

Univerzális programozás

Így neveld a programozód!

Ed. BHAX, DEBRECEN,
2020. március 2, v. 0.0.5

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, 2020, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Ács Bátfai, Margaréta	2020. április 1.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	8
2.3. Változók értékének felcserélése	10
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	12
2.6. Helló, Google!	14
2.7. A Monty Hall probléma	16
2.8. 100 éves a Brun tétel	18
3. Helló, Chomsky!	23
3.1. Decimálisból unárisba átváltó Turing gép	23
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	24
3.3. Hivatkozási nyelv	26
3.4. Saját lexikális elemző	28
3.5. Leetspeak	29
3.6. A források olvasása	33
3.7. Logikus	34
3.8. Deklaráció	35

4. Helló, Caesar!	39
4.1. double ** háromszögmátrix	39
4.2. C EXOR titkosító	42
4.3. Java EXOR titkosító	45
4.4. C EXOR törő	47
4.5. Neurális OR, AND és EXOR kapu	51
4.6. Hiba-visszaterjesztéses perceptron	55
5. Helló, Mandelbrot!	58
5.1. A Mandelbrot halmaz	58
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	59
5.3. Biomorfok	61
5.4. A Mandelbrot halmaz CUDA megvalósítása	65
5.5. Mandelbrot nagyító és utazó C++ nyelven	65
5.6. Mandelbrot nagyító és utazó Java nyelven	65
6. Helló, Welch!	66
6.1. Első osztályom	66
6.2. LZW	66
6.3. Fabejárás	66
6.4. Tag a gyökér	66
6.5. Mutató a gyökér	67
6.6. Mozgató szemantika	67
7. Helló, Conway!	68
7.1. Hangyaszimulációk	68
7.2. Java életjáték	68
7.3. Qt C++ életjáték	68
7.4. BrainB Benchmark	69
8. Helló, Schwarzenegger!	70
8.1. Szoftmax Py MNIST	70
8.2. Mély MNIST	70
8.3. Minecraft-MALMÖ	70

9. Helló, Chaitin!	71
9.1. Iteratív és rekurzív faktoriális Lisp-ben	71
9.2. Gimp Scheme Script-fu: króm effekt	71
9.3. Gimp Scheme Script-fu: név mandala	71
10. Helló, Gutenberg!	72
10.1. Programozási alapfogalmak	72
10.2. Programozás bevezetés	72
10.3. Programozás	72
III. Második felvonás	73
11. Helló, Arroway!	75
11.1. A BPP algoritmus Java megvalósítása	75
11.2. Java osztályok a Pi-ben	75
IV. Irodalomjegyzék	76
11.3. Általános	77
11.4. C	77
11.5. C++	77
11.6. Lisp	77

Ábrák jegyzéke

2.1. A B_2 konstans közelítése	22
4.1. A double ** háromszögmátrix a memóriában	41

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogy lássuk mást 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 dlatex programmal a könyved pdf változatát, íme:

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- C kapcsán a [**KERNIGHANRITCHIE**] könyv adott részei.
- C++ kapcsán a [**BMECPP**] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány **ISO/IEC 9899:2017** kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a **The GNU C Reference Manual**, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [**BMECPP**] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

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

- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c](#), [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozo/Turing/infty-w.c](#).

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

Egy mag 100 százalékban:

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

    return 0;
}
```

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

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

    return 0;
}
```


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

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

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

Egy mag 0 százalékban:

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

    return 0;
}
```

Minden mag 100 százalékban:

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

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

```
top - 20:09:06 up 3:35, 1 user, load average: 5,68, 2,91, 1,38
Tasks: 329 total, 2 running, 256 sleeping, 0 stopped, 1 zombie
%Cpu0 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

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

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



Werkfilm

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

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

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

Megoldás videó:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

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

Mit for kiírni erre a T1000(T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

A feladatnak nincs megoldása mert: ha a program tartalmaz végtelen ciklus akkor lefagy = true és akkor lefagy2 is true lesz ha meg a programban nincs végtelen ciklus akkor lefagy = false és akkor lefagy 2 egy végtelen ciklussá alakul át.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ha fel szeretnénk két változónak értékét cserélni, akkor a legegyszerűbb megoldásnak egy segédváltozó bevezetése tűnhet. De ennél sokkal jobb módszerek vannak. Az egyik ilyen megoldás az hogy változó értékét összeadjuk majd ebből az összegből kivonjuk a régi változók értékét.

```
int a = 1;
int b = 2;
a = a+b;
b = a-b;
a = a-b;
```

De nem csak matematikai módszerekkel tudunk két változót felcserélni hanem xorral is. Ennek a típusnak annyi lenne a lényege hogy a számokat kettes számrendszerben tároljuk vagyis 0-ból és 1-esekből áll. Xor művelet minden esetben 1-et ad vissza kivéve ha a két változónknak az értéke megegyezik.

```
int a = 1;
int b = 2;
a = a^b;
b = a^b;
a = a^b;
```

2.4. Labdapattogás

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A labdapattogtatás gyakorlatilag annyiból áll hogy a terminálban egy karakter pattog az ablak jelenlegi méretében. Az initscr feladata az hogy beolvassa az ablak méretét. Ezután létrehozunk változókat hogy eltároljuk a labda helyzetét x és y, lépésközt a xnov és ynov-ban, ablak mérete mx és my. Mivel initscr-rel olvassuk be az ablak méretét így ha pattogás közbe változtatunk az ablak nagyságán azt érzékelni fogja a program.

```
WINDOW *ablak;
ablak = initscr ();
```

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

int xnov = 1;
int ynov = 1;

int mx;
int my;
```

Miután végeztünk a változók létrehozásával. Csinálunk egy végtelenciklust ebbe fogjuk pattogtatni a labdát. Ehhez használnunk kell `getmaxyx()` függvényt melynek átadjuk az ablakban eltárolt értékeket. A `mvpprintw()` függvény fogja a "labdánkat" a megfelelő kordinátákba pattogtatni. Most már tudjuk az ablak tulajdonságait. Az `x` és `y` változó folyamatosan növekedni fog +1-el és így érjük el a pattogást a terminálban. A labda mozgásának a sebességét a `usleep()` függvénnyel tudjuk megadni minnél nagyobb annál lassabb. Ám nem kell félni a nagyobb számoktól mert a `usleep` mikroszekundumba számol tehát ha azt akarjuk hogy a labdánk 1/s sebességgel menjen egymilliót kell megadni.

```
for ( ;; ) {

    getmaxyx ( ablak, my , mx );

    mvprintw ( y, x, "O" );

    refresh ();
    usleep ( 100000 );
```

Ennél a résznél fog különbözni az if-es és a nem ifes megoldás. Elsőnek az if-est nézve

```
if ( x>=mx-1 ) { // jobb oldalt ellenőrzi
    xnov = xnov * -1;
}
if ( x<=0 ) { // bal oldalt ellenőrzi
    xnov = xnov * -1;
}
if ( y<=0 ) { // tetejét ellenőrzi
    ynov = ynov * -1;
}
if ( y>=my-1 ) { // alját ellenőrzi
    ynov = ynov * -1;
}
```

Tehát ha valamelyik érték eléri a határát beszorozzuk minusz egyel és így elérjük hogy visszafele pattogjon. A nem if-es megoldásnál maradékos osztást használunk hogy a labda egy bizonyos értéknél visszapattanjon. Ennek a megoldásnak annyi lenne a lényege hogy a szám értékét fogja visszaadni amit osztunk addig amíg a két számnak az értéke vagyis az ablak szélességével és magaságával.

```
getmaxyx(ablak, my, mx);
xj = (xj - 1) % mx;
xk = (xk + 1) % mx;
yj = (yj - 1) % my;
```

```
yk = (yk + 1) % my;
//clear ();
mvprintw (abs (yj + (my - yk)),
          abs (xj + (mx - xk)), "X");
refresh ();
usleep (100000);
```

2.5. Szóhossz és a Linus Torvalds féle BogomIPS

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

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

Megoldás forrása: [bhaax/thematic-tutorials/bhaax-textbook-IgyNeveldaProgramozod/Turing/bogomips.c](https://bhaax.com/thematic-tutorials/bhaax-textbook-IgyNeveldaProgramozod/Turing/bogomips.c)

Tanulságok, tapasztalatok, magyarázat...

```
#include <stdio.h>

int main()
{
    int word = 1; //kettes számrendszerben: 00000000 ←
                00000000 00000000 00000001
    int length = 0;

    do
    {
        length++;
    }
    while (word<=1); //itt léptetjük eggyel:00000000 ←
                00000000 00000000 00000010
                // újra: 00000000 00000000 00000000 ←
                00000100
                //...

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

    return 0;
}
```

A típus méretét, illetve hogy hány bitet foglal, bitshifteléssel könnyen meghatározhatjuk. A while ciklusunkban mindig shiftelünk egyet balra a biteken és addig növeljük a length változót, amíg csupa 0 bitet nem fog tartalmazni a word változónk.

Ahogy a feladatban már említve lett a BogomIPS hát akkor ő következik. Azért foglalkozunk ezzel, mert ennek a while fejléce ugyan azt a módszert használja amit az előbb csináltunk.

```
while (loops_per_sec <= 1)
{
    ;
}
```

Először deklarálunk 2 változót, az első a `loops_per_sec`, melynek kezdetkor az értékét egyre állítjuk. A bitshiftelés hatására ebbe 2 hatványokat fogunk tárolni. A `ticks` pedig a CPU időt fogja tárolni.

```
while (loops_per_sec <= 1 )
{
    ticks = clock();
    delay (loops_per_sec);
    ticks = clock() - ticks;

    ...
}
```

A `while` ciklus addig tart, ameddig a `loops_per_sec` le nem nullázódik. A ciklusba belépve, minden iterációban, az aktuális CPU időt, eltároljuk a `ticks` változóban. Ezután pedig meghívjuk a `delay` függvényt.

```
void delay (unsigned long long loops)
{
    unsigned long long i;
    for ( i=0; i<loops; i++);
}
```

Ez a függvény egy hosszú egész számot kér paraméterként, ezek után a `for` ciklus megy végig 0-tól paraméter-1-ig. Utána a `while` ciklusban újra lekérjük az aktuális processzor időt és kivonjuk belőle a kezdeti értéket. Így megtudjuk, hogy mennyi ideig tartott a `cpu`-nak befejezni a `delay` függvényben lévő `for` ciklust. Ezt egészen addig ismételjük, ameddig nem teljesül az `if`-ben lévő feltétel.

```
while (loops_per_sec <= 1 )
{
    ...

    if (ticks >= CLOCKS_PER_SEC)
    {
        loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC; // ←
        loops_per_sec/ticks = ???/CLOCKS_PER_SEC

        printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec/500000,
            (loops_per_sec/5000) % 100);
        return 0;
    }
}
```

A `CLOCKS_PER_SEC` szabvány értéke 1.000.000, vagyis akkor teljesül az `if`-ben lévő feltétel ha a szabvány értéket eléri vagy meghaladja a processzor idő. Majd kiszámoljuk mennyi `CLOCKS_PER_SEC` idő alatt milyen hosszú ciklust képes végrehajtani a gép. A eredmény megadásához használhatjuk a `log` függvényt a következő módosítások után.

