

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert	2019. április 9.	

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	Aktualizálás, javítások.	nbatfai

Ajánlás

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

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

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	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	7
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun tétel	13
2.8. A Monty Hall probléma	15
3. Helló, Chomsky!	17
3.1. Decimálisból unárisba átváltó Turing gép	17
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	18
3.3. Hivatkozási nyelv	18
3.4. Saját lexikális elemző	20
3.5. l33t.1	21
3.6. A források olvasása	22
3.7. Logikus	26
3.8. Deklaráció	32

4. Helló, Caesar!	34
4.1. double ** háromszögmátrix	34
4.2. C EXOR titkosító	35
4.3. Java EXOR titkosító	36
4.4. C EXOR törő	37
4.5. Neurális OR, AND és EXOR kapu	39
4.6. Hiba-visszaterjesztéses perceptron	41
5. Helló, Mandelbrot!	43
5.1. A Mandelbrot halmaz	43
5.2. A Mandelbrot halmaz a std::complex osztállyal	46
5.3. Biomorfok	49
5.4. A Mandelbrot halmaz CUDA megvalósítása	52
5.5. Mandelbrot nagyító és utazó C++ nyelven	53
5.6. Mandelbrot nagyító és utazó Java nyelven	55
6. Helló, Welch!	58
6.1. Első osztályom	58
6.2. LZW	60
6.3. Fabejárás	63
6.4. Tag a gyökér	65
6.5. Mutató a gyökér	67
6.6. Mozgató szemantika	68
7. Helló, Conway!	69
7.1. Hangyaszimulációk	69
7.2. Java életjáték	70
7.3. Qt C++ életjáték	70
7.4. BrainB Benchmark	73
8. Helló, Schwarzenegger!	75
8.1. Szoftmax Py MNIST	75
8.2. Szoftmax R MNIST	75
8.3. Mély MNIST	75
8.4. Deep dream	75
8.5. Robotpszichológia	76

9. Helló, Chaitin!	77
9.1. Iteratív és rekurzív faktoriális Lisp-ben	77
9.2. Weizenbaum Eliza programja	77
9.3. Gimp Scheme Script-fu: króm effekt	77
9.4. Gimp Scheme Script-fu: név mandala	77
9.5. Lambda	78
9.6. Omega	78
10. Helló, Olvasónapló!	79
10.1. Juhász István féle könyv	79
10.2. K & R, A C programozási nyelv	79
10.3.	80
III. Második felvonás	81
11. Helló, Arroway!	83
11.1. A BPP algoritmus Java megvalósítása	83
11.2. Java osztályok a Pi-ben	83
IV. Irodalomjegyzék	84
11.3. Általános	85
11.4. C	85
11.5. C++	85
11.6. Lisp	85

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?

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?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/turing/inf_sleep.c](#)
[bhax/thematic_tutorials/bhax_textbook/turing/infinite_single.c](#)
[bhax/thematic_tutorials/bhax_textbook/turing/infinite_multi.c](#)

A végtelen ciklusok nagyon hasznosak tudnak lenni például ha egy menüt akarunk írni ahol semmi mást nem csinálunk, csak várunk valamilyen bemenetre és az alapján végrehajtunk valamit, majd újra várakozunk. C-ben 2 féle módon írhatunk végtelen ciklust:

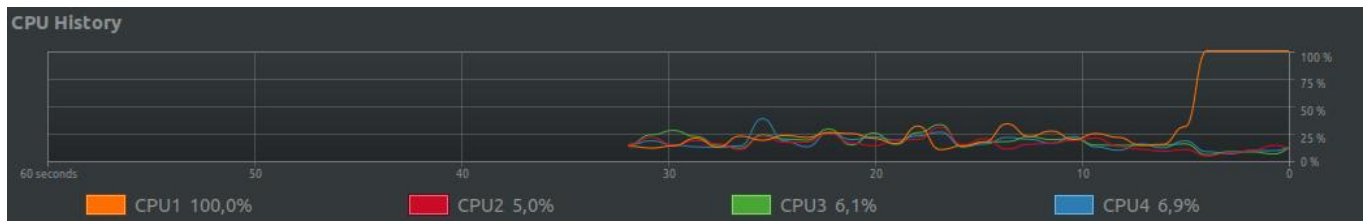
- for ciklussal

```
for(;;) {  
    //infinite loop  
}
```

- while ciklussal

```
while(1) {  
    //infinite loop  
}
```

Először nézzük meg a processzort 100%-on futtató programot. A könnyebb olvashatóság érdekében a while ciklust használtam és az "1"-es értékkel (amit a fordító logikai igaz értéként értelmez ebben az esetben) a ciklus soha nem ér véget. Mivel a ciklusba nem írtunk semmit, a program a while ciklus feltételének vizsgálásával tölti minden idejét, ami a processzor 100% leterheltségét eredményezi.



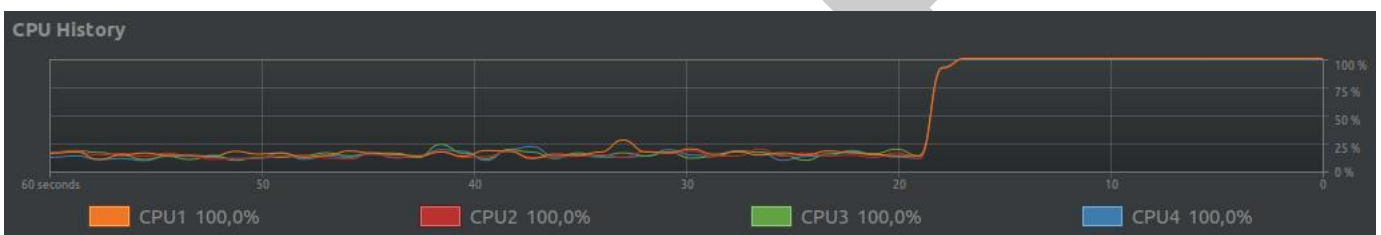
Ahhoz hogy az összes magot 100%-on pörgessük, csak 1 sorral kell kibővítenünk a programot, de ehhez szükségünk lesz az `omp.h` könyvtár implementálására és fordításkor a `"-fopenmp"` kapcsolóra.

```
#include <omp.h>

int main() {

    #pragma omp parallel
    while(1) {}

    return 0;
}
```



Ha 0%-os terhelést akarunk, akkor csökkenteni kell a feltételvizsgálat gyakoriságát. Vagyis a `sleep(seconds)` paranccsal várakozásra utasítjuk a processzort.

```
#include <unistd.h>

int main() {

    while(1) sleep(1);

    return 0;
}
```

1

[|||||]

10.0%

Tasks: 169, 690 thr; 1 running

2

[|||||]

14.1%

Load average: 0.49 0.68 0.72

3

[|||||]

10.6%

Uptime: 07:15:23

4

[|||||]

9.9%

Mem

[|||||]

3.77G/7.70G

Swp

[|||||]

0K/3.81G

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1070	gdm	20	0	111M	2724	12	S	0.0	0.0	0:00.00	(sd-pam)
1408	my	20	0	111M	2728	12	S	0.0	0.0	0:00.00	(sd-pam)
25849	my	20	0	4376	740	680	S	0.0	0.0	0:00.00	./inf sleep
11075	my	20	0	31388	5068	3568	S	0.0	0.1	0:00.02	/bin/bash
1069	gdm	20	0	77136	8316	6780	S	0.0	0.1	0:00.08	/lib/systemd/systemd --user

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

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

Megoldás videó:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

Nézzünk erre egy másik példát: Tegyük fel, hogy Lajos bárkiről megtudja mondani, hogy csődbe fog-e menni vagy sem (Lefagy fv.). A fia, Szabolcs ugyanazt csinálja, de ő már egy kicsit másképp csinálja (Lefagy2 fv.). Egyszer, amikor Szabolcs a saját pénzügyeit vizsgálta, válaszát apja válaszára építette. Ekkor 2 lehetőség merül fel. Egy, hogy ha Lajos azt mondja, hogy Szabolcs csődbe megy, akkor annak ellenére, hogy igazat ad apjának, nem megy csődbe, ha nemmel válszol, akkor viszont csődbe megy. Amint láthatjuk, ellentmondásba ütköztünk, vagyis ez a módszer így nem fog működni.

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: bhax/thematic_tutorials/bhax_textbook/turing/valtcserere.c

Változók értékének felcserélésére leggyakrabban a rendezési algoritmusoknál van szükség. Ilyenkor a legegyszerűbb módszer jut az eszembe, miszerint veszek egy harmadik, úgymond segédváltozót, amiben ideiglenesen eltárolom az egyik változó értékét, hogy azt majd a másik változó kaphassa meg.

```
int c = a;
a = b;
b = c;
```

Ha segédváltozó nélkül szeretnénk megoldani a problémát, akkor kénytelenek vagyunk valamilyen műveleteket alkalmazni. Nem kell sokáig keresgélni, mivel az összeadás és kivonás tökéletesen megfelel erre a feladatra. Ehhez először összeadjuk őket az *a* változóba, majd abból kivonva a *b*-t megkapjuk először a *b*, majd az *a* értékét, ahogy azt a programkód szemlélteti:

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

2.4. Labdapattogás

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/turing/labdaIf.c](#) [bhax/thematic_tutorials/bhax_textbook/turing/labdaIfNelkul.c](#)

Ez a feladat sokkal egyszerűbb, mint az ember gondolná, legalábbis ha használhatunk *if*-eket. Csak annyit kell csinálni, hogy ciklusonként növeljük a labda pozícióját és ha elérjük az ablak kereteit, megfordítjuk a labda irányát. Ezután már csak meg kell jeleníteni a labdát.

```
#include <stdio.h>  
  
#define width 80  
#define height 24  
  
void draw(int x, int y){  
  
    for(int i = 0; i < y; i++) printf("\n");  
    for(int k = 0; k < x; k++) printf(" ");  
    printf("@\n");  
  
}  
  
int main(){  
  
    int x = 1;  
    int y = 1;  
    int a = 1;  
    int b = 1;  
  
    while(1){  
  
        system("clear");  
  
        if(x >= width || x <= 0) a *= -1;  
        if(y >= height || y <= 0) b *= -1;
```

```
    draw(x, y);
    x += a;
    y += b;

    usleep(15000);

}

return 0;
}
```

If-ek nélkül azonban már kicsit nehezebb a dolgunk, ugyanis egy kicsit gondolkodni kell. De ekkor se kell kétségbe esni, mert ugyebár a matematika a "barátunk" és mindig nyújt számunkra egy bonyolultnak látszó képletet, hogy azt egyszer bepötyögve örökre megszabaduljunk tőle. Most mi is ezt tesszük és még egy magyarázatot is csatolok hozzá.

```
draw(abs(width-(x++%(width*2))), abs(height-(y++%(height*2))));
```

Tegyük fel hogy az `x` értéke 1 és a `width` értéke pedig 5. Ha az 1-et elosztjuk maradékos osztással a `width` kétszeresével, akkor 1-et kapunk, amit kivonunk a szélességből. Ennek eredményeként a labda pattogása az ablak jobb alsó sarkából indul.

```
x = 1; width = 5;
|5 - ( 1 % ( 5 * 2 ) )| =
|5 - ( 1 % 10 )| =
|5 - 1| = 4
```

Ha az `x` eléri a 6-ot, az abszolútérték miatt a végeredmény növekedni kezd, egészen amíg el nem éri a `width` kétszeresét, amikor is újra csökkenni kezd és ez így megy körbe-körbe. A `++` az `x` után a számolások után megnöveli 1-el az `x`-et.

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:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/turing/bogomips.c

Mi is ez a BogomIPS valójában? Megméri, hogy 1 másodperc alatt hányszor fut le a programunk egy ciklusa. Sokan abba a hibába esnek, hogy ez alapján hasonlítják össze a gépüket, hogy kié gyorsabb. Egyedül a linux bootolásakor érdemes odafigyelni rá, hogy a processzor és a cache rendben van-e.

A `ticks` változóba elmentjük a jelenlegi processzoridőt, majd a `for` ciklus egy bizonyos számú lefutása után a jelenlegi processzoridőből kivonjuk a `ticks` értékét, így megkapjuk hogy mennyi ideig tartott a ciklus `x`-szeri lefutása.

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

Ha ez az idő kisebb mint 1 másodperc, akkor legközelebb többször futtatjuk le a for ciklust, ellenben viszont némi átalakításokkal kiírjuk a gépünk Bogomips értékét.

```
Calibrating delay loop..ok - 690.00 BogomIPS
```

A Bogomips while ciklus fejét használva mostmár könnyen megkapjuk szóhosszt. Konkrétan itt az történik, hogy az `int` típusú változó elején levő bitet ciklusonként eltoljuk és hozzáadunk egyet a szóhossz tároló `length` változóhoz.

```
int x = 1;
int length = 0;

do
    length++;
while((x <= 1));
```

Az én gépemen a szóhossz mérete 32.

```
$ ./szohossz
A szóhossz mérete: 32
```

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: [bhax/thematic-tutorials/bhax-textbook/turing/helloGoogle.c](https://bhamilth.github.io/thematic-tutorials/bhax-textbook/turing/helloGoogle.c)

Most megvizsgáljuk a PageRank algoritmust, ami hatalmas fellendülést hozott a Google-nek.

