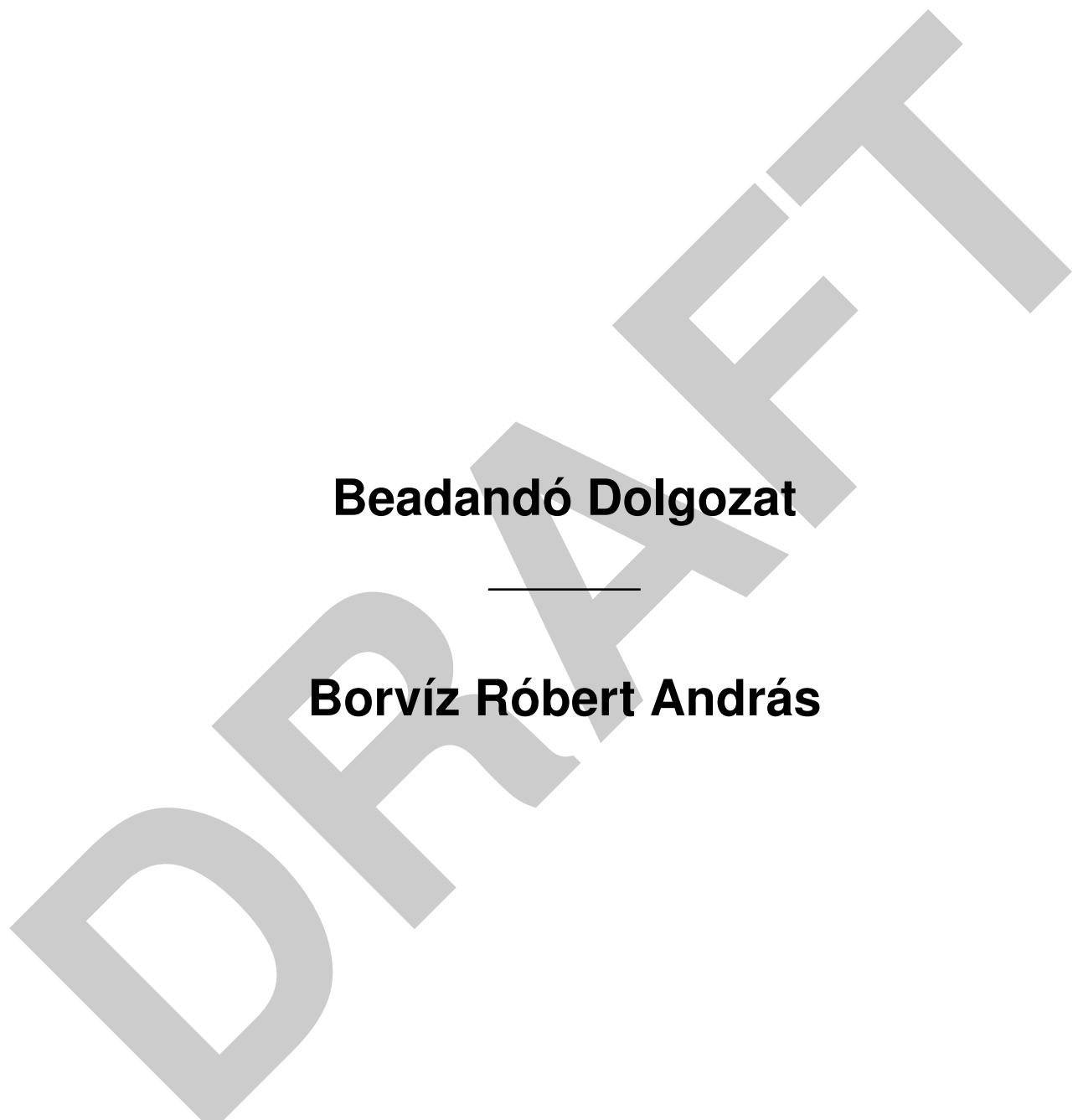


Beadandó Dolgozat

Borvíz Róbert András



Copyright © 2019 Borvíz Róbert András

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>

DRAFT

COLLABORATORS

	<i>TITLE :</i>		
	Beadandó Dolgozat		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Borvíz, Róbert András	2019. szeptember 15.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Iniciális 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	2019-02-20	Feladatlisták átvétele, a BHAX repójából. Feladtok kidolgozásának megkezdése.	brobert
0.0.5	2019-03-12	Az eddig elvégzett munka feltöltése a github repómra: https://github.com/BorvizRobi/prog_1_textbook	brobert
0.0.6	2019-03-13	Folyamatos munka a feladatakon, github repóm folyamatos frissítése.	brobert

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0.0	2019-05-1	Feladatok befejezése, ellenőrzése, utolsó simítások elvégzése. Végeleges változat feltöltése.	brobert
1.1.0	2019-09-09	Munka megkezdése a második fejezeten.	brobert

DRAFT

Tartalomjegyzék

I. Tematikus feladatok	1
1. Helló, Turing!	2
1.1. Végtelen ciklus	2
1.2. Lefagyott, nem fagyott, akkor most mi van?	5
1.3. Változók értékének felcserélése	7
1.4. Labdapattogás	8
1.5. Szóhossz és a Linus Torvalds féle BogoMIPS	11
1.6. Helló, Google!	12
1.7. 100 éves a Brun téTEL	16
1.8. A Monty Hall probléma	18
2. Helló, Chomsky!	22
2.1. Decimálisból unárisba átváltó Turing gép	22
2.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	25
2.3. Hivatalos nyelv	26
2.4. Saját lexikális elemző	31
2.5. I33t.l	32
2.6. A források olvasása	36
2.7. Logikus	37
2.8. Deklaráció	38
3. Helló, Caesar!	41
3.1. double ** háromszögmátrix	41
3.2. C EXOR titkosító	46
3.3. Java EXOR titkosító	48

3.4. C EXOR törő	51
3.5. Neurális OR, AND és EXOR kapu	56
3.6. Hiba-visszaterjesztéses perceptron	64
4. Helló, Mandelbrot!	70
4.1. A Mandelbrot halmaz	70
4.2. A Mandelbrot halmaz a std::complex osztálytal	77
4.3. Biomorfok	81
4.4. A Mandelbrot halmaz CUDA megvalósítása	87
4.5. Mandelbrot nagyító és utazó C++ nyelven	90
4.6. Mandelbrot nagyító és utazó Java nyelven (Passz)	103
5. Helló, Welch!	104
5.1. Első osztályom	104
5.2. LZW	109
5.3. Fabejárás	116
5.4. Tag a gyökér	120
5.5. Mutató a gyökér	135
5.6. Mozgató szemantika	136
6. Helló, Conway!	138
6.1. Hangyaszimulációk	138
6.2. Java életjáték (Passz)	157
6.3. Qt C++ életjáték (Passz)	166
6.4. BrainB Benchmark (Passz)	173
7. Helló, Schwarzenegger!	193
7.1. Szoftmax Py MNIST (Passz)	193
7.2. Mély MNIST (Passz)	193
7.3. Minecroft-MALMÖ (Passz)	193
8. Helló, Chaitin!	194
8.1. Iteratív és rekurzív faktoriális Lisp-ben	194
8.2. Gimp Scheme Script-fu: név mandala	199
8.3. Gimp Scheme Script-fu: króm effekt	209

9. Helló, Gutenberg!	218
9.1. Juhász István: Magas szintű programozási nyelvek 1	218
9.2. Benedek Zoltán: Szoftverfejlesztés C++ nyelven	220
9.3. Brian W. Kernighan, Dennis M. Ritchie: The C programming language	223
10. Összegzés:	224
10.1. Passzolt feladatok:	224
10.2. Tutoráltaim:	224
10.3. Tutorok:	225
II. Második felvonás	226
11. Helló, Berners-Lee!	227
11.1. C++: Benedek Zoltán, Levendovszky Tíhamér Szoftverfejlesztés C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.	227
11.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)	229
12. Helló, Arroway!	230
12.1. OO szemlélet	230
12.2. Homokatózó	237
12.3. „Gagyí”	248
12.4. Yoda	249
12.5. Kódolás from scratch	252
13. Helló, Liskov!	253
13.1. Liskov helyettesítés sértése	253
13.2. Szülő-gyerek	253
13.3. Anti OO	253
13.4. Hello, Android!	253
13.5. Ciklomatikus komplexitás	254

Ábrák jegyzéke

1.1. Egy mag 100%-ban	3
1.2. Egy mag 0%-ban	4
1.3. Összes mag 100%-ban	5
1.4. A "H" program	6
1.5. A "H+" program	7
1.6. Fűrészfog	10
1.7. Rangsorolandó honlapok kapcsolatai.	13
1.8. Pagerank értékek.	16
1.9. Brun téTEL	18
1.10. Monty	19
2.1. Turing gép felépítése.	23
2.2. Állapotátmenet gráfja	24
3.1. Háromszögmátrix	42
3.2. ** háromszögmátrix a memóriában	44
3.3. háromszögmátrix	46
3.4. Neuron	57
3.5. NN	60
3.6. NN	61
3.7. NN	62
3.8. NN	64
3.9. Mandel	66
3.10. A visszaadott eredmények.	69
4.1. Mandel	76
4.2. Mandel	81

4.3. Julia halmaz	82
4.4. Biomorf	86
4.5. Mandelnagyító	101
4.6. Mandelnagyító	102
4.7. Mandelnagyító	103
5.1. polargen.cpp	108
5.2. z.c	116
5.3. Preorder	118
5.4. Postorder	120
6.1. UML osztálydiagram	139
6.2. Hangya szimuláció	157
6.3. sikló-kilövő	166
6.4. Sikló-kilövő	173
6.5. BrainB	192
8.1. Lisp	195
8.2. Lisp Rekurzív faktoriális	197
8.3. Lisp iteratív faktoriális	198
8.4. Gimp szkript	200
8.5. Gimp mandala	201
8.6. Mandala	209
8.7. Chrome ablak	210
8.8. Bátfai Haxor felirat	211
10.1. +2 passz	224
12.1.	233
12.2. polargen.cpp	237
12.3.	252

I. rész

Tematikus feladatok

DRAFT

1. fejezet

Helló, Turing!

1.1. Végtelen ciklus

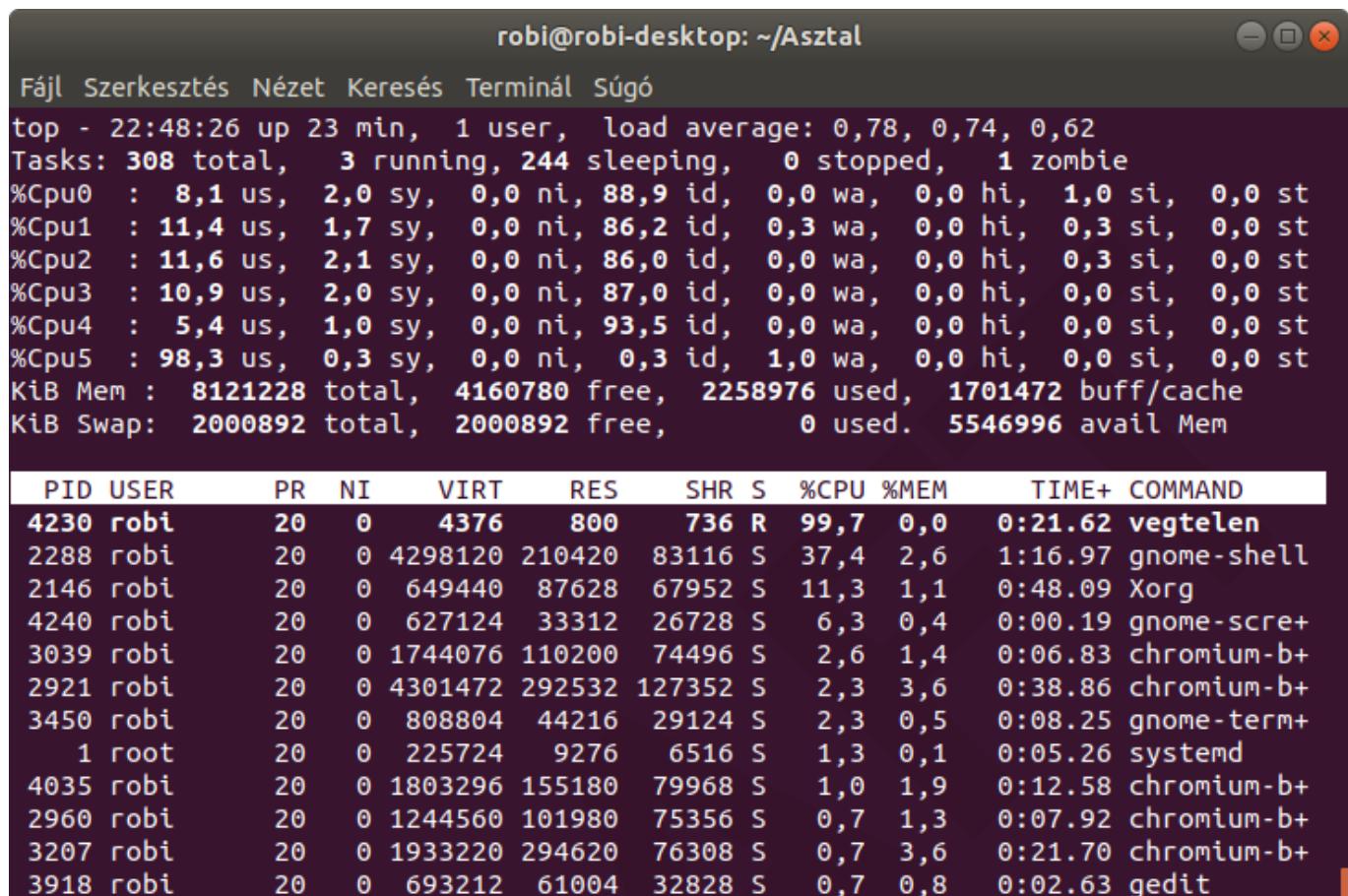
Írj olyan C végtelen ciklusokat, amelyek 0 illetve 100 százalékban dolgoztatnak egy magot és egy olyat, amely 100 százalékban minden magot!

Megoldás forrása: https://github.com/BorvizRobi/Prog_1_vegtelenciklus

Egy mag 100%-ban való dolgoztatásához nincs más dolgunk, mint írni egy végtelen ciklust és elindítani:

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

Parancssorban a top parancs kiadásával, majd az 1-es gomb megnyomásával megtekinthetjük az eredményt:



The terminal window title is "robi@robi-desktop: ~/Asztal". The output shows system statistics and a detailed process list.

```

Fájl Szerkesztés Nézet Keresés Terminál Súgó
top - 22:48:26 up 23 min, 1 user, load average: 0,78, 0,74, 0,62
Tasks: 308 total, 3 running, 244 sleeping, 0 stopped, 1 zombie
%Cpu0 : 8,1 us, 2,0 sy, 0,0 ni, 88,9 id, 0,0 wa, 0,0 hi, 1,0 si, 0,0 st
%Cpu1 : 11,4 us, 1,7 sy, 0,0 ni, 86,2 id, 0,3 wa, 0,0 hi, 0,3 si, 0,0 st
%Cpu2 : 11,6 us, 2,1 sy, 0,0 ni, 86,0 id, 0,0 wa, 0,0 hi, 0,3 si, 0,0 st
%Cpu3 : 10,9 us, 2,0 sy, 0,0 ni, 87,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 : 5,4 us, 1,0 sy, 0,0 ni, 93,5 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 : 98,3 us, 0,3 sy, 0,0 ni, 0,3 id, 1,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 8121228 total, 4160780 free, 2258976 used, 1701472 buff/cache
KiB Swap: 2000892 total, 2000892 free, 0 used. 5546996 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
4230 robi 20 0 4376 800 736 R 99,7 0,0 0:21.62 vegetelen
2288 robi 20 0 4298120 210420 83116 S 37,4 2,6 1:16.97 gnome-shell
2146 robi 20 0 649440 87628 67952 S 11,3 1,1 0:48.09 Xorg
4240 robi 20 0 627124 33312 26728 S 6,3 0,4 0:00.19 gnome-scre+
3039 robi 20 0 1744076 110200 74496 S 2,6 1,4 0:06.83 chromium-b+
2921 robi 20 0 4301472 292532 127352 S 2,3 3,6 0:38.86 chromium-b+
3450 robi 20 0 808804 44216 29124 S 2,3 0,5 0:08.25 gnome-term+
1 root 20 0 225724 9276 6516 S 1,3 0,1 0:05.26 systemd
4035 robi 20 0 1803296 155180 79968 S 1,0 1,9 0:12.58 chromium-b+
2960 robi 20 0 1244560 101980 75356 S 0,7 1,3 0:07.92 chromium-b+
3207 robi 20 0 1933220 294620 76308 S 0,7 3,6 0:21.70 chromium-b+
3918 robi 20 0 693212 61004 32828 S 0,7 0,8 0:02.63 gedit

```

1.1. ábra. Egy mag 100%-ban

Egy mag 0%-ban való dolgoztatásánál már más a helyzet itt segítségül hívjuk a sleep() funkciót ami felfüggeszti a folyamat végrehajtását annyi másodpercre amekkora számot argumentumként megadunk, mivel itt egy végtelen for ciklusba tettük be ezért a programunk a for ciklus minden egyes végrehajtásakor várakozásra van ítélezve ezzel nullára redukáljuk a cpu használatot.

```
#include <unistd.h>
int main(){

    for(;;)
        sleep(1000);
}
```

Eredmény:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
top - 22:49:53 up 24 min, 1 user, load average: 0,29, 0,60, 0,58
Tasks: 311 total, 1 running, 244 sleeping, 0 stopped, 1 zombie
%Cpu0 : 1,0 us, 0,3 sy, 0,0 ni, 98,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 0,3 us, 0,0 sy, 0,0 ni, 99,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 : 0,7 us, 0,3 sy, 0,0 ni, 99,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 0,3 us, 0,7 sy, 0,0 ni, 99,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 : 0,3 us, 0,0 sy, 0,0 ni, 99,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 : 1,7 us, 0,7 sy, 0,0 ni, 97,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 8121228 total, 4160136 free, 2257892 used, 1703200 buff/cache
KiB Swap: 2000892 total, 2000892 free, 0 used. 5548016 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM TIME+ COMMAND
2288 robi      20   0 4298132 211536 83116 S  4,0  2,6 1:23.64 gnome-shell
2146 robi      20   0 648968  87624 67948 S  3,0  1,1 0:52.51 Xorg
3039 robi      20   0 1744076 109792 74496 S  1,3  1,4 0:06.87 chromium-b+
2921 robi      20   0 4293276 292452 127292 S  0,7  3,6 0:39.28 chromium-b+
3450 robi      20   0 808804  44236 29124 S  0,7  0,5 0:09.91 gnome-term+
4285 robi      20   0 52876  4088  3400 R  0,7  0,1 0:00.41 top
  1 root       20   0 225724   9276  6516 S  0,3  0,1 0:05.71 systemd
3207 robi      20   0 2078764 296716 76308 S  0,3  3,7 0:22.17 chromium-b+
3918 robi      20   0 693212  61028 32852 S  0,3  0,8 0:03.47 gedit
4051 robi      20   0 1762248 135920 71148 S  0,3  1,7 0:05.40 chromium-b+
  2 root       20   0      0      0      0 S  0,0  0,0 0:00.00 kthreadd
  3 root       20   0      0      0      0 I  0,0  0,0 0:00.59 kworker/0:0
```

1.2. ábra. Egy mag 0%-ban

Az összes mag 100%-on való dolgoztatásához így kell fordítanunk a programunkat:

gcc vegtelen.c -o vegtelen -fopenmp

Az **-fopenmp** kapcsolóra van szükség az OpenMP eszközeinek használatához, a kapcsoló nélkül a fordító figyelmen kívül hagyja az OpenMP pragmáit.

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

A **#pragma omp parallel** sor új szálakat forkol, amik tovább viszik a sor után következő utasítást, itt a kapcsos zárójelek közötti részt, vagyis a végtelen ciklust, így elérve az összes magunk 100%-ban való dolgoztatását.

Eredmény:

```
robi@robi-desktop: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
top - 23:01:08 up 35 min, 1 user, load average: 1,82, 1,46, 1,07
Tasks: 318 total, 3 running, 257 sleeping, 0 stopped, 1 zombie
%Cpu0 : 99,3 us, 0,7 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 : 99,3 us, 0,7 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 98,3 us, 1,7 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 : 96,0 us, 4,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 : 99,0 us, 1,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 8121228 total, 3430228 free, 2857276 used, 1833724 buff/cache
KiB Swap: 2000892 total, 2000892 free, 0 used. 4881204 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
4858 robi 20 0 51968 968 872 R 560,3 0,0 1:31.91 vegetelen
2288 robi 20 0 4299276 212008 83116 S 18,9 2,6 2:09.02 gnome-shell
2146 robi 20 0 650140 88032 68356 S 8,9 1,1 1:23.37 Xorg
2770 robi 20 0 1072116 147604 46880 S 4,3 1,8 0:23.69 evince
4870 robi 20 0 627152 33360 26748 S 4,3 0,4 0:00.20 gnome-scre+
3450 robi 20 0 808804 44272 29124 S 0,7 0,5 0:13.49 gnome-term+
4864 robi 20 0 52876 4116 3428 R 0,7 0,1 0:00.06 top
1 root 20 0 225724 9276 6516 S 0,3 0,1 0:08.67 systemd
333 root 0 -20 0 0 0 I 0,3 0,0 0:00.30 kworker/4:+
2921 robi 20 0 4417692 316772 127916 S 0,3 3,9 1:14.76 chromium-b+
3207 robi 20 0 2140084 324452 80544 S 0,3 4,0 0:29.03 chromium-b+
3287 robi 20 0 1832108 186188 82172 S 0,3 2,3 0:09.73 chromium-b+
```

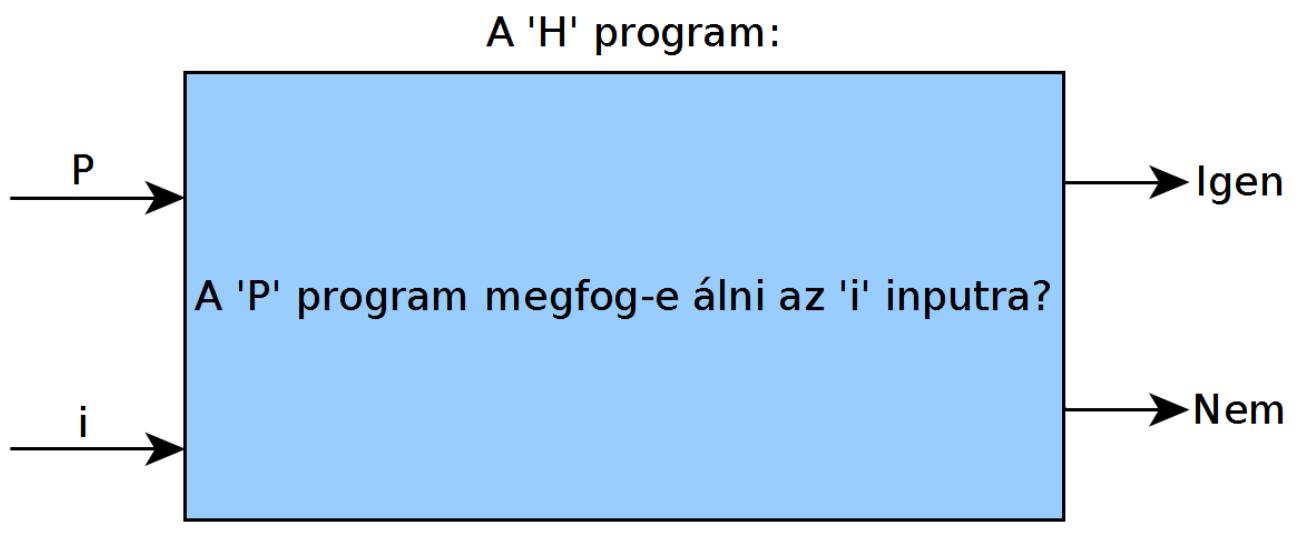
1.3. ábra. Összes mag 100%-ban

1.2. 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(van-e benne végtelen ciklus) vagy sem!

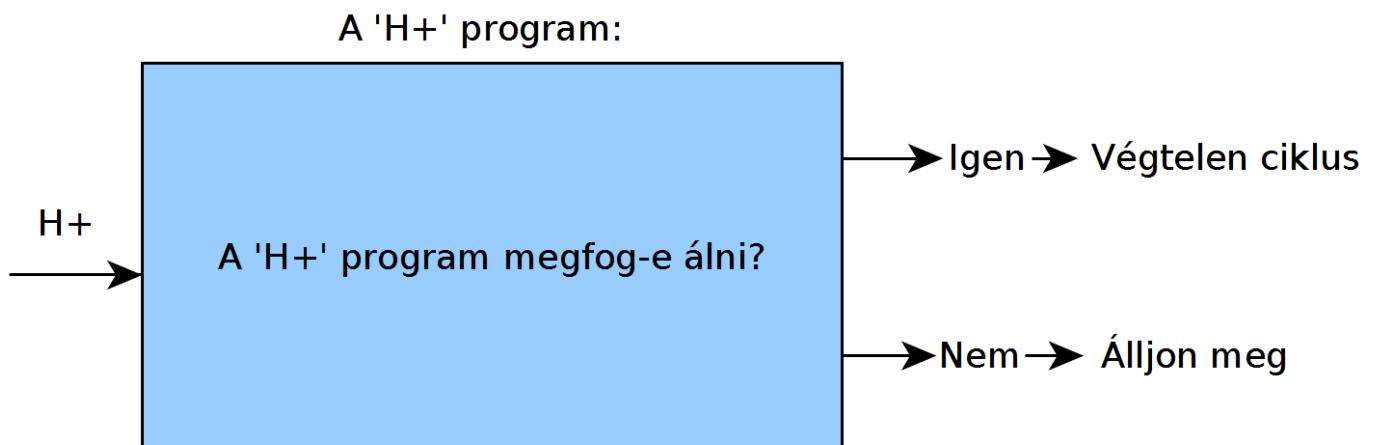
A feladat megoldásához nem szükséges konkrét programkódot írni, feltételezéssel megoldható a feladat:

Tegyük fel, hogy tudunk olyan programot létrehozni, ami eldönti egy másik programról, hogy megfog-e állni vagy sem, nevezzük ezt a programot "H"-nak. "H" bemenetének egy programot adunk meg és "H" eldönti, hogy ez a program lefagy(végtelen ciklusba lép) vagy nem egy adott inputra:



1.4. ábra. A "H" program

Most alakítsuk át ezt a programot úgy hogy ha igent mond, akkor fusson örökké, ha nemet akkor meg álljon meg. Nevezük ezt az új gépet "H+"-nak. Adjuk "H+" bemenetének saját magát, így ha "H+" igen választ ad akkor örökké fog futni, dehát így nem áll meg. Ha viszont nem áll meg, akkor nem választ kapunk, de akkor megáll. Ezzel ellentmondásba ütköztünk tehát ilyen program nem létezhet.



1.5. ábra. A "H+" program

1.3. Változók értékének felcserélése

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

Megoldás forrása: <https://github.com/BorvizRobi/valtozocsere-segedvaltozonelkul>

Több megoldás is létezik, fel lehet cserélni két változó értékét segédváltozó nélkül különbséggel:

```
#include <iostream>

int main()
{
    int a=8;
    int b=11;

    a=a-b; //vesszük a két szám különbségét
    b=b+a; // "b"-hez hozzáadjuk ezt a különbséget így "a"-t kapjuk
    a=b-a; // "b" jelenlegi értékéből kivonjuk ezt a különbséget
            //így "a" egyenlő lesz eredeti "b"-vel

    std::cout<<"a= "<<a<<" b= "<<b<<std::endl;

}
```

Lehet őket cserélni szorzattal:

```
#include <iostream>

int main()
{
    int a=8;
    int b=11;

    a=a*b; //vesszük a két szám szorzatát
    b=a/b; //ezt a szorzatot elosztjuk b vel így b=a
    a=a/b; //a szorzatot elosztjuk "a" értékével ami most "b"

    std::cout<<"a= "<<a<<" b= "<<b<<std::endl;

}
```

Lehet őket cserélni EXOR-al:

```
#include <iostream>

int main()
{
    int a=8;
    int b=11;

    a=a^b; //XOR művelet az "a" és "b" bitjein
    b=a^b; //XOR az így kapott "a" bitjein "b"-vel,
            // "b" egyenlő lesz eredeti a-val
    a=a^b; //XOR az első "a" XOR "b" eredményén b vel így a egyenlő lesz b-val

    std::cout<<"a= "<<a<<" b= "<<b<<std::endl;

}
```

1.4. Labdapattogás

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

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

Megoldás forrása: <https://github.com/BorvizRobi/labdapatogtatas>

-Incurses parancs szükséges a fordításhoz, ezzel adjuk hozzá a könyvtárat ami a képernyőkezeléshez kell.

Akkor nézzük a forráskódot, itt kommenteken keresztül fogom elmagyarázni a program működését:

Labdapatogtatás if-el:

```
#include <curses.h> //itt találhatóak képernyőkezeléshez szükséges eszközök
#include <unistd.h> // usleep() funkció használatához

int main ( void )
{
    WINDOW *ablak;      //WINDOW tipusú pointer ebben tároljuk a képernyő adatait
    ablak = initscr (); // hozzárendeljük a képernyő adatait WINDOW pointerünkhöz.
    int x = 0; //a kezdő x és y értékek ahonnan a labda fog indulni.
    int y = 0;

    int xnov = 1; //lépésközöket tároljuk
    int ynov = 1;

    int mx; //hány sorból és oszlopból áll az ablakunk.
    int my;

    for ( ; ) { //végtelen ciklus hogy folyamatosan fusson

        getmaxyx ( ablak, my , mx ); // lekérdezzük hogy hány sorból és oszlopból áll az ablak.

        mvprintw ( y, x, "O" ); //mvprintw aktuális képernyőre ír x és y pozícióba

        refresh (); //frissítés bufferben lévő karakter megjelenítésére

        usleep ( 50000 ); //felfüggeszti a program futását miliszekundumokban mérve

        clear(); //törli a képernyőt ha nem teszük be akkor az összes eddig kiírt labdát látni fogjuk

        x = x + xnov; //x és y érékét növeljük
        y = y + ynov;

        if ( x>=mx-1 ) { // elérte-e a jobb oldalt?
            xnov = xnov * -1;
        }
        if ( x<=0 ) { // elérte-e a bal oldalt?
            xnov = xnov * -1;
        }
    }
}
```

```
        }
        if ( y<=0 ) { // elérte-e a tetejet?
            ynov = ynov * -1;
        }
        if ( y>=my-1 ) { // elérte-e a aljat?
            ynov = ynov * -1;
        }

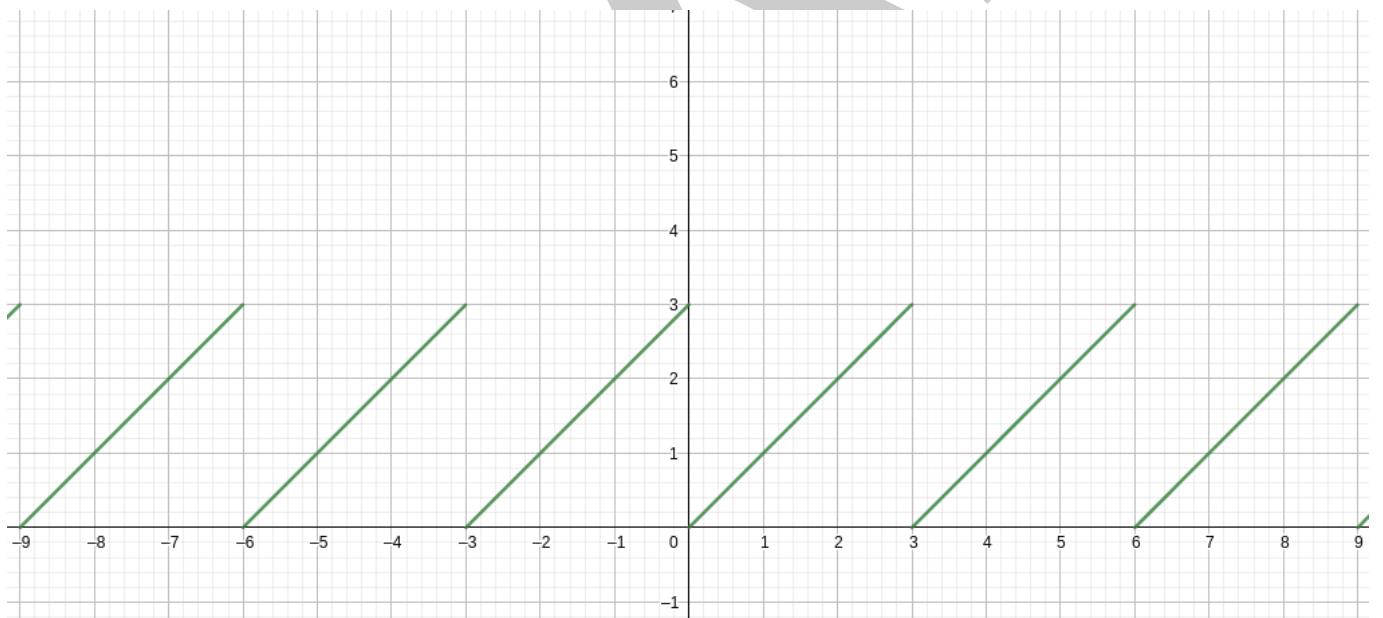
    }

    return 0;
}
```

Labdapatogtatás if nélkül:

A feladat megoldható a maradékos osztás művelet használatával(%) hiszen az előző példából tudhatjuk, hogy a labda mozgatása során például az oszlop kordinátája így változik 1,2,3,4,5,4,3,2,1,2... tehát folyamatosan hullámzik a határértékig és vissza, ez a maradékos osztással is elérhető de mivel az, az elért érték után rögtön nullára esik vissza: 1,2,3,4,5,0,1,2,3,4,5,0...(Grafikonjának fűrészfog alakja van) ezért 2 sorozat összesítésére van szükségünk.

A maradékos osztás grafikonjának "fűrészfog" alakja:



1.6. ábra. Fűrészfog

Nézzük a kódot:

```
#include <curses.h> //itt találhatóak képernyőkezeléshez szükséges eszközök
#include <unistd.h> // usleep() funkció használatához
```

```
#include <stdlib.h> // abs() funkció használatához

int main ( void )
{

    WINDOW *ablak;      //WINDOW tipusú pointer ebben tároljuk a képernyő ←
                        //adatait
    ablak = initscr (); // hozzárendeljük a képernyő adatait WINDOW ←
                        //pointerünkhöz.

    int mx;    //itt tároljuk hogy hány sorból és oszlopból áll az ablakunk.
    int my;

    int xj = 0, xk = 0, yj = 0, yk = 0; //százalékos osztás eredményének ←
                                         //tárolására.

    for ( ; ) { //végtelen ciklus hogy folyamatosan fusson

        getmaxyx ( ablak, my , mx ); // lekérdezzük hogy hány sorból és ←
                                         //oszlopból áll az ablak.

        //elvégezzük a %-os osztásokat
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;

        yj = (yj - 1) % my;
        yk = (yk + 1) % my;

        //mvprintw aktuális képernyőre a megadott pozícióba ír.
        mvprintw ( abs (yj + (my - yk) ), abs (xj + (mx - xk)) , "O" );

        refresh (); //frissítés bufferben lévő karakter megjelenítésére
        usleep ( 50000 ); //felfüggeszti a program futását ←
                           //miliszekundumokban mérve

        clear(); //törli a képernyőt ha nem tesszük be akkor az összes eddig ←
                 //kiírt labdát látni fogjuk.
    }
    return 0;
}
```

1.5. 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 forrása: <https://github.com/BorvizRobi/BevProg/blob/master/szohosszshift.c>

cpp

A következő részletet ezen a címen találtam, itt jól össze van foglalva hogy mi az a BogoMIPS:

<https://www.szabilinux.hu/forditasok/BogoMips/BogoMips-2.html>

"A MIPS a Millió Utasítás Másodpercenként rövidítése. Ez a számítógép számítási sebességének mértékegsége. Ezzel gyakrabban élnek vissza, mint amennyiszer helyesen használják, de ez a legtöbb ilyen mértékegséggel így van. Nagyon nehéz igazságosan összemérni különböző számítógépek MIPS értékeit.

A BogoMIPS-et Linus találta ki. A kernelnek (vagy egy eszközmeghajtónak?) szüksége van egy időzítő ciklusra (egy rövid de pontos időtartamra, amennyit várakozik bizonyos helyzetekben), amelyet a processzor sebességéből kell kiszámítani. Éppen ezért a kernel a rendszerinduláskor megméri, hogy egy bizonyos ciklus mennyi idő alatt fut le az adott számítógépen. A ``Bogo'' szó az angol ``bogus'' szóból ered, ami az jelenti: hamis, nem igazi. Ezért a BogoMIPS érték következtetni enged a processzor sebességére, de annyira áltudományos, hogy csak a BogoMIPS kifejezés illik rá.

Két oka van annak, hogy ez az érték megjelenik a képernyőn a rendszerinduláskor: a) elég jó hibakereséshez és annak ellenőrzésére, hogy a számítógépben be van-e kapcsolva a cache és a turbó gomb; b) Linus szeret kuncogni a hírcsoportokba írkáló megzavarodott embereken."

Linus Torvalds a BogoMIPS rutinjában a következő operátort használta:

```
<<=
```

Ez a bitshift operátor, a megadott változó memóriában tárolt bitjein végez műveletet, eltolja azokat annyival (itt balra) amekkora értéket megadunk neki.

Program egy gépi szó hosszának mérésére:

```
#include <stdio.h>

int main()
{
    int szo=1; //szó amit mérni fogunk itt az int 1-es szám.
    int lepes=1; //számláló, egyről indítjuk mert az első bitről indulunk.

    while(szo<<=1) //ameddig eltudjuk tolni az első bitet addig megismétlődik a ←
        ciklus.
    ++lepes; //növeljük 1 el a lepes-t itt tárolva hogy hányszor mentünk ←
        balra.

    printf("%i\n",lepes); //végül kiíratjuk az eredményt.
}
```

A képernyőn megjelenő szám a 32, tehát ennyi bitből áll egy gépi szó.

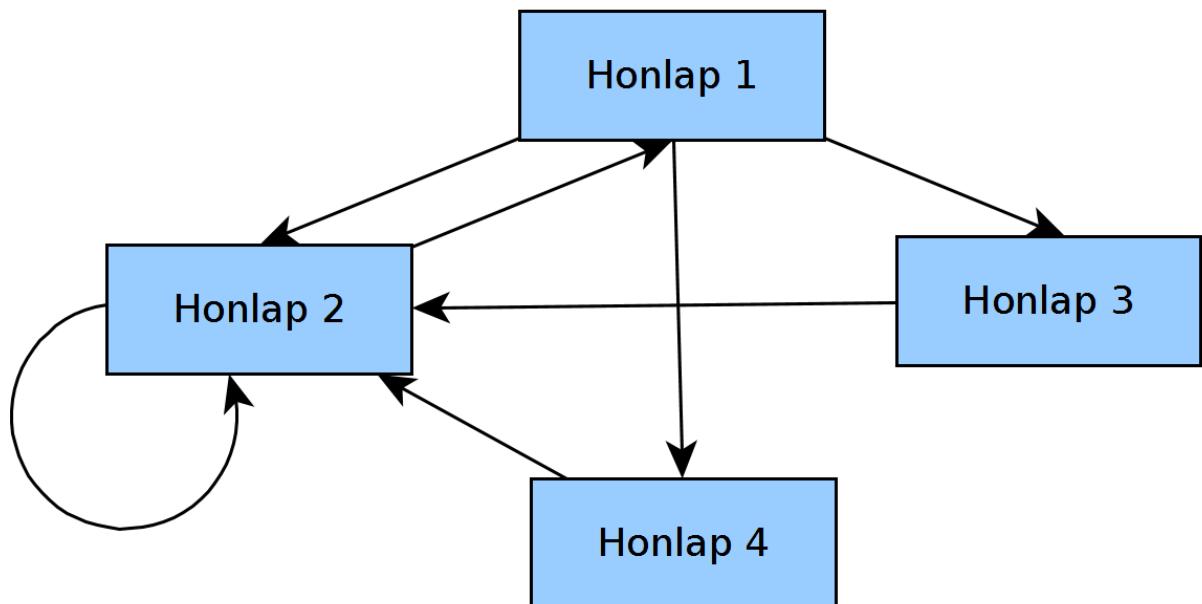
1.6. Helló, Google!

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

Megoldás forrása: <https://github.com/BorvizRobi/BevProg/blob/master/pagerank.cpp>

A PageRank egy módszer weblapok osztályozására, az alapötlet: azok a lapok jobb minőségűek, amelyekre jobb minőségű lapok mutatnak, ezzel az algoritmussal indult anno a Google is.

A Program által kiszámolt honlapok kapcsolata:



1.7. ábra. Rangsorolandó honlapok kapcsolatai.

Forráskód:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.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 +=abs(pagerank[i] - pagerank_temp[i]); 

return 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};

int i, j;

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

    if ( tavolsag(PR,PRv, 4) < 0.0000001)
        break;

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

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

Elemezzük részről részre:

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

```
}
```

kiir funkció, kapott PageRank értékek kiírására használjuk.

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

tavolsag funkció, kiugrásra használjuk a végtelen for ciklusból. A PageRank algoritmus miután megadtuk a lapok kezdő pagerank értékét itt ez 1/4 (mivel 4 lapunk van), minden egyes iterációval egyre pontosabb pagerank értéket számol ki.

```
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}
};
```

Link mátrix ebben rögzítjük az oldalak között fenálló kapcsolatot.

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

Itt a kezdő PageRank értékeket adjuk meg.

```
int i, j;

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

    if ( tavolsag(PR,PRv, 4) < 0.0000001)
        break;

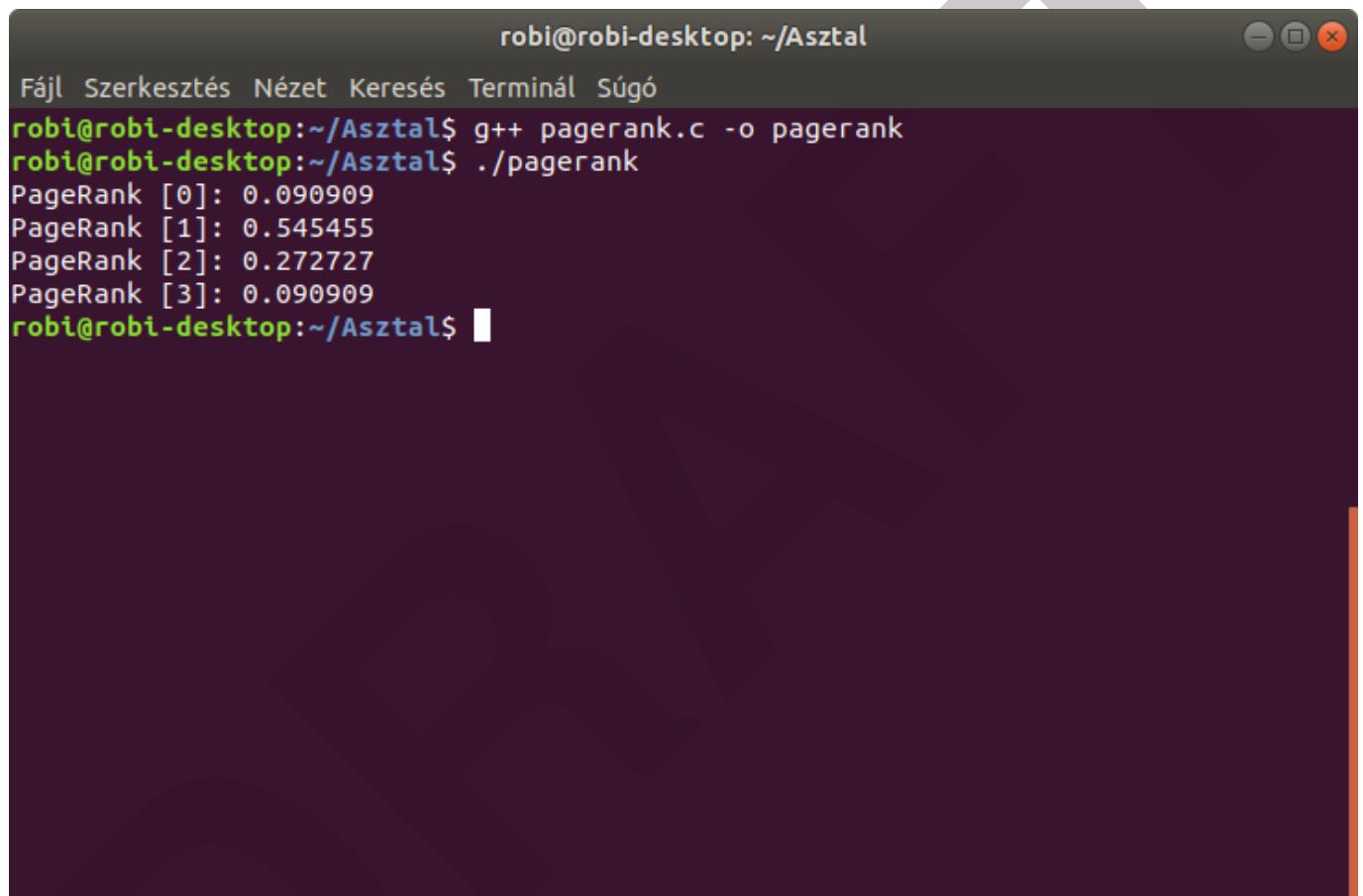
    for(i=0;i<4;i++)
        PRv[i] = PR[i];

    kiir (PR,4);
}
```

```
    return 0;  
}
```

PageRank értékek kiszámolása végtelen for ciklusban mátrix-szorzással,a végtelen ciklus minden egyes lefutásával egyre jobb közelítőértéket kapunk, ha elérünk a kívánt pontosságot, megszakítjuk a ciklust majd kiíratjuk az eredményt.

A program a következő értékeket számolja ki:



The terminal window shows the following session:

```
robi@robi-desktop:~/Asztal$ g++ pagerank.c -o pagerank
robi@robi-desktop:~/Asztal$ ./pagerank
PageRank [0]: 0.090909
PageRank [1]: 0.545455
PageRank [2]: 0.272727
PageRank [3]: 0.090909
robi@robi-desktop:~/Asztal$
```

1.8. ábra. Pagerank értékek.

1.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/Brunn>

A téTEL megértéséhez tudnunk kell hogy mik azok az ikerprímek:

Ikerprímeknek nevezzük azokat a prímpárokat amelyeknek a különbsége kettő.

Brun tétele azt mondja ki, hogy az ikerprímek reciprokokösszege a Brun konstansnak nevezett véges értékhez konvergál.

A következő program közelíteni próbálja a Brun konstans értékét:

```
library(matlab)

stp <- function(x) {

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

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

Nézzük meg soronként:

```
primes = primes(x)
```

Megadja a prímszámokat a megadott értéig.

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

Az egymás után következő prímek különbségét képzi és eltárolja őket a diff-be.

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

Ha két egymást követő prím különbsége kettő akkor ikerprímek lesznek, ezt ellenőrzi ez a rész és eltárolja a helyzetüket.

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

A primes tömből kiveszi az idx helyen lévő elemeket és tárolja t1primes-ben.

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

Az ikerprímek második tagjait megkapjuk úgy, hogy az elsőkhöz hozzáadunk 2-öt, t2primes tömbben tároljuk őket.

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

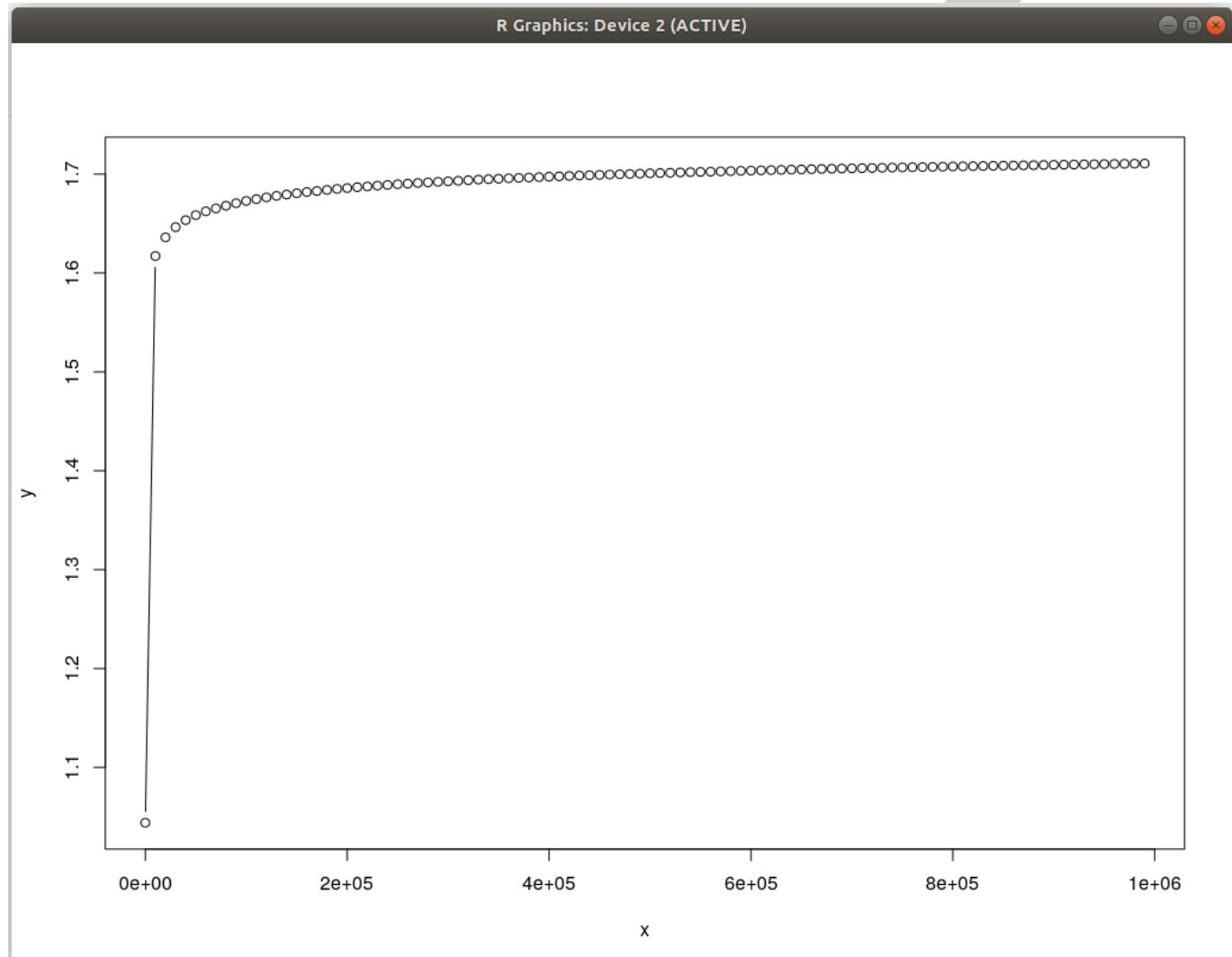
Az ikerprímek minden tagjának vesszük a reciprokértékét és összeadjuk őket.

```
return(sum(rt1plust2))
```

Összeadjuk az összegeket.

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

Ábrázoljuk: 10-től indulunk 1000000-ig, 10000-es lépésközel és minden egyes ilyen x-re kiszámoltatjuk az értékét és ábrázoljuk őket.



