs_via_modified_iterations.pdf). Az is jó gyakorlat, ha magából ebből a cikkből from scratch kódoljuk be a sajátunkat, de mi a királyi úton járva a korábbi **Mandelbrot halmazt** kiszámoló forrásunkat módosítjuk. Visszont a program változóinak elnevezését összhangba hozzuk a közlemény jelöléseivel:

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatás:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ↫
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↫
// color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
```

```
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ←
// Vol19_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 < szelessseg; ++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;
            }
        }

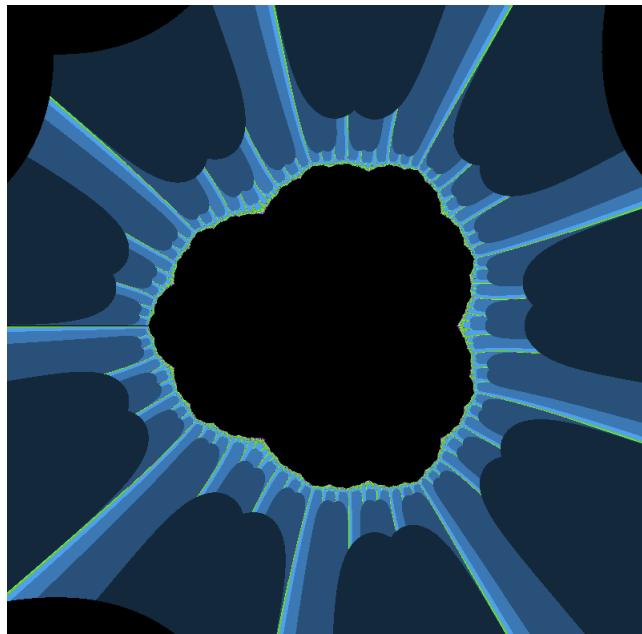
        kep.set_pixel ( x, y,
                        png::rgb_pixel ( (iteracio*20)%255, (iteracio ←
                        *40)%255, (iteracio*60)%255 ) );
    }

    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;

}
```

Ismét a Mandelbrot halmazos programhoz hasonlóan kezdődik a programunk. Viszont most a felhasználó adja meg a cc komplex szám valós és imaginárius részét és a küszöbszámot. A belső for ciklus amiben kiszámoljuk a függvényértékeket addig fut, amíg el nem érjük az iterációs határt, de akkor is megszakad ha nem teljesül ez a feltétel "`std::real ( z_n ) > R || std::imag ( z_n ) > R`". Ezután beállítjuk a pixelek színét és összeállítjuk a képet.



```
kewo@kewo-X540SA:~/Downloads/wat/bhax/attention_raising/Biomorf$ g++ 3.1.3. ←
    cpp -lpng -O3 -o 3.1.3
kewo@kewo-X540SA:~/Downloads/wat/bhax/attention_raising/Biomorf$ ./3.1.3 ←
    bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
Szamitas
bmorf.png mentve.
```

## A Mandelbrot halmaz CUDA megvalósítása

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

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

Ebben a feladatban az Nvidia CUDA technológiájával próbáljuk meg felgyorsítani a Mandelbrot halmazos programunk képgenerálását. A nevéből is látható, hogy ehhez egy Nvidia GPU-ra lesz szükségünk és fel kell ránkunk az nvidia-cuda-toolkit -et. A programunk párhuzamosítását szeretnénk elérni, ezt pedig úgy oldjuk meg, hogy egy 600x600 darab blokkból álló hálót generálunk és a blokkokhoz tartozni fog egy-egy szál.

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
```

```
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.  
//  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <https://www.gnu.org/licenses/>.  
//  
// Version history  
//  
// Mandelbrot png  
// Programozó Páternoszter/PARP  
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←  
_01_parhuzamos_prog_linux  
//  
// https://youtu.be/gvaqijHlRUs  
  
#include <png++/image.hpp>  
#include <png++/rgb_pixel.hpp>  
  
#include <sys/times.h>  
#include <iostream>  
  
  
#define MERET 600  
#define ITER_HAT 32000  
  
__device__ int  
mandel (int k, int j)  
{  
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:  
    // most eppen a j. sor k. oszlopaban vagyunk  
  
    // számítás adatai  
    float a = -2.0, b = .7, c = -1.35, d = 1.35;  
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;  
  
    // a számítás  
    float dx = (b - a) / szelesseg;  
    float dy = (d - c) / magassag;  
    float reC, imC, reZ, imZ, ujreZ, ujimZ;  
    // Hány iterációt csináltunk?  
    int iteracio = 0;  
  
    // c = (reC, imC) a rács csomópontjainak  
    // megfelelő komplex szám  
    reC = a + k * dx;  
    imC = d - j * dy;
```

