

Univerzális programozás

Így neveld a programozód!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert Á&s Simion, Tamás	2019. október 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai

Ajánlás

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

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

DRAFT

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	9
2.3. Változók értékének felcserélése	11
2.4. Labdapattogás	12
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	14
2.6. Helló, Google!	14
2.7. A Monty Hall probléma	17
2.8. 100 éves a Brun téTEL	19
3. Helló, Chomsky!	24
3.1. Decimálisból unárisba átváltó Turing gép	24
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	25
3.3. Hivatkozási nyelv	26
3.4. Saját lexikális elemző	29
3.5. Leetspeak	31
3.6. A források olvasása	35
3.7. Logikus	37
3.8. Deklaráció	37

4. Helló, Caesar!	41
4.1. double ** háromszög mátrix	41
4.2. C EXOR titkosító	43
4.3. Java EXOR titkosító	45
4.4. C EXOR törő	47
4.5. Neurális OR, AND és EXOR kapu	50
4.6. Hiba-visszaterjesztéses perceptron	56
5. Helló, Mandelbrot!	57
5.1. A Mandelbrot halmaz	57
5.2. A Mandelbrot halmaz a std::complex osztálytalál	59
5.3. Biomorfok	63
5.4. A Mandelbrot halmaz CUDA megvalósítása	68
5.5. Mandelbrot nagyító és utazó C++ nyelven	69
5.6. Mandelbrot nagyító és utazó Java nyelven	72
6. Helló, Welch!	78
6.1. Első osztályom	78
6.2. LZW	82
6.3. Fabejárás	86
6.4. Tag a gyökér	87
6.5. Mutató a gyökér	87
6.6. Mozgató szemantika	88
7. Helló, Conway!	91
7.1. Hangyszimulációk	91
7.2. Java életjáték	108
7.3. Qt C++ életjáték	115
7.4. BrainB Benchmark	116
8. Helló, Schwarzenegger!	137
8.1. Szoftmax Py MNIST	137
8.2. Mély MNIST	137
8.3. Minecraft-MALMÖ	137

9. Helló, Chaitin!	138
9.1. Iteratív és rekurzív faktoriális Lisp-ben	138
9.2. Gimp Scheme Script-fu: króm effekt	138
9.3. Gimp Scheme Script-fu: név mandala	138
10. Helló, Gutenberg!	139
10.1. Programozási alapfogalmak	139
10.2. Programozás bevezetés	141
10.3. Programozás	142
III. Második felvonás	144
11. Helló, Berners-Lee!	146
11.1. Java 2 útikalauz programozóknak 5.0 I-II. és C++ összehasonlítás	146
11.2. Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)	147
12. Helló, Arroway!	148
12.1. OO szemlélet	148
12.2. Homokatózó	152
12.3. „Gagyí”	152
12.4. Yoda	154
12.5. Kódolás from scratch	155
13. Helló, Liskov!	159
13.1. Liskov helyettesítés sértése	159
13.2. Szülő-gyerek	161
13.3. Anti OO	163
13.4. deprecated - Hello, Android! (zöld feladat)	165
13.5. Hello, Android!	165
13.6. Hello, SMNIST for Humans! (piros feladat)	166
13.7. Ciklomatikus komplexitás	166

14. Helló, Mandelbrot!	170
14.1. Reverse engineering UML osztálydiagram	170
14.2. Forward engineering UML osztálydiagram	171
14.3. Egy esettan	172
14.4. BPMN	173
14.5. BPEL Helló, Világ! - egy visszhang folyamat (zöld)	173
14.6. TeX UML	173
15. Helló, Chomsky!	175
15.1. Encoding	175
15.2. OOCWC lexer	175
15.3. l334d1c4 ⁵ (zöld)	175
15.4. Full screen(zöld)	176
15.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció	176
15.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció	176
15.7. Perceptron osztály	176
16. Helló, Stroustrup!	177
16.1. OO szemlélet	177
16.2. Homokatózó	177
16.3. „Gagyi”	177
16.4. Yoda	177
16.5. Kódolás from scratch	178
17. Helló, Gödel!	179
17.1. OO szemlélet	179
17.2. Homokatózó	179
17.3. „Gagyi”	179
17.4. Yoda	179
17.5. Kódolás from scratch	180
18. Helló, hetedikfejezet!	181
18.1. OO szemlélet	181
18.2. Homokatózó	181
18.3. „Gagyi”	181
18.4. Yoda	181
18.5. Kódolás from scratch	182