Egy oldal PageRankját az határozza meg, hogy mennyi oldal hivatkozik rá. Mivel ez így magába nem lesz elég, mert az algoritmus azt is figyelembe veszi, hogy mekkora a PageRankjuk a rá hivatkozó oldalaknak. Például ha a Facebook hivatkozik a te oldaladra, akkor az sokkal többet jelent, mintha csak valami ismeretlen oldal hivatkozna rá, még akkor is ha több van belőlük.

A program elején létrehozunk egy kétdimenziós tömböt, ami meghatározza, hogy melyik mire hivatkozik. Ezután egy végtelen ciklusba bizonyos számításokkal addig finomítgatjuk az oldalak PR értékét, amíg már nincs értelme tovább számolgatni.

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

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

double tavolsag(double pagerank[], double pagerank_temp[], int db)
{
    double tav = 0.0;
    int i;
    for (i=0; i<db; i++)
        tav += (pagerank_temp[i] - pagerank[i]) * (pagerank_temp[i] - pagerank[i] ←
        ]);
    return sqrt(tav);
}

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

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

    long int i, j, h;
    i=0; j=0; h=5;

    for (;;)
    {
        for (i=0; i<4; i++)
            PR[i] = PRv[i];
        for (i=0; i<4; i++)
        {
            double temp=0;
            for (j=0; j<4; j++)
                temp+=L[i][j]*PR[j];
            PRv[i]=temp;
        }

        if ( tavolsag(PR, PRv, 4) < 0.00001)
            break;
    }
    kiir (PR, 4);
}
```

```
return 0;  
  
}
```

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

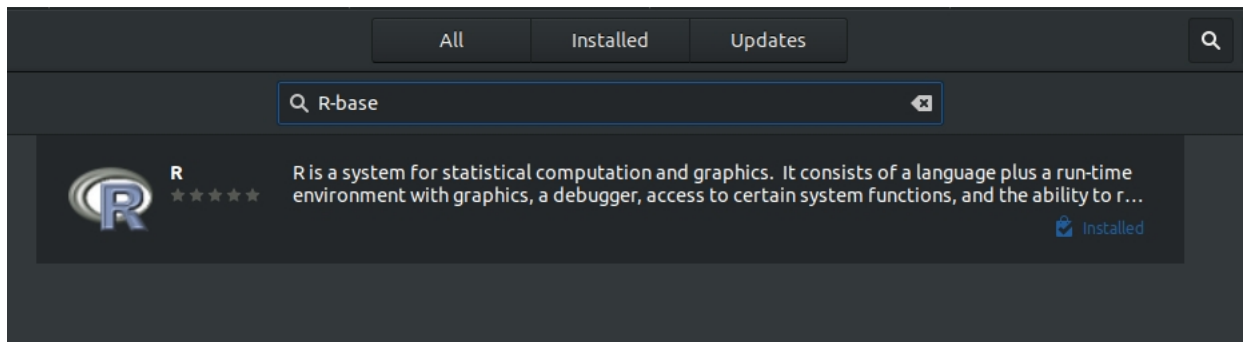
Megoldás videó: <https://youtu.be/xbYhp9G6VqQ>

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/turing/brun.r](https://bhax.thematic_tutorials.com/bhax_textbook/turing/brun.r)

```
# Copyright (C) 2019 Dr. Norbert Bاتفai, 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  
  rtlplust2 = 1/t1primes+1/t2primes  
  return(sum(rtlplust2))  
}  
  
x=seq(13, 1000000, by=10000)  
y=sapply(x, FUN = stp)  
plot(x,y,type="b")
```

A Brun tétel szerint ha az ikerprímek reciprokát összeadjuk, akkor ez az összeg egy konkrét szám, ún. Brun-konstans. Hogy az ikerprímek száma véges vagy végtelen azt senki se tudja.

Az R nyelv használatához szükségünk lesz az R-base szoftverkörnyezet telepítésére, amit az Ubuntu Software programon keresztül tehetünk meg a legegyszerűbben.



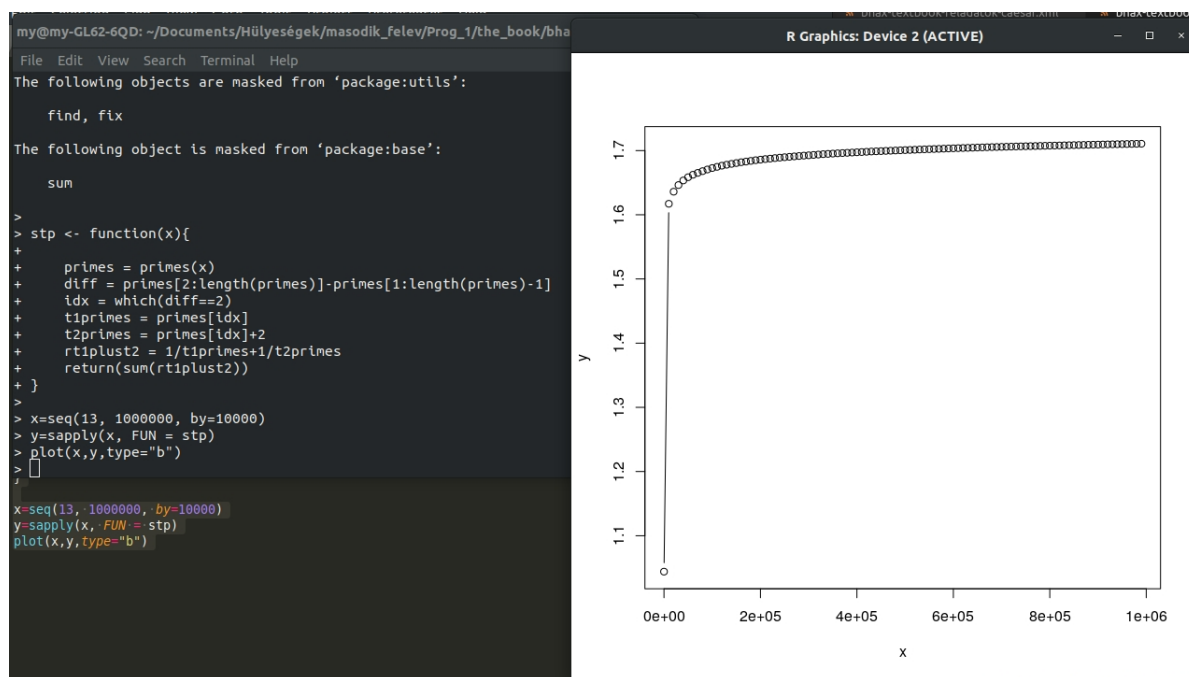
Ezután a terminálba megnyitjuk az R környezetét (az **R** paranccsal) és telepítjük a `matlab` csomagot, hogy kezelni tudjuk a prímeket. (Majd a `library(matlab)` paranccsal betöltjük a csomagot)

```
my@my-GL62-6QD: ~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/t...  
File Edit View Search Terminal Help  
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/themati  
tutorials/bhax_textbook$ R  
  
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"  
Copyright (C) 2018 The R Foundation for Statistical Computing  
Platform: x86_64-pc-linux-gnu (64-bit)  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> install.packages("matlab")
```

A `primes(meggig)` függvénnyel lekérhetjük a prím számokat és a `primes[hanyadiktól : hanyadikig]` módon lekérhetjük annak tetszőleges intervallumát (Arra vigyázzunk, hogy amíg a `primes` először függvényként, másodszor viszont már azonosítóként szerepel). Az egymás mellett levő prímek egymásból való kivonásával megkapjuk azok különbségét. Ahol a különbség 2, ott ikerprímekről beszélünk és a `which()` függvény visszaadja, hogy hanyadik helyen találjuk őket.

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

Ezután 2 vektorba párhuzamosan eltároljuk ezeket az ikerprímeket és összeadjuk azok reciprokát. A `plot()` paranccsal megjelenítjük az összeget és láthatjuk, hogy ez a függvény egy konkrét számhoz divergál.



2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan](https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall_paradoxon_kapcsan)

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/turing/monty.r](https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan)

A probléma szemléltetéséként tegyük fel, hogy az előtted levő 3 láda közül az egyikbe kincs van a másik kettő üres. A játék elején választasz egyet, de mielőtt kinyithatnád, a játékvezető kinyitja az egyik üres ládát (természetesen sose nem azt, amelyiket kiválasztottad). Ezután újra meggondolhatod, hogy a maradék 2 láda közül melyiket akarsz kinyitni. Bármennyire is hihetetlennek tűnik, de nagyobb a valószínűsége, hogy a másikat választva megkapod a kincset, ellenben azzal, ha kitartasz az első döntésed mellett.

```

# 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)
```

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: [bhax/thematic-tutorials/bhax-textbook/chomsky/toUnary.cpp](https://bhax.thematic-tutorials.com/bhax-textbook/chomsky/toUnary.cpp)

Míg a 10-es számrendszerben 10 számjegyük van, az unárisban csak egy, az 1. Éppen ezért az átalakítás nagyon egyszerű lesz, hiszen csak annyi 1-et kell kiíratni, amennyi a megadott szám.

```
#include <iostream>
#include <limits>

void DecimalToUnary(int x)
{
    for(int i=0; i<x; i++)
    {
        std::cout << "1";
    }
    std::cout << "\n";
}

int main()
{
    int szam;

    std::cout << "Adj meg egy szamot:\n";

    std::cin >> szam;

    DecimalToUnary(szam);

    return 0;
```

```
}
```

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:

Környezetfüggő(hossz nem csökkenítő) $P_1 X P_2 \rightarrow P_1 Q P_2$, P_1, P_2 eleme $(V_N \cup V_T)^*$, $X \in V_N$, $Q \in (V_N \cup V_T)^+$, kivéve $S \rightarrow \epsilon$, de akkor S nem lehet jobb oldali egyetlen szabályban sem.

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: [bhax/thematic-tutorials/bhax_textbook/chomsky/hivatk.c](https://github.com/bhax/thematic-tutorials/blob/master/bhax_textbook/chomsky/hivatk.c)

<https://cs.wmich.edu/~gupta/teaching/cs4850/summer06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm>

Először nézzük meg mi is az a BNF. A BNF (Backus-Naur Form) egy olyan nyelv, ami más nyelvek helyesírását írja le. Nézzünk meg ráegy egyszerű példát:

```
<egész szám> ::= <előjel><szám>
<előjel> ::= [-|+]
<szám> ::= <számjegy>{<számjegy>}
<számjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

Ez alapján egy c utasítás BNF-ben így fog kinézni:

```
<statement> ::= <labeled_statement>
               | <compound_statement>
               | <expression_statement>
               | <selection_statement>
               | <iteration_statement>
               | <jump_statement>

<labeled_statement> ::= <identifier> : <statement>
                     | case <constant_expression> : <statement>
                     | default : <statement>

<compound_statement> ::= { <declaration_list> }
                      | { <statement_list> }
                      | { <declaration_list> <statement_list> }
```

```

<declaration_list> ::= <declaration>
                      | <declaration_list> <declaration>

<statement_list> ::= <statement>
                    | <statement_list> <statement>

<expression_statement> ::= <expression>

<selection_statement> ::= if ( <expression> ) <statement>
                          | if ( <expression> ) <statement> else <statement>
                          | switch ( <expression> ) <statement>

<iteration_statement> ::= while ( <expression> ) <statement>
                        | do <statement> while ( <expression> ) ;
                        | for ( <expression_statement> < ←
                            expression_statement> ) <statement>
                        | for ( <expression_statement> < ←
                            expression_statement> <expression> ) <statement>

<jump_statement> ::= goto <identifier> ;
                   | continue ;
                   | break ;
                   | return <expression> ;

```

A következő példava szemlélteti a c nyelv fejlődését:

```

int main(){

    // can't compile me with: -std=c89

    for(int i = 0; i < 10; i++)
        printf("Succes!\n");

    return 0;
}

```

Ha a "-std=c89" kapcsolóval próbáljuk meg lefordíttatni ezt a programot, akkor hibába ütközünk.

```

hivatk.c: In function 'main':
hivatk.c:5:2: error: C++ style comments are not allowed in ISO C90
    // can't compile me with: -std=c89
    ^
hivatk.c:5:2: error: (this will be reported only once per input file)
hivatk.c:7:2: error: 'for' loop initial declarations are only allowed in ←
    C99 or C11 mode
    for(int i = 0; i < 10; i++)
    ^~~
hivatk.c:7:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 ←
    to compile your code

```

Tehát a c89-es szabvánnyal még nem lehetett dupla `"/`-jeles kommentet írni és a `for` ciklus ciklusváltozóját a ciklus előtt kellett deklarálni.

