

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. március 16.	

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	11
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	14
2.6. Helló, Google!	15
2.7. A Monty Hall probléma	16
2.8. 100 éves a Brun tétel	18
3. Helló, Chomsky!	22
3.1. Decimálisból unárisba átváltó Turing gép	22
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	23
3.3. Hivatkozási nyelv	23
3.4. Saját lexikális elemző	24
3.5. Leetspeak	25
3.6. A források olvasása	28
3.7. Logikus	29
3.8. Deklaráció	29

4. Helló, Caesar!	34
4.1. double ** háromszögmátrix	34
4.2. C EXOR titkosító	37
4.3. Java EXOR titkosító	38
4.4. C EXOR törő	39
4.5. Neurális OR, AND és EXOR kapu	42
4.6. Hiba-visszaterjesztéses perceptron	42
5. Helló, Mandelbrot!	43
5.1. A Mandelbrot halmaz	43
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	44
5.3. Biomorfok	46
5.4. A Mandelbrot halmaz CUDA megvalósítása	50
5.5. Mandelbrot nagyító és utazó C++ nyelven	50
5.6. Mandelbrot nagyító és utazó Java nyelven	51
6. Helló, Welch!	52
6.1. Első osztályom	52
6.2. LZW	52
6.3. Fabejárás	52
6.4. Tag a gyökér	52
6.5. Mutató a gyökér	53
6.6. Mozgató szemantika	53
7. Helló, Conway!	54
7.1. Hangyaszimulációk	54
7.2. Java életjáték	54
7.3. Qt C++ életjáték	54
7.4. BrainB Benchmark	55
8. Helló, Schwarzenegger!	56
8.1. Szoftmax Py MNIST	56
8.2. Mély MNIST	56
8.3. Minecraft-MALMÖ	56

9. Helló, Chaitin!	57
9.1. Iteratív és rekurzív faktoriális Lisp-ben	57
9.2. Gimp Scheme Script-fu: króm effekt	57
9.3. Gimp Scheme Script-fu: név mandala	57
10. Helló, Gutenberg!	58
10.1. Programozási alapfogalmak	58
10.2. Programozás bevezetés	58
10.3. Programozás	58
III. Második felvonás	59
11. Helló, Arroway!	61
11.1. A BPP algoritmus Java megvalósítása	61
11.2. Java osztályok a Pi-ben	61
IV. Irodalomjegyzék	62
11.3. Általános	63
11.4. C	63
11.5. C++	63
11.6. Lisp	63

Ábrák jegyzéke

2.1. A B_2 konstans közelítése	21
4.1. A <code>double **</code> háromszögmátrix a memóriában	36
5.1. A Mandelbrot halmaz a komplex síkon	43

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 (hogyan lássuk mást is) példával.

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

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 lényegi mondanivalója a példának az, hogy az önhivatkozó rendszerek általában hibásak, vagy már koncepcionálisan is ellentmondásosak, akár egy program amely megmondja bármely programról, hogy az lefagy-e? (Akár saját magát is beleértve.)

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:

```
#include <stdio.h>

int main()
{

    int a;
    int b;

    printf("Enter two numbers: \n");

    scanf("%d", &a);
    scanf("%d", &b);

    printf("%s", "a=");
    printf("%d", a);

    printf("%s", " ");

    printf("%s", "b=");
    printf("%d", b);

    printf("%s", "\n");
    printf("%s", "\n");

    b = b - a;
    a = a + b;
    b = a - b;

    printf("%s", "a=");
    printf("%d", a);

    printf("%s", " ");

    printf("%s", "b=");
    printf("%d", b);

    printf("%s", "\n");
    printf("%s", "\n");

    a = a * b;
    b = a / b;
    a = a / b;
```

```
printf("%s", "a=");  
printf("%d", a);  
  
printf("%s", "  ");  
  
printf("%s", "b=");  
printf("%d", b);  
  
printf("%s", "\n");  
printf("%s", "\n");  
  
a = a ^ b;  
b = a ^ b;  
a = a ^ b;  
  
printf("%s", "a=");  
printf("%d", a);  
  
printf("%s", "  ");  
  
printf("%s", "b=");  
printf("%d", b);  
  
printf("%s", "\n");  
printf("%s", "\n");  
  
return 0;  
  
}
```

Tanulságok, tapasztalatok, magyarázat...