19. Helló, Schwarzenegger!	183
19.1. OO szemlélet	183
19.2. Homokatózó	183
19.3. „Gagyí”	183
19.4. Yoda	183
19.5. Kódolás from scratch	184
20. Helló, Calvin!	185
20.1. MNIST	185
20.2. Deep MNIST	185
20.3. CIFAR-10 (zöld feladat)	185
20.4. Android telefonra a TF objektum detektálója	186
20.5. SMNIST for Machines	186
20.6. SMNIST for Machines	186
IV. Irodalomjegyzék	187
20.7. Általános	188
20.8. C	188
20.9. C++	188
20.10Lisp	188

Ábrák jegyzéke

2.1.	7
2.2.	8
2.3.	9
2.4.	15
2.5. Egyserűsítve a lehetőségek	18
2.6.	19
2.7.	21
2.8. A B_2 konstans közelítése	23
3.1.	25
3.2.	27
3.3.	28
3.4.	28
3.5.	29
3.6.	33
4.1.	43
4.2. A double ** háromszögmátrix a memóriában	43
4.3.	45
4.4.	47
4.5.	50
4.6.	50
4.7. Neurális OR	52
4.8. Neurális AND	53
4.9. Neurális EXOR	54
4.10. Neurális EXOR	55
4.11. Neurális EXOR	56

5.1.	Eltet idő	58
5.2.	Eredmény	58
5.3.	A Mandelbrot halmaz a komplex síkon	60
5.4.	A Mandelbrot halmaz a komplex síkon	61
5.5.	Eredmény	68
5.6.	Nagyítás előtt	72
5.7.	72
5.8.	76
5.9.	76
5.10.	77
5.11.	77
6.1.	80
6.2.	82
6.3.	85
6.4.	90
7.1.	107
7.2.	108
7.3.	115
7.4.	116
7.5.	136
12.1.	Eredmény	149
12.2.	Eredmény	152
12.3.	Eredmény	153
12.4.	Eredmény	154
12.5.	Eredmény	154
12.6.	Eredmény	155
12.7.	Eredmény	158
13.1.	Eredmény	162
13.2.	Eredmény	163
13.3.	C eredmény	164
13.4.	C++ eredmény	164
13.5.	Java eredmény	164

13.6. C# eredmény	165
13.7. Összesített eredmény	165
13.8. Cikomatikus komplexitás Java	167
13.9. Cikomatikus komplexitás C++	169
14.1. UML diagramm	171
14.2. Eredmény	172
14.3. BPMN	173

DRAFT

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

A ciklusok segytenek abban hogy egy kódot ne kelljen ismétleni, hanem egyszeri megírással többször is végig tudunk futni rajta. A ciklus addig fut amit a ciklusfejben meg adott feltételt teljesül. Ha a feltétel elírjuk vagy szándékosan olyan feltételt adunk meg amiről tudjuk hogy sose fog teljesülni akkor jön létre egy végtelen ciklus. A végtelen ciklus sosem áll meg.

Megoldás videó:

Megoldás forrása:

100 százalékban dolgoztat egy magot

Forrákód linkje: [..../bhax/blob/master/Forrasok/2.Fejezet/2.1/2.1.1.c](#)

Egy mag 100%-on való futtatása egyszerű, csak egy végtelen ciklust kell irni és futtatni.

```
#include <stdio.h>

int main()
{
    int x=0;
    while (x<1)
    {
    }
}
```