```
printf("ok - %lld %f BogoMIPS\n", loops_per_sec, log( ←  
    loops_per_sec));
```

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A PageRank képlete: $PR(A) = (1-d) + d (PR(T1)/C(T1) + \dots + PR(Tn)/C(Tn))$ A: az aktuális oldal T: oldalak amik az a-ra hoz vannak linkelve PR(A): a oldal pagerank száma PR(Tn): azok oldalak pagerank száma amik az a-ra mutatnak C(Tn): t oldalon az összes link ami a-ra mutatnak d: 0 és 1 közé esik

Elsőnek is a kapcsolatokat a mátrixba tároljuk. Sorok és oszlopok metszetébe láthatjuk milyen kapcsolat van az oldalak között. Ezt a mátrixot adjuk a `pagerank()` - nak függvénynek. A tömbben tároljuk az oldalak értékét, és a PR-ben tároljuk el a mátrixszorzás eredményét. A mátrixszorzást a L és PRv tömbökkel hajtjuk végre. Ezek a tömbök összeszorzásával kapunk egy 4x1 oszlopvektort ami a 4 oldalunk pagerankja lesz.

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

    int i, j;

    for(;;){

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

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

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

    kiir (PR, 4);
}

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}
    };
};
```

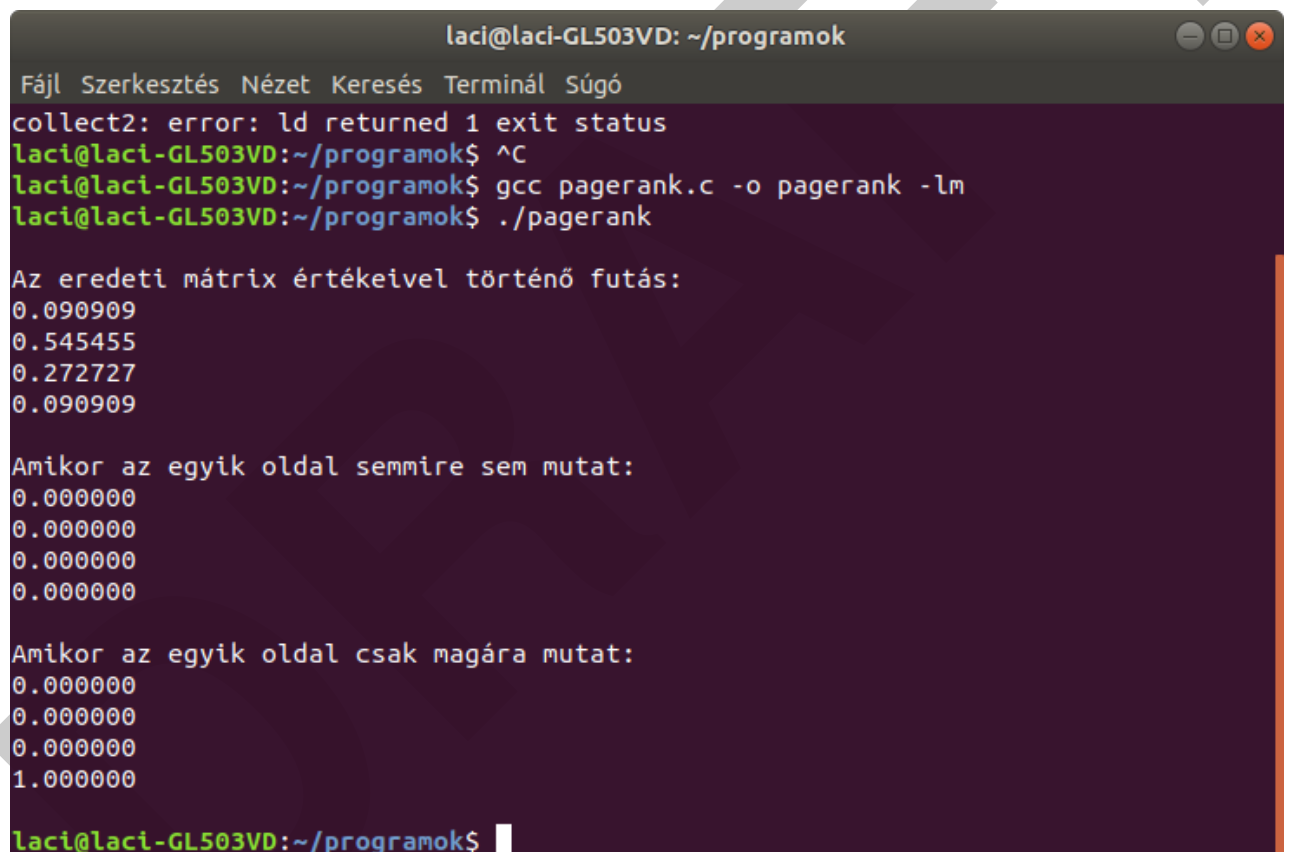
```
printf("\nAz eredeti mátrix értékeivel történő futás:\n");
pagerank(L);

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;
}
```



```
laci@laci-GL503VD: ~/programok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
collect2: error: ld returned 1 exit status
laci@laci-GL503VD:~/programok$ ^C
laci@laci-GL503VD:~/programok$ gcc pagerank.c -o pagerank -lm
laci@laci-GL503VD:~/programok$ ./pagerank

Az eredeti mátrix értékeivel történő futás:
0.090909
0.545455
0.272727
0.090909

Amikor az egyik oldal semmire sem mutat:
0.000000
0.000000
0.000000
0.000000

Amikor az egyik oldal csak magára mutat:
0.000000
0.000000
0.000000
1.000000

laci@laci-GL503VD:~/programok$
```

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall probléma alapja három ajtó, mely mindegyike rejt valamit: két ajtó mögött egy-egy roncs autó van, egy ajtó mögött pedig egy vadiúj autó. Ez annyit jelent, hogy két ajtó értéktelen és egy ajtó pedig értékes ajándékot rejt. Azért, hogy könnyebben tudjunk beszélni az egészeről, mondjuk azt, hogy két ajtó mögött nincs és egy ajtó alatt pedig van ajándék.

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

for (i in 1:kiserletek_szama) {

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

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

  }else{

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

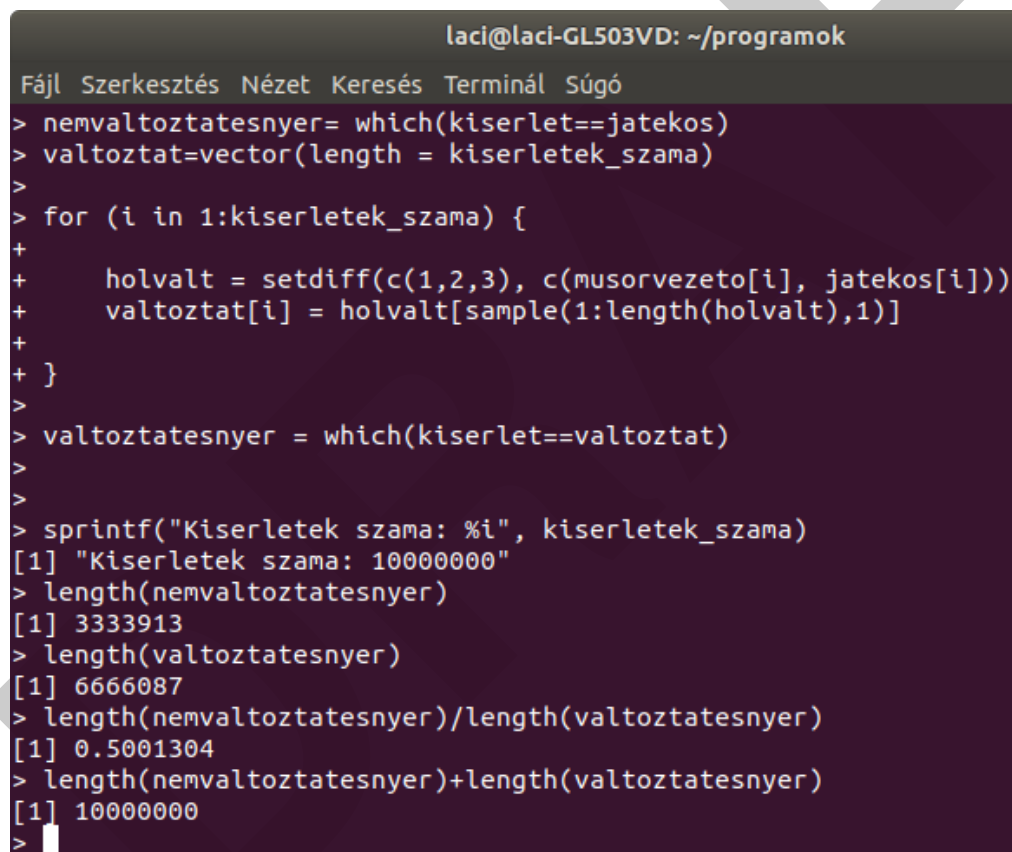
  }

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

}

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

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



```
laci@laci-GL503VD: ~/programok  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
> nemvaltoztatesnyer= which(kiserlet==jatekos)  
> valtoztat=vector(length = kiserletek_szama)  
>  
> for (i in 1:kiserletek_szama) {  
+   holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
+   valtoztat[i] = holvalt[sample(1:length(holvalt),1)]  
+ }  
> valtoztatesnyer = which(kiserlet==valtoztat)  
>  
>  
> sprintf("Kiserletek szama: %i", kiserletek_szama)  
[1] "Kiserletek szama: 10000000"  
> length(nemvaltoztatesnyer)  
[1] 3333913  
> length(valtoztatesnyer)  
[1] 6666087  
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)  
[1] 0.5001304  
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)  
[1] 10000000  
>
```

2.8. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

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

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

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

Majd vizsgáljuk meg az a sorozatot:

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

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

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

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

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

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

```
# Copyright (C) 2019 Dr. Norbert Bاتفai, nbاتفai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
```

```
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>

library(matlab)

stp <- function(x){

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

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

Soronként értelmezzük ezt a programot:

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

Kiszámolja a megadott számig a prímeket.

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

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

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

Az egymást követő prímelek különbségét képezi, tehát 3-2, 5-3, 7-5, 11-7, 13-11.

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

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

```
[1] 2 3 5
```

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

```
tlprimes = primes[idx]
```

Kivette a `primes`-ből a párok első tagját.

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

```
rt1plust2 = 1/tlprimes+1/t2primes
```

Az $1/tlprimes$ a `tlprimes` 3,5,11 értékéből az alábbi reciprokokat képi:

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

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

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

Az $1/tlprimes + 1/t2primes$ pedig ezeket a törtet rendre összeadja.

```
> 1/tlprimes+1/t2primes  
[1] 0.53333333 0.3428571 0.1678322
```

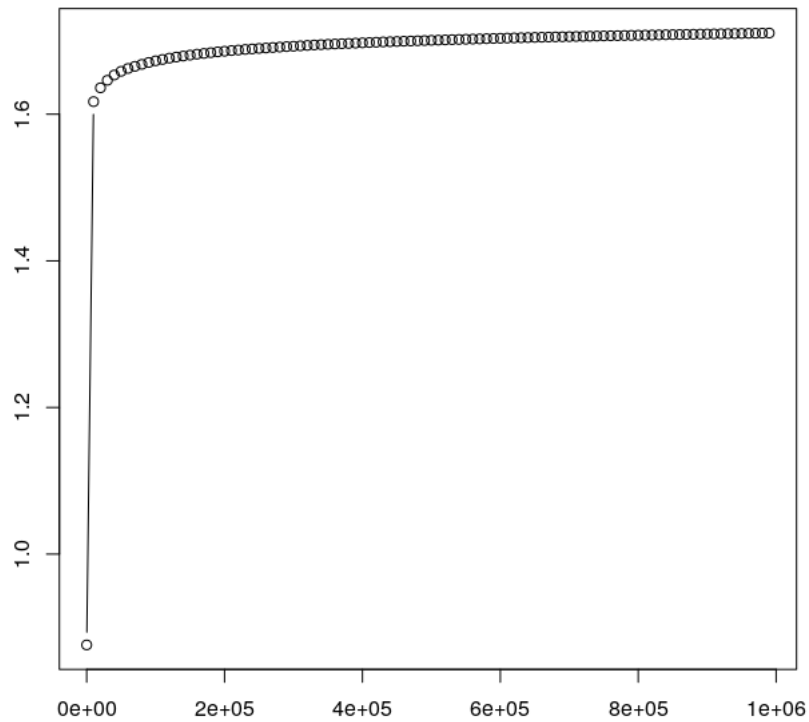
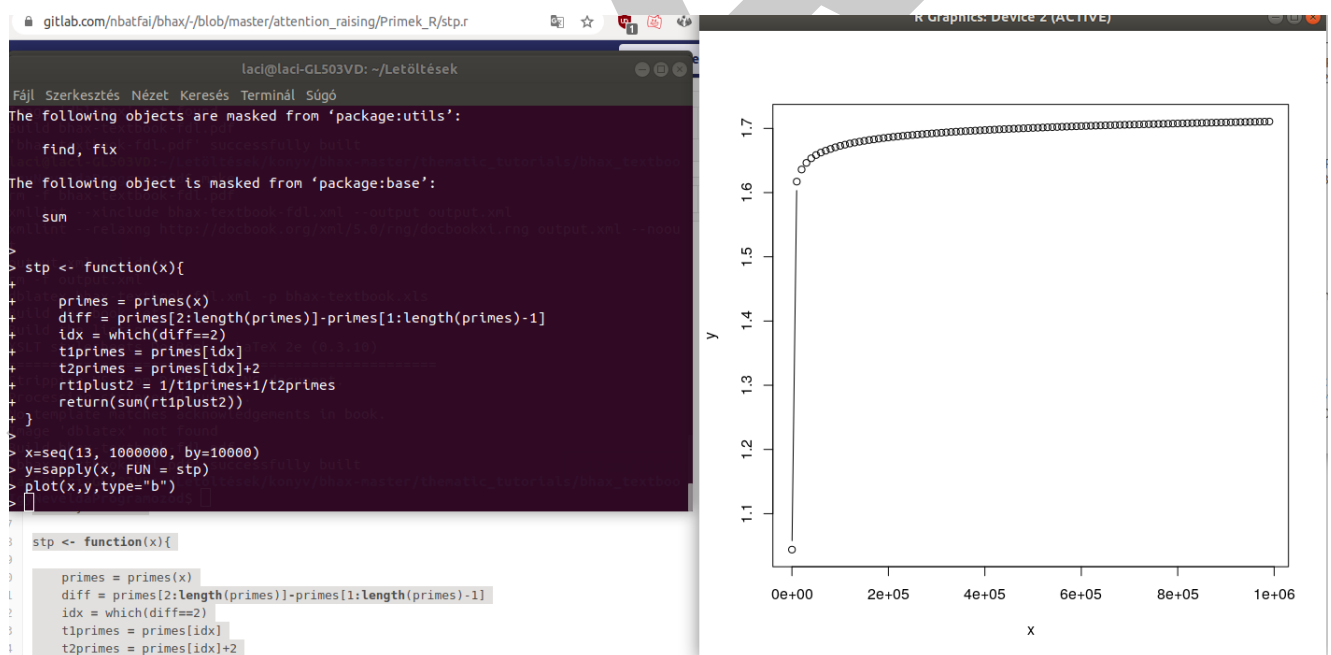
Nincs más dolgunk, mint ezeket a törtet összeadni a `sum` függvénnyel.