2 változó értékeinek felcserélése teljesen alapvető feladat. A legegyszerűbb opció egy harmadik (segéd)változó bevezetése lenne, de a feladat kedvéért most aritmetikai megoldásokat választottam. A lényeg, hogy akármelyik megoldást használjuk, az egyszerű és átlátható legyen.

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:

Először nézzük az if-es megoldást:

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

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

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

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, "O" );

        refresh ();
        usleep ( 100000 );

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

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

    }

    return 0;
}
```

Most nézzük meg ugyanezt if nélkül:

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

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

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, "O" );

        refresh ();
        usleep ( 100000 );

        //clear();

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

    for (int i=x;i+xnov>mx-1;i-1)
    {
        xnov=-xnov;
    }

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

    for (int i=y;i+ynov>my-1;i-1)
    {
```

```
ynov=-ynov;
}

for (int i=y;i+ynov<0;i+1)
{
ynov=-ynov;
}

}

return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A második verzió kérdése az volt, hogy mivel kerülhetnénk ki az if-ek használatát. Erre én egy garantáltan 1-szer leforduló for ciklus-t használtam, de számtalan más megoldás is létezhet. (Például a for ciklusainkat probléma nélkül while ciklusokra cserélhetnénk.)

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

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

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

```
#include <stdio.h>

int
main (void)
{
    int integer = 1;
    int counter = 1;

    while ((integer <= 1))
    {
        counter = counter +1;
    }

    printf("%d", counter);

    return 0;
}
```


Tanulságok, tapasztalatok, magyarázat...

A program lényegében felvesz egy 1-es kezdőértéket egy int-nek, ami bányárisan valahogy így néz ki: 000...0001 (tetszőleges számú nulla, amelyet egy 1-es követ). A shift operátor lényegében ezt az 1-es tologatja balra, addig amíg csak tudja, illetve közben számon tartva az eltolások számát. Ez a szám attól függ, hogy a gépünk hány bitet foglal le egy int ábrázolására.

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:

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

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

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

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

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

    int i, j;

    for (;;) {

        for (i = 0; i < 4; i++) {

            PR[i] = 0.0;
            for (j = 0; j < 4; j++)
```

```
        PR[i] += (L[i][j] * PRv[j]);
    }
    if (tavolsag(PR, PRv, 4) < 0.00000001)
        break;

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

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

Tanulságok, tapasztalatok, magyarázat...

A program matematikai háttére sajnos számomra még mindig ködös, de a koncepció az lenne, hogy veszünk egy mindössze néhány honlapból álló zárt modellt, amelyben egy mátrixall reprezentáljuk a bennük lévő linkeket. (Az értékek körülbelül úgy jönnek ki, hogy megnézzük, honnan mutatnak linkek egy adott lapra, majd megnézzük, hogy ezek a linkek mennyire értékesek. Az értéket abból vezetjük le, hogy hány másik link szerepel egy-egy adott honlapon.)

Ezt követően egy végtelen ciklust használunk ami (számomra nem teljesen érthető) matematikai módszerekkel közelíti a honlapok "értékét". A programunk közben figyeli, hogy mikor lesz 2 egymást követő ciklusban az értékek közti különbség kisebb egy adott számnál, majd ha ez teljesül, akkor megáll és kiírja az aktuális értékeket.

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