Fordítás: **gcc fájlnév.c -o fájlnév**

Futtatás: **./fájlnév**

```

Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~$ top

top - 16:50:50 up 2 days, 22:51, 1 user, load average: 2,22, 1,62, 0,95
Tasks: 281 total, 2 running, 231 sleeping, 0 stopped, 1 zombie
%Cpu0 : 2,3 us, 0,0 sy, 0,0 ni, 97,3 id, 0,3 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 1,3 us, 0,7 sy, 0,0 ni, 98,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 : 1,3 us, 0,7 sy, 0,0 ni, 98,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 8062064 total, 534596 free, 4673716 used, 2853752 buff/cache
KiB Swap: 15998972 total, 15998972 free, 0 used. 2301072 avail Mem

          PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  5865 simi      20   0   4376    756   692 R 100,0  0,0   0:11.19 2.1.1
 1496 simi      20   0 3978744  447644  83492 S  2,0  5,6  46:25.34 gnome-shell
 1379 simi      20   0  698704 141688 124608 S  1,0  1,8 27:29.83 Xorg
 3202 simi      20   0 12,135g 267364  82400 S  1,0  3,3  4:02.34 chrome
 4714 simi      20   0  732760  40416 29992 S  0,7  0,5  0:03.57 gnome-terminal
23229 simi      20   0 1435880 505580 115652 S  0,7  6,3 272:04.51 chrome
 1536 simi      20   0  364204   9028  6472 S  0,3  0,1  0:43.22 ibus-daemon
 4895 simi      20   0 1466912 602600 154784 S  0,3  7,5 14:20.50 chrome
 5774 simi      20   0  52992   4296  3420 R  0,3  0,1  0:00.96 top
10467 simi      20   0 1684996 626432  95440 S  0,3  7,8  86:42.06 chrome
  1 root       20   0  225832   9588  6692 S  0,0  0,1  0:38.74 systemd
  2 root       20   0      0      0      0 S  0,0  0,0  0:00.03 kthreadd

```

2.1. ábra.

100 százalékban dolgoztat minden magot

Forrákód linkje: [./bhax/blob/master/Forrasok/2.Fejezet/2.1/2.1.2.c](#)

A magok 100%-on való futtatásához felhasználjk az omp.h header fájt ami tartalmazza a **#pragma omp parallel{ }** parancsot. Ez a parancs elágazásokat hoz létre amik a kpacsos zárojelek közti utasítást folytatják, így a processzor minden szálát 100%-ig dolgoztatva.

```

#include <stdio.h>
#include <omp.h>

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

Fordítás: **gcc fájlnév.c -o fájlnév -fopenmp**

Futtatás: **./fájlnév**

```

Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~$ top

top - 16:49:13 up 2 days, 22:50,  1 user,  load average: 3,28, 1,26,  0,76
Tasks: 279 total,   2 running, 229 sleeping,   0 stopped,   1 zombie
%Cpu0 : 98,7 us,  1,3 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 : 99,7 us,  0,3 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu3 : 99,7 us,  0,3 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem : 8062064 total, 549316 free, 4644368 used, 2868380 buff/cache
KiB Swap: 15998972 total, 15998972 free,      0 used. 2315076 avail Mem

          PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
      5782 simi      20   0  35576   884    788 R 374,4  0,0  4:49.84 2.1.2
  1496 simi      20   0 3977912 446984  82496 S 12,0  5,5 46:21.40 gnome-shell
 1379 simi      20   0  700976 141900 124820 S  3,7  1,8 27:28.31 Xorg
23808 simi      20   0  856792 55156  39208 S  2,7  0,7  0:10.04 nautilus
 2785 simi      20   0 1716432 316892 129424 S  2,3  3,9 47:09.67 chrome
 2828 simi      20   0  731832 236228 158356 S  1,0  2,9 64:29.74 chrome
23229 simi      20   0 1435880 505464 115652 S  0,7  6,3 272:03.46 chrome
 1307 root      20   0 1186700 11076  9620 S  0,3  0,1  0:50.67 teamviewererd
 1475 simi      20   0  50052  4452  3856 S  0,3  0,1  0:01.03 dbus-daemon
 1477 simi      20   0  220772  7024  6316 S  0,3  0,1  0:05.97 at-spi2-re+
 1536 simi      20   0  364204  9028  6472 S  0,3  0,1  0:42.92 ibus-daemon
 4714 simi      20   0  732760 40416 29992 S  0,3  0,5  0:03.11 gnome-term+

```

2.2. ábra.

0 százalékban dolgoztatja a magokat

Forrákód linkje: [./bhax/blob/master/Forrasok/2.Fejezet/2.1/2.1.3.c](#)

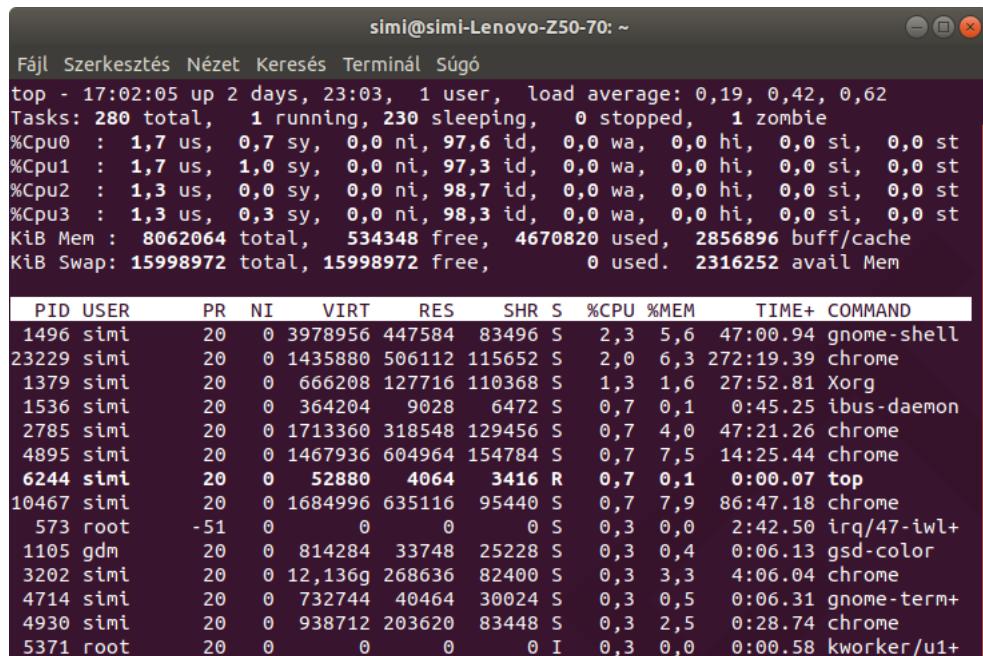
Ahhoz hogy egy magot 0%-ban dolgoztassunk segítégül hívjuk a **sleep()** függvényt, amiről a man 3 sleep kézikönyv böngészésével többet is megtudhatunk. A **sleep()** felülggeszti a folyamat végrehajtását annyi másodpercre amit argumentumként megadunk neki.

```
#include <unistd.h>

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

Fordítás: **gcc fájlnév.c -o fájlnév**

Futtatás: **./fájlnév**



```

Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70: ~
top - 17:02:05 up 2 days, 23:03, 1 user, load average: 0,19, 0,42, 0,62
Tasks: 280 total, 1 running, 230 sleeping, 0 stopped, 1 zombie
%Cpu0 : 1,7 us, 0,7 sy, 0,0 ni, 97,6 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 1,7 us, 1,0 sy, 0,0 ni, 97,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 : 1,3 us, 0,0 sy, 0,0 ni, 98,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 1,3 us, 0,3 sy, 0,0 ni, 98,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 8062064 total, 534348 free, 4670820 used, 2856896 buff/cache
KiB Swap: 15998972 total, 15998972 free, 0 used. 2316252 avail Mem

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1496 simi      20   0 3978956 447584 83496 S  2,3  5,6 47:00.94 gnome-shell
23229 simi      20   0 1435880 506112 115652 S  2,0  6,3 272:19.39 chrome
1379 simi      20   0 666208 127716 110368 S  1,3  1,6 27:52.81 Xorg
1536 simi      20   0 364204  9028  6472 S  0,7  0,1 0:45.25 ibus-daemon
2785 simi      20   0 1713360 318548 129456 S  0,7  4,0 47:21.26 chrome
4895 simi      20   0 1467936 604964 154784 S  0,7  7,5 14:25.44 chrome
6244 simi      20   0 52880  4064  3416 R  0,7  0,1 0:00.07 top
10467 simi      20   0 1684996 635116 95440 S  0,7  7,9 86:47.18 chrome
 573 root      -51   0      0      0      0 S  0,3  0,0 2:42.50 irq/47-iwl+
1105 gdm      20   0 814284 33748 25228 S  0,3  0,4 0:06.13 gsd-color
3202 simi      20   0 12,136g 268636 82400 S  0,3  3,3 4:06.04 chrome
4714 simi      20   0 732744 40464 30024 S  0,3  0,5 0:06.31 gnome-terminal
4930 simi      20   0 938712 203620 83448 S  0,3  2,5 0:28.74 chrome
5371 root      20   0      0      0      0 I  0,3  0,0 0:00.58 kworker/u1+

```

2.3. ábra.