1.9. ábra. Brun téTEL

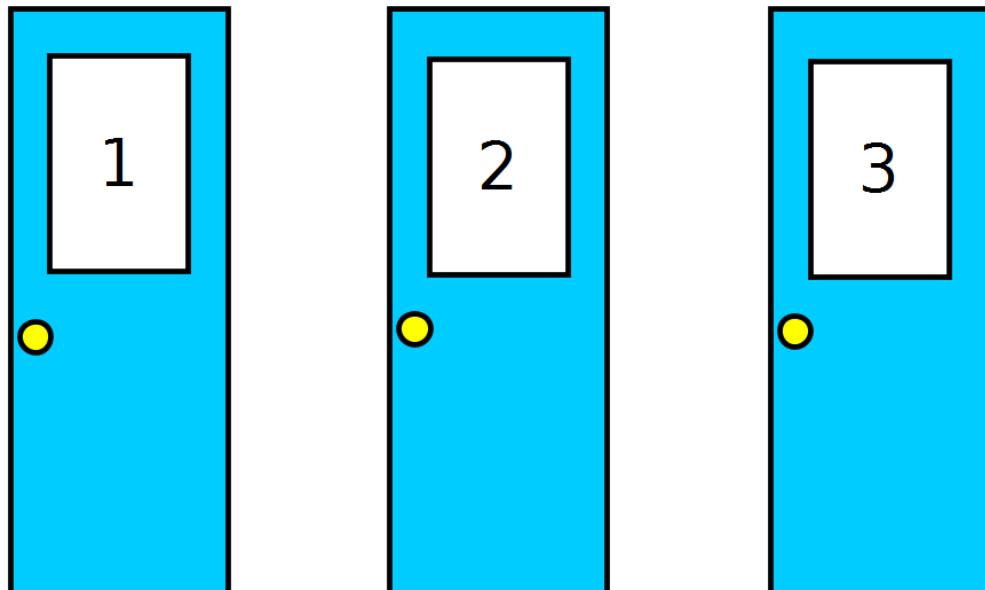
1.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/montyhall>

Monty Hall probléma:

Játékosok vagyunk egy televíziós műsorban, három ajtó közül választhatunk, az egyikben értékes nyeremény van a másik kettőben semmi, miután kiválasztottuk az ajtónkat a műsorvezető kinyitja a másik két ajtó közül az egyiket (amelyikben nincs nyeremény), majd felteszi a kérdést, hogy szeretnénk-e változtatni? Az emberek többsége ekkor úgy véli hogy 50%-50% esélyünk van a nyerésre, azonban ez még sincs így ugyanis ha a játékos megváltoztatja a döntését akkor 2/3 esélye van a győzelemre.



1.10. ábra. Monty

Teljes forráskód:

```
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)
```

Nézzük soronként:

```
kiserletek_szama=10000000
```

Itt adjuk meg hogy hány kísérletet akarunk futtatni.

```
kiserlet = sample(1:3, kiserletek_szama, replace=T)
```

Itt tároljuk, hogy hol van a nyeremény, 3 ajtó közül csak az egyikben van értékes nyeremény, tehát 1-től 3-ig véletlen számokkal töltjük fel a vektort.

```
jatekos = sample(1:3, kiserletek_szama, replace=T)
```

Ugyanúgy véletlenszerű számok 1-től 3-ig, a játékos választását szimbolizálja.

```
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)]  
}
```

Itt létrehozunk a műsorvezetőnek egy vektort, aminek a mérete megegyezik a kiserletek_szama méretével, a műsorvezető nem véletlenszerűen választ ajtót, hanem attól függően hogy mit választott előzőleg a játékos, ha eltalálta akkor két lehetőség közül kell választania, ha nem akkor egy lehetősége van.

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

nemvaltoztatesnyer-ben tároljuk amikor eltalálta a játékos, a valtozatban amikor változtat. A for ciklussal véig megyünk az összes kísérlethez, holvált-tal megadom azt az ajtót amire a játékos változtat, majd valtoztat vektorban tárolom az értékeket. A valtoztatesnyer-ben tároljuk azokat az eseteket amiket a játékos változtat és nyer. Ezután már csak ki kell íratni az eredményt.

2. fejezet

Helló, Chomsky!

2.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Tutoriáltam:

Tóth Csaba:<https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeaTjhhiMRoPXff9ftfWW8Uk9M1QM>

A Turing-gép fogalmát Alan Turing angol matematikus dolgozta ki 1936-ban. A Turing-gép egy absztrakt automata: a valóságos digitális számítógépek leegyszerűsített modellje.

Három nagyobb fizikai részből áll:

- Egy végtelen szalag formájában létező részekre osztott memóriából.
- Egy vezérlőegységből, ez tartalmazza a gép programját.
- Egy író-olvasó fejből, ami szimbólumkat ír vagy olvas.

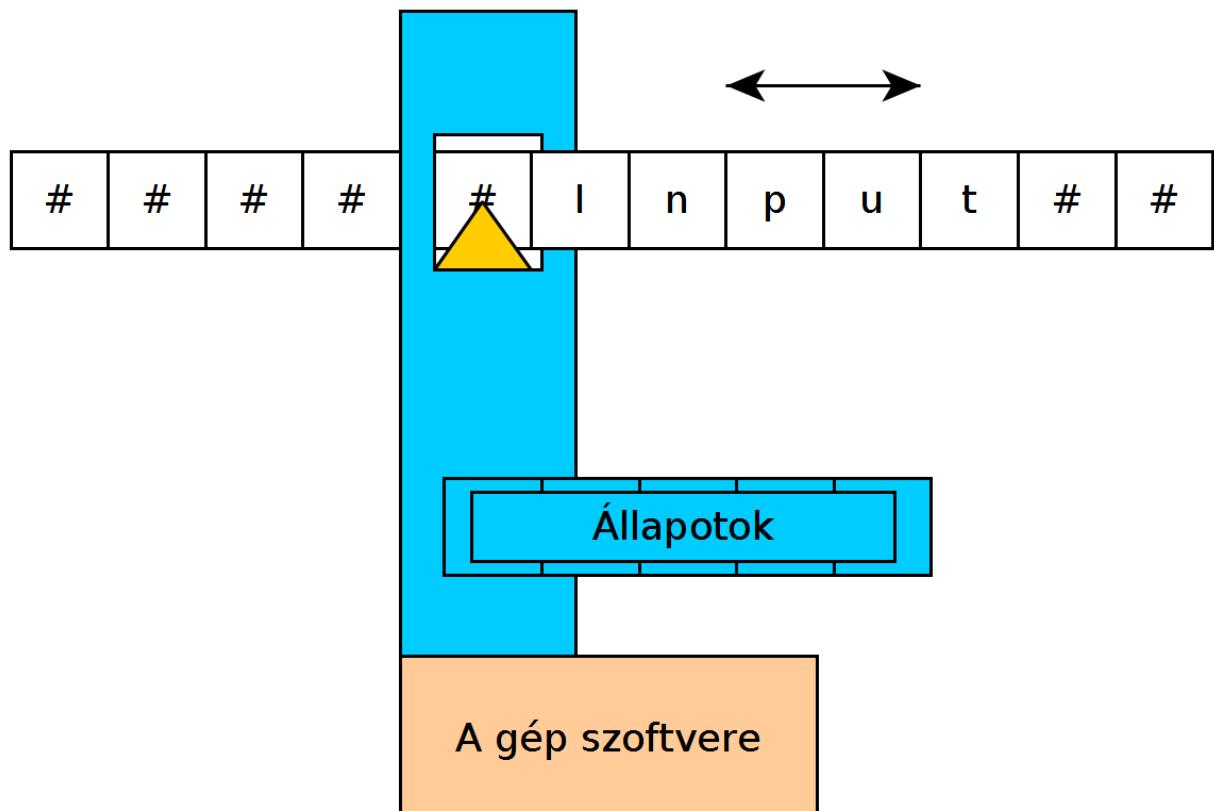
Továbbá egy szoftveregységből, ez az átmenet tábla ami vezérli a gép működését.

A Turing-gép működése:

A Turing-gépnek minden van egy aktuális pozíciója a memóriaszalagon, ahol az aktuális cella helyezkedik el, minden van egy állapota ami az aktuális állapot. Az aktuális állapot meghatározása része a gép programozásának. A gép minden lépésben beolvas egy szimbólumot a társzalag aktuális cellájáról, ezután a program attól függően hogy milyen az aktuális állapot, és a beolvasott szimbólum, a következő három lehetőség közül az egyiket írja elő:

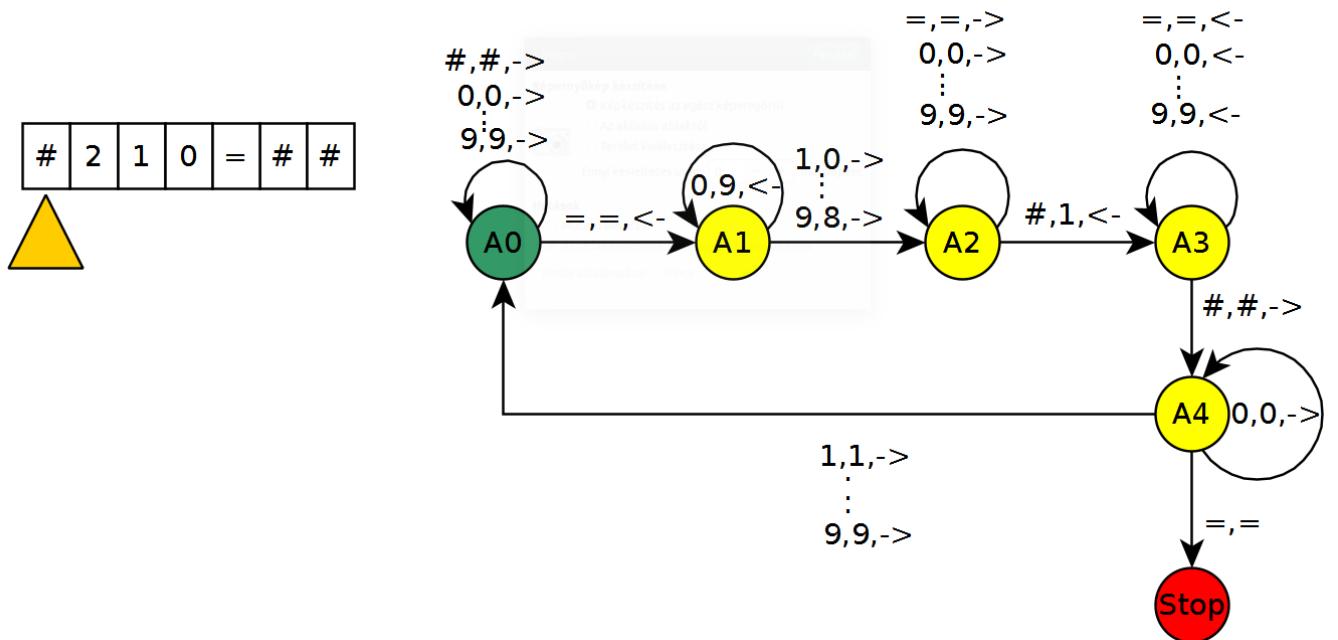
- Beír az aktuális cellába egy szimbólumot.
- Az olvasófej a társzalagon balra vagy jobbra lép, vagy helyben marad.
- A vezérlőegység átvált a jelenlegi állapotról egy másikra (ez persze lehet ugyanaz, mint a megelőző állapot), vagy átvált a stop állapotra, amely után a Turing gép megáll.

Turing gép:



2.1. ábra. Turing gép felépítése.

A következő kép az állapotátmenet gráfját fogja ábrázolni egy decimálisból unárisba átváltó turing gépnek. Az unáris számrendszer az egyes számrendszer, ez a legegyszerűbb számrendszer, itt a számok ábrázolása úgy történik, hogy kiválasztunk egy 1 értékű szinbólumot például a "|" jelet,(a turing gépünk itt '1'-eseket fog írni) ésannyiszor leírjuk amekkora az ábrázolandó szám értéke pl: $3 = |||$.



2.2. ábra. Állapotátmenet gráfja

Magyarázat:

- A0:** Kezdő állapot, innen indulunk ha #(üreset) olvasunk #(üreset) írunk, ha számot akkor leírjuk ugyanazt a számot, ha = jelet olvasunk akkor = írunk és balra lépünk a szalagon valamint tovább a következő állapotra.
- A1:** Ha 0-át olvasunk 9-est írunk majd balra lépünk a szalagon, (maradunk ebben az állapotban), ha más számjegyet olvasunk akkor 1-el kevesebbet írunk és jobbra lépünk a szalagon és a következő állapotba.
- A2:** Azt írjuk le amit olvasunk majd jobbra lépünk és maradunk ebben az állapotban kivéve, ha üres helyet (#)-ot olvasunk, ekkor 1-est írunk majd balra lépünk és a következő állapotba.
- A3:** Azt írjuk le amit olvasunk majd balra lépünk, ha üres helyet találunk üreset írunk majd jobbra lépünk és a következő állapotba.
- A4:** Ha 0-át olvasunk 0-át írunk és jobbra lépünk, maradunk ebben az állapotban, ha ilyen módon egyenlőségejlet olvasunk az azt jelenti, hogy kinulláltuk az összes helyértéket, ekkor lépünk tovább a stop állapotra, ami a gép megállását jelenti, ha 0-tól különböző számot olvasunk, akkor leírjuk ezt a számot jobbra lépünk és vissza a kezdő állapotra és kezdődik minden előlről.

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

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

Noam Chomsky egy amerikai nyelvész, ő alkotta meg a generatív nyelvtan elméletét, valamint dolgozta ki a Chomsky-hierarchiákat.

A formális nyelvtan egy absztrakt struktúra, amely pontosan leír egy formális nyelvet. A formális nyelvtanok két fő csoportba oszthatóak: generatív és analitikus nyelvtanok.

A generatív nyelvtan a legismertebb csoport, azoknak a szabályoknak a halmaza, amivel minden a nyelvben szereplő lehetséges jelsorozat előállítható, vagyis leírja, hogy hogyan lehet egy átírási eljárással a kezdő szimbólumból a többi jelsorozatot a szabályok egymás utáni alkalmazásával előállítani. A generatív nyelvtan tehát egy algoritmust formalizál, ami a nyelv összes jelsorozatát generálja.

A nyelvet alkotó jelsorozatok létrehozásához szükséges, hogy legyen egy egyedi kezdő szimbólum, ezután csak a szabályokat kell egymás után alkalmazni a kezdő szimbólum átírására. A nyelv az összes ilyen módon előállítható jelsorozatokból áll.

Egy G generatív grammatikán a következő rendezett négyest (adott sorrendben vett négy dolog együttesét) értjük:

$$G = (V_n, V_t, S, F)$$

V_n és V_t diszjunkt véges ábécék, S eleme V_n -nek, az F pedig olyan rendezett (P, Q) szópároknak egy véges halmaza, amelyekre P, Q eleme $(V_n \cup V_t)^*$ és P legalább egy V_n -beli elemet tartalmaz. (A '*' a zárójel után a $(V_n \cup V_t)$ kifejezésnél, a kifejezés által alkotott összes szót jelenti.)

V_n elemeit nemterminális jeleknek vagy változóknak nevezzük és nagybetűkkel jelöljük. V_t elemeit terminális (befejező) jeleknek nevezzük és kisbetűkkel jelöljük.

Az F elemeit képező (P, Q) rendezett párokat helyettesítési szabályoknak nevezzük és általában $P \rightarrow Q$ alakban írjuk. Az F szabályrendszer elemeit átírási szabályoknak nevezzük.

Az S kitüntetett nemterminális jel, amely a G grammatikában a generálás kiinduló vagy kezdő eleme.

Az környezetfüggő nyelvtanok a Chomsky féle hierarchia 1-es osztályába tartoznak.

Az 1-es típusú nyelvtanoknál egy szabály egyetlen X nemterminális jel helyére egy nem üres P szónak a beírását teszi lehetővé, de csak akkor ha az adott X változó közvetlen környezete $(V_n \cup V_t)^*$ által alkotott szavakból áll.

Az $a^n b^n c^n$ nyelv:

abc	n=1
aabbcc	n=2
aaabbbccc	n=3
.	.
.	.
.	.

Az $a^n b^n c^n$ nyelv, tehát azokból a szavakból áll amelyekben az a-k és b-k és c-k száma megegyezik.

Ezek alapján környezetfüggő generatív grammatikák amik az $a^n b^n c^n$ nyelvet generálják:

Változók = { S, X, Y }

Konstansok = { a, b, c }

Átváltási szabályok:

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

S lesz a kezdő elem.

S --> aXbc (S --> abc)

aXbc --> abXc (Xb --> bX)

abXc --> abYbcc (Xc --> Ybcc)

abYbcc --> aYbbcc (bY --> Yb)

aYbbcc --> aabbcc (aY --> aaX)

Változók = { A, B, C }

Konstansok = { a, b, c }

Átváltási szabályok:

A --> aAB, A --> aC, CB --> bCc, cB --> Bc, C --> bc

A lesz a kezdő elem.

A --> aAB (A --> aAB)

A --> aAB (A --> aAB)

aAB --> aaCB (A --> aC)

aAB --> aaABB (A --> aAB)

aaCB --> aabCc (CB --> bCc)

aaABB --> aaaABBB (A --> aAB)

aabCc --> aabbcc (C --> bc)

aaaABBB --> aaaaCBBB (A --> aC)

aaaaCBBB --> aaaabCcBB (CB --> bCc)

aaaabCcBB --> aaaabCBcB (cB --> Bc)

aaaabCBcB --> aaaabCBBc (cB --> Bc)

aaaabCBBc --> aaaabbCcBc (CB --> bCc)

aaaabbCcBc --> aaaabbCBcc (cB --> BC)

aaaabbCBcc --> aaaabbbCccc (CB --> bCc)

aaaabbbCccc --> aaaabbbbcccc (C --> bc)

2.3. Hivatkozási nyelv

A Dennis Ritchie és Brian W. Kernighan könyv C referencia-kézikönyv/Utasítások melléklete alapján definiálód BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

A BNF(vagy más néven Backus–Naur-forma) környezetfüggetlen nyelvtanok leírására használható metaszintaxis, széles körben használják számítógépek programozási nyelveinek nyelvtanának leírására, a BNF-nek több változata is létezik és van használatban.

A BNF leszármaztatási szabályok halmaza, amiket az alábbi formában írunk fel:

```
<szimbólum> ::= <kifejezés a szimbólumra>
```

Ahol a szimbólum egy nemterminális elem, az a feladata hogy rá szabályt alkalmazva behelyettesítsük őt, a ::= jelölés választja el a leszármaztatási szabálytól ami egy kifejezés. Ez a kifejezés tartalmazza szinbólumok sorozatát amikkel behelyettesíthetjük a ::= bal oldalán található szinbólumot.

Egyébb jelölések BNF-ben:

```
Választás:   |
Opció:      []
Iteráció:   {}
```

Az összefűzésnek nincs külön jele egyszerűen egymás után írjuk a megfelelő elemeket.

Pár példa BNF-ben történő definiálásra:

```
<Számjegy> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
<Előjel> ::= + | -
```

```
<Egész szám> ::= [ <Előjel> ] <Számjegy> {<Számjegy>}
```

Ezek szerint a C utasítás fogalma BNF-ben:

```
<utasítás> ::= <for utasítás> | <goto utasítás> | <switch utasítás> |
               <do while utasítás> | <while utasítás> |
               <feltételes utasítás> | <return utasítás> |
               <break utasítás> | <kifejezés utasítás> | <üres utasítás>

<üres utasítás> ::= ";"
```



```
<kifejezés utasítás> ::= <kifejezés> ;
```



```
<feltételes utasítás> ::= if(<kifejezés>) <utasítás>
                           [ else <kifejezés> <utasítás> ]
```



```
<while utasítás> ::= while(<kifejezés>) <utasítás>
```



```
<do while utasítás> ::= do <utasítás> while(<kifejezés>)
```



```
<for utasítás> ::= (<kifejezés>; <kifejezés>; <kifejezés>)
```



```
<switch utasítás> ::= switch(<kifejezés>) <utasítás>
```



```
<break utasítás> ::= break;
```

```
<return utasítás> ::= return; | return <kifejezés> ;  
<goto utasítás> ::= "goto azonosító;"
```

A fent említett C-s utasításokról részletesebben:

Utasítás:

Az utasítást egy programozási nyelvben úgy lehet felfogni, mint egy imperativ programozási nyelv legkisebb elemét. A programot egy vagy több utasítás alkotja, ezek az utasítások egymást követően sorban hajtódnak végre.

A kifejezés utasítás:

A leggyakoribb utasítási forma. Így néz ki:

```
kifejezés;
```

Ezek legtöbbször értékkadások vagy függvényhívások.

A feltételes utasítás:

```
if (kifejezés)  
    utasítás  
  
if (kifejezés)  
    utasítás  
else  
    utasítás
```

A számítógép kiértékeli a kifejezést és ha igaz akkor az első alutasítás teljesül, ha hamis akkor tovább lép a következő sorra. Az else-vel kapcsolatos kétértelműséget a C úgy oldja meg hogy az else a legutóbb talált if-hez tartozik.

A while utasítás:

```
while (kifejezés)  
    utasítás
```

Az utasítás teljesítése mindaddig ismétlődik, ameddig a kifejezés igaz. A kifejezés vizsgálata mindenkor az utasítás megkezdése előtt történik.

A do utasítás:

```
do  
    utasítás  
while (kifejezés)
```

Az alutasítás végrehajtása addig ismétlődik, ameddig a kifejezés értéke hamis nem lesz. A kifejezés vizsgálata mindenkor az utasítás végrehajtása után történik.

A for utasítás:

```
for (1._kifejezés; 2._kifejezés; 3._kifejezés)  
    utasítás
```

Ez az utasítás egyenértékű a következő while utasítással:

```
1._kifejezés;  
while (2._kifejezés) {  
    utasítás  
    3._kifejezés;  
}
```

Az 1._kifejezés a ciklust inicializálja, a második az iterációt megelőző vizsgálat, ha ez teljesül akkor lesz az iteráció végrehajtva. A 3._kifejezés gyakran az egyes iterációk után végrehajtandó inkrementálást határozza meg, a kifejezések közül tetszőleges számú elhagyható.

A switch utasítás:

```
switch (kifejezés)  
utasítás
```

A kifejezés értékétől függően a vezérlés valamelyik megadott utasításra adódik át. A switch utasításon belül előforduló utasítások megcímkézhetőek a case előtaggal:

```
case állandó_kifejezés:
```

Ugyanazon a switchen belül két állandónak nem lehet egyforma az értéke.

```
default:
```

Ha a kifejezés értéke megegyezik a case állandók valamelyikével, akkor a vezérlés a case előtagot követő utasításra ugrik át. Ha nem egyezik egyik sem akkor a default után megadott utasítás hajtódik végre, ha nincs default megadva akkor a gép a switchben lévő egyik utasítást sem hajtja végre.

A break utasítás:

```
break;
```

Hatására befejeződik a break-et körülvevő while,do,for vagy switch utasítás.

A return utasítás:

```
return;  
return kifejezés;
```

A függvény hívójához tér vissza, az első esetben a viszaadott érték határozatlan a másodikban a megadott kifejezés értékét adja vissza.

A goto utasítás:

```
goto azonosító;
```

A vezérlés feltétel nélkül a goto utasítással adható át, az azonosító a végrehajtott függvényen belül elhelyezett címke kell hogy legyen.

A címkezett utasítás:

```
azonosító:
```

bármelyik utasítást megelőzheti, a címke egyedül a goto utasítás célpontjaként szolgál.

A nulla utasítás:

```
;
```

Hordozhat címkét valamelyik összetett utasítás előtt, vagy while-hoz hasonló valamelyik ciklusutasítás számára üres ciklustörzset képez.

Ha egy adott szabvány szerint szeretnénk fordítani akkor azt a következőképpen tehetjük meg:

```
gcc -std=c99 hello.c -o hello
```

-std=c99 kapcsoló szükséges a c99-es szabványhoz és -std=c89 a c89-eshez.

Több különbség is van a két szabvány között, az egyik a kommentek megadásának módja, a következő kód például: -std=c89 kapcsolóval nem fordul le, de a -std=c99 már igen:

```
#include <stdio.h>

int main ()
{
    // Print string to screen.
    printf ("Hello World\n");
}
```

A hiba a komment megadásában van a c89-es standard nem fogad el "//"-el kezdődő kommentet.

Az alábbi kód szintén csak a -std=c99 kapcsolóval fog fordulni:

```
#include <stdio.h>

void main ()
{
int alma=2;

for(int i=0;i<5;++i)
    printf ("%d\n",alma);
}
```

A c89-es szabványban a for ciklus előtt kell deklarálni a változót:

```
#include <stdio.h>

void main ()
{
int alma=2;
int i;
for(i=0;i<5;++i)
    printf ("%d\n",alma);
}
```

Így már működik a -std=c89-es szabvány szerint is.

2.4. Saját lexikális elemző

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/lex>

A programot a Flex nevű lexikális elemző generátor segítségével fogjuk elkészíteni, ami a megadott kód alapján lexikális elemzöt generál számunkra C-s kód formájában.

A ".l" kiterjesztésű fájlt így fordítjuk:

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

Ez a parancs legenerálja a ".l" kiterjesztésű fájlból a C-s kódót, majd ezt a kódot fordítjuk gcc-vel "-lfl" kapcsoló szükséges a linkeléshez:

```
gcc realnumber.c -o realnumber -lfl
```

Teljes forráskód:

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

Három fő részből áll a program, nézzük meg részenként:

```
%{  
#include <stdio.h>  
int realnumbers = 0;  
}  
digit [0-9]
```

A %-jelek közötti kódot berakja a lexer a készítendő C programba. A realnumbers int változót deklaráljuk 0 kezdőértékkel, itt tároljuk hány számot olvastunk be, standard input/output könyvtárat tartalmazzuk. A digit-et definiáljuk, karaktercsokrot adunk meg a kockás zárójelek között itt [0-9].

Következő rész, a fordítási szabályok:

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

A digit definíciókat használjuk {digit}* itt a '*' azt jelenti, hogy a digitból akármennyi lehet, 0 db-is. (\.{digit}+)? -nél a '.' az előtte lévő levédő karakter '\' miatt, itt a karakter '.'-ot jelenti, egyébként a levédő karakter nélkül azt jelentené hogy bármire rálehet illeszteni, a '+' karakter jelentése pedig hogy akárhány darad de legalább 1. A '?' karakter az egész kerekzárójelben lévő kifejezésre vonatkozik, azt jelenti vagy van, vagy nincs.

Ha a leírtaknak megfelelőt talál akkor növeli a realnumbers változót 1-el. Ezután kiíratjuk printf-el először string-ként majd, az atof függvény segítségével kiíratjuk számként is.

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

A harmadik részben pedig, yylex() funkciót hívjuk, ezzel indítjuk a lexikális elemzést, majd printf-el kiírjuk a végeredményt.

2.5. l33t.l

Lexelj össze egy l33t cipher!

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/l33t.l>

A l33t, vagy más néven leetspeak egy olyan írásmód, amelyben a hagyományos latin írásjeleket egyéb ASCII karakterekkel helyettesítjük pl: leet=l33t. Ezt a programot szintén a Flex nevű lexikális elemző generátor segítségével fogjuk elkészíteni.

Teljes forráskód:

```
/*
Forditas:
$ lex -o 1337d1c7.c 1337d1c7.l
```

Futtatas:

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

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

```
#include <ctype.h>

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

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

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

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

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

```
%}
%%
. {

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

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

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

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

            found = 1;
            break;
        }
    }

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

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

Nézzük részenként:

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

A cipher strukturában van egy karakter meg 4 db char pointerból álló tömb, a karakterben tároljuk majd azt

a betűt amit ki szeretnénk cserélni, a leet nevű tömben pedig azokat az írásjeleket amire cserélni szeretnénk. A l337d1c7 tömböt ilyen cipher strukturákkal töltjük fel.

```
%%
. {

    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);
}

%%
```

A '.' itt bármelyik karaktert jelenti (kivéve az üres sort), ez azt jelenti hogy a program indítása után bárminelyen karakter ütünk be aktiválódik a blokk. A blokk a for ciklussal kezdődik ami elmegy L337SIZE méretéig. L337SIZE egy nevesített konstans aminek a mérete l337d1c7 struktura mérete bájtokban számolva elosztva 1 db cipher struktúra méretével.

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

Ha a l337d1c7 i-edik eleme egyenlő a bevitt karakterrel (tolower funkció kisbetűt ad vissza az inputból).

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

Véletlenszerű számgenerálás 0-tól 100-ig, ezt az 'r' nevű változóban tároljuk.

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

```
    printf("%s", 1337d1c7[i].leet[1]);
else if(r<98)
    printf("%s", 1337d1c7[i].leet[2]);
else
    printf("%s", 1337d1c7[i].leet[3]);
```

Kiíratjuk 'r' értékéhez mérten megfelelő elemet.

```
found = 1;
```

Itt tároljuk hogy volt-e kiíratás.

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

Ha nem volt akkor kiíratjuk az eredetileg bevitt karaktert.

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

A main-ben inicializáljuk a randomszámlálót, majd yylex()-el indítjuk a lexikális elemzést.

2.6. A források olvasása

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

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

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

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



Bugok

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

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

Egy for ciklus ami 0-tól indul 5-ig, minden iterációban i értékét növelte eggyel prefix.

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

Egy for ciklus ami 0-tól indul 5-ig, minden iterációban i értékét növelte 1 el postfix, for ciklusnál nincs jelentősége hogy prefix vagy postfix növeljük az i értékét.

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

Egy for ciklus ami 0-tól indul 5-ig, tomb i-edik eleméhez hozzárendeli i-t majd megnöveli 1-el az i értékét. Viszont a tomb[i] = i++ kifejezés hibát visz a programba mert a végrehajtás sorrendje nem megfelelően definiált, másik számítógépen fordítva vagy másik fordítóprogramot használva más eredményt kaphatunk.

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

Egy for ciklus ami 0-tól indul, n-ig tart és addig ameddig azt amire az 's' pointer mutat az értékét hozzárendeli ahoz a tárhelyhez ahova 'd' pointer mutat, majd lépteti a pointereket 1 el, majd minden iteráció végén növeli 'i' értékét 1-el.

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

A printf használatával kiíratunk két integert amit f(a, ++a) és f(++a, a) függvények adnak vissza, viszont a függvényargumentumok kiértékelési sorrendje nincs meghatározva, ne írunk olyan kódot ami feltételez egy kiértékelési sorrendet mert hibát visz a programba.

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

A printf használatával kiíratunk két integert az egyik, amit 'f' függvény ad vissza, a másik az 'a' változó értéke.

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

A printf használatával kiíratunk két integert az egyik amit az 'f' függvény ad vissza, itt 'f' argumentumának egy referenciát adunk 'a'-ra, a másik az 'a' változó értéke, mivel referenciát adtunk át feltételeznünk kell hogy az 'f' funkció módosítani fogja 'a' értékét ezért ez egy bugos kód mert nem tudhatjuk hogy printf-en belül milyen sorrendben fognak a változók kiértékelődni.

2.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

Végtelen sok prímszám van:

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

Végtelen sok iker-prímszám van:

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

Véges sok prímszám van:

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

Véges sok prímszám van:

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

2.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- Egész.
- Egészre mutató mutató.
- Egész referenciajára.
- Egészek tömbje.
- Egészek tömbjének referenciajára. (nem az első elemé)
- Egészre mutató mutatók tömbje.
- Egészre mutató mutatót visszaadó függvény.
- Egészre mutató mutatót visszaadó függvényre mutató mutató.
- Egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény.
- Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre.

```
int main ()  
{  
    //egész  
    int a=5;  
  
    //egészre mutató mutató
```

```
int* point_to_a = &a;

//egész referenciajá
int& ref_to_a = a;

//egészek tömbje
int tomb[5] = {1,2,3,4,5};

//egészek tömbjének referenciajá (nem az első elemé)
int (&tomb_ref)[5] = tomb;

//egészre mutató mutatók tömbje
int* pointers_to_int[5];

//egészre mutató mutatót visszaadó függvény
int* function();

//egészre mutató mutatót visszaadó függvényre mutató mutató
int* (*function_pointer)();

//egészet visszaadó és két egészet kapó függvényre mutató mutatót visszadó ←
//, egészet kapó függvény
int (*function_1(int)) (int a,int b);

//függvénymutató egy egészet visszaadó és két egészet kapó függvényre ←
//mutató mutatót visszadó, egészet kapó függvényre
int (*(*function_pointer_1)(int)) (int a,int b);
}
```

Mit vezetnek be a programba a következő nevek?

```
int a;
```

Egész típusú változó.

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

Egész típusú pointer a-ra mutat.

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

Egész típusú referencia a-ra hivatkozik.

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

Egészek tömbje aminek a mérete 5.

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

Egészek tömbjének referenciajá aminek mérete 5, és 'c' tömb címére hivatkozik.

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

Egészre mutató pointert tartalmazó tömb aminek mérete 5.

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

Egészre mutató pointert visszaadó funkció ami nem kér értéket.

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

Egészre mutató pointert visszaadó funkcióra mutató pointer.

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

Egészet visszaadó és két egészet bekérő funkcióra mutató pointert visszaadó intet bekérő funkció.

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

Egészet visszaadó és két egészet bekérő funkcióra mutató pointert visszaadó intet bekérő funkcióra mutató pointer.

DRAFT

3. fejezet

Helló, Caesar!

3.1. double ** háromszögmátrix

Megoldás forrása: https://github.com/BorvizRobi/vegyes/blob/master/**haromszogmatrix.cpp

A double** típusú pointer egy pointer, ami double értékre mutató pointerre mutat, ilyen pointer alkalmazásával fogjuk létrehozni egy double háromszögmátrixot.

A mátrix a matematikában mennyiségek téglalap alakú elrendezése, táblázata. Háromszögmátrixnak nevezük az olyan kvadratikus(négyzetes) mátrixot, amelynek a főátlója alatti összes eleme, vagy főátlója fölötti összes eleme nulla.

Például a következő mátrix egy háromszögmátrix:

3	0	0	0
2	2	0	0
4	9	1	0
12	6	5	3

3.1. ábra. Háromszögmátrix

Nézzük a kódot:

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

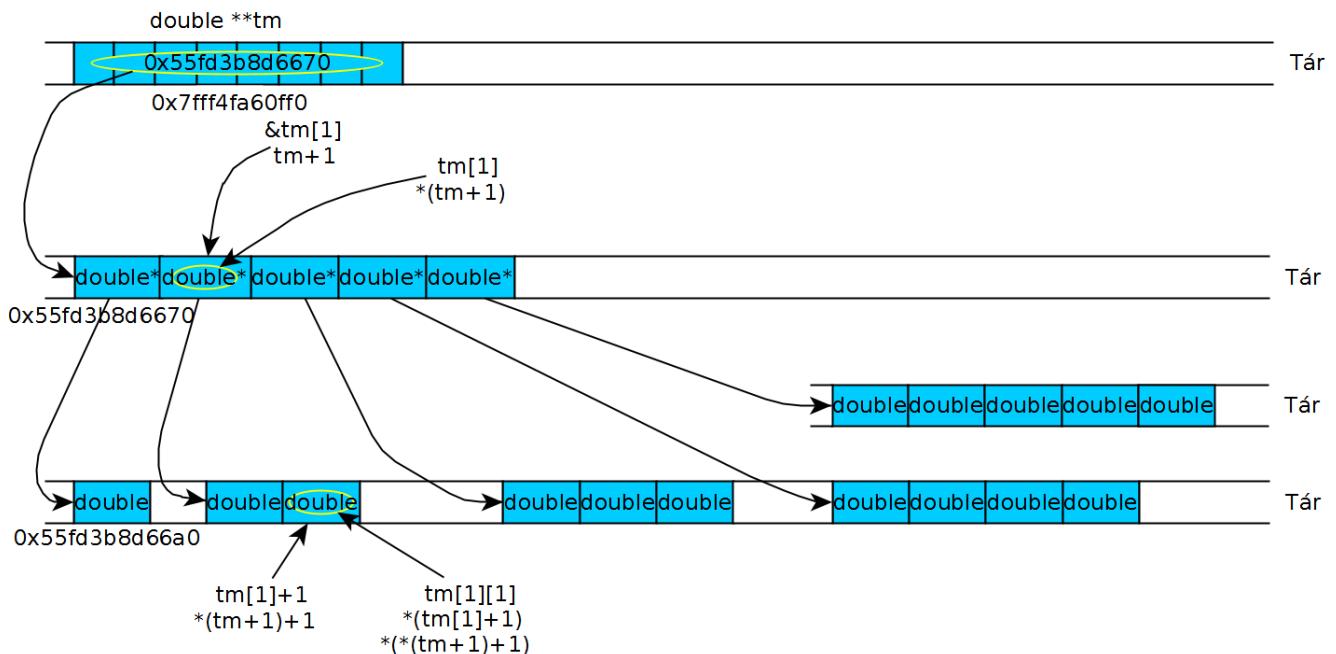
int
main ()
{
    int nr = 5;
    double **tm;

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

    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
```

```
{  
    return -1;  
}  
  
printf("%p\n", tm);  
  
for (int i = 0; i < nr; ++i)  
{  
    if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL)  
    {  
        return -1;  
    }  
  
}  
  
printf("%p\n", tm[0]);  
  
for (int i = 0; i < nr; ++i)  
    for (int j = 0; j < i + 1; ++j)  
        tm[i][j] = i * (i + 1) / 2 + j;  
  
for (int i = 0; i < nr; ++i)  
{  
    for (int j = 0; j < i + 1; ++j)  
        printf ("%f, ", tm[i][j]);  
    printf ("\n");  
}  
  
tm[3][0] = 42.0;  
(* (tm + 3))[1] = 43.0;  
*(tm[3] + 2) = 44.0;  
* (* (tm + 3) + 3) = 45.0;  
  
for (int i = 0; i < nr; ++i)  
{  
    for (int j = 0; j < i + 1; ++j)  
        printf ("%f, ", tm[i][j]);  
    printf ("\n");  
}  
  
for (int i = 0; i < nr; ++i)  
    free (tm[i]);  
  
free (tm);  
  
return 0;  
}
```

A double** szemléltetése:



3.2. ábra. ** háromszögmátrix a memóriában

Nézzük részenként:

```
int nr = 5;
```

5 soros lesz a mátrix.

```
double **tm;
```

Pointer ami double-ra mutató pointerre mutat 8 bájtot foglal le.

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

Kiírja a tm (**double típusú) pointer memóriacímét, azt a címet amin a memóriában megtalálható nem pedig azt amire mutat.

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

A malloc függvény az operációs rendszertől kér memóriát, itt 40 bájtot foglal le, mert $5 * (\text{double pointer mérete} = 8)$. A malloc visszaad a double*-okra egy mutatót, ha nem sikerül memóriát foglalnia vagyis "`== NULL`" akkor -1 el kilépünk a programból.

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

A printf itt kiíratja tm értékét, vagyis azt a memóriacímet amire tm mutat.

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

}
```

A for ciklus itt 0-tól nr-ig megy vagyis 5-ig. A cikluson belül a malloc a tm[i]-edik elemének lefoglalja a megfelelő méretű helyet, majd erről visszaad egy pointert, a double mérete 8 bájt tehát, az elsőnek 8 bájtot a másodiknak 16-ot és így tovább.

```
printf("%p\n", tm[0]);
```

Itt a printf kiírja a tm által mutatott, egybefüggő memóriacím első elemének értékét, ez a memóriacím egy pointert tartalmaz, tehát azt a memóriacímet íratjuk ki amire ez a pointer mutat.

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

A for ciklus alatt feltöljük a mátrixot 0-14-ig.

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

Itt kiíratjuk a mártix értékeit.

```
tm[3][0] = 42.0;
(*tm + 3)[1] = 43.0;
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;
```

Itt 42-től 45-ig értékeket hozzárendeljük a 4. sor elemeihez.

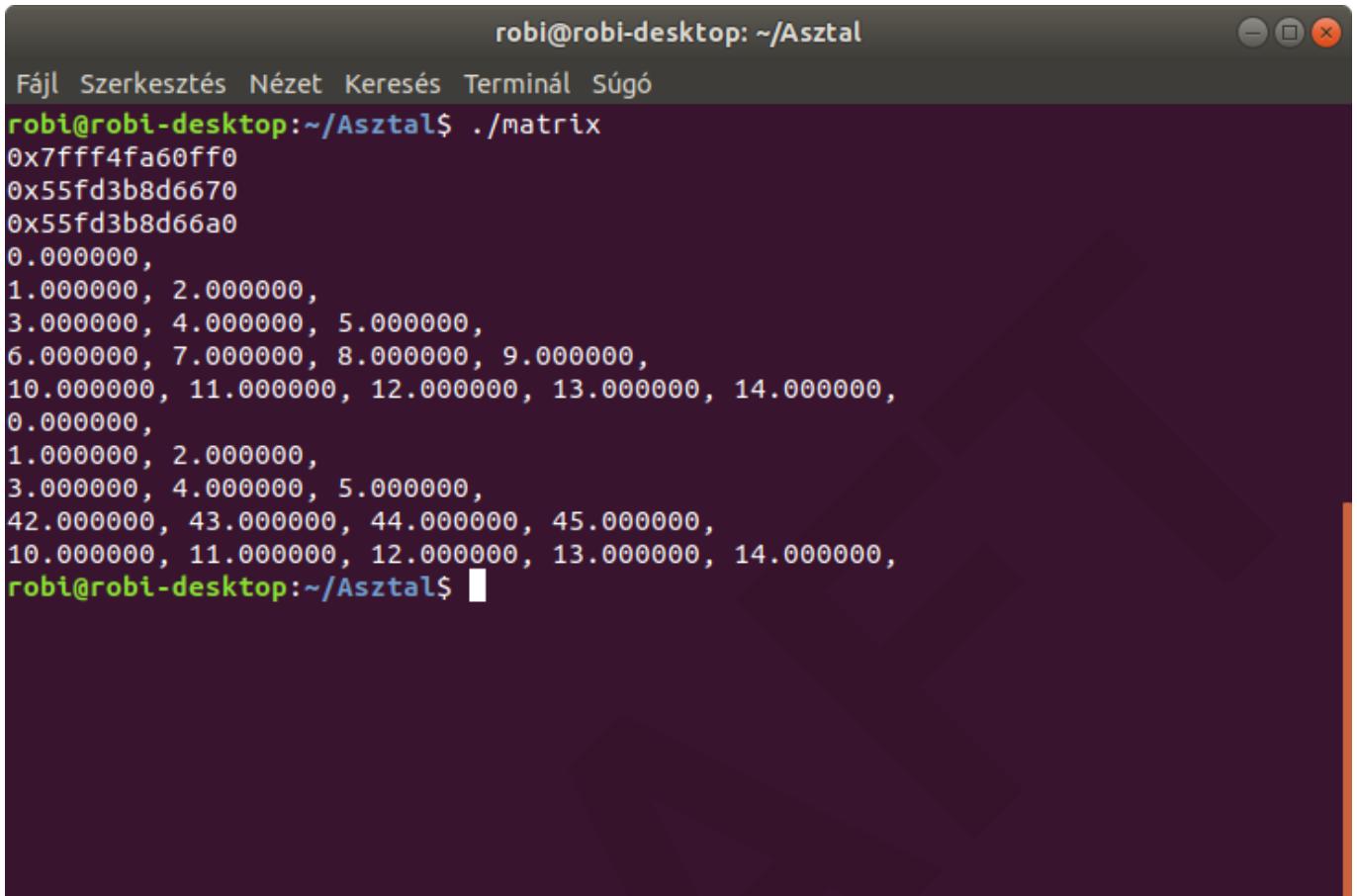
```
for (int i = 0; i < nr; ++i)
    free (tm[i]);

free (tm);

return 0;
```

Szabadítjuk a pointereket, return 0-val befejezzük a programot.

A program eredménye:



```
robi@robi-desktop: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
robi@robi-desktop:~/Asztal$ ./matrix
0x7fff4fa60ff0
0x55fd3b8d6670
0x55fd3b8d66a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
robi@robi-desktop:~/Asztal$
```

3.3. ábra. háromszögmátrix

3.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/e.c>

EXOR vagy másnéven kizáró vagy, egy logikai művelet, amely akkor hamis ha minden két állítás logikai értéke megegyezik. A következő program az EXOR művelet segítségével fog egy szöveges fájl-t titkosítani.

Így fordítjuk:

```
gcc e.c -o e -std=c99
```

Használata:

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

A kercerece a titkos jelszó amivel titkosítani szeretnénk, tiszta.szoveg az a szöveges fájl amit titkosítani szeretnénk, titkos.szoveg pedig az a fájl ahol tárolni szeretnénk a titkos szövegünket.

Dekódolás:

```
./e kercerece <titkos.szoveg
```

Teljes forráskód:

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

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

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

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

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

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

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

Nézzük részenként:

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

Nevesített konstansok, MAX_KULCS értéke innentől kezdve 100, BUFFER_MERET értéke pedig 256.

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

argc és **argv a parancssori argumentumok kezeléséhez szükséges, argc-ben tároljuk parancssorban megadott stringek számát, char **argv pedig a pointereket tartalmazza, amik a megadott stringek első karaktereire mutatnak.

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

Két char tömb, a kulcs-nak a mérete 100, a buffer mérete 256.

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

A kulcs_index-et növelve fogunk végig lépkedni a megadott kulcson, olvasott_bajtok-ban tároljuk a beolvasott bajtokat.

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

Az strlen funkcióval lekérdezzük a kulcs hosszát(a kulcsnak megfelelő parancssori argumentum argv[1]-ben van tárolva) és kulcs_meret-ben tároljuk. Az strncpy egy funkció stringek másolására, itt a kulcs nevű karakter tömbbe másoljuk argv[1] tartalmát(vagyis a kulcsot), MAX_KULCS-a adjuk meg a max méretet, amit másolni szeretnénk ez itt 100.

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

A read funkcióval olvasunk be a buffer-be a standard inputról (0-val jelezzük ezt a funkciónak), BUFFER_MERETnek megfelelő mennyiségű bajtot. A funkció a beolvasott bajtok számát adja vissza, ezt tároljuk olvasott_bajtokban, amíg tudunk olvasni addig fut a ciklus.

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

A for ciklus 0-tól indul az olvasott bajtok számáig, i értékét minden iteráció után 1- el növelve. A ciklus-magban a buffer i-edik elemét összexorozzuk az aktuális kulcsindexel, majd kulcs_index értékéhez mindig hozzáadunk egyet majd %-os osztást végezünk a kulcs méretével így a kulcsindex minden 0-tól indul és ha eléri a kulcs méretét akkor ismét nullárol indul.

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

write funkcióval írunk a standard outputra ezt jelenti itt az 1-es,a buffer-ből írunk ki, az olvasott_bajtok méretével megegyező bajtot írunk ki.

3.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/exortitkosito.java>

A java kódot a következőképpen fordítjuk:

```
javac ExorTitkosító.java
```

Így futtatjuk:

```
java ExorTitkosító alma > titkosított.szöveg
```

Itt alma lesz a titkos kulcs és a titkosított szöveget a "titkosított.szöveg" nevű fájlba irányítjuk.

A titkosított szöveget így törjük:

```
java ExorTitkosító alma < titkosított.szöveg
```

Ezután a program kiírja nekünk a képernyőre a tiszta szöveget.

Teljes kód:

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

```
        new ExorTitkosító(args[0], System.in, System.out);  
  
    } catch(java.io.IOException e) {  
  
        e.printStackTrace();  
  
    }  
  
}
```

Nézzük soronként:

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

A funkció test előtt azt jelenti, hogy eldobunk egy I/O exception-t ha valami gond adódna a I/O streammel.

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

Java-ban a legkisesebb integer típus a byte típus, 8 bit tárolására használjuk. Ebből a típusból hozunk létre két tömböt, a kulcs-ot és a buffer-t. A getBytes() funkcióval megkapjuk a kulcs bitjeit és ezeket a biteket tároljuk a kulcs tömbben. A buffer méretét 256-ra állítjuk be, kulcsIndex-ben tároljuk majd az éppen aktuális kulcsindexet, olvasottBájtok -ban pedig a beolvasott bájtok számát.

```
while((olvasottBájtok =  
       bejövőCsatorna.read(buffer)) != -1) {  
  
    for(int i=0; i<olvasottBájtok; ++i) {  
  
        buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);  
        kulcsIndex = (kulcsIndex+1) % kulcs.length;  
    }  
}
```

Ameddig tudunk addig olvasunk be a buffer-be, olvasottBájtok-ban tároljuk a beolvasott bájtok számát. A for ciklus 0-tól indul az olvasott bájtok számáig, i értékét minden iteráció után 1- el növelve. A ciklusmagban a buffer i-edik elemét összexorozzuk az aktuális kulcsindexel, majd kulcs_index értékéhez mindig

hozzáadunk egyet majd %-os osztást végzünk a kulcs méretével így a kulcsindex mindig 0-tól indul, és ha eléri a kulcs méretét akkor ismét nulláról indul.

```
kimenőCsatorna.write(buffer, 0, olvasottBájtok);
```

A write funkció a buffer tartalmát(a titkosított szöveget) kiírja a standard output-ra, olvasottBájtok a bájtok számát tartalmazza amit kifog írni.

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

A main-ben meghívjuk az ExorTitkosító-funkciót, bemenetének megadjuk a kulcsot és az input/output streameket, majd try blockba tesszük ezzel kezelve az exception-t, ha volt exception akkor azt a catch-al elkapjuk, a printStackTrace() funkció kiírja, hogy mi volt az oka az exception-nak és hol.

3.4. C EXOR törő

Tutoriáltam:

Tóth Csaba:<https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeaTjhhiMRoPXff9ftfWW8Uk9M1QM>

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/create/master/t.c>

A következő program feltöri az előző feladatban előállított titkos szöveget, ebben a fomájában 8-as hosszúságú kulcsméretre és 0-9 ig terjedő számokból álló kulcs feltörésére használható.

Teljes forráskód:

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256  
#define KULCS_MERET 8  
#define _GNU_SOURCE  
  
#include <stdio.h>  
#include <unistd.h>  
#include <string.h>
```

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

    int kulcs_index = 0;

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

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

    }

}

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

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

```
return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

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

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

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

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

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

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

```
    }

    return 0;
}
```

Nézzük részenként:

```
#define _GNU_SOURCE
```

A strcasestr funkció használatához szükséges.

```
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;
}
```

Funkció az átlagos szóhossz kiszámítására, const char* -ot és int-et kér be double értéket ad vissza, 'sz'-ben tároljuk hogy hány szóközt találtunk, a for ciklus 0-tól titkos_meret-ig megy, ha titkos[i]-edik eleme szóköz akkor 'sz' értékét növeljük 1-el, majd titkos_meret-et elosztjuk a szóköözök számával így megkapjuk az átlagos szóhoszt, ezt az értéket adja vissza a funkció.

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

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

A tiszta_lehet funkció szintén egy const char* -ot kér be és egy intet, viszont int értéket ad vissza 0-át vagy 1-et, ezzel jelzi hogy tiszta-e a szöveg ha tiszta akkor sikerült az exor törés, 0 hamisat 1 igazat jelent. A szohossz-ban tároljuk az atlagos_szohossz által visszaadott értéket, ha az átlagos szóhossz nagyobb mint 6 de kisebb mint 9 valamint a strcasestr() funkció megtalálja a megadott szavakat a szövegen akkor a tiszta_lehet funkció igaz értékkel tér vissza.

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

{
    int kulcs_index = 0;

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

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

}

}

Az exor funkció nem ad vissza értéket, ezt a funkciót használjuk az exor művelet elvégzésére. A for ciklus 0-tól indul az olvasott bajtok számáig, i értékét minden iteráció után 1- el növelve. A ciklusmagban a titkos i-edik elemét összeexorozzuk az aktuális kulcsindexel, majd kulcs_index értékéhez mindig hozzáadunk egyet majd %-os osztást végeünk a kulcs méretével így a kulcsindex mindig 0-tól indul és ha eléri a kulcs méretét akkor ismét nulláról indul.

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

    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);

}
```

Az exor_tores funkcióval indítjuk az exor funkciót majd tiszta_lehet funkcióval ellenőrizzük hogy sikerült-e a törés.

```
int
main (void)
{

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

A main-on belül kulcs nevű char tömbben fogjuk tárolni az előállított kulcsokat, titkos nevű char tömbben pedig a titkos szöveget. A 'p' nevű karakter pointer a titkos szöveg első karakterére mutat, olvasott_bajtokban pedig a beolvasott bajtok mennyiségét fogjuk tárolni.