```
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)
```

Tanulságok, tapasztalatok, magyarázat...

A program alapvetően egy híres matematikai oximoron-ra (megmagyarázható paradoxon-ra) épül. A háttértörténet egy televíziós játékból ered, ahol a versenyző három ajtó előtt áll, amelyből 1 egy nagy értékű nyereményt (autót), míg a másik kettő értéktelen részvételi díjat (egy kecskét) rejt maga mögött. A játékos kiválaszt egy ajtót, majd ezt követően a műsorvezető megmutatja, hogy a másik 2 ajtó közül melyik mögött nincs az autó. Ezt követően megkérdezi a játékost, hogy szeretne-e másik ajtót választani.

A kérdés elsőre banálisnak tűnhet, mivel a józanész azt sugja, hogy a megmaradt 2 ajtó ugyanakkora (50%-50%) eséllyel rejt autót. Azonban a statisztika rácsáfolt a józanészre, ugyanis feltűnt az embereknek, hogy látványosan gyakrabban nyertek azok, akik megváltoztatták eredeti döntésüket. A probléma ezt követően híresült el. Később több statisztikai kísérlet is bizonyította, hogy az esélyek arány nem 50-50, hanem 66-33, azaz ha megváltoztatod a döntésed, akkor kétszer akkora eséllyel nyersz.

A mi programunk ugyanezt a statisztikai számítást végzi el azáltal, hogy nagy számú virtuális játékokat játszik, majd összesíti az eredményeket.

A jelenség magyarázata az, hogy bármely választás esetén 66%-os eséllyel a másik 2 ajtó valamelyike mögött lesz az autó. Az pedig, hogy ezek közül a rosszat felfedik, nem csökkenti ezt az esélyt. (A józanész itt abban tévedhet, hogy amikor 2 ajtó közti választásra redukálja a szituációt rengeteg előzetes már ismert információt hagy figyelmen kívül.)

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:

$(n+1)!+2$, $(n+1)!+3$, ..., $(n+1)!+n$, $(n+1)!+(n+1)$ 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://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R/stp.r) nevű állománya kiszámolja az ikerprímeket, összegzi a reciprokaikat és vizualizálja a kapott részeredményt.

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

library(matlab)

stp <- function(x) {

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

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

Soronk nt  rtelemezz k ezt a programot:

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

Kisz molja a megadott sz mig a pr meket.

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

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

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

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

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

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

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

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

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

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

Az $1/t1primes$ a `t1primes` 3,5,11 értékéből az alábbi reciprokot képi:

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

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

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

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

```
> 1/t1primes+1/t2primes
[1] 0.53333333 0.3428571 0.1678322
```

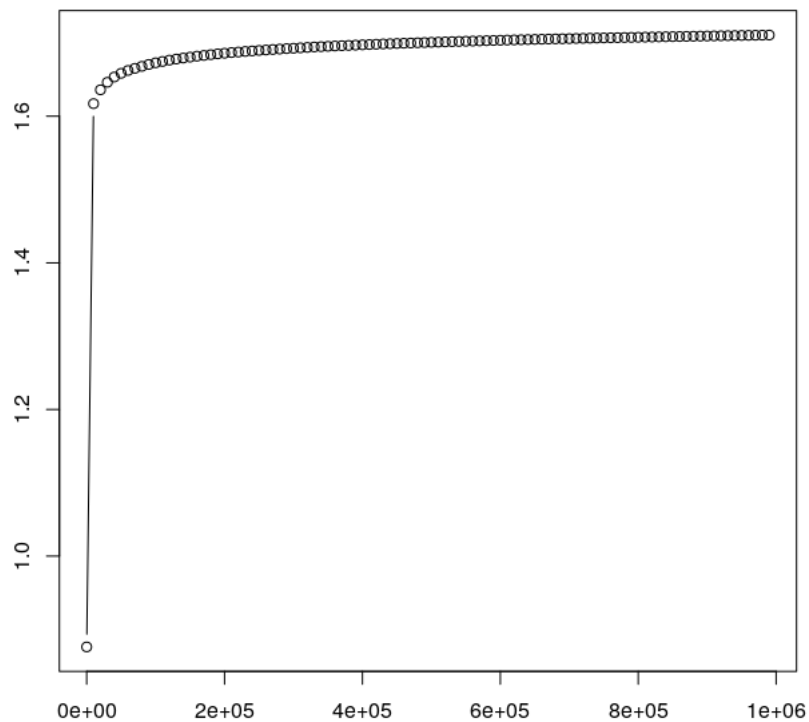
Nincs más dolgunk, mint ezeket a törtet összeadni a `sum` függvénnyel.

```
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 gráfiájával 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!

Én az eredeti (fólia-beli) megoldást nem tudtam értelmezni, így inkább utánna olvastam a neten, hogy hogyan is működnek ezek az ábrák. Sajnos néhány jelölést még ezt követően sem értettem, ezért inkább írtam egy saját megoldást az interneten talált jelöléseket használva.

A saját megoldáson be megtalálható a "Megoldás forrása:" részben jelzett mappában.