```
sum(rt1plust2)
```

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

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

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

2.1. ábra. A B_2 konstans közelítése**Werkfilm**

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

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: az első előadás [27 fólia](#).

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

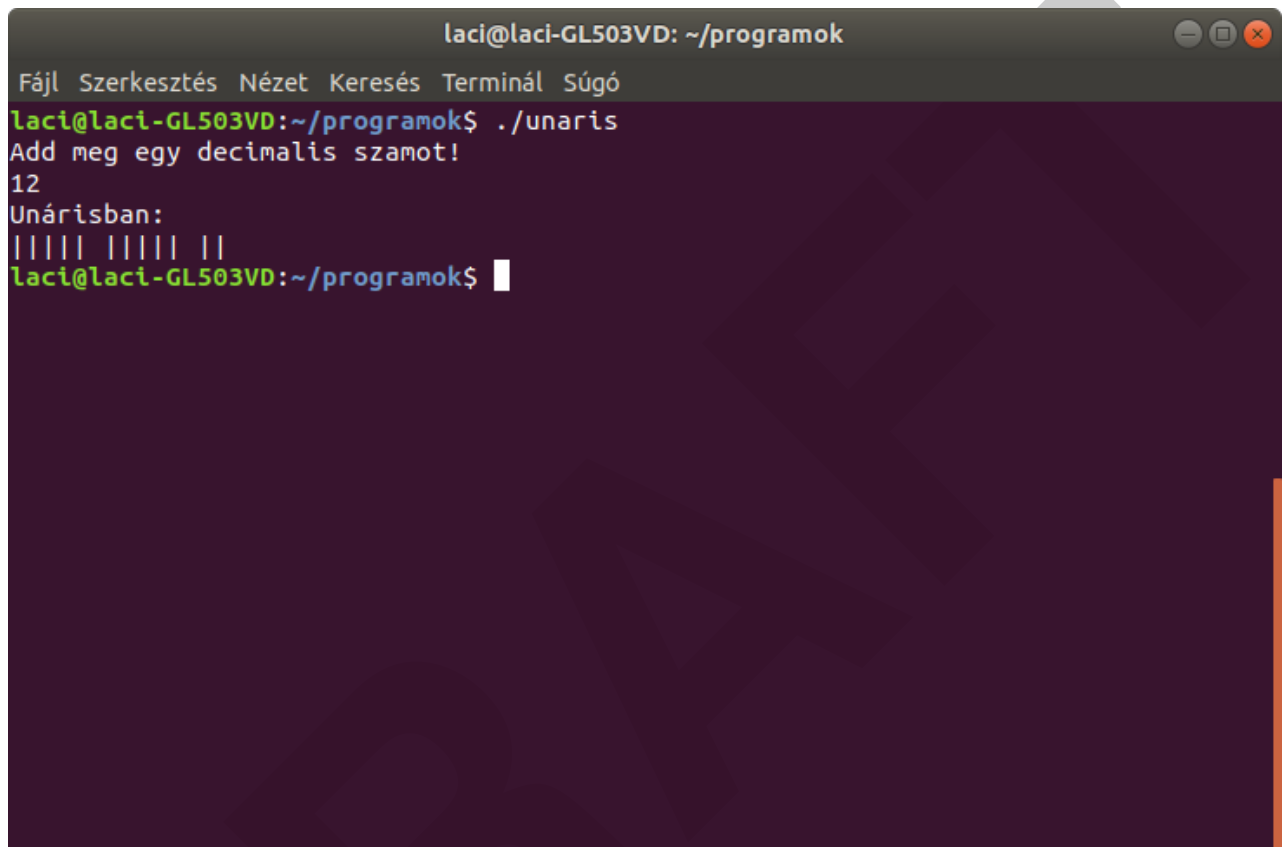
A unáris számrendszer a legegyszerűbb az összes számrendszer közül. Gyakorlatilag ugyanaz, mintha az ujjaink segítségével számolnánk, tehát a számokat vonalakkal jelöljük. A szám ábrázolása pont annyi darab vonalból áll, amennyi a szám. A könnyebb olvashatóság érdekében minden ötödik után rakhatunk helyközt vagy bármilyen karaktert amit tetszik.

A forrásként megadott program lényegében egy átváltó, mely függőleges vonalakat ír ki a bemenettől függően. Mivel ez egy C++ program, ezért ennek a fordításához a g++-t érdemes használni, a szintaxisa teljesen megegyezik a gcc-nél megszokottakkal. Ha futtatjuk ezt kell látnunk.

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

int main()
{
    int b;
    int szamlalo = 0;
    cout<<"Add meg egy decimalis szamot!\n";
    cin >> b;
    cout<<"Unárisban:\n";
    for (int i = 0; i < b; ++i)
    {
        cout<<"1";
        ++szamlalo;
        if (szamlalo % 5 == 0)
            cout<<" ";
    }
}
```

```
}  
cout<<'\\n';  
return 0;  
}
```



The screenshot shows a terminal window titled 'laci@laci-GL503VD: ~/programok'. The menu bar includes 'Fájl', 'Szerkesztés', 'Nézet', 'Keresés', 'Terminál', and 'Súgó'. The terminal content is as follows:

```
laci@laci-GL503VD:~/programok$ ./unaris  
Add meg egy decimalis szamot!  
12  
Unárisban:  
||||| ||||| ||  
laci@laci-GL503VD:~/programok$
```

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

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

Megoldás videó:

Megoldás forrása: az első előadás [30-32 fólia](#).

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

A generatív grammatika Noam Chomsky nevéhez kötődik. Úgy vizsgálja a nyelvtant mint az ismeret alapját, "hiszen ha nem lenne nyelvtanunk, akkor a tudást nem tudnánk se megörökíteni, se továbbadni. Nézete azt a vállalta, hogy a tudás és az ismeret többnyire öröklött (generációról generációra terjed), vagyis univerzális (gondolván a gyerekekre, akik könnyedén elsajátítják anyanyelvüket). A generatív grammatikának négy fő része van: nemterminális jelek, terminális jelek, helyettesítési/képzési szabályok és mondat/kezdő szimbólumok, illetve három nyelvtan fajtája: környezetfüggő, környezetfüggetlen és reguláris. Nézzünk meg két példát környezetfüggő leírásra (a nyílak jelölik majd a képzési szabályokat)

1. pelda

S, X, Y "változók"

a, b, c "konstansok"

S → abc, S → aXbc, Xb → bX, Xc → Ybcc, bY → Yb, aY → aaX, aY → aa

S-ből indulunk ki

S (S → aXbc)

aXbc (Xb → bX)

abXc (Xc → Ybcc)

abYbcc (bY → Yb)

aYbbcc (aY → aa)

aabbcc

S (S → aXbc)

aXbc (Xb → bX)

abXc (Xc → Ybcc)

abYbcc (bY → Yb)

aYbbcc (aY → aaX)

aaXbbcc (Xb → bX)

aabXbcc (Xb → bX)

aabbXcc (Xc → Ybcc)

aabbYbcc (bY → Yb)

aabYbbccc (bY → Yb)

aaYbbbccc (aY → aa)

aaabbbccc

2. pelda

A, B, C "változók"

a, b, c "konstansok"

A → aAB, A → aC, CB → bCc, cB → Bc, C → bc

S-ből indulunk ki

A (A → aAB)

aAB (A → aC)

aaCB (CB → bCc)

aabCc (C → bc)

aabbcc

A (A → aAB)

aAB (A → aAB)

aaABB (A → aAB)

aaaABBB (A → aC)

aaaaCBBB (CB → bCc)

```
aaaabCcBB (cB -> Bc)
aaaabCBcB (cB -> Bc)
aaaabCBBc (CB -> bCc)
aaaabbCcBc (cB -> Bc)
aaaabbCBcc (CB -> bCc)
aaaabbbCccc (C -> bc)
aaaabbbbcccc
```

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása: az első előadás [63-65 fólia](#).

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

A C utasítás fogalma BNF-ben való definiálása:

```
<utasítás> ::= <kifejezés> | <összetett_utasítás> | <feltételes_utasítás> | ↔
    <while_utasítás> | <do_utasítás> | <for_utasítás> | <switch_utasítás> | ↔
    <break_utasítás> | <continue_utasítás> | <return_utasítás> | <↵
    goto_utasítás> | <cimke_utasítás> | <nulla_utasítás>

<kifejezés> ::= <értékadás> | <függvényhívás>
<értékadás> ::= <változó><szám>
<változó> ::= <betű>{<betű>}
<betű> ::= a-z
<szám> ::= <számjegy>{<számjegy>}
<számjegy> ::= 0|1|2|3|4|5|6|7|8|9
<függvényhívás> ::= <típus><függvéynév>
<típus> ::= <betű>{<betű>}
<függvéynév> ::= <betű>{<betű>}

<összetett_utasítás> ::= <deklarációlista> | <utasításlista>
<deklarációlista> ::= <deklaráció>{<deklaráció>}
<deklaráció> ::= <típus><változó>
<utasításlista> ::= <utasítás>{<utasítás>}

<feltételes_utasítás> ::= if<kifejezés><utasítás> | if<kifejezés><utasítás> ↔
    else<utasítás>

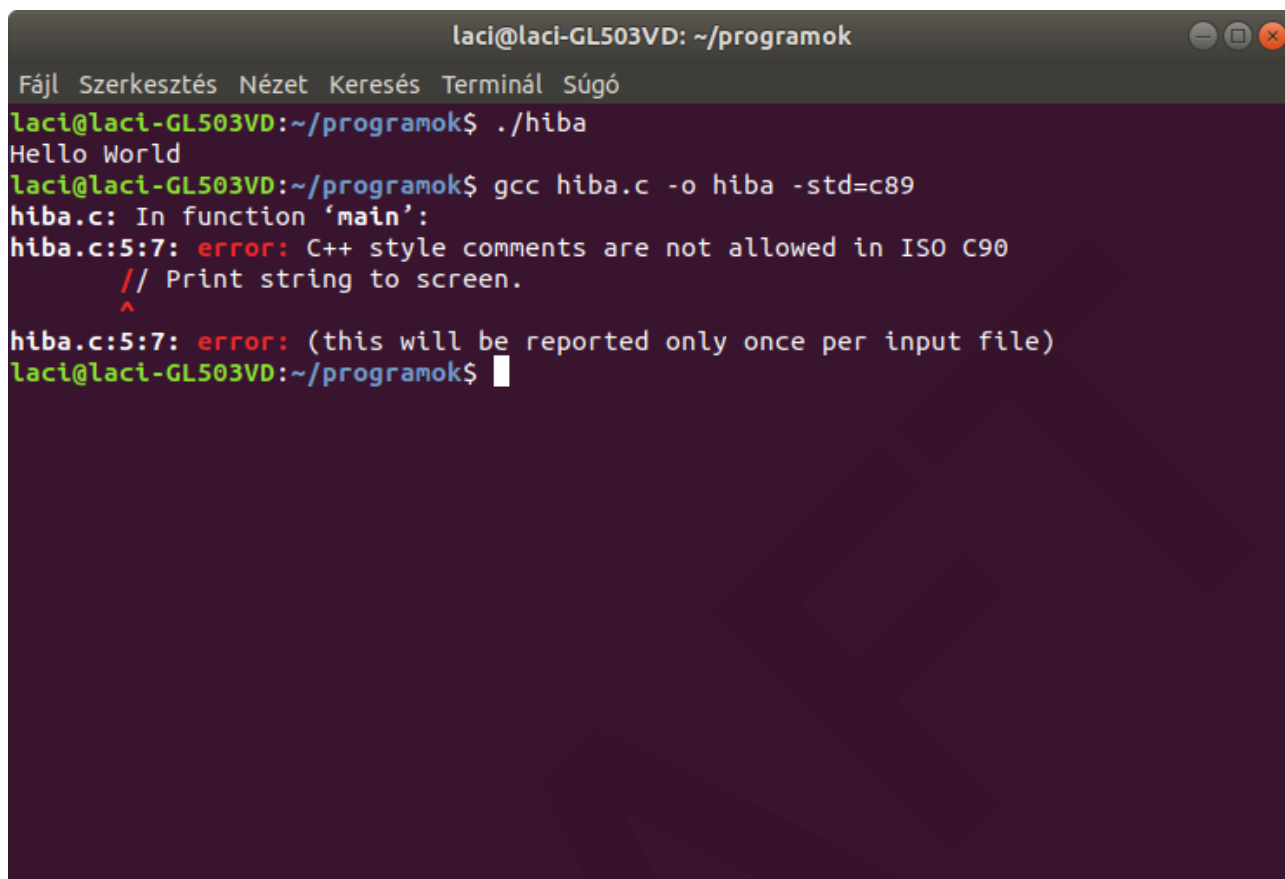
<break_utasítás> ::= break
```

```
<while_utasítás> ::= while<kifejezés><utasítás> | while<kifejezés><utasítás> ←  
    ><break_utasítás>  
  
<do_utasítás> ::= do<utasítás>while<kifejezés> | do<utasítás>while< ←  
    kifejezés><break_utasítás>  
  
<for_utasítás> ::= for([<kifejezés>][<kifejezés>][<kifejezés>])<utasítás> | ←  
    for([<kifejezés>][<kifejezés>][<kifejezés>])<utasítás><break_utasítás>  
  
<switch_utasítás> ::= switch<kifejezés><utasítás> | switch<kifejezés>< ←  
    utasítás><case><kifejezés><default> | switch<kifejezés><utasítás>< ←  
    break_utasítás>  
  
<continue_utasítás> ::= continue  
  
<return_utasítás> ::= return | return<kifejezés>  
  
<goto_utasítás> ::= goto<azonosító>  
  
<cimke_utasítás> ::= <azonosító>  
<azonosító> ::= <cimke>  
<cimke> ::= <betű>{<betű>}  
  
<nulla_utasítás> ::=
```