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

A titkos fájl beolvasását végezzük, a read funkció visszaadja értéknek hogy hány bajtot olvastunk be összesen, a read funkció 3 argumentumot kér tőlünk, az első argumentumnak azt kell megadni ahonnan olvasni

szeretnénk, 0-val azt adjuk meg hogy a standard inputról olvasunk. A második argumentum hogy hol szeretnénk tárolni a beolvasott szöveget, a harmadik argumentum pedig hogy mennyi bájtot szeretnénk olvasni összesen. A harmadik argumentumot itt egy aritmetikai if-el adjuk meg a "? :" operátor azt jelenti hogy ha a '?' előtti kifejezés igaz akkor a '?' utáni kifejezés hajtódik végre vagyis a beolvasni kívánt bájtok mérete OLVASAS_BUFFER méretével fog megegyezni, ha hamis akkor "titkos + MAX_TITKOS - p" értékével. A p pointert az utolsó beolvasott bájtot tartalmazó memóriacímre fog mutatni.

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

A titkos tömbben feltöljük '\0'-val az üres helyeket.

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

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

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

Majd for ciklusokkal előállítjuk az összes lehetséges kulcsot (itt a 8 db for ciklus egy 8 jegyű kulcsot feltételez és mivel '0'-'9'-ig tartanak a ciklusok ezért csak a 0-9 számjegyekből álló titkos kulcsot fogja feltörni). Ha sikerült az exor törés akkor kiíratjuk a kulcsot és a tiszta szöveget, ha nem sikerült akkor újra exorozunk hogy visszakapjuk az eredeti titkos szöveget majd újra próbálkozunk.

3.5. Neurális OR, AND és EXOR kapu

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/nn.r>

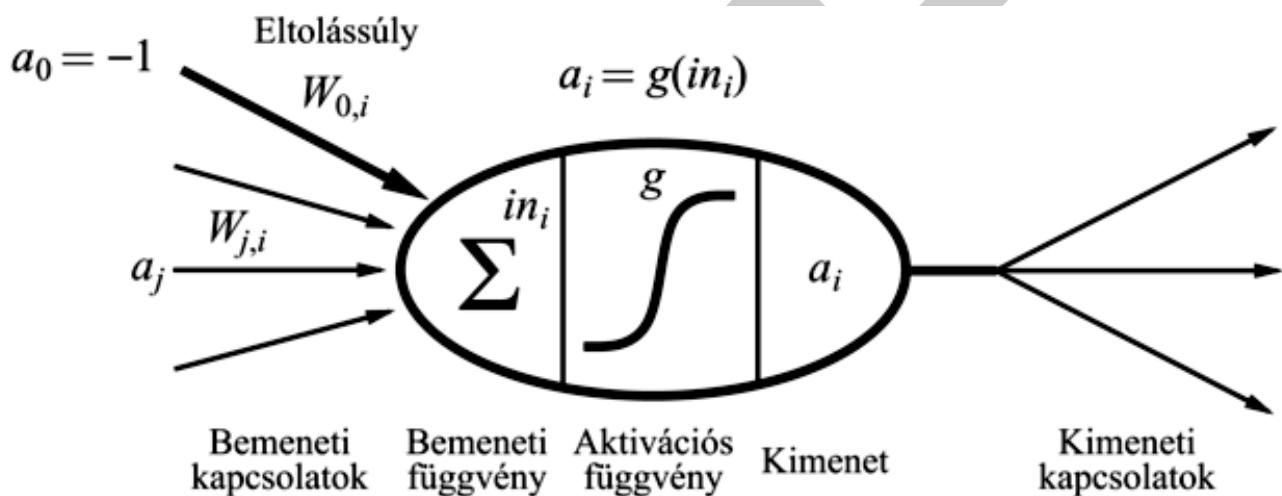
A következő program R-nyelvben van írva, és neurális hálókat fogunk létrehozni, amelyeknek betanítjuk a OR, AND és EXOR műveleteket.

A program működésének megértéséhez először nézzük meg, hogy mik is azok a neuronok és neurális hálók:

Az idegrendszer legkisebb egysége a neuron, neuronnak nevezzük az idegsejtet és nyúlványainak együttesét. A neuronok ingerlékeny sejtek, amik ingerlettelvételre és idegi ingerületek továbbítására specializálódottak. Azt gondoljuk hogy az agy információfeldolgozó kapacitása a neuronoknak köszönhető. Ezért a korai mesterséges intelligencia kutatás is az ilyen emberi testben található neuronok mesterséges létrehozására irányult. Egy neuron akkor tüzel, amikor a bemeneti értékek súlyozott összege meghalad egy küszöbot.

A neuronokról bővebben: http://project.mit.bme.hu/mi_almanach/books/aima/ch20s05

A neuron egyszerű matematikai modellje:



3.4. ábra. Neuron

A neurális hálók, ilyen irányított kapcsolatokkal összekötött csomópontokból állnak, minden kapcsolat rendelkezik egy hozzá tartozó súlyval, ami meghatározza a kapcsolat erősséget és előjelét, mindenek először a bemeneteinek súlyozott összegét számolja ki, ebből a kimenetet úgy kapja hogy egy aktivációs függvényt alkalmaz a kapott összegre.

Neurális OR, AND és EXOR kapu R-ben:

```
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])
```

Nézzük részenként:

```
library(neuralnet)
```

A neuralnet csomag betöltése.

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

Itt adjuk meg a szabályokat, a1-ben és a2-ben megadjuk a bemenetet és megmondjuk hogy az OR-ban mit kell kapni, pl.: a1=0 és a2=0 akkor OR=0, vagy a1=1 és a2=1 akkor OR=1. Ezután elkezdi tanítani magát és beállítja a súlyokat, amikor már futtatjuk akkor befejeződik a tanítás.

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

Adatot csinálunk belölle.

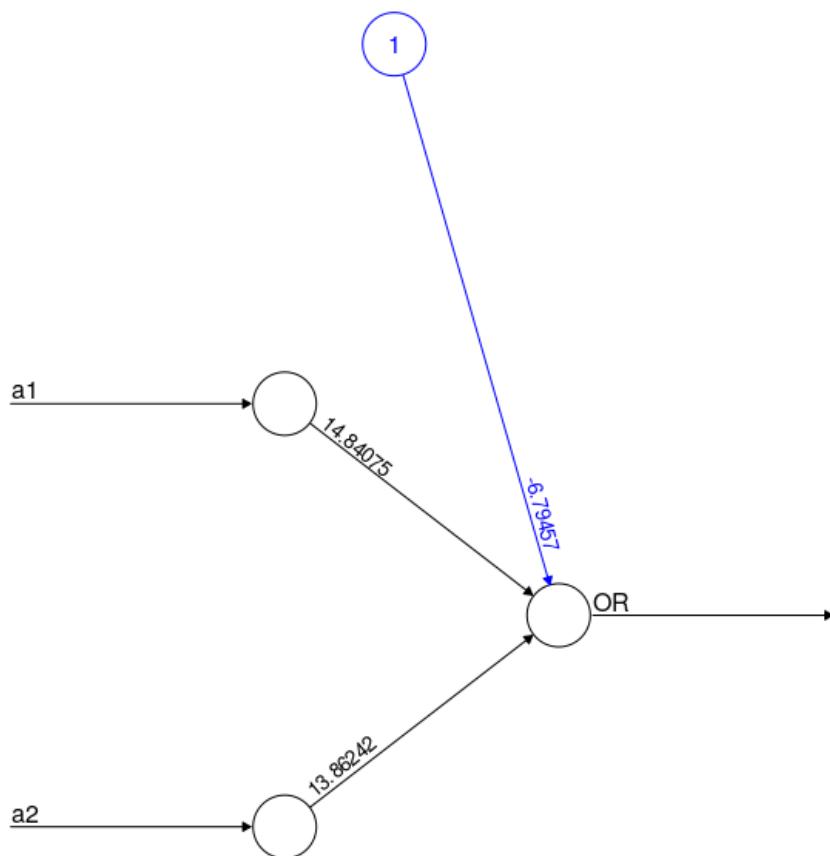
```
nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE, ←
  stepmax = 1e+07, threshold = 0.000001)
```

R-be a neuralnet függvény csinálja meg a neurális hálót.

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

A compute parancssal ellenőrizzük, itt ténylegesen beadjuk neki a megfelelő értékeket és ellenőrizzük az eredményt.

R Graphics: Device 2 (ACTIVE)



Error: 1e-06 Steps: 145

3.5. ábra. NN

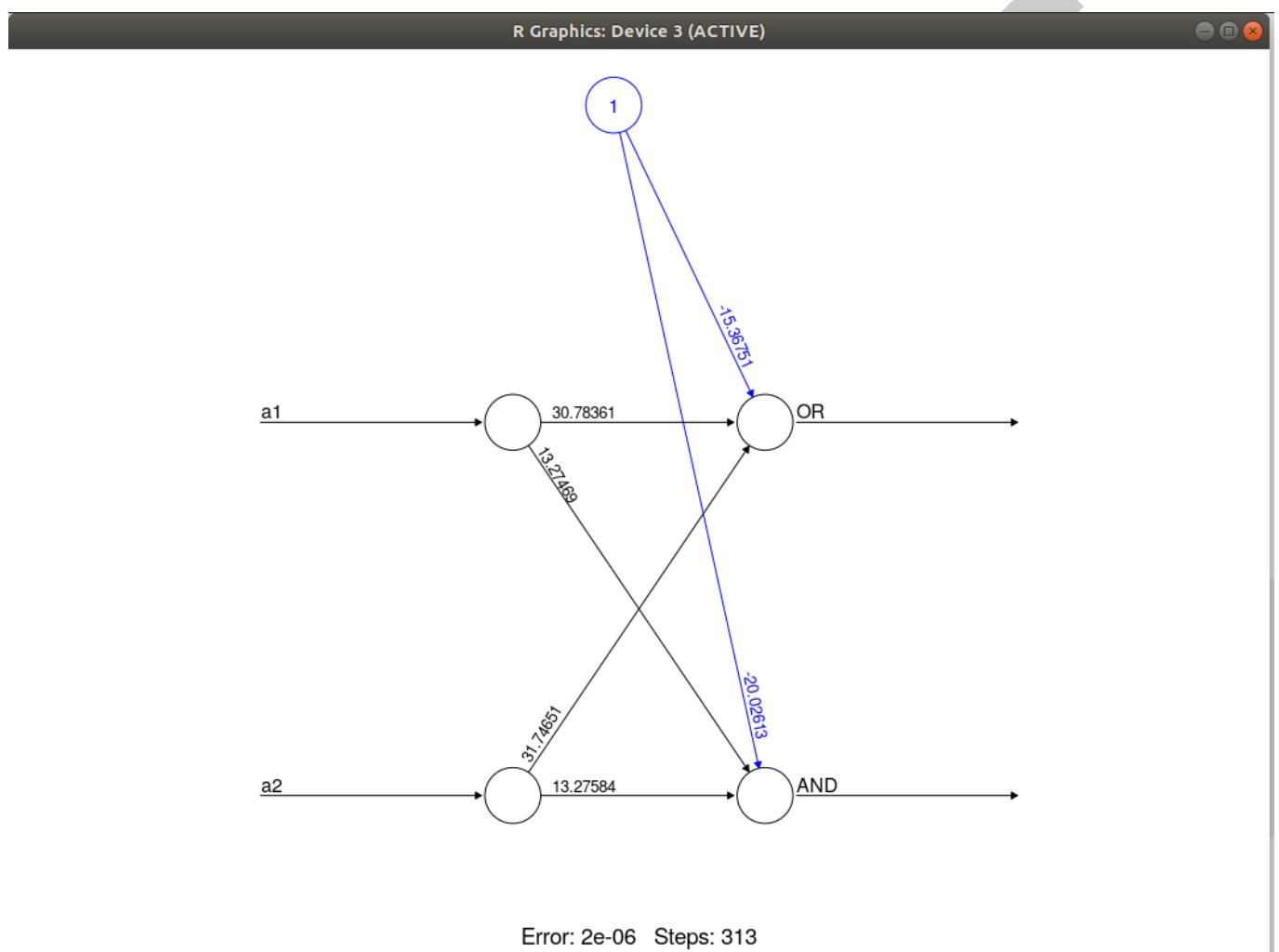
```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR     <- c(0,1,1,1)
AND    <- c(0,0,0,1)

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

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

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

Itt annyi különbség történik, hogy felvesszük mellé az AND műveletet is.



3.6. ábra. NN

```
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)

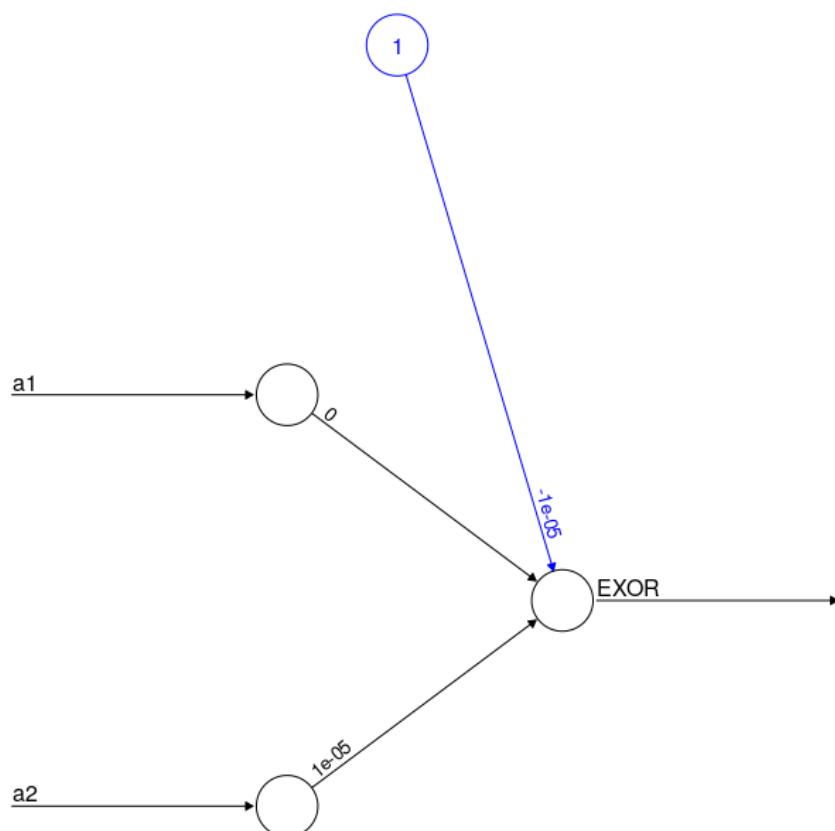
nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=0, linear.output=FALSE, ←
  stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

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

Az EXOR művelet betanítása ugyanazzal a módszerrel nem működik, a sikeres tanításhoz több rétegre van szükség.

R Graphics: Device 2 (ACTIVE)



Error: 0.5 Steps: 92

3.7. ábra. NN

```
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)

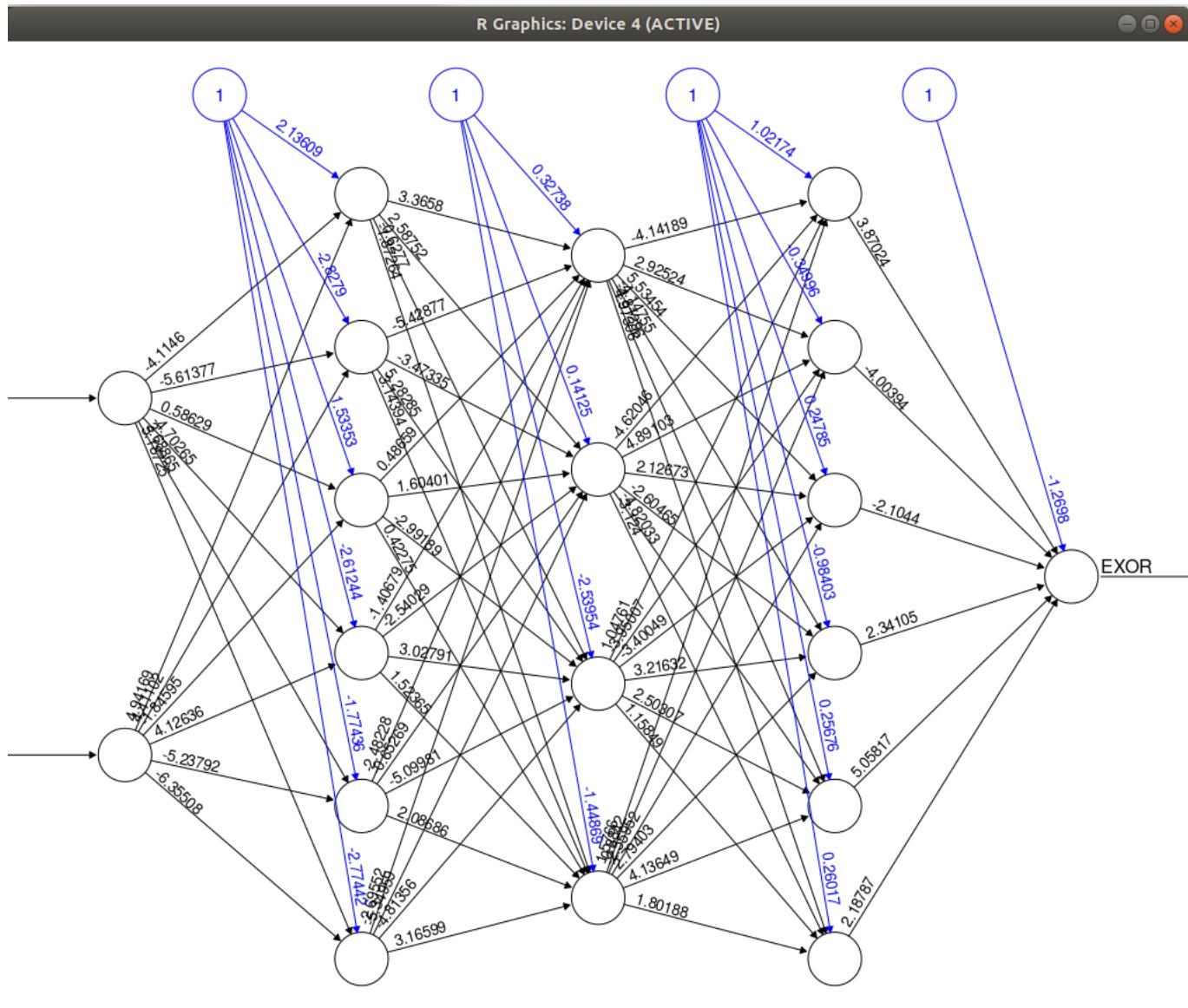
nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ←
  output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

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

Itt már használunk rejtett neuronokat, így már sikeres lesz a tanítás:

DRAFT



3.8. ábra. NN

3.6. Hiba-visszaterjesztéses perceptron

C++

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

mlp.hpp (tartalmazza a perceptron osztályt) forrása: <https://github.com/BorvizRobi/vegyes/blob/master/mlp.hpp>

main.cpp forrása: <https://github.com/BorvizRobi/vegyes/blob/master/main.cpp>

mandel.cpp forrása: <https://github.com/BorvizRobi/vegyes/blob/master/mandelpngt.c%2B%2B>

A gépi tanulásban a perceptron egy algoritmus, a bináris osztályozás tanítására. A bináris osztályozás azt jelenti, hogy az osztályozandó elemeket két részre osztjuk pl.: vagy benne van valami egy adott osztályban vagy nincs. A következő programban ilyen algoritmust fogunk használni.

A nahshon-ból kivettük a perceptron osztályt ezt fogja tartalmazni a mlp.hpp ezt használjuk, valamint a mandel.cpp-t a kép elkészítéséhez és a main.cpp-t. A mandel.cpp-t a következő fejezetben fogjuk részletebben tárgyalni, itt csak a kép készítése lesz a feladata. A perceptron osztályt külön fájlban való tárolásával érjük el hogy átláthatób legyen a main.

A mandel.cpp-t fodítjuk:

```
g++ mandel.cpp -lpng16 -o3 -o mandel
```

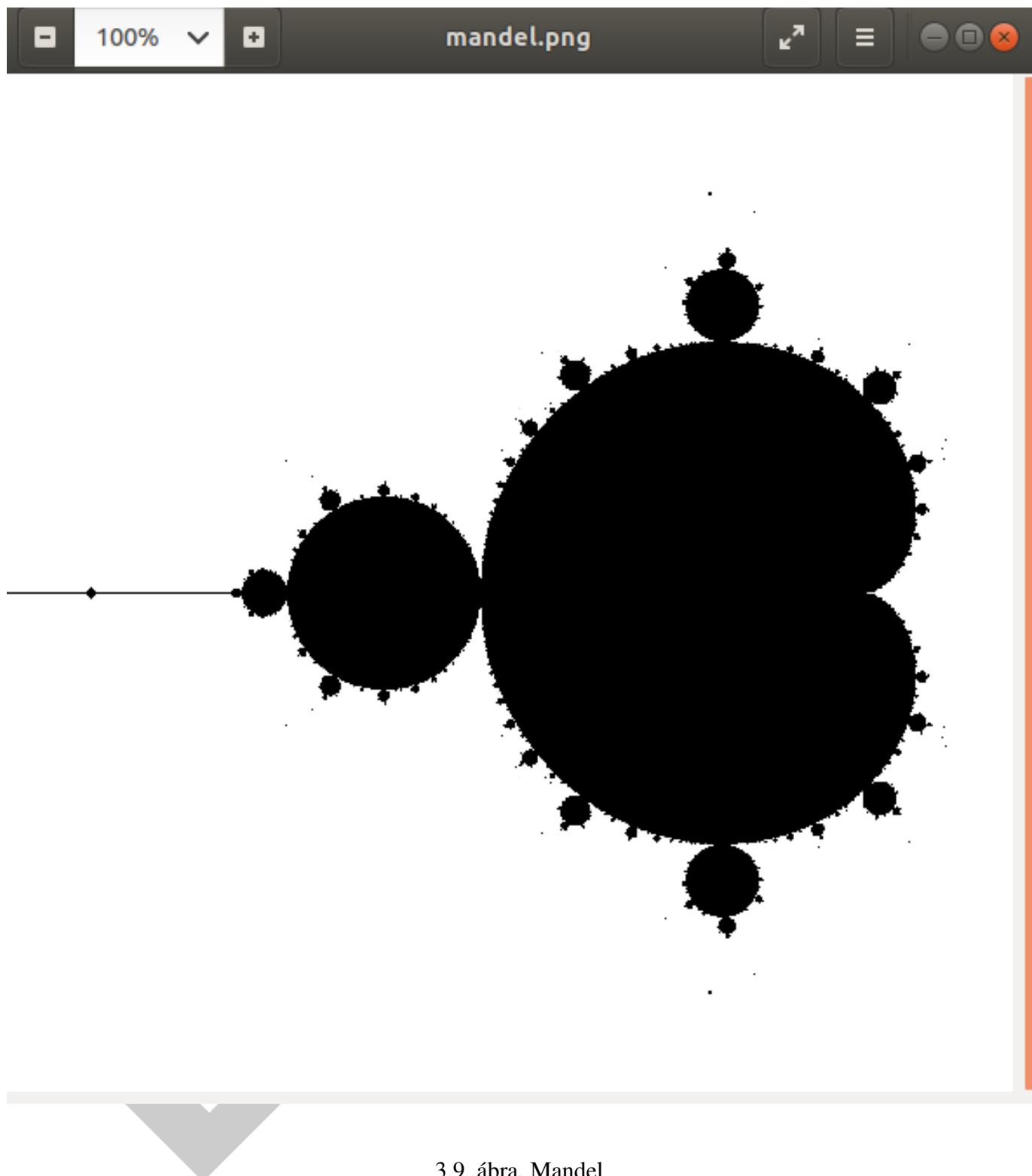
Itt újdonság a -o3 kapcsoló, ami optimalizálást jelent 3. szinten, vagyis a fordítóprogram úgy próbálja fordítani a programot, hogy az minél jobb teljesítménnyel fusson. Az ilyen típusú fordítás viszont növelheti a fordításhoz szükséges időt. A -lpng16 kapcsolóval linkeljük a libpng könyvtárat.

Kép készítése a mandel.cpp-vel:

```
./mandel mandel.png
```

Itt a mandel.png a készítendő képnek a neve.

A mandel programunk a következő képet fogja elkészíteni:



3.9. ábra. Mandel

Ez a kép lesz a bemenete a programunknak, ennek a képnak a pixeleinek az rgb kódjaival fog számításokat végezni.

Ezután fordítjuk a programot:

```
g++ mlp.hpp main.cpp -o perc -lpng -std=c++11
```

-lpng kapcsoló szükséges, mert használjuk a png.hpp fájlt, -std=c++11 kapcsoló a megfelelő szabvány szerinti fordításhoz.

A libpng++ ami tartalmazza a png.hpp fájlt a következő parancsal telepíthető:

```
sudo apt-get install libpng++-dev
```

main.cpp:

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

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

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

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

    double* image = new double[size];

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

    double value = (*p) (image);

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

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

Nézzük részenként:

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

Tárralmazzuk a iostream-et, az input output kezeléséhez, a mlp.hpp valamint a png.hpp-t.

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

Rögzítjük a parancssori argumentumokat.

```
png::image<png::rgb_pixel> png_image (argv[1]);
```

Ezzel a sorral létrehozunk egy üres képet, amelynek a mérete megegyezik a bemenetként megadott kép méretével.

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

A size nevű változóban eltároljuk a kép méretét.

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

Létrehozzuk a Perceptron osztályt, a 'p' pointer egy Perceptron-ra mutató pointer lesz, a new-al lefoglaljuk a szükséges helyet a memóriában a Perceptron (3, size, 256, 1) által létrehozott Perceptron osztályunk tárolásához. Első argumentumnak a rétegek számát adjuk meg ez itt 3 lesz, majd a size nevű változónkat amiben a kép mérete van tárolva, majd azt adjuk meg hogy hány neuront szeretnénk az egyes rétegekben, a 4. helyre pedig 1 -et írunk, ez az 1 fogja az eredményt visszaadni.

```
double* image = new double[size];
```

Létrehozzuk egy image nevű double pointert ami, a memóriában egy size-méretű helyre mutat.

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

Egymásba ágyazott for ciklusokkal bemásoljuk a lefoglalt tárhelybe a bemenetként megadott kép pixeljeinek piros komponensét.

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

std::cout << value << std::endl;
delete p;
delete [] image;
```

Mehívjuk a Perceptron classunk () operátorát, amely által visszaadott eredményt a value nevű változónkba tároljuk. Ezután egyszerűen kiíratjuk az eredményt, majd szabadítjuk a pointerek által mutatott memóriaterületet.

A program a következő eredményeket adta:

```
robi@robi-desktop: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
robi@robi-desktop:~/Asztal$ ./perc mandel.png
0.731054
robi@robi-desktop:~/Asztal$ ./perc mandel.png
0.700214
robi@robi-desktop:~/Asztal$ ./perc mandel.png
0.637075
robi@robi-desktop:~/Asztal$ ./perc mandel.png
0.686352
robi@robi-desktop:~/Asztal$
```

3.10. ábra. A visszaadott eredmények.

4. fejezet

Helló, Mandelbrot!

4.1. A Mandelbrot halmaz

Tutor:

Tóth Csaba:<https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeaTjhhiMRoPXff9ftfWW8Uk9M1QM>

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/mandelpngt.c%2B%2B>

A Mandelbrot-halmaz ábrázolásához nem elég a valós számok halmaza, a komplex számok halmazán kell dolgoznunk. Az $a+bi$ alakú számokat, ahol a és b eleme a valós számok halmazának és $i^2 = -1$ komplex számoknak nevezzük. A mandelbrot halmaz azokból a c komplex számokból áll amelyekre a következő x_n rekurzív sorozat nem tart a végtelenbe:

$$z_{n+1} = (z_n)^2 + c$$

Ha a Manderbrot halmazt a komplex számsíkon ábrázoljuk, akkor egy nevezetes fraktál alakzatot kapunk, a fraktálok végtelenül komplex geometriai alakzatok, egyik jellemző tulajdonságuk hogy határól vonalaik végtelenül gyűröttek, szakadásosak.

A Manderbrot halmazt úgy ábrázolhatjuk program segítségével, hogy például a sík origója középpontú négyzetbe lefektetünk egy 600x600-as rácsot és ennek a rácsnak minden pontját megvizsgáljuk a képlet segítségével úgy, hogy c a vizsgálandó rácspont és z_0 az origó, vagyis elindulunk az origóból és átmegyünk a rács első pontjára $z_1 = c$ aztán c től függően a további z -kre. Ha ez a rekurzív sorozat kivezet a körből akkor a vizsgált pont nem a Manderbot halmaz eleme, nem tudunk végtelen sok z -t megnézni ezért véges sok z elemet nézünk meg minden rácsponthoz.

A program futtatásához a libpng csomag szükséges, (Portable Network Graphics library) ez egy könyvtár ami C-s funkciókat tartalmaz PNG képek kezeléséhez. A következő parancssal tudjuk telepíteni:

```
sudo apt-get install libpng-dev
```

A programunk egy png képet fog készíteni, ehhez a módosított BSD licenccel rendelkező png++ csomagot használjuk ami egy C++ wrapper a libpng csomagunkhoz, ezt a következő linkről tudjuk letölteni:

png++ link: <http://savannah.nongnu.org/projects/pngpp/>

Létrehozunk egy png++ nevű mappát a home mappánkban, majd itt kicsomagoljuk és a következő parancssal telepítjük:

```
sudo make install
```

Ezután a következő parancssal tudjuk fordítani a programunkat:

```
g++ mandel.cpp -lpng16 -o3 -o mandel
```

Itt újdonság a -o3 kapcsoló, ami optimalizálást jelent 3. szinten, vagyis a fordítóprogram úgy próbálja fordítani a programot hogy az minél jobb teljesítménnyel fusson. Az ilyen típusú fordítás viszont növelheti a fordításhoz szükséges időt. A -lpng16 kapcsolóval linkeljük a libpng könyvtárat.

Majd így futatjuk:

```
./mandel fájlnév.png
```

Itt a fájlnévnek, azt adjuk meg, amit a készítendő kép nevének szeretnénk kapni.

Teljes forráskód:

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

#define MERET 600
#define ITER_HAT 32000

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

    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

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

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, rez, imZ, ujrez, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rácson 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 elértek, akkor úgy vesszük,
// hogy a kiinduláci c komplex számra
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while (reZ * reZ + imZ * imZ < 4 && iteracio < 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;

}
```

Nézzük részenként:

```
#include "png++/png.hpp"
```

Ezzel tesszük elérhetővé az összes header fájlt a png++-ból.

```
#include <sys/types.h>
```

Az idő kezeléséhez:

```
#define MERET 600
#define ITER_HAT 32000
```

Nevesített konstansok a kép mérete 600x600 lesz az iterációs határ pedig 32000, ennyiszer fogjuk iterálni a számítást mielőtt eldöntjük hogy az adott pont a halmaz része-e.

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

Ezt a funkciót használjuk a számoláshoz egy MERETxMERET int tömböt kér be, az lesz a kép mérete itt ez 600x600 lesz.

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

Megadjuk a számítás adatait, azt a tartományt amin a függvény értelmezni fogjuk, a szélességet és magasságot, valamint az iterációs határt.

```
// a számítás
float dx = (b - a) / szelesseg;
float dy = (d - c) / magassag;
float reC, imC, reZ, imZ, ujreZ, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
```

Lépésköz kiszámolása, deklaráljuk a változókat amiben a komplex szám valós és képzetes részét fogjuk tárolni, iteráció-ban fogjuk majd tárolni hogy hanyadik iterációban járunk éppen.

```
// Végigzongorázzuk a szélesség x magasság rácson:
for (int j = 0; j < magassag; ++j)
{
    //sor = j;
    for (int k = 0; k < szelesseg; ++k)
    {
        // c = (reC, imC) a rácson 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 kiinduláció 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;
}
}
```

Végig haladunk a rács pontjain két for ciklus segítségével, kiszámoljuk a rács pontjának megfelelő komplex számot majd egy while ciklussal ellenőrizük hogy a Mandelbro halmaz számítására használható képlet segítségével, hogy az adott szám benne van-e a halmazban, majd kepadat[j][k]-adik elemében tároljuk hogy hanyadik iterációra tudtuk ezt eldönteni, ha elértek az iterációs határt akkor, ezt a határszámot tároljuk.

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

A main-ben tároljuk a parancssori argumentumokat argc-ben a számukat, az *argv[] karakter pointer tömbben pedig a karakterszorozatokat.

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

Ha nem adtunk meg fájlnevet ahova a kimenetet készíti akkor kiírjuk a használati utasítást.

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

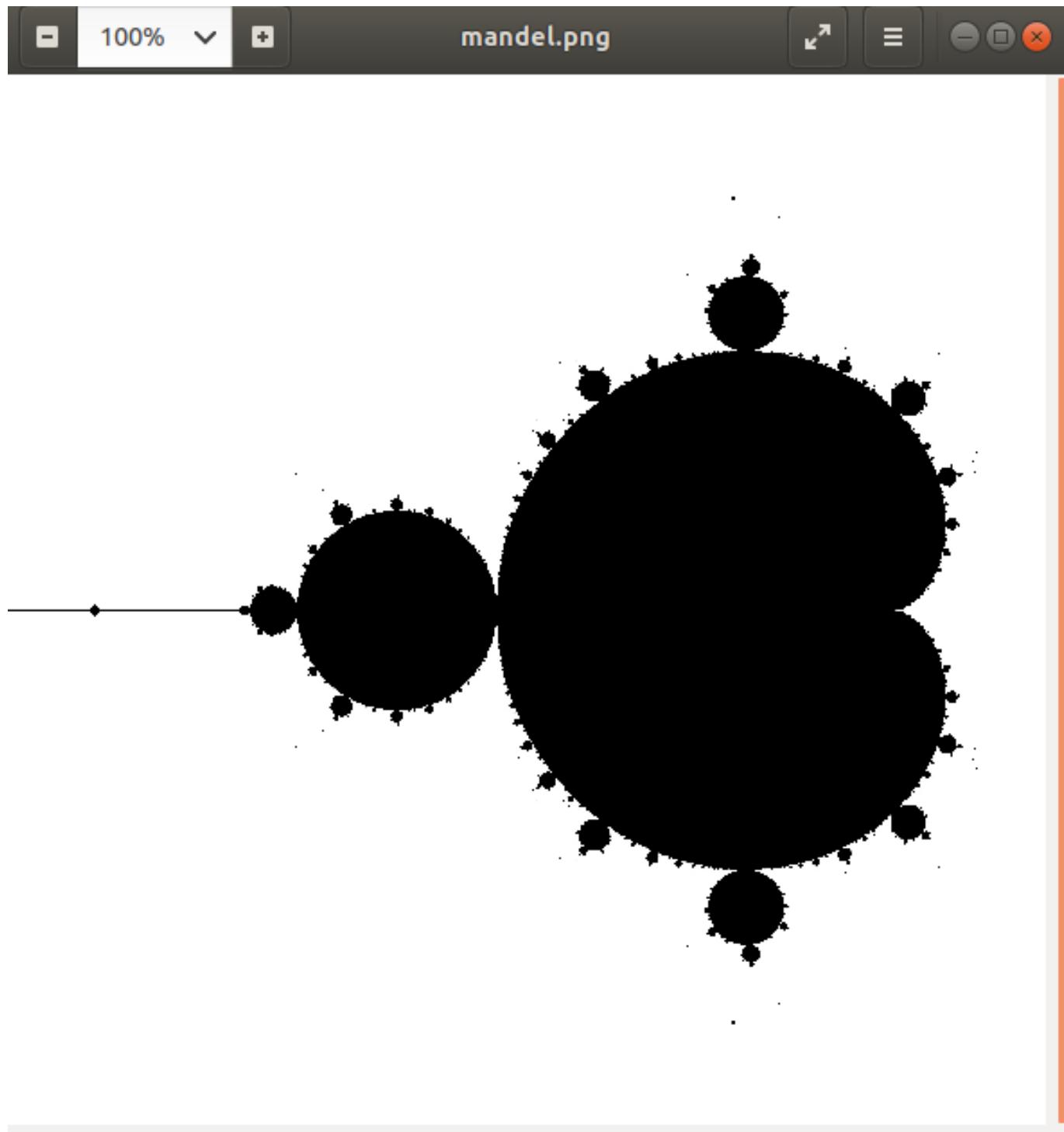
Itt hozunk létre egy üres képet, a megfelelő magasságban és szélességen(MERETxMERET)

```
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;
```

Két egymásba ágyazott for ciklussal feltöljük a kép megfelelő pixelei, majd kiírjuk a kimeneti fájlba az eredményt.

Ezt a képet kapjuk eredményül:



4.1. ábra. Mandel

4.2. A Mandelbrot halmaz a std::complex osztállyal

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

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/mandel3.1.2.cpp>

A következő program is a Mandelbrot halmazt fogja kiszámolni viszont az std::complex osztály használatával, ezáltal a program egyszerűbb lesz az előző programban a komplex számot két részben tároltuk, egy változóban a valós részt és egy külön változóban az imaginárius részt, itt egy egységeként fogjuk tudni kezelni a komplex számot.

A programot a következőképpen fordítjuk:

```
g++ mandel3.1.2.cpp -lpng -O3 -o mandel3.1.2
```

Majd futtathatjuk a következő parancsal:

```
./3.1.2 mandel.png 1920 1080 2040 ←  
-0.01947381057309366392260585598705802112818 ←  
-0.0194738105725413418456426484226540196687 ←  
0.7985057569338268601555341774655971676111 ←  
0.798505756934379196110285192844457924366
```

Itt eltérés a következő programhoz képest hogy megadhatjuk a parancssori argumentumok között a számítási adatokat.

Teljes forráskód:

```
#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] );
```



```
    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;

}
```

Nézzük részenként:

```
#include <complex>
```

Tartalmazzuk a complex library-t.

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

A main-ben argc-vel számoljuk az argumentumok számát és *argv[] tömbben tároljuk azokat.

```
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;
```

Beállítjuk az alapértékeket.

```
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] );
}
```

linux Ha a parancssori argumentumok száma megegyezik 9-el, akkor ezeket az argumentumokat használ-juk értékként, itt atof() funkció segítségével a megadott "string" argumentumot double értékké alakítjuk, valamint atoi() funkció segítségével int értékké ezután hozzárendeljük öket a megadott változókhöz.

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

Ha a parancssori argumentumok száma nem eggyezik meg 9-el, akkor kiírjuk a használati utasítást.

```
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;
```

Itt is mint az előző programban létrehozzuk az üres képet, megadjuk a lépésközt és deklaráljuk a változókat.

```
// 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;  
}
```

Két for ciklus segítségével végigmegyünk a rács pontjain, itt viszont az előző programmal ellentétben használjuk a complex tipust, aminek két double megadásával tudjuk megadni az értékét, az első a valós a második a képzetes rész, c és z_n itt ilyen komplex tipusú változók lesznek. A while ciklusban ellenőrizzük hogy benne van-e a halmazban a megfelelő pont, ezután beállítjuk a kép megfelelő pixeljét, de itt már szinezést is alkalmazunk

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

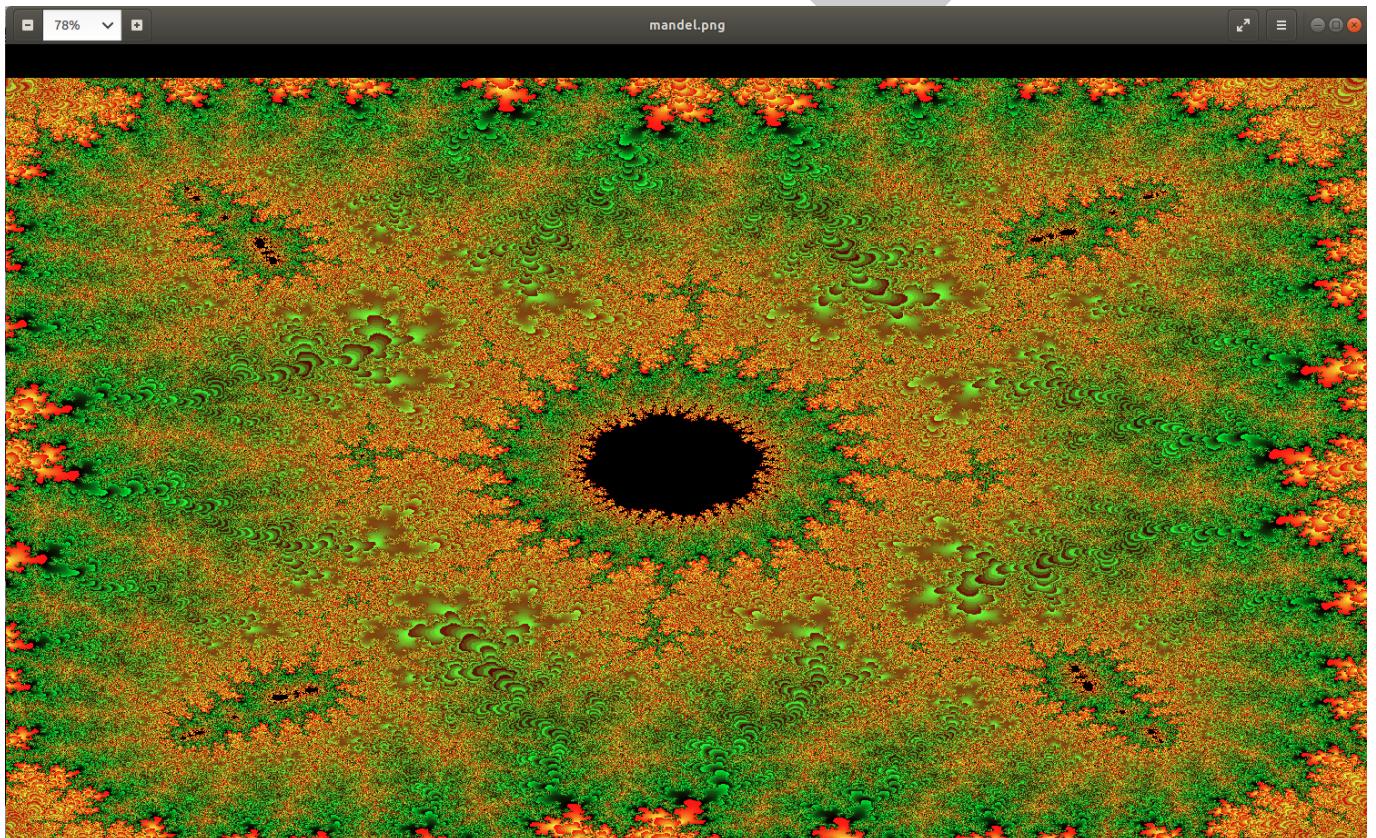
Miközben a program dolgozik itt folyamatosan kiíratjuk hogy hány százaléknál járunk a flush funkció segítségével érjük el hogy ne soronként írunk hanem mindenkorán frissítsük az aktuális %-ot.

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

A `kep.write()` funkcióval kiírjuk a képet a 2. argumentumként megadott fájlba.

```
./3.1.2 mandel.png 1920 1080 2040 ←
-0.01947381057309366392260585598705802112818 ←
-0.0194738105725413418456426484226540196687 ←
0.7985057569338268601555341774655971676111 ←
0.798505756934379196110285192844457924366
```

Ezekkel a számítási adatokkal futtatva a következő képet kapjuk:



4.2. ábra. Mandel

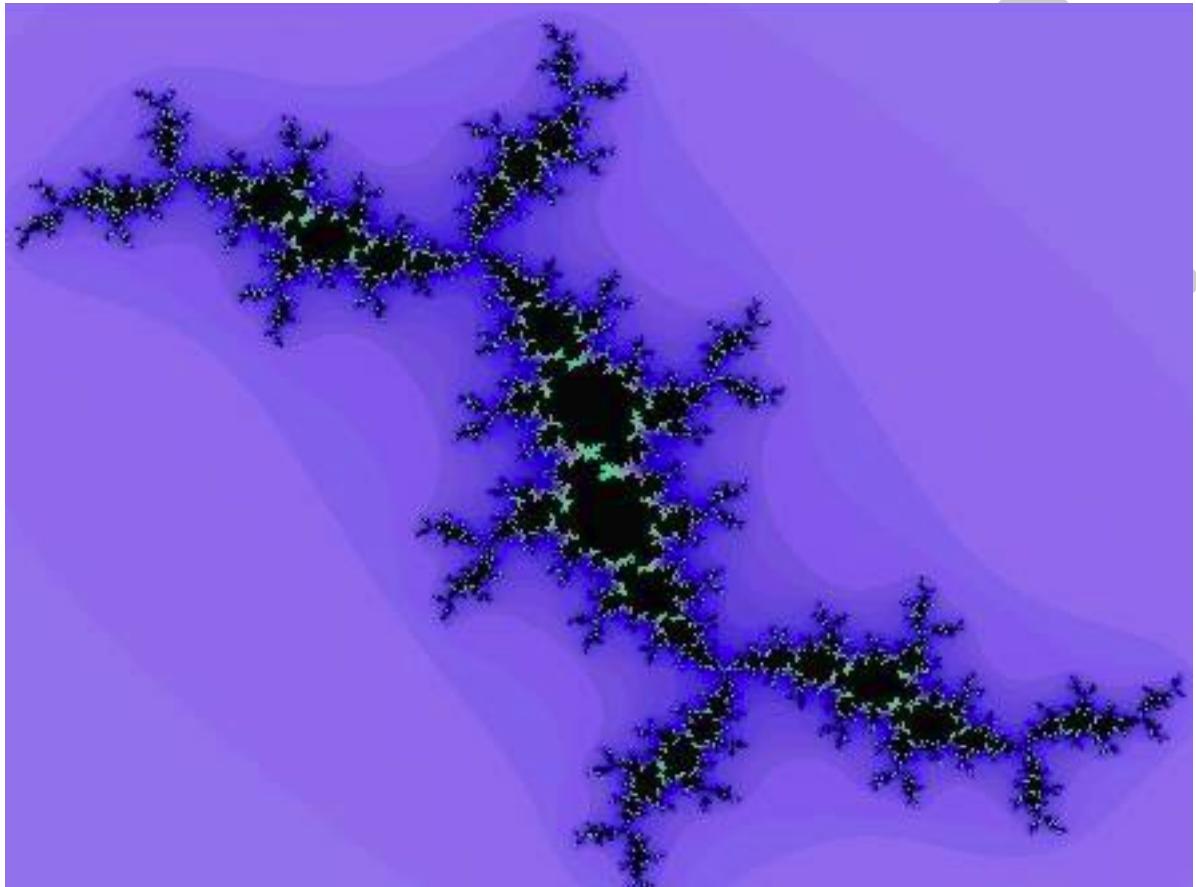
4.3. Biomorfok

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/biomorf.cpp>

Gaston Julia és Pierre Fauto francia matematikusok kutatásai során: komplex egyenletek megoldása iterációval, megoldáshalmaz ábrázolása a kplex számsíkon, különös halmazokat kaptak eredményűl, ezeket a későbbiekben Julia halmazoknak nevezték el.

Példa egy ilyen Julia halmazra:



4.3. ábra. Julia halmaz

A Julia halmazokat rajzoló bugsos programjával Clifford Pickover találta meg a biomorfokat, aki azt hitte természeti törvényre bukkant.

A Mandelbrot halmazt számoló program átalakítása a következő program, ezért ebben a részben csak a különbségeket fogom bemutatni. Az egyik különbség hogy itt a komplex iterációban a c állandó érték lesz.

Így fordítjuk:

```
g++ biomorf.cpp -lpng -O3 -o biomorf
```

Majd a következő parancssal futtatható:

```
./biomorf bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
```

Különbség az előző programhoz képest, hogy itt futtatáskor megadjuk az iterációs határt és a c konstanst, valamint a kör sugarát amin belül vizsgáljuk a konverenciát.

Teljes forráskód:

```
#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;

}
```

Vegyük sorra a különbségeket:

```
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] );
}
```

Mivel több argumentumot kérünk be ezért többet is tárolunk, reC a c konstansunk valós része, imC a képzetes része, R pedig a kör sugara.

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

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

for ( int y = 0; y < magassag; ++y )
{
    // ki 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 < iteracionsHatar; ++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 ) );
    }

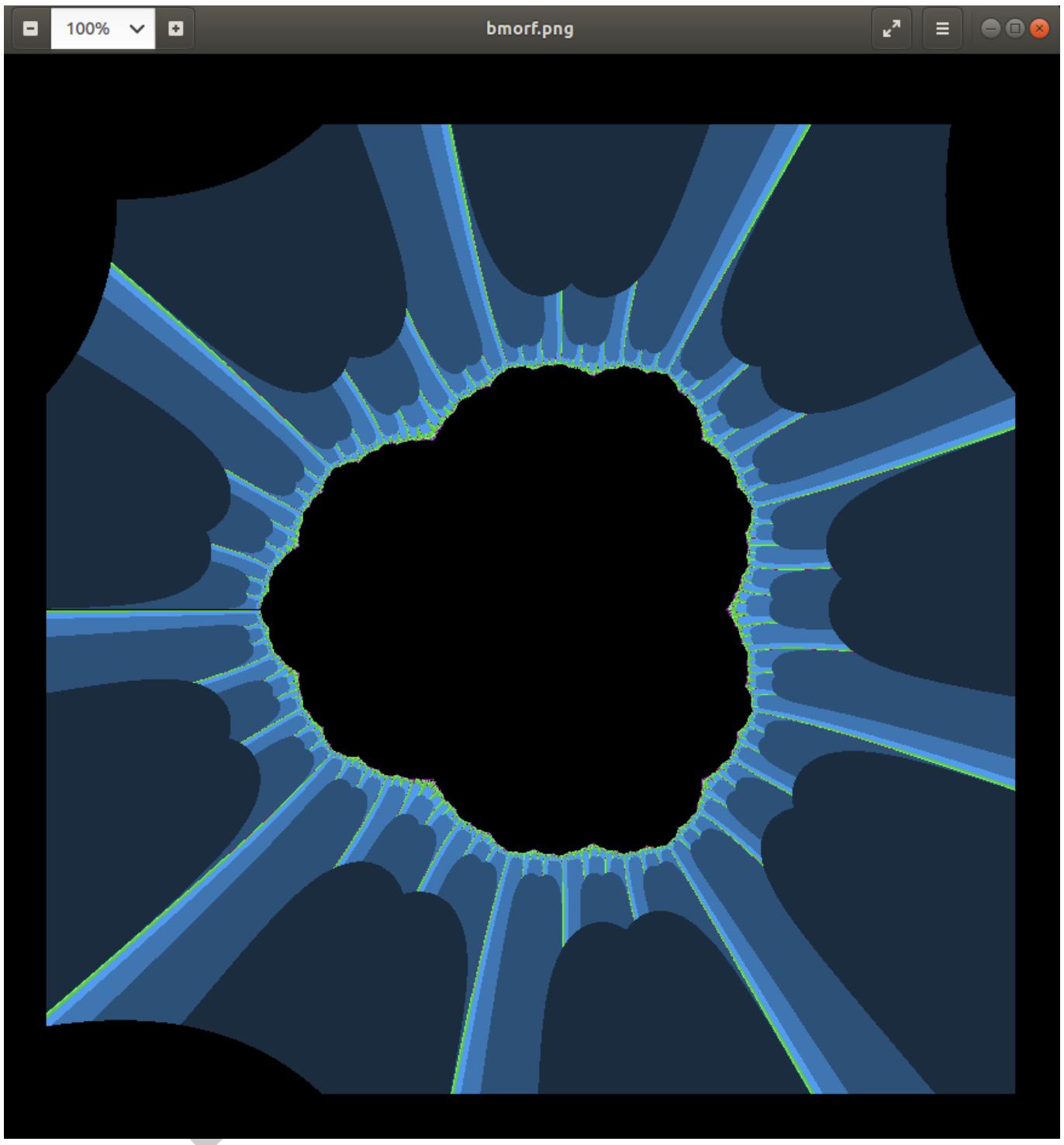
    int szazalek = ( double ) y / ( double ) magassag * 100.0;
    std::cout << "\r" << szazalek << "%" << std::flush;
}
```

Itt végezzük el a lényegi számítást, az előző programmal elentében itt c (programkódban cc) értéke nem változik, végighaladunk a rács pontjain, miután megállapítottuk hogy az adott pont az iterációs határon belül benne marad az egység sugarú körben, beállítjuk a pixeleket majd kiíratjuk a képet.