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:

Megoldás forrása: [bhax/thematic-tutorials/bhax-textbook/chomsky/realnum.1](https://bhax.thematic-tutorials.com/bhax-textbook/chomsky/realnum.1)

Ez a lexer úgy működik, hogy az általunk megadott minták alapján átnézi az általunk megadott szöveget. Először megírjuk a mintákat ami alapján a lexer legenerálja a `c` programunkat és az fogja elvégezni a szövegelemzést.

A `"%{"` és `"%}"` jelek közé berakjuk a generálandó program elejére szánt sorokat, mint pl.: könyvtár importálás vagy deklaráció:

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

Ezután jönnek a definíciók. Jelen esetben egyedül a számjegy fogalmát kell definiálnunk, mivel ez már elegendő egy valós szám leírására.

```
digit [0-9]
```

Az első `"%%"` jel után megadjuk, hogy mit keresünk és hogy mit csináljunk, ha megtaláltuk a keresett szövegrészt.

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

Az utolsó rész - a második `"%%"` jel után - pedig a programunk, ahol elindítjuk a lexikális elemzést a `yylex()` függvény hívásával.

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

A `flex` csomag telepítése után már hozzá is láthatunk a fordításhoz. Először a lexerral legeneráltatjuk a `c` fájlt, amit majd lefordítunk és futtatunk.

```
lex -o realnum.c realnum.l
```

```
gcc realnum.c -o realnum -lfl
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: [bhex/thematic-tutorials/bhex-textbook/chomsky/i33t.1](https://bhex.thematic-tutorials.com/bhex-textbook/chomsky/i33t.1)

A l337 (Leet) egy - az interneten használt - ábécé, ami a betűket számokkal vagy szimbólumokkal helyettesíti. Először hacker-ek és cracker-ek használták, de később sokaknak megtetszett és a felhasználóneveiket ilyen módon módosítva kezdték el használni.

A változatosság kedvéért minden betűhöz hozzárendelünk 4 alternatívát és egy random generált szám fogja megmondani, hogy melyiket válasszuk. Ezt úgy oldjuk meg, hogy a `struct` segítségével létrehozunk egy típust, ami egy karaktert és egy 4 elemű tömböt tartalmaz, majd ebből egy kétdimenziós tömböt készítünk.

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

    {'a', {"4", "4", "@", "/-\\"}},
    {'b', {"|o", "8", "|3", "|"}},
    {'c', {"c", "(", "<", "{"}},
    ...
    {'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);

}
```

Amint azt láthatjuk, az "a" betűt felcserélte egy "4"-esre és az "e"-t "3"-asra (ezek a leggyakrabban használt átalakítások).

```
A feladat teljesítve.
4 f3l/-\d4t t3lj3sítv3.
A feladat teljesítve.
4 f3l4d4t t3|_j3s1t\3.
```

A fordításhoz és futtatáshoz ugyanúgy kell eljárni, mint az előző feladatban.

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 megyránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

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

Itt viszont akkor kezeli a jelkezelő a SIGINT jelet, ha a SIGINT jel nem lett figyelmen kívül hagyva.

```

/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/forrasolv.c: ↵
    (in function main)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/forrasolv.c:10:8:
      Test expression for while not boolean, type int: 1
      Test expression type is not boolean or int. (Use -predboolint to ↵
        inhibit
        warning)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/forrasolv.c:14:4:
      Return value (type [function (int) returns void]) ignored:
      signal(SIGINT, j...
      Result returned by function call is not used. If this is intended, ↵
        can cast
        result to (void) to eliminate message. (Use -retvalother to inhibit ↵
        warning)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/forrasolv.c:4:6:
      Function exported but not used outside forrasolv: jelkezeslo
      A declaration is exported, but not used outside this module. ↵
        Declaration can
        use static qualifier. (Use -exportlocal to inhibit warning)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bha
    x/thematic_tutorials/bhax_textbook/chomsky/forrasolv.c:6:1:
      Definition of jelkezeslo

Finished checking --- 3 code warnings

```

ii.

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

Ez egy `for` ciklus, ami 5-ször fog lefutni és minden egyes alkalommal 1-el növeli az `i` értékét `preorder` alakban.

iii.

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

Ez egy ugyanúgy 5-ször lefutó `for` ciklus, ami mostmár `postorder` alakban növeli a ciklusváltozót, de ettől függetlenül a ciklus ugyanannyiszor fog lefutni.

iv.

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

Ez nem úgy fog működni, ahogy azt szeretnénk. A **splint** paranccsal megvizsgálva a programkódot rájövünk hogy miért. A program futása során nem egyértelmű, hogy mikor lesz növelve az `i` értéke, így például az `i` értéke a tömb nem várt pozíciójára kerülhet.


```

/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/for52.c: (in ↵
function main)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
the_book/bhax/t
hematic_tutorials/bhax_textbook/chomsky/for52.c:6:31:
Expression has undefined behavior (left operand uses i, modified ↵
by right
operand): tomb[i] = i++
Code has unspecified behavior. Order of evaluation of function ↵
parameters or
subexpressions is not defined, so if a value is used and modified in
different places not separated by a sequence point constraining ↵
evaluation
order, then the result of the expression is unspecified. (Use - ↵
evalorder to
inhibit warning)

Finished checking --- 1 code warning

```

v.

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

Ez a ciklus minimum n -szer le fog futni. Hogy maximum hányszor az attól függ, hogy mi lesz a $*d++$ visszatérési értéke.

Abban az esetben, ha a d és az s int típusú pointerek, akkor a **splint** hibát jelez, mivel az s jelek jobb oldalán nem logikai értéket kapunk.

```

/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/for53.c: (in ↵
function main)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
the_book/bhax/t
hematic_tutorials/bhax_textbook/chomsky/for53.c:9:18:
Right operand of && is non-boolean (int): i < n && (*d++ = *s++)
The operand of a boolean operator is not a boolean. Use +ptrnegate ↵
to allow !
to be used on pointers. (Use -boolops to inhibit warning)

Finished checking --- 1 code warning

```

vi.

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

Ez a kód ugyanazt a hibát követi el, mint a tömbös, ezen felül amikor az egyik f függvénynek átadjuk az a -t és növeljük, akkor az módosítja a másik f függvénynek átadott értéket is.

```
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/print1.c: ( ↵
    in function main)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/print1.c:11:23:
      Argument 2 modifies a, used by argument 1 (order of evaluation of ↵
        actual
        parameters is undefined): f(a, ++a)
Code has unspecified behavior. Order of evaluation of function ↵
  parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining ↵
    evaluation
order, then the result of the expression is unspecified. (Use - ↵
  evalorder to
  inhibit warning)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/print1.c:11:31:
      Argument 1 modifies a, used by argument 2 (order of evaluation of ↵
        actual
        parameters is undefined): f(++a, a)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/print1.c:11:18:
      Argument 2 modifies a, used by argument 3 (order of evaluation of ↵
        actual
        parameters is undefined): printf("%d %d", f(a, ++a), f(++a, a))
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/print1.c:11:29:
      Argument 3 modifies a, used by argument 2 (order of evaluation of ↵
        actual
        parameters is undefined): printf("%d %d", f(a, ++a), f(++a, a))
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bhax/t
    hematic_tutorials/bhax_textbook/chomsky/print1.c:3:5:
      Function exported but not used outside print1: f
A declaration is exported, but not used outside this module. ↵
  Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ↵
  the_book/bha
x/thematic_tutorials/bhax_textbook/chomsky/print1.c:5:1: Definition ↵
  of f
```

Finished checking --- 5 code warnings

vii.

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

Tegyük fel, hogy az f függvény visszatérési értéke az argumentum négyzete. Ekkor először kiírja az a négyzetét, majd magát az f -t.

viii.

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

Ha az f függvény 1-el nagyobb értéket ad vissza, mint az az érték amire az argumentum mutat, akkor először kiírjuk az $a+1$ -et, majd az a értékét.

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/chomsky/forrasolv.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/chomsky/forrasolv.c)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for51.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/chomsky/for51.c)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for52.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/chomsky/for52.c)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for53.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/chomsky/for53.c)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for51.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/chomsky/for51.c)

Megoldás videó:

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

A matematikai logikai (matlog) nyelv a beszélt nyelvnél pontosabb kifejező eszköz. A beszélt nyelv szavakból és mondatokból épül, még a matlog nyelv termékből és formulákból. A termék változókból és függvényekből épülnek. Term például az az x változó, ami befutotta az embereket. A formulák termékből és prédikátumokból épülnek. Formula például a Szeret(x , y) két változós prédikátum. Illetve a logikai jelekkel a formulákból további formulák építhetők fel.

```
% http://detexify.kirelabs.org/classify.html
```

```
\documentclass{article}
```

```
% magyar betűk ebben a forrásban. hogy ne kelljen \'a, hanem elég legyen az ←  
á
```

```
\usepackage[utf8]{inputenc}
% legyen pl. magyar elválasztása a sorvégén a szavaknak:
\usepackage[magyar]{babel}

\usepackage{amsmath}
\usepackage{amssymb}
\usepackage{url}
\usepackage{listings}

\title{K\'onyvolvas\'as}
\date{\today}
\author{N\'andi B\'atfai \and Gr\'eta B\'atfai \and Matyi B\'atfai \and
      Norbi B\'atfai }

\begin{document}

\maketitle

\section{MatLog}

A Péter Rózsa: Játék a végtelennel című könyvével kezdtünk. A 261. oldal 3. ↵
      bekezdésétől kb. a 265. oldalig jeleltük le:

\begin{itemize}
\item
Bevezetés a matematikai logikába 1., \url{https://youtu.be/jlIBkFO3UNk}
\end{itemize}

Majd Dragálin Albert: Bevezetés a matematikai logikába című könyvének ↵
      jelöléseivel (használva közben a \url{http://detexify.kirelabs.org/ ↵
      classify.html} lapot a legmegfelelőbb logikai szimbólumok megtalálásához ↵
      ) próbálkoztunk latex-ben:
\begin{itemize}
\item
Második TeXelés, \url{https://youtu.be/qly_9ECViBM}
\end{itemize}

Aztán a Dragálin könyv 44. oldalával folytattuk, a természetes nyelvű ↵
      mondatok matlog átíratával.

\subsection{Természetes nyelvű mondatok matlog átírata}

Dragálin Albert: Bevezetés a matematikai logikába, 44. oldal.

\subsubsection{A könyv példái}

Első stream.

\begin{itemize}
\item
```

```

Minden hal, kivéve a cápát, szereti a gyerekeket. \\

$$\forall x (\neg \text{Cápa}(x) \supset \text{SzeretiGyerekek}(x))$$


\item
Minden ember, kivéve a pacifistákat, szereti a fegyvereket. \\

$$\forall x (\neg \text{pacifista}(x) \supset \text{Szeretifegyver}(x))$$


\item
Péter akkor szellemes, ha részeg\\

$$\text{PéterRészeg} \supset \text{PéterSzellemes}$$


\item
Péter csak akkor szellemes, ha részeg.\\

$$\text{PéterSzellemes} \supset \text{PéterRészeg}$$


\item
Péter ha részeg, akkor szellemes.\\

$$\text{PéterRészeg} \supset \text{PéterSzellemes}$$


\item
Péter akkor és csak akkor szellemes, ha részeg.\\

$$(\text{PéterRészeg} \supset \text{PéterSzellemes}) \wedge (\text{PéterSzellemes} \supset \text{PéterRészeg})$$


\end{itemize}

Második stream.

\begin{itemize}
\item
Minden ember szeret valakit. \\

$$\forall x \exists y \text{ Szeret}(x, y)$$


\item
Valakit minden ember szeret. \\

$$\exists y \forall x \text{ Szeret}(x, y)$$

\textit{vegyük észre, hogy mi a } \Leftarrow \text{ különbség az előző formulával összehasonlítva: a kvantokok sorrendje } \Leftarrow \text{ számít!}

\item
Senki nem szeret mindenkit. \\

$$\neg \exists x \forall y \text{ Szeret}(x, y)$$


\item
Valaki szeret mindenkit. \\

$$\exists x \forall y \text{ Szeret}(x, y)$$


\item

```

```

Valaki senkit sem szeret. \\
 $\exists x \forall y \neg \text{Szeret}(x, y)$ 

\item
Mindenki szeret valakit vagy valaki szeret mindenkit. \\
 $\forall x \exists y \text{Szeret}(x, y) \vee \exists x \forall y \text{Szeret}(x, y)$ 

\end{itemize}

\subsection{A matlog nyelv}

A matematikai logikai (matlog) nyelv a beszélt nyelvnél pontosabb kifejező ←
eszköz. A beszélt nyelv szavakból és mondatokból épül, még a matlog ←
nyelv termékből és formulákból. A termék változókból és függvényekből ←
épülnek. Term például az az  $x$  változó, ami befutotta az embereket. A ←
formulák termékből és prédikátumokból épülnek. Formula például a ←
korábban használt  $\text{Szeret}(x, y)$  két változós prédikátum. Illetve a ←
logikai jelekkel a formulákból további formulák építhetők fel. Ha  $A$  és ←
 $B$  formula, akkor a

\begin{itemize}
\item
 $(A \wedge B)$  is az, olvasva " $A$  és  $B$ ", \LaTeX -ben leírva:  $\verb!$(A \wedge B)$!$ 

\item
 $(A \vee B)$  is az, olvasva " $A$  vagy  $B$ ", \LaTeX -ben leírva:  $\verb!$(A \vee B)$!$ 

\item
 $(A \supset B)$  is az, olvasva " $A$ -ból következik  $B$ ", \LaTeX -ben leírva:  $\verb!$(A \supset B)$!$ 

\item
 $\neg A$  is az, olvasva "Nem  $A$ ", \LaTeX -ben leírva:  $\verb!$\neg A$!$ 

\item
 $\exists x A$  is az, olvasva "Van olyan  $x$ , hogy  $A$ ", \LaTeX -ben leírva:  $\verb!$\exists x A$!$ 

\item
 $\forall x A$  is az, olvasva "Minden  $x$ -re  $A$ ", \LaTeX -ben leírva:  $\verb!$\forall x A$!$ 
\end{itemize}

\subsubsection{Az Ar matlog nyelv}

A  $0$ ,  $x$ ,  $y$ ,  $z \dots$  változók természetes szám típusúak (a  $0$  egy ←
konstans, azaz nem változó változó, a  $0$  természetes szám jele).

Az  $SS$ ,  $+$ ,  $\cdot$  függvények.

A termék építésének kódja:
\begin{enumerate}
\item

```

minden változó neve term, például 0 , x

\item

ha t term, akkor St is az, például Sx , $SSS0$, $S(0+x)$

\item

ha t , v termék, akkor $(t+v)$ és $(t \cdot v)$ is termék, például $(x \leftarrow +0)$, $(Sx+SSS0)$.