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

    ++iteracio;

}
return iteracio;
}

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

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

}
*/

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

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

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

```
}

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

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

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

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

int
main (int argc, char *argv[])
{
    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

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

    int kepadat [MERET] [MERET];

    cudamandel (kepadat);

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

    for (int j = 0; j < MERET; ++j)
    {
        //sor = j;
        for (int k = 0; k < MERET; ++k)
    }
```

```
kep.set_pixel (k, j,
    png::rgb_pixel (255 -
        (255 * kepadat[j][k]) / ITER_HAT,
        255 -
        (255 * kepadat[j][k]) / ITER_HAT,
        255 -
        (255 * kepadat[j][k]) / ITER_HAT));
}
}
kep.write (argv[1]);

std::cout << argv[1] << " mentve" << std::endl;

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

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

}
```

A mandel függvény hozza létre a Mandelbrot halmazunkat. Ez a függvény megegyezik az első Mandelbrotos feladattal, mivel itt sem használjuk a complex osztályt. Az nvcc-vel vaó fordításkor a fordító két részre osztja a programot. Az egyik részt az Nvidia fordítóval, a másik pedig gcc-vel fog lefordulni. Az Nvidia fordító foglalkozik azokkal a deklarációkkal, amelyek előtt ott van a `_device_` vagy `_global_` kifejezés. A mandelkernel-ben történik a halmazunk indexelése, itt állapítjuk meg, hogy melyik blokkban van benne az aktuálisan feldolgozás allat lévő szám. A cudamandel függvénynek átadunk egy 600x600-as tömböt. Létrehozunk egy pointert és a memoriában foglalunk neki egy az átadott tömmbbel megegyező méretű területet, erre pedig ráállítjuk. Végül ezt a pointert átadjuk a mandelkernel függvénynek. Amint befejeződött ez a folyamat átmásoljuk az értékeket az argumentumként megadott tömbbe és a lefoglalt területet felszabadítjuk. A main az előző feladatokhoz hasonlóan működik, viszont most kiírjuk a futási időt is.