Majd a következő parancssal futtatva:

```
./biomorf bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
```

A program ezt a képet készíti el:



4.4. ábra. Biomorf

4.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/mandelcuda.cpp>

A következő programban a Mandelbrot halmazt számoló program CUDA-s változatát fogom bemutatni. A program futtatásához NVIDIA grafikus kártya szükséges, valamint a NVIDIA CUDA Toolkit , a szükséges információk, használati utasítás elérhető az alábbi linken:

<https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html>

A CUDA egy párhuzamos számítási platform és programozási modell amit az NVIDIA fejlesztett ki, jelen-tős teljesítménynövekedést tud elérni a GPU erejének kihasználásával. A CUDA technológiával rendelkező GPU-kban több száz mag található, amik több ezer szálat képesek futtatni egyszerre.

Teljes forráskód:

```
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>

#include <sys/times.h>
#include <iostream>

#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:
    // most eppen a j. sor k. oszlopban vagyunk

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

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;

    // c = (reC, imC) a rács csomópontjainak
    // megfelelő komplex szám
    reC = a + k * dx;
    imC = d - j * dy;
    // z_0 = 0 = (reZ, imZ)
    reZ = 0.0;
```

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

    ++iteracio;
}

return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{
    int j = blockIdx.x;
    int k = blockIdx.y;
    kepadat[j + k * MERET] = mandel (j, k);
}
*/

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

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

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

void
cudamandel (int kepadat[MERET] [MERET])
```



```
    255 -  
    (255 * kepadat[j][k]) / ITER_HAT));  
}  
}  
kep.write(argv[1]);  
  
std::cout << argv[1] << " mentve" << std::endl;  
  
times(&tmsbuf2);  
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime  
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;  
  
delta = clock() - delta;  
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;  
  
}
```

A programot sajnos nem tudtam futtatni mert nincs NVIDIA kártyám, de a CUDA technológia miatt ez a változat az eredetitől sokkal gyorsabban fog lefutni.

4.5. 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: <https://github.com/BorvizRobi/mandelnagyit>

A GUI Qt segítségével lesz elkészítve, ami egy keresztpalatformos alkalmazás-keretrendszer, amit GUI-s alkalmazások, illetve nem GUI-s programok fejlesztésére használnak.

A Qt részletes leírása ezen a linken megtalálható: <https://hu.wikipedia.org/wiki/Qt>

A Qt innen tölthető le: <https://www.qt.io/>

A mappában, ami tartalmazza a forráskódokat kiadjuk a következő parancsot:

```
~/Qt/5.12.2/gcc_64/bin/qmake -project
```

Értelemszerűen, onnan indítjuk a qmake parancsot, ahova telepítve van a Qt, a -project kapcsolóval létrehozzuk a project fájlt. Ez a fájl fogja tartalmazni azokat az információkat amelyek a qmake számára szükségesek a program létrehozásához.

Majd erre a létrehozott projekt fájlra ismételten futtatjuk a qmake-t:

```
~/Qt/5.12.2/gcc_64/bin/qmake mandelnagyit.pro
```

A qmake egy Makefile-t fog készíteni nekünk a projektfájlból található infomációk alapján, majd a "make" parancsot kiadva elkészül a futatható állományunk amit a következő parancssal futathatunk:

```
./mandelnagyit
```

Nézzük a forráskódokat:

A kódjaink, ebben az esetben külön vannak választva .h (header) és .cpp kiterjesztésű fájlokra. A header fájlok tartalmazzák a deklarációkat és a .cpp kiterjesztésű fájlok a definíciókat, például a frakablak.h tartalmazza a frakablak class-unk deklarációt a frakablak.cpp pedig ezeknek a deklarációknak a definíciót, ezután az #include-t használva tartalmazzuk a header fájlt, az #include annyit csinál hogy bemásolja a deklarációkat abba a sorba ahol van. Ez a kettévalasztás gyakran alkamazott technika C-ben és C++-ban.

main.cpp:

```
#include <QApplication>
#include "frakablak.h"

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

    FrakAblak w1;
    w1.show();

    return a.exec();
}
```

Tartalmazzuk a QApplication-t, valamint a frakablak.h header fájlt, a mainban rögzítjük a parancssori argumentumokat.

```
QApplication a(argc, argv);
```

Ez a sor meghívja QApplication konstruktőrét, amely létrehoz egy QApplication típusú 'a' nevű objektumot az argumentumaként megadott argc és argv-al.

```
FrakAblak w1;
w1.show();
```

Létrehozzuk a fő ablakunkat w1 névvel, majd a .show() funkciójával láthatóvá tesszük.

A mainben FrakAblak nevű objektumot hoztunk létre, ami egy QMainWindow típusú objektumtól örököl, a QMainWindow nyújtja számunkra a főablakot, és a kezeléshez szükséges funkciókat, úgyhogy most nézzük meg a frakablak.cpp-t:

```
#include "frakablak.h"

FrakAblak::FrakAblak(double a, double b, double c, double d,
                      int szelesseg, int iteraciosHatar, QWidget *parent)
    : QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");

    szamitasFut = true;
    x = y = mx = my = 0;
    this->a = a;
    this->b = b;
```

```
this->c = c;
this->d = d;
this->szelesseg = szelesseg;
this->iteraciosHatar = iteraciosHatar;
magassag = (int)(szelesseg * ((d-c) / (b-a)));

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

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

FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}

void FrakAblak::paintEvent(QPaintEvent*)
{
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);

    }
    qpainter.end();
}

void FrakAblak::mousePressEvent(QMouseEvent* event)
{
    // A nagyítandó kijelölt területet bal felső sarka:
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;

    update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event)
{
    // A nagyítandó kijelölt terület szélessége és magassága:
    mx = event->x() - x;
    my = mx; // négyzet alakú

    update();
}
```

```
void FrakAblak::mouseReleaseEvent (QMouseEvent* event) {  
  
    if(szamitasFut)  
        return;  
  
    szamitasFut = true;  
  
    double dx = (b-a)/szelesseg;  
    double dy = (d-c)/magassag;  
  
    double a = this->a+x*dx;  
    double b = this->a+x*dx+mx*dx;  
    double c = this->d-y*dy-my*dy;  
    double d = this->d-y*dy;  
  
    this->a = a;  
    this->b = b;  
    this->c = c;  
    this->d = d;  
  
    delete mandelbrot;  
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←  
        iteraciosHatar, this);  
    mandelbrot->start();  
  
    update();  
}  
  
void FrakAblak::keyPressEvent (QKeyEvent *event)  
{  
  
    if(szamitasFut)  
        return;  
  
    if (event->key() == Qt::Key_N)  
        iteraciosHatar *= 2;  
    szamitasFut = true;  
  
    delete mandelbrot;  
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←  
        iteraciosHatar, this);  
    mandelbrot->start();  
  
}  
  
void FrakAblak::vissza(int magassag, int *sor, int meret)  
{  
    for(int i=0; i<meret; ++i) {
```

```
    QRgb szin = qRgb(0, 255-sor[i], 0);
    fraktal->setPixel(i, magassag, szin);
}
update();
}

void FrakAblak::vissza(void)
{
    szamitasFut = false;
    x = y = mx = my = 0;
}
```

Kezdésnek include-oljuk a frakablak.h header fájlt, majd a konstruktur definiálása következik.

```
FrakAblak::FrakAblak(double a, double b, double c, double d,
                      int szelesseg, int iteraciosHatar, QWidget *parent)
: QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");

    szamitasFut = true;
    x = y = mx = my = 0;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    magassag = (int)(szelesseg * ((d-c) / (b-a)));

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

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

A konstruktörben beállítjuk a Mandelbrot halmaz tartományát, megadjuk ablak címének a "Mandelbrot halmaz"-t, valamint beállítjuk hogy, az ablak ne legyen újraméretezhető. Helyet foglalunk a memóriában a FrakSzal osztálynak valamint a Qt által biztosított QImage osztálynak. A QImage osztály fogja számunkra biztosítani a képek kezeléséhez szükséges funkciókat.

```
FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}
```

Mivel mi foglaltunk helyet a memóriában az előzőekben említett objektumoknak, ezért a destruktörben szabadítjuk őket.

```
void FrakAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);

    }
    qpainter.end();
}

void FrakAblak::mousePressEvent(QMouseEvent* event) {

    // A nagyítandó kijelölt területet bal felső sarka:
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;

    update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event) {

    // A nagyítandó kijelölt terület szélessége és magassága:
    mx = event->x() - x;
    my = mx; // négyzet alakú

    update();
}

void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {

    if(szamitasFut)
        return;

    szamitasFut = true;

    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;

    double a = this->a+x*dx;
    double b = this->a+x*dx+mx*dx;
    double c = this->d-y*dy-my*dy;
    double d = this->d-y*dy;

    this->a = a;
```

```
this->b = b;
this->c = c;
this->d = d;

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

update();
}

void FrakAblak::keyPressEvent (QKeyEvent *event)
{

    if (szamitasFut)
        return;

    if (event->key() == Qt::Key_N)
        iteraciosHatar *= 2;
    szamitasFut = true;

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

}
```

Itt vannak definiálva a különböző események(Event-ek) kezelése. A paintEvent fogja kivinni a képernyőre az elvégzett számítás eredményét, ezeket a számításokat külön szálon végezzük, hogy ne blokkolják az ablakot. Itt van kezelve a feladat 2. része is vagyis ha a felhasználó jobb egérgombbal kattint akkor kirajzolja a program az oda vezető utat, azaz összeköti egy vonallal a számítás során kiszámolt komplex számokat.

Ezután az egérrel kapcsolatos események kezelése következik. A programnak figyelnie kell a bal egérgomb lenyomását, lenyomva tartását majd elengedését, ez fogja kijelölni a nagyítandó területet. Ha a jobb egérgombot nyomjuk le, akkor a kiválasztott ponthoz vezető utat fogja a program kirajzolatni.

```
void FrakAblak::mouseMoveEvent (QMouseEvent* event) {

    // A nagyítandó kijelölt terület szélessége és magassága:
    mx = event->x() - x;
    my = mx; // négyzet alakú

    update();
}

void FrakAblak::mouseReleaseEvent (QMouseEvent* event) {
```

```
if(szamitasFut)
    return;

szamitasFut = true;

double dx = (b-a)/szelessseg;
double dy = (d-c)/magassag;

double a = this->a+x*dx;
double b = this->a+x*dx+mx*dx;
double c = this->d-y*dy-my*dy;
double d = this->d-y*dy;

this->a = a;
this->b = b;
this->c = c;
this->d = d;

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

update();
}
```

A balegér gomb lenyomásának pozíciójának, illetve felengedésének helyét kezeli a mouseMoveEvent és mouseReleaseEvent, amikor elengedjük a gombot a kapott infomációk alapján nagyítjuk a Mandelbrot-halmazt. Majd töröljük a regi számítást tartalmazó mandelbrot nevű objektumot és létrehozunk egy újat az újonnan rögzített paraméterekkel.

```
void FrakAblak::keyPressEvent (QKeyEvent *event)
{
    if(szamitasFut)
        return;

    if (event->key() == Qt::Key_N)
        iteraciosHatar *= 2;
    szamitasFut = true;

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

}
```

```
void FrakAblak::vissza(int magassag, int *sor, int meret)
{
    for(int i=0; i<meret; ++i) {
        QRgb szin = qRgb(0, 255-sor[i], 0);
        fraktal->setPixel(i, magassag, szin);
    }
    update();
}

void FrakAblak::vissza(void)
{
    szamitasFut = false;
    x = y = mx = my = 0;
}
```

A nagyítások elvégzése után az 'n' gomb lenyomásával élesíteni tudjuk a képet ezt úgy értjük el hogy növeljük az iterációshatárt a kétszeresére, ezt a keyPressEvent kezeli. A vissza funkciókat nem használjuk ebben az osztályban, majd a Frakszalban fogjuk ami azt teszi lehetővé hogy frissítsük ezt az osztályt a számításai során.

Nézzük most a frakszal.cpp-t:

```
#include "frakszal.h"

FrakSzal::FrakSzal(double a, double b, double c, double d,
                     int szelessseg, int magassag, int iteraciosHatar, ←
                     FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelessseg = szelessseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;

    egySor = new int[szelessseg];
}

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

void FrakSzal::run()
{
    // A [a,b]x[c,d] tartományon milyen sűrű a
    // megadott szélesség, magasság háló:
```

```
double dx = (b-a)/szelessseg;
double dy = (d-c)/magassag;
double reC, imC, rez, imZ, ujrez, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
// Végigzongorázzuk a szélesség x magasság hálót:
for(int j=0; j<magassag; ++j) {
    //sor = j;
    for(int k=0; k<szelessseg; ++k) {
        // c = (reC, imC) a háló rácspontjainak
        // megfelelő komplex szám
        reC = a+k*dx;
        imC = d-j*dy;
        // z_0 = 0 = (rez, imZ)
        rez = 0;
        imZ = 0;
        iteracio = 0;
        // z_{n+1} = z_n * z_n + c iterációk
        // számítása, amíg |z_n| < 2 vagy még
        // nem értük el a 255 iterációt, ha
        // viszont elértek, akkor úgy vesszük,
        // hogy a kiinduláci c komplex számra
        // az iteráció konvergens, azaz a c a
        // Mandelbrot halma eleme
        while(rez*rez + imZ*imZ < 4 && iteracio < iteraciosHatar) {
            // z_{n+1} = z_n * z_n + c

            ujrez = rez*rez - imZ*imZ + reC;
            ujimZ = 2*rez*imZ + imC;

            rez = ujrez;
            imZ = ujimZ;

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

        iteracio %= 256;

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

```
    }  
    frakAblak->vissza();  
  
}
```

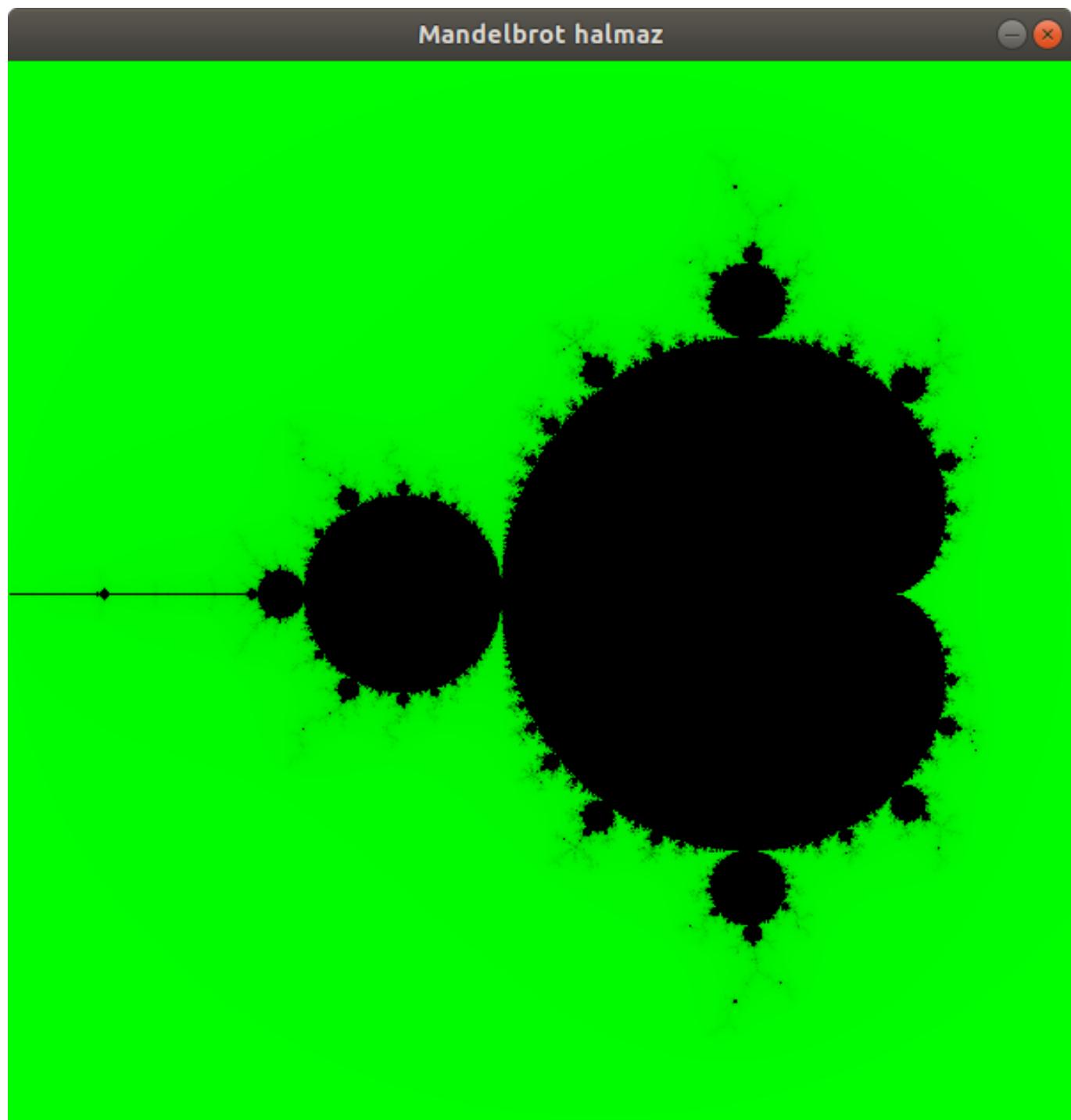
Tartalmazzuk a fraksal.h-t. A konstruktor a Mandelbrot-halmaz számításához szükséges változókat állítja be.

Ez az osztály a Qthread osztályból öröklődött vagyis tartalmazza a run() funkciót, azért örökítettük onnan, hogy a számításokat el tudjuk különíteni a main threadtől így nem blokkolva a main-t.

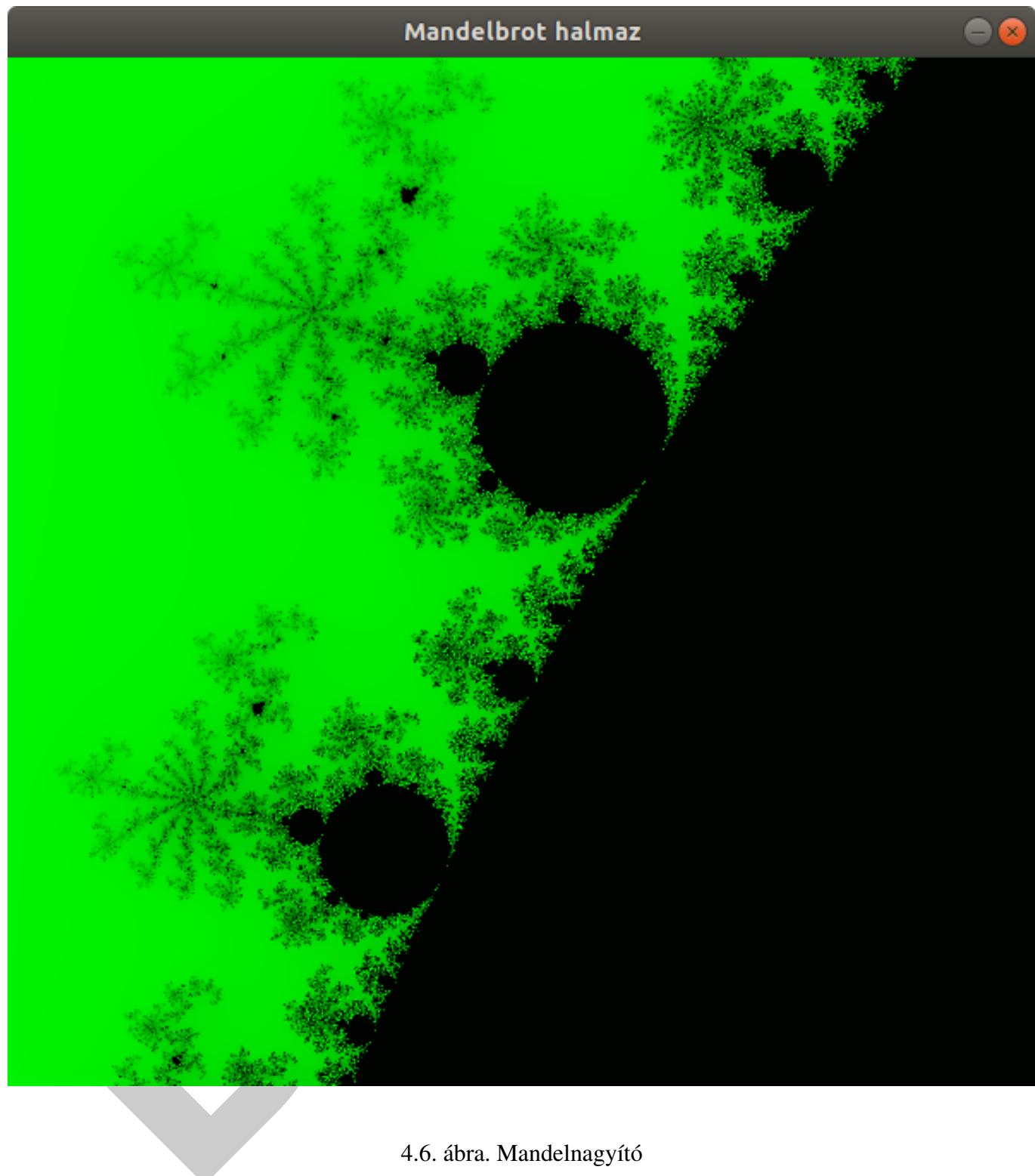
A run végzi a korábbi részekben tárgyalt Mandelbrot számításokat (most külön szalon), a számítások elvezetével értesítjük az argumentumkánt átadott FrakAblak objektumot hogy frissíthesi a tartalmát.

A program működés közben:

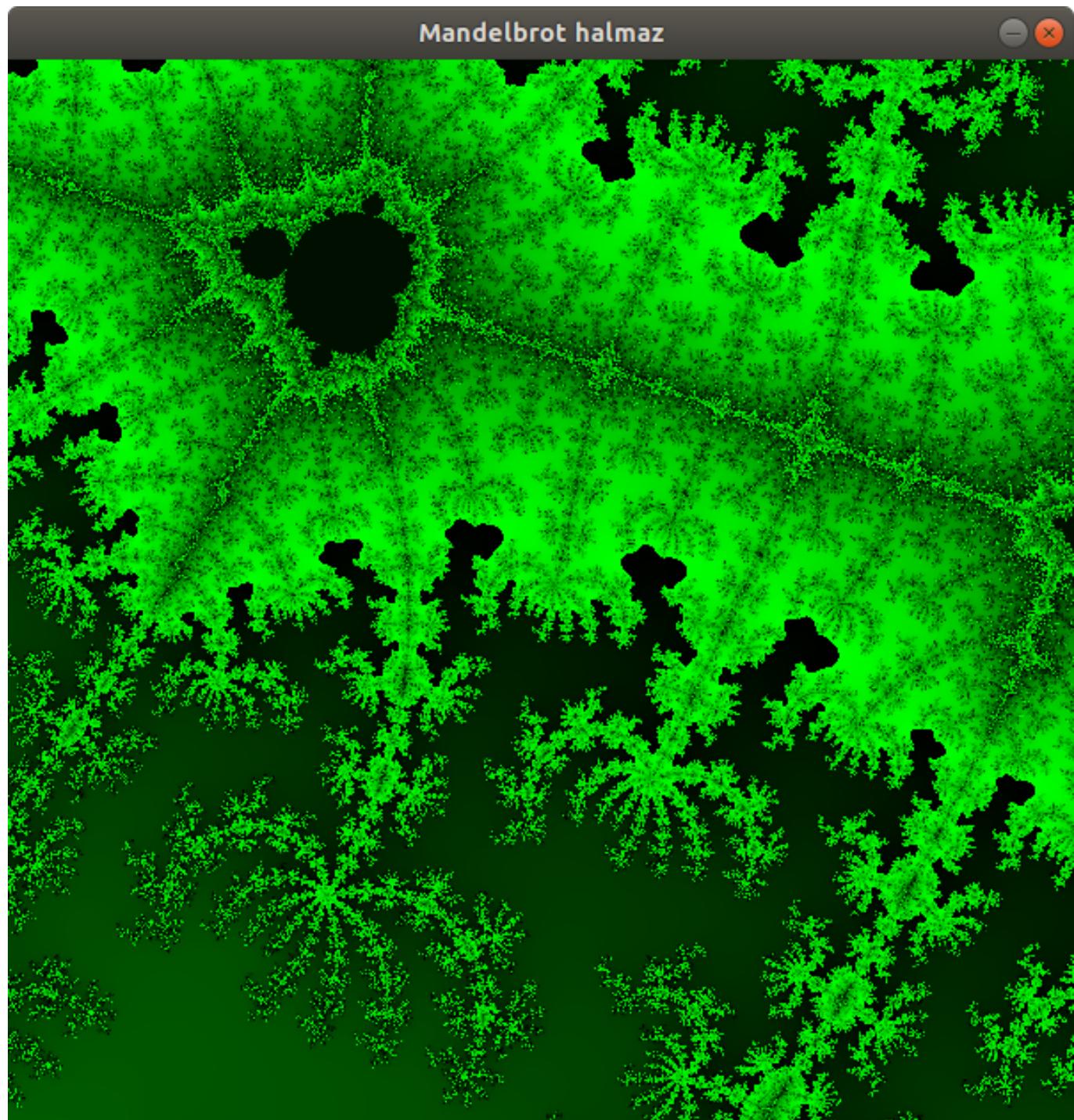
DRAFT



4.5. ábra. Mandelnagyító



4.6. ábra. Mandelnagyító



4.7. ábra. Mandelnagyító

4.6. Mandelbrot nagyító és utazó Java nyelven (Passz)

5. fejezet

Helló, Welch!

5.1. Első osztályom

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/polargen.cpp>

Polártranszformáció segítségével fogunk random számokat generálni, ez az algoritmus annyira népszerű hogy a java random szám generátora is ezt használja.

A C++ Forráskód:

```
class PolarGen
{
public:
    PolarGen()
    {
        nincsTarolt = true;
        std::srand (std::time(NULL));
    }
    ~PolarGen()
    {
    }
    double kovetkezo();

private:
    bool nincsTarolt;
    double tarolt;

};
```

```
double PolarGen::kovetkezo ()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1= std::rand() / (RAND_MAX +1.0);
            u2= std::rand() / (RAND_MAX +1.0);
            v1=2*u1-1;
            v2=2*u1-1;
            w=v1*v1+v2*v2;
        }
        while (w>1);

        double r =std::sqrt ((-2 * std::log(w)) /w);

        tarolt=r*v2;
        nincsTarolt =!nincsTarolt;

        return r* v1;
    }

    else
    {
        nincsTarolt =!nincsTarolt;
        return tarolt;
    }
}

int main (int argc, char **argv)
{
    PolarGen pg;

    for (int i= 0; i<10;++i)
        std::cout<<pg.kovetkezo ()<< std::endl;

    return 0;
}
```

Nézzük részenként:

```
class PolarGen
{
public:
    PolarGen()
    {
        nincsTarolt = true;
        std::srand (std::time(NULL));
    }
    ~PolarGen()
```

```
{  
}  
double kovetkezo();  
  
private:  
    bool nincsTarolt;  
    double tarolt;  
};
```

PolarGen classunk, ami két fő részből áll: public és private. A publikus rész elérhető a classon kívülről is, ezzel szemben a privát viszont csak a classon belül érhető el. A classon belüli funkció, aminek a neve megegyezik a class nevével speciális, konstruktornak hívjuk, a konstruktor tartalma akkor hajtódi végre, amikor új objektumot hozunk létre a classból. A classon belüli funkció, aminek a neve megegyezik a class nevével és van előtte egy '~' jel szintén speciális, destruktornak hívjuk. A destruktur tartalma akkor hajtódi végre amikor az objektumunk kimegy az aktuális 'scope'-ból vagy a delete utasítást használjuk a classból létrehozott objektumra mutató pointere.

```
double kovetkezo();
```

Ez a funkció végzi a lényegi számítást.

```
private:  
    bool nincsTarolt;  
    double tarolt;
```

A classunk privát része, nincsTarolt egy bool-típus(igaz-hamis) benne jelezük hogy van-e tárolt érték, double tarolt-ban pedig tároljuk az aktuális értéket.

```
double PolarGen::kovetkezo ()  
{  
    if (nincsTarolt)  
    {  
        double u1, u2, v1, v2, w;  
        do  
        {  
            u1= std::rand() / (RAND_MAX +1.0);  
            u2= std::rand() / (RAND_MAX +1.0);  
            v1=2*u1-1;  
            v2=2*u1-1;  
            w=v1*v1+v2*v2;  
        }  
        while (w>1);  
  
        double r =std::sqrt ((-2 * std::log (w)) /w);  
  
        tarolt=r*v2;  
        nincsTarolt =!nincsTarolt;  
  
        return r* v1;
```

```
    }

    else
    {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
}
```

A kovetkezo() funkció végzi a lényegi számítást. Ha nincs tárolt érték akkor számolunk két értéket, az egyiket visszaadjuk a másikat eltároljuk, ha van tárolt érték akkor azt adjuk vissza.

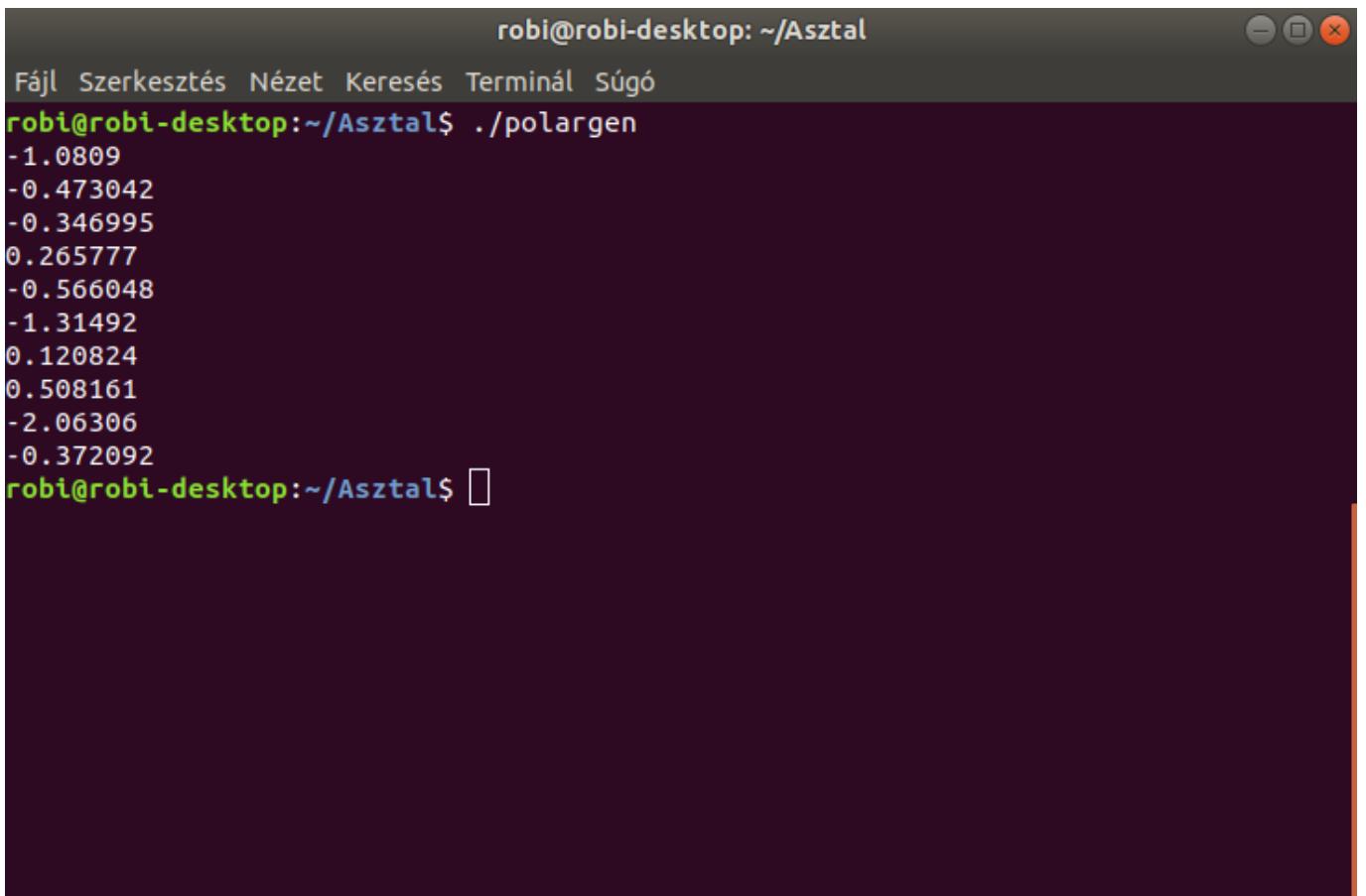
```
int main (int argc, char **argv)
{
    PolarGen pg;

    for (int i= 0; i<10;++i)
        std::cout<<pg.kovetkezo ()<< std::endl;

    return 0;
}
```

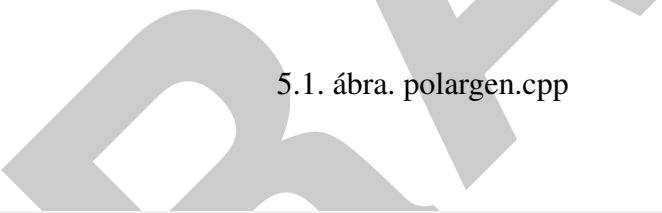
A main-ben létrehozzuk a pg PolarGen osztályt, majd for ciklussal 10-szer, kiíratjuk kovetkezo() funkció által visszaadott értéket.

A program futtatása után a következő eredményt kapjuk:



Fájl Szerkesztés Nézet Keresés Terminál Súgó
robi@robi-desktop:~/Asztal\$./polargen
-1.0809
-0.473042
-0.346995
0.265777
-0.566048
-1.31492
0.120824
0.508161
-2.06306
-0.372092
robi@robi-desktop:~/Asztal\$ █

5.1. ábra. polargen.cpp



A Java kód:

```
public class PolarGenerator
{
    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
        if(nincsTarolt)
        {
            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));
    tarolt = r * v2;
    nincsTarolt = !nincsTarolt;
    return r * v1;
}
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}

public static void main(String[] args)
{
    PolarGenerator g = new PolarGenerator();
    for (int i = 0; i < 10; ++i)
    {
        System.out.println(g.kovetkezo());
    }
}
```

Hasonló a C++ kódhoz, viszont rövidebb, könnyebben olvasható jávában az egész kód egy class része, még a main is. Jávában külön kell megadnunk minden funkciónak hogy privát vagy publikus szemben a C++-al, ahol fel lehetett sorolni a funkciókat a megfelelő részeknél.

5.2. LZW

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/z.c>

Az LZW (Lempel-Ziv-Welch) algoritmus egy veszteségmentes tömörítési eljárás, széles körben használják az informatikában, például a gif formátuma, a tömörítők is használják pl:compress, gzip, zip.

Az a tömörítés alapja hogy a kódoló csak a szótábeli indexet küldi át, a szótár a kódolás során dinamikusan bővül, kiinduló állapotában az összes egybetűs szinbólumot tartalmazza. A kódolás elve a következő: a kezdő poziciótól kezdve, addig kell a szinbólumokat olvasni, amíg a sorozat szerepel a szótárban. Ezt követően elküldjük ennek a sorozatnak az indexét és a szótárba felvesszük kiegészítve a következő szinbólummal, és az algoritmust ettől a szinbólumtól folytatjuk.

A következő programunk is ezen az elven fogja a tördelést megvalósítani például a következő 0-ból és 1-esekből álló bemenetre:

00011101110

A következő bináris fát fogjuk kapni:



A programot a következőképpen fordítjuk:

```
gcc z.c -o z
```

Majd futtatjuk:

```
./z < b.txt
```

Itt b.txt tartalmát irányítjuk a programba, programunk karakterekől álló sorozatot fog tördelni.

Teljes forráskód:

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

typedef struct binfa
{
    int ertek;
    struct binfa *bal nulla;
    struct binfa *jobb egy;
} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

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

extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);

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

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)
printf ("---");
    printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
,
    melyseg);
    kiir (elem->bal nulla);
    --melyseg;
}

void
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal nulla);
        free (elem);
    }
}
```

Nézzük részenként:

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

Az unistd.h header fájlt tartalmazzuk, read and write funkciók használatához szükséges.

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

A binfa strukturánk definiálása, tartalmazni fog egy értéket, valamint két binfa pointert, egyet a bal nullának, és egyet a jobb egyeseknek, ugyanis ha nullát teszünk be a fába akkor balra lépünk, ha pedig egyest akkor jobbra. Ezután deklarálunk két típuskulcsszót BINFA-t és *BINFA_PTR, BINFA a binfa strukturánk kulcsszava lesz, *BINFA_PTR pedig egy binfa típusú struktúrára mutató, mutató kulcsszava lesz.

```
BINFA_PTR
uj_elemt ()
{
```

```
BINFA_PTR p;

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

Az uj_elem funkció ami egy BINFA_PTR fog visszadni, új csomópontok létrehozására használjuk majd, kezdésnek deklaráljuk p-t ami BINFA_PTR tipusú mutató lesz, majd a malloc függvény segítségével memóriát foglalunk neki(a malloc lefoglalja a szükséges memóriaterületet bájtokban mérve majd erről a területről ad vissza egy pointer, ezt a pointert tesszük egyenlővé p-vel. Ha nem sikerült a memória foglalás (vagyis a visszaadott pointer == NULL, azaz nem mutat sehol) akkor kiíratunk egy error üzenetet a perror() funkció segítségével és exit() funkcióval bezárjuk a programot, EXIT_FAILURE-vel jelezük hogy a program futása sikertelen volt. Ha minden rendben ment, akkor erre a lefoglalt memóriacímre mutató pointert ad vissza a funkció.

```
extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);
```

Deklaráljuk a kiir és szabadit funkciót, az extern kulcsszó annyit jelent hogy a funkciók deklarációk nem pedig definíciók, C-ben ha használni szeretnénk egy funkciót akkor először minden képpen deklarálunk kell őket és mivel itt csak a program végén a main után vannak definiálva, de viszont a mainen belül már használjuk őket, ezért szükség van a deklarációjukat előre megadni.

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

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

A main-en belül deklarálunk egy char-t, a 'b'-t, ebbe a karakter típusú változóba fogjuk olvasni a szöveget karakterenként. Lérehozzuk a gyoker nevű csomópontmutatókat, majd a gyökér mutató által mutatott elem értékének beállítjuk a '/' karaktert, ez lesz a gyökér elemünk értéke. Ezután létrehozunk még egy csomópont mutatót, a fa-t ami kezdésnek a gyoker-re fog mutatni.

```
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;
}
}

```

A read funkcióval olvasunk a standard inputról, 'b' karakter változónkba olvassuk be, és 1 bájtnyi adatot olvasunk egyszerre, write funkcióval kiírjuk az éppen 'b'-ben lévő karakter a standard outputra, szintén bájtonként haladunk. Ha 'b' karakter tartalma egyenlő '0' karakterrel, akkor megnézzük hogy a fa mutató által mutatott aktuális csomópontnak van-e nullás gyermeke, ha nincs csinálunk egyet, ennek az új csomópont gyerekeinek egyenlőre nem állítunk be gyerekeket, majd visszaállítjuk a fa mutatót a gyökérre. Ha van az aktuális csomópontnak gyereke, akkor a fa mutatót erre a gyerekre állítjuk.

```

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;
    }
}
}

```

Ha b tartalma nem egyenlő nullával akkor 1-es értéket fogunk beírni, ugyanúgy mint az előbb megnézzük hogy a fa mutató által mutatott aktuális csomópontnak van-e egyes gyermeke, ha nincs csinálunk egyet és visszaállunk a gyökérre, ha van akkor arra mutatunk a fa mutatónkal.

```

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

```

Ezután printf-el kiírunk egy sorközt, majd a kiir() funkcióval, kiíratjuk a fa szerkezetét, majd a mélységét és ezután szabadítjuk a létrehozott pointereket.

```

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)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
                ,
                melyseg);
        kiir (elem->bal nulla);
        --melyseg;
    }
}
```

A kiir funkciót használjuk a létrehozott fánk kiírására, ez egy rekurvív funkció vagyis önmagát hívja, ha a kiírandó elem nem egyenlő NULL-al, akkor növeljük a mélységet 1-el, valamint tároljuk az eddig legnagyobb mélységet, először az aktuális elem jobb oldali részfáját dolgozzuk fel majd magát az elemet, ezután az elem bal oldali részfáját dolgozzuk fel, ezt a bejárási sorrendet nevezzük inorder bejárásnak.

```
void
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal nulla);
        free (elem);
    }
}
```

A szabadít funkció szintén egy rekurzív funkció, ezzel végezzük a malloc által lefoglalt memória szabadítását, először az elem jobb oldali gyerekére hívjuk meg, azután a balra, majd felszabadítjuk az elemet.

A program a következő bemenetre:

```
00011101110
```

Ezt a kimenetet produkálja:

```
robi@robi-desktop: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
robi@robi-desktop:~/Asztal$ ./z < b.txt
00011101110
-----
1(3)
-----0(4)
-----1(2)
---/(1)
-----1(3)
-----0(2)
-----0(3)
melyseg=4robi@robi-desktop:~/Asztal$ 
```

5.2. ábra. z.c

5.3. Fabejárás

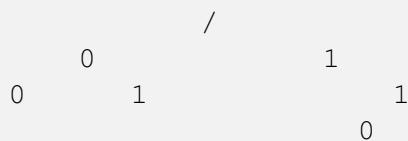
Járd be az előző (inorder bejárású) fát pre- és posztorder is!

A bináris fák bejárásának 3 módja van: inorder, preorder és posztorder. Az előző feladatban az inorder bejárás volt megvalósítva, most nézzük meg a másik kettőt:

A preorder bejárás algoritmusa:

- Ha a bejárandó fa üres az algoritmus véget ér.
- Dolgozzuk fel a gyökérelemet.
- Járjuk be a gyökérelem jobb oldali részfáját preorder módon.
- Járjuk be a gyökérelem bal oldali részfáját preorder módon.

Tehát a következő fát preorder bejárva:



Ez a sorrend: /,1,1,0,0,1,0

Ennek az eléréséhez az előző programban, csak a kiir funkció módosítására van szükség a következőképpen:

```
void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' : elem->ertek ↔
               ,
               melyseg);
        kiir (elem->jobb_egy);

        kiir (elem->bal_nulla);
        --melyseg;
    }
}
```

A for ciklushoz tartozó printf funkciót, meg a rákövetkező printf funkciót, vagyis az elem feldolgozását előrré vettük, így a preorder bejárás fog megvalósulni:

```
robi@robi-desktop: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
robi@robi-desktop:~/Asztal$ ./z < b.txt
00011101110

---/(1)
----1(2)
-----1(3)
-----0(4)
----0(2)
-----1(3)
-----0(3)
melyseg=4robi@robi-desktop:~/Asztal$ □
```

5.3. ábra. Preorder

A postorder bejárás algoritmusa:

- Ha a bejárandó fa üres az algoritmus véget ér.
- Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- Dolgozzuk fel a gyökérelemet.

Tehát a következő fát postorder bejárva:



Ez a sorrend: 0,1,1,1,0,0,/

Szintén a kiir funkció módosítására van szükség:

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

        kiir (elem->jobb_egy);

        kiir (elem->bal_nulla);
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
                ,
                melyseg);
        --melyseg;
    }
}
```

A for ciklushoz tartozó printf funkciót, meg a rákövetkező printf funkciót, vagyis az elem feldolgozását a kiir() funkciók után raktuk, ezáltal a postorder bejárás fog megvalósulni:

DRAFT

```
robi@robi-desktop: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
robi@robi-desktop:~/Asztal$ ./z < b.txt
00011101110
-----0(4)
-----1(3)
----1(2)
-----1(3)
-----0(3)
-----0(2)
---/(1)
melyseg=4robi@robi-desktop:~/Asztal$ 
```

5.4. ábra. Postorder

5.4. Tag a gyökér

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/z3a7.cpp>

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

Az LZW algoritmust C++ ba ültetve az alábbi forráskódon keresztül fogom bemutatni.

Így futtatjuk:

```
./z3a7 be.txt -o ki.txt
```

Itt be.txt a bemeneti fájl, ki.txt pedig a kimeneti.

Forráskód:

```
#include <iostream>    // mert olvassuk a std::cin, írjuk a std::cout ←
    csatornákat
#include <cmath>      // mert vonunk gyököt a szóráshoz: std::sqrt
```

```
#include <fstream>      // fájlból olvasunk, írunk majd

class LZWBInFa
{
public:

    LZWBInFa () :fa (&gyoker)
    {
    }
    ~LZWBInFa ()
    {
        szabadit (gyoker.egyesGyermek ());
        szabadit (gyoker.nullasGyermek ());
    }

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

            if (!fa->nullasGyermek ())
            {

                Csomopont *uj = new Csomopont ('0');
                fa->ujNullasGyermek (uj);
                fa = &gyoker;
            }
            else
            {

                fa = fa->>nullasGyermek ();
            }
        }
        else
        {
            if (!fa->egyesGyermek ())
            {

                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }
}
```

```
void kiir (void)
{
    melyseg = 0;

    kiir (&gyoker, std::cout);
}

void szabadit (void)
{
    szabadit (gyoker.egyesGyermek ());
    szabadit (gyoker.nullasGyermek ());
    // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a ←
    // szabad tárban (halmon).
}

friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)
{
    bf.kiir (os);
    return os;
}
void kiir (std::ostream & os)
{
    melyseg = 0;
    kiir (&gyoker, os);
}

private:
    class Csomopont
    {
public:

    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
    {
    };
    ~Csmopont ()
    {
    };

    Csmopont *nullasGyermek () const
    {
        return balNulla;
    }

    Csmopont *egyesGyermek () const
    {
        return jobbEgy;
```

```
}

void ujNullasGyermek (Csomopont * gy)
{
    balNulla = gy;
}

void ujEgyesGyermek (Csmopont * gy)
{
    jobbEgy = gy;
}

char getBetu () const
{
    return betu;
}

private:

char betu;
Csmopont *balNulla;
Csmopont *jobbEgy;

Csmopont (const Csmopont &);
Csmopont & operator= (const Csmopont &);

};

Csmopont *fa;

LZWBinFa (const LZWBinFa &);
LZWBinFa & operator= (const LZWBinFa &);

void kiir (Csmopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);

        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::left;
        --melyseg;
    }
}
```

```
void szabadit (Csomopont * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek ());
        szabadit (elem->>nullasGyermek ());

        delete elem;
    }
}

protected:

    Csomopont gyoker;
    int maxMelyseg;
    double atlag, szoras;

    void rmelyseg (Csmopont * elem);
    void ratlag (Csmopont * elem);
    void rszoras (Csmopont * elem);

};

void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    if (argc != 4)
    {
        usage ();
        return -1;
    }

    char *inFile = *++argv;

    // a -o kapcsoló jön?
    if ((*(*++argv) + 1) != 'o')
    {
```

```
usage ();
return -2;
}

std::fstream beFile (inFile, std::ios_base::in);

if (!beFile)
{
    std::cout << inFile << " nem letezik..." << std::endl;
    usage ();
    return -3;
}

std::fstream kiFile (*++argv, std::ios_base::out);

unsigned char b;      // ide olvassuk majd a bejövő fájl bájtjait
LZWBinFa binFa;      // s nyomjuk majd be az LZW fa objektumunkba

while (beFile.read ((char *) &b, sizeof (unsigned char)))
    if (b == 0x0a)
        break;

bool kommentben = false;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
{
    if (b == 0x3e)
    {      // > karakter
        kommentben = true;
        continue;
    }

    if (b == 0x0a)
    {      // újsor
        kommentben = false;
        continue;
    }

    if (kommentben)
        continue;

    if (b == 0x4e)      // N betű
        continue;

    for (int i = 0; i < 8; ++i)
    {
        if (b == 0x0a)
            continue;
        else
            binFa.insert (b);
    }
}
```

```
// 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 még a '0' betűt:
    binFa << '0';
b <<= 1;
}

}

kiFile << binFa;

kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;

kiFile.close ();
beFile.close ();

return 0;
}
```

LZWBinFa osztályunk, ami tartalmazni fogja tagként a csomópont osztályt, nézzük most a publikus részét, a publikus azt jelenti hogy bárki számára hozzáférhető a programon belül.

```
class LZWBinFa
{
public:

    LZWBinFa () :fa (&gyoker)
    {
    }
    ~LZWBinFa ()
    {
        szabadit (gyoker.egyesGyermekek ());
        szabadit (gyoker.nullasGyermekek ());
    }

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

```
{  
  
    if (!fa->nullasGyermek ())  
    {  
  
        Csomopont *uj = new Csomopont ('0');  
        fa->ujNullasGyermek (uj);  
        fa = &gyoker;  
    }  
    else  
    {  
  
        fa = fa->nullasGyermek ();  
    }  
}  
  
else  
{  
    if (!fa->egyesGyermek ())  
    {  
        Csomopont *uj = new Csomopont ('1');  
        fa->ujEgyesGyermek (uj);  
        fa = &gyoker;  
    }  
    else  
    {  
        fa = fa->egyesGyermek ();  
    }  
}  
}  
  
void kiir (void)  
{  
  
    melyseg = 0;  
  
    kiir (&gyoker, std::cout);  
}  
  
friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)  
{  
    bf.kiir (os);  
    return os;  
}  
void kiir (std::ostream & os)  
{  
    melyseg = 0;  
    kiir (&gyoker, os);  
}
```

```
}
```

```
LZWBinFa () :fa (&gyoker)
{
}
```

Az osztályban lévő funkció aminek a neve megegyezik az osztály nevével speciális, konstruktornak hívjuk és akkor fut le amikor új objektumot hozunk létre a class típusából, szemben a korábbi bináris fánkkal, a fa gyökere nem egy pointer hanem a '/' betűt tartalmazó objektum. A védett tagok között lesz: Csomopont gyoker, a fa pedig már pointer lesz ami mindig az épülő bináris fánk azon csomopontjára mutat amit az LZW fa építő algoritmus logikája diktál. A konstruktor itt annyit csinál, hogy ráállítja a fa mutatót a gyökérre.

```
~LZWBinFa ()
{
    szabadít (gyoker.egyesGyermek ());
    szabadít (gyoker.nullasGyermek ());
}
```

Az ~LZWBinFa () szintén egy speciális funkció, az LZWBinFa neve előtt szereplő '~' azt jelenti hogy ez a funkció a destruktur, akkor fut le amikor a létrehozott classunk ki megy a scope-ból vagy a delete kifejezést alkalmazzuk egy pointerre, ami a class-ra mutat. A destrukturban a szabadít nevű funkciót hívjuk meg a gyoker egyesGyermek-ére, majd a nullasGyermek-ére, ami felszabadítja a lefoglalt memóriaterületeket.

```
void operator<< (char b)
{
    // Mit kell betenni éppen, '0'-t?
    if (b == '0')
    {
        if (!fa->nullasGyermek ())
        {

            Csomopont *uj = new Csomopont ('0');
            fa->ujNullasGyermek (uj);
            fa = &gyoker;
        }
        else
        {

            fa = fa->nullasGyermek ();
        }
    }
    else
    {
        if (!fa->egyesGyermek ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyesGyermek (uj);
        }
    }
}
```

```
        fa = &gyoker;
    }
else
{
    fa = fa->egyesGyermek ();
}
}
```

Túlterheljük az operátort:

```
void operator<< (char b)
```

C++ ban lehetőségünk van túlterhelni a már meglévő operátorokat, a túlterhelés azt jelenti hogy definiáljuk az operátor számára hogy hogyan működjön, milyen utasításokat hajtson végre, ha például a saját magunk által megalkotott típusokra használjuk.