Az ábrán röviden végigmennék szövegesen is, hogy érthetőek legyenek a jelölések:

A felső szalag ábrázolja az adataink alapvető formáját: A legelső üres elem azt jelzi, hogy ez az első olyan cella, amely már nem része a kezdeti decimális bemenetünknek. Ezt követi tetszőleges számú decimális számjegy, melyet egy = jel követ. A = jel jobb oldalán található tetszőleges számú 1-es (a program indításakor pontosan 0), melyeket egy üres mező zár, amely itt is az unáris szám végét jelzi.

A program az = jelről indul, majd lép egyet balra, ezt követően addig lépeget balra, amíg nem talál egy nem-nulla értéket. Amennyiben üres mezőt talál az azt jelzi hogy a program végzett. (Ha feltételezzük, hogy a kezdeti decimális szám nem-nulla, akkor ez nem fog egyből teljesülni.) Amennyiben ez a talált szám 1 és 9 közé esik, akkor a számot 1-gyel csökkenti majd elkezd jobbra lépegetni. Addíg fog lépegetni, amíg el nem ér az = jelhez. (Fontos, hogy eközben minden 0 értéket 9-re ír át, így éri el, hogy a decimális szám értéke pontosan 1-gyel csökkenjen.) Ha elértük az = jelet, akkor elkezdünk jobbra lépegetni, amíg meg nem találjuk az első üres értékünket az unáris reprezentációunkban. (A közben talált egyeseket változtatlanul hagyjuk.) Az első üres helyet 1-re módosítjuk, így az unáris számunk értéke 1-gyel nőtt. Ezután az előzőkhöz hasonlóan vissza lépegetünk az = jelhez, és ezzel zárul egy ciklus. Ezt a ciklust n alkalommal végrehajtva a program decimálisból unárisba váltja a megadott számunkat.

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 nyelv lényege az, hogy előre megadott nyelvtani szabályokat tetszőleges sorrendben és módon alkalmazva a kapott karaktersorozat mindig egy olyan szorzat lesz amely n darab a -t, b -t és c -t tartalmaz. A fóliákon látható 2 különböző nyelv is ezt mutatja be.

Az érdekessége a feladatnak az, hogy ugyanazt a kimenetet több egymástól eltérő szabályokkal is elérhetjük. Ami több érdekes tanulságot jelez:

- Egy kimenetből nem feltétlen tudunk következtetni a bemenetre, vagy akár az azon végrehajtott műveletekre,
- Egy kívánt kimenetet gyakran több különböző módszerrel is elérhetünk.
- Ugyanazon a bemeneten, ugyanazokat a műveleteket, de más sorrendben végezve, a kimenet is módosulhat.

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](#).

Először nézzük a C utasítás definíciót:

```
<utasítás> ::=
<címkézett_utasítás> |
<kifejezés_utasítás> |
<összetett_utasítás> |
<kiválasztó_utasítás> |
<iterációs_utasítás> |
<vezérlésátadó_utasítás>
<címkézett_utasítás> ::=
    <azonosító> : <utasítás> |
    case állandó_kifejezés : <utasítás> |
    default : <utasítás>
<kifejezés_utasítás> ::=
    [<kifejezés>]
<összetett_utasítás> ::=
    „{” [<deklarációs_lista>] [<utasítás_lista>] „}”
```

```
//itt a „{}” nem iteráció, hanem konkrét karakter
<deklarációs_lista>::=
  <deklaráció>|
  <deklarációs_lista> <deklaráció>

<utasítás_lista>::=
  <utasítás>|
  <utasítás_lista> <utasítás>
<kiválasztó_utasítás>::=
  if ( <kifejezés> ) <utasítás>|
  if ( <kifejezés> ) <utasítás> else <utasítás>|
  switch ( <kifejezés> ) <utasítás>
<iterációs_utasítás>::=
  while ( <kifejezés> ) <utasítás>|
  do <utasítás> while ( <kifejezés> ) ; |
  for ( [<kifejezés>]; [<kifejezés>]; [<kifejezés>] ) <utasítás>
<vezérlésátadó_utasítás>::=
  goto <azanosító> ; |
  continue ; |
  break ; |
  return [<kifejezés>] ;
```

Most pedig jöjjön a a szabványokkal kapcsolatos példa:

A "long long int" típust, csak a c99-es szabványtól támogatják, így egy egyszerű deklaráció:

```
long long int a;
```

A c99 szabvánnyal probléma nélkül lefordul. Korábbi szabvánnyal viszont nem.