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

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

Megoldás videó:

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

```

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

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

```

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

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

akár önmagára

```
T100 (T100)
false
```

ezt a kimenetet adja.

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

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Változók felcserélése az exor művelettel

Forráskód linkje: <..//bhax/blob/master/Forrasok/2.Fejezet/2.3/csereexor.c>

```
#include <stdio.h>

int main()
{
    printf("szamok\n");
    int szam1=3; //binarisan 11
    int szam2=5; //binarisan 101

    printf("%d\n", szam1);
    printf("%d\n", szam2);

    szam1=szam1^szam2; //xor művelet után 110
    szam2=szam1^szam2; //110^101=11, megkapjuk a szam2 új értékét ami ←
                        decimáisan 3
    szam1=szam1^szam2; //110^11=101, ami decimálisan 5

    printf("A szamok csere után:\n");
    printf("%d\n", szam1);
    printf("%d\n", szam2);
}
```

Változók felcserélése összeadás és kivonás segítségével

Forráskód linkje: <..//bhax/blob/master/Forrasok/2.Fejezet/2.3/cserekivonas.c>

```
#include <stdio.h>

int main()
{
    printf("szamok\n");
    int szam1=3;
    int szam2=5;

    printf("%d\n", szam1);
    printf("%d\n", szam2);

    szam1=szam1-szam2; //a két szám különbözege
    szam2=szam1+szam2; //szam2-höz hozzá adva a különbséget megkapja a szam1- ←
                        et
```

```
szam1=szam2-szam1; //szam2 jelenlegi értékéből kivonva a különbösséget ←  
megkapjuk a szam1-t  
  
printf("A szamok csere után:\n");  
printf("%d\n",szam1);  
printf("%d\n",szam2);  
}
```

Váltzók felcserélése szorzás és összeadás segítégével

```
#include <stdio.h>  
  
int main()  
{  
    printf("szamok\n");  
    int szam1=3;  
    int szam2=5;  
  
    printf("%d\n",szam1);  
    printf("%d\n",szam2);  
  
    szam1=szam1*szam2; //számok szorzata  
    szam2=szam1/szam2; //a számok szorzatát elosztva a második számmal ←  
    megkapjuk a szam2-t  
    szam1=szam1/szam2; //a számok szorzatát elosztva a második szám ←  
    jelenlegi értékével megkapjuk a szam1-et  
  
    printf("A szamok csere után:\n");  
    printf("%d\n",szam1);  
    printf("%d\n",szam2);  
}
```

Forráskód linkje: [./bhax/blob/master/Forrasok/2.Fejezet/2.3/csereszorzas.c](#)

2.4. Labdapattogás

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

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

Megoldás forrása:

Labdapattogás if-fel

Forráskód linkje: [./bhax/blob/master/Forrasok/2.Fejezet/2.4/labda.c](#)

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

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

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ; ) {

        getmaxyx ( ablak, my , mx ); //ablak mérete

        mvprintw ( y, x, "O" ); //kiiratjuk a "O"-t az x,y koordinátára

        refresh ();
        usleep ( 100000 );

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

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

    }

    return 0;
}
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

A képernyőn megjelenő érték 32, vagyis ennyi bitból áll egy gépi szó.

Forráskód linkje: [/bhax/blob/master/Forrasok/2.Fejezet/2.5/szohossz.c](#)

```
#include <stdio.h>

int main()
{
    printf("%ld\n", sizeof(int)*8);
    return 0;
}
```

A "Bogo" szó az angol "bogus" szóból ered, ami az jelenti: hamis. A MIPS a Millió Utasítás Másodpercenként rövidítése, ez a számítógép számítási sebességének mértékegysége.

A BogoMIPS Linus Torvalds találmánya. Az eredményből pedig a processzor sebességére tudunk következtetni.

A BogoMIPS ugyancsak a szóhosszt nézi csak bitműveletes megoldással.