```
if (b == '0')
{
    if (!fa->nullasGyermek ())
    {

        Csomopont *uj = new Csomopont ('0');
        fa->ujNullasGyermek (uj);
        fa = &gyoker;
    }
    else
    {

        fa = fa->nullasGyermek ();
    }
}
```

Ha a fába éppen benyomandó karakter a '0', akkor megnézük hogy a fa mutató által mutatott csomópontnak van-e nullás gyermeke, ha nincs akkor csinálunk.A new operátorral foglalunk helyet a memóriában, majd az ujNullasGyermek funkcióval, beállítjuk hogy ez az újonnan létre hozott csomópont legyen az új nullásgyereke, majd visszaállunk a gyökérre. Ha van a fa által mutatott csomópontnak '0'-ás gyereke akkor ráállítjuk a famutatót.

```
else
{
    if (!fa->egyesGyermek ())
    {
        Csomopont *uj = new Csomopont ('1');
        fa->ujEgyesGyermek (uj);
        fa = &gyoker;
    }
    else
    {
        fa = fa->egyesGyermek ();
    }
}
```

}

Ugynéz a logika az egyesgyermek esetén, ha nincs akkor csinálunk eggyet, ha pedig van akkor rálépünk a fa mutatóval.

```
void kiir (void)
{
    melyseg = 0;

    kiir (&gyoker, std::cout);
}
```

Ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát akkor a standard coutra nyomjuk. Ez a kiir funkció az azonos nevű kiir funkciót fogja meghívni, ami a tényleges kiírást végzi.

Ezután következik az LZWBinFa osztály privát része, egy class privát része közvetlenül nem elérhető, a class publikus funkcióin kell keresztsülmenni az eléréséhez, ez a privát rész fog tartalmazni a classon belül méggye class-t a Csomopont class-unkat, ami egy Csomopontot fog szimbolizálni a fán belül.

Csomopont class:

```
class Csomopont
{
public:

    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
    {
    };
    ~ Csomopont ()
    {
    };

    Csomopont * nullasGyermek () const
    {
        return balNulla;
    }

    Csomopont * egyesGyermek () const
    {
        return jobbEgy;
    }

    void ujNullasGyermek ( Csomopont * gy)
    {
        balNulla = gy;
    }

    void ujEgyesGyermek ( Csomopont * gy)
    {
        jobbEgy = gy;
    }
}
```

```
}

char getBetu () const
{
    return betu;
}

private:

    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;

    Csomopont (const Csomopont &);
    Csomopont & operator= (const Csomopont &);

};
```

A Csomopont classunk szintén publikus és privát részből áll

```
Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0)
{
};

~Csonopont ()
{
```

A konstruktor és destruktur. A konstruktorunk ha paraméter nélkül hívjuk akkor a '/' betűt állítja be a létrehozott objektum értékének, egyébként azt a betűt amit megadunk neki, majd a balNulla és jobbEgy pointereket NULL-ra állítja, C++-ban ez a 0-érték megadásával is elérhető.

```
Csonopont *nullasGyermekek () const
{
    return balNulla;
}

Csonopont *egyesGyermekek () const
{
    return jobbEgy;
```

Az nullasGyermekek, egyesGyermekek funkciók, Csonopont* adnak vissza ezeket a funkciókat fogjuk használni a csomópont által privát tagként tartalmazott balNulla valamint jobbEgy pointerek visszaadására.

```
void ujNullasGyermekek (Csonopont * gy)
{
    balNulla = gy;
}

void ujEgyesGyermekek (Csonopont * gy)
{
    jobbEgy = gy;
```

}

Az ujNullasGyermek és ujEgyesGyermek funkciók beállítják az argumentumként kapott csomópont pointert, a csomópont gyerekének.

```
char getBetu () const
{
    return betu;
}
```

A getBetu funkcióval fogjuk visszaadni a csomópont által tartalmazott betűt, a funkció utáni const jelöléssel jelezzük hogy a funkció nem fogja módosítani.

private:

```
char betu;
Csomopont *balNulla;
Csonopont *jobbEgy;

Csonopont (const Csonopont &);
Csonopont & operator= (const Csonopont &);
};
```

Ezután következik a Csonopont class privát része, ami tartalmazni fogja a betűt amit az aktuális csonopont tárol valamint két mutatót, az egyik a csonópont nullásgyerekére a másik az egyesgyerekére fog mutatni. Valamint tartalmazni fogja a másoló konstruktort és másoló értékkadást, ezeket a funkciókat akkor használjuk, akkor hívodnak meg amikor az adott objektumot szeretnénk másolni. Ha a másoló konstruktort és másoló értékkadás privát tagként szerepel azt jelenti hogy le vannak tiltva, nem használhatóak. Itt szükség van erre a tiltásra hiszen a class pointert tartalmaz és a C++ alapértelmezetten sekélymásolást végez, ami itt a pointert fogja lemásolni vagyis az új objektum pointere ugyanoda fog mutatni mint a régié, ha ettől eltérő másolást szeretnénk, akkor saját magunknak kell definiálni a másoló konstruktort és értékkadást.

Ezután tovább folytatódik az LZWBInFa osztály privát része:

```
Csonopont *fa;

LZWBInFa (const LZWBInFa &);
LZWBInFa & operator= (const LZWBInFa &);
```

Itt deklaráljuk a fa nevű Csonopont pointerünket valamint a Csonopont osztályhoz hasonlóan itt is tiltjuk a másoló konstruktort és értékkadást.

```
void kiir (Csonopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);
```

```
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->nullasGyermek (), os);
        --melyseg;
    }
}

void szabadit (Csomopont * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek ());
        szabadit (elem->>nullasGyermek ());

        delete elem;
    }
}
```

A kiir és szabadit funkciókat korábban már bemutattam, úgyhogy itt csak röviden írnám le. A kiir funkció rekurzív módon járja be a fát, az inorder bejárást megvalósítva, ezzel a funkcióval íratjuk ki a fa elemeit. A szabadít funkcióval pedig a memória szabadítását végezzük szintén rekurzív módon.

```
Csomopont gyoker;
int maxMelyseg;
double atlag, szoras;

void rmelyseg (Csmopont * elem);
void ratlag (Csmopont * elem);
void rszoras (Csmopont * elem);

};
```

LZWBinFa osztály további része deklaráljuk a gyökeret valamint a maxMelyseg, atlag, szoras változókat és rmelyseg, ratlag, rszoras funkciókat.

```
void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}
```

Az usage funkció definíciója, ezzel a funkcióval fogjuk kiiratni a használati utasítást.

Ezután következik a main:

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

```
if (argc != 4)
{
    usage ();
    return -1;
}
```

Rögzítjük a parancssori argumentumokat, ha az argumentumok száma nem egyenlő 4-el akkor kiíratjuk a használati utasítást.

```
char *inFile = *++argv;

// a -o kapcsoló jön?
if (*((++argv) + 1) != 'o')
{
    usage ();
    return -2;
}
```

Az inFile nevű char pointer a második argumentum első karakterére fog mutatni vagyis a bemeneti fájl első karakterére. Ezután ellenőrizzük hogy az 'o' kapcsoló jön-e ha nem akkor kiíratjuk a használati utasítást.

```
std::fstream beFile (inFile, std::ios_base::in);

if (!beFile)
{
    std::cout << inFile << " nem létezik..." << std::endl;
    usage ();
    return -3;
}

std::fstream kiFile (*++argv, std::ios_base::out);

unsigned char b;      // ide olvassuk majd a bejövő fájl bájtjait
LZWBinFa binFa;      // s nyomjuk majd be az LZW fa objektumunkba
```

Mivel fájlba írunk és fájlból olvasunk ezért fstream objektumokat hozunk létre, ha nem létezik a bemeneti fájl akkor kiíratjuk a használati utasítást. Létrehozzuk char 'b'-t amibe a beolvasott karaktereket fogjuk tárolni, valamint a binFa nevű LZWBinFa objektumunkat.

```
kiFile << binFa;

kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;

kiFile.close ();
beFile.close ();
```

```
return 0;
```

Ezután kiíratjuk a kimeneti fájlba a binfánk tartalmát valamint a mélységet, átlagot és szórást, majd zárjuk a be- és kimeneti fájlokat.

5.5. Mutató a gyökér

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/z3a7mutatogyok.cpp>

Aggregáció:

Rész-egész kapcsolat, a részek fogják alkotni az egészet. Például a kerékpár váz, kerekek, és kormány aggregációja, itt a részek túlélhetik az egészet. Akkor valósul meg ha egy objektumnak része, tulajdona egy másik. A tartalmazó objektum megszünésével a tartalmazott objektum tovább élhet.

Kompozíció:

A kompozíció egy speciális aggregáció, itt a rész szorosan hozzátarozik az egészhez, a rész nem éli túl az egészet, például az emberi agy szorosan hozzátarozik az emberhez. Itt a tartalmazott objektum nem lehet egyszerre több objektum része, önmagában nem létezhet, minden van tartalmazott objektuma és az élettartama azonos a tartalmazó objektuméval, a tartalmazó objektum konstruktora hozza létre és a destruktora törli.

Tehát a jelenlegi objektumunkban a gyökér csomópont kompozícióban van a fával, de ha átírjuk mutatóvá akkor már az agrregációs kapcsolat fog megvalósulni.

Átírjuk a gyökeret hogy innentől kezdve mutató legyen.

```
Csomopont *gyoker;
```

Ezután javítjuk a következő szintaktikai hibákat:

```
szabadit (gyoker.egyesGyermekek ());
szabadit (gyoker.nullasGyermekek ());
```

Mivel a gyökerünk inentől kezdve pointer ezért a `.' helyett itt a nyíl operátort fogjuk használni, ami a pointer által mutatott objektum tagjára hivatkozik:

```
szabadit (gyoker->egyesGyermekek ());
szabadit (gyoker->>nullasGyermekek ());
```

Valamint ahol a következő kifejezés szerepelt:

```
&gyoker
```

Kitöröljük az előtte lévő 'and' jelet, hiszen a pointer tartalmazza a címét az objektumnak, nem pedig a pointer címére van szükségünk:

```
gyoker
```

Valamint a konstruktort átírjuk a következőképpen:

```
LZWBinFa ()  
{  
    gyoker=new Csomopont();  
    fa=gyoker;  
}
```

Itt helyet foglalunk a memóriában, a gyoker által mutatott csomópont számára, valamint a fa mutatót ráálítjuk a gyökérre.

```
~LZWBinFa ()  
{  
    szabadit (gyoker);  
}
```

Mivel most már a gyökér is pointer, ezért használhatjuk rá a szabadit funkciót.

5.6. Mozgató szemantika

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

Tutor:

Racs Tamás: <https://gitlab.com/cant0r/bhax?fbclid=IwAR1xwKuz0cDUxX5tbmKI5YUChA>

Tutoriáltam:

Szabó Benedek: https://gitlab.com/benedekszabo/bhax?fbclid=IwAR1A8OPNSYIYFzNXbxvgNZatYny6kHrDXx_cly2LBBygBIRJAE4W1ZvLu8

A mozgató szemantika, egy nyelvi eszköz a felesleges másolások csökkentésére, hiszen nem minden van szükség a másolandó adat megőrzésére. A mozgatás gyorsabb, olcsóbb művelet mint a másolás.

LZWBinFa osztályunk esetében következő speciális funkciók segítségével adhatjuk meg:

Mozgató konstruktor:

```
LZWBinFa(LZWBinFa&& regi) {  
  
    std::cout<<"Mozgató konstruktor\n";  
    gyoker=regi.gyoker;  
    regi.gyoker=nullptr;  
}
```

A dupla "and" jel itt jobb értéket jelent(rvalue). A gyoker a "régi" gyökerével lesz egyenlő vagyis ugyanarra a fa szerkezetre fog mutatni ahova a régi. A régi gyökeret pedig kinullázzuk nullptr-re állítjuk hogy ne hívódjon meg az objektum destruktora.

Mozgató értékadás:

```
LZWBInFa& operator=(LZWBInFa&& regi) {  
  
    std::cout<<"Mozgató értékkadás\n";  
    std::swap(gyoker, regi.gyoker);  
  
}
```

Túlterheljük az '=' operátort, itt viszont a funkciótestben csak annyit csinálunk, hogy a swap funkció segítségével megcseréljük a két pointert.

A mozgató konstruktor a mozgató értékkadásra alapozva:

```
LZWBInFa(LZWBInFa&& regi) {  
  
    gyoker=nullptr;  
    *this = std::move(regi);  
  
}  
  
LZWBInFa& operator=(LZWBInFa&& regi) {  
  
    std::swap(gyoker, regi.gyoker);  
    return *this;  
  
}
```

A mozgató konstruktorban a jelenlegi gyökeret nullptr-re állítjuk, a "this" egy speciális kulcszó ami, egy mutató-t jelent a class-ból létrehozott aktuális objektumra, a "*this" ennek a pointernek a dereferenciálása, vagyis az aktuális objektumot jelenti. Az std::move funkció a neki megadott balértékből(lvalue) jobbértéket(rvalue) készít, ezzel kikényszerítve a mozgató szemantikát, tehát a következő sorban:

```
*this = std::move(regi);
```

Azt mondjuk hogy a jelenlegi objektumunk legyen egyenlő std::move(regi)-vel de mivel a jelenlegi objektumunkat már inicializáltuk a gyoker=nullptr sorban, ezért ez a sor a mozgató értékkadást fogja meghívni, ami a swap funkcióval kicseréli a jelenlegi gyökeret(ami nullptr) a régivel.

6. fejezet

Helló, Conway!

6.1. Hangyszimulációk

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

Tutorok:

Tóth Csaba: <https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeaTjhhiMRoPXff9fTfWW8Uk9M1QM>

Racs Tamás: <https://gitlab.com/cant0r/bhax?fbclid=IwAR1xwKuz0cDUxX5tbtmKI5YUCHA>

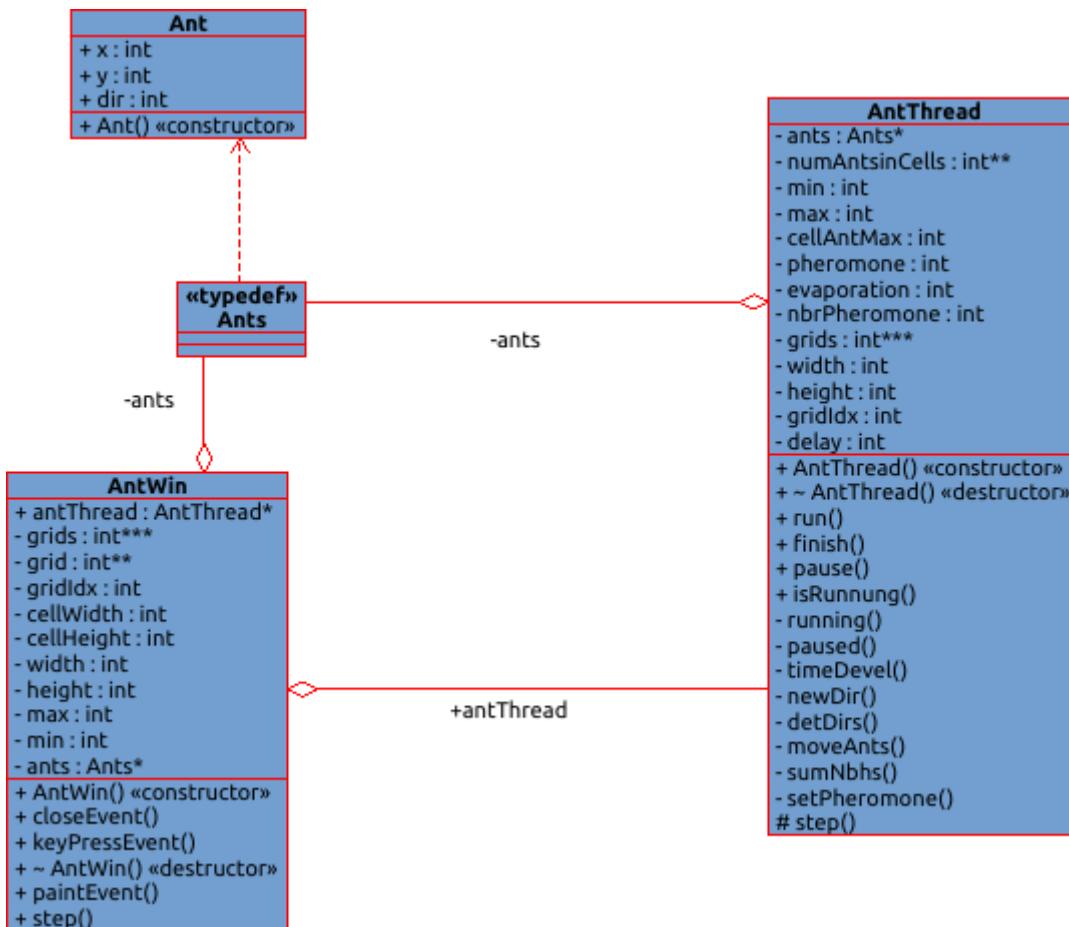
Megoldás videó: <https://bhaxor.blog.hu/2018/10/10/myrmecologist>

Megoldás forrása: <https://github.com/BorvizRobi/hangya>

A következő program egy hangyaboly tipikus viselkedését, hangyák mozgását fogja szimulálni.

Az UML (Unified Modeling Language), egy általános célú modellező nyelv, használják például üzleti elemzők, rendszertervezők, szoftvermérnökök. Az UML nyelv segítségével szöveges és grafikus modellek készíthetők például szoftverekről, programokról is. Az UML a szabványos specifikációs nyelve az objektum orientált programozásnak.

A programunk osztályait és kapcsolataikat a következő UML diagramm szemlélteti:



6.1. ábra. UML osztálydiagram

Az osztály jele UML-ben alapesetben egy függőlegesen három részre osztott téglalap. A felső részben van az osztály neve, a középsőben az attributumok, a legalsóban pedig a funkciók. A az osztályok közötti kapcsolatok jelölésénél a nyíl végén egy üres rombusszal az aggregációt jelöli. A szagatott nyíl pedig a függőséget jelzi azaz ha változik a független elem akkor változik a függő is, a nyíl a független elem felé mutat.

A láthatóságra az alábbi jelöléseket használjuk:

```

+ public
# protected
- private
  
```

A program Qt segítségével lesz elkészítve:

```
~/Qt/5.12.2/gcc_64/bin/qmake myrmecologist.pro
```

A qmake-t onnan indítjuk ahova telepítettük a Qt-t.

Ezután a parancssorba kiadjuk a make parancsot, majd futtatjuk a programot:

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

Itt a -w kapcsolóval adjuk meg az ablak szélességét a -m kapcsólóval a magasságát, a -n kapcslóló a hangyák számát adja meg, a -t kapcsólóval a lépések gyakoriságát állítjuk, a -p vel a feromonok párolgásának gyorsaságát állítjuk be, az -f el azt állítjuk hogy ha egy hangya belép egy cellába, akkor ott mennyivel növeli meg a cella feromonszintjét, a -d kapcsoló a cellák kezdő feromonszintjét adja meg, a '-a' és '-i' kapcsolóval a maximum és minimum feromonértékeket adjuk meg, a -s kapcsolóval azt adjuk meg, hogy egy hangya mennyi feromont hagy a szomszédos cellákban a -c vel pedig hogy hány hangya lehet egyszere egy cellában.

Foráskódok:

ant.h:

```
#ifndef ANT_H  
#define ANT_H  
  
class Ant  
{  
  
public:  
    int x;  
    int y;  
    int dir;  
  
    Ant(int x, int y): x(x), y(y) {  
  
        dir = qrand() % 8;  
  
    }  
  
};  
  
typedef std::vector<Ant> Ants;  
#endif
```

Ant nevű class-unk, ami egy hangya tulajdonságait fogja tartalmazni, 'x' és 'y' nevű változókban azt tároljuk hogy melyik oszloban és sorban van a hangya a "dir" nevű változóban pedig az írányát. A konstruktur a paraméterként megkapott két számot beálítja x és y értékének, valamint dir értékének megadja qrand() segítségével generált random számot amit előtte maradékos osztással elosztunk 8-al.

```
typedef std::vector<Ant> Ants;
```

Ez a sor egy típust definiál az Ants-ot, tehát ezen túl ha Ants előtaggal hozunk létre egy változót, az egy Ant-okból álló vektort fog jelenteni.

main.cpp:

```
#include <QApplication>
```

```
#include <QDesktopWidget>
#include <QDebug>
#include <QDateTime>
#include <QCommandLineOption>
#include <QCommandLineParser>

#include "antwin.h"

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

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

    QApplication a ( argc, argv );

    QCommandLineOption szeles_opt ( { "w", "szelesseg" }, "Oszlopok (cellakban ←
        ) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( { "m", "magassag" }, "Sorok (cellakban) ←
        szama.", "magassag", "150" );
    QCommandLineOption hangyaszam_opt ( { "n", "hangyaszam" }, "Hangyak szama ←
        .", "hangyaszam", "100" );
    QCommandLineOption sebesseg_opt ( { "t", "sebesseg" }, "2 lepes kozotti ←
        ido (millisec-ben).", "sebesseg", "100" );
    QCommandLineOption parolgas_opt ( { "p", "parolgas" }, "A parolgas erteke ←
        .", "parolgas", "8" );
    QCommandLineOption feromon_opt ( { "f", "feromon" }, "A hagyott nyom ←
        erteke.", "feromon", "11" );
    QCommandLineOption szomszed_opt ( { "s", "szomszed" }, "A hagyott nyom ←
        erteke a szomszedokban.", "szomszed", "3" );
    QCommandLineOption alapertek_opt ( { "d", "alapertek" }, "Indulo ertek a ←
        cellakban.", "alapertek", "1" );
    QCommandLineOption maxcella_opt ( { "a", "maxcella" }, "Cella max erteke ←
        .", "maxcella", "50" );
    QCommandLineOption mincella_opt ( { "i", "mincella" }, "Cella min erteke ←
        .", "mincella", "2" );
    QCommandLineOption cellamerete_opt ( { "c", "cellameret" }, "Hany hangya ←
        fer egy cellaba.", "cellameret", "4" );
    QCommandLineParser parser;

    parser.addHelpOption();
    parser.addVersionOption();
    parser.addOption ( szeles_opt );
    parser.addOption ( magas_opt );
    parser.addOption ( hangyaszam_opt );
    parser.addOption ( sebesseg_opt );
```

```
parser.addOption ( parolgas_opt );
parser.addOption ( feromon_opt );
parser.addOption ( szomszed_opt );
parser.addOption ( alapertek_opt );
parser.addOption ( maxcella_opt );
parser.addOption ( mincella_opt );
parser.addOption ( cellamerete_opt );

parser.process ( a );

QString szeles = parser.value ( szeles_opt );
QString magas = parser.value ( magas_opt );
QString n = parser.value ( hangyaszam_opt );
QString t = parser.value ( sebesseg_opt );
QString parolgas = parser.value ( parolgas_opt );
QString feromon = parser.value ( feromon_opt );
QString szomszed = parser.value ( szomszed_opt );
QString alapertek = parser.value ( alapertek_opt );
QString maxcella = parser.value ( maxcella_opt );
QString mincella = parser.value ( mincella_opt );
QString cellameret = parser.value ( cellamerete_opt );

qsrand ( QDateTime::currentMsecsSinceEpoch() );

AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), feromon.toInt(),
           szomszed.toInt(), parolgas.toInt(),
           alapertek.toInt(), mincella.toInt(), maxcella.toInt(),
           cellameret.toInt() );

w.show();

return a.exec();
}
```

A mainben Qt-s módon kezeljük a parancssori argumentumokat, majd ezeket átadjuk a létrehozott AntWin osztálynak.

antwin.h:

```
#ifndef ANTWIN_H
#define ANTWIN_H

#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCloseEvent>
#include "antthread.h"
#include "ant.h"

class AntWin : public QMainWindow
{
```

```
Q_OBJECT

public:
    AntWin(int width = 100, int height = 75,
            int delay = 120, int numAnts = 100,
            int pheromone = 10, int nbhPheromon = 3,
            int evaporation = 2, int cellDef = 1,
            int min = 2, int max = 50,
            int cellAntMax = 4, QWidget *parent = 0);

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {

        antThread->finish();
        antThread->wait();
        event->accept();
    }

    void keyPressEvent ( QKeyEvent *event )
    {

        if ( event->key() == Qt::Key_P ) {
            antThread->pause();
        } else if ( event->key() == Qt::Key_Q
                    || event->key() == Qt::Key_Escape ) {
            close();
        }
    }

    virtual ~AntWin();
    void paintEvent(QPaintEvent*);

private:

    int ***grids;
    int **grid;
    int gridIdx;
    int cellWidth;
    int cellHeight;
    int width;
    int height;
    int max;
    int min;
    Ants* ants;

public slots :
    void step ( const int &);
```

```
};  
  
#endif
```

A antwin classunk deklarációit az antwin.h a definíciókat az antwin.cpp tartalmazza. Az antwin.h ban includoljuk a Qt-s osztályokat a QMainWindow osztály szükséges az ablakunk létrehozásához, a QPainter a szimuláció kirajzolásához, QString osztály segítségével tudunk Unicode kódolású stringeket tárolni, a QCloseEvent segítségével pedig a program bezárását tudjuk szabályozni. Ebben az osztályban hozzuk létre az ablakot. Itt állítjuk be az ablak paramétereit amit parancssori argumentumként adunk át a programnak, valamint itt van a billentyűk lenyomásának kezelése.

antwin.cpp:

```
#include "antwin.h"  
#include <QDebug>  
  
AntWin::AntWin ( int width, int height, int delay, int numAnts,  
                 int pheromone, int nbhPheromon, int evaporation, int ←  
                 cellDef,  
                 int min, int max, int cellAntMax, QWidget *parent ) : ←  
                 QMainWindow ( parent )  
{  
    setWindowTitle ( "Ant Simulation" );  
  
    this->width = width;  
    this->height = height;  
    this->max = max;  
    this->min = min;  
  
    cellWidth = 6;  
    cellHeight = 6;  
  
    setFixedSize ( QSize ( width*cellWidth, height*cellHeight ) );  
  
    grids = new int**[2];  
    grids[0] = new int*[height];  
    for ( int i=0; i<height; ++i ) {  
        grids[0][i] = new int [width];  
    }  
    grids[1] = new int*[height];  
    for ( int i=0; i<height; ++i ) {  
        grids[1][i] = new int [width];  
    }  
  
    gridIdx = 0;  
    grid = grids[gridIdx];  
  
    for ( int i=0; i<height; ++i )  
        for ( int j=0; j<width; ++j ) {  
            grid[i][j] = cellDef;  
        }
```

```
ants = new Ants();

antThread = new AntThread ( ants, grids, width, height, delay, numAnts, ←
    pheromone,
                           nbhPheromon, evaporation, min, max, ←
                           cellAntMax);

connect ( antThread, SIGNAL ( step ( int ) ),
          this, SLOT ( step ( int ) ) );

antThread->start();

}

void AntWin::paintEvent ( QPaintEvent* )
{
    QPainter qpainter ( this );

    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i ) {
        for ( int j=0; j<width; ++j ) {

            double rel = 255.0/max;

            qpainter.fillRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight,
                                QColor ( 255 - grid[i][j]*rel,
                                          255,
                                          255 - grid[i][j]*rel ) );

            if ( grid[i][j] != min )
            {
                qpainter.setPen (
                    QPen (
                        QColor ( 255 - grid[i][j]*rel,
                                  255 - grid[i][j]*rel, 255 ),
                        1 )
                );

                qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                    cellWidth, cellHeight );
            }

            qpainter.setPen (
                QPen (
                    QColor ( 0, 0, 0 ),
```

```
    1 )
);

qpainter.drawRect ( j*cellWidth, i*cellHeight,
                     cellWidth, cellHeight );

}

}

for ( auto h: *ants) {
    qpainter.setPen ( QPen ( Qt::black, 1 ) );
    qpainter.drawRect ( h.x*cellWidth+1, h.y*cellHeight+1,
                        cellWidth-2, cellHeight-2 );
}

qpainter.end();
}

AntWin::~AntWin()
{
    delete antThread;

    for ( int i=0; i<height; ++i ) {
        delete[] grids[0][i];
        delete[] grids[1][i];
    }

    delete[] grids[0];
    delete[] grids[1];
    delete[] grids;

    delete ants;
}

void AntWin::step ( const int &gridIdx )
{
    this->gridIdx = gridIdx;
    update();
}
```

Akkor nézzük most a definíciókat. Az osztályunk örköl a QMainWindow osztályból, ezért rendelkezik azon funkcióival, tehát itt állítjuk be az ablak tulajdonságait: címét, méretét valamint hogy ne legyen átméretezhető. A grids fogja tárolni a programban szereplő mozgásokat, két rácst hozunk létre az egyik rácson tároljuk, hogy hova lép a hangya a másikon pedig a feromonokat nézzük. A hangyákat is létrehozzuk. A gridIdx-ben tároljuk hogy melyik rácst használjuk.

A paintEvent-el már találkoztunk ez a GUI újra rajzolásához szükséges, az update() függvény hívja meg.

Kirajzoljuk a megjelenítendő adatokat: hangyákat, feromonokat, stb..;

A destrukturban mivel mi foglaltunk helyet a memóriában, ezért gondoskodnunk kell a törlésükről először a mutatók által mutatott objektumokat töröljük azután magukat az objektumokat is, majd töröljük az ants-okat is.

antthread.h:

```
#ifndef ANTTHREAD_H
#define ANTTHREAD_H

#include <QThread>
#include "ant.h"

class AntThread : public QThread
{
    Q_OBJECT

public:
    AntThread(Ants * ants, int ***grids, int width, int height,
               int delay, int numAnts, int pheromone, int nbrPheromone,
               int evaporation, int min, int max, int cellAntMax);

    ~AntThread();

    void run();
    void finish()
    {
        running = false;
    }

    void pause()
    {
        paused = !paused;
    }

    bool isRunning()
    {
        return running;
    }

private:
    bool running {true};
    bool paused {false};
    Ants* ants;
    int** numAntsinCells;
    int min, max;
    int cellAntMax;
    int pheromone;
    int evaporation;
    int nbrPheromone;
    int ***grids;
```

```
int width;
int height;
int gridIdx;
int delay;

void timeDevel();

int newDir(int sor, int oszlop, int vsor, int voszlop);
void detDirs(int irany, int& ifrom, int& ito, int& jfrom, int& jto );
int moveAnts(int **grid, int row, int col, int& retrow, int& retcol, ←
    int);
double sumNbhs(int **grid, int row, int col, int);
void setPheromone(int **grid, int row, int col);

signals:
    void step ( const int &);

};

#endif
```

Az AntThread class-unk egy QThread-ból származtatott class, QThread osztály segítségével kezeljük a program szálait. A konstruktora megkapja a korábban rögzített paramétereket. Ez a class fogja elvégezni a hangyák mozgatásával kapcsolatos számításokat: meghatározza az irányokat, beállítja a feromonszinteket, meghatározza hogy melyik cellákat kell átszínezni.

antthread.cpp:

```
#include "antthread.h"
#include <QDebug>
#include <cmath>
#include <QDateTime>

AntThread::AntThread ( Ants* ants, int*** grids,
                      int width, int height,
                      int delay, int numAnts,
                      int pheromone, int nbrPheromone,
                      int evaporation,
                      int min, int max, int cellAntMax)
{
    this->ants = ants;
    this->grids = grids;
    this->width = width;
    this->height = height;
    this->delay = delay;
    this->pheromone = pheromone;
    this->evaporation = evaporation;
    this->min = min;
    this->max = max;
    this->cellAntMax = cellAntMax;
    this->nbrPheromone = nbrPheromone;
```

```
numAntsinCells = new int*[height];
for ( int i=0; i<height; ++i ) {
    numAntsinCells[i] = new int [width];
}

for ( int i=0; i<height; ++i )
    for ( int j=0; j<width; ++j ) {
        numAntsinCells[i][j] = 0;
    }

qsrand ( QDateTime::currentMSecsSinceEpoch() );

Ant h {0, 0};
for ( int i {0}; i<numAnts; ++i ) {

    h.y = height/2 + qrand() % 40-20;
    h.x = width/2 + qrand() % 40-20;

    ++numAntsinCells[h.y][h.x];

    ants->push_back ( h );
}

gridIdx = 0;
}

double AntThread::sumNbhs ( int **grid, int row, int col, int dir )
{
    double sum = 0.0;

    int ifrom, ito;
    int jfrom, jto;

    detDirs ( dir, ifrom, ito, jfrom, jto );

    for ( int i=ifrom; i<ito; ++i )
        for ( int j=jfrom; j<jto; ++j )

            if ( ! ( ( i==0 ) && ( j==0 ) ) ) {
                int o = col + j;
                if ( o < 0 ) {
                    o = width-1;
                } else if ( o >= width ) {
                    o = 0;
                }

                int s = row + i;
                if ( s < 0 ) {

```

```
        s = height-1;
    } else if ( s >= height ) {
        s = 0;
    }

    sum += (grid[s][o]+1)*(grid[s][o]+1)*(grid[s][o]+1);

}

return sum;
}

int AntThread::newDir ( int sor, int oszlop, int vsor, int voszlop )
{

if ( vsor == 0 && sor == height -1 ) {
    if ( voszlop < oszlop ) {
        return 5;
    } else if ( voszlop > oszlop ) {
        return 3;
    } else {
        return 4;
    }
} else if ( vsor == height - 1 && sor == 0 ) {
    if ( voszlop < oszlop ) {
        return 7;
    } else if ( voszlop > oszlop ) {
        return 1;
    } else {
        return 0;
    }
} else if ( voszlop == 0 && oszlop == width - 1 ) {
    if ( vsor < sor ) {
        return 1;
    } else if ( vsor > sor ) {
        return 3;
    } else {
        return 2;
    }
} else if ( voszlop == width && oszlop == 0 ) {
    if ( vsor < sor ) {
        return 7;
    } else if ( vsor > sor ) {
        return 5;
    } else {
        return 6;
    }
} else if ( vsor < sor && voszlop < oszlop ) {
    return 7;
} else if ( vsor < sor && voszlop == oszlop ) {
```

```
        return 0;
    } else if ( vsor < sor && voszlop > oszlop ) {
        return 1;
    }

    else if ( vsor > sor && voszlop < oszlop ) {
        return 5;
    } else if ( vsor > sor && voszlop == oszlop ) {
        return 4;
    } else if ( vsor > sor && voszlop > oszlop ) {
        return 3;
    }

    else if ( vsor == sor && voszlop < oszlop ) {
        return 6;
    } else if ( vsor == sor && voszlop > oszlop ) {
        return 2;
    }

    else { // (vsor == sor && voszlop == oszlop)
        qDebug() << "ZAVAR AZ EROBEN az iranynal";
        return -1;
    }
}

void AntThread::detDirs ( int dir, int& ifrom, int& ito, int& jfrom, int& ←
    jto )
{
    switch ( dir ) {
        case 0:
            ifrom = -1;
            ito = 0;
            jfrom = -1;
            jto = 2;
            break;
        case 1:
            ifrom = -1;
            ito = 1;
            jfrom = 0;
            jto = 2;
            break;
        case 2:
            ifrom = -1;
            ito = 2;
            jfrom = 1;
            jto = 2;
            break;
    }
}
```

```
case 3:
    ifrom =
        0;
    ito = 2;
    jfrom = 0;
    jto = 2;
    break;
case 4:
    ifrom = 1;
    ito = 2;
    jfrom = -1;
    jto = 2;
    break;
case 5:
    ifrom = 0;
    ito = 2;
    jfrom = -1;
    jto = 1;
    break;
case 6:
    ifrom = -1;
    ito = 2;
    jfrom = -1;
    jto = 0;
    break;
case 7:
    ifrom = -1;
    ito = 1;
    jfrom = -1;
    jto = 1;
    break;
}

int AntThread::moveAnts ( int **racs,
                        int sor, int oszlop,
                        int& vsor, int& voszlop, int dir )
{
    int y = sor;
    int x = oszlop;

    int ifrom, ito;
    int jfrom, jto;

    detDirs ( dir, ifrom, ito, jfrom, jto );

    double osszes = sumNbhs ( racs, sor, oszlop, dir );
```

```
double random = ( double ) ( qrand() %1000000 ) / ( double ) 1000000.0;
double gvalseg = 0.0;

for ( int i=ifrom; i<ito; ++i )
    for ( int j=jfrom; j<jto; ++j )
        if ( ! ( ( i==0 ) && ( j==0 ) ) )
        {
            int o = oszlop + j;
            if ( o < 0 ) {
                o = width-1;
            } else if ( o >= width ) {
                o = 0;
            }

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

            //double kedvezo = std::sqrt((double)(racs[s][o]+2)); // ( ←
            //racs[s][o]+2)*(racs[s][o]+2);
            //double kedvezo = (racs[s][o]+b)*(racs[s][o]+b);
            //double kedvezo = ( racs[s][o]+1 );
            double kedvezo = (racs[s][o]+1)*(racs[s][o]+1)*(racs[s][o ←
                ]+1);

            double valseg = kedvezo/osszes;
            gvalseg += valseg;

            if ( gvalseg >= random ) {

                vsor = s;
                voszlop = o;

                return newDir ( sor, oszlop, vsor, voszlop );

            }
        }
    }

qDebug() << "ZAVAR AZ EROBEN a lepesnel";
vsor = y;
voszlop = x;

return dir;
}
```

```
void AntThread::timeDevel()
{
    int **racsElotte = grids[gridIdx];
    int **racsUtana = grids[ ( gridIdx+1 ) %2 ];

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j )
    {
        racsUtana[i][j] = racsElotte[i][j];

        if ( racsUtana[i][j] - evaporation >= 0 ) {
            racsUtana[i][j] -= evaporation;
        } else {
            racsUtana[i][j] = 0;
        }
    }

    for ( Ant &h: *ants )
    {

        int sor {-1}, oszlop {-1};
        int ujirany = moveAnts( racsElotte, h.y, h.x, sor, oszlop, h.dir );

        setPheromone ( racsUtana, h.y, h.x );

        if ( numAntsinCells[sor][oszlop] <cellAntMax ) {

            --numAntsinCells[h.y][h.x];
            ++numAntsinCells[sor][oszlop];

            h.x = oszlop;
            h.y = sor;
            h.dir = ujirany;

        }
    }

    gridIdx = ( gridIdx+1 ) %2;
}

void AntThread::setPheromone ( int **racs,
                               int sor, int oszlop )
{
    for ( int i=-1; i<2; ++i )
        for ( int j=-1; j<2; ++j )

```

```
if ( ! ( ( i==0 ) && ( j==0 ) ) )
{
    int o = oszlop + j;
    {
        if ( o < 0 ) {
            o = width-1;
        } else if ( o >= width ) {
            o = 0;
        }
    }
    int s = sor + i;
    {
        if ( s < 0 ) {
            s = height-1;
        } else if ( s >= height ) {
            s = 0;
        }
    }

    if ( racs[s][o] + nbrPheromone <= max ) {
        racs[s][o] += nbrPheromone;
    } else {
        racs[s][o] = max;
    }
}

if ( racs[sor][oszlop] + pheromone <= max ) {
    racs[sor][oszlop] += pheromone;
} else {
    racs[sor][oszlop] = max;
}

}

void AntThread::run()
{
    running = true;
    while ( running ) {

        QThread::msleep ( delay );

        if ( !paused ) {
            timeDevel();
        }

        emit step ( gridIdx );
    }
}
```

```
}
```

```
AntThread::~AntThread()
```

```
{
```

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

```
        delete [] numAntsinCells[i];
```

```
    }
```

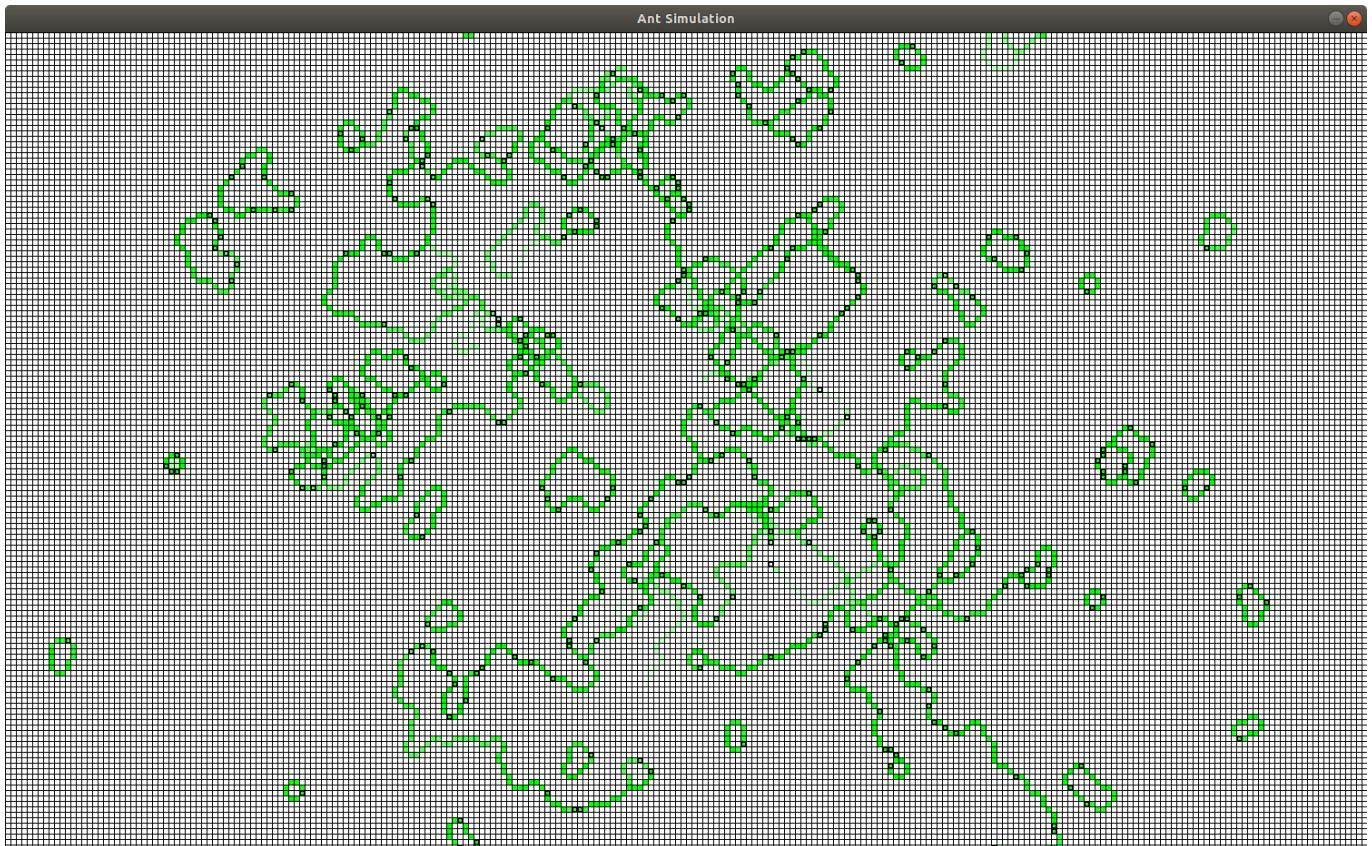
```
    delete [] numAntsinCells;
```

```
}
```

Tehát akkor itt történnek a számítások, külön szálön. qrand() segítségével generáljuk a random számokat, erre azért van szükség hogy a hangyáink ne ugyanazon a helyen kezdjenek. A sumNbhs funkciót arra használjuk, hogy megnézzük hány szomszédja van egy hangyának, tehát a szomszédos mezőket vizsgáljuk.

A newDir funkció fogja kiszámolni a hangya írányát. A detDir funkció végzi a hangya mozgatását a rácson, a hangyáknak a legnagyobb feromonszinttel rendelkező cellába kell lépniük, de ez nem minden így van a valóságban sem ezért randomszámgenerálással oldjuk meg a problémát. A timeDevel() funkció, az idő műlásának függvényében kezeli a feromonszinteket, itt van szükség két rácsra, az eredeti rács újraszámolt feromonszintjeit kapja meg a racsUtanna által mutatott rács. A setPheromone() funkcióval állítjuk be a rács feromonszintjét, ennek van egy maximum értéke, tehát ha a cella már maximumon van akkor ezt a maximumot állítjuk be. A run() funkció a QThread-ból örökölt funkció, ameddig nem szüneteltetjük a szálat addig fut a szimuláció. Ezután a destrukturálva szabadítjuk a lefoglalt memóriaterületet.

A program futás közben:



6.2. ábra. Hangya szimuláció

6.2. Java életjáték (Passz)

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/Sejtautomata.java>

Az életjátékot John Horton Conway a Cambridge Egyetem matematikusa találta ki, ez egy egyszemélyes játék a játékos feladata annyi hogy megad egy kezdőalakzatot és ezután figyeli az eredményt.

A játék részletes leírása a következő linken: <https://hu.wikipedia.org/wiki/%C3%89letj%C3%A1t%C3%A9k>

A játékban alakzatoknak nevezzük sejtek egy halmazát, néhány ilyen halmaz hamar kihal(egy sejtje se marad életben), vannak amik változatlanok maradnak, és vannak olyanok is amelyek ciklikusan önmagukba térnek vissza, ilyen jellegzetes alakzat a sikló is, a program ilyen siklókat a végtelenbe lövő alakzatot fog készíteni.

A programot a következőképpen fordítjuk:

```
javac Sejtautomata.java
```

Majd futtatjuk:

```
java Sejtautomata
```

Teljes forráskód:

```
/*
 * Sejtautomata.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * Sejtautomata osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class Sejtautomata extends java.awt.Frame implements Runnable {
    /** Egy sejt lehet élő */
    public static final boolean ÉLŐ = true;
    /** vagy halott */
    public static final boolean HALOTT = false;
    /** Két rácsot használunk majd, az egyik a sejttér állapotát
     * a t_n, a másik a t_n+1 időpillanatban jellemzi. */
    protected boolean [][] [] rácok = new boolean [2][][];
    /** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
     * [2][][]-ból az első dimenziót használni, mert vagy az egyikre
     * állítjuk, vagy a másikra. */
    protected boolean [] [] rács;
    /** Megmutatja melyik rács az aktuális: [räcsIndex][][] */
    protected int rácsIndex = 0;
    /** Pixelben egy cella adatai. */
    protected int cellaSzélesség = 20;
    protected int cellaMagasság = 20;
    /** A sejttér nagysága, azaz hányszor hány cella van? */
    protected int szélesség = 20;
    protected int magasság = 10;
    /** A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
     * közötti valós idő. */
    protected int várakozás = 1000;
    // Pillanatfelvétel készítéséhez
    private java.awt.Robot robot;
    /** Készítsünk pillanatfelvételt? */
    private boolean pillanatfelvétel = false;
    /** A pillanatfelvételek számozásához. */
    private static int pillanatfelvételSzámláló = 0;
    /**
     * Létrehoz egy <code>Sejtautomata</code> objektumot.
     *
     * @param szélesség a sejttér szélessége.
```

```
* @param      magasság      a sejttér szélessége.  
*/  
public Sejtautomata(int szélesség, int magasság) {  
    this.szélesség = szélesség;  
    this.magasság = magasság;  
    // A két rács elkészítése  
    rácsok[0] = new boolean[magasság][szélesség];  
    rácsok[1] = new boolean[magasság][szélesség];  
    rácsIndex = 0;  
    rács = rácsok[rácsIndex];  
    // A kiinduló rács minden cellája HALOTT  
    for(int i=0; i<rács.length; ++i)  
        for(int j=0; j<rács[0].length; ++j)  
            rács[i][j] = HALOTT;  
    // A kiinduló rácsra "élőlényeket" helyezünk  
    //sikló(rács, 2, 2);  
    siklóKilövő(rács, 5, 60);  
    // Az ablak bezárásakor kilépünk a programból.  
    addWindowListener(new java.awt.event.WindowAdapter() {  
        public void windowClosing(java.awt.event.WindowEvent e) {  
            setVisible(false);  
            System.exit(0);  
        }  
    });  
    // A billentyűzetről érkező események feldolgozása  
    addKeyListener(new java.awt.event.KeyAdapter() {  
        // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük  
        public void keyPressed(java.awt.event.KeyEvent e) {  
            if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {  
                // Felezük a cella méreteit:  
                cellaSzélesség /= 2;  
                cellaMagasság /= 2;  
                setSize(Sejtautomata.this.szélesség*cellaSzélesség,  
                        Sejtautomata.this.magasság*cellaMagasság);  
                validate();  
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {  
                // Duplázzuk a cella méreteit:  
                cellaSzélesség *= 2;  
                cellaMagasság *= 2;  
                setSize(Sejtautomata.this.szélesség*cellaSzélesség,  
                        Sejtautomata.this.magasság*cellaMagasság);  
                validate();  
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)  
                pillanatfelvétel = !pillanatfelvétel;  
            else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)  
                várakozás /= 2;  
            else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)  
                várakozás *= 2;  
            repaint();  
        }  
    });
```

```
});  
// Egér kattintó események feldolgozása:  
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  
        int x = m.getX()/cellaSzélesség;  
        int y = m.getY()/cellaMagasság;  
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];  
        repaint();  
    }  
});  
// Egér mozgás események feldolgozása:  
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {  
    // Vonszolással jelöljük ki a négyzetet:  
    public void mouseDragged(java.awt.event.MouseEvent m) {  
        int x = m.getX()/cellaSzélesség;  
        int y = m.getY()/cellaMagasság;  
        rácsok[rácsIndex][y][x] = ÉLŐ;  
        repaint();  
    }  
});  
// Cellaméretek kezdetben  
cellaSzélesség = 10;  
cellaMagasság = 10;  
// Pillanatfelvétel készítéséhez:  
try {  
    robot = new java.awt.Robot(  
        java.awt.GraphicsEnvironment.  
        getLocalGraphicsEnvironment().  
        getDefaultScreenDevice());  
} catch(java.awt.AWTException e) {  
    e.printStackTrace();  
}  
// A program ablakának adatai:  
setTitle("Sejtautomata");  
setResizable(false);  
setSize(szélesség*cellaSzélesség,  
        magasság*cellaMagasság);  
setVisible(true);  
// A sejttér életrekeltése:  
new Thread(this).start();  
}  
/** A sejttér kirajzolása. */  
public void paint(java.awt.Graphics g) {  
    // Az aktuális  
    boolean [][] rács = rácsok[rácsIndex];  
    // rácsot rajzoljuk ki:
```

```
for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon
    for(int j=0; j<rács[0].length; ++j) { // s az oszlopok
        // Sejt cella kirajzolása
        if(rács[i][j] == ÉLŐ)
            g.setColor(java.awt.Color.BLACK);
        else
            g.setColor(java.awt.Color.WHITE);
        g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                   cellaSzélesség, cellaMagasság);
        // Rács kirajzolása
        g.setColor(java.awt.Color.LIGHT_GRAY);
        g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                   cellaSzélesség, cellaMagasság);
    }
}
// Készítünk pillanatfelvételt?
if(pillanatfelvétel) {
    // a biztonság kedvéért egy kép készítése után
    // kikapcsoljuk a pillanatfelvételt, hogy a
    // programmal ismerkedő Olvasó ne írja tele a
    // fájlrendszerét a pillanatfelvétellekkel
    pillanatfelvétel = false;
    pillanatfelvétel(robot.createScreenCapture
                      (new java.awt.Rectangle
                       (getLocation().x, getLocation().y,
                        szélesség*cellaSzélesség,
                        magasság*cellaMagasság)));
}
/**
 * Az kérdezett állapotban lévő nyolcszomszédok száma.
 *
 * @param rács a sejttér rács
 * @param sor a rács vizsgált sora
 * @param oszlop a rács vizsgált oszlopa
 * @param állapor a nyolcszomszédok vizsgált állapota
 * @return int a kérdezett állapotbeli nyolcszomszédok száma.
 */
public int szomszédochSzáma(boolean [][] rács,
                             int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 0;
    // A nyolcszomszédok végigzongorázása:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgált sejtet magát kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejttérből szélének szomszédai
                // a szembe oldalakon ("periódikus határfeltétel")
                int o = oszlop + j;
                if(o < 0)
```

```
        o = szélesség-1;
    else if(o >= szélesség)
        o = 0;