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!

Fontos tapasztalata a szabványos példának, hogy attól hogy egy program nem fordul le, még nem feltétlen magában a programban keresendő a hiba.

A c utasításos példa pedig remekül illusztrálja, hogy milyen összetett definiálásokra volt szükség ahhoz, hogy mi egy magasabb szintű nyelvben, kényelmesebben dolgozhassunk.

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: [bhex/thematic-tutorials/bhex-textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1](https://bhex.thematic-tutorials.com/bhex-textbook/IgyNeveldaProgramozod/Chomsky/realnumber.1)

```
%{
```

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

Tanulságok, tapasztalatok, magyarázat...

A program használatához először a C kódot kell generáltatnunk a "lex" utasítással az alábbi módon:

```
lex -o realnumber.c realnumber.l
```

Csakis ezt követően fordíthatunk gcc-vel. (A végére fontos a -lfl kapcsoló, különben a linker hibákat dob vissza.)

Ezt követően már futtathatjuk a programot, amely egy tetszőleges bemenetben megkeresi az egész számokat, ezeket zárójelekkel elküldi, illetve megszámlálja őket.

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)

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

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

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

    {'a', {"4", "4", "@", "/-\\\""}},
```

```

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

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

```

```

// https://simple.wikipedia.org/wiki/Leet
};

```

```

%}
%%
. {

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

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

```

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

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

found = 1;
break;
}

}

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

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

Tanulságok, tapasztalatok, magyarázat...

Hasonlóan a lexikális elemzőhöz, itt is először a "lex" utasítással C kódot generálunk, majd azt fordítjuk, figyelve a végén a -lfl kapcsolóra.

A program lényegében egy egyszerű titkosítást hajt végre. Minden karakternek megadtunk 4 alternatív verziót, amelyre a program véletlenszerűen cseréli, vagy nem cseréli le a karaktert, illetve az is random, hogy ha cseréli, akkor a 4 közül melyikre. (Amennyiben olyan karaktereket használunk a szövegben, amihez nem adtunk meg alternatívákat, akkor a program az adott karakteren nem változtat.)

A program lényegében a beírt szöveg random módosított változatát adja vissza. Ez viszont még akár szabad szemmel is viszonylag jól olvasható mivel a karakterek közel 90% százaléka változatlan marad. (Ezen persze könnyen változtathatnánk.)

A feladat elsősorban nem is a gyakorlati titkosítás szempontjából érdekes, mivel az ehhez hasonló karaktercserélgetős titkosítások feltörése ma már nem okoz problémát. (Erre sokkal jobb példa lesz majd az Exor-os titkosítás a Ceasar fejezetben.) Itt inkább a lexelésre akartunk példát mutatni.

3.6. A források olvasása

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

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

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



Bugok

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

i.

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

Ha a signal nem volt ignorálva, akkor kezelje a jelkezelő! Ha ignorálva volt, akkor továbbra is legyen ignorálva!

ii.

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

Egy for ciklus, ami 4-szer fut le.

iii.

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

Egy for ciklus, ami 5-ször fut le.

iv.

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

Egy for ciklus, ami 5-ször fut le és közben 0-tól 5-ig minden számot, ilyen sorrendben tölt be egy tömbbe.

v.

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

Egy for ciklus, ahol az i 0-tól indul és addig fut a program, amíg i kisebb egy n számnál és d pointer 1-gyel növelt értéke megegyezik az s pointer 1-gyel növelt értékével. Minden ciklus előtt az i értéke 1-gyel nő.

vi.

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

Kiír 2 integer-t, az első az f függvény return értéke a, illetve a++ paraméterekkel, a második ugyanez, csak a paraméterek fordított sorrendbe szerepelnek.

vii.

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

Kiír 2 integer-t, az első az f függvény return értéke a paraméterrel, a második maga az a .

viii.

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

Kiír 2 integer-t, az első az f függvény return értéke a pointer dereferenciált értékével paraméterül, a második maga az a .

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

"Minden x -hez létezik olyan y , hogy x kisebb y -nél és y prím."

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

"Minden x -hez létezik olyan y , hogy x kisebb y -nél és y prím és $y+2$ prím."

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

"Minden x -hez létezik olyan y , hogy x prím, akkor x kisebb y -nél."

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

"Minden x -hez létezik olyan y , hogy ha y kisebb x -nél, akkor x nem 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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a;
```