Forráskód linkje: [../bhax/blob/master/Forrasok/2.Fejezet/2.5/bogomips.c](#)

2.6. Helló, Google!

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

Megoldás videó:

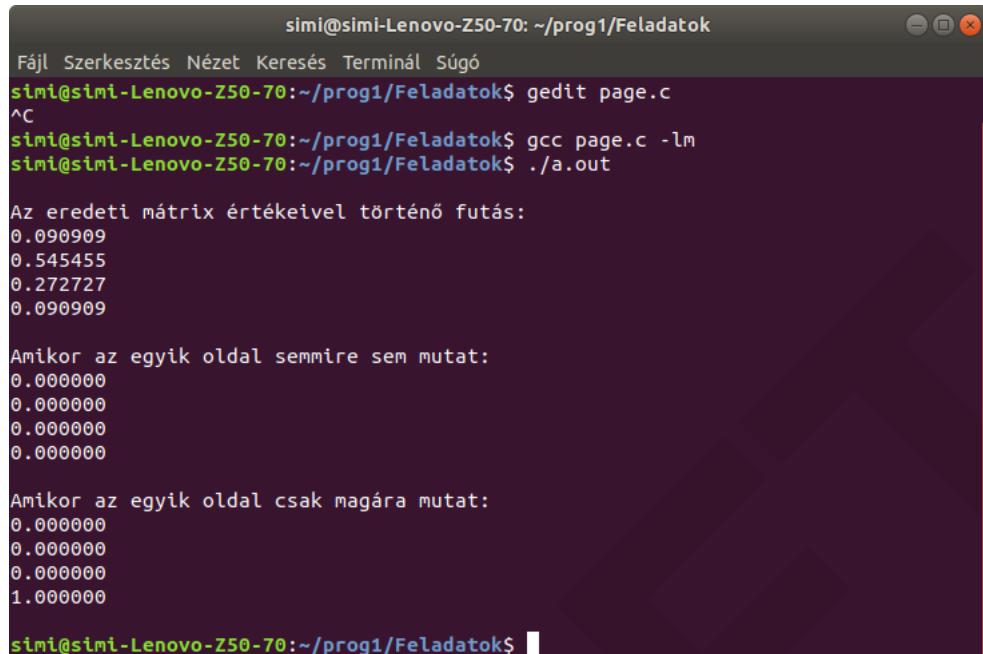
Megoldás forrása:

Forráskód linkje: [/bhax/blob/master/Forrasok/2.Fejezet/2.6/page.c](#)

Fordítás: **gcc fájlnév.c -lm**

Futtatás: **./a.out**

Az ötlet Larry Page és Sergery Brin a Google alapítói találták ki és a mai napig a Google keresőjének egyik legfőbb része. A pagerank megadja a weboldal nice ness értékét. Egy weboldalra minél több másik oldal mutatt annál nagyobb a nice ness, "jóság" értéke.



```
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gedit page.c
^C
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc page.c -lm
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./a.out

Az eredeti mátrix értékeivel történő futás:
0.090909
0.545455
0.272727
0.090909

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

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

simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$
```

2.4. ábra.

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

void
kiir (double tomb[], int db) {

    int i;

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

double
tavolsag (double PR[], double PRv[], int n) {

    int i;
    double osszeg=0;

    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

    return sqrt(osszeg);
}

void
pagerank(double T[4][4]){
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 };      //ebbe megy az eredmény
```

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

int i, j;

for(;;) {

    // ide jön a mátrix művelet

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

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

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

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

kiir (PR, 4);
}

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

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

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

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

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

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

printf("\n");

return 0;
}
```