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

    if(rács[s][o] == állapot)
        ++állapotúSzomszéd;
    }

    return állapotúSzomszéd;
}
/***
 * A sejttér időbeli fejlődése a John H. Conway féle
 * életjáték sejtautomata szabályai alapján történik.
 * A szabályok részletes ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 171. oldal.)
 */
public void időFejlődés() {

    boolean [][] rácsElőtte = rácsok[rácsIndex];
    boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];

    for(int i=0; i<rácsElőtte.length; ++i) { // sorok
        for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok

            int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);

            if(rácsElőtte[i][j] == ÉLŐ) {
                /* Elő élő marad, ha kettő vagy három élő
                 szomszedja van, különben halott lesz. */
                if(élők==2 || élők==3)
                    rácsUtána[i][j] = ÉLŐ;
                else
                    rácsUtána[i][j] = HALOTT;
            } else {
                /* Halott halott marad, ha három élő
                 szomszedja van, különben élő lesz. */
                if(élők==3)
                    rácsUtána[i][j] = ÉLŐ;
                else
                    rácsUtána[i][j] = HALOTT;
            }
        }
    }
}
```

```
rácsIndex = (rácsIndex+1) %2;
}
/** A sejttér időbeli fejlődése. */
public void run() {

    while(true) {
        try {
            Thread.sleep(várakozás);
        } catch (InterruptedException e) {}

        időFejlődés();
        repaint();
    }
}
/**
 * A sejttérbe "élőlényeket" helyezünk, ez a "sikló".
 * Adott irányban halad, másolja magát a sejttérben.
 * Az élőlény ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 172. oldal.)
 *
 * @param rács      a sejttér ahová ezt az állatkát helyezzük
 * @param x         a befoglaló téglalap bal felső sarkának oszlopá
 * @param y         a befoglaló téglalap bal felső sarkának sora
 */
public void sikló(boolean[][][] rács, int x, int y) {

    rács[y+ 0][x+ 2] = ÉLŐ;
    rács[y+ 1][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 2] = ÉLŐ;
    rács[y+ 2][x+ 3] = ÉLŐ;

}
/**
 * A sejttérbe "élőlényeket" helyezünk, ez a "sikló ágyú".
 * Adott irányban siklókat lő ki.
 * Az élőlény ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban /Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 173. oldal./,
 * de itt az ábra hibás, egy oszloppal told még balra a
 * bal oldali 4 sejtes négyzetet. A helyes ágyú rajzát
 * láasd pl. az [ÉLET CIKK] hivatkozásban /Robert T.
 * Wainwright: Life is Universal./ (Megemlíthetjük, hogy
 * minden kettő tartalmaz két felesleges sejtet is.)
 *
 * @param rács      a sejttér ahová ezt az állatkát helyezzük
 * @param x         a befoglaló téglalap bal felső sarkának oszlopá
 * @param y         a befoglaló téglalap bal felső sarkának sora
*/
```

```
public void siklóKilövő(boolean [][] rács, int x, int y) {  
  
    rács[y+ 6][x+ 0] = ÉLŐ;  
    rács[y+ 6][x+ 1] = ÉLŐ;  
    rács[y+ 7][x+ 0] = ÉLŐ;  
    rács[y+ 7][x+ 1] = ÉLŐ;  
  
    rács[y+ 3][x+ 13] = ÉLŐ;  
  
    rács[y+ 4][x+ 12] = ÉLŐ;  
    rács[y+ 4][x+ 14] = ÉLŐ;  
  
    rács[y+ 5][x+ 11] = ÉLŐ;  
    rács[y+ 5][x+ 15] = ÉLŐ;  
    rács[y+ 5][x+ 16] = ÉLŐ;  
    rács[y+ 5][x+ 25] = ÉLŐ;  
  
    rács[y+ 6][x+ 11] = ÉLŐ;  
    rács[y+ 6][x+ 15] = ÉLŐ;  
    rács[y+ 6][x+ 16] = ÉLŐ;  
    rács[y+ 6][x+ 22] = ÉLŐ;  
    rács[y+ 6][x+ 23] = ÉLŐ;  
    rács[y+ 6][x+ 24] = ÉLŐ;  
    rács[y+ 6][x+ 25] = ÉLŐ;  
  
    rács[y+ 7][x+ 11] = ÉLŐ;  
    rács[y+ 7][x+ 15] = ÉLŐ;  
    rács[y+ 7][x+ 16] = ÉLŐ;  
    rács[y+ 7][x+ 21] = ÉLŐ;  
    rács[y+ 7][x+ 22] = ÉLŐ;  
    rács[y+ 7][x+ 23] = ÉLŐ;  
    rács[y+ 7][x+ 24] = ÉLŐ;  
  
    rács[y+ 8][x+ 12] = ÉLŐ;  
    rács[y+ 8][x+ 14] = ÉLŐ;  
    rács[y+ 8][x+ 21] = ÉLŐ;  
    rács[y+ 8][x+ 24] = ÉLŐ;  
    rács[y+ 8][x+ 34] = ÉLŐ;  
    rács[y+ 8][x+ 35] = ÉLŐ;  
  
    rács[y+ 9][x+ 13] = ÉLŐ;  
    rács[y+ 9][x+ 21] = ÉLŐ;  
    rács[y+ 9][x+ 22] = ÉLŐ;  
    rács[y+ 9][x+ 23] = ÉLŐ;  
    rács[y+ 9][x+ 24] = ÉLŐ;  
    rács[y+ 9][x+ 34] = ÉLŐ;  
    rács[y+ 9][x+ 35] = ÉLŐ;  
  
    rács[y+ 10][x+ 22] = ÉLŐ;  
    rács[y+ 10][x+ 23] = ÉLŐ;
```

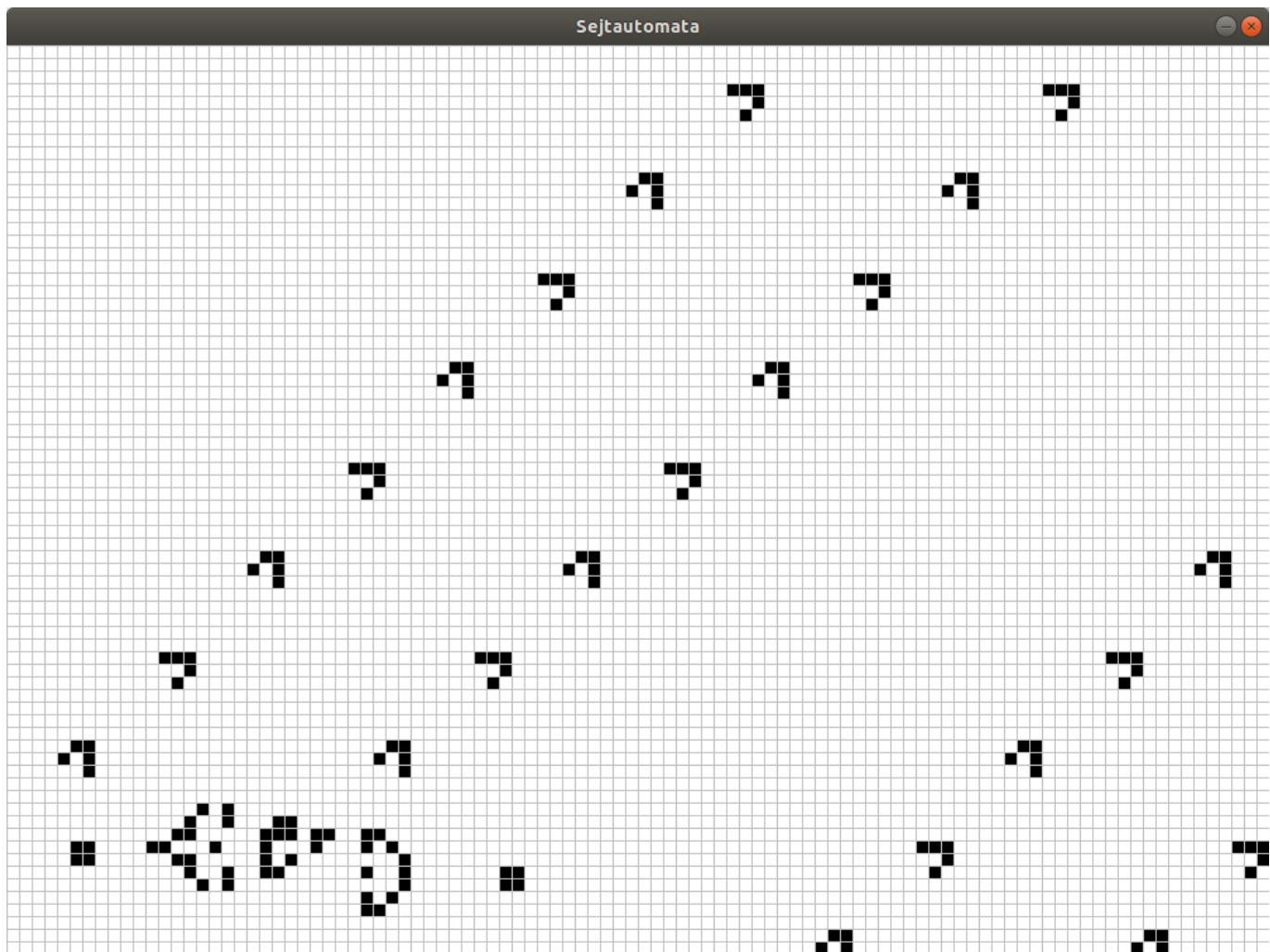
```
rács[y+ 10][x+ 24] = ÉLŐ;
rács[y+ 10][x+ 25] = ÉLŐ;

rács[y+ 11][x+ 25] = ÉLŐ;

}

/** Pillanatfelvételek készítése. */
public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    // A pillanatfelvétel kép fájlneve
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételszámláló);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
    paint(g);
}
/**
 * Példányosít egy Conway-féle életjáték szabályos
 * sejttér obektumot.
 */
public static void main(String[] args) {
    // 100 oszlop, 75 sor mérettel:
    new Sejtautomata(100, 75);
}
```

A program álltal készített sikló-kilővő működés közben:



6.3. ábra. sikló-kilövő

6.3. Qt C++ életjáték (Passz)

Most Qt C++-ban!

Megoldás forrása: <https://github.com/BorvizRobi/sejtautomata>

A mappában, ami tartalmazza a forráskódokat kiadjuk a következő parancsot:

```
~/Qt/5.12.2/gcc_64/bin/qmake -project
```

Értelemszerűen, onnan indítjuk a qmake parancsot, ahova telepítve van a Qt, a -project kapcsolóval létrehozzuk a project fájlt. Ez a fájl fogja tartalmazni azokat az információkat amelyek a qmake számára szükségesek a program létrehozásához.

Majd erre a létrehozott projekt fájlra ismételten futtatjuk a qmake-t:

```
~/Qt/5.12.2/gcc_64/bin/qmake sejtautomata.pro
```

A qmake egy Makefile-t fog készíteni nekünk a projektfájlból található infomációk alapján, majd a "make" parancsot kiadva elkészül a futatható állományunk amit a következő parancsal futathatunk:

```
./sejtautomata
```

Forráskódok:

sejtszal.h:

```
#ifndef SEJTSZAL_H
#define SEJTSZAL_H

#include <QThread>
#include "sejtablak.h"

class SejtAblak;

class SejtSzal : public QThread
{
    Q_OBJECT

public:
    SejtSzal(bool ***racsok, int szelesseg, int magassag,
              int varakozas, SejtAblak *sejtAblak);
    ~SejtSzal();
    void run();

protected:
    bool ***racsok;
    int szelesseg, magassag;
    // Megmutatja melyik rács az aktuális: [rácsIndex][][][]
    int racsIndex;
    // A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
    // közötti valós idő.
    int varakozas;
    void idoFejlodes();
    int szomszedokSzama(bool **racs,
                         int sor, int oszlop, bool allapot);
    SejtAblak* sejtAblak;

};

#endif // SEJTSZAL_H
```

sejtablak.h:

```
#ifndef SEJTABLAK_H
#define SEJTABLAK_H

#include <QMainWindow>
#include <QPainter>
#include "sejtszal.h"
```

```
class SejtSzal;

class SejtAblak : public QMainWindow
{
    Q_OBJECT

public:
    SejtAblak(int szelesseg = 100, int magassag = 75, QWidget *parent = 0);
    ~SejtAblak();
    // Egy sejt lehet élő
    static const bool ELO = true;
    // vagy halott
    static const bool HALOTT = false;
    void vissza(int racsIndex);

protected:
    // Két rácsot használunk majd, az egyik a sejttér állapotát
    // a t_n, a másik a t_n+1 időpillanatban jellemzi.
    bool ***racsok;
    // Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
    // [2][][]-ból az első dimenziót használni, mert vagy az egyikre
    // állítjuk, vagy a másikra.
    bool **racs;
    // Megmutatja melyik rács az aktuális: [räcsIndex] []
    int räcsIndex;
    // Pixelben egy cella adatai.
    int cellaSzelesseg;
    int cellaMagassag;
    // A sejttér nagysága, azaz hányszor hány cella van?
    int szelesseg;
    int magassag;
    void paintEvent(QPaintEvent* );
    void siklo(bool **racs, int x, int y);
    void sikloKilovo(bool **racs, int x, int y);

private:
    SejtSzal* eletjatek;

};

#endif // SEJTABLAK_H
```

sejtablak.cpp:

```
#include "sejtablak.h"

SejtAblak::SejtAblak(int szelesseg, int magassag, QWidget *parent)
    : QMainWindow(parent)
{
```

```
setWindowTitle("A John Horton Conway-féle életjáték");

this->magassag = magassag;
this->szelesseg = szelesseg;

cellaSzelesseg = 6;
cellaMagassag = 6;

setFixedSize(QSize(szelesseg*cellaSzelesseg, magassag*cellaMagassag));

racsok = new bool**[2];
racsok[0] = new bool*[magassag];
for(int i=0; i<magassag; ++i)
    racsok[0][i] = new bool [szelesseg];
racsok[1] = new bool*[magassag];
for(int i=0; i<magassag; ++i)
    racsok[1][i] = new bool [szelesseg];

racsIndex = 0;
racs = racsok[racsIndex];
// A kiinduló racs minden cellája HALOTT
for(int i=0; i<magassag; ++i)
    for(int j=0; j<szelesseg; ++j)
        racs[i][j] = HALOTT;
// A kiinduló racsra "ELOlényeket" helyezünk
//siklo(racs, 2, 2);
sikloKilovo(racs, 5, 60);

eletjatek = new SejtSzal(racsok, szelesseg, magassag, 120, this);
eletjatek->start();

}

void SejtAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);

    // Az aktuális
    bool **racs = racsok[racsIndex];
    // racsot rajzoljuk ki:
    for(int i=0; i<magassag; ++i) { // végig lépked a sorokon
        for(int j=0; j<szelesseg; ++j) { // s az oszlopok
            // Sejt cella kirajzolása
            if(racs[i][j] == ELO)
                qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                                  cellaSzelesseg, cellaMagassag, Qt::black) ←
                ;
            else
                qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                                  cellaSzelesseg, cellaMagassag, Qt::white) ←
```

```
        ;
    qpainter.setPen(QPen(Qt::gray, 1));

    qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
                      cellaSzelesseg, cellaMagassag);
}
}

qpainter.end();
}

SejtAblak::~SejtAblak()
{
    delete eletjatek;

    for(int i=0; i<magassag; ++i) {
        delete[] racsok[0][i];
        delete[] racsok[1][i];
    }

    delete[] racsok[0];
    delete[] racsok[1];
    delete[] racsok;
}

void SejtAblak::vissza(int racsIndex)
{
    this->racsIndex = racsIndex;
    update();
}

/**
 * A sejttérbe "ELOlényeket" helyezünk, ez a "sikló".
 * Adott irányban halad, másolja magát a sejttérben.
 * Az ELOlény ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 172. oldal.)
 *
 * @param racs a sejttér ahová ezt az állatkát helyezzük
 * @param x a befoglaló téglalal bal felső sarkának oszlopa
 * @param y a befoglaló téglalal bal felső sarkának sora
 */
void SejtAblak::siklo(bool **racs, int x, int y) {

    racs[y+ 0][x+ 2] = ELO;
    racs[y+ 1][x+ 1] = ELO;
    racs[y+ 2][x+ 1] = ELO;
```

```
racs[y+ 2][x+ 2] = ELO;
racs[y+ 2][x+ 3] = ELO;

}

/** 
 * A sejttérbe "ELOlényeket" helyezünk, ez a "sikló ágyú".
 * Adott irányban siklókat lő ki.
 * Az ELOlény ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban /Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 173. oldal./,
 * de itt az ábra hibás, egy oszloppal told még balra a
 * bal oldali 4 sejtes négyzetet. A helyes ágyú rajzát
 * lásd pl. az [ÉLET CIKK] hivatkozásban /Robert T.
 * Wainwright: Life is Universal./ (Megemlíthetjük, hogy
 * minden kettő tartalmaz két felesleges sejtet is.)
 *
 * @param racs    a sejttér ahol ezt az állatkát helyezzük
 * @param x      a befoglaló téglalap bal felső sarkának oszlopa
 * @param y      a befoglaló téglalap bal felső sarkának sora
 */
void SejtAblak::sikloKilovo(bool **racs, int x, int y) {

    racs[y+ 6][x+ 0] = ELO;
    racs[y+ 6][x+ 1] = ELO;
    racs[y+ 7][x+ 0] = ELO;
    racs[y+ 7][x+ 1] = ELO;

    racs[y+ 3][x+ 13] = ELO;

    racs[y+ 4][x+ 12] = ELO;
    racs[y+ 4][x+ 14] = ELO;

    racs[y+ 5][x+ 11] = ELO;
    racs[y+ 5][x+ 15] = ELO;
    racs[y+ 5][x+ 16] = ELO;
    racs[y+ 5][x+ 25] = ELO;

    racs[y+ 6][x+ 11] = ELO;
    racs[y+ 6][x+ 15] = ELO;
    racs[y+ 6][x+ 16] = ELO;
    racs[y+ 6][x+ 22] = ELO;
    racs[y+ 6][x+ 23] = ELO;
    racs[y+ 6][x+ 24] = ELO;
    racs[y+ 6][x+ 25] = ELO;

    racs[y+ 7][x+ 11] = ELO;
    racs[y+ 7][x+ 15] = ELO;
    racs[y+ 7][x+ 16] = ELO;
    racs[y+ 7][x+ 21] = ELO;
    racs[y+ 7][x+ 22] = ELO;
```

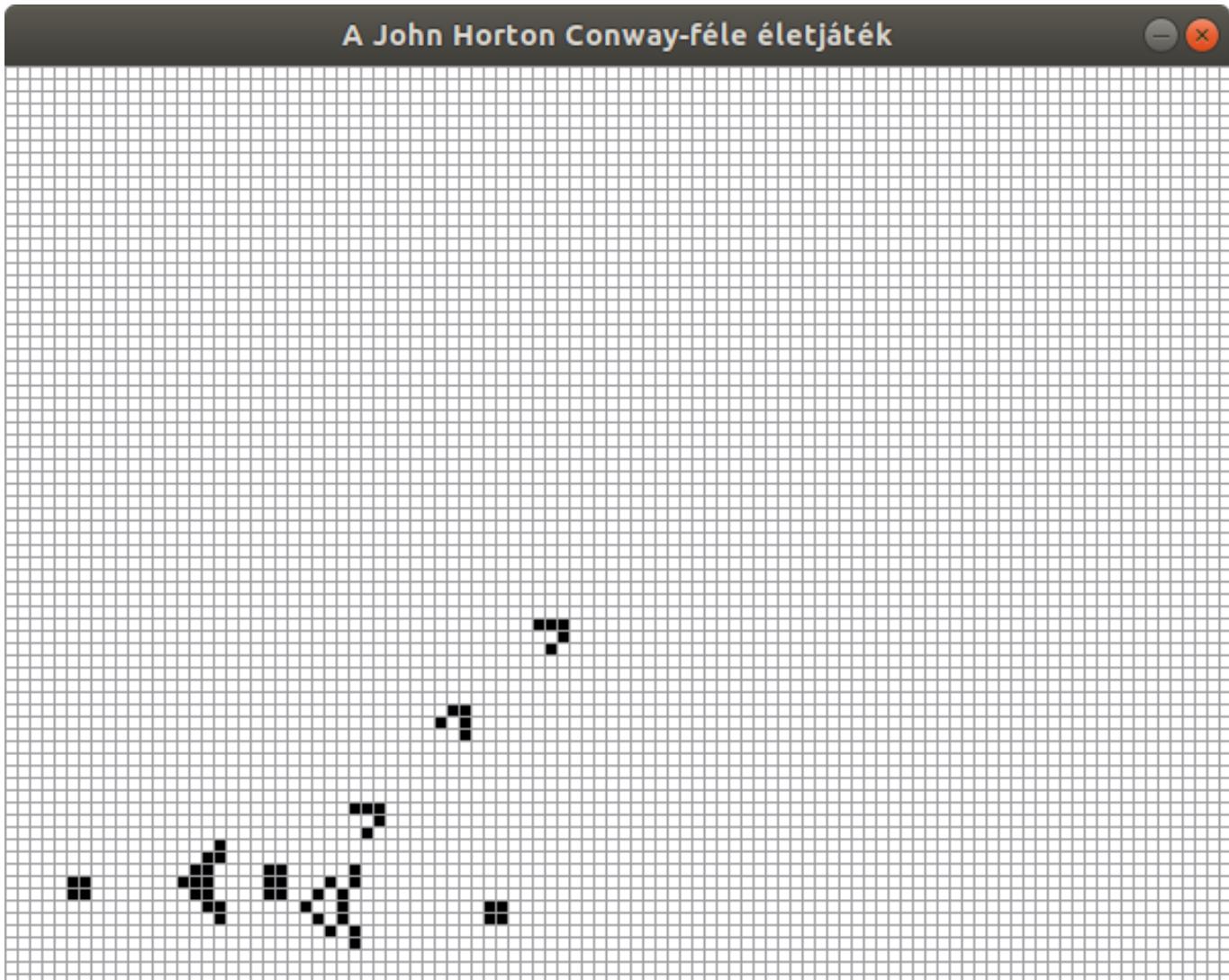
```
racs[y+ 7][x+ 23] = ELO;  
racs[y+ 7][x+ 24] = ELO;  
  
racs[y+ 8][x+ 12] = ELO;  
racs[y+ 8][x+ 14] = ELO;  
racs[y+ 8][x+ 21] = ELO;  
racs[y+ 8][x+ 24] = ELO;  
racs[y+ 8][x+ 34] = ELO;  
racs[y+ 8][x+ 35] = ELO;  
  
racs[y+ 9][x+ 13] = ELO;  
racs[y+ 9][x+ 21] = ELO;  
racs[y+ 9][x+ 22] = ELO;  
racs[y+ 9][x+ 23] = ELO;  
racs[y+ 9][x+ 24] = ELO;  
racs[y+ 9][x+ 34] = ELO;  
racs[y+ 9][x+ 35] = ELO;  
  
racs[y+ 10][x+ 22] = ELO;  
racs[y+ 10][x+ 23] = ELO;  
racs[y+ 10][x+ 24] = ELO;  
racs[y+ 10][x+ 25] = ELO;  
  
racs[y+ 11][x+ 25] = ELO;
```

}

main.cpp:

```
#include <QApplication>  
#include "sejtablak.h"  
  
int main(int argc, char *argv[])  
{  
    QApplication a(argc, argv);  
    SejtAblak w(100, 75);  
    w.show();  
  
    return a.exec();  
}
```

Ez a program is ugyanazt az alakzatot produkálja mint az előző:



6.4. ábra. Sikló-kilövő

6.4. BrainB Benchmark (Passz)

Megoldás forrása: <https://github.com/nbatfai/esport-talent-search>

A BrainB feladata a tehetségkutatás az esportban, az egyes játékokban (pl:League of legends) előforduló 'karakterelvesztést' fogja előidézni, a karaktervesztés akkor következik be amikor egyszerre annyi minden történik a képernyőn, hogy nem tudjuk már követni hogy hol is van a karakterünk.

A program 10 percig fut, ezalatt az idő alatt az a feladatunk hogy a bal egérgombot lenyomva Samu Entropy-n tartuk az egeret. A program futás közben statisztikát készít az eredményeinkről amit a program végeztével megtekinthetünk.

A program használatához a következők telepítése a szükséges:

```
sudo apt-get install libqt4-dev  
sudo apt-get install opencv-data
```

```
sudo apt-get install libopencv-dev
```

Az előző programokhoz hasonlóan futtajuk a qmake-t:

```
~/Qt/5.12.2/gcc_64/bin/qmake BrainB.pro
```

Majd a make parancssal futtatjuk a létrehozott Makefile-t.

Ezután már futtatható a program:

```
./BrainB
```

Foráskódok:

BrainBWin.h:

```
#ifndef BrainBWin_H
#define BrainBWin_H

#include <QKeyEvent>
#include <QMainWindow>
#include <QPixmap>
#include <QPainter>
#include <QFont>
#include <QFile>
#include <QString>
#include <QCloseEvent>
#include <QDate>
#include <QDir>
#include <QDateTime>
#include "BrainBThread.h"

enum playerstate {
    lost,
    found
};

class BrainBWin : public QMainWindow
{
    Q_OBJECT

    BrainBThread *brainBThread;
    QPixmap pixmap;
    Heroes *heroes;

    int mouse_x;
    int mouse_y;
    int yshift {50};
    int nofLost {0};
    int nofFound {0};

    int xs, ys;
```

```
bool firstLost {false};
bool start {false};
playerstate state = lost;
std::vector<int> lost2found;
std::vector<int> found2lost;

QString statDir;

public:
    static const QString appName;
    static const QString appVersion;
    BrainBWin ( int w = 256, int h = 256, QWidget *parent = 0 );
    void closeEvent ( QCloseEvent *e ) {
        if ( save ( brainBThread->getT() ) ) {
            brainBThread->finish();
            e->accept();
        } else {
            e->ignore();
        }
    }
    virtual ~BrainBWin();
    void paintEvent ( QPaintEvent * );
    void keyPressEvent ( QKeyEvent *event );
    void mouseMoveEvent ( QMouseEvent *event );
    void mousePressEvent ( QMouseEvent *event );
    void mouseReleaseEvent ( QMouseEvent *event );

    double mean ( std::vector<int> vect ) {
        if ( vect.size() > 0 ) {
            double sum = std::accumulate ( vect.begin (), vect.end (), 0.0 );
            return sum / vect.size();
        } else {
            return 0.0;
        }
    }

    double var ( std::vector<int> vect, double mean ) {
        if ( vect.size() > 1 ) {
            double accum = 0.0;
            std::for_each ( vect.begin (), vect.end (), [&] ( const double &
```

```
        d ) {
            accum += ( d - mean ) * ( d - mean );
        } );
    }

    return sqrt ( accum / ( vect.size()-1 ) );
} else {
    return 0.0;
}

}

void millis2minsec ( int millis, int &min, int &sec ) {

    sec = ( millis * 100 ) / 1000;
    min = sec / 60;
    sec = sec - min * 60;

}

bool save ( int t ) {

    bool ret = false;

    if ( !QDir ( statDir ).exists() )
        if ( !QDir().mkdir ( statDir ) ) {
            return false;
        }

    QString name = statDir + "/Test-" + QString::number ( t );
    QFile file ( name + "-screenimage.png" );
    if ( file.open ( QIODevice::WriteOnly ) ) {
        ret = pixmap.save ( &file, "PNG" );
    }

    QFile tfile ( name + "-stats.txt" );
    ret = tfile.open ( QIODevice::WriteOnly | QIODevice::Text );
    if ( ret ) {
        QTextStream textStremam ( &tfile );

        textStremam << appName + " " + appVersion << "\n";
        textStremam << "time      : " << brainBThread->getT() << "\n";
        textStremam << "bps       : " << brainBThread->get_bps() << "\n";
        textStremam << "noc      : " << brainBThread->nofHeroes() << "\n";
        textStremam << "nop      : " << brainBThread->get_nofPaused() << "\n";

        textStremam << "lost      : " << "\n";
        std::vector<int> l = brainBThread->lostV();
```

```
for ( int n : l ) {
    textStremam << n << ' ';
}
textStremam << "\n";
int m = mean ( l );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( l, m ) << "\n";

textStremam << "found      : " ;
std::vector<int> f = brainBThread->foundV();
for ( int n : f ) {
    textStremam << n << ' ';
}
textStremam << "\n";
m = mean ( f );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( f, m ) << "\n";

textStremam << "lost2found: " ;
for ( int n : lost2found ) {
    textStremam << n << ' ';
}
textStremam << "\n";
int m1 = m = mean ( lost2found );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( lost2found, m ) << "\n" <-
"';

textStremam << "found2lost: " ;
for ( int n : found2lost ) {
    textStremam << n << ' ';
}
textStremam << "\n";
int m2 = m = mean ( found2lost );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( found2lost, m ) << "\n" <-
"';

if ( m1 < m2 ) {
    textStremam << "mean(lost2found) < mean(found2lost)" << "\n" <-
"";
}

int min, sec;
millis2minsec ( t, min, sec );
textStremam << "time      : " << min << ":" << sec << "\n";

double res = ( ( ( double ) m1+ ( double ) m2 ) /2.0 ) /8.0 ) <-
/1024.0;
textStremam << "U R about " << res << " Kilobytes\n";
```

```
        tfile.close();
    }
    return ret;
}

public slots :

void updateHeroes ( const QImage &image, const int &x, const int &y );
//void stats ( const int &t );
void endAndStats ( const int &t );
};

#endif // BrainBWin
```

BrainBWin.cpp:

```
#include "BrainBWin.h"

const QString BrainBWin::appName = "NEMESPOR BrainB Test";
const QString BrainBWin::appVersion = "6.0.3";

BrainBWin::BrainBWin ( int w, int h, QWidget *parent ) : QMainWindow ( ←
    parent )
{
    //    setWindowTitle(appName + " " + appVersion);
    //    setFixedSize(QSize(w, h));

    statDir = appName + " " + appVersion + " - " + QDate::currentDate() ←
        .toString() + QString::number ( QDateTime::←
        currentMSecsSinceEpoch() );

    brainBThread = new BrainBThread ( w, h - yshift );
    brainBThread->start();

    connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ←
        ),
              this, SLOT ( updateHeroes ( QImage, int, int ) ) );

    connect ( brainBThread, SIGNAL ( endAndStats ( int ) ),
              this, SLOT ( endAndStats ( int ) ) );
}

void BrainBWin::endAndStats ( const int &t )
{
    qDebug() << "\n\n\n";
    qDebug() << "Thank you for using " + appName;
    qDebug() << "The result can be found in the directory " + statDir;
```

```
qDebug() << "\n\n\n";  
  
    save ( t );  
    close();  
}  
  
void BrainBWin::updateHeroes ( const QImage &image, const int &x, const int &  
    &y )  
{  
  
    if ( start && !brainBThread->get_paused() ) {  
  
        int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) + &  
            ( this->mouse_y - y ) * ( this->mouse_y - y );  
  
        if ( dist > 121 ) {  
            ++nofLost;  
            nofFound = 0;  
            if ( nofLost > 12 ) {  
  
                if ( state == found && firstLost ) {  
                    found2lost.push_back ( brainBThread &  
                        ->get_bps() );  
                }  
  
                firstLost = true;  
  
                state = lost;  
                nofLost = 0;  
                //qDebug() << "LOST";  
                //double mean = brainBThread->meanLost();  
                //qDebug() << mean;  
  
                brainBThread->decComp();  
            }  
        } else {  
            ++nofFound;  
            nofLost = 0;  
            if ( nofFound > 12 ) {  
  
                if ( state == lost && firstLost ) {  
                    lost2found.push_back ( brainBThread &  
                        ->get_bps() );  
                }  
  
                state = found;  
                nofFound = 0;  
                //qDebug() << "FOUND";  
                //double mean = brainBThread->meanFound();  
                //qDebug() << mean;  
            }  
        }  
    }  
}
```

```
        brainBThread->incComp();
    }

}

pixmap = QPixmap::fromImage ( image );
update();
}

void BrainBWin::paintEvent ( QPaintEvent * )
{
    if ( pixmap.isNull() ) {
        return;
    }

    QPainter qpainter ( this );

    xs = ( qpainter.device()->width() - pixmap.width() ) /2;
    ys = ( qpainter.device()->height() - pixmap.height() +yshift ) /2;

    qpainter.drawPixmap ( xs, ys, pixmap );

    qpainter.drawText ( 10, 20, "Press and hold the mouse button on the ←
center of Samu Entropy" );

    int time = brainBThread->getT();
    int min, sec;
    millis2minsec ( time, min, sec );
    QString timestr = QString::number ( min ) + ":" + QString::number ( ←
        sec ) + "/10:0";
    qpainter.drawText ( 10, 40, timestr );

    int bps = brainBThread->get_bps();
    QString bpsstr = QString::number ( bps ) + " bps";
    qpainter.drawText ( 110, 40, bpsstr );

    if ( brainBThread->get_paused() ) {
        QString pausedstr = "PAUSED (" + QString::number ( ←
            brainBThread->get_nofPaused() ) + ")";
        qpainter.drawText ( 210, 40, pausedstr );
    }

    qpainter.end();
}

void BrainBWin::mousePressEvent ( QMouseEvent *event )
{
```

```
brainBThread->set_paused ( false );  
}  
  
void BrainBWin::mouseReleaseEvent ( QMouseEvent *event )  
{  
  
    //brainBThread->set_paused(true);  
  
}  
  
void BrainBWin::mouseMoveEvent ( QMouseEvent *event )  
{  
  
    start = true;  
  
    mouse_x = event->pos().x() -xs - 60;  
    //mouse_y = event->pos().y() - yshift - 60;  
    mouse_y = event->pos().y() - ys - 60;  
}  
  
void BrainBWin::keyPressEvent ( QKeyEvent *event )  
{  
  
    if ( event->key() == Qt::Key_S ) {  
        save ( brainBThread->getT() );  
    } else if ( event->key() == Qt::Key_P ) {  
        brainBThread->pause();  
    } else if ( event->key() == Qt::Key_Q || event->key() == Qt::Key_Escape ) {  
        close();  
    }  
}  
  
BrainBWin::~BrainBWin()  
{  
}
```

BrainBThread.h:

```
#ifndef BrainBThread_H  
#define BrainBThread_H  
  
#include <QThread>  
#include <QSize>  
#include <QImage>  
#include <QDebug>
```

```
#include <sstream>
#include <QPainter>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>

class Hero;
typedef std::vector<Hero> Heroes;

class Hero
{

public:
    int x;
    int y;
    int color;
    int agility;
    int conds {0};
    std::string name;

    Hero ( int x=0, int y=0, int color=0, int agility=1, std::string name ←
          ="Samu Entropy" ) :
        x ( x ), y ( y ), color ( color ), agility ( agility ), name ( name ←
          )
    {}
    ~Hero() {}

    void move ( int maxx, int maxy, int env ) {

        int newx = x+ ( ( double ) agility*1.0 ) * ( double ) ( std::rand ←
            () / ( RAND_MAX+1.0 ) )-agility/2 ;
        if ( newx-env > 0 && newx+env < maxx ) {
            x = newx;
        }
        int newy = y+ ( ( double ) agility*1.0 ) * ( double ) ( std::rand ←
            () / ( RAND_MAX+1.0 ) )-agility/2 ;
        if ( newy-env > 0 && newy+env < maxy ) {
            y = newy;
        }

    }

};

class BrainBThread : public QThread
{
    Q_OBJECT
```

```
//Norbi
cv::Scalar cBg { 247, 223, 208 };
cv::Scalar cBorderAndText { 47, 8, 4 };
cv::Scalar cCenter { 170, 18, 1 };
cv::Scalar cBoxes { 10, 235, 252 };

/*
//Matyi
cv::Scalar cBg { 86, 26, 228 };
cv::Scalar cBorderAndText { 14, 177, 232 };
cv::Scalar cCenter { 232, 14, 103 };
cv::Scalar cBoxes { 14, 232, 195 };
*/
Heroes heroes;
int heroRectSize {40};

cv::Mat prev {3*heroRectSize, 3*heroRectSize, CV_8UC3, cBg };
int bps;
long time {0};
long endTime {10*60*10};
int delay {100};

bool paused {true};
int nofPaused {0};

std::vector<int> lostBPS;
std::vector<int> foundBPS;

int w;
int h;
int dispShift {40};

public:
    BrainBThread ( int w = 256, int h = 256 );
    ~BrainBThread();

    void run();
    void pause();
    void set_paused ( bool p );
    int getDelay() const {

        return delay;

    }
    void setDelay ( int delay ) {
```

```
if ( delay > 0 ) {
    delay = delay;
}

void devel() {

    for ( Hero & hero : heroes ) {

        hero.move ( w, h, ( h<w ) ?h/10:w/10 );

    }

}

int nofHeroes () {

    return heroes.size();

}

std::vector<int> &lostV () {

    return lostBPS;

}

std::vector<int> &foundV () {

    return foundBPS;

}

double meanLost () {

    return mean ( lostBPS );

}

double varLost ( double mean ) {

    return var ( lostBPS, mean );

}

double meanFound () {

    return mean ( foundBPS );
```

```
}

double varFound ( double mean ) {

    return var ( foundBPS, mean );

}

double mean ( std::vector<int> vect ) {

    double sum = std::accumulate ( vect.begin (), vect.end (), 0.0 );
    return sum / vect.size();

}

double var ( std::vector<int> vect, double mean ) {

    double accum = 0.0;
    std::for_each ( vect.begin (), vect.end (), [&] ( const double d ) ←
    {
        accum += ( d - mean ) * ( d - mean );
    } );

    return sqrt ( accum / ( vect.size()-1 ) );
}

int get_bps() const {

    return bps;

}

int get_w() const {

    return w;

}

bool get_paused() const {

    return paused;

}

int get_nofPaused() const {

    return nofPaused;

}
```

```
void decComp() {  
  
    lostBPS.push_back ( bps );  
  
    if ( heroes.size() > 1 ) {  
        heroes.pop_back();  
    }  
  
    for ( Hero & hero : heroes ) {  
        if ( hero.agility >= 5 ) {  
            hero.agility -= 2;  
        }  
    }  
  
}  
  
void incComp() {  
  
    foundBPS.push_back ( bps );  
  
    if ( heroes.size() > 300 ) {  
  
        return;  
    }  
  
    /*  
    Hero other ( w/2 + 200.0*std::rand() / ( RAND_MAX+1.0 )-100,  
                 h/2 + 200.0*std::rand() / ( RAND_MAX+1.0 )-100,  
                 255.0*std::rand() / ( RAND_MAX+1.0 ), 11, "New Entropy ←  
                 " );  
    */  
  
    double rx = 200.0;  
    if(heroes[0].x - 200 < 0)  
        rx = heroes[0].x;  
    else if(heroes[0].x + 200 > w)  
        rx = w - heroes[0].x;  
  
    double ry = 200.0;  
    if(heroes[0].y - 200 < 0)  
        ry = heroes[0].y;  
    else if(heroes[0].y + 200 > h)  
        ry = h - heroes[0].y;  
  
    Hero other ( heroes[0].x + rx*std::rand() / ( RAND_MAX+1.0 )-rx/2,  
                 heroes[0].y + ry*std::rand() / ( RAND_MAX+1.0 )-ry/2,  
                 255.0*std::rand() / ( RAND_MAX+1.0 ), 11, "New Entropy ←
```

```
    " ) ;

heroes.push_back ( other );

for ( Hero & hero : heroes ) {

    ++hero.conds;
    if ( hero.conds == 3 ) {
        hero.conds = 0;
        hero.agility += 2;
    }
}

void draw () {

cv::Mat src ( h+3*heroRectSize, w+3*heroRectSize, CV_8UC3, cBg );

for ( Hero & hero : heroes ) {

    cv::Point x ( hero.x-heroRectSize+dispShift, hero.y- ↵
        heroRectSize+dispShift );
    cv::Point y ( hero.x+heroRectSize+dispShift, hero.y+ ↵
        heroRectSize+dispShift );

    cv::rectangle ( src, x, y, cBorderAndText );

    cv::putText ( src, hero.name, x, cv::FONT_HERSHEY_SIMPLEX, .35, ↵
        cBorderAndText, 1 );

    cv::Point xc ( hero.x+dispShift , hero.y+dispShift );

    cv::circle ( src, xc, 11, cCenter, CV_FILLED, 8, 0 );

    cv::Mat box = src ( cv::Rect ( x, y ) );

    cv::Mat cbox ( 2*heroRectSize, 2*heroRectSize, CV_8UC3, cBoxes ↵
        );
    box = cbox*.3 + box*.7;

}

cv::Mat comp;

cv::Point focusx ( heroes[0].x- ( 3*heroRectSize ) /2+dispShift, ↵
    heroes[0].y- ( 3*heroRectSize ) /2+dispShift );
cv::Point focusy ( heroes[0].x+ ( 3*heroRectSize ) /2+dispShift, ↵
    heroes[0].y+ ( 3*heroRectSize ) /2+dispShift );
```

```
cv::Mat focus = src ( cv::Rect ( focusx, focusy ) );

cv::compare ( prev, focus, comp, cv::CMP_NE );

cv::Mat aRgb;
cv::extractChannel ( comp, aRgb, 0 );

bps = cv::countNonZero ( aRgb ) * 10;

//qDebug() << bps << " bits/sec";

prev = focus;

QImage dest ( src.data, src.cols, src.rows, src.step, QImage::Format_RGB888 );
dest=dest.rgbSwapped();
dest.bits();

emit heroesChanged ( dest, heroes[0].x, heroes[0].y );

}

long getT() const {

    return time;

}

void finish () {

    time = endTime;

}

signals:

void heroesChanged ( const QImage &image, const int &x, const int &y );
void endAndStats ( const int &t );

};

#endif // BrainBThread_H
```

BrainBThread.cpp:

```
#include "BrainBThread.h"

BrainBThread::BrainBThread ( int w, int h )
{
```

```
    dispShift = heroRectSize+heroRectSize/2;

    this->w = w - 3 * heroRectSize;
    this->h = h - 3 * heroRectSize;

    std::srand ( std::time ( 0 ) );

    Hero me ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100,
              this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 9 );

    Hero other1 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  5, "Norbi Entropy" );
    Hero other2 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  3, "Greta Entropy" );
    Hero other4 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  5, "Nandi Entropy" );
    Hero other5 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  7, "Matyi Entropy" );

    heroes.push_back ( me );
    heroes.push_back ( other1 );
    heroes.push_back ( other2 );
    heroes.push_back ( other4 );
    heroes.push_back ( other5 );

}

BrainBThread::~BrainBThread()
{
}

void BrainBThread::run()
{
    while ( time < endTime ) {
```

```
QThread::msleep ( delay );

    if ( !paused ) {

        ++time;

        devel();

    }

    draw();

}

emit endAndStats ( endTime );

}

void BrainBThread::pause()
{

    paused = !paused;
    if ( paused ) {
        ++nofPaused;
    }

}

void BrainBThread::set_paused ( bool p )
{

    if ( !paused && p ) {
        ++nofPaused;
    }

    paused = p;

}
```

Main.cpp:

```
#include <QApplication>
#include <QTTextStream>
#include <QtWidgets>
#include "BrainBWin.h"

int main ( int argc, char **argv )
{
    QApplication app ( argc, argv );
    QTextStream qout ( stdout );
```

```
qout.setCodec ( "UTF-8" );

qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( " ↵
Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;

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

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

qout << QString::fromUtf8 ( "Ez a program szabad szoftver; ↵
terjeszthető illetve módosítható a Free Software" ) << endl;
qout << QString::fromUtf8 ( "Foundation által kiadott GNU General ↵
Public License dokumentumában leírtak;" ) << endl;
qout << QString::fromUtf8 ( "akár a licenc 3-as, akár (tetszőleges) ↵
későbbi változata szerint.\n" ) << endl;

qout << QString::fromUtf8 ( "Ez a program abban a reményben kerül ↵
közreadásra, hogy hasznos lesz, de minden" ) << endl;
qout << QString::fromUtf8 ( "egyéb GARANCIA NÉLKÜL, az ↵
ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ" ) << endl;
qout << QString::fromUtf8 ( "ALKALMAZHATÓSÁGRA való származtatott ↵
garanciát is beleértve. További" ) << endl;
qout << QString::fromUtf8 ( "részleteket a GNU General Public ↵
License tartalmaz.\n" ) << endl;

qout << "http://gnu.hu/gplv3.html" << endl;

QRect rect = QApplication::desktop()->availableGeometry();
BrainBWin brainBWin ( rect.width(), rect.height() );
brainBWin.setWindowState ( brainBWin.windowState() ^ Qt:: ↵
WindowFullScreen );
brainBWin.show();
return app.exec();
}
```

Futás közben:



6.5. ábra. BrainB

7. fejezet

Helló, Schwarzenegger!

7.1. Szoftmax Py MNIST (Passz)

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Mély MNIST (Passz)

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Minecroft-MALMÖ (Passz)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Chaitin!

8.1. Iteratív és rekurzív faktoriális Lisp-ben

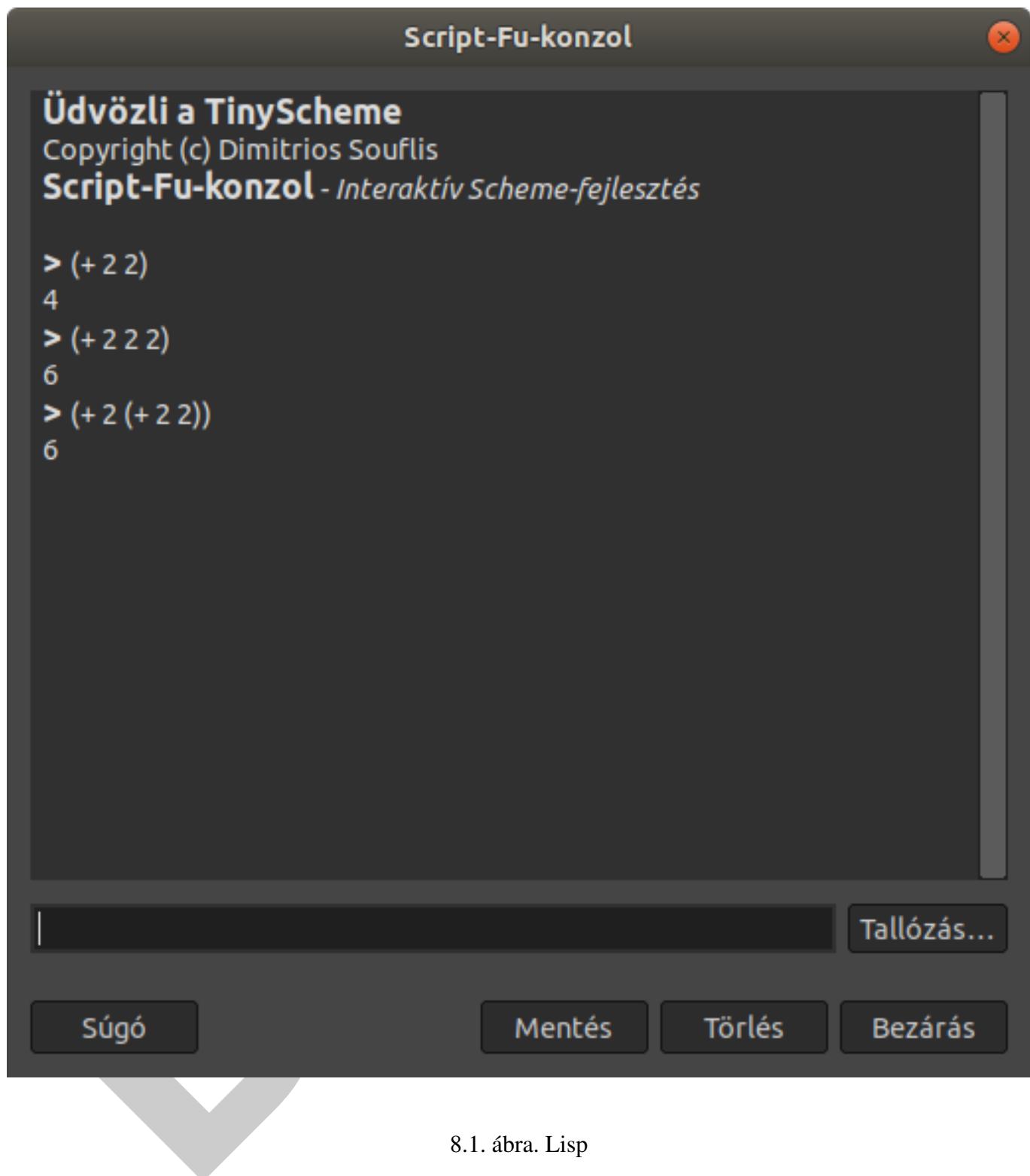
Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

A Gimp egy bittérképes képszerkesztő program, ez azt jelenti, hogy minden egyes pixel külön van definiálva a szerkesztendő képen. A gimp sok lehetőséget biztosít számunkra például: rengeteg szerkesztő eszköz megtalálható benne: ecset, toll festékszóró, a szerkeszteni kívánt kép méretének csak a tárhely szab határt, külső programokból hívható funkciók és makrók, több képformárum támogatása és még sorolhatnám.

A Gimp műveleteit automatizálni lehet script nyelvekkel, a Script-Fu egy Scheme alapú nyelv, ami a Gimp-be épített TinyScheme interpreter által van implementálva. A Scheme programozási nyelv a Lisp nyelvcsalád egyik tagja.

A Lisp név a List Processing névre vezethető vissza, a Lisp nyelvek legfőbb adatstruktúrája a láncolt lista. Egy Lisp nyelven írt kód azonnal könnyen felismerhető a szintaktikájáról, mivel a program egymásba ágyazott kifejezések sorozata. A következő feladatokat ilyen nyelven fogom elkészíteni.

Ismerkedés a Lips-el:



8.1. ábra. Lisp

A fentebb bevitt kifejezések például a következők megfelelői:

$$(+ 2 2) = 2+2$$

$$(+ 2 2 2) = 2+2+2$$

$$(+ 2 (+ 2 2)) = 2+2+2$$

Térjünk rá akkor a faktoriálisra. Egy n nem negatív egész szám faktoriálisának az n-nél kisebb vagy egyenlő pozitív egész számok szorzatát nevezzük. Így jelöljük: n!

$$4! = 1 \cdot 2 \cdot 3 \cdot 4$$

$$1! = 1$$

$$0! = 1$$

A fenti meghatározás alapján a rekurzív faktoriális Lisp-ben:

DRAFT

The screenshot shows a terminal window titled "Script-Fu-konzol". The console output is as follows:

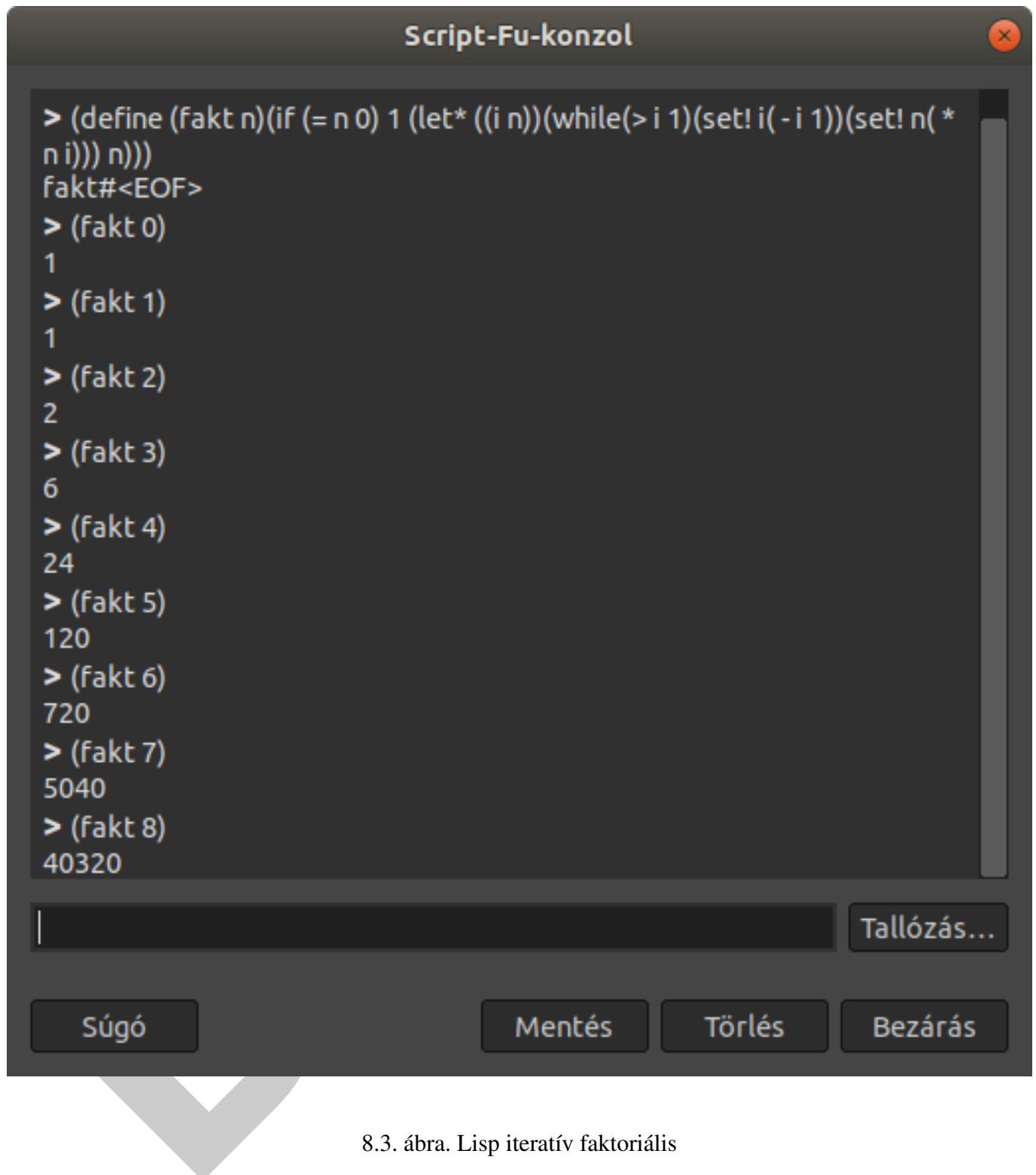
```
> (define (fakt n) (if(< n 1) 1 (* n (fakt(- n 1)))))  
fakt  
> (fakt 1)  
1  
> (fakt 2)  
2  
> (fakt 3)  
6  
> (fakt 4)  
24  
> (fakt 5)  
120  
> (fakt 6)  
720  
> (fakt 7)  
5040  
> (fakt 8)  
40320  
> (fakt 0)  
1
```

The console has a dark background. At the bottom, there are four buttons: "Súgó" (Help), "Mentés" (Save), "Törlés" (Delete), and "Bezárás" (Close). To the right of the "Bezárás" button is a "Tallózás..." (Search...) button.

8.2. ábra. Lisp Rekurzív faktoriális

A define után adjuk meg, hogy mit határozunk meg itt (fakt n), n lesz az argumentuma. A következő zárójelben azt adjuk meg hogy (fakt n) mivel egyenlő, de a következő rész egy if-es kifejezést tartalmaz aminek a jelentése a következő: ha az if után lévő zárójelben lévő kifejezés igaz, akkor a zárójel utáni kifejezés hajtódik végre itt az 1, ha nem akkor az azutáni vagyis itt a (* n (fakt(- n 1))).

Az iteratív faktoriális:



8.3. ábra. Lisp iteratív faktoriális