- egészre mutató mutató

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

- egész referenciája

```
int &r = a;
```

- egészek tömbje

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

- egészek tömbjének referenciája (nem az első elemé)

```
int (&tr)[5] = 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 c)) (int a, int b)
```

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

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

egész

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

egészre mutató mutató

- ```
int &r = a;
```

egész referenciája



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

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

egészek tömbjének referenciája (nem az első elemé)
- ```
int *d[5];
```

egészre mutató mutatók tömbje
- ```
int *h ();
```

egészre mutató mutatót visszaadó függvény
- ```
int *(*l) ();
```

egészre mutató mutatót visszaadó függvényre mutató mutató
- ```
int (*v (int c)) (int a, int b)
```

egészre mutató mutatót visszaadó függvényre mutató mutatót visszaadó, egészet kapó függvény
- ```
int ((*z) (int)) (int, int);
```

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

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://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr.c), [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr2.c](https://github.com/bhax/thematic_tutorials/blob/master/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)

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

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

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

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

    for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
```

```
        tm[i][j] = i * (i + 1) / 2 + j;

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

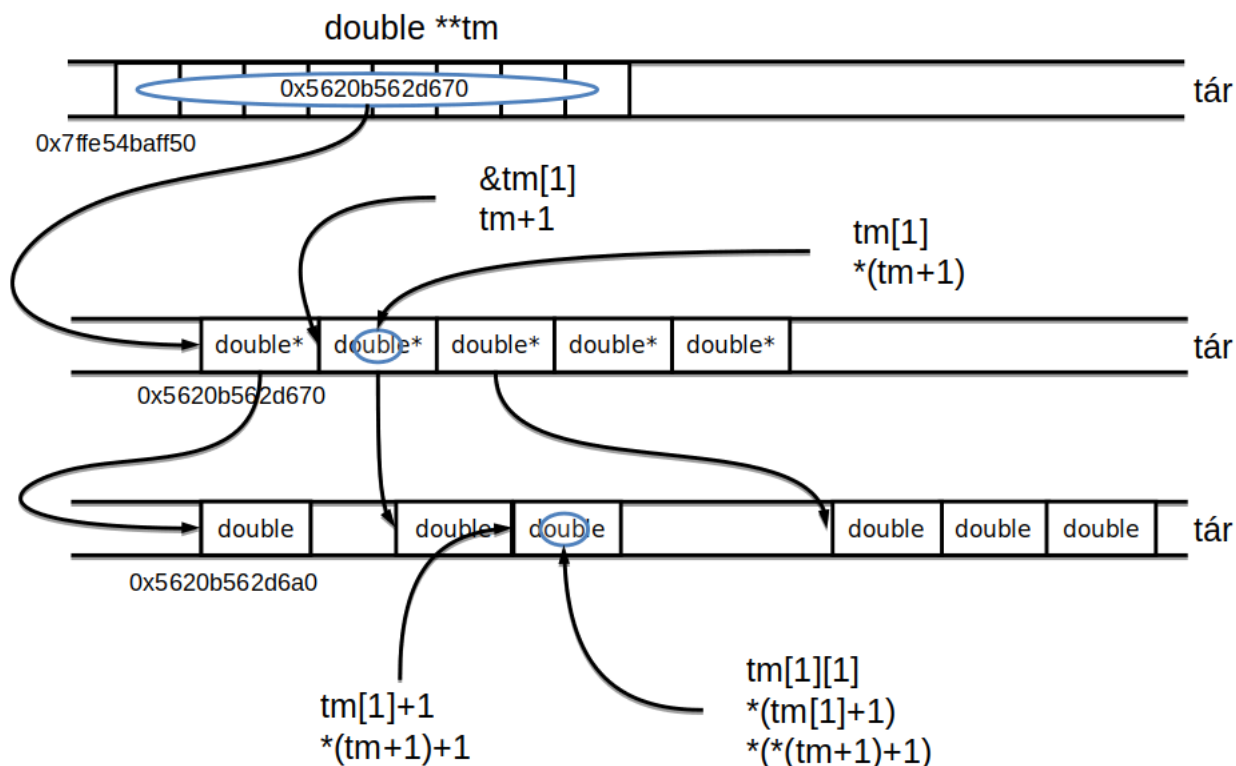
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