\end{enumerate}

Formulák építéséhez egyetlen prédikátum van, az $\text{Egyenlő}(t, v)$, \leftarrow amit olvashatóbban így írunk $(t=v)$.

Például az $((x+0) = SSS0)$ egy formula. Ez a formula az $x=3$ kiértékelés mellett igaz egyébként hamis.

Mikor igaz az $(x+y = Sx+x)$ formula? Mikor igaz az $(x = SSy)$ formula?

\paragraph{Alapozzuk meg a következő formulákat!}

Lásd a [\url{https://www.twitch.tv/nbatfai}](https://www.twitch.tv/nbatfai) csatornán közvetített élő adások archívumát a [\url{https://www.youtube.com/c/nbatfai}](https://www.youtube.com/c/nbatfai) YTB csatornán, konkrétan például: [\url{https://youtu.be/ZexiPy3ZxsA}](https://youtu.be/ZexiPy3ZxsA) és [\url{https://youtu.be/DUGPRlXk_2w}](https://youtu.be/DUGPRlXk_2w).

\begin{itemize}

\item

$(x \leq y) \rightarrow \exists z (z+x=y)$

\item

$(x \neq y) \rightarrow \neg (x=y)$

\item

$(x < y) \rightarrow \exists z (z+x=y) \wedge \neg (x=y)$

\item

$(x < y) \rightarrow (x \leq y) \wedge (x \neq y)$

\item

$(x \perp y) \rightarrow \exists z (z \cdot x=y) \wedge (x \neq 0)$

\item

$(x \text{ páros}) \rightarrow (SS0 \perp x)$

\item

$(\infty \text{ sok szám van}) \rightarrow (\forall x \exists y (x < y \rightarrow))$

\item

$(\text{véges sok szám van}) \rightarrow (\exists y \forall x (x < y \rightarrow))$

\item

```

$$(x \text{ \textit{prím}}) \leftrightharpoons (\forall z (z \text{ \textit{vert } } x) \supset (z = x \vee z=S0)) \wedge (x \neq 0) \wedge (x \neq S0)$$

```

```
\item
```

```

$$(\infty \text{ \textit{sok prím szám van}}) \leftrightharpoons (\forall x \exists y ((x < y) \wedge (y \text{ \textit{prím}})))$$

```

```
\item
```

```

$$(\infty \text{ \textit{sok iker-prím szám van}}) \leftrightharpoons (\forall x \exists y ((x < y) \wedge (y \text{ \textit{prím}}) \wedge (S y \text{ \textit{prím}})))$$

```

```
\item
```

```

$$(\text{véges sok prím szám van}) \leftrightharpoons (\exists y \forall x (x \text{ \textit{prím}}) \supset (x < y))$$

```

```
\item
```

```

$$(\text{véges sok prím szám van}) \leftrightharpoons (\exists y \forall x (y < x) \supset \neg (x \text{ \textit{prím}}))$$

```

```
\end{itemize}
```

Olvassuk el a most feldolgozott, Dragálin könyv 15-19 oldalait!

```
\paragraph{"Hol a tagadás lábát megveti"}
```

Felmerült, hogy a $(\text{véges sok prím szám van})$ kifejezhető lenne a $(\infty \text{ \textit{sok prím szám van}})$ tagadásaként, azaz $(\text{véges sok prím szám van}) \leftrightharpoons \neg(\infty \text{ \textit{sok prím szám van}})$

Tagadjuk az egyik végeességet kifejező formulánkat!

```
\begin{align*}
```

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

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

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

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

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

```
\end{align*}
```

az eredmény éppen a végtelenséges formulánk volt!

Ismételjük meg ezt a számítást a másik formulával!

```
\begin{align*}
```

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

```
\end{align*}
```



```
\end{document}
```

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

Ez egy `int` típusú változó lesz.

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

Ez egy `int` típusú mutató, aminek átadjuk az `a` változó referenciaértékét (tehát a memóriacímet).

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

Ilyen nincs a C nyelvben. A "&"-jellel csak átadni tudjuk egy változó referenciáját, de így deklarálni egy referenciát nem lehet.

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

Ez egy `int` típusú, 5 elemű egydimenziós tömb.

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

Még mindig nem lehet így változót deklarálni C-ben.

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

Ez egy 5 elemű pointertömb, amely elemei `int` típusú értékekre mutatnak.

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

Egy olyan függvény, ami egészre mutató pointert ad vissza.

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

Egy olyan függvényre mutató pointer, ami egészre mutató pointert ad vissza.

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

Egészret visszaadó és két egészet kapó függvényre mutató pointert visszaadó, egészet kapó függvény.

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

Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató pointert visszaadó, egészet kapó függvény.

Megoldás videó:

Megoldás forrása:

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/caesar/haromszog.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/caesar/haromszog.c)

Egy 5 soros háromszögmátrixot fogunk létrehozni. Ezt egy double típusú pointerekre mutató pointerrel fogjuk megvalósítani. A malloc(size) paranccsal lefoglaljuk a sorok mutatói számára a memóriát. Hogy tudjuk mennyi bájtot foglaljunk le, a sizeof() függvénnyel lekérjük egy double méretét és megszorozzuk a sorok számával és a visszaadott pointert double ** típusúra kényszerítjük. Megvizsgáljuk, hogy nem e kapott valamelyik elem NULL értéket, mert az azt jelentené, hogy nincs elég memória és kilép a programból.

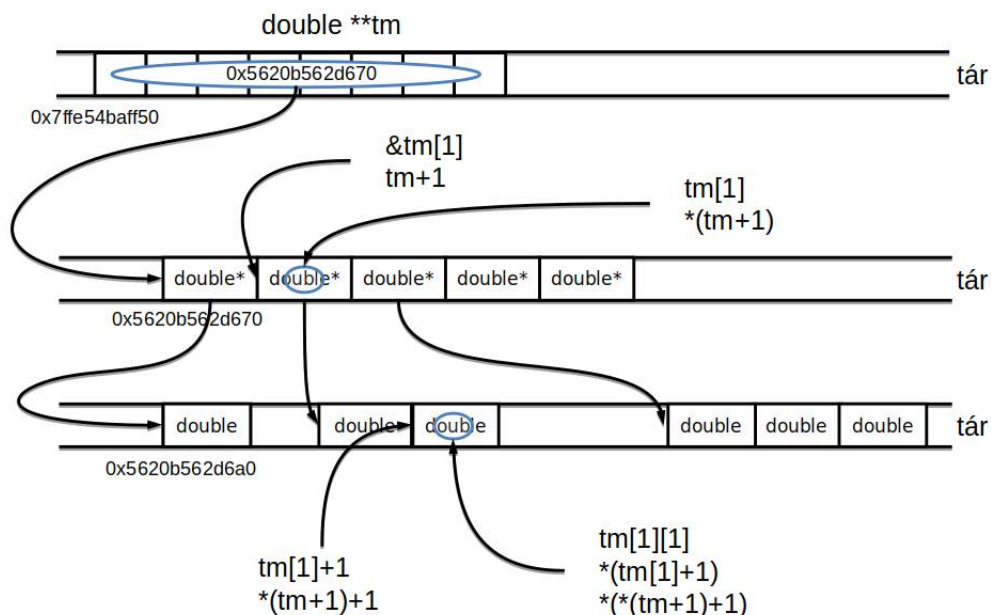
```
int nr = 5;
double **tm;

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

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

A for ciklussal lefoglaljuk az egyre növekvő elemszámú sorok elemeinek a memóriáját, amit szintén megvizsgálunk a Null értékkel.

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



Az elemek értékeinek módosítgatási után a `free()` eljárás használatával felszabadítjuk először az elemekre, majd a sorokra lefoglalt memóriát.

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

free (tm);
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/caesar/e.c

A titkosítást EXOR művelettel fogjuk végrehajtani egy általunk megadott kulcs és a titkosítandó szöveg között.

A `kulcs_meret` változóban eltároljuk a futtatáskor átadott második argumentum (tehát a kulcs) hosszát. Majd a `strncpy(hova, honnan, méret)` függvénnyel a `kulcs` változóba másoljuk a kulcsot.

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

Folyamatosan beolvassuk a szöveget és a kulcs egyes elemeivel végrehajtjuk rajta a műveletet, miközben kiírjuk a titkosított szöveget a standard outputra.

```
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 futtatáskor megadjuk a kulcsot a titkosításhoz és a kimenetet átírányítjuk a titkos.szoveg állományba:

`./e 56789012 <tiszta.txt > titkos.szoveg`

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/caesar/ExorTitkosító.java

```
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 olvasottBajtok = 0;

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

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

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

            }

            kimenőCsatorna.write(buffer, 0, olvasottBajtok);

        }

    }
}
```

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

A programkód átírása során néhány változó deklarálása feleslegessé vált. Ilyen például a MAX_KULCS állandó. Míg a c verzióban előre deklaráltuk a kulcs tömböt és csak azután adtunk neki értéket, így volt haszna. Itt viszont a kulcs tömb megfelelő méretűre deklarálódik. A BUFFER_MERET változót is elhagytuk, bár az az előző kódban sem volt szükséges.

```
byte [] kulcs = kulcsSzöveg.getBytes();  
byte [] buffer = new byte[256];
```

A kulcs_meret változót is elhagytuk, mivel azt bármikor lekérhetjük, mint egy tömb méretét:

```
kulcsIndex = (kulcsIndex+1) % kulcs.length;
```

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/caesar/t.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/caesar/t.c)

[bhax/thematic_tutorials/bhax_textbook/caesar/t_par.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/caesar/t_par.c)

Hiába tudjuk milyen módon lett titkosítva a szöveg, ahhoz hogy dekódoljuk, meg kell találnunk a kulcsot. Ezt úgy fogjuk csinálni, hogy szépen sorba végigmegyünk az összes kulcson és közbe keresünk néhány "kulcsszót", amelyek gyakran előfordulhatnak, ezért ha megegyeznek a dekódolt szóval, akkor kiíratjuk ezt a kulcsot és a szöveget, mint lehetséges megoldás és tovább keresünk. Erre azért van szükség, mert megeshet, hogy a kulcsszavak helyesen lettek dekódolva, de ettől még sok karakter eltér a várt eredménytől.

A kulcsok legenerálása és kipróbálása annyi egymásba ágyazott for ciklust igényel, amekkora a kulcs mérete. Ez esetben tudjuk, hogy 8 karakter hosszú.

```
// 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 ←
                    // második buffer
                    exor (kulcs, KULCS_MERET, titkos, p - ←
                        titkos);
                }
}
```

A következő 2 függvény fogja nekünk eldönteni, hogy rátaláltunk-e a keresett kulcsra. Ha az átlagos szóhossz 6 és 9 közé esik, valamint a dekódolt szövegben megtalálható a "hogya", a "nem", az "az" és a "ha" szavak, akkor kiírjuk a dekódolt szöveget.

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

    return (double) titkos_meret / sz;
}

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

Mivel a kulcsok egyesével való kipróbálgatása elég időigényes, a for ciklusokat párhuzamosítjuk a végtelen ciklusoknál használt módon.

```
// osszes kulcs eloallitasa
#pragma omp parallel for private(kulcs)
  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;

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

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...