Program nyelveket nem hiába nevezzük "nyelvnek" mert olyan mint egy normális nyelv fejlődik új funkciókat tulajdonságokat kap mint ebbe a feladatban is.c89 és a c99 verziók is ilyenek.Ez a példaprogram azért nem fut le c89-be mert c99-ben implementálták a kommenteket így c89 nem ismeri fel így hát hibüzenetet dob be.

```
[  
#include <stdio.h>  
  
int main ()  
{  
    // Print string to screen.  
    printf ("Hello World\n");  
    return 0;  
}
```

```
for (int i = 0; i < 10; i++) {  
    x=i*i;  
    printf("%d", x);  
}
```



```
laci@laci-GL503VD: ~/programok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ ./hiba
Hello World
laci@laci-GL503VD:~/programok$ gcc hiba.c -o hiba -std=c89
hiba.c: In function 'main':
hiba.c:5:7: error: C++ style comments are not allowed in ISO C90
      // Print string to screen.
      ^
hiba.c:5:7: error: (this will be reported only once per input file)
laci@laci-GL503VD:~/programok$
```

3.4. Saját lexikális elemző

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1)

Tanulságok, tapasztalatok, magyarázat...

A forrásból kapott program egy lex program melyel egy lexikális elemzőt lehet készíteni. Szövegfájlokl olvassa be a lexikális szabályokat, és egy c forráskódot készít melyet gcc-vel tudunk fordítani. A mi lex forráskódunk 3 részből áll ezeket a részeket %% -al vannak elválasztva egymástól. Az első az a definíciós rész ahol bármilyen c-s forrást lehet használni itt lehet a header-et meghívni. A kapcsos zárójelekbe levő kódot ahogy beírtuk úgy másolja be a c programunkba. Az elején behívjuk a szokásos include-dal deklaráljuk a studio header fájlt ezután létrehozunk egy realnumber nevezetű int-et melyet majd arra fogjuk használni hogy tárolja hány számot olvas be a program. Ezek után jönnek a definíciók és a mi helyzetünkben a digit nevű definícióval a szögletes zárójelben való 0-9 ig való számokat vesszük.

```
%{
#include <stdio.h>
int realnumbers = 0;
```

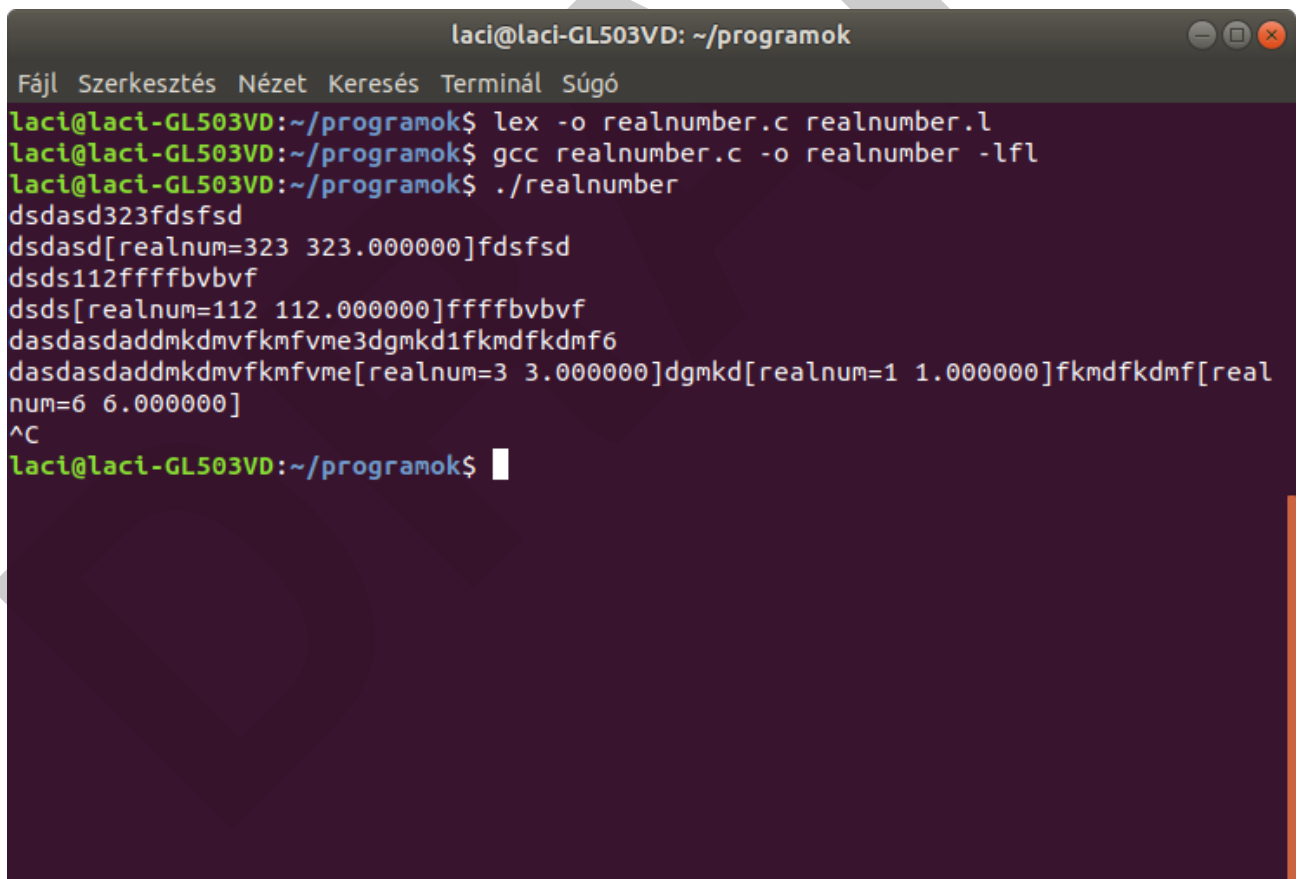
```
%}  
digit [0-9]
```

A második részünk a fordítási szabályoknak van fenntartva. Ez 2 részből áll reguláris kifejezésekből és az azokhoz tartozó c utasításokhoz.

```
%%  
{digit}* (\.{digit}+)? {++realnumbers;  
    printf("[realnum=%s %f]", yytext, atof(yytext));}
```

A harmadik rész az már a tényleges program, ahol meghívjuk az előbb létrehozott yylex függvényt.

```
%%  
int  
main ()  
{  
    yylex ();  
    printf("The number of real numbers is %d\n", realnumbers);  
    return 0;  
}
```



```
laci@laci-GL503VD: ~/programok  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
laci@laci-GL503VD:~/programok$ lex -o realnumber.l realnumber.l  
laci@laci-GL503VD:~/programok$ gcc realnumber.c -o realnumber -lfl  
laci@laci-GL503VD:~/programok$ ./realnumber  
dsdasd323fdsfsd  
dsdasd[realnum=323 323.000000]fdsfsd  
dsds112fffffbvbf  
dsds[realnum=112 112.000000]fffffbvbf  
dasdasdaddmkmvfkfmfvme3dgmkd1fkmdfkdmf6  
dasdasdaddmkmvfkfmfvme[realnum=3 3.000000]dgmkd[realnum=1 1.000000]fkmdfkdmf[real  
num=6 6.000000]  
^C  
laci@laci-GL503VD:~/programok$
```

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/1337d1c7.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/1337d1c7.1)

Mi előtt átnéznénk a forráskódot találjuk ki hogy mi is az a leet nyelv. A leet nyelv arra lett létrehozva hogy a szavakban lévő betűket számokra vagy speciális karakterekre cseréljük le. Az 3.4-es feladatban megnéztük hogy hogy épül fel egy lexer és hogy hogy lehet használni. A program elején mint az előző feladatban importálva vannak a header fájlok.

```
%{  
    #include <stdio.h>  
    #include <stdlib.h>  
    #include <time.h>  
    #include <ctype.h>
```

A második részben találkozunk egy #define-al amivel azt tudjuk elérni ha a programban valahol hivatkozunk a L337size-ra akkor mellette lévő értékkel fogja helyettesíteni. A következő részben láthatjuk a cipher struktúrát amely egy char-c-ből és egy 4 elemű tömbre mutató char *-ből áll. Ezek alapján hozzuk létre a l337d1c7 [] tömböt. Ez a tömb tárolja az egyes betűket és a hozzá tartozó helyettesítő karaktereket.

```
%{  
    #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
};
```

```
%}
```

A harmadik részben a for ciklusunkba lekérjük a `l337size` értékét ami a `l337dlc7[]` tömb oszta a `chiper` struktúra méretével. Mivel `l337dlc7[]` egy struktúrált tömb ezért `l337dlc7[i].c` ként tudunk hivatkozni a tömb egy bizonyos elemére. A `tolower` az átalakítja a nagybetűs bemenetet kisbetűkre. Ezek után pörgetünk egy random számot 1-100 között. Ettől a számtól fog eldőlni hogy a karakterünket melyik másik karakterre írja át a programunk pl:ha a gép 92-öt sorsol akkor a `char *leet[0]` tömb első elemére cserélünk. Az `int found` változónak az a feladat hogy jelzni hogy a tömbünkbe megtalálható-e a karakterünk ha nem akkor visszaadjuk úgy ahogy van.

```
{
%%
. {

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

        if(l337dlc7[i].c == tolower(*yytext))
        {

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

            if(r<91)
                printf("%s", l337dlc7[i].leet[0]);
            else if(r<95)
                printf("%s", l337dlc7[i].leet[1]);
            else if(r<98)
                printf("%s", l337dlc7[i].leet[2]);
            else
                printf("%s", l337dlc7[i].leet[3]);
```

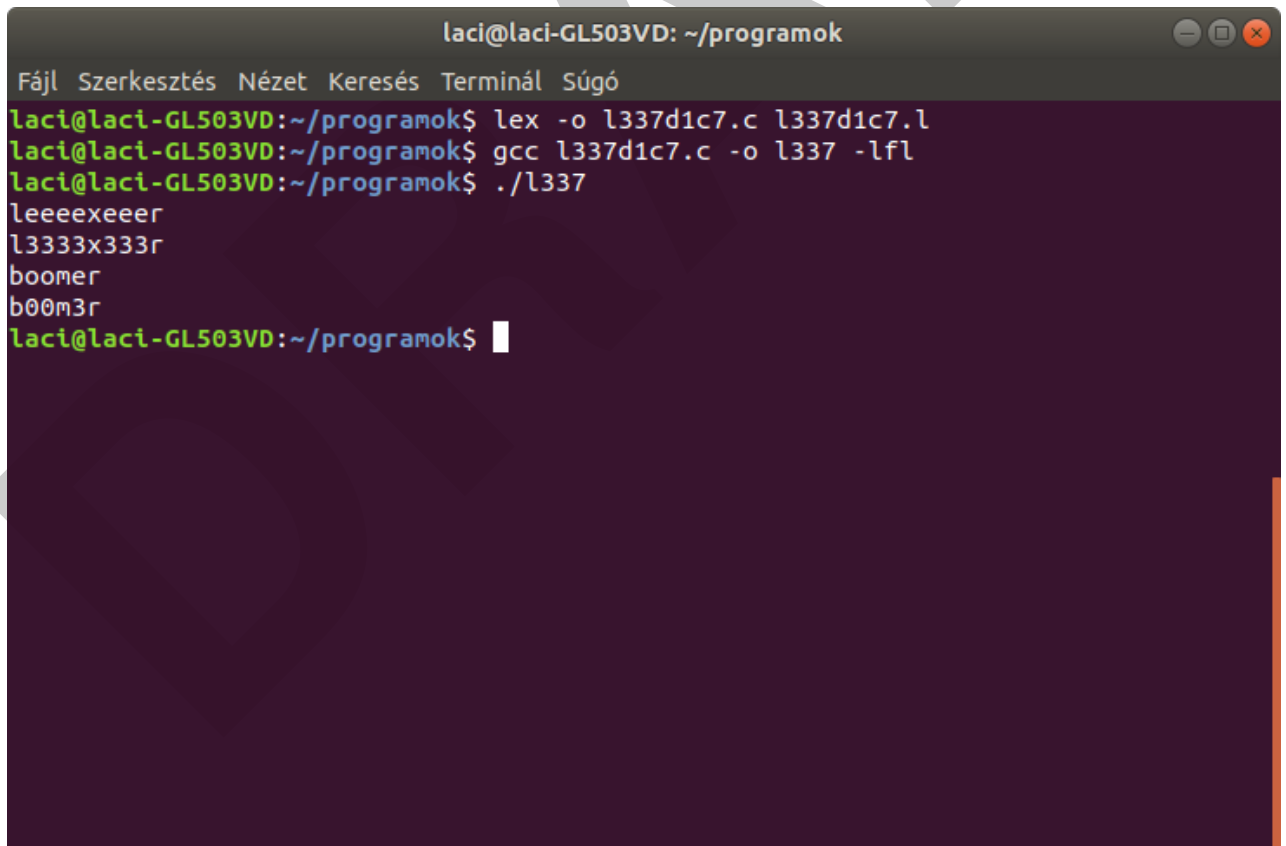
```
        found = 1;
        break;
    }

}

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

A legutolsó részben elindítjuk a lexelést.

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



The screenshot shows a terminal window titled "laci@laci-GL503VD: ~/programok". The terminal displays the following commands and output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ lex -o l337d1c7.c l337d1c7.l
laci@laci-GL503VD:~/programok$ gcc l337d1c7.c -o l337 -lfl
laci@laci-GL503VD:~/programok$ ./l337
leeeexeeer
l3333x333r
boomer
b00m3r
laci@laci-GL503VD:~/programok$
```