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

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

free (tm);

return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

A feladatnak a számunkra jelenleg érdekes része a "double **tm" kifejezés szintaktikája:

(A magyarázatot érdemes a felső ábrát figyelve olvasni) Először is, vizsgáljuk meg hogy mi is a "double **tm" alapvető szintaktikája. Ez alapvetően egy pointer, a kérdés már csak az, hogy mire mutat. A válasz pedig az, hogy "double *tm"-ekre, azaz double-re mutató pointerekre mutat. És végül, ezek a "double *tm" pointerok mutatnak majd a double értékekre. (Ezt jelenti az ábra 3. sora.)

A következő kérdés az lehet, hogy hogyan értelmezzük a "+1" részeket a kifejezésekben. Az alapvető elvünk az lesz, hogy amikor egy pointeren belül látunk "+1" kifejezést, akkor az általa mutatott érték rákövetkezőjét vesszük. (Ez a mi programunkban az adott tömb következő eleme.)

Értelmezzük akkor a "*(*(tm+1)+1)" kifejezést.

A "*(tm+1)" egy pointer, ami egy adott tömb második elemére mutat. A "*(*(tm+1)+1)" egy olyan pointer, ami egy tömb második elemére mutat, amely elem egy pointer, ami egy tömb második elemére mutat, ami egy double érték.

Az utolsó kérdés(, ami a programban kommentként szerepel is) az, hogy mi lenne, ha a "(*(tm+3))[1]" kifejezést, "*(tm+3)[1]"-re cserélnénk. Először is a "[1]"-et "...+1)"-re cserélhetjük:

Az első kifejezésünk így: `*(*(tm+3)+1)` Ez pedig egy olyan pointer, ami egy tömb második elemére mutat, ez a második elem, egy olyan pointer lesz, ami egy tömb negyedik elemére mutat, ami pedig egy double érték lesz.

A második kifejezésünk pedig: `*(tm+3)+1` Ez egy olyan pointer, ami egy pointerre mutat, amely pointer egy tömb ötödik elemére mutat, ami egy double érték lesz.

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.

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

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

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

A programot így fogjuk tudni futtatni:

```
./e kulcs <tiszta.szoveg >titkos.szoveg
```

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 program láthatóan végigmegy a bemeneti fájlon és bitenként végez exor műveletet. Ennek eredménye pedig egy teljesen olvashatatlan karakterlánc.

Vegyük észre, hogy az exor művelet tulajdonságaiból fakadóan, ha ugyanazon fájlban ugyanazzal a kulccsal 2-szer végezzük ugyanazt a műveletet, akkor visszakapjuk az eredeti szöveget. Tehát a kulcs birtokában ugyanezen program alkalmas a dekódolásra is, az alábbi módon:

```
./e kulcs <titkos.szoveg
```

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

```
public class ExorTitkosító {

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

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

    public static void main(String[] args) {
        try {
            new ExorTitkosító(args[0], System.in, System.out);
        } catch(java.io.IOException e) {

            e.printStackTrace();
        }
    }
}
```


Tanulságok, tapasztalatok, magyarázat...

Ez a program gyakorlatilag teljesen megegyezik az előbbi c programmal, persze eltekintve a szintaktikától.

A programot így fordítjuk:

```
javac ej.java
```

Majd így futtatjuk:

```
java ej kulcs > titkosított.szöveg
```

A változatosság kedvéért, itt nem fájlból olvasunk majd, hanem a terminál bemenetéből. Tehát az előző parancs után a programunk még vár egy bemenetre. (Ezt szimplán csak gépeljük be, vagy másoljuk át a terminálba.)