```
# 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://youtu.be/Koyw6IH5ScQ

library(neuralnet)

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

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

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

plot(nn.or)

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

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

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

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

plot(nn.orand)

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

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

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)

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

4.6. Hiba-visszaterjesztéses perceptron

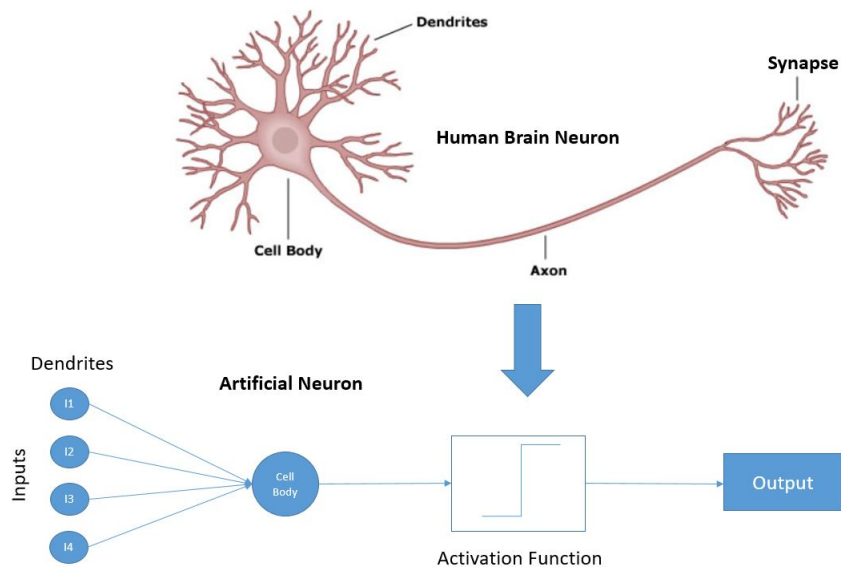
C++

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/caesar/percr.hpp

http://www.cs.ubbcluj.ro/~csatol/mestint/pdfs/neur_halo_alap.pdf

A feladat megoldásában tutorált engem: Szilágyi Csaba.



Az agyi neuronok bemenetei a dendritek, amelyek más neuronokhoz csatlakoznak. A tényleges feldolgozást a neuron tulajdonképpeni sejtmagja végzi, és az eredményt az axonon keresztül juttatja el a többi neuronhoz. Az axon végződése több másik neuron dendritjeihez csatlakozhatnak.

Egy neurális hálózat több neuronból állnak, amelyek rétegekbe vannak sorolva:

- Bemeneti réteg:** azok a neuronok találhatók itt, amelyek a bemeneti jel továbbítását végzik a hálózat felé. A legtöbb esetben nem jelöljük őket külön.
- Rejtett réteg:** a tulajdonképpeni feldolgozást végző neuronok tartoznak ide. Egy hálózaton belül több rejtett réteg is lehet.
- Kimeneti réteg:** az itt található neuronok a külvilág felé továbbítják az információt. A feladatuk ugyanaz, mint a rejtett rétegbeli neuronoké.

A neurális hálózatok újralfedezése akkor kezdődött, amikor a 80-as évek elején felfedezték a tanítására alkalmas hibavisszaterjesztéses algoritmust. A többrétegű perceptron hálózatokban perceptronokat kötünk össze.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

telepíteni kell a qt-t!

Megoldás videó:

Megoldás forrása: [bmax/thematic_tutorials/bmax_textbook/mandelbrot/mandelpngt.cpp](https://bmax.thematic_tutorials/bmax_textbook/mandelbrot/mandelpngt.cpp)

<https://hirmagazin.sulinet.hu/HU/pedagogia/mandelbrot-halmaz>

Ez a halmaz Benoit B. Mandelbrot, lengyel származású matematikusról, a megalkotójától kapta a nevét. (A képe e cikk ajánlója mellett látható.) Hogyan építhető fel ez a halmaz? A komplex számsíkon veszünk egy C pontot.

Erre képezzük a következő sorozatot:

$$Z_0 := C$$

$$Z_{i+1} := Z_i^2 + C$$

Ezzel az egyszerű rekurzióval definiált sorozatról be lehet bizonyítani, hogy bizonyos C számok választása esetében végtelenbe tart, vagy más C -k esetében pedig nullához tart. Más eset nem lehetséges. A Mandelbrot-halmaznak azok és csak azok a C komplex számok az elemei, amelyek esetében a fenti sorozat nullához tart. Ábrázolás esetében ezeket általában feketére szokták festeni, míg a többi pontot attól függően, hogy "milyen gyorsan" tart a végtelenbe.

Az elso programunk a CPUt használva számolja ki a Mandelbrot halmaz elemeit. A fordításhoz szükségünk lesz néhány kapcsolóra: **g++ mandelpngt.cpp -o mandelpngt -lpng16 -O3**

```
// mandelpngt.c++  
// Copyright (C) 2019  
// Norbert Bاتفai, 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, ↵  
ormandelbrot/mandelpngt.cpp
```

```
// (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.
//mandelbrot/mandelpngt.cppmandelbrot/mandelpngt.cpp
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//mandelbrot/mandelpngt.cpp
// Mandelbrot png
// Programozó Páternosztter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↵
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
//
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
#define ITER_HAT 32000

void
mandel (int kepadat[MERET][MERET]) {

    // Mérünk időt (PP 64)mandelbrot/mandelpngt.cppmandelbrot/mandelpngt. ↵
    cppmandelbrot/mandelpngt.cpp
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reZ, imZ, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
```

```
for (int k = 0; k < szelesseg; ++k)
{
    // c = (reC, imC) a rács csomópontjainak
    // megfelelő komplex szám
    reC = a + k * dx;
    imC = d - j * dy;
    // z_0 = 0 = (reZ, imZ)
    reZ = 0;
    imZ = 0;
    iteracio = 0;
    // z_{n+1} = z_n * z_n + c iterációk
    // számítása, amíg |z_n| < 2 vagy még
    // nem értük el a 255 iterációt, ha
    // viszont elértük, akkor úgy vesszük,
    // hogy a kiindulási c komplex számra
    // az iteráció konvergens, azaz a c a
    // Mandelbrot halmaz eleme
    while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
        // z_{n+1} = z_n * z_n + c
        ujreZ = reZ * reZ - imZ * imZ + reC;
        ujimZ = 2 * reZ * imZ + imC;
        reZ = ujreZ;
        imZ = ujimZ;

        ++iteracio;
    }

    kepadat[j][k] = iteracio;
}

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

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

int
main (int argc, char *argv[])
{
    if (argc != 2)
    {
        std::cout << "Hasznalat: ./mandelpng fajlnev";
        return -1;
    }
}
```

```
}

int kepadat[MERET][MERET];

mandel(kepadat);

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

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

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

}
```

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

Megoldás videó:

Megoldás forrása: [bmax/thematic-tutorials/bmax-textbook/mandelbrot/mandelpngt.cpp](https://bmax.thematic-tutorials.com/bmax-textbook/mandelbrot/mandelpngt.cpp)

A feladat ismét a Mandelbrot-halmaz kirajzoltatása egy fájlba, most viszont a komplex számok használatahoz `complex` típust fogunk használni. Egy ilyen típus deklarációjakor először megadjuk a valós, majd az imaginárius értékét a komplex számnak.

```
std::complex<double> c ( valos, imaginarius );
```

A fordításhoz és futtatáshoz szükséges parancsok a forráskód elején szerepelnek. Ez a verzió már sokkal több lehetőséget tár elénk, mivel megadhatjuk a számolás paramétereit, mint pl. a kimeneti kép mérete vagy az n értéke.

```
// 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 -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 Bاتفai, 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: bmax/thematic_tutorials/bmax_textbook/mandelbrot/3.1.3.cpp
https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_31_fraktalok/a_juliahalmazok.scorm1

Nem véletlen a többes szám, mivel Julia halmazból végtelen sok van. Hasonlóan indulunk, mint a Mandelbrot halmaznál, veszünk egy $f(z)=z^2+c$ kvadratikus komplex együtthatós polinomot. Most viszont a c konstansot rögzítjük és különböző x értékekre vizsgáljuk a pályákat. Hasonlóan a Mandelbrot halmazhoz, itt is a nem divergens pályájú pontok fognak a keresett halmazhoz tartozni. Mivel a c értéket bárhogy választhatjuk, végtelen sok Julia halmazt kapunk.

Ezekre a biomorfokra Pickover bukkant rá, amikor egy Júlia-halmazt akart kirajzoltatni, viszont az iteráció feltételvizsgálatát hibásan írta meg: minden egyes pont kirajzolása előtt megvizsgálta, hogy a komplex szám valós vagy képzetes része kisebb-e, mint az iterációs határ.

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatás:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" <-
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= <-
color
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// 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:

Megoldás forrása:

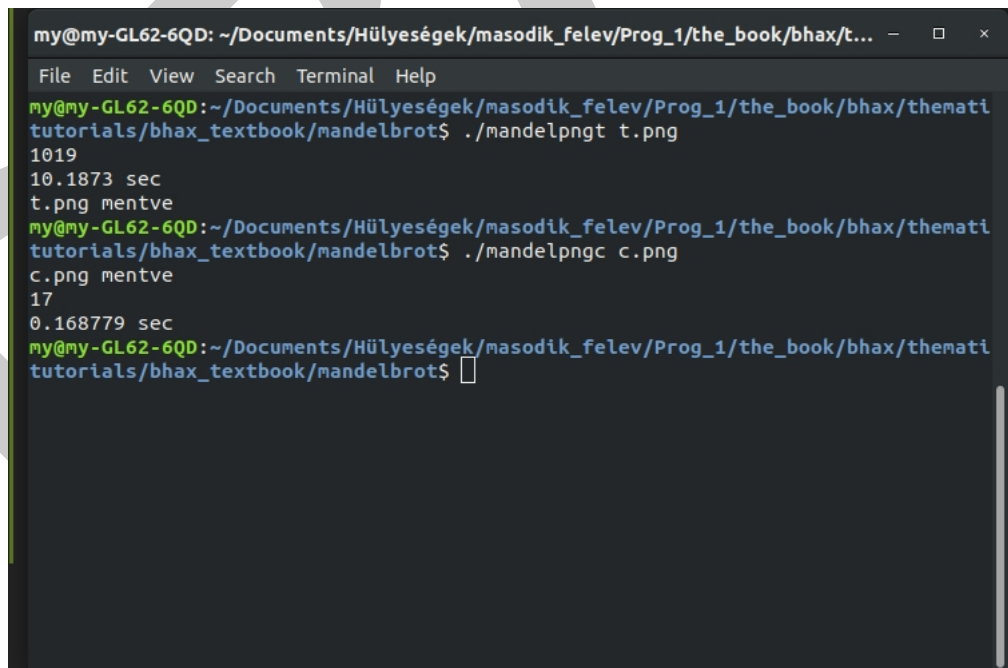
[bhax/thematic-tutorials/bhax-textbook/mandelbrot/mandelpngc_60x60_100.cu](https://bhax.thematic-tutorials.com/bhax-textbook/mandelbrot/mandelpngc_60x60_100.cu)

Hogy sokkal gyorsabbá tegyük a programot, az Nvidia által kifejlesztett CUDA nyújtotta lehetőséget fogjuk kihasználni. Ehhez szükséges a csomag telepítése:

sudo apt install nvidia-cuda-toolkit

nvcc mandelpngc_60x60_100.cu -lpng16 -O3 -o mandelpngc

Ezután ha lefordítjuk és futtatjuk a kódot, láthatjuk, hogy a sebesség kb. a 60-szorosára nő.