3.6. A források olvasása

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

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

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



Bugok

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

i.

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

Ez azt jelenti, hogyha eddig nem volt figyelmen kívül hagyva a SIGINT jel, akkor a jelkezelő függvény kezelje. Ellenkező esetben hagyjuk figyelmen kívül.

ii.

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

Ez egy teljesen alap for ciklus ahol első iterációba az i az 0 majd ellenőrizzük hogy i kisebb mint 5 és minden körrel növeljük az i-t 1-el

iii.

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

Gyakorlatilag ez a for ciklus megegyezik a felette lévővel de nem mindegy melyik oldalra tesszük a ++-okat mert ha bal oldalt tesszük akkor megváltoztatjuk a változó értékét de az i++-al nem változik a változó értéke

```
i = 1;
j = ++i;
(i az 2, j az 2)
```

```
i = 1;
j = i++;
(i az 1, j az 2)
```

iv.

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

Ez egy bugos program mert egyszerre hozzuk létre az i-t és hivatkozunk a tomb[i]-re és ez nem fog lefutni.

v.

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

Ez a program is bugos mivel értékadó operátort használunk ezért a `&&` jobb oldalán nem egy logikai operandus áll

vi.

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

Ez a kódcsipet is hibás mivel két `int`-et adunk de nincs kiértékelési sorrendjük

vii.

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

Ez a programrész nincs baj kiírjuk az a értékét utánna meg a módosított a-t

viii.

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

Ennek a kódcsipetnek is kiértékelési problémája van.

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Λ nyelvű formulákat?

```

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

```

```

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

```

```

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

```

```

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

```

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

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

1. Minden számnál van nagyobb prím szám.
2. Minden számnál létezik nagyobb ikerprím számpáros.
3. Van olyan szám amelynél minden prím szám kisebb.
4. Van olyan szám aminél bármely nagyobb szám nem prím.

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Mit vezetnek be a programba a következő nevek?

- egész

```
int a;
```

- egészre mutató mutató

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

- egész referenciája

```
int &c = a;
```

- egészek tömbje

```
int d[5];
```

- egészek tömbjének referenciája (nem az első elemé)

```
int (&tr)[2] = c;
```

- egészre mutató mutatók tömbje

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

- egészre mutató mutatót visszaadó függvény

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

- egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

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

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

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

Vezesd be egy programba (forduljon le) a következőket:

- ```
int a;
```

- `int *b = &a;`
- `int &r = a;`
- `int c[5];`
- `int (&tr)[5] = c;`
- `int *d[5];`
- `int *h ();`
- `int *(*l) ();`
- `int (*v (int c)) (int a, int b)`
- `int (*(z) (int)) (int, int);`

Megoldás videó:

Megoldás forrása:

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

```
#include <stdio.h>

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

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

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

```
 return sum;

 }

 int
 main ()
 {

 int (*f) (int, int);

 f = sum;

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

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

 g = sumormul;

 f = *g (42);

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

 return 0;
 }
```

```
#include <stdio.h>

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

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

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

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

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

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

 G g = sumormul;

 f = *g (42);

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

 return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...



## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\* háromszögmátrix

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

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

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

Elsőnek tisztáznunk kell mi is az a háromszögmátrix. Háromszögmátrixnak 2 tulajdonsága van az első hogy négyzetes, tehát a sorai és oszlopai száma megegyeznek a második tulajdonsága az hogy a főátlója alatt csupa nulla szerepel a mi mátrixunk az egy alsó mátrix. Most már tudjuk mit várunk el a programtól nézzünk is bele a forrásba.

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

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

Először is include-áljuk a megszokott header file-okat. Ezek után létrehozunk egy int típusú nr nevű változót aminek az értéke 5 lesz. A későbbiekben ez fogja majd meghatározni, hogy a kirajzolásnál hány sort fog a program a futtatás után megjeleníteni. Még itt deklaráljuk a \*\*tm pointert is.

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

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

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

Először kiíratjuk a tm memóriacímét majd ezek után az if-en belül látható egy malloc nevezetű függvény ami annyit csinál, hogy helyet foglal a memóriában jelen esetben 40 bájtot, mert az nr értéke 5 a double 8 így ha összeszorozzuk a kettőt, akkor megkapjuk a 40 bájtot. A malloc alapból egy void típust ad vissza vagyis ha típuskényszerítést alkalmazunk, akkor bármilyen tetszőleges típust vissza tud adni. Jelen esetben ezt is használtuk, ezért látható itt a double \*\* mert ezzel elérjük azt hogy a visszatérés típusa egy double \*\* legyen. Az if még annyit tesz, hogy leellenőrzi, hogy a malloc sikeresen lefoglalta-e a helyet a memóriában és hogy sikeresen vissza adta-e a double \*\* mutatót. Ha ez nem sikerült, akkor egyenlő a NULL-al, vagyis nem mutat sehová és kilép a programból, de ha sikerült akkor kiírjuk a memóriacímét.

```
for (int i = 0; i < nr; ++i)
{
 if ((tm[i] = (double *) malloc ((i + 1) * sizeof (←
double))) == NULL)
 {
 return -1;
 }
}
printf("%p\n", tm[0]);
```

A forciklussal végigmegyünk az 5 soron és mindennyik sorban újra megtörténik a memóriefoglalás. Például nézzük az alábbi esetet amikor is az i=2: A malloc a tm[2]-nek most (2+1 \* 8) bájtot foglal, vagyis a harmadik sorban három 8 bájtnyi helyet foglal le. Ez az egész egy if feltételeként szerepel, melyben ismét ellenőrizzük, hogy sikeres volt-e a helyfoglalás. Ha sikerrel jártunk, akkor kiíratjuk a tm[0] memóriacímét.

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

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

A következő forciklusokkal létrehozuk az alsó háromszögmátrixot. A for cikluson belül értéket adunk a harmadik sori int-eknek. Az i-vel megyünk a 4-ig, vagyis nr-1-ig, j-vel pedig mindig 0-tól i-ig. Az i jelöli a sorok számát, a j pedig az oszlopokét. Mátrix minden eleméhez a sorszám\*(szorszám+1)/2+oszlopszám, és ezzel megkapjuk a feladat legelején felvázolt mátrixot, amit a következő for-ban már csak elemenként kiíratunk.

```
tm[3][0] = 42;
(* (tm + 3)) [1] = 43; // mi van, ha itt hiányzik a külső ←
()
*(tm[3] + 2) = 44;
* (* (tm + 3) + 3) = 45;
for (int i = 0; i < nr; ++i)
{
 for (int j = 0; j < i + 1; ++j)
 printf ("%f, ", tm[i][j]);
```

```

 printf ("\n");
 }

```

A tm 3. sorának első elemének értékét 42-re módosítjuk. Utána a harmadik sor második elemének az értékét változtatjuk, majd a harmadik sor harmadik elemét, végül pedig a harmadik sor negyedik elemét. A második lehetőségénél felmerül a kérdés, hogy elhagyható-e a külső zárójel. Elhagyható viszont, így nem a harmadik sorba lesz a módosítás, hanem a 4. sor első eleménél, mivel  $*(tm + 3)[1]$  azzal ekvivalens, hogy  $*(tm+4)$ . Az értékek megváltoztatása után újra kiíratjuk az alsó háromszögmátrixunkat a forciklus segítségével.

```

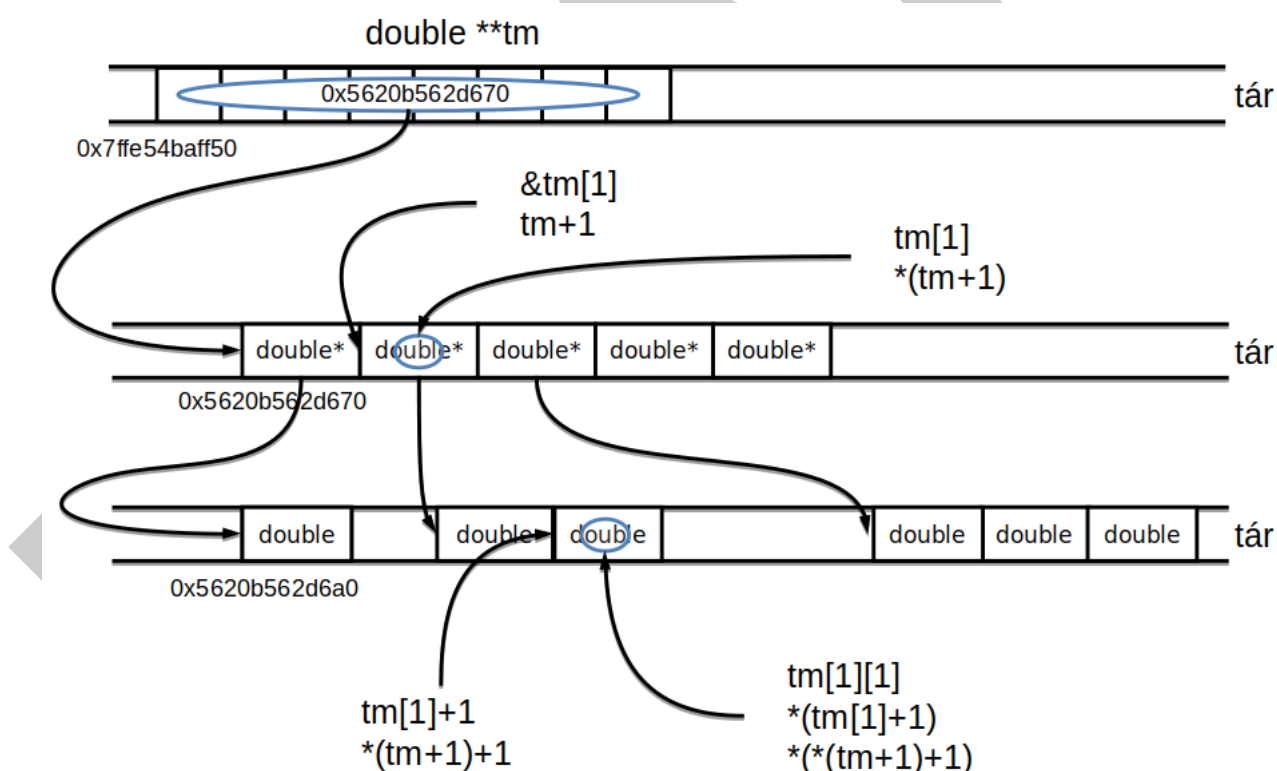
 for (int i = 0; i < nr; ++i)
 free (tm[i]);

 free (tm);

 return 0;
}

```

A program utolsó részében a free függvény segítségével felszabadítjuk az egyes sorokban illetve az egész tm által foglalt memóriát.



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

```
laci@laci-GL503VD: ~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textbook_Ig...
Fájl Szerkesztés Nézet Keresés Terminál Súly
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textboo
IgyNeveldaProgramozod/Caesar$ ls
tm.c
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textboo
IgyNeveldaProgramozod/Caesar$ gcc tm.c -o tm
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textboo
IgyNeveldaProgramozod/Caesar$./tm
0x7fffd0742770
0x556ea2bcf670
0x556ea2bcf6a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textboo
IgyNeveldaProgramozod/Caesar$
```

Tanulságok, tapasztalatok, magyarázat...