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

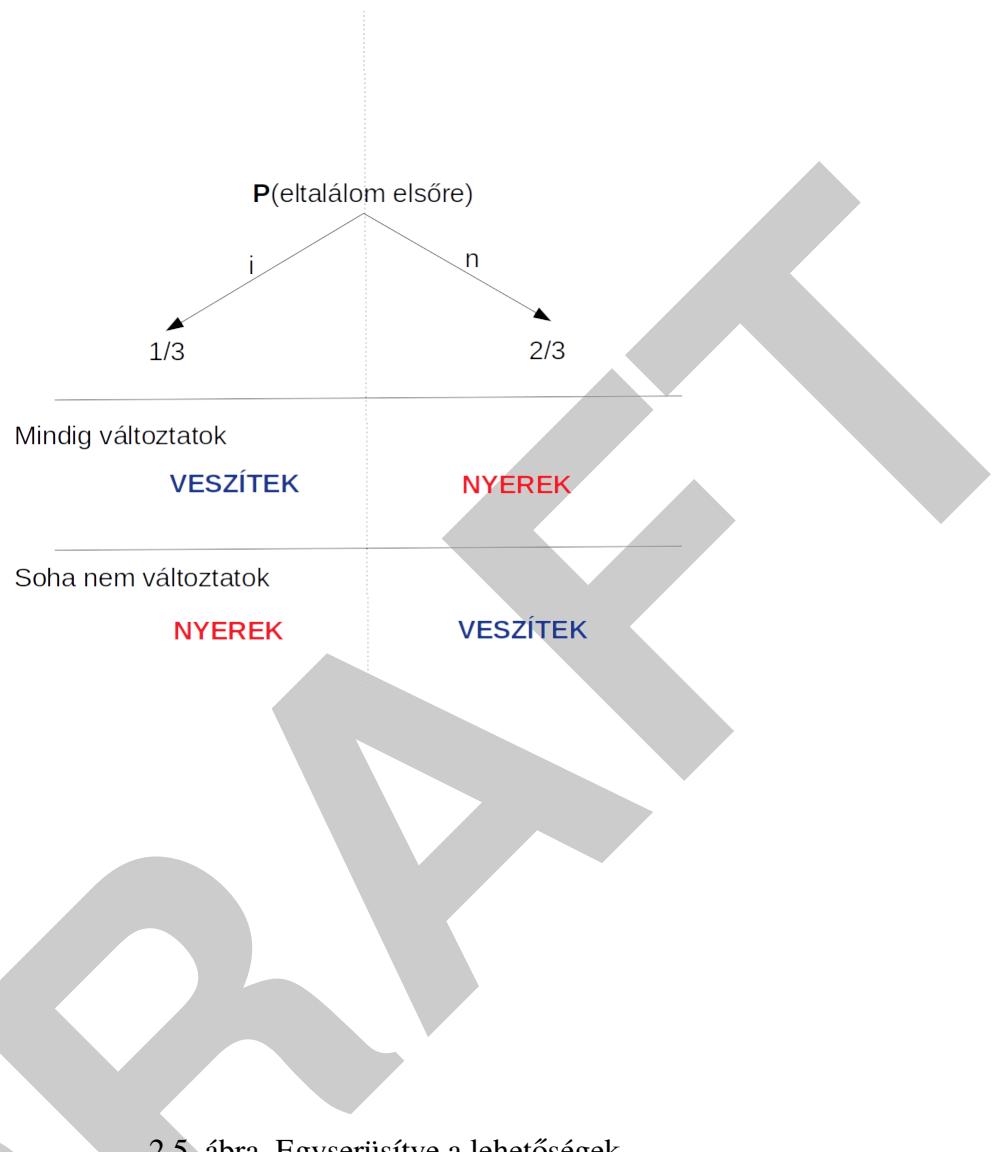
Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R

A Monty Hall probléma probléma. A probléma arról szól, hogy van 3 ajtó, ezek közül 2 értektelen dolgot, 1 pedig egy értékes dolgot rejti. A valószínűség számítást használva ki akarjuk deryteni mennyi az esélye annak hogy a 3 ajtó közül eltaláljuk azt amelyik mögött ott az érték. Ez a probléma egy amerikai játékból ered, ahol egy játékonak 3 ajtó közül kell egyet választania ($1/3$ esélye van rá hogy jó ajtót választ), miután választott a műsorvezető kinyit neki egy ajtot(nem lehet az amit a játékos választott, és nem lehet a nyereményes ajtó), majd változtat-e azon hogy melyik ajtot választja. Itt válik ketté a dolog.

Ha a játékos elsőre a nyerő ajtót választotta, akkor mindegy melyik ajtót nyitja ki a műsorvezető értektelen dolgot lesz mögötte. Ha úgy dönt a játékos hogy cserél biztos veszít, ellenkező esetben övé a nyeremény.

Ha viszont olyan ajtót választ elsőre (ennek $2/3$ az esélye) akkor a műsorvezetőnek csak egy lehetséges esete van, hogy melyik ajtót mutatja meg a játékosnak. Ha nem cserél biztos veszít, ha cserél biztos nyer, mivel a másik nyereménytelen ajtót kinyitotta a műsorvezető.

A programban megadott kísérletszámra kiszámoljuk mennyi az esélyünk nyerni akkor ha sose változtatunk azon az ajtón amit elsőre választottunk és mennyi akkor ha minden változtatunk.