## Mandelbrot nagyító és utazó C++ nyelven

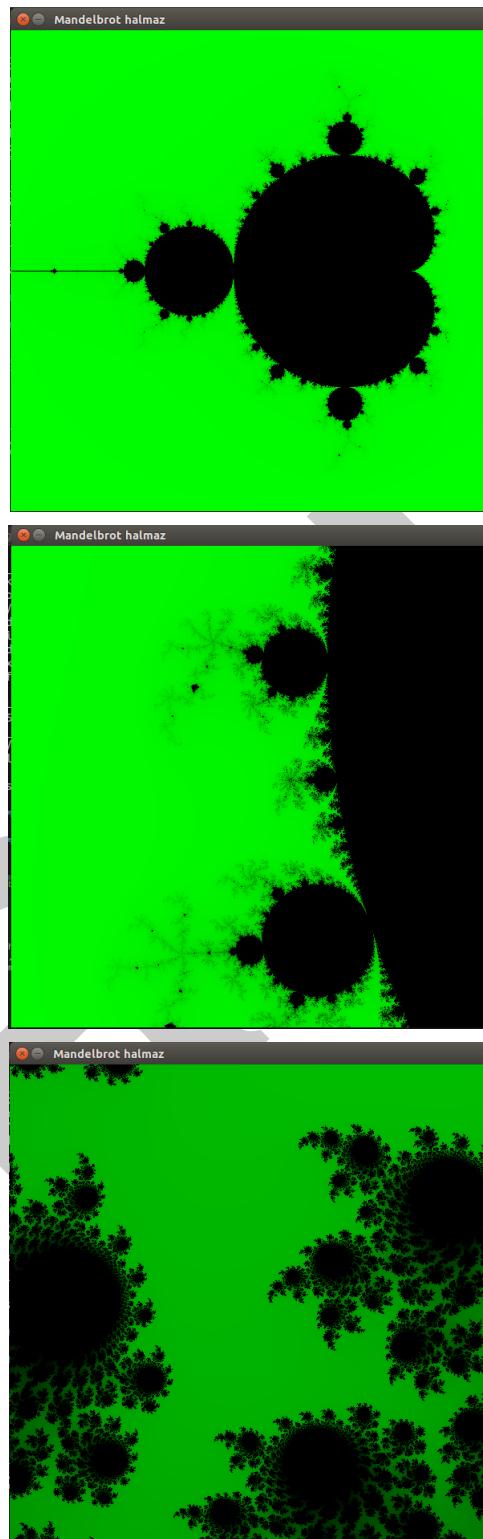
Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteráció bejárta  $z_n$  komplex számokat!

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

Megoldás forrása:

A programunk elkészítéséhez a QT GUI-t fogjuk használni. Ez az egyik legtöbbet használt grafikus interfész. A qmake -project parancs létrehoz nekünk egy .pro kiterjesztésű fájlt. Ebbe a pro fájlba bele kell írjuk, hogy QT += widgets. Így mostmár használhatjuk a qtwidget-eket. A qmake \*.pro parancs létre-

hoz egy Makefile-t, így ha kiadjuk a make parancsot létrejön egy futtatható fájl. Élesítés az n billentyűvel lehetséges.

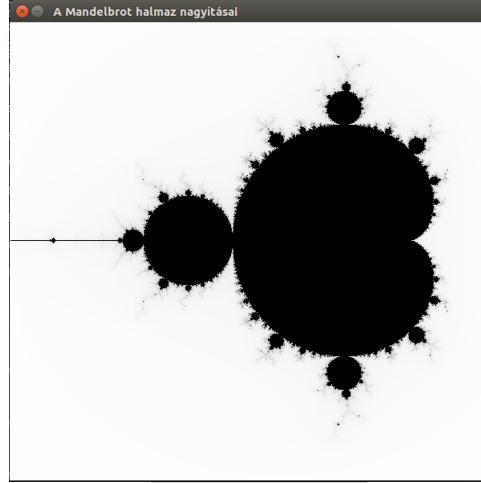


## Mandelbrot nagyító és utazó Java nyelven

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

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

```
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/bhax_textbook/ ←  
programs$ javac MandelbrotHalmazNagyító.java  
kewo@kewo-X540SA:~/Downloads/wat/bhax/thematic_tutorials/bhax_textbook/ ←  
programs$ java MandelbrotHalmazNagyító
```



## 6. fejezet

### Helló, Welch!

#### 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!

#### LZW

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

Megoldás videó:

Megoldás forrása:

#### Fabejárás

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

Megoldás videó:

Megoldás forrása:

#### 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:

## Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

## 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!

#### 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...

#### Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

#### Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 8. fejezet

### Helló, Schwarzenegger!

#### Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

#### Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

#### Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

#### Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 9. fejezet

### Helló, Chaitin!

#### Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

#### Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

#### 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](https://youtu.be/OKdAkI_c7Sc)

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

## Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Omega

Megoldás videó:

Megoldás forrása:

DRAFT

## 10. fejezet

# Helló, Gutenberg!

### Juhász István - Magas szintű programozási nyelvek 1

A programnyelvek három szintjét különböztetjük meg: gépi nyelv, assembly szintű nyelv, magas szintű nyelv. A magas szintű nyelven megírt programot forrásprogramnak, vagy forrásszövegnek nevezzük. Az összeállításra vonatkozó szabályokat szintaktikai, a jelentésbelieket pedig szemantikai szabályoknak nevezzük. Ezek a szabályok határozzák meg a programot. A fordító program a következő lépések alapján teszi a megírt kódot a számítógép által értelmezhetővé: lexikális,szintaktikai,szemantikai elemzés -> kódgenerálás. Vannak imperatív/algoritmikus és deklaratív/nem algoritmikus nyelvek. Olyan nyelvek is léteznek amelyek nem sorolhatóak be ezekbe a kategóriákba. minden program forrásszövegének legkisebb alkotórészei a karakterek. A forrásszöveg összeállításánál alapvető a karakterkészlet, ennek elemei jelenhetnek meg az adott nyelv programjaiban, ezekből állíthatók össze a bonyolultabb nyelvi elemek. Az eljárásorientált nyelvek esetén ezek a következők: lexikális,fordítási és szintaktikai egységek, utasítások, programegységek, program. Egy nyelv saját karakterkészletében az alábbiak találhatóak: betűk,számjegyek,egyéb karakterek.

### Programozás bevezetés

[KERNIGHANRITCHIE]

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

```
main ()  
{  
  
printf ("Figyelem, ");  
printf ("emberek!");  
printf ("\n");  
}
```

A { } kapcsos zárójelek a függvényt alkotó utasításokat zárják közre. Egy tetszőleges számú karakterből álló, idézőjelek ("") közé zárt karaktersorozatot karakterláncnak, karakter-állandónak (stringnek, ill. string-

konstansnak) nevezünk. A karakterláncban előforduló \n karakter sorozat az újsor karakter C-beli jelölés-módja. Hatására a kiírás a következő sor bal szélén folytatódik. A printf-el kiíratjuk a benne lévő stringeket.

```
/* FahrenheitCelsius táblázat kinyomtatása  f = 0, 20, . . . , 300
értékekre */
main ()
{
int lower, upper, step;

float fahr, celsius;
lower = 0; /* A hőmérséklettáblázat alsó határa */
upper = 300; /* felső határ */

step = 20; /* lépésköz */

fahr = lower;

while (fahr <= upper) {

celsius = (5.0 / 9.0) * (fahr - 32.0);

printf ("%4.0f %6.1f \n", fahr, celsius);

fahr = fahr + step;
}
}
```

Az /\* .. \*/-ba írt szöveg csak megjegyzés, így ami itt van az nem vesz részt a programunk működésében. Az int típus azt jelenti, hogy a felsorolt változók egész (integer) típusúak. float jelöli a lebegőpontos (floating point) változókat, vagyis az olyan számokat, amelyeknek tört részük is van. Megjegyzések mindenütt előfordulhatnak, ahol szóköz vagy újsor előfordulhat. A C nyelvben használat előtt minden változót deklarálni kell, általában a függvény elején, az első végrehajtható utasítás előtt.

## Programozás

### [BMECPP]

A c++ a c nyelvre épül. Az első c++ fordítók c kódokat generáltak. A ma használt fordítók c és c++ ra is tudnak fordítani. C++ nyelvben az üres paraméterlista egy void paraméter megadásával egyenlő. C-ben írnánk: void f(){} , C++-ban pedig: void f(void){}. C++-ban a main függvény kétféle képpen adhatjuk meg: int main(){}, int main(int argc, char\* argv[]){}. Az argc-ben a parancssor-argumentumok számát, argv-ben pedig a parancssor-argumentumokat kapjuk. Ha nem írunk return-t, akkor a fordító autómatikusan return 0; utasítást fordít a kódba. A bool típus a logika igaz/hamis értéket képes jelenteni. Ezek a változók true vagy false értéket vehetnek fel: bool a = true; bool b = false;. minden olyan helyen állhat változódeklaráció, ahol utasítás állhat. C++ nyelven megadhatunk alapértelmezett függvényargumentumokat. Ha nem adunk meg saját értékeket, akkor ezek kerülnek felhasználásra.

**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!

#### A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

#### 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**

DRAFT

## Általános

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

## C

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

## C++

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

## Lisp

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

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

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