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: egy részletes feldolgozása az [e.c](#) és [t.c](#) forrásoknak.

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

Az EXOR titkosító lényegében a logikai vagyra, azaz a XOR műveletre utal, mely bitenként összehasonlítja a két operandust, és mindig 1-et ad vissza, kivéve, amikor az összehasonlított 2 Bit megegyezik, mert akkor nullát. Tehát 2 operandusra van szükségünk, ez jelen esetben a titkosítandó bemenet, és a titkosításhoz használt kulcs. Ideális esetben a kettő mérete megegyezik, így garantálható, hogy szinte feltörhetetlen kódot kapunk, mivel túl sokáig tart annak megfejtése. Viszont ha a kulcs rövidebb, mint a titkosítandó szöveg, akkor a kulcs elkezd ismétlődni, ami biztonsági kockázatot rejt magában.

```
#define MAX_KULCS 100
#define BUFFER_MERET 256
```

Először a kulcs és a buffer méretének maximumát konstansban tároljuk el, ezek nem módosíthatóak.

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

A main függvénynek argumentumokat adunk, argc-vel adjuk át az argumentumok számát, és az argumentumokra mutató mutatókat pedig az argv tömbben tároljuk el.

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

A `main()` belül deklarálunk két tömböt, egyikbe a kulcsot tároljuk, a másikban pedig a beolvasott karaktereket, mind a kettőnek a mérete korlátozott.

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

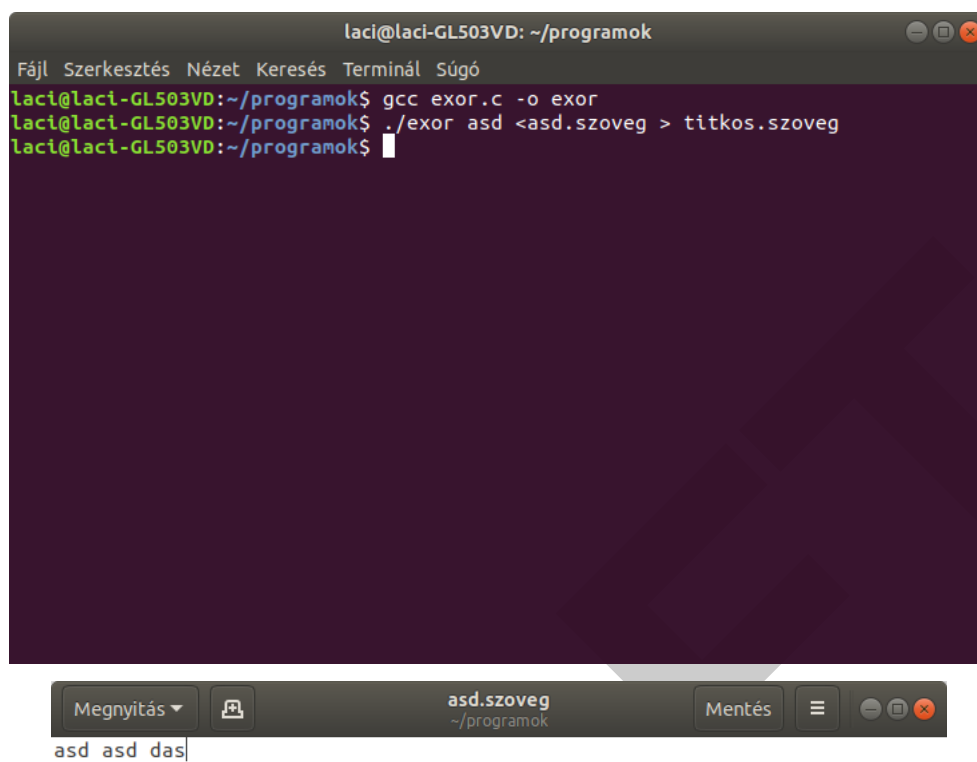
int kulcs_meret = strlen (argv[1]);
strncpy (kulcs, argv[1], MAX_KULCS);
```

Létrehozunk számlálót, melyek segítségével bejárjuk majd a kulcs tömböt, és számoljuk a beolvasott bajtokat. A kulcs méretét a `strlen()` függvénnyel kapjuk meg, amely visszadja a másodjára megadott érték hosszát. Ezután a `strncpy()` függvénnyel átmásoljuk az `argv[1]`-ben tárolt sztringet karakterenként a kulcs tömbbe, lényegében mindegyikhez visszaad egy pointert. A `MAX_KULCS`-sal pedig meghatározzuk, hogy mennyi karaktert másoljon át.

```
while ((olvasott_bajtok = read (0, (void *) buffer, ←
 BUFFER_MERET)))
{
 for (int i = 0; i < olvasott_bajtok; ++i)
 {
 buffer[i] = buffer[i] ^ kulcs[kulcs_index];
 kulcs_index = (kulcs_index + 1) % kulcs_meret;
 }

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

A `while` ciklus feltétele addig lesz igaz, ameddig a `read` parancs beolvassa a megadott mennyiségű bajtokat. A `read` 3 argumentumot kap: az első az, hogy honnan olvassuk be a bajtokat, jelen esetben a standard input-ról olvasunk, a beolvasott bajtokat a `buffer`-ben tároljuk egészen addig, ameddig el nem érjük a megadott mennyiséget, amit `BUFFER_MERET` definiál. Ezután pedig végigmegyünk elemenként a `buffer`-ben eltárolt karaktereken és össze EXOR-ozzuk a kulcs tömb megfelelő elemével, majd inkrementáljuk a `kulcs_index`-et 1-el, mely egészen addig nő, ameddig el nem érjük a `kulcs_meret`-et, ekkor lenullázódik. Végezetül pedig kiírjuk a `buffer` tartalmát a standard outputra.

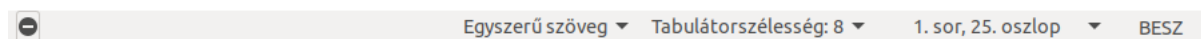
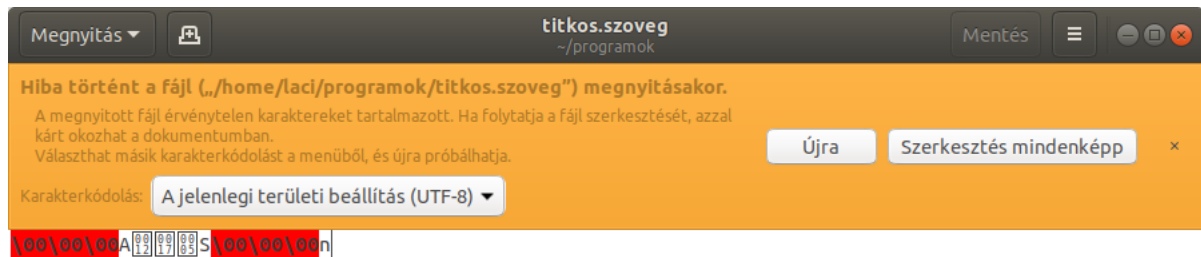


The image shows two overlapping windows from a Linux desktop environment. The top window is a terminal titled 'laci@laci-GL503VD: ~/programok'. It contains the following commands and output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ gcc exor.c -o exor
laci@laci-GL503VD:~/programok$./exor asd <asd.szoveg > titkos.szoveg
laci@laci-GL503VD:~/programok$
```

The bottom window is a text editor titled 'asd.szoveg ~/programok'. It shows the text 'asd asd das' on a single line. The editor's menu bar includes 'Megnyitás', 'Mentés', and other standard icons. A large, light gray watermark 'DRAFT' is visible across the lower half of the page.

Egyszerű szöveg ▾ Tabulátorszélesség: 8 ▾ 1. sor, 12. oszlop ▾ BESZ



### 4.3. 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)

Tanulságok, tapasztalatok, magyarázat...

A legfontosabb dolog hogy a java az egy objektum orientált programozási nyelv azaz létre tudunk hozni objektumokat másnéven class-okat amikkel bizonyos utasításokat végezhetünk. Javában a classok nem különülnek el mint c++-ba hanem main() függvénybe számítanak. A jexor függvény meghívjuk a input és outstream függvényeket hogy tudjuk majd olvasni a bájtokat és ha ez nem sikerül throws hibát fog kiírni. Ezek után byte-okból álló tömböt hozunk létre kulcs és buffer néven. A getBytes() függvénnyel olvassuk be a kulcsot a kulcs tömbbe. A buffer tömbnek foglaltunk 256 bájt-nyi helyet a memóriába. Majd definiáljuk a kulcs tömb bejárásához, és a beolvasott bájtok számlálására. A while ciklus addig fog menni amíg nem olvassuk be a buffert vagy már nem tudunk többet beolvasni. Majd a beágyazott for ciklussal elemenként összeszorozzuk a buffer tartalmát a kulccsal és növeljük a kulcsindexet a % operátorral. Ennek következtében ha elérjük a kulcs hosszát, akkor lenullázódik.

```
public class jexor {

 public jexor(String kulcsSzoveg,
 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 olvasottBájtok = 0;

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

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

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

 }

 kimenőCsatorna.write(buffer, 0, olvasottBájtok);

}

}
```

Ebben a részben csak meghívjuk ez elején deklarált jexort és futtatjuk a try catch pedig ha valami baj történik hibát fog kidobni. Ha nem adunk meg kulcsot akkor hibát fogunk kapni.

```
public static void main(String[] args) {

 try {

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

 } catch(java.io.IOException e) {

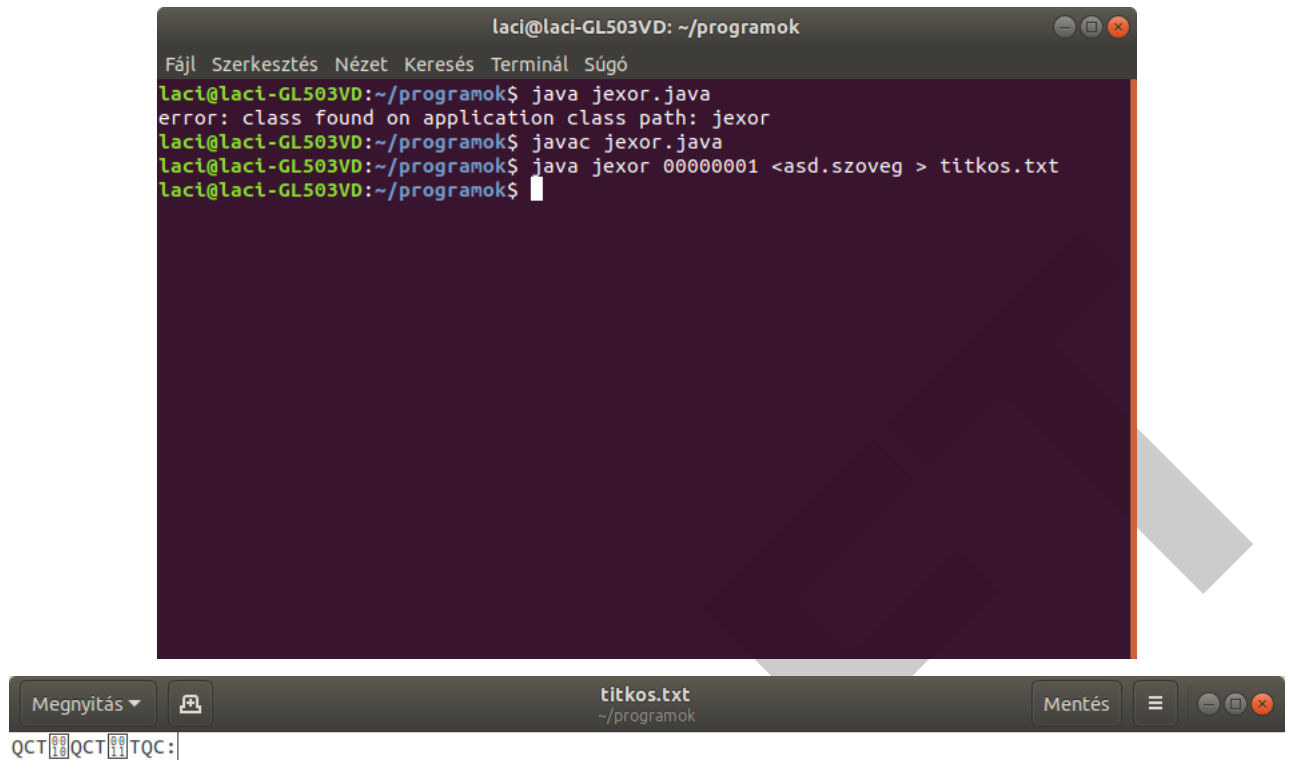
 e.printStackTrace();

 }

}

}
```





```
laci@laci-GL503VD: ~/programok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ java jexor.java
error: class found on application class path: jexor
laci@laci-GL503VD:~/programok$ javac jexor.java
laci@laci-GL503VD:~/programok$ java jexor 00000001 <asd.szoveg > titkos.txt
laci@laci-GL503VD:~/programok$
```

Megnyitás titkos.txt Mentés

QCT QCT TQC:

## 4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

Mint az exorban is itt is meghatározunk bizonyos konstansokat, és includoljuk a header fájlokat.