```
my@my-GL62-6QD: ~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/t...
File Edit View Search Terminal Help
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/themati
tutorials/bhax_textbook/mandelbrot$ ./mandelpngt t.png
1019
10.1873 sec
t.png mentve
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/themati
tutorials/bhax_textbook/mandelbrot$ ./mandelpngc c.png
c.png mentve
17
0.168779 sec
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/themati
tutorials/bhax_textbook/mandelbrot$
```

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 forrása:

Megoldás videó:

Megoldás forrása:

A feladat megoldásában tutorált engem: Tóth Balázs.

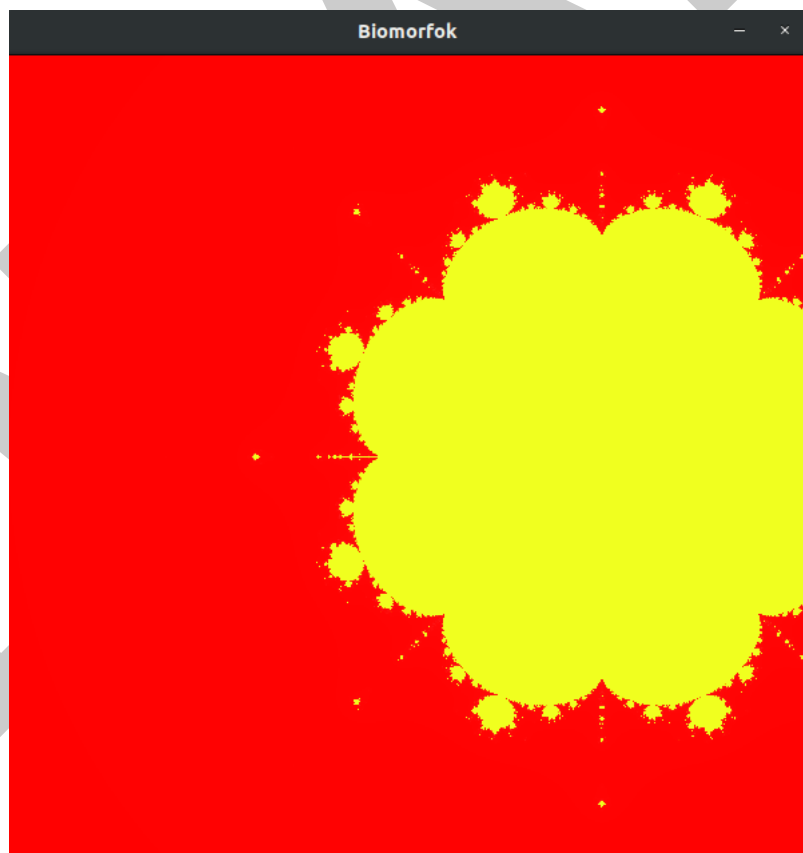
Ehhez a feladathoz szükséges a QT telepítése:

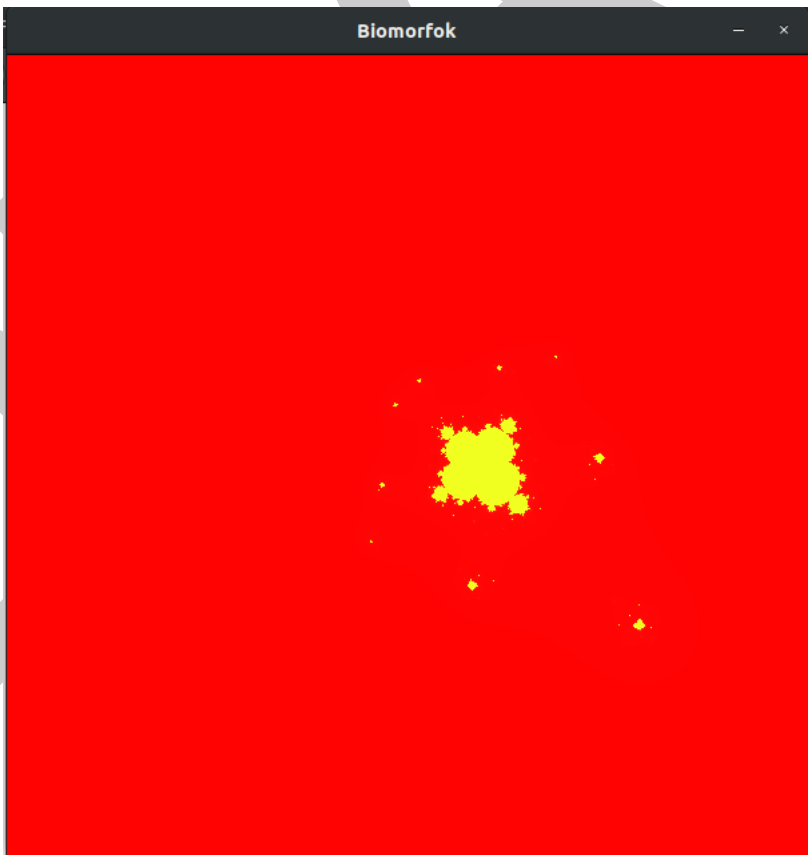
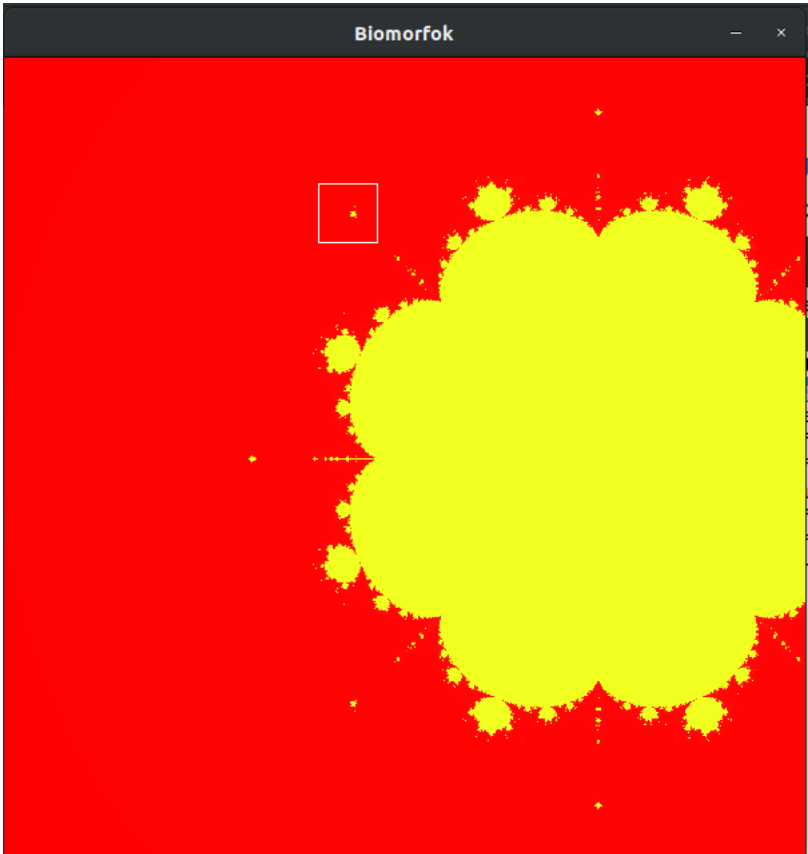
sudo apt-get install qt5-default

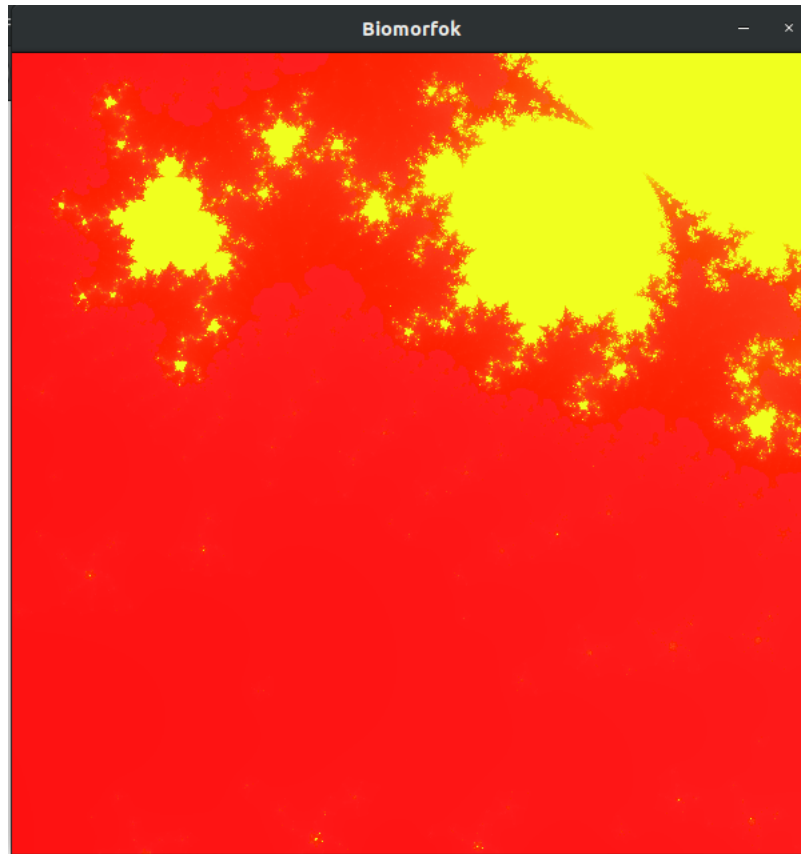
Ezután érdemes az összes forráskódot elkülöníteni egy mappába, ahol a **qmake -project** paranccsal létrehozuk a valami.pro fájlt, majd a **qmake valami.pro** paranccsal a MakeFile-t. A futtatáshoz ki kell egészítünk a valami.pro fájlt a következő sorokkal:

```
QT += widgets
CONFIG += c++11
```

Egy make parancs után már futtatható is a program.







5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása:

[bhax/thematic_tutorials/bhax_textbook/mandelbrot/java/MandelbrotHalmazNagyító.java](#)

[bhax/thematic_tutorials/bhax_textbook/mandelbrot/java/MandelbrotHalmaz.java](#)

[bhax/thematic_tutorials/bhax_textbook/mandelbrot/java/MandelbrotIterációk.java](#)

Itt a grafikus megjelenítéshez az AWT csomagot fogjuk használni. Az általa nyújtott `BufferedImage` osztály nagyban megkönnyíti a dolgunkat, főleg amikor pillanatfelvételt készítünk.

```
public void pillanatfelvétel() {  
    // Az elmentendő kép elkészítése:  
    java.awt.image.BufferedImage mentKép =  
        new java.awt.image.BufferedImage(szélesség, magasság,  
            java.awt.image.BufferedImage.TYPE_INT_RGB);  
    java.awt.Graphics g = mentKép.getGraphics();  
    g.drawImage(kép, 0, 0, this);  
    g.setColor(java.awt.Color.BLACK);  
    g.drawString("a=" + a, 10, 15);  
    g.drawString("b=" + b, 10, 30);  
}
```



```
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iterációsHatár, 10, 75);
if(számításFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
// A pillanatfelvétel képfájl nevének képzése:
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyitas_");
sb.append(++pillanatfelvételSzámláló);
sb.append("_");
// A fájl nevébe bele vesszük, hogy melyik tartományban
// találtuk a halmazt:
sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");
// png formátumú képet mentünk
try {
    javax.imageio.ImageIO.write(mentKép, "png",
        new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
    e.printStackTrace();
}
}
```

A kijelöléshez és nagyításhoz elmentjük az egér pozícióját az egérgomb lenyomásakor és felengedésekor, valamint amíg lenyomva tartjuk, folyamatosan újrajzolunk mindent hogy látszódjon a négyzet amivel kijelölünk.

```
addMouseListener(new java.awt.event.MouseAdapter() {
    // Egér kattintással jelöljük ki a nagyítandó területet
    // bal felső sarkát vagy ugyancsak egér kattintással
    // vizsgáljuk egy adott pont iterációit:
    public void mousePressed(java.awt.event.MouseEvent m) {
        // Az egérmutató pozíciója
        x = m.getX();
        y = m.getY();
        // Az 1. egér gombbal a nagyítandó terület kijelölését
        // végezzük:
        if(m.getButton() == java.awt.event.MouseEvent.BUTTON1 ) {
```

```
        // A nagyítandó kijelölt területet bal felső sarka: (x, ↵
        y)
        // és szélessége (majd a vonszolás növeli)
        mx = 0;
        my = 0;
        repaint();
    } else {
        // Nem az 1. egér gombbal az egérmutató mutatta c
        // komplex számból indított iterációkat vizsgálhatjuk
        MandelbrotIterációk iterációk =
            new MandelbrotIterációk(
                MandelbrotHalmazNagyító.this, 50);
        new Thread(iterációk).start();
    }
}
// Vonszolva kijelölünk egy területet...
// Ha felengedjük, akkor a kijelölt terület
// újraszámítása indul:
public void mouseReleased(java.awt.event.MouseEvent m) {
    if(m.getButton() == java.awt.event.MouseEvent.BUTTON1 ) {
        double dx = (MandelbrotHalmazNagyító.this.b
            - MandelbrotHalmazNagyító.this.a)
            /MandelbrotHalmazNagyító.this.szélesség;
        double dy = (MandelbrotHalmazNagyító.this.d
            - MandelbrotHalmazNagyító.this.c)
            /MandelbrotHalmazNagyító.this.magasság;
        // Az új Mandelbrot nagyító objektum elkészítése:
        new MandelbrotHalmazNagyító(
            MandelbrotHalmazNagyító.this.a+x*dx,
            MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
            MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
            MandelbrotHalmazNagyító.this.d-y*dy,
            600,
            MandelbrotHalmazNagyító.this.iterációsHatár);
    }
}
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // A nagyítandó kijelölt terület szélessége és magassága:
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
});
```

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/welch/PolárGenerátor.java

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!