```
(define (fakt n)(if (= n 0) 1 (let* ((i n))(while(> i 1)(set! i(- i 1))(←
    set! n(* n i))))n)))
```

A sok zárójel miatt elég nehezen olvasható, érdemes tördelni a kódot a könnyebb átláthatóság kedvéért:

```
(define (fakt n)
  (if (= n 0)
```

```
1
(let*
  ((i n))
  (while(> i 1)
    (set! i(- i 1))
    (set! n(* n i))
  ) n )
)
```

Ha n egyenlő 0 val akkor 1-et adunk vissza, ellenkező esetben a let* parancsal definiálunk egy lokális változót az i-t és az értéke egyenlő lesz n értékével, ezután a while ciklus addig megy ameddig i nagyobb mint 1, a while cikloson belül minden iterációban i értékét csökkentjük 1-el, majd n értékét beállítjuk ($n * i$)-re, majd a while ciklus végeztével n-t adjuk vissza.

8.2. Gimp Scheme Script-fu: név mandala

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

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

Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/mandala>

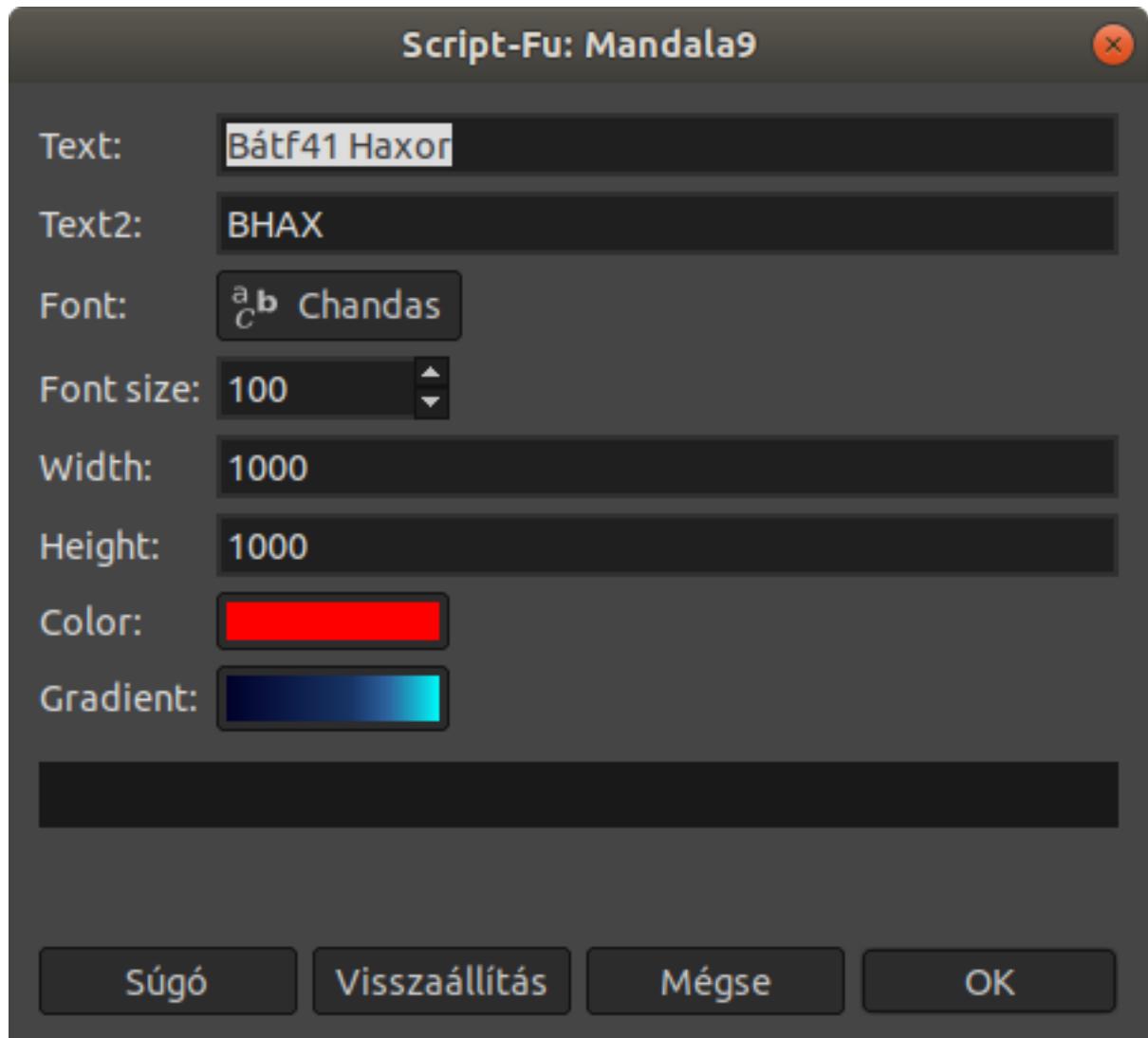
A következő kód egy Gimp szkript lesz, a szcript a programban elvégezhető műveletek automatizálására szolgál.

A kód a gimpnek a menüjébe is beépíthető, ha bemásoljuk a szkriptet a Gimp egyik mappájába, nekem ez linux alatt a következő:

```
~/snap/gimp/165/.config/GIMP/2.10/scripts
```

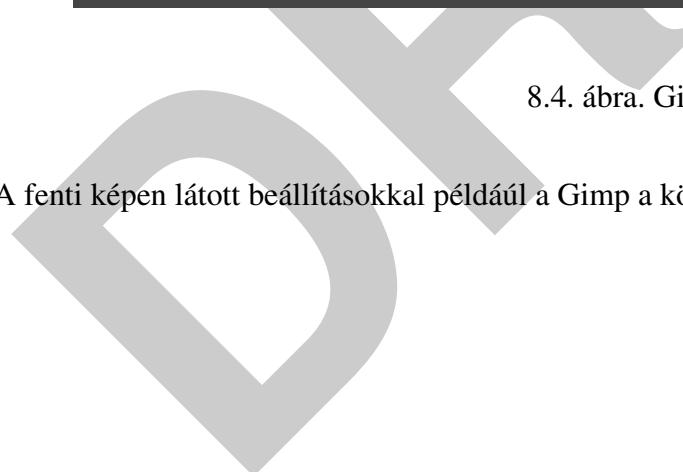
Ezek után a szkript elindítható a Gimp menüjéből a fájl menüpontot kiválasztva, ezután a létrehozás-ra megyünk majd a BHAX-ra kattintva előjön az ablak, ahol megadhatjuk a kívánt paramétereket.

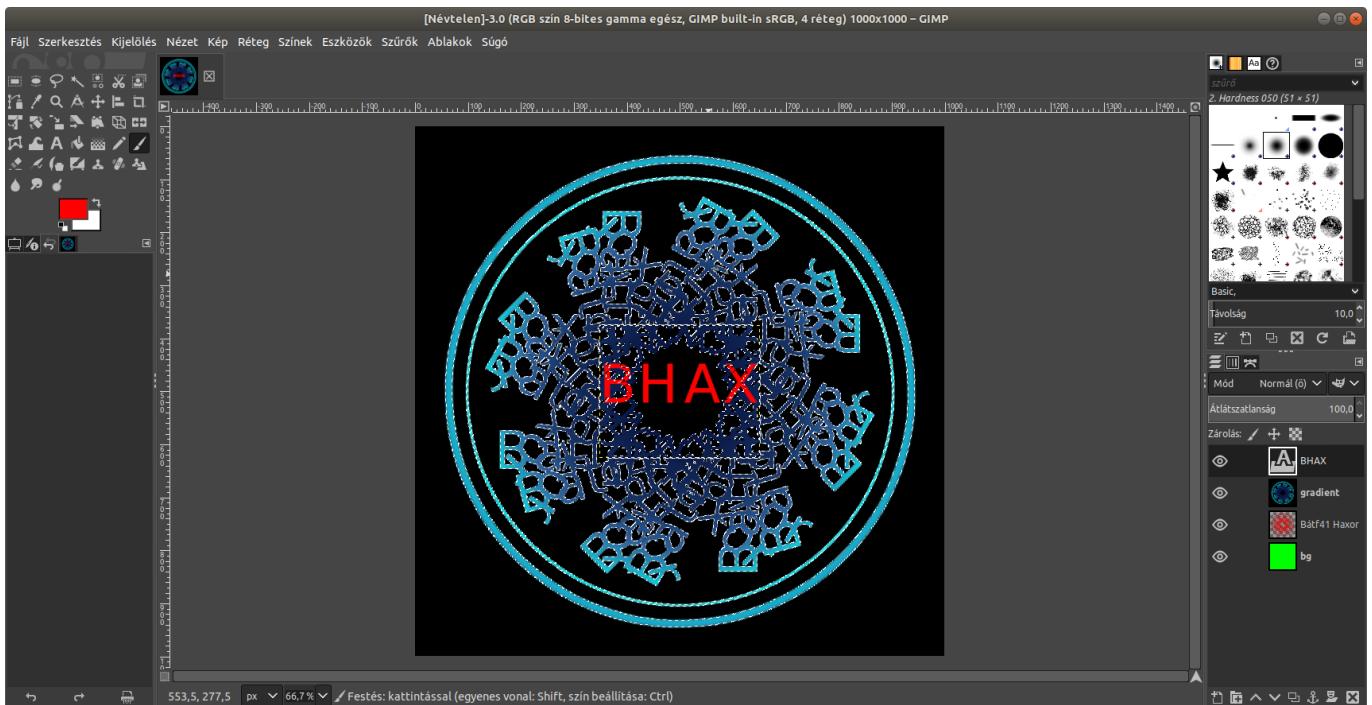
A következő ablakban adhatjuk meg a paramétereket:



8.4. ábra. Gimp szkript

A fenti képen látott beállításokkal például a Gimp a következő képet készíti:





8.5. ábra. Gimp mandala

A kód beilleszthető a Gimpen belül elérhető Script-Fu konzolba, így is elérhető a képkészítés, a kódot egészen a következő sorig kell bemásolni:

```
; (script-fu-bhx-mandala "Bátfa Norbert" "BHAX" "Ruge Boogie" 120 1920 ←
 1080 '(255 0 0) "Shadows 3")
```

Ezután beillesztjük magát a sort, persze a ';' jel kihagyásával hiszen az itt kommentet jelent, a sorban lévő paraméterek változatátásával változtathatunk a készítendő képen, ezután már készíti is a program a képunket.

Nézzük akkor magát a kódot:

```
(define (elem x lista)

  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )

)

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
```

text-width

```
)  
)  
  
(define (text-wh text font fontsize)  
(let*  
  (  
   (text-width 1)  
   (text-height 1)  
  )  
  ;;;  
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←  
    PIXELS font)))  
  ;;; ved ki a lista 2. elemét  
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←  
    fontsize PIXELS font)))  
  ;;  
  
  (list text-width text-height)  
 )  
)  
  
; (text-width "alma" "Sans" 100)  
  
(define (script-fu-bhax-mandala text text2 font fontsize width height color ←  
  gradient)  
(let*  
  (  
   (image (car (gimp-image-new width height 0)))  
   (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←  
     LAYER-MODE-NORMAL-LEGACY)))  
   (textfs)  
   (text-layer)  
   (text-width (text-width text font fontsize))  
  ;;;  
  (text2-width (car (text-wh text2 font fontsize)))  
  (text2-height (elem 2 (text-wh text2 font fontsize)))  
  ;;  
  (textfs-width)  
  (textfs-height)  
  (gradient-layer)  
 )  
  
(gimp-image-insert-layer image layer 0 0)  
  
(gimp-context-set-foreground '(0 255 0))  
(gimp-drawable-fill layer FILL-FOREGROUND)  
(gimp-image-undo-disable image)  
  
(gimp-context-set-foreground color)
```

```
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
  ))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
  height 2))
(gimp-layer-resize-to-image-size textfs)

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
  -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
  -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
  -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
  -LAYER)))

(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car (gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car (gimp-drawable-height textfs)) 100))

(gimp-layer-resize-to-image-size textfs)

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
  (/ textfs-width 2)) 18)
  (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
  textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))
```

```
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
    (/ textfs-width 2)) 18)
    (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
        textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ←
    "gradient" 100 LAYER-MODE-NORMAL-LEGACY)))

(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
    GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ (/ ←
    width 2) (/ textfs-width 2)) 8) (/ height 2))

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
    )))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
    height 2) (/ text2-height 2)))

;(gimp-selection-none image)
;(gimp-image-flatten image)

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-mandala "Bátfa Norbert" "BHAX" "Ruge Boogie" 120 1920 ←
1080 '(255 0 0) "Shadows 3")

(script-fu-register "script-fu-bhax-mandala"
    "Mandala9"
    "Creates a mandala from a text box."
    "Norbert Bátfa"
    "Copyright 2019, Norbert Bátfa"
    "January 9, 2019"
    ""
    SF-STRING      "Text"      "Bátf41 Haxor"
    SF-STRING      "Text2"     "BHAX"
    SF-FONT       "Font"      "Sans"
```

```

SF-ADJUSTMENT      "Font size"   '(100 1 1000 1 10 0 1)
SF-VALUE          "Width"       "1000"
SF-VALUE          "Height"      "1000"
SF-COLOR          "Color"       '(255 0 0)
SF-GRADIENT        "Gradient"    "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)

```

```

(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

```

Definiáljuk az elem nevű függvényt, x és lista lesz a bemenete, a lista lesz az a lista, amire meghívjuk az x pedig, hogy hanyadik elemét szeretnénk visszakapni a listából. Ha az if utáni zárójelés rész igaz vagyis x egyenlő 1-el akkor a (car lista) fog lefutni ami kiveszi a lista első elemét, ellenkező esetben újra az elem függvényt hívjuk, tehát ez egy rekurzív függvény lesz, mert önmagát hívja, egyel kisebb x-et adunk paraméternek, majd a cdr lista-t, a cdr parancs pedig viszaadja az lista összes elemét kivéve az elsőt. Tehát ha nagyobb x-et adunk meg 1-nél, akkor x-et egyel csökkentve meghívjuk az elem függvényt és a listából minden kirakjuk az első elemet, ezt addig ismételjük ameddig x egyenlő nem lesz 1-el, ebben az esetben car-al kiveszük az éppen aktuális lista első elemét, ez az elem annyiadik elem lesz amit x-ként megadunk, tehát ha 3-at adunk meg akkor a lista harmadik elemét kapjuk vissza a rekurzió után.

```

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  text-width
)
)

```

Definiáljuk a text-width nevű függvényt, a let*-el definiálunk lokális változót, ami a let előtt szereplő zárójel bezárásáig van érvényben, itt a text-width nevű változót definiáljuk. A text-width értékét a set!-el állítjuk be, a car függvény kiveszi a neki megadott lista első elemét vagyis itt a text-width értéke, a gimp-text-get-extents-fontname nevű függvény a text, fontsize, PIXELS, font paraméterekkel hívott visszatérési értéke lesz, ez a függvény visszaad négy számot az első a szélesség lesz, ezt fogjuk kivenni és értékül adni a text-width-nek.

```

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )

```

```
)  
;;;  
(set! text-width (car (gimp-text-get-extents-fontname text fontsize ←  
    PIXELS font)))  
;;; ved ki a lista 2. elemét  
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ←  
    fontsize PIXELS font)))  
;;;  
  
(list text-width text-height)  
)  
)
```

A text-wh nevű függvényünk, az előző függvényhez hasonlóan van definiálva, itt viszont nem egy elemet ad vissza a függvény, hanem egy két elemből álló listát, ami a megadott szöveg szélességet és magasságot fogja tartalmazni.

```
(define (script-fu-bhax-mandala text text2 font fontsize width height color ←  
    gradient)
```

Itt definiáljuk a script-fu-bhax-mandala függvényt, az első paraméter a függvény neve a többi az argumentuma. Tehát itt a függvény neve: script-fu-bhax-mandala. Az argumentumai pedig: text, text2, font, fontsize, width, height, color, gradient.

```
(let*  
  (  
      (image (car (gimp-image-new width height 0)))  
      (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←  
          LAYER-MODE-NORMAL-LEGACY)))  
      (textfs)  
      (text-layer)  
      (text-width (text-width text font fontsize))  
    ;;;  
      (text2-width (car (text-wh text2 font fontsize)))  
      (text2-height (elem 2 (text-wh text2 font fontsize)))  
    ;;;  
      (textfs-width)  
      (textfs-height)  
      (gradient-layer)  
  )
```

A let* kulcsszóval tudunk lokális változokat definiálni, amik abban a scope-ban léteznek ameddig a let* előtti zárójel bezárásra nem kerül, itt a függvény összes további része ebben a scope-ban van benne. Például definiáljuk az image nevű változót a car függvény segítségével, a car azt csinálja, hogy kiveszi az utána következő lista első elemét.

```
(gimp-context-set-foreground '(0 255 0))
```

Itt állítjuk be az előtér színét a ' jel azt jelenti a (0 255 0) előtt hogy ez RGB színkód lesz.

```
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
    height 2))
(gimp-layer-resize-to-image-size textfs)

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))
```

Itt tesszük be az alapszöveget majd, ebből a szövegből új layer-eket hozunk létre forgatással, először 180 fokkal forgatjuk el azután ezt tekintjük 1 layernek, majd pi/2 vel elforgatjuk és azután ezt az egészet tekintjük layernek, azután ugyanígy tovább p/4, és pi/6-al.

```
(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
    )))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
    height 2) (/ text2-height 2)))
```

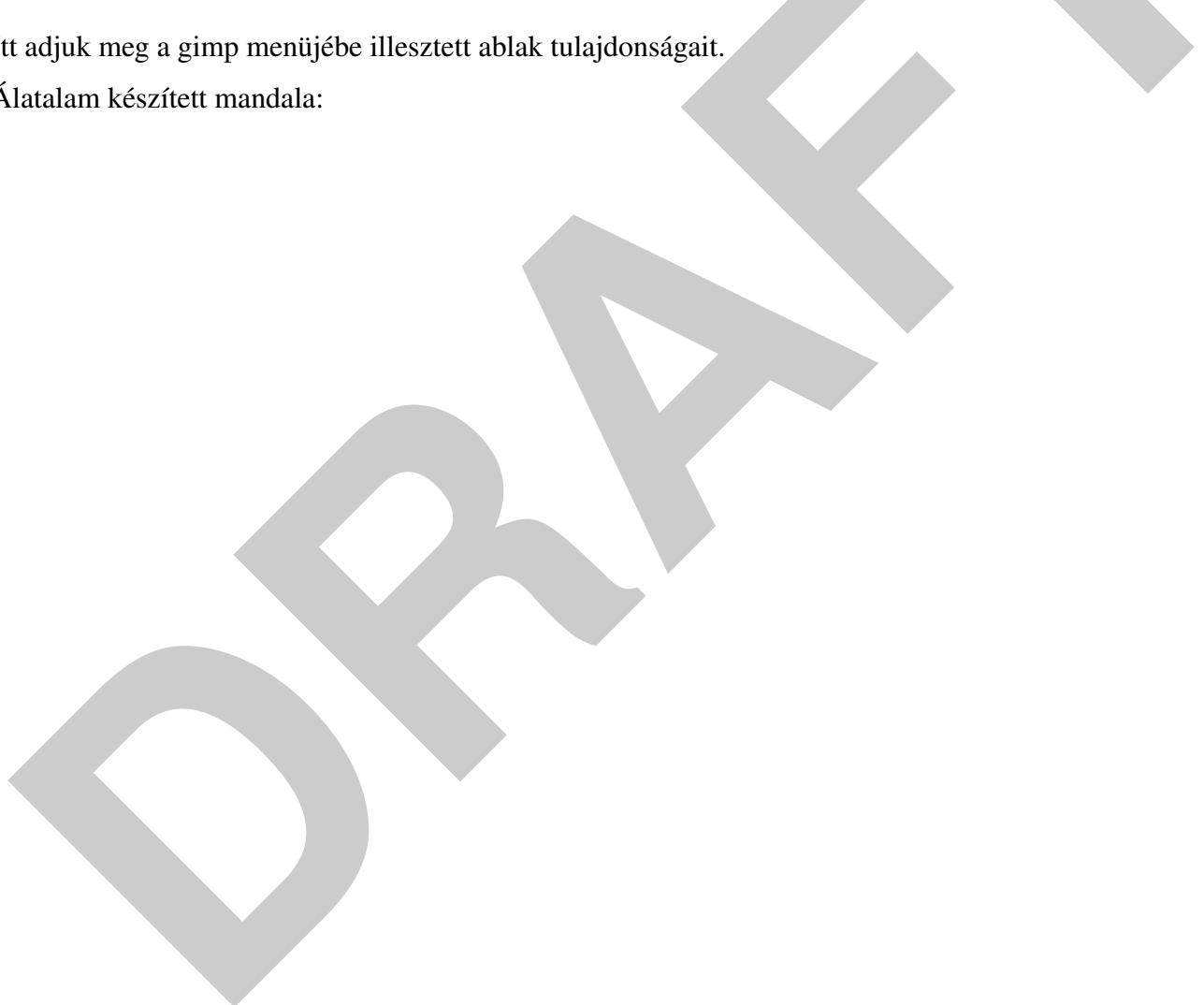
Itt új layer-t hozunk létre a text2-ből, ez a réteg lesz a BHAX felirat a mandala közepén, valamint itt adjuk meg a kép eltolásait, hogy középre kerüljön.

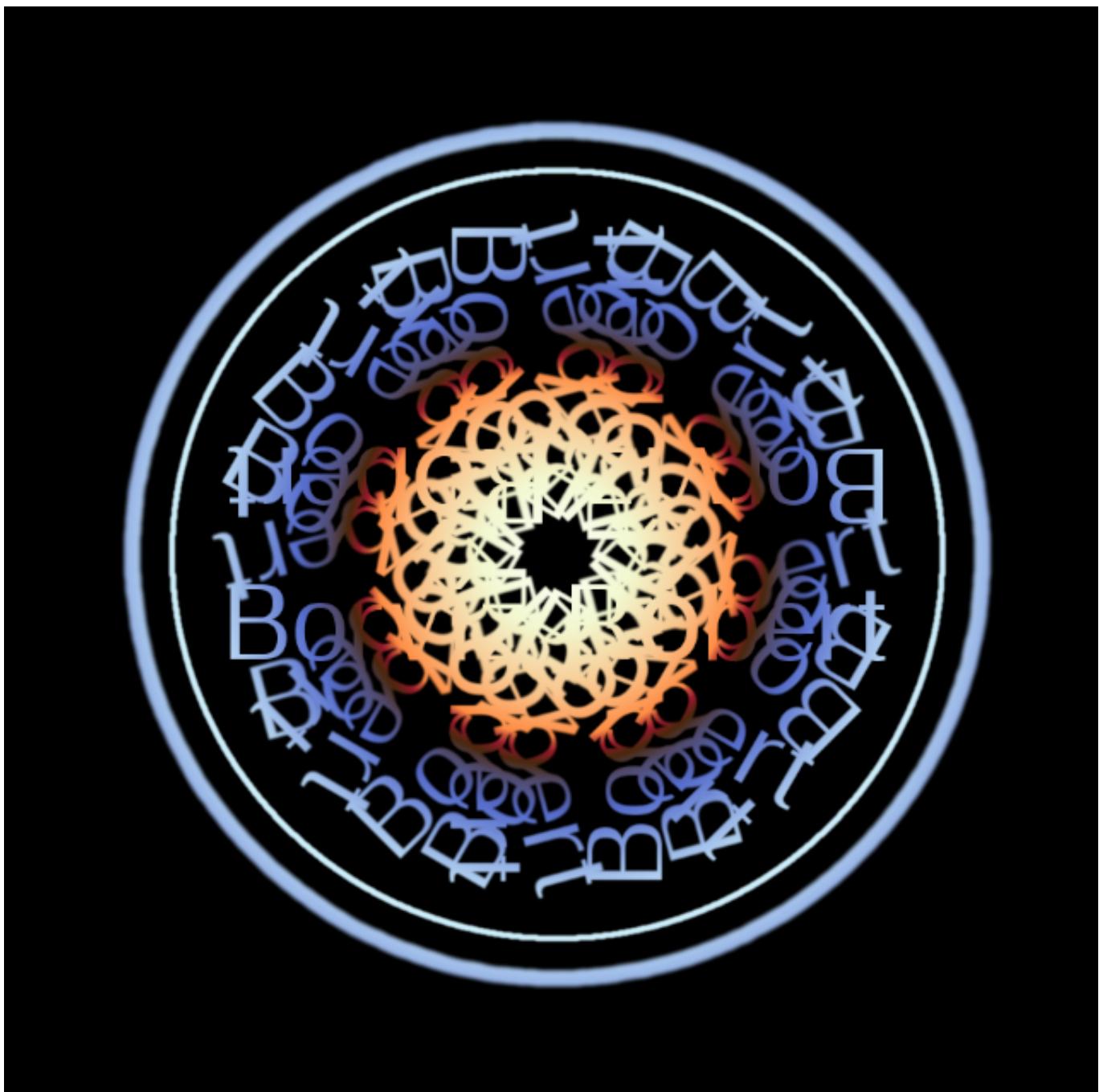
```
(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
```

```
" "
SF-STRING      "Text"        "Bátf41 Haxor"
SF-STRING      "Text2"       "BHAX"
SF-FONT        "Font"        "Sans"
SF-ADJUSTMENT  "Font size"   '(100 1 1000 1 10 0 1)
SF-VALUE        "Width"       "1000"
SF-VALUE        "Height"      "1000"
SF-COLOR        "Color"       '(255 0 0)
SF-GRADIENT    "Gradient"    "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

Itt adjuk meg a gimp menüjébe illesztett ablak tulajdonságait.

Álatalam készített mandala:





8.6. ábra. Mandala

8.3. 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

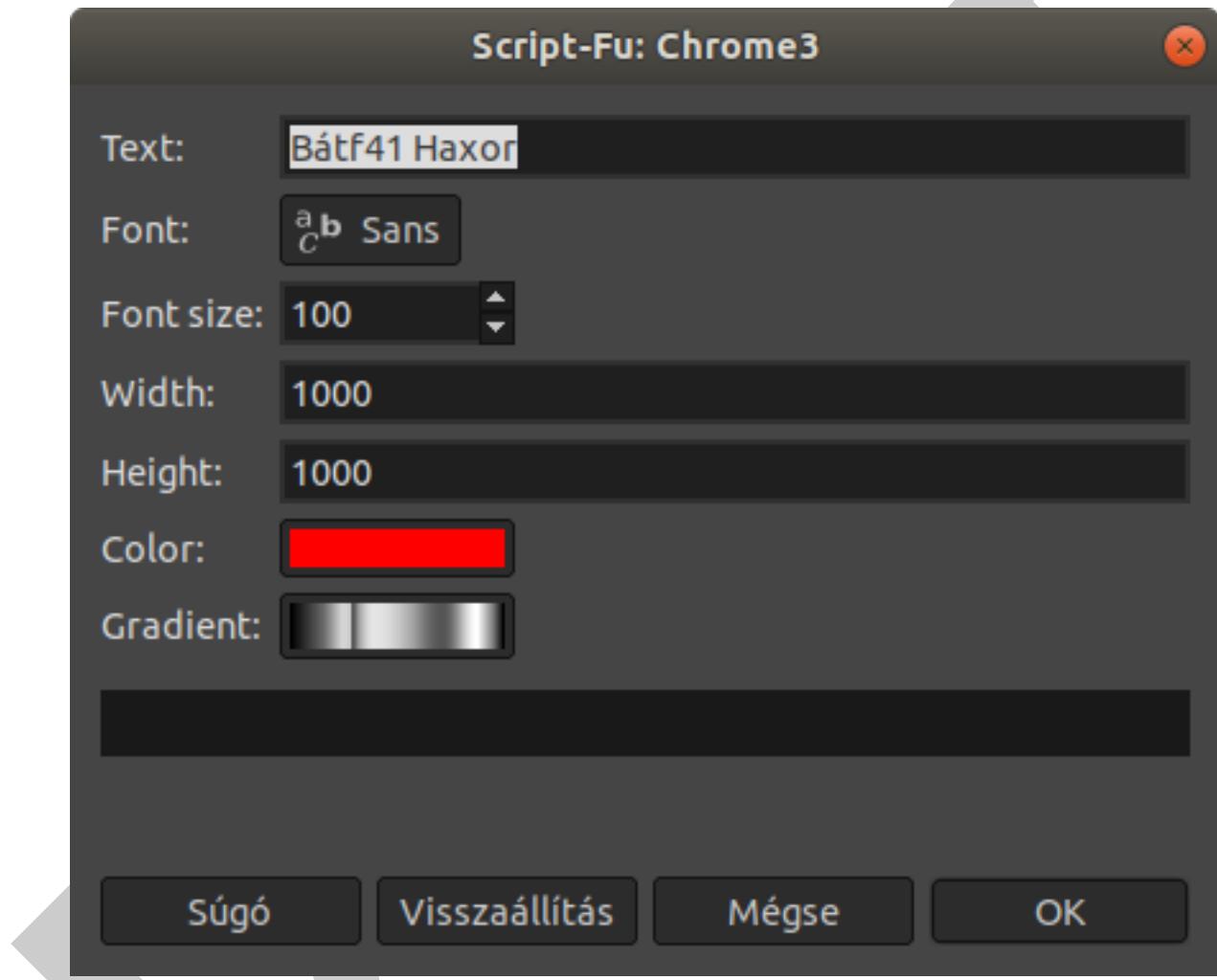
Megoldás forrása: <https://github.com/BorvizRobi/vegyes/blob/master/crome.scm>

A következő szkript, egy bemenő szövegre fog króm effektet megvalósítani. Az előző feladatban leírtakhoz hasonlóan bemásoljuk a következő mappába:

```
~/snap/gimp/165/.config/GIMP/2.10/scripts
```

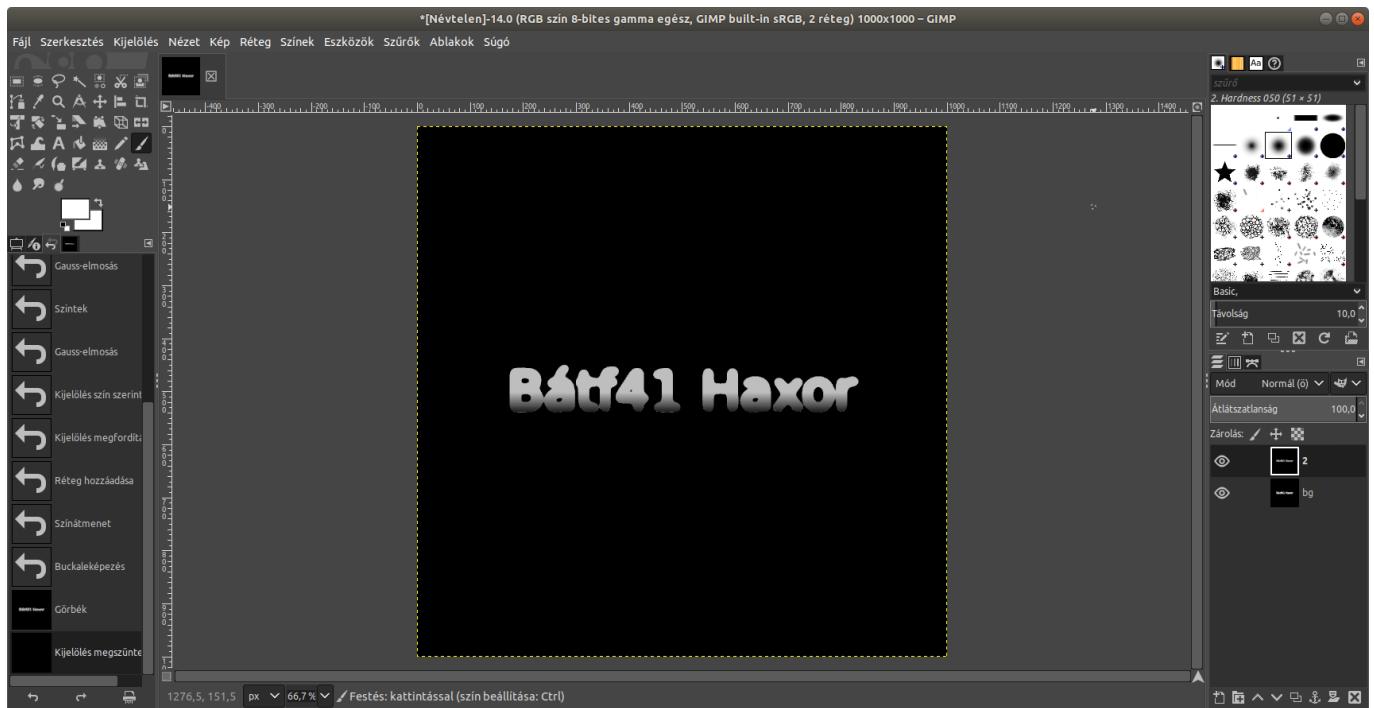
Ezek után a szkript elindítható lesz a Gimp menüjéből, de ha akarjuk a Scrip-Fu konzolból is el tudjuk indítani.

A menüből indítva a következő ablak fogad, ahol beállíthatjuk a paramétereket a létrehozáshoz:



8.7. ábra. Chrome ablak

Az alap paraméterekkel a következő feliratot generálja:



8.8. ábra. Bátfai Haxor felirat

Teljes forráskód:

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)

; (color-curve)

(define (elem x lista)

  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) ) )

(define (text-wh text font fontsize)
(let*
```

```
(  
  (text-width 1)  
  (text-height 1)  
)  
  
(set! text-width (car (gimp-text-get-extents-fontname text fontsize ←  
  PIXELS font)))  
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ←  
  fontsize PIXELS font)))  
  
(list text-width text-height)  
)  
)  
  
;(text-width "alma" "Sans" 100)  
  
(define (script-fu-bhax-chrome text font fontsize width height color ←  
  gradient)  
(let*  
  (  
    (image (car (gimp-image-new width height 0)))  
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←  
      LAYER-MODE-NORMAL-LEGACY)))  
    (textfs)  
    (text-width (car (text-wh text font fontsize)))  
    (text-height (elem 2 (text-wh text font fontsize)))  
    (layer2)  
  )  
  
  ;step 1  
  (gimp-image-insert-layer image layer 0 0)  
  (gimp-context-set-foreground '(0 0 0))  
  (gimp-drawable-fill layer FILL-FOREGROUND)  
  (gimp-context-set-foreground '(255 255 255))  
  
  (set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←  
    ))  
  (gimp-image-insert-layer image textfs 0 0)  
  (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←  
    height 2) (/ text-height 2)))  
  
  (set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←  
    LAYER)))  
  
  ;step 2  
  (plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)  
  
  ;step 3  
  (gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
```

```
;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)

;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
    LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)

;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT-←
    LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
        3)))

;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
    0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 '(255 0 0) "←
    Crown molding")

(script-fu-register "script-fu-bhax-chrome"
    "Chrome3"
    "Creates a chrome effect on a given text."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 19, 2019"
    ""
    SF-STRING      "Text"      "Bátf41 Haxor"
    SF-FONT        "Font"       "Sans"
    SF-ADJUSTMENT  "Font size"  '(100 1 1000 1 10 0 1)
    SF-VALUE       "Width"     "1000"
    SF-VALUE       "Height"    "1000"
    SF-COLOR       "Color"     '(255 0 0)
    SF-GRADIENT   "Gradient"  "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
```

```
"<Image>/File/Create/BHAX"
)
```

Nézzük részenként:

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)
; (color-curve)

(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
  (list text-width text-height)
)
)
```

A korábban definiált függvényeink, működésüket az előző feladatban leírtam, az újdonság itt a color-curve függvény, ami egy tömböt fog visszaadni, a tömb elemei lesznek a paraméterei a gimp-curves-spline eljárásnak a későbbiekben.

```
(define (script-fu-bhax-chrome text font fontsize width height color ←
  gradient)
```

```
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
      LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-width (car (text-wh text font fontsize)))
    (text-height (elem 2 (text-wh text font fontsize)))
    (layer2)
  )
)
```

Ismét ismerős rész következik, itt definiáljuk a függvény nevét, ezután megadjuk az argumentumokat, majd a let* után létrehozzuk a lokális változókat, nemelyiket a car valamint az elem függvényel inicializálunk.

```
;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND )
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
  ))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
  height 2) (/ text-height 2)))

(set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←
  LAYER) ))
```

1. Lépés:

A gimp-image-insert-layer eljárással beszűrünk egy új réteget, majd a gimp-context-set-foreground eljárás-sal beállítjuk a kitöltő színét a következő RGB kódra: 0 0 0 (ez lesz a fekete szín). A gimp-drawable-fill eljárással pedig kitölti a layert a korábban kiválasztott színnel, majd a kitöltőszínt fehérre állítjuk(255 255 255). Ezután a set!-el beállítjuk a textfs nevű változót a car kiveszi az utána lévő lista első elemét ami a gimp-text-layer-new eljárás által létre hozott text layer vagyis a szövegünk. Ezután be is szúrjuk az előbb beállított text layer a képünkbe a gimp-image-insert-layer eljárással. A gimp-layer-set-offsets eljárással pedig középre helyezzük a szöveget. A gimp-image-merge-down eljárással pedig az előbb említett text layerünket és az alatta lévő réteget vagyis a háttért egy layerre füzzük össze.

```
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

2. Lépés:

A plug-in-gauss-iir eljárás alkalmaz egy gauss szűrőt a megadott paraméterekkel, ami a kép elmosódását fogja eredményezni.

```
;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
```

3. Lépés:

A gimp-drawable-levels eljárás a színszinteket fogja beállítani a megadott layeren a megadott paraméterekkel.

```
;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

4. Lépés:

Ismét alkalmazzuk a plug-in-gauss-iir eljárást csak más paraméterrel.

```
;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)
```

5. Lépés:

A gimp-image-select-color eljárással kiválasztjuk a fehér színű pixeleket.

```
;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100
                                     ←
                                     LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)
```

6. Lépés:

Új réteget hozunk létre ami átlátszó lesz, majd ezt beszúrjuk a képünkre.

```
;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT-
                  ←
                  LINEAR 100 0 REPEAT-NONE
                  FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height
                  ←
                  3)))
```

7. Lépés:

Itt először kiválasztjuk a megfelelő gradienst, a gimp-edit-blend eljárással alkalmazzuk a gradienst a megadott paramétereken.

```
;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0
                                         ←
                                         0 TRUE FALSE 2)
```

8. Lépés:

A plug-in-bump-map eljárás a térhatást fogja kialakítani.

```
;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))
```

9. Lépés:

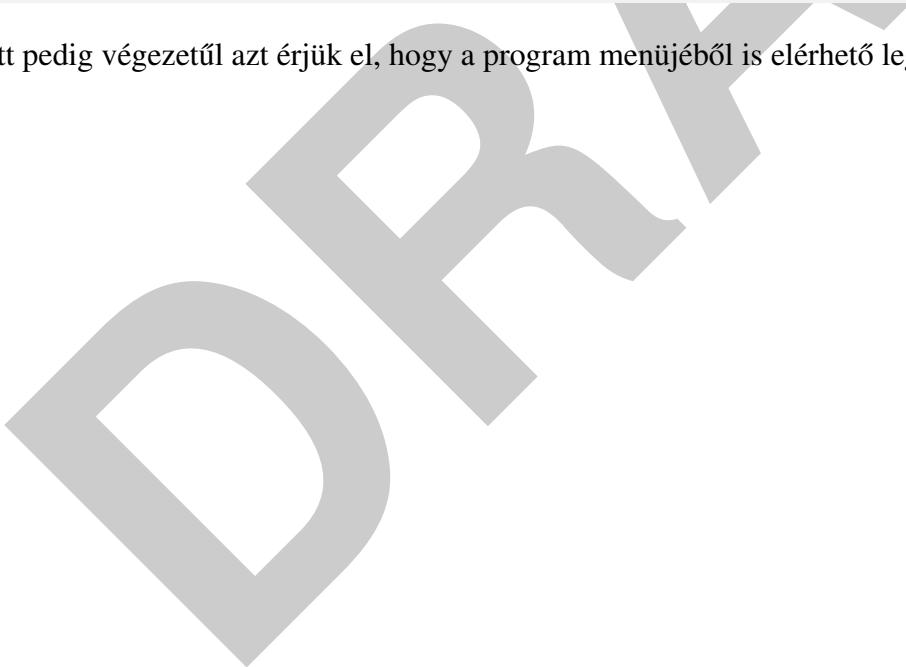
A gimp-curves-spline eljárás a színgörbék fogja beállítani, ezzel fogjuk beállítani a fémes hatást. A szkript elején definiált color-curve függvényünket fogja meghívni.

```
(gimp-display-new image)
(gimp-image-clean-all image)
```

A gimp-display-new eljárás létrehozza a display-t az image-nkből, a gimp-image-clean-all eljárás pedig a képünk dirty count-ját 0-ra állítja.

```
(script-fu-register "script-fu-bhax-chrome"
  "Chrome3"
  "Creates a chrome effect on a given text."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 19, 2019"
  ""
  SF-STRING      "Text"        "Bátf41 Haxor"
  SF-FONT        "Font"        "Sans"
  SF-ADJUSTMENT   "Font size"   '(100 1 1000 1 10 0 1)
  SF-VALUE        "Width"       "1000"
  SF-VALUE        "Height"      "1000"
  SF-COLOR        "Color"       '(255 0 0)
  SF-GRADIENT     "Gradient"    "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```

Itt pedig végezetű azt érjük el, hogy a program menüjéből is elérhető legyen a szkript.



9. fejezet

Helló, Gutenberg!

9.1. Juhász István: Magas szintű programozási nyelvek 1

11.-15. oldal:

Ez a rész az alapfogalmakról szólt. A programozásra kialakult nyelveknek három szintjét különböztetjük meg: gépi nyelv, assembly nyelv, magas szintű nyelv. A magas szintű nyelven íródott programot forrásprogramnak nevezzük. Szintaktikai szabályoknak nevezzük a forrásszöveg összeállítására vonatkozó formai szabályokat. A tartalmi, jelentésbeli szabályok pedig a szemantikai szabályok. Egy magas szintű programozási nyelvet ez a két szabály határozza meg. A programozási nyelvek osztályozása: Imperatív nyelvek, Deklaratív nyelvek, máselvű nyelvek.

17.-27. oldal:

Ebben a részben az alapelemeit ismertem meg egy programozási nyelvnek. minden program forrásszövegének a legkisebb alkotóelemei a karakterek. minden nyelv definiálja a saját karakterkészletét, ezek általában a következők: betűk, számjegyek, egyébb karakterek.

A lexikális egységek a programszöveg azon elemei, amiket a fordító a lexikális elemzés során felismer és tokenizál. A programszövegen elhelyezhető megjegyzés, ez nem a fordítóprogramnak szól hanem a programozónak. A literálok vagy más néven konstansok olyan programozási eszközök, amelyek segítségével fix értékek építhetők be a programba.

28. oldal:

Adattípusok, egy adattípust 3 dolgot határoz meg: tartomány, műveletek, reprezentáció. minden adattípus mögött van egy megfelelő belső ábrázolási mód. A legtöbb nyelvben a programozó is definiálhat típusokat.

34. oldal:

A nevesített konstans olyan programozási eszköz, amelynek három komponense van: név, típus, érték. Mindig deklarálni kell. A program szövegében a névvel szerepel, de ez a név az értéket fogja jelenti.

35. oldal:

A változó olyan programozási eszköz, amelynek 4 komponense van: név, attribútumok, cím, érték. A változó a program szövegében a névvel jelenik meg, az viszont minden komponenset jelölheti. Az attributumok olyan jellemzők, amelyek a változó futás közbeni viselkedését határozzák meg.

46.-55. oldal:

Kifejezések: szintaktikai eszközök, arra valók hogy a program egy pontján, ott már ismert értékekből új értéket határozzunk meg. Formálisan a következő összetevőkből áll: operandusok, operátorok, kerek zárójelek. Lehetséges kifejezés alakok: prefix, infix, postfix.

Konstans kifejezés: az értéke a fordítási időben eldől, később nem változhat.

Operátortípusok: szorzás, osztás, maradékképzés, összeadás, kivonás operátora.

56.-71. oldal:

Az utasítások alkotják az eljárásorientált nyelvek olyan egységeit, amelyeket az algoritmusok egyes lépései megadjuk, másrészt a fordítóprogram ennek segítségével generálja a tárgyprogramot. Két nagy csoportjuk van: deklarációs és végrehajtható utasítások.

Az értékadó utasításnak az a feladata, hogy beállítsa vagy módosítsa egy változó értékrészét. Az eljárásorientált programnyelvek legtöbbször tartalmaznak üres utasítást is, hatására a processzor, egy üres gépi utasítást hajt végre. A ciklusszervező utasítások lehetővé teszik, hogy a program egy pontján bizonyos tevékenységet többször megismételjünk. A ciklus felépítése: fej, mag, vég.

72.-78. oldal:

Az eljárásorientált nyelvekben a szöveg szuverén részekre úgynevezett programegységekre bontható.

Az alprogram az eljárásorientált nyelvekben a procedurális absztrakció első megjelenési formája. Az alprogram az újrafelhasználás eszköze. Akkor alkalmazható ha a program különböző pontjain ugyanaz a programrész megismétlődik. Az alprogram környezete alatt a globális változók együttesét értjük.

Hívási lánc, rekurzió: Egy programegység meghívhat egy másik programegységet, az egy újabb programegységet és így tovább. Így kialakul egy hívási lánc. A hívási lánc első tagja mindig a főprogram. A hívási lánc minden tagja aktív, de csak a legutoljára meghívott programegység működik. Szabályos esetben minden a legutolsó meghívott programegység fejezi be a legelőször a működését, és a vezérlés visszatér az őt meghívó programegységbe. A hívási lánc futás közben dinamikusan épül fel és bomlik le.

A rekurzió kétféle lehet: közvetlen és közvetett. A rekurzióval megvalósított algoritmus minden átirható iteratív algoritmussá. Az iteratív algoritmusok általában gyorsabbak, mivel kevesebb memória foglalással járnak.

78.-82. oldal:

Paraméterkiértékelés alatt azt a folyamatot értjük, amikor egy alprogram hívásánál egymáshoz rendelődnek a formális- és aktuális paraméterek, és meghatározódnak azok az információk, amelyek a paraméterátadásnál a kommunikációt szolgáltatják.

Paraméterátadás az alprogramok és más programegységek közötti kommunikáció egy formája. A paraméterátadásnál minden van egy hívó, ez tetszőleges programegység, és egy hívott, amelyik minden alprogram, Kérdés, hogy melyik irányban és milyen információ mozog. A nyelvek a következő paraméterátadási módot ismerik: érték szerint, cím szerint, eredmény szerint, érték-eredmény szerint, név szerinti, szöveg szerinti.

82.-85. oldal:

A blokk olyan programegység, ami egy másik programegység belsejében helyezkedik el, külső szinten nem állhat. A blokknak van kezdete törzse és vége. A hatáskör a nevekhez kapcsolódó fogalom. Egy név hatásköre alatt értjük a program szövegének azon részét, ahol az adott név ugyanazt a programozási eszközt hivatkozza.

112.-113. oldal:

A kivételkezelési eszközrendszer azt teszi lehetővé, hogy az operációs rendszertől átvegyük a megszakítások kezelését, felhuzzuk azt a program szintjére, ezek olyan események amelyek megszakításokat okoznak. A kivételkezelés egy olyan tevékenység, amelyet a program végez, ha egy egy kivétel következik be. A kivételkezelő egy olyan programrészt jelent ami akkor lép működésbe, amikor egy adott kivétel bekövetkezik, így reagálva az eseményre.

A kivételkezelési eszközrendszerrel kapcsolatban a nyelveknek a következő kérdéseket kell megválaszolni:

- Milyen beépített kivételek vannak a nyelvben?
- Definiálhat-e a programozó saját kivételt?
- Milyenek a kivételkezelő hatásköri szabályai?
- A kivételkezelés köthető-e programelemekhez?
- Hogyan folytatódik a program a kivételkezelés után?
- Mi történik, ha kivételkezelőben következik be kivétel?
- Van-e a nyelvben beépített kivételkezelő?
- Van-e lehetőség arra, hogy bármely kivételt kezelő kivételt kezelőt írunk?
- Lehet-e parametrizálni a kivételkezelőt?

134.-138. oldal:

Az I/O az a területe a programnyelveknek, ahol azok leginkább eltérnek egymástól. Az I/O platform operációs rendszer valamint implementációfüggő. Az I/O az az eszközrendszer a programozási nyelvben ami a perifériákkal történő komunikációért felelős. Az I/O középpontjában az állomány áll. Egy állomány a funkciója szerint lehet: input állomány, output állomány , input-output állomány.

9.2. Benedek Zoltán: Szoftverfejlesztés C++ nyelven

1.-16. oldal:

Fügvényparaméterek és visszatérési érték, ha C++-ban egy függvényt üres paraméterlistával definiálunk, az azt jelenti hogy nincsen paramétere, ha viszont három pontot adunk meg paraméternek akkor tetszőleges számú paraméterrel hívható. C++ nyelvben vezették a bool típus, ami a logikai igaz hamis értéket reprezentálja. C++ ban minden olyen helyen állhat változódékláció ahol utasítás állhat.

Egy függvényt a C nyelvben a neve azonosít egyértelműen, így egy adott hatókörben két azonos nevű függvény nem deklarálható. A C++ nyelvben a függvényeket a nevük és az argumentumlistájuk együttesen azonosítja. Így lehetőség van azonos nevű függvények létrehozására.

A C++ nyelvben lehetőség van arra hogy a függvények argumentumainak alapértelmezett értéket adjunk meg. Amikor ezen argumentumoknak a függvény hívásakor nem adunk meg értéket akkor, a függvény az alapértelmezett értékekkel kerül meghívásra.

17.-59. oldal:

Az objektumorientált szemlélet alapgondolatai, egysége zárás, egysége záró adatstruktúra neve az osztály, ami egyfajta kategóriát definiál. Az osztálynak lehetnek példányai, amik mint önálló egyed jelennek meg, ezeket objektumoknak nevezzük.

C-ben a dinamikus memóriakezelést a malloc és a free függvényekkel, valamint ezek variánsaival végezhetjük. A paraméterátadás miatt C++-ban már nem is függvény hanem operátor felelős a dinamikus memóriakezelésért, ennek a neve a new, a használat után a lefoglalt helyet a delete operátorral szabadítjuk fel.

73.-90. oldal:

A C++ adatfolyamokban gondolkodik(stream), amik tulajdonképpen bájtok sorozatát jentik. A C++ I/O használatához be kell építenünk az iostream állományt.

Az adatfolyam objektumoknak természetesen vannak tagfüggvényeik, amelyekkel beállíthatjuk az állapotát, adatokat olvashatunk, írhatunk, valamint egyébb műveleteket olvashatunk az adatfolyamon, de az adatfolyamok manipulálásának nem ez az egyetlen módja: használhatunk úgynevezett manipulátorokat is.

A C szabványos állománykezelése egy FILE* típusú leíró köré csoportosul, amelyet a fopen függvény ad vissza. Amikor egy megnyitott állományon valamilyen műveletet akarunk végezni, akkor át kell adnunk a leírót, hogy az adott függvény tudja hogy melyik állományba kell elvégezni azt a bizonyos műveletet. A C++ megoldásban a leírót egy objektum zárja egysége, és annak tagfüggvényei segítségével végezhetjük el az állományműveleteket. A C++ az állománykezeléshez is adatfolyamokat használ, amelyeket az ifstream(input file stream), valamint az ofstream(output file stream) osztályok reprezentálnak. Az állományok megnyitását a konstruktorok, lezárásukat pedig a destruktörök végzik, így sokkal biztonságosabb az állománykezelés.

187.-197. oldal:

A program futása során a hibás esetek kezelése C-ben függvények által visszaadott hibakódok vagy globális változóban az utolsó hibára vonatkozóan nyilvántartott hibakód alapján történik. C++-ban a kivételek(exceptions) alkalmazásával ennél sokkal átláthatóbb és könnyebben karbantartható hibakezelési lehetőség áll rendelkezésünkre.

A kivételkezelés olyan mechanizmus, amely biztosítja, hogy ha hibát észlelünk valahol, akkor a futás azonnal a hibakezelő ágon folytatódjon. Ez a megoldás nemcsak hiba hanem bármilyen kivételes helyzet esetén használható.

A kivételkezelés néhány fontos aspektusa bemutatása a következő példán keresztül:

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

int main()
{
    try
    {
        double d;
        cout << "Enter a nonzero number: ";
        cin >> d;
        if (d==0)
            throw "The number can not be zero.";
        cout << "The reciprocal is: " << 1/d << endl;
    }
}
```

```
    catch (const char* exc)
    {
        cout<< " Error! The error text is: " << exc << endl;
    }
    cout<< "Done."<<endl;
}
```

Kimenet ha a felhasználó nem nullát ad meg:

```
Enter a nonzero number: 2
The reciprocal is: 0.5
Done.
```

Kimenet ha a felhasználó nullát ad meg:

```
Enter a nonzero number: 0
Error! The error text is: The number can not be zero.
Done.
```

A kódban a main-ben egy try-catch blokkot találunk, egy catch ággal. A try védett blokkba kapcsoszárójelek közé írtuk a normál működésnek a kódját, a catch ágba kapcsoszárójelek közé a hibakezelő kódot.