```
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;
}
```

Az `atlagos_szohossz` függvénnyel kiszámoljuk a bemenetünk átlagos szóhosszát, odaadunk neki egy tömböt és a tömbnek a méretét. Majd a `for` ciklussal körbejárjuk a tömböt és minden elem után hozzáadunk 1-et az `sz`-hez. `Return`-ként pedig a tömb méretét/`sz` értéket adjuk.

```
int
tisza_lehet (const char *titkos, int titkos_meret)
{
 // a tiszta szoveg 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");
}
```

Ebbe `tisza_lehet` függvényben azt akarnánk ellenőrizni hogy a bemenetként adott szövegünk tiszta szöveg e. Ezt úgy ellenőrizzük hogy megnézzük hogy tartalmazza e a szövegünk a leggyakoribb magyar szavakat. Ám mi van ha szövegünk nem tartalmazza a kért szavakat de mégis tiszta szöveg kap nem tudja visszafejteni.

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

```
int kulcs_index = 0;

for (int i = 0; i < titkos_meret; ++i)
{
 titkos[i] = titkos[i] ^ kulcs[kulcs_index];
 kulcs_index = (kulcs_index + 1) % kulcs_meret;
}

}
```

Az exor függvény ugyan azt csinálja mint az exor titkosító mivel ha valamit 2x exorozunk akkor visszakapjuk a tiszta szöveget. Az argumentumban kap egy lehetséges kulcsot és annak a méretét meg a titkos szöveget a méretével.

```
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);
}
```

Az exor\_tores meghívja az előtte definiált függvényeket és vagy 0-t vagy 1-et ad vissza ami attól függ hogy a szövegünk tiszta vagy nem.

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

Most a mainben elsőnek deklarálunk egy kulcs tömböt és egy titkos tömböt melyeknek a mérete a program elején megadott konstansok. Még definiálunk egy mutatót, mely a titkos tömbre mutat és létrehozunk egy számlálót az olvasott\_bajtok néven.

```
// 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;
```

Ezzel a while ciklussal addig olvassuk a bájtokat amíg a buffer tele nem lesz.

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

A for ciklussal kinullázzuk a buffer megmaradt helyeit és előállítjuk az összes létező kulcsot.

```
// 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%c]\nTiszta ←
 szoveg: [%s]\n",
 ii, ji, ki, li, mi, ni, oi, pi, ←
 titkos);

 // ujra EXOR-ozunk, így nem kell egy ←
 masodik buffer
 exor (kulcs, KULCS_MERET, titkos, p - ←
 titkos);
 }

return 0;
}
```

Itt a végén lefuttatjuk az összes kulcsot és meghívjuk az exor\_tores ←  
függvényt és ha igazat ad  
kiirajta a szöveget és a használt kulcsot

## 4.5. Neurális OR, AND és EXOR kapu

R

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

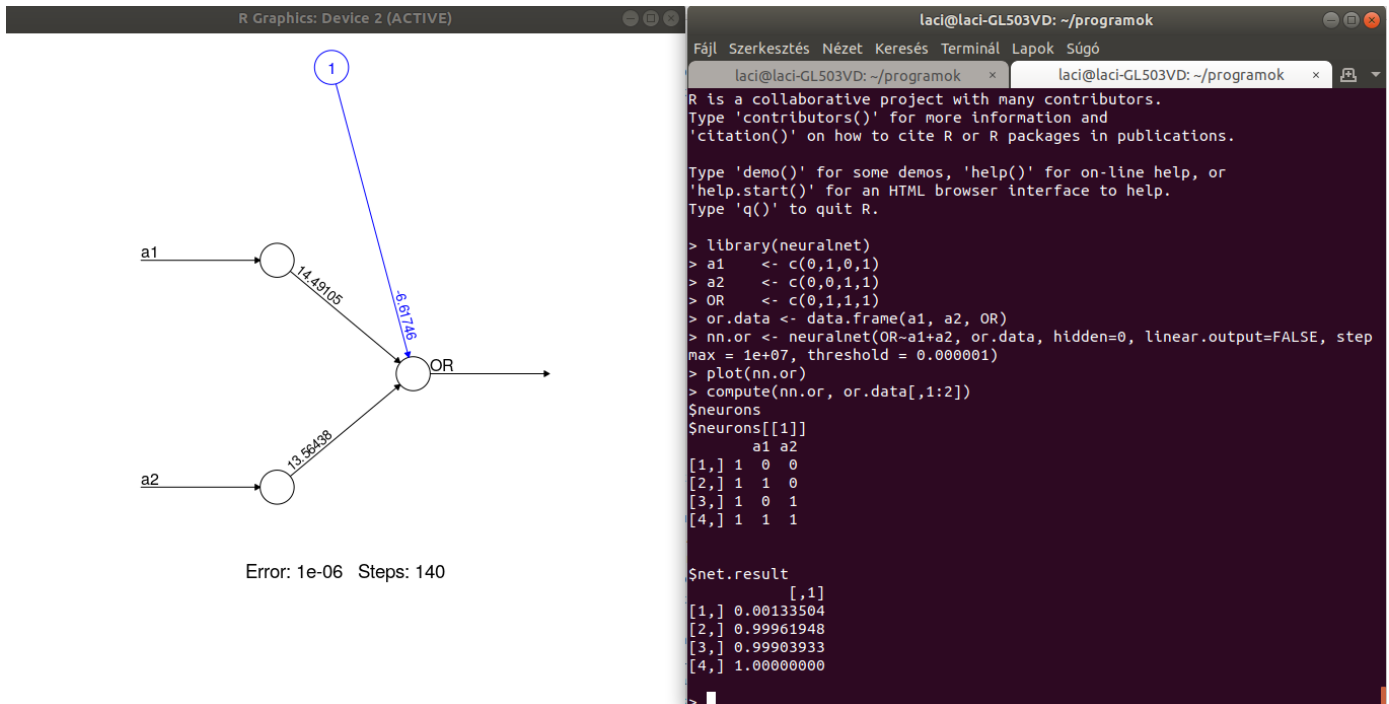
Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/NN\\_R](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R)

Tanulságok, tapasztalatok, magyarázat...

OR

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

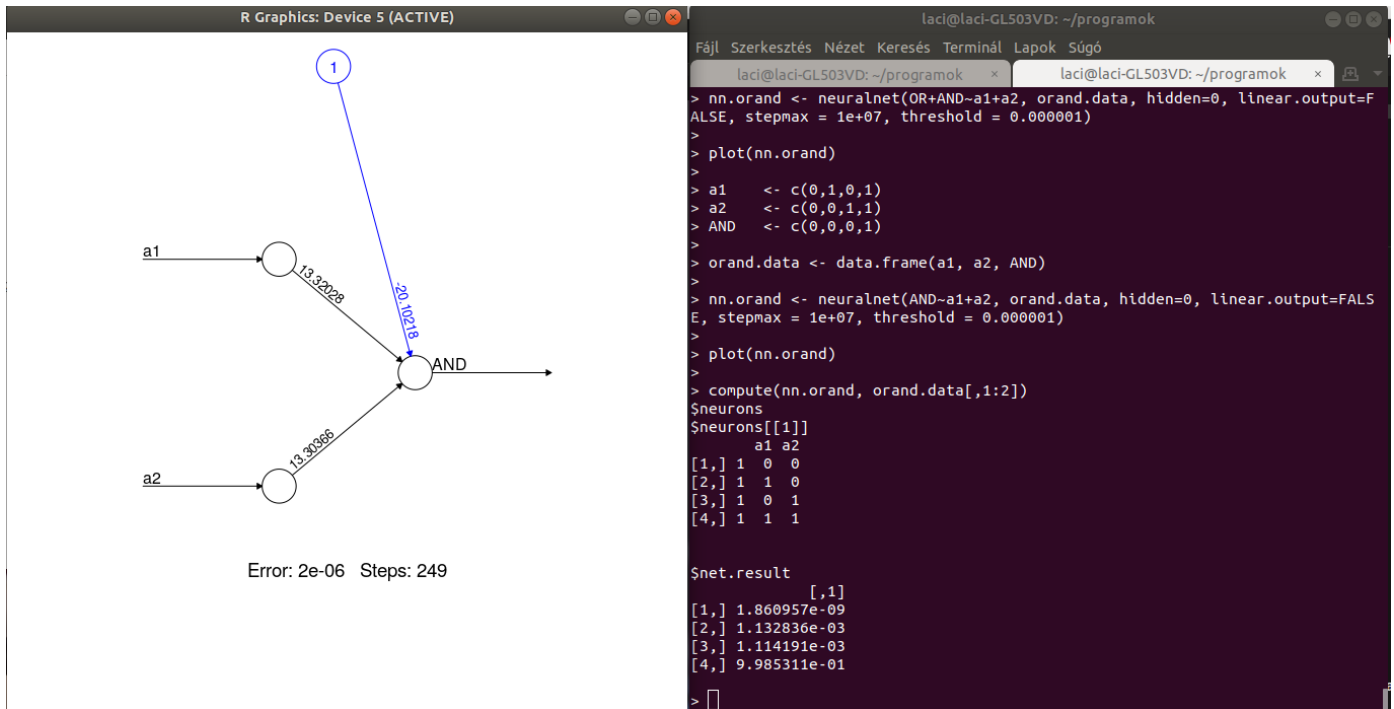
Az a1 és az a2-ben megadtunk 0-k és 1-sek sorozatát majd az OR-ba az "a1 or a2" művelet eredményét, majd ebből adatot csinálunk az or.data változóba. Az nn.or változóba használjuk a neuralnet függvényt, amely kiszámolja nekünk a neutrális hálót. Átadjuk neki az adatokat (amit az előbb előállítottunk az a1, a2 és az OR-ból), tehát megtanítjuk neki ezt az OR műveletet, majd saját magát továbbfejlesztve beállítja a súlyokat úgy hogy megtanulja a dolgok menetét. Fontos hogy a neuralnet függvényt nem a logikai művelet végzi el, hanem tényleges tanulás után próbálja megmondani az eredményt nekünk. A plot függvénnyel kirajzoltatjuk az első csatolt képen látható ábrát, ami illusztrálja a számításokat. Itt látható az ismétlések száma, hogy hányszor végezte a műveletet (steps) és hogy ezekből hányszor kapott rossz eredményt (errors, nagyon kicsi). A compute paranccsal beadjuk neki az adatokat és számításra utasítjuk a függvényünket, tehát kikérdezzük tőle azt amit megtanítottunk neki az elején. És ahogy a második kép mutatja úgyesen megtanulta és vissza is adta a helyes eredményeket.



## AND

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

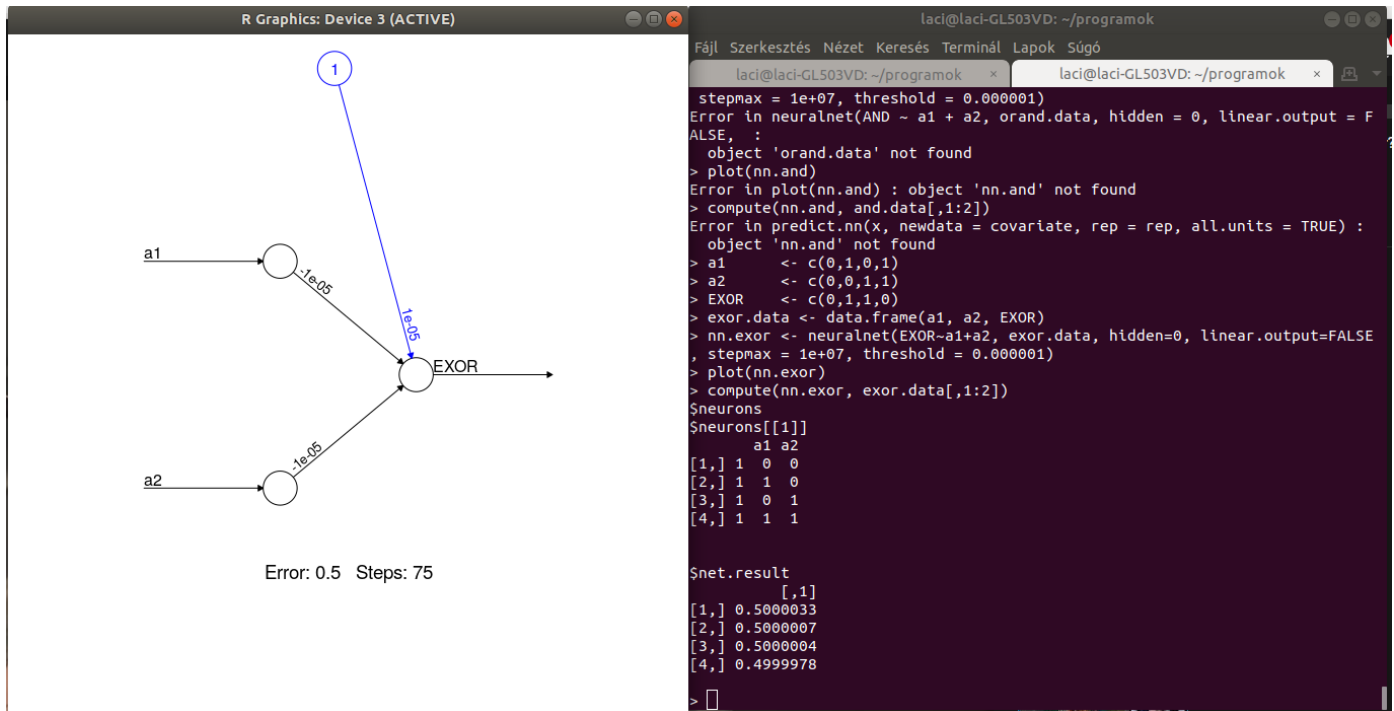
Ugyan ez folyik le az AND műveletnél is.



## EXOR

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

Ezután pedig jönne az EXOR, viszont ez már nem annyira egyszerű. Amikor régen ezt a technológiát kitalálták, és az EXOR nem működött, sokan elpártoltak tőle. Majd a kor nagy matematikusai megfejtették, hogy nem lehetetlen feladat, csak egy apróságra van szükség, létre kell hozni a rejtett neuronokat, melyek segítik a tanulást.



### EXOR neuronokkal

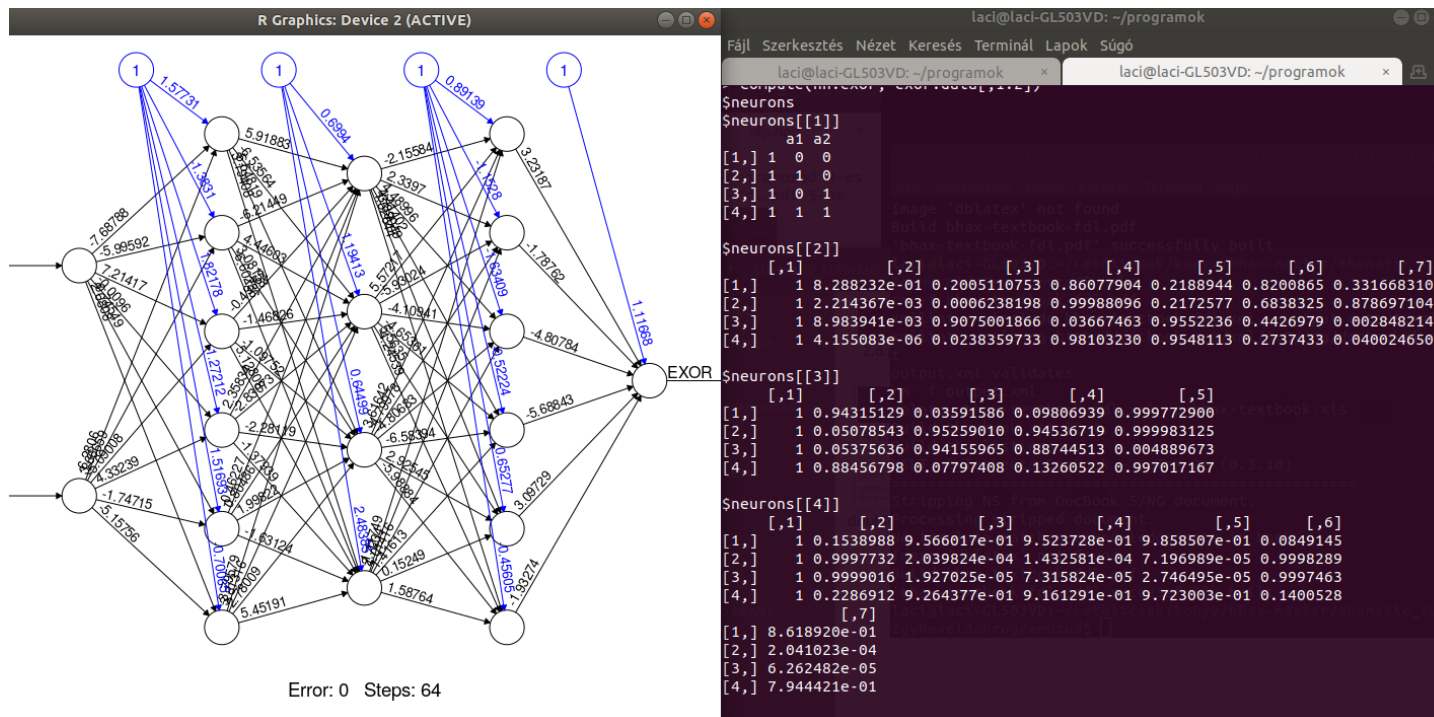
```