```
public class PolárGenerátor {

    boolean nincsTárolt = true;
    double tárolt;

    public PolárGenerátor() {

        nincsTárolt = true;

    }

    public double következő() {

        if(nincsTárolt) {

            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();

                v1 = 2*u1 - 1;
```

```
        v2 = 2*u2 - 1;

        w = v1*v1 + v2*v2;

        } while(w > 1);

        double r = Math.sqrt((-2*Math.log(w))/w);

        tárolt = r*v2;
        nincsTárolt = !nincsTárolt;

        return r*v1;

    } else {
        nincsTárolt = !nincsTárolt;
        return tárolt;
    }
}

public static void main(String[] args) {

    PolárGenerátor g = new PolárGenerátor();

    for(int i=0; i<10; ++i)
        System.out.println(g.következő());

}

}
```

Ha eltekintünk attól, hogy a logikai változót fordítva használták, a kód tényleg ugyanúgy van megírva, ahogyan azt mi tettük.

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

```
}
```

Most hogy önbizalmat kaptunk, csináljuk me a c++ verziót is:

```
#include "std_lib_facilities.h"
class PolarGenerator{
bool nincsTarolt=true;
double tarolt;
public :
double kovetkezo()
{
    if(nincsTarolt)
    {
        double u1,u2,v1,v2,w;
        u1= ((double) rand() / (double) (RAND_MAX));
        u2= ((double) rand() / (double) (RAND_MAX));
        v1=(2*u1)-1;
        v2=(2*u2)-1;

        w=(v1*v1)+(v2*v2);
        while(w>1)
        {double r = sqrt((-2*log(w))/w);
            tarolt=r*v2;
            nincsTarolt=!nincsTarolt;
            return r*v1;
        }
    }
    else
    {
        nincsTarolt=!nincsTarolt;
        return tarolt;
    }
};
};
int main()
{
std::srand(std::time(0));
PolarGenerator g;
for(int i=0; i<10; ++i)
    std::cout<<g.kovetkezo()<<std::endl;
return 0;
}
```

6.2. LZW

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/welch/binfa_c.c

Egy fa adatszerkezet leprogramozására c-ben a `struct` típus fog segíteni. Egy ilyen fa ugyanolyan szerkezetű elemekből áll: érték(gyökér), mutató a jobboldali levélelemre, mutató a beloldali levélelemre.

```
typedef struct binfa
{
    int ertek;
    struct binfa *bal_nulla;
    struct binfa *jobb_egy;
} BINFA, *BINFA_PTR;
```

Ezután az `uj_elem()` függvénnyel lefoglaljuk a memóriát a fa gyökér elemének és ha nincs neki elég memória, akkor egy hibaüzenet után kilépünk a programból.

```
BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

    if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
        perror ("memoria");
        exit (EXIT_FAILURE);
    }
    return p;
}
```

A `while` ciklus fejébe folyamatosan beolvassuk a futtatáskor átadott txt állomány karaktereit, amelyek a fa egyes elemei. A 0-ást balra, az 1-est jobbra fogjuk beágyazni a fába úgy, hogy ha ismeretlen bitsorozathoz jutunk, akkor azt "letörjük" és beletesszük a fába:

00011101110 -> 0 00 1 11 01 110

```

      /
     0      1
    0      1      1
           0
```

```
char b;

BINFA_PTR gyoker = uj_elem ();
gyoker->ertek = '/';
BINFA_PTR fa = gyoker;

while (read (0, (void *) &b, 1))
{
    write (1, &b, 1);
    if (b == '0')
    {
        if (fa->bal_nulla == NULL)
```

```
    {
        fa->bal_nulla = uj_elem ();
        fa->bal_nulla->ertek = 0;
        fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
        fa = gyoker;
    }
    else
    {
        fa = fa->bal_nulla;
    }
}
else
{
    if (fa->jobb_egy == NULL)
    {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
    }
    else
    {
        fa = fa->jobb_egy;
    }
}
}
```

A kiíratásnál a fát inorder módon bejárva kiíratjuk a fa elemeit és azok mélységét. Majd a végén felszabadítjuk a memóriát, amelyet elemenként kell elvégeznünk.

```
printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d", max_melyseg);
szabadit (gyoker);
}

static int melyseg = 0;
int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        kiir (elem->jobb_egy);
        for (int i = 0; i < melyseg; ++i)
```



```
{
    ++melyseg;
    if (melyseg > max_melyseg)
        max_melyseg = melyseg;

    // gyokerelem feldolgozasa
    for (int i = 0; i < melyseg; ++i)
        printf ("---");

    printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek <=
        , melyseg);

    // baloldali reszfa preorder bejarasa
    kiir (elem->bal_nulla);

    // jobboldali reszfa preorder bejarasa
    kiir (elem->jobb_egy);

    --melyseg;
}
}
```

Posztorder bejárás: ha a fa üres, akkor vége az eljárásnak. Ha nem, akkor posztorder módon járjuk be a bal oldali, majd a jobb oldali részfát, végül dolgozzuk fel a gyökérelemet.

```
void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        // baloldali reszfa preorder bejarasa
        kiir (elem->bal_nulla);

        // jobboldali reszfa preorder bejarasa
        kiir (elem->jobb_egy);

        // gyokerelem feldolgozasa
        for (int i = 0; i < melyseg; ++i)
            printf ("---");

        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek <=
            , melyseg);

        --melyseg;
    }
}
```

A pre- és posztorder módon bejárt fa nem valami látványos kirajzolva, viszont más célú adatfeldolgozásnál hasznosak lehetnek.

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: [bmax/thematic-tutorials/bmax-textbook/welch/z3a7.cpp](https://bmax.thematic-tutorials.com/bmax-textbook/welch/z3a7.cpp)

C++ nyelven a fát osztályok/objektumok segítségével fogjuk megvalósítani, aminek következtében a memória foglалás és felszabadítása az osztály része lesz konstruktor és destruktör szerepében.

A feladatnak megfelelően elkészítjük a LZWBinFa (Tree) osztályt és benne a Csomopont (Node) beágyazott osztályt, majd a fa mutatót a gyöker elemre állítjuk a konstruktorban.

```
class LZWBinFa
{
    LZWBinFa ():fa (&gyoker)
    {
    }
    ~LZWBinFa ()
    {
        szabadit (gyoker.egyesGyermekek ());
        szabadit (gyoker.nullasGyermekek ());
    }
    ...
    class Csomopont
    {
        char betu;
        Csomopont *balNulla;
        Csomopont *jobbEgy;
    };
    ...
    Csomopont *fa;
    ...
    Csomopont gyoker;
    int maxMelyseg;
    ...
};
```

Az osztályoknak köszönhetően mindent ami a fával kapcsolatos, egy egységbe kezelhetjük, ezért a fa feltöltését isbeletettük. Ezt úgy csináltuk, hogy átírtuk a "<<" operátort.

```
void operator<< (char b)
{
    // Mit kell betenni éppen, '0'-t?
```

```
if (b == '0')
{
    /* Van '0'-s gyermeke az aktuális csomópontnak?
    megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
    if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
    {
        // elkészítjük, azaz példányosítunk a '0' betű akt. ←
        parammal
        Csomopont *uj = new Csomopont ('0');
        // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
        // jegyezze már be magának, hogy nullás gyereke mostantól ←
        van
        // küldjük is Neki a gyerek címét:
        fa->ujNullasGyermek (uj);
        // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
        fa = &gyoker;
    }
    else // ha van, arra rálépünk
    {
        // azaz a "fa" pointer már majd a szóban forgó gyermekre ←
        mutat:
        fa = fa->nullasGyermek ();
    }
}
// Mit kell betenni éppen, vagy '1'-et?
else
{
    if (!fa->egyenesGyermek ())
    {
        Csomopont *uj = new Csomopont ('1');
        fa->ujEgyenesGyermek (uj);
        fa = &gyoker;
    }
    else
    {
        fa = fa->egyenesGyermek ();
    }
}
}
```

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

    while (beFile.read ((char *) &b, sizeof (unsigned char)))
    {
```

```
...

for (int i = 0; i < 8; ++i)
{
    // maszkolunk eddig..., most már simán írjuk az if fejébe a ↵
    // legmagasabb helyiértékű bit vizsgálatát
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ↵
    // megmondja melyik:
    if (b & 0x80)
        // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW ↵
        // fa objektumunkba
        binFa << '1';
    else
        // különben meg a '0' betűt:
        binFa << '0';
    b <= 1;
}

}
...
}
```

6.5. Mutató a gyöker

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

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/welch/binfa_gyoker.cpp](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/welch/binfa_gyoker.cpp)

Ha a diff paranccsal megvizsgáljuk a 2 kódot, gyoker értékének átadása referenciahivatkozás nélkül történik.

```
128c135
<          fa = &gyoker;
---
>          fa = gyoker;
```

Ez azért van, mert mostmár a gyoker is egy mutató és így egyenesen a memóriacímet adjuk át a fa mutatónak. Emiatt a fa konstruktora a következő verzióra fejlődött:

```
LZWBinFa ():fa (gyoker = new Csomopont ('/'))
{
}
```

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

A mozgató szemantika azért lesz hasznos, mert amíg a másoló konstruktor a másoláskor megduplázza a lefoglalt memóriát, addig ezt a mozgással elkerülhetjük a mutatók használatával.

```
LZWBInFa ( LZWBInFa && regi ) {  
    std::cout << "LZWBInFa move ctor" << std::endl;  
    gyoker = regi.gyoker;  
    regi.gyoker = nullptr  
}
```

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: [bhax/thematic_tutorials/bhax_textbook/conway/hangya/main.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/hangya/antwin.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/hangya/antthread.cpp](#)

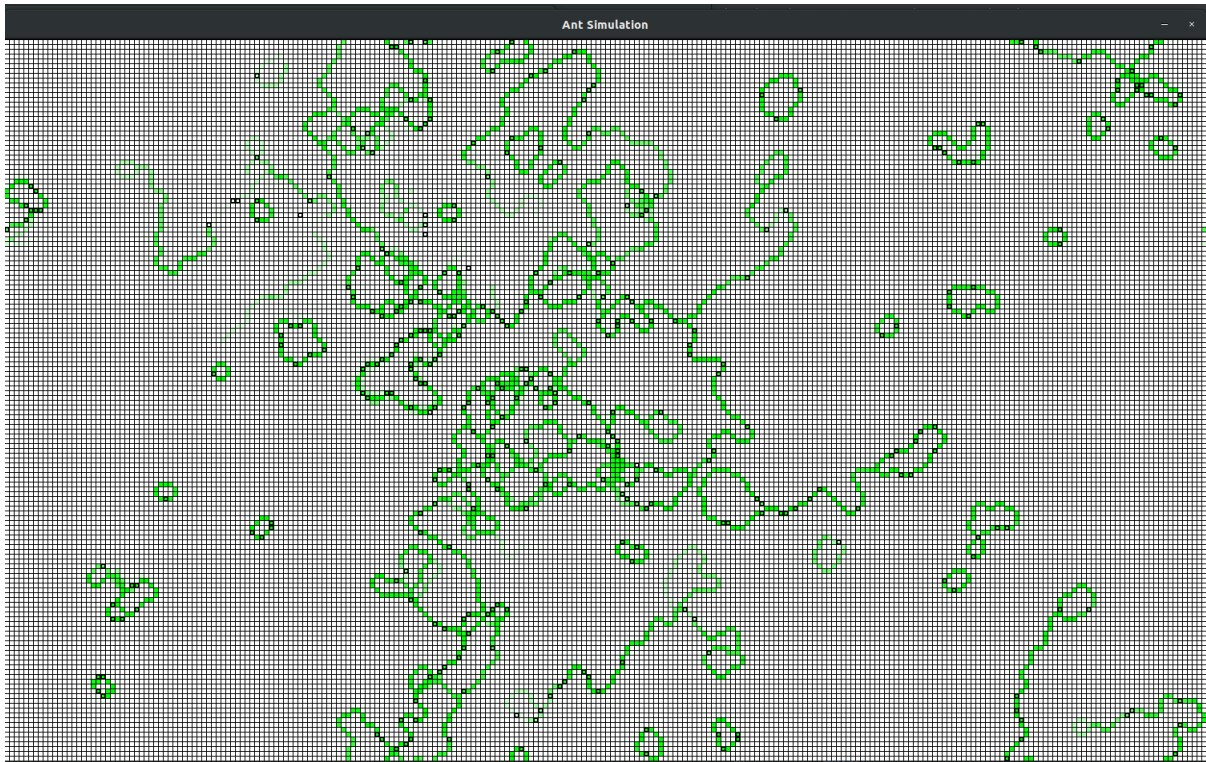
A mostani Qt projektünkkel a hangyák útvonalait fogjuk szimulálni. A hangyák igyekeznek egymás szaga alapján közlekedni, ha viszont nem érznek maguk körül feromont, akkor az irány véletlenszerű. A hangyák számát, a feromon párolgását, a szimuláció sebességét és egyéb paramétereket parancssori argumentumként adjuk meg futtatáskor.

Options:

-h, --help	Displays this help.
-v, --version	Displays version information.
-w, --szelesseg <szelesseg>	Oszlopok (cellakban) szama.
-m, --magassag <magassag>	Sorok (cellakban) szama.
-n, --hangyaszam <hangyaszam>	Hangyak szama.
-t, --sebesseg <sebesseg>	2 lepes kozotti ido (millisec-ben).
-p, --parolgas <parolgas>	A parolgas erteke.
-f, --feromon <feromon>	A hagyott nyom erteke.
-s, --szomszed <szomszed>	A hagyott nyom erteke a szomszedokban.
-d, --alapertek <alapertek>	Indulo erteke a cellakban.
-a, --maxcella <maxcella>	Cella max erteke.
-i, --mincella <mincella>	Cella min erteke.
-c, --cellameret <cellameret>	Hany hangya fer egy cellaba.

```
./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s 3 -c 22
```

Minden egyes hangyának van pozíciója és iránya. A számításokhoz 2 rácsot használunk - egyik tartalmazza a jelenlegi állapotot, a másik pedig az újat - , így olyan, mintha mind egyszerre mozognának.