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

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

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

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

    return (double) titkos_meret / sz;
}

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

```
double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

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

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

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

    return tiszta_lehet (titkos, titkos_meret);
}

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

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

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

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

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

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

return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Ez a program alapvetően azt csinálja, hogy rengeteg egymásba ágyazott for ciklus segítségével, szisztematikusan végigpróbál minden számkonbinációt. (Természetesen a feltételek módosíthatóak, például számok helyett betűket, vagy számok és betűk, de akár speciális karakterek halmazának minden elemével is dolgozhatunk.)

A program továbbá tartalmaz szűrőket is. Ez azért kell, mert a rengeteg próbálkozásból, mi csakis az eredeti értelmes szövegre vagyunk kíváncsiak, így megadtunk pár feltételt. Az egyik ilyen feltétel, hogy az átlagos szóhossz, bizonyos hibahatáron belül kell, hogy legyen az átlagos magyar szóhosszhoz képest. A második feltételünk, hogy a szöveg tartalmazza a magyar nyelv leggyakoribb kifejezéseit. (hogy, nem, az, ha)

A program emellett remekül mutatja, hogy miért érdemes a #define parancsot használni a konstans értékek-nél, mert például, ha a kulcsról tudjuk, hogy 10 karakteres, akkor a programot e szerint kell átírnunk. Ebben

a konkrét esetben pedig, ha nem használtuk volna a `#define` parancsot, akkor az adott érték összes előfordulását manuálisan kellett volna módosítani, a `#define` által pedig elég csak 1 módosítás. (Ez a nagyobb programoknál jelentős mennyiségű időt takarít meg, illetve rengeteg olyan hibát előzhetünk meg vele, amit egy félregépelt, vagy elmulasztott módosítás okozhatott volna.)

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

Tanulságok, tapasztalatok, magyarázat...

A programunk itt egy R-ben ábrázolt neurális háló. De először is, mi az a neurális háló? A neurális háló egy az egyhez hasonló felépítésű rendszer, amely neuronokból, illetve köztük lévő kapcsolatokból áll.

A konkrét program most arról szól, hogy megadtuk egy-egy logikai művelet (és, vagy, exor) eredményeit, anélkül, hogy a gépnek megadtuk volna a műveletet. A gép tehát pusztán az a kapott adatokból próbálja kitalálni, hogy adott bemenetekből milyen matematikai műveletekkel kaphatja meg a kívánt kimenetet. (Ez a művelet egyébként általában egy súlyozott összegzés.)

Az "és", illetve "vagy" műveleteknél ez működött is, azonban a harmadik programunk nem működött. Ez azért történt, mivel az exor műveletet már nem lehet direktbe egy súlyozott összeadással megoldani, itt már szükség van további "neuronokra" a hálón. Ha a negyedik programot futtatjuk, akkor az ugyanerre az exor műveletre már helyes eredményt ad.

Érdekesség: Az exor műveletnél feljött probléma egyébként, olyan zavarónak tűnt, hogy ezért sokáig nem vették komolyan a neurális hálókat, mint programozási eszközöket, mivel etéren az exor művelet elég-gé alapvető. Az utókor viszont rácsáfolt erre a hozzáállásra, ugyanis a mai kor legfontosabb informatikai áttöréseit (arcfelismerés, szimulációk, kép- és hangmanipuláció, ...) neurális hálózatokkal érték el.

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...

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++](#) nevű állománya.

A Mandelbrot halmaz a komplex síkon

5.1. ábra. A Mandelbrot halmaz a komplex síkon

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

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

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

Azaz kiindulunk az origóból (z_0) és elugrunk a rács első pontjába a $z_1 = c$ -be, aztán a c -től függően a további z -kbe. Ha ez az utazás kivezet a 2 sugarú körből, akkor azt mondjuk, hogy az a vizsgált rácpont nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végtelen sok z -t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácponthoz. Ha közben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácpont 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 [bhaxor/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
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --line-numbers=1 --left-footer=" ↵
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

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

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

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

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

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

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

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

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

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

```
{
    // k megy az oszlopokon

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

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

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

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

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

            ++iteracio;
        }

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

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

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

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbGRzY76E>

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

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

A különbség a **Mandelbrot halmaz** és a Julia halmazok között az, hogy a komplex iterációban az előbbiben

a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

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

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

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

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

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

            ++iteracio;
        }
    }
}
```

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

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

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

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// 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://bhax.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.

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJssY>, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a

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

DRAFT

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

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

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 , 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.