a1 <- c(0,1,0,1)
a2 <- c(0,0,1,1)
EXOR <- c(0,1,1,0)
exor.data <- data.frame(a1, a2, EXOR)
nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. <-
 output=FALSE, stepmax = 1e+07, threshold = 0.000001)
plot(nn.exor)
compute(nn.exor, exor.data[,1:2])

```

Most hogy 6 4 6 neuron beállítást adtunk meg a programunk hibarátlója lement 0.5-ről 0-ra esett.





## 4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

Tanulságok, tapasztalatok, magyarázat...

```
#include <iostream>
#include "perceptron.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;

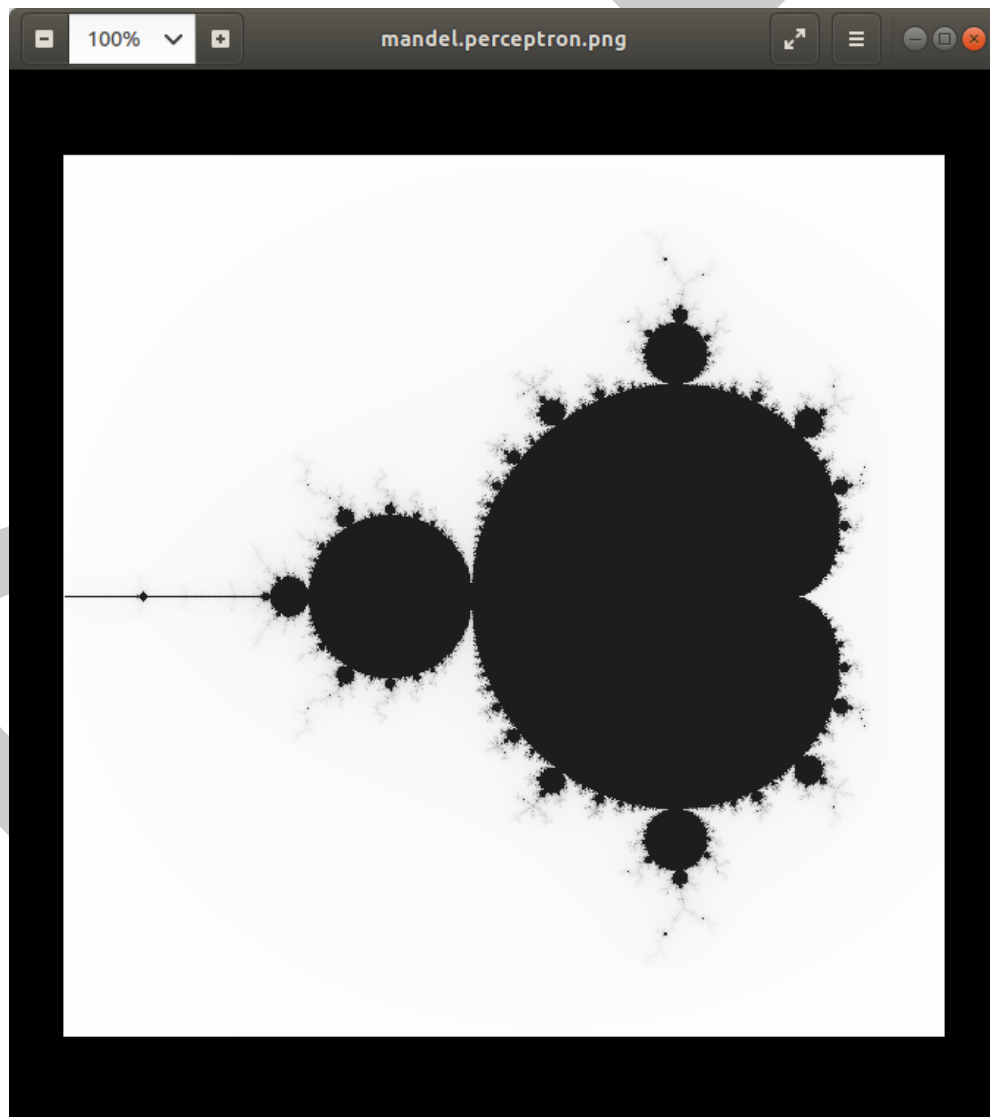
 double value = (*p) (image);
```

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

delete p;
delete [] image;

}
```

A fentebb említett perceptron.cpp programot a mainben meg is hívjuk header fájlként, átláthatóbbá téve a főprogramot. A fő számítások viszont a perceptron.hpp-ben vannak, a mainben az ott deklarált Perceptron osztályt hívjuk segítségül meg az eredmény kiszámolásának céljából. A mainben a png.hpp header fájl segítségével létrehozunk egy új png kiterjesztésű képet, ugyanolyan szélességgel és magassággal mint a mandelbrotos kép volt. A két egymásbaágyazódó for ciklus segítségével végigmegyünk a kép minden pixelén és az előzőekben lementett mandel\_perceptron.png pixeleinek piros (red) komponenseit rámásoljuk a most létrehozott kép pixeleire. A program végén pedig kiíratjuk ezt a perceptron értéket a value változó segítségével.



```
laci@laci-GL503VD: ~/programok/perceptron
```

Fájl Szerkesztés Nézet Keresés Terminál Súgó

```
laci@laci-GL503VD:~/programok/perceptron$ g++ mandelpng.cpp -o mandel -lpng
laci@laci-GL503VD:~/programok/perceptron$./mandel mandel.perceptron.png
Szamitas.....
.....
.....
.....
.....
.....
.....
.....
.....
.....mandel.perceptron.png mentve
laci@laci-GL503VD:~/programok/perceptron$ g++ perceptron.hpp main.cpp -o main -l
png -std=c++11
laci@laci-GL503VD:~/programok/perceptron$./main mandel.perceptron.png
0.506489
laci@laci-GL503VD:~/programok/perceptron$ █
```

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása: [bhax/attention-raising/CUDA/mandelpngt.c++](https://github.com/bhax/attention-raising-CUDA-mandelpngt.c++) nevű állománya.

A Mandelbrot halmaz a komplex síkon

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

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

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

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

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

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

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

Megoldás forrása:

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

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

```
#include <complex>

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

 int szelesseg = 1920;
 int magassag = 1080;
 int iteraciosHatar = 255;
 double a = -1.9;
 double b = 0.7;
 double c = -1.3;
 double d = 1.3;

 if (argc == 9)
 {
 szelesseg = atoi (argv[2]);
 magassag = atoi (argv[3]);
 iteraciosHatar = atoi (argv[4]);
 a = atof (argv[5]);
 b = atof (argv[6]);
 c = atof (argv[7]);
 d = atof (argv[8]);
 }
 else
 {
 std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↵
 " << std::endl;
 return -1;
 }

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

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

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

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

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

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

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

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

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

 ++iteracio;
}

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

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

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

## 5.3. Biomorfok

Megoldás videó: <https://youtu.be/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 halmaz** és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a *c* változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a *c* befutja a vizsgált összes rácspontot.

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

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

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

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

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

 ++iteracio;
}
```

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

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

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

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

```
// Verzio: 3.1.3.cpp
// Forditas:
```



```
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ←
 BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
 color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ←
 Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
 iteracio = i;
 break;
 }
}

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

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

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

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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

## 5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása:

## 5.6. Mandelbrot nagyító és utazó Java nyelven

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

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

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

### 6.2. LZW

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

Megoldás videó:

Megoldás forrása:

### 6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

### 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

## 6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

## 6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

Python

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

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa  
[https://progpater.blog.hu/2016/11/13/hello\\_samu\\_a\\_tensorflow-bol](https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol)

Tanulságok, tapasztalatok, magyarázat...

### 8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...



## 9. fejezet

# Helló, Chaitin!

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

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

Megoldás forrása:

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

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

Megoldás videó: [https://youtu.be/OKdAkI\\_c7Sc](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...

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

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

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

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Programozási alapfogalmak

[?]

### 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

### 10.3. Programozás

[BMECPP]

## **III. rész**

### **Második felvonás**

**Bátf41 Haxor Stream**

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

---

DRAFT

---

## 11. fejezet

# Helló, Arroway!

### 11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## **IV. rész**

### **Irodalomjegyzék**

DRAFT

## 11.3. Általános

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

## 11.4. C

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

## 11.5. C++

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

## 11.6. Lisp

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

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

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