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:

[bhax/thematic_tutorials/bhax_textbook/conway/életjatek/main.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/életjatek/sejtblak.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/életjatek/sejtszal.cpp](#)

Ez az ún. játék valójában egy sejtszimulátor, amit John Horton Conway talált ki 1970-ben. A játék lényege, hogy az előre elhelyezett selytek mivé formálódnak vagy hogy életben maradnak e egyáltalán.

A sejtek alakulását a következő szabályok irányítják:

- Minden sejtnak 2 állapota lehet: élő vagy halott.
- Ha egy sejtnak kevesebb mint 2 élő szomszédja van, az meghal.
- Ha egy sejtnak 2 vagy 3 élő szomszédja van, az életben marad.
- Ha egy sejtnak több mint 3 élő szomszédja van, az meghal.
- Ha egy halott sejtnek pontosan 3 élő szomszédja van, az életre kel.

```
int SejtSzal::szomszedokSzama(bool **racs,
                                int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
    // A nyolcszomszedok vgigzongorzsza:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsglt sejtet magt kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejttrbl szlnek szomszdai
                // a szembe oldalakon ("peridikus hatrfelttel")
                int o = oszlop + j;
                if(o < 0)
                    o = szelesseg-1;
                else if(o >= szelesseg)
                    o = 0;

                int s = sor + i;
                if(s < 0)
                    s = magassag-1;
                else if(s >= magassag)
                    s = 0;

                if(racs[s][o] == allapot)
                    ++allapotuSzomszed;
            }

    return allapotuSzomszed;
}

/**
 * A sejttr idbeli fejlde a John H. Conway fle
 * letjtk sejtautomata szablyai alapjn trtnik.
 * A szablyok rszletes ismerttetst lsd pldul a
 * [MATEK JTK] hivatkozsban (Cskny Bla: Diszkr
 * matematikai jtkok. Polygon, Szeged 1998. 171. oldal.)
 */
void SejtSzal::idoFejlodes() {
    bool **racsElotte = racsok[racsIndex];
    bool **racsUtana = racsok[(racsIndex+1)%2];
```

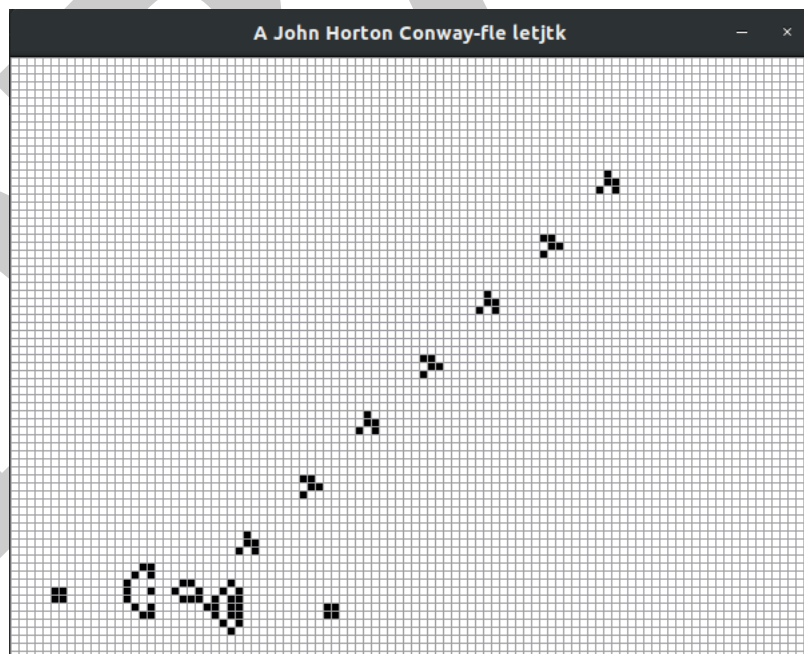


```
for(int i=0; i<magassag; ++i) { // sorok
    for(int j=0; j<szelesseg; ++j) { // oszlopok

        int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);

        if(racsElotte[i][j] == SejtAblak::ELO) {
            /* 1 1 marad, ha kett vagy hrom 1
            szomszedja van, klnben halott lesz. */
            if(elok==2 || elok==3)
                racsUtana[i][j] = SejtAblak::ELO;
            else
                racsUtana[i][j] = SejtAblak::HALOTT;
        } else {
            /* Halott halott marad, ha hrom 1
            szomszedja van, klnben 1 lesz. */
            if(elok==3)
                racsUtana[i][j] = SejtAblak::ELO;
            else
                racsUtana[i][j] = SejtAblak::HALOTT;
        }
    }
}
racsIndex = (racsIndex+1)%2;
}
```

A sikló-kilövő egy olyan alakzat, amelyben 2 ide-oda pattogó alakzat minden 15. ciklusban egy harmadikat produkál és "kilövi" azt.



7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

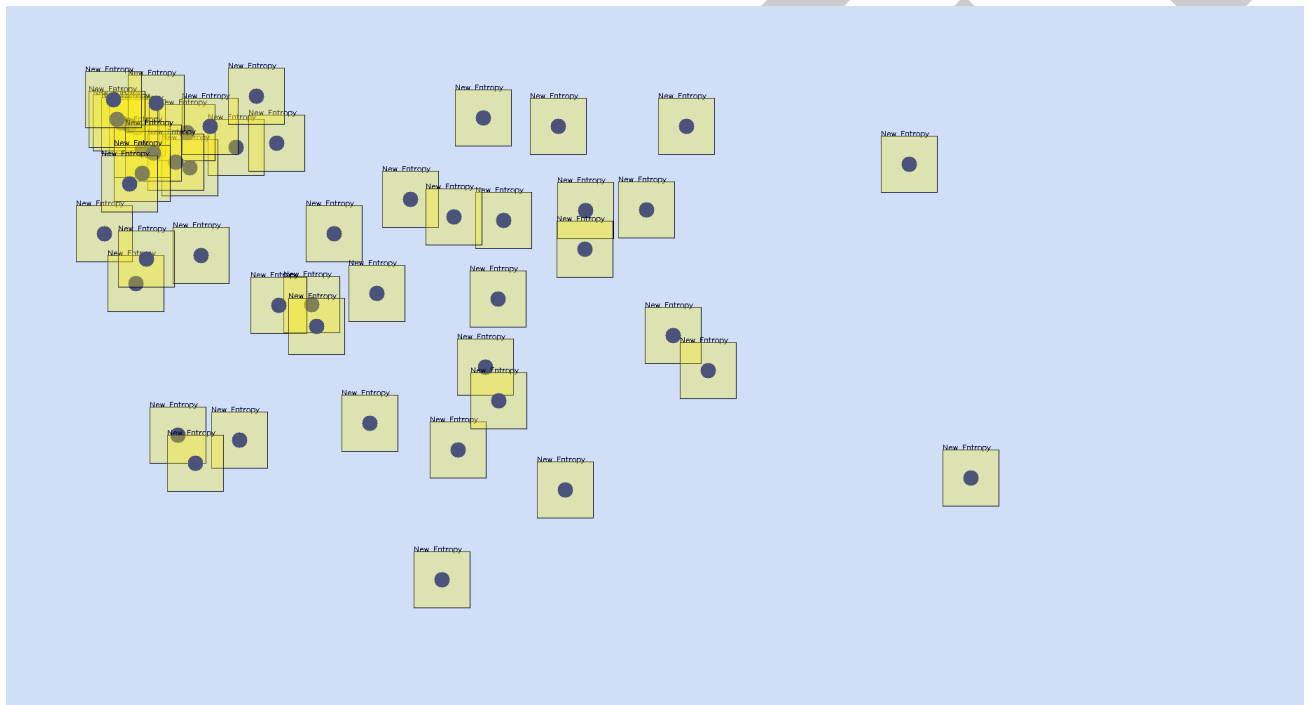
[bhax/thematic_tutorials/bhax_textbook/conway/brainb/main.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/brainb/BrainBWin.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/brainb/BrainBThread.cpp](#)

Ehhez a programhoz szükséges az opencv telepítése: **sudo apt-get install libopencv-dev**

Az esport kezd egyre inkább elterjedni. Ez a program az esportolóknak nyújt lehetőséget arra, hogy lemérjék teljesítményüket, pontosabban hogy mennyire képesek a zavaró tényezők mellett a kijelölt célra koncentrálni. Az méréshez 10 percig kell követni a kijelölt négyzetet lenyomott egérgombbal.



Ha végeztünk, akkor az eredményt kiírja nekünk egy fájlba.

```
NEMESPOR BrainB Test 6.0.3
time      : 6000
bps       : 59860
noc       : 49
nop       : 0
lost      :
2230 820 3380 2260 61370 34150 23480 4850 0 21450 38690 30700 17570 15690  ←
    9910 29510 20380 25880 42260 43430 54240 73360 49950 59940 55620
mean      : 28844
var       : 21808.5
found     : 0 12400 15360 18040 40460 32870 20320 38910 39440 37640 31740  ←
    48490 49680 44280 57730 33190 16150 11870 19730 7320 8540 9720 14310  ←
    31040 48770 23730 20870 0 0 5950 19370 12320 11440 14170 24210 12440  ←
    23500 18670 25330 15740 8890 29860 23810 45750 39770 37730 41060 24600  ←
```

```
22720 30870 27360 28220 42620 27790 36650 48690 39250 61970 51660 58200 ↔
54050 66020 61920 54830 64910 61410 45240 47300 49080
mean      : 31130
var       : 17635.2
lost2found: 0 49680 48770 0 0 12320 11440 15740 22720 48690 39250 54050 ↔
54830 45240
mean      : 28766
var       : 21827.6
found2lost: 61370 34150 23480 0 21450 30700 17570 29510 42260 43430 54240 ↔
73360 59940
mean      : 37804
var       : 20565
mean(lost2found) < mean(found2lost)
time      : 10:0
U R about 4.06311 Kilobytes
```

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. 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...

9.4. 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...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

DRAFT

10. fejezet

Helló, Olvasónapló!

10.1. Juhász István féle könyv

Egy számítógép programozására három nyelvi szintet különböztetünk meg:

- Gépi kód
- assembly szint
- Magas szint

Mink a magas szintű programozási nyelvekkel foglalkozunk. A magas szintű nyelven megírt forrásszöveg helyesírását szintaktikának, jelentését szemantikának nevezzük. Egy magas szintű programozási nyelvet ez a kettő határoz meg. A processzor csak a gépi kódot érti, ezért a gépi kód előállításához vagy fordítót vagy interpretert kell használnunk. A fordítóprogram a lexikális elemzés, szintaktikai elemzés, szemantikai elemzés és kódgenerálás lépések végrehajtásával állítja elő a forrásszövegből a tárgyprogramot. Az interpreteres módszer is elvégzi az első 3 lépést, viszont tárgyprogramot nem készít, ehelyett soronként értelmezi és végrehajtja az utasításokat. A programnyelv szabványát hivatkozási nyelvnek hívjuk, ahol a szintaktikai és szemantikai szabályok vannak részletezve. A programok írásához leggyakrabban integrált fejlesztői környezetet (IDE - Integrated Development Environment) használunk, amelyek nagyban megkönnyítik a nagyobb projektek kezelését. Ezek tartalmazzák a szövegszerkesztőt, fordítót / interpretert és futtató rendszert. Az imperatív nyelvek algoritmusokból állnak. Ezek lehetnek eljárásorientált vagy objektumorientált nyelvek. És vannak a deklaratív nyelvek. Ezek nem algoritmikusak és a programozó csak a problémát adja meg, a nyelvi implementációba van beépítve a megoldás módja.

10.2. K & R, A C programozási nyelv

A változók és az állandók a programok alapvető részei. A deklarációjuk során megadjuk a nevüket, típusukat és esetleg a kezdeti értéküket. A nevekre nézve vannak némi megkötések, pl.: nem tartalmazhat pontot, nem lehetnek kulcsszavak (mint if, else, int), stb.

A c-ben csak néhány alapvető adattípus van:

- int - egész szám

- float - egyszeres pontosságú lebegőpontos szám
- double - kétszeres pontosságú lebegőpontos szám

Ezt kiegészíthetjük a short, long és unsigned minősítő szimbólumokkal. A short és a long a méretét szabályozza. Az unsigned(előjel nélküli) számokra a modulo 2^n aritmetika szabályai vonatkoznak, ahol n az int típust ábrázoló bit-ek száma; az unsigned számok mindig pozitívak.

```
short int x;  
long int y;  
unsigned int z;
```

10.3.

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.