A verem visszacsévélése:

Egy kivétel dobásakor annak elkapásáig a függvények hívási láncában haladva az egyes függvények lokális változói felszabadulnak. Ezt a folyamatot a hívási verem viszacsévélésének nevezzük. Nézzük a következő kódot:

```
int main()
{
    try
    {
f1();
    }
    catch(const char* errorText)
    {
cerr<<errorText<<endl;
    }
}

void f1()
{
    Fifo fifo; // Tegyük fel hogy fifo egy általunk megírt osztály
f2();
...
}

void f2()
{
    int i=1;
    throw "error1";
}
```

A lépések a példában:

1. Az f2 kivételt dob.
2. Az f2-ben definiált i lokális változó felszabadul.
3. Az f1-ben lefoglalt Fifo fifo objektum felszabadul, meghívódik a destruktora.
4. Lefut a main függvényben levő catch blokk.

Bár nem gondolnánk, a kivétel dobása és elkapása között kód futhat le, ugyanis meghívódnak a verem visszacsévélése során felszabadított objektumok destruktoraival. Lényeges szabály, hogy a kivétel dobása és elkapása között ne dobjunk újabb kivételt, mert ennek kezelése már nem lesz lehetséges, meghívódik a terminate, és az alkalmazás futása befejeződik.

Destruktorból soha ne dobjunk kivételt, illetve ha olyan kódot hívunk, amely kivételt dobhat, akkor a kivételeket a destruktordan alkalmazott try-catch blokkal kezeljük!

9.3. Brian W. Kernighan, Dennis M. Ritchie: The C programming language

Vezérlési szerkezetek: c.fejezet:

Vezérlési szerkezetek: Utasítások és blokkok, if-else utasítás, switch, while és for utasítás, do-while utasítás, for utasítás, break, continue.

A C nyelv egy makro előfeldolgozó segítségével bizonyos nyelvi kiterjesztéseket nyújt számunkra, ezek közül a legközönségesebb a #define, de ide tartozik még az a nyelvi eszköz, amely állományok tartalmának a fordítás során történő beiktatását teszi lehetővé.

A #define szimbólumok, deklarációk és más nyelvi objektumok kezelését is megkönnyíti a C állománybeiktatási szolgáltatása. minden sor amelynek alakja: #include "állománynév" az állománynév nevű állomány tartalmával helyettesítődik, itt az idézőjel kötelező. Legtöbbször minden állomány megjelenik ilyen alakú sor, ami a közös #define utasításoknak és a globális változók extern _deklarációinak beiktatására szolgál. Az #include parancsok egymásba skatulyázhatók.

A **#define YES 1** alakú definícióval a legegyszerűbb típusú makrohelyettesítést valósítjuk meg, egy nevet egy karakterláncnal helyettesítünk. A #define-ban előforduló nevek alakja azonos a C azonosítóival, a helyettesítő szöveg tetszőleges. A helyettesítő szöveg általában a sor további része, hosszú definíciók úgy folytathatóak, hogy a folytatni kívánt sor végére \t helyezünk. A #define szimbólummal definiált név érvényességi tartománya a definíció helyétől az adott forrásállomány végéig tart.

```
printf ("YES")
```

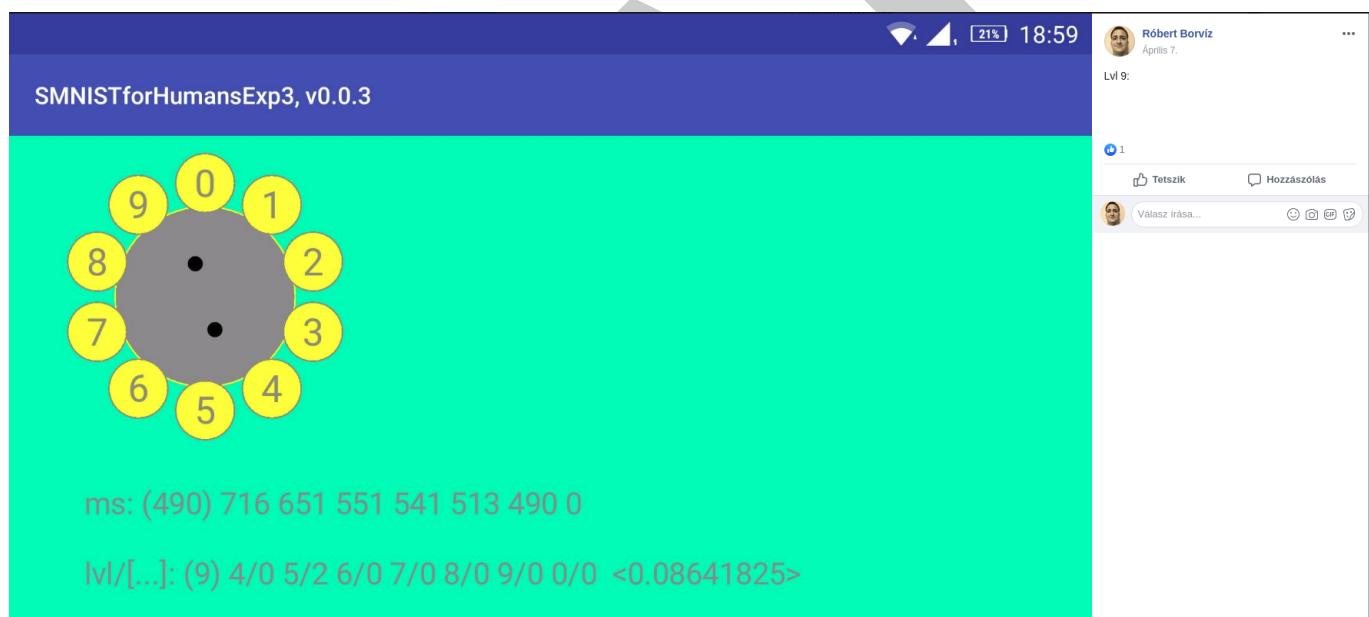
Amikor a YES egy definiált név, akkor a printf-ben nem történik helyettesítés.

10. fejezet

Összegzés:

10.1. Passzolt feladatok:

Összesen 7db passzolt feladat, de ebből 2 db kiváltva a következő feladat teljesítése miatt:



10.1. ábra. +2 passz

10.2. Tutoriáltaim:

Tóth Csaba Decimálisból unárisba átváltó Turing gép:https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeaTjhhiMRoPXFF9fTnwkq5UWkNZsV6RkUem5RMd_fWW8Uk9M1QM

Tóth Csaba C EXOR törő:https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeaTjhhiMRoPXFF9fTnwkq5UWkNZsV6RkUem5RMd_fWW8Uk9M1QM

Szabó Benedek WELCH Mozgató szemantika: https://gitlab.com/benedekszabo/bhax?fbclid=IwAR1A8OPNSYIYFzNXbx_vgNZatYny6kHrDXx_cly2lBBygBIRJAE4WlZvLu8

10.3. Tutorok:

Tóth Csaba Hangyaszimulációk: <https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeATfWW8Uk9M1QM>

Racs Tamás Mozgató szemantika: <https://gitlab.com/cant0r/bhax?fbclid=IwAR1xwKuz0cDUx>

Racs Tamás Hangyaszimulációk: <https://gitlab.com/cant0r/bhax?fbclid=IwAR1xwKuz0cDUx>

Tóth Csaba A Mandelbrot halmaz: <https://gitlab.com/tocsika7/bhax?fbclid=IwAR1kJr4KeATfWW8Uk9M1QM>

DRAFT

II. rész

Második felvonás

DRAFT

11. fejezet

Helló, Berners-Lee!

11.1. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.

Java és C++ összehasonlítás:

A Java nyelv és környezet tervezői a C és C++ nyelvek szintaxisát vették alapul. Rengeteg C++ utasítás, kifejezés szintaktikailag helyes Javaban is, és sokszor hasonló jelentéssel is bír. A hasonlóság azonban nem azonosság, előfordulhat hogy a hasonló nyelvi elemek eltérő jelentése meglepetést okoz. A Java mint nyelv szűkebb a C++-nál, viszont szabványos osztálykönyvtárai révén szélesebb alkalmazási területet fed le, például: nyelvi szinten támogatja a thread-eket, a grafikus felhasználói felület programozását, a hálózati programozást, a perzisztenciát, adatbázisok tartalmának elérését és így tovább. Ezeket a fogalmakat persze C++-ban is lehet kezelní, de a C++ nyelv illetve a szabványos könyvtár közvetlenül nem terjed ki ezekre a részekre, ekkor valamely egyedi külső könyvtár segítségét kell igénybe vegyük.

A C++ lehetővé teszi forrásszinten hordozható programok írását. A szabványos módon megírt forrást a célgépen újrafordítva és szerkesztve helyes, futó programot kapunk. Az összeszerkesztett, futtatható bináris kód azonban nem, vagy nagyon ritka esetekben hordozható. A lefordított kód ugyanis tartalmaz a helyi operációs rendszerre, hardverre vonatkozó speciális feltételezéseket. A Java célkitűzései között szerepel a platformok közötti bináris hordozhatóság. A bájtkodá lefordított programot átvihetjük más gépekre, ahol szabványos Java virtuális gép környezetet találunk. Ilyen előfeltételezések alapján joggal számíthatunk arra, hogy a Java sokkal szigorúbb előírásokat tesz az egyes nyelvi elemekre: típusok méretére, belső szerkezetére, kifejezések kiértékelésére, kivételek kiváltásának ellenőrzésére.

Javában minden függvény virtuális, extenzív futási idejű mechanizmusok léteznek az egyes osztályok, objektumok, metódusok azonosítására, akár létrehozására is. Javában a statikus változók inicializálása is futási időben történik. Azért hogy a hordozható kód viselkedése kevésbé függjön a platformtól és az implementációtól, a Java kevesebb dolgot bíz az implementációra, mint a C és C++. Azért hogy a kód viselkedése a lehető legfontosabb definiált legyen, a Java fordító ellenőriz néhány olyan dolgot is, amit a C és C++ fordító nem jelez hibának. Például ellenörzi, hogy használatuk előtt a lokális változók kapnak-e értéket, velük szemben a nem-lokális, azaz tag vagy statikus változók a típusnak megfelelő alapértelmezett értéket kapnak ha nem adunk nekik explicit értéket.

Alapvető különbségek adódnak a két nyelv között az objektummodellek különbözőségéből. A C++ az objektumokat, mint a memória egy összefüggő területén elhelyezkedő bájtsorozatot fogja fel, ami ismert memóriakiosztással rendelkezik, és amit ennek megfelelően manipulál a lefordított program, ezért például ha egy bázisosztály megváltozik akkor újra kell fordítani az öröklő osztályokat is. A C++-ban mutatók segítségével közvetlenül manipulálhatjuk a memóriát. A Java programok virtuális gépben futnak a memóriát közvetlenül nem tudjuk elérni, hanem csak szimbolikusan, hivatkozásokon keresztül. A Java esetén nincsen linker, amelyik címekké oldaná fel a hivatkozást. Egy osztály egy önálló class-fájllá fordul le. A class-fájl formátuma szabványos, platform-független. A Java virtuális gép betöltéskor ellenörzi a class-fájlokat. A külső hivatkozások név szerint kerülnek a class-fájlba. A Java ismeri egy osztály kompatibilis megváltoztatásának fogalmát. Például ha a hivatkozott osztályban megváltozik a tagok vagy metódusok deklarációs sorrendje, vagy új tagok lépnek be, akkor a hivatkozó osztály újrafordítás nélkül is érvényes marad. A Java nyelv megkülönböztet primitív típusokat és objektumokat, ez utóbbiakat hivatkozásokon keresztül érjük el. Primitív adattípusok és objektumhivatkozások lehetnek automatikus vagy statikus adatterületen, de maguk az objektumok nem! A Java virtuális gép az objektumokat egy automatikus szemétfelügyelet mechanizmus által felügyelt véletlen elérésű tárterületen tárolja. Ennek megfelelően minden metódus paramétere és visszatérési értéke is csak primitív típusú vagy objektumhivatkozás lehet, maga az objektum nem. Ezért hívási paraméter vagy visszatérési érték átadáskor vagy kivétel elkapáskor nem lépnek fel a C++-ban ismert másolási és csonkolási jelenségek. Mivel csak az objektum-hivatkozások lehetnek automatikus (helyi) változók, a C++-ból ismert automatikus destrukturáló mechanizmus nincsen Java-ban. A dinamikus tárban lefoglalt memória programozói felszabadítása közvetlenül nem lehetséges: nincsen delete és delete[]. A Java nyelvből hiányzik a destrukturáló mechanizmus, ezt a finalize metódus csak úgy tudja végezni. Javaban nincsen érvénytelen hivatkozás, amíg referenciaink van egy objektumra, addig az az objektum garantáltan létezik.

A main methódus paramétere egyetlen String tömb argumentum. Mivel a Java tömbök tudják a méretüket, szükségtelen a C++ argc paraméter átadása. A C,C++-al ellentétben a Java-ban a main első (0 indexű) argumentuma nem tartalmazza a program nevét, ha létezik 0 indexű paraméter az az első parancssori argumentumot tartalmazza.

A Java-ban a C++-ban megszokott /* blokk-kommenten */ és a // sorkommenten kívül a blokk-komment speciális eseteként létezik a /** jelekkel elindított úgynevezett dokumentációs komment melyekből a javadoc segédprogram segítségével HTML-formátumú dokumentációt generálhatunk.

A Java nyelvben nincsenek külön objektumok és mutatók. Az objektumok a dinamikus tárterületen jönnek létre, és csak hivatkozásokon keresztül érjük el őket, külön mutató vagy referencia szintaxis nélkül. A Java-ban nincsen lehetőségünk felhasználói operátorok definiálására, be kell érni a gyáriakkal. A == összehasonlítás Javában csak az azonos objektumokra hivatkozó referenciakat hasonlítja össze.

A Java nyelvi szinten megkülönböztet osztályokat valamint interfészket. Az osztályok körében a Java egyszeres öröklődést támogat, és ha egy osztálynak nincsen a forrásban megadott szülőosztálya, akkor implicit módon az Object osztály lesz az. Az öröklődést az extends kulcsszó jelzi, és a C++-től eltérően nincsen megkülönböztetés privát, protected és publikus öröklődés között. Ezenkívül létezik az abstract kulcsszó is. Ha egy osztályt abstract-nak deklarálunk, akkor egyszer nem példányosítható, másrészt lehetnek benne tisztán virtuális, azaz csak deklarált, de nem implementált metódusok. Ezek az abstract minősítőről ismerszenek meg, és arról, hogy a testük helyett egy ; áll csak. Csak nem absztrakt osztály példányosítható.

Hatókör operátor (scope resolution operator, ::) nincsen Java-ban. Egy osztály metódusából egy szülőosztálybeli tag szükség esetén a super kulcsszó használatával érhető el, akkor is ha túl van terhelve vagy el van fedve.

11.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)

A Python általános célú programozási nyelv, a fejlesztők számára rengeteg pozitív tulajdonsággal rendelkezik. A Python magas szintű, objektum orientált dinamikus és platform független. A fejlesztés fázisai közül leginkább prototípus készítésre, tesztelésére érdemes alkalmazni, de egyszerűbb alkalmazások is hatékonyan készíthetőek vele. Más nyelveken megírt modulokkal is együtt tud működni egy Python komponens.

A Python tulajdonképpen egy szkriptnyelv, de nagyon sok csomagot is és beépített eljárást is tartalmaz, ezért komolyabb alkalmazások megírására és komolyabb problémák megoldására is használható. Magas szintű típusokat is támogat listák, szótárak.

Python esetén nincs szükség fordítás-ra, az értelmezőnek elég a Python forrást megadni, és az futtaja is az alkalmazást. A Python interpreter elérhető számos platformon. A Python egyszerű használni, megbízható és jelentős támogatást biztosít hibák javítására. A Python egy nagyon magas szintű programozási nyelv.

A Python nyelv legfőbb jellemzője, hogy behúzásalapú a szintaxisa. A programban szereplő állításokat az azonos szintű behúzásokkal tudjuk csoportokba szervezni, nincs szükség kapcsos zárójelekre, vagy explicit kulcsszavakra. Egy adott blokk végét egy kisebb behúzású sor jelzi, ezért üres sor lehet a blokkon belül. A szkript első utasítása nem lehet behúzott. A nyelv másik tulajdonsága hogy a sor végéig tart egy utasítás, nincs szükség a ';' használatára.

Pythonban minden adatot objektumok reprezentálnak. Az adatokon végezhető műveleteket az objektum típusa határozza meg. Pythonban nincs szükség a változók típusainak explicit megadására, a rendszer futási időben automatikusan kitalálja a változók típusát a hozzárendelt érték alapján.

Pythonban a változók alatt az egyes objektumokra mutató referenciakat értjük. Maguknak a változóknak nincsenek típusai, így egy szkript futása során bármely, akár különböző objektumra is hivatkozhatnak. Amennyiben egy objektumra az utolsó hivatkozást is töröljük, az automatikus garbage collector szabdTja fel a memóriaterületet. A nem létező változókra való hivatkozás kivételek okoz.

Pythonban a változók alatt az egyes objektumokra mutató referenciakat értjük. Maguknak a változóknak nincsenek típusai, így egy szkript futása során bármely, akár különböző objektumra is hivatkozhatnak. Amennyiben egy objektumra az utolsó hivatkozást is töröljük, az automatikus garbage collector szabdTja fel a memóriaterületet. A nem létező változókra való hivatkozás kivételek okoz.

A Python nyelv is támogatja a váratlan helyzetek a kivételeket. Ebben, egyezrű esetben try kulcsszó után írva szerepel az a kódblokk, amelyben a kivételes helyzet előállhat, majd ezt except blokk követi, amire hiba esetén kerül a vezérlés, illetve opcionálisan egy else ág is lőfordulhat. Az except ág akkor fut le, ha a kifejezést illeszteni lehet arra a kivételobjektumra, amelyre a try kódban, amelyet a try kódblokkja előállított.

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.!

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf(16-22 fólia)

Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPORG repó: source/labor/polargen)

Polártranszformáció segítségével fogunk random számokat generálni, ez az algoritmus annyira népszerű hogy a java random szám generátora is ezt használja.

A Java kód:

```
public class PolarGenerator
{
    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
        if(nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do{
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2* u1 -1;
                v2 = 2* u2 -1;
            }
            while(u1 == 0 && u2 == 0);
            nincsTarolt = false;
        }
        return w;
    }
}
```

```
w = v1*v1 + v2*v2;
} while (w>1);

double r = Math.sqrt((-2 * Math.log(w) / w));
tarolt = r * v2;
nincsTarolt = !nincsTarolt;
return r * v1;

}
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}

public static void main(String[] args)
{
    PolarGenerator g = new PolarGenerator();
    for (int i = 0; i < 10; ++i)
    {
        System.out.println(g.kovetkezo());
    }
}
```

Nézzük részenként:

```
boolean nincsTarolt = true;
double tarolt;
```

PolarGenerator classunk két változót fog tartalmazni, nincsTarolt egy boolean típus, ebben fogjuk állítani hogy van-e már korábban kiszámolt eltárolt tag. A double tarolt-ban pedig a korábban kiszámolt tag értékét fogjuk tárolni.

```
public PolarGenerator()
{
    nincsTarolt = true;
}
```

PolarGenerator class konstruktora, ami annyit csinál, hogy nincsTarolt-at igazra állítja.

```
public double kovetkezo()
{
    if(nincsTarolt)
    {
        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));
tarolt = r * v2;
nincsTarolt = !nincsTarolt;
return r * v1;
}
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}
```

A kovetkezo() metódus végzi a lényegi számítást, ha nincs tárolt érték akkor számolunk két értéket, az egyiket visszaadjuk a másikat eltároljuk, ha van tárolt érték akkor azt adjuk vissza.

```
public static void main(String[] args)
{
    PolarGenerator g = new PolarGenerator();
    for (int i = 0; i < 10; ++i)
    {
        System.out.println(g.kovetkezo());
    }
}
```

A program futása a main-el kezdődik, itt létrehozunk egy objektumot a classunkból, majd for ciklus segítségével 10-szer meghívjuk a korábban létrehozott objektumunk kovetkezo() nevű metódusát ami 10 db random számot fog nekünk generálni a korábban említett módon.

A program a következő parancsokkal fordítható illetve futtatható, nekem az alábbi kimenetet produkálta:

```
robi@ms-7c02: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
robi@ms-7c02:~$ cd Asztal
robi@ms-7c02:~/Asztal$ touch PolarGenerator.java
robi@ms-7c02:~/Asztal$ javac PolarGenerator.java
robi@ms-7c02:~/Asztal$ java PolarGenerator
-0.16752626182170194
0.9353200998838356
0.6083061735939136
-0.08894317232335264
2.037273347521166
-1.2380684613919257
0.3297571186481272
0.11127659741837453
1.5129093213694313
-0.9031365782801662
robi@ms-7c02:~/Asztal$
```

12.1. ábra.

A C++ Forráskód:

```
class PolarGen
{
public:
    PolarGen()
    {
        nincsTarolt = true;
        std::srand (std::time(NULL));
    }
    ~PolarGen()
    {
    }
    double kovetkezo();

private:
    bool nincsTarolt;
    double tarolt;

};
```

```
double PolarGen::kovetkezo ()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1= std::rand() / (RAND_MAX +1.0);
            u2= std::rand() / (RAND_MAX +1.0);
            v1=2*u1-1;
            v2=2*u1-1;
            w=v1*v1+v2*v2;
        }
        while (w>1);

        double r =std::sqrt ((-2 * std::log(w)) /w);

        tarolt=r*v2;
        nincsTarolt =!nincsTarolt;

        return r* v1;
    }

    else
    {
        nincsTarolt =!nincsTarolt;
        return tarolt;
    }
}

int main (int argc, char **argv)
{
    PolarGen pg;

    for (int i= 0; i<10;++i)
        std::cout<<pg.kovetkezo ()<< std::endl;

    return 0;
}
```

Nézzük részenként:

```
class PolarGen
{
public:
    PolarGen()
    {
        nincsTarolt = true;
        std::srand (std::time(NULL));
    }
}
```

```
}

~PolarGen()
{
}

double kovetkezo();

private:
    bool nincsTarolt;
    double tarolt;

};
```

PolarGen classunk, ami két fő részből áll: public és private. A publikus rész elérhető a classon kívülről is, ezzel szemben a privát viszont csak a classon belül érhető el. A classon belüli funkció, aminek a neve megegyezik a class nevével speciális, konstruktornak hívjuk, a konstruktor tartalma akkor hajtódi végre, amikor új objektumot hozunk létre a classból. A classon belüli funkció, aminek a neve megegyezik a class nevével és van előtte egy '~' jel szintén speciális, destruktonak hívjuk. A destruktor tartalma akkor hajtódi végre amikor az objektumunk kimegy az aktuális 'scope'-ból vagy a delete utasítást használjuk a classból létrehozott objektumra mutató pointere.

```
double kovetkezo();
```

Ez a funkció végzi a lényegi számítást.

```
private:
    bool nincsTarolt;
    double tarolt;
```

A classunk privát része, nincsTarolt egy bool-típus(igaz-hamis) benne jelezzen, hogy van-e tárolt érték, double tarolt-ban pedig tároljuk az aktuális értéket.

```
double PolarGen::kovetkezo ()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1= std::rand() / (RAND_MAX +1.0);
            u2= std::rand() / (RAND_MAX +1.0);
            v1=2*u1-1;
            v2=2*u1-1;
            w=v1*v1+v2*v2;
        }
        while (w>1);

        double r =std::sqrt ((-2 * std::log (w) ) /w);

        tarolt=r*v2;
        nincsTarolt =!nincsTarolt;
    }
}
```

```
    return r* v1;
}

else
{
nincsTarolt =!nincsTarolt;
return tarolt;
}
}
```

A kovetkezo() funkció végzi a lényegi számítást. Ha nincs tárolt érték akkor számolunk két értéket, az egyiket visszaadjuk a másikat eltároljuk, ha van tárolt érték akkor azt adjuk vissza.

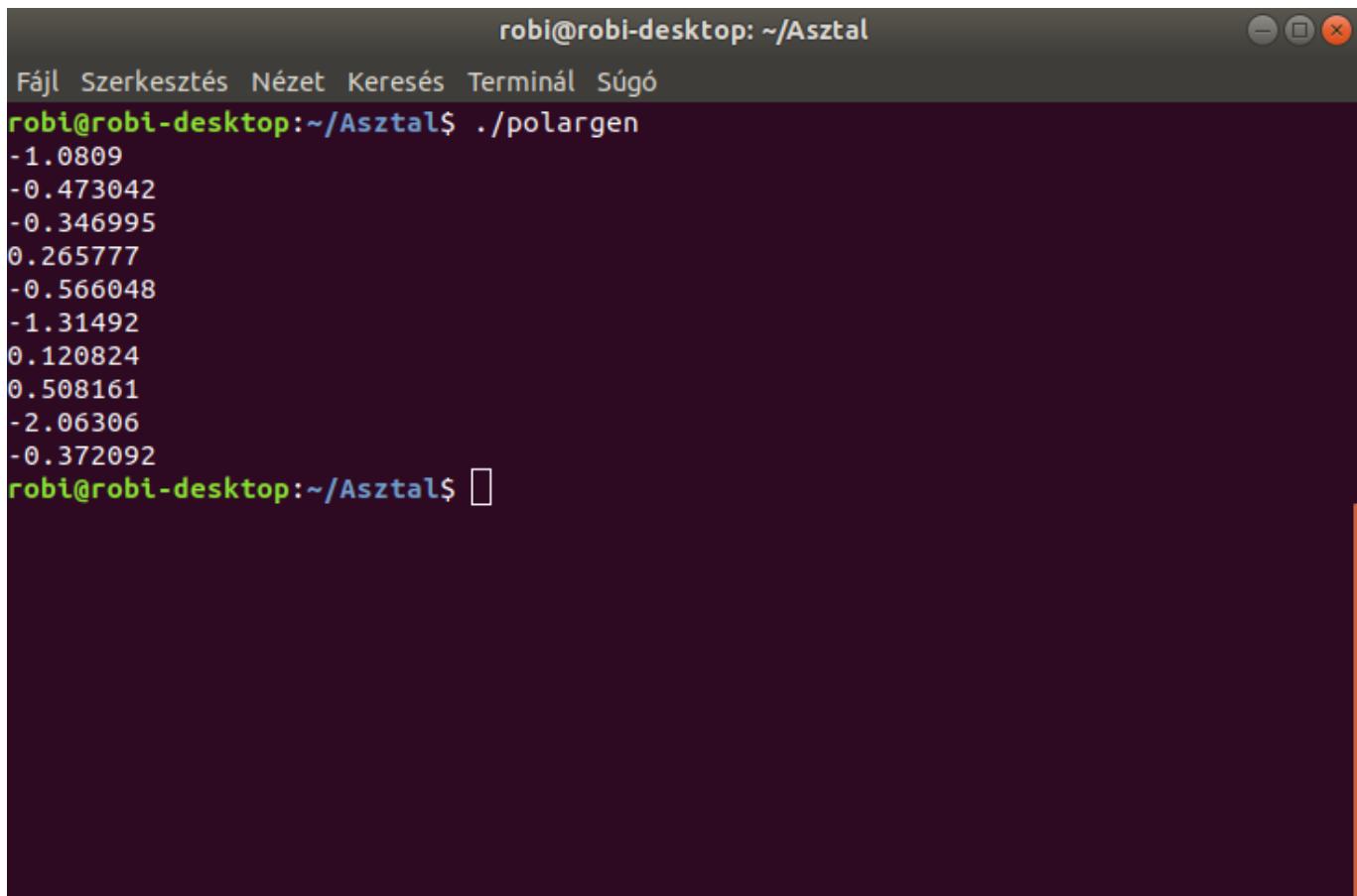
```
int main (int argc, char **argv)
{
PolarGen pg;

for (int i= 0; i<10;++i)
std::cout<<pg.kovetkezo ()<< std::endl;

return 0;
}
```

A main-ben létrehozzuk a pg PolarGen osztályt, majd for ciklussal 10-szer, kiíratjuk kovetkezo() funkció által visszaadott értéket.

A program futtatása után a következő eredményt kapjuk:



```
robi@robi-desktop: ~/Asztal$ ./polargen
-1.0809
-0.473042
-0.346995
0.265777
-0.566048
-1.31492
0.120824
0.508161
-2.06306
-0.372092
robi@robi-desktop:~/Asztal$
```

12.2. ábra. polargen.cpp

12.2. Homokató

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciaikat kell kiirtani és minden már más működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Forrás: <https://github.com/BorvizRobi/vegyes/blob/master/z3a7.java>

Teljes kód:

```
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;

class LZWBinFa
```

```
{  
  
    public LZWBinFa ()  
    {  
        gyoker=new Csomopont ('/');  
        fa=gyoker;  
    }  
  
    void add (char b)  
    {  
        if (b == '0')  
        {  
  
            if (fa.nullasGyermek ()==null) // ha nincs, hát akkor ←  
                csinálunk  
            {  
  
                Csomopont uj = new Csomopont ('0');  
  
                fa.ujNullasGyermek (uj);  
  
                fa = gyoker;  
            }  
            else // ha van, arra rálépünk  
            {  
  
                fa = fa.nullasGyermek ();  
            }  
        }  
        // Mit kell betenni éppen, vagy '1'-et?  
        else  
        {  
            if (fa.egyesGyermek ()==null)  
            {  
                Csomopont uj = new Csomopont ('1');  
                fa.ujEgyesGyermek (uj);  
                fa = gyoker;  
            }  
            else  
            {  
                fa = fa.egyesGyermek ();  
            }  
        }  
    }  
  
    public int getMelyseg ()  
    {  
        melyseg = maxMelyseg = 0;
```

```
rmelyseg (gyoker);
    return maxMelyseg - 1;
}

public double getAtlag ()
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}

public double getSzoras ()
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (gyoker);

    if (atlagdb - 1 > 0)
        szoras = Math.sqrt (szorasosszeg / (atlagdb - 1));
    else
        szoras = Math.sqrt (szorasosszeg);

    return szoras;
}

public void rmelyseg (Csomopont elem)
{
    if (elem != null)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem.egyesGyermek ());
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        rmelyseg (elem.nullasGyermek ());
        --melyseg;
    }
}

public void ratlag (Csmopont elem)
{
    if (elem != null)
    {
        ++melyseg;
        ratlag (elem.egyesGyermek ());
        ratlag (elem.nullasGyermek ());
```

```
--melyseg;
if (elem.egyesGyermek () == null && elem.nullasGyermek () == null)
{
    ++atlagdb;
    atlagosszeg += melyseg;
}
}

public void rszoras (Csomopont  elem)
{
    if (elem != null)
    {
        ++melyseg;
        rszoras (elem.egyesGyermek ());
        rszoras (elem.nullasGyermek ());
        --melyseg;
        if (elem.egyesGyermek () == null && elem.nullasGyermek () == null)
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

public  void kiir (PrintWriter os)
{
    melyseg = 0;

    kiir (gyoker, os);
}

class Csomopont
{

    Csomopont (char b)
    {
        if (b=='0' || b=='1')
            betu=b;
        else
            betu='/';
    }

    Csomopont nullasGyermek ()
    {
```

```
        return balNulla;
    }

    Csomopont egyesGyermek ()
    {
        return jobbEgy;
    }

    void ujNullasGyermek (Csomopont gy)
    {
        balNulla = gy;
    }

    void ujEgyesGyermek (Csomopont gy)
    {
        jobbEgy = gy;
    }

    char getBetu ()
    {
        return betu;
    }

private     char betu;

private     Csomopont balNulla;
private     Csomopont jobbEgy;

};

Csomopont fa;

private int melyseg, atlagosszeg, atlagdb;
private double szorasosszeg;

public void kiir (Csomopont elem, PrintWriter os)
{
    if (elem != null)
    {
        ++melyseg;
        kiir (elem.egyesGyermek (), os);

        for (int i = 0; i < melyseg; ++i)
            os.print ("---");
    }
}
```

```
        os.print( elem.getBetu () + "(" + (melyseg-1) + ")\\n" );
        kiir (elem.nullasGyermek (), os);
        --melyseg;
    }
}

protected Csomopont gyoker;
protected int maxMelyseg;
protected double atlag, szoras;
public static void
usage ()
{
    System.out.println( "Usage: lzwtree in_file -o out_file");
}

public static void  main (String[] args) throws FileNotFoundException,  ←
IOException

{
if (args.length < 3)
{
    usage ();
    return ;
}

String inFile = args[0];

if (!args[1].equals("-o"))
{
    System.out.println(args[1]);
    usage ();
    return;
}

BufferedReader input = new BufferedReader(new FileReader(inFile));

if (input == null)
{
    System.out.println("Nem létezik");
    usage ();
    return;
}

PrintWriter output = new PrintWriter(args[2]);
```

```
int b;
LZWBinFa binfa = new LZWBinFa();

while ((b = input.read ()) != -1)
{
    if ((char)b == '1' || (char)b == '0')
        binfa.add((char)b);

}

binfa.kiir(output);

output.print("depth " + binfa.getMelyseg () + "\n");
output.print("mean " + binfa.getAtlag () + "\n");
output.print("var " + binfa.getSzoras () + "\n");

output.close ();
input.close ();

return;

}

};

}
```

Mivel Javában minden referencia, ezért itt lényegében a pointereket és a C/C++-os referencia jelöléseket kell kiírtani plusz átírni a kódot a Java szintaxisára. Javában nincs operátor túlterhelés sem, ezért a korábbi kódban szereplő output operátor tulterhelését egy funkcióval helyettesítjük. A program működését mér leírtam az első fejezetben ezért itt csak a lényegesebb különbségek mutatnék rá:

```
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;
```

Javában a C/C++-s include# helyett az import-ot használunk lényegében ugyanaz a szerepe mint az #include-nak. Importoljuk a PrintWriter-t a print() használatához, BufferedReader-t a karakter input streamból való olvasáshoz, FileReader-t I/O filestreamok használatához, valamint IOException-t és FileNotFoundException-t a megfelelő kivételkezeléshez.

```
class LZWBinFa
{

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

```
void add (char b)
{
    if (b == '0')
    {

        if (fa.nullasGyermek ()==null) // ha nincs, hát akkor ←
            csinálunk
    {

        Csomopont uj = new Csomopont ('0');

        fa.ujNullasGyermek (uj);

        fa = gyoker;
    }
    else // ha van, arra rálépünk
    {

        fa = fa.nullasGyermek ();
    }
}
// Mit kell betenni éppen, vagy '1'-et?
else
{
    if (fa.egyesGyermek ()==null)
    {
        Csomopont uj = new Csomopont ('1');
        fa.ujEgyesGyermek (uj);
        fa = gyoker;
    }
    else
    {
        fa = fa.egyesGyermek ();
    }
}
}
```

LZWBinFa classunk, azonban itt már referenciák valamint destrukturor nélkül. Javában nincsen destrukturor, ha egy objektumra az összes hivatkozás megszünik akkor egy automatikus szemétygyűjtőgető mechanizmus felszabadítja a memóriaterületet. Az add() funkcióval helyettesítjük a C++-os programban lévő operátor túlterhelést. Szembetűnő eltérés még hogy Javában nincs külön privát és publikus rész egy class-on belül, azaz itt minden tag neve előtt külön fell kell tüntetnünk hogy privát vagy publikus.

```
public int getMelyseg ()
{
    melyseg = maxMelyseg = 0;
    rmelyseg (gyoker);
```

```
        return maxMelyseg - 1;
    }

public double getAtlag ()
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}

public double getSzoras ()
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (gyoker);

    if (atlagdb - 1 > 0)
        szoras = Math.sqrt (szorasosszeg / (atlagdb - 1));
    else
        szoras = Math.sqrt (szorasosszeg);

    return szoras;
}

public void rmelyseg (Csomopont elem)
{
    if (elem != null)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem.egyesGyermek ());
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        rmelyseg (elem.nullasGyermek ());
        --melyseg;
    }
}

public void ratlag (Csomopont elem)
{
    if (elem != null)
    {
        ++melyseg;
        ratlag (elem.egyesGyermek ());
        ratlag (elem.nullasGyermek ());
        --melyseg;
    }
}
```

```
        if (elem.egyesGyermek () == null && elem.nullasGyermek () == null)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }

public void rszoras (Csomopont elem)
{
    if (elem != null)
    {
        ++melyseg;
        rszoras (elem.egyesGyermek ());
        rszoras (elem.nullasGyermek ());
        --melyseg;
        if (elem.egyesGyermek () == null && elem.nullasGyermek () == null)
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}
```

A korábbról ismert funkciók, azonban most a class-on belül definiálva hiszen a Jáva nem engedélyezi class-on kívül definiálni funkciókat.

```
class Csomopont
{

    Csomopont (char b)
    {
        if (b=='0' || b=='1')
            betu=b;
        else
            betu='/';
    }

    Csomopont nullasGyermek ()
    {
        return balNulla;
    }

    Csomopont egyesGyermek ()
    {
        return jobbEgy;
    }
}
```

```
void ujNullasGyermek (Csomopont gy)
{
    balNulla = gy;
}

void ujEgyesGyermek (Csmopont gy)
{
    jobbEgy = gy;
}

char getBetu ()
{
    return betu;
}

private char betu;

private Csmopont balNulla;
private Csmopont jobbEgy;

};
```

Csomopont class-unk itt is a lényegi eltérés a pointerek, referenciaik kiírtása, plusz a Javás szintaxis.

```
public static void main (String[] args) throws FileNotFoundException, ↵
    IOException

{
if (args.length < 3)
{
    usage ();
    return ;
}

String inFile = args[0];

if (!args[1].equals ("-o"))
{
    System.out.println(args[1]);
    usage ();
    return;
}

BufferedReader input = new BufferedReader(new FileReader(inFile));
```

```
if (input == null)
{
    System.out.println("Nem létezik");
    usage ();
    return;
}

PrintWriter output = new PrintWriter(args[2]);

int b;
LZWBinFa binfa = new LZWBinFa();

while ((b = input.read ()) != -1)
{
    if ((char)b == '1' || (char)b == '0')
        binfa.add((char)b);

}

binfa.kiir(output);

output.print("depth " + binfa.getMelyseg () + "\n");
output.print("mean " + binfa.getAtlag () + "\n");
output.print("var " + binfa.getSzoras () + "\n");

output.close ();
input.close ();

return;
}
```

A program main része, ezuttal mivel Javában vagyunk ezért már a class-on belül. Javában ez első parancs-sori argumentum, a C/C++-al elentében már nem a program neve ezért az args tömbben ennek megfelelően átírjuk a számozást. Nyitjuk az input/output streamokat beolvassuk a fába a szöveges fájl-t, kiíratjuk a bin-fát, meghívjuk a mélység,átlag,szórás-t kiszámoló függvényeinket majd zárjuk az input,output streamokat majd returnal visszatérünk.

12.3. „Gagyí”

Az ismert formális:

```
while (x <= t && x >= t && t != x);
```

tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására, hogy a 128-nál inkluzív objektum példányokat poolozza!

```
public static Integer valueOf(int i) {  
    if (i >= IntegerCache.low && i <= IntegerCache.high)  
        return IntegerCache.cache[i + (-IntegerCache.low)];  
    return new Integer(i);  
}
```

A Java feltételezi, hogy a programok sokat dolgoznak majd kis számokkal, ha Integer-re van szükség akkor nem készít új objektumot minden esetben hanem kivesz a pool-ból egy készlet, mivel Javaban a !=, == operátorunk címeket fog összehasonlítani ezért ha a pool-ból vesszük ki a két Integer-t akkor azok egyenlőek lesznek, ellenkező esetben pedig nem. A poolban 127 és -128 közötti értékek találhatóak.

Ebből következnek az alábbi eredmények:

```
public class helloworld{  
  
    public static void main(String args[]){  
        Integer t = 128;  
        Integer x = 128;  
  
        while(x <= t && x>=t && t != x)  
            System.out.println("Végtelen");  
    }  
}
```

Itt mivel a 128, kívül esik a pool-on (új objektumnak foglalunk helyet minden esetben) ezért, a feltételek teljesülnek és végtelen ciklust kapunk.

```
public class helloworld{  
  
    public static void main(String args[]){  
        Integer t = 127;  
        Integer x = 127;  
  
        while(x <= t && x>=t && t != x)  
            System.out.println("Végtelen");  
    }  
}
```

Itt pedig mivel a 127, benne van az előre lefoglalt tartományban, ezért a feltételek nem teljesülnek és nem lépünk végtelen ciklusba.

12.4. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-leáll, ha nem követjük a Yoda conditions-t!
https://en.wikipedia.org/wiki/Yoda_conditions

Yoda feltételek:

A Yoda feltételek, egy programozási stílus, melyben a kifejezés két részét megcseréljük a feltételes utasításokban, eltérően a megszokott rendtől, a Yoda feltételekben a konstans része fog balra kerülni a kifejezésnek. A Yoda név a Star Wars-ból származik, aki szintén a megszokotttól eltérő stílusban, sorrendben beszéli az angolt, nagyjából tárgy-alany-ige stílusban.

Hagyományos feltétel:

```
if (változó == 12)  
// Ha a változó egyenlő 12-vel
```

Yoda feltételel:

```
if (12 == változó)  
// Ha 12 egyenlő a változóval
```

A konstanst a Yoda feltételben az összehasonllító operátor bal oldalára írjuk, és a változót aminek az értékét hasonlítjuk a konstansal jobra.

A Yoda stílus előnye:

A konstans bal oldalra helyezése nem változtatja meg a program viselkedését ,kivéve ha az értékek hamisra lesznek kiértékelve.

Például a következő kifejezésben:

```
if (változó = 12) /* ... */  
// Ez a kifejezés hozzárendeli a 12 számot a változóhoz  
  
if (12 = változó) /* ... */  
// Ez pedig egy szintaktikai hiba ami nem fog fordulni
```

Az előző példában mivel 12 egy konstans, vagyis nem változhat, ezért ezt a hibát elkapja a fordító.

```
Boolean sajátboolean = true;  
if(sajátboolean = null) /* ... */  
// Ez egy NullPointerException-t okoz Java-ban, de le lehet fordítani
```

A Yoda feltételekkel ekerülhető néhány nem biztonságos null viselkedés:

```
String sajátString = null;  
if(sajátstring.equals("valami")) /* ... */  
//Ez NullPointerException-t okoz Java-ban
```

Viszont Yoda feltétellel:

```
String sajátstring = null;  
if("valami".equals(sajátstring)) /* ... */  
//Ez viszont hamis, ahogy várható lenne
```

Yoda feltételek hátránya:

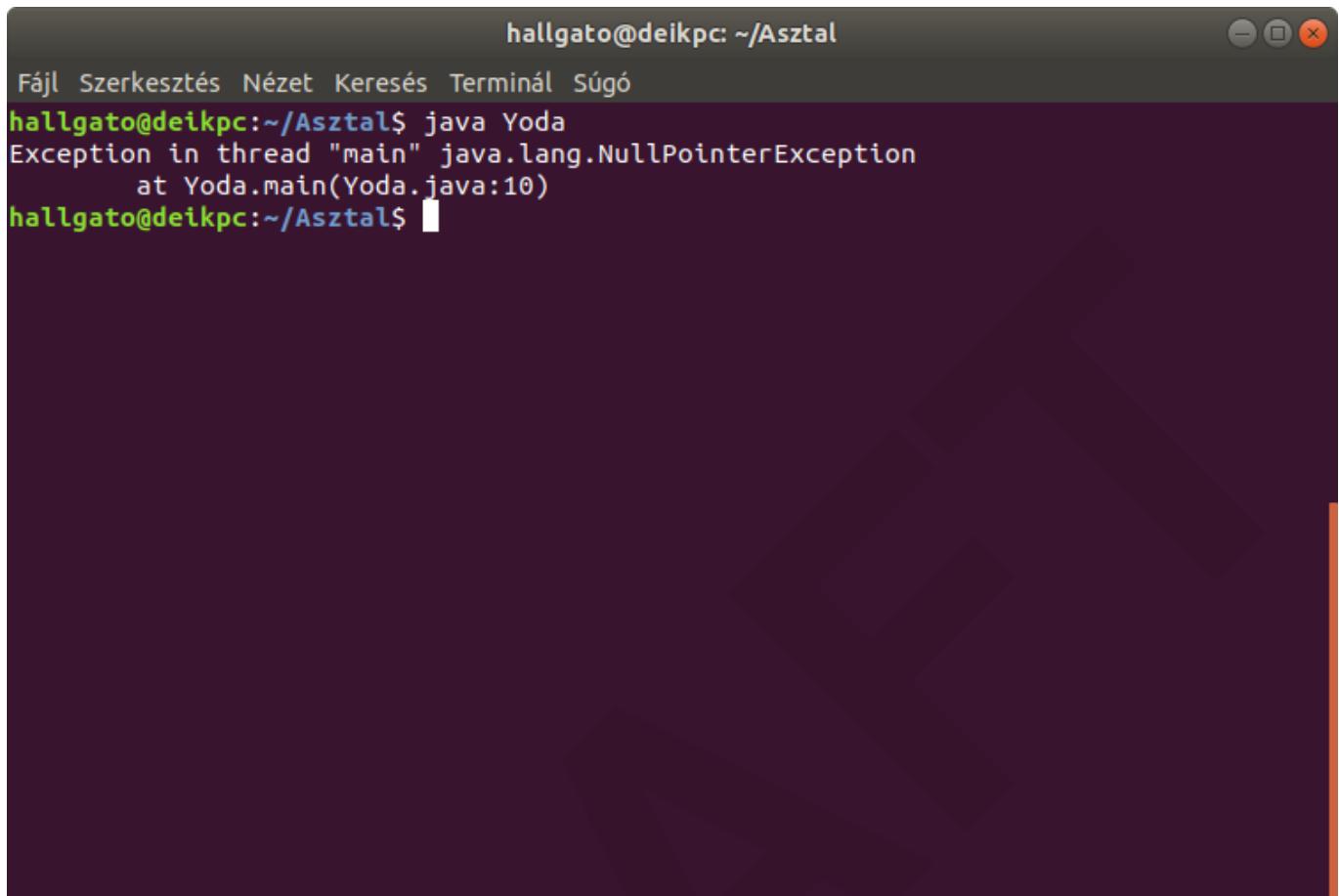
A Yoda feltételek kritikusai szerint így nehezebben olvasható a kód, ami tulsúlyozza a fent említett előnyeit. Néhány programozási nyelv például a Swift, nem engedélyez változó hozzárendeléseket feltételes

kifejezésekben. Az előny amivel elkerüljük a null pointer hibát, hátrányaiknak is tekinthető, mivel így a null pointer hibák rejtettek maradhatnak és később okozhatnak gondot a programban.

A következő programkód java.lang.NullPointerException-el leáll:

```
class Yoda{  
  
    public static void main(String args[])  
    {  
  
        String sajátstring = null;  
  
        if(sajátstring.equals("valami"))  
        {  
  
            System.out.println("Yoda");  
  
        }  
  
    }  
}
```

A kód a következő hibaüzenetet produkálja:



The screenshot shows a terminal window with a dark background and light-colored text. At the top, it displays the user's name, host, and current directory: `hallgato@deikpc: ~/Asztal`. Below this, the terminal menu bar includes options for Fájl, Szerkesztés, Nézet, Keresés, Terminál, and Súgó. The main area of the terminal shows the command `java Yoda` being run, followed by the resulting exception message:

```
Exception in thread "main" java.lang.NullPointerException
  at Yoda.main(Yoda.java:10)
```

12.3. ábra.

Ez a Yoda feltétele kivédhető lett volna.

12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhabailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Tanulságok, tapasztalatok, magyarázat...

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai- Barki/madarak/)

13.2. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

13.3. Anti OO

A BBP algoritmussal 4 a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10 6, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apas03.html#id561066>

13.4. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNISTforHumansExp3/app/src/main> Apró módosításokat eszközölj benne, pl. színvilág.

13.5. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf (77-79 fóliát)!

DRAFT