2.5. ábra. Egyserűsítve a lehetőségek

Forráskód linkje: [..../bhax/blob/master/Forrasok/2.Fejezet/2.7/mh.r](#)

```

sim@simi-Lenovo-Z50-70: ~
Fájl Szöveges Nézet Keresés Terminál Súgó
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 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: %", kiserletek_szama)
[1] "Kiserletek szama: 10000000"
> length(nemvaltoztatesnyer)
[1] 33348
> length(valtoztatesnyer)
[1] 6664532
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.5004805
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 10000000
>

```

2.6. ábra.

2.8. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

Forráskód linkje: [..../bhax/blob/master/Forrasok/2.Fejezet/2.8/stp.r](#)

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

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

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

Majd vizsgáljuk meg az a sorozatot:

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

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

- ...
- $(n+1)!+(n-1)=1*2*3*\dots*(n-1)*n*(n+1)+(n-1)$, azaz $(n-1)*\text{valamennyi}+(n-1)$, ami osztható $(n-1)$ -el
- $(n+1)!+n=1*2*3*\dots*(n-1)*n*(n+1)+n$, azaz $n*\text{valamennyi}+n-$, ami osztható n -el
- $(n+1)!+(n+1)=1*2*3*\dots*(n-1)*n*(n+1)+(n-1)$, azaz $(n+1)*\text{valamennyi}+(n+1)$, ami osztható $(n+1)$ -el

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

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

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

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

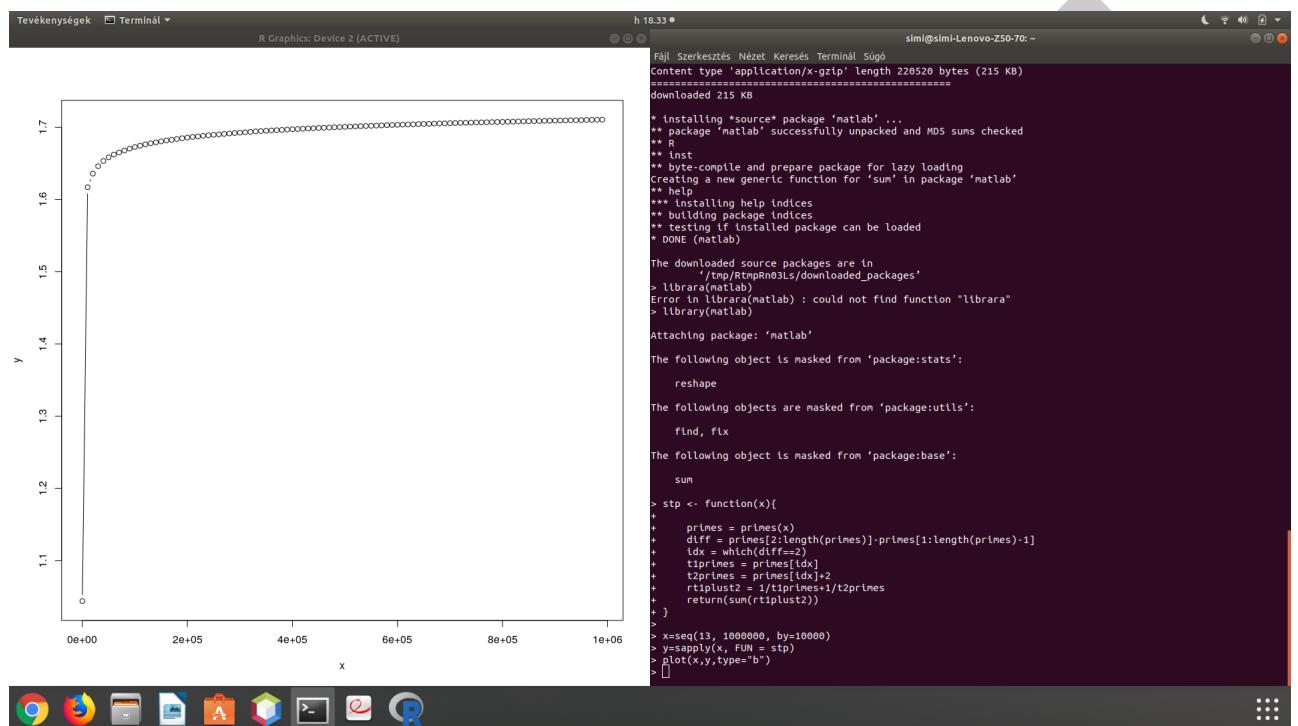
```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later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

stp <- function(x) {

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

```
}
```

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



2.7. ábra.

Soronként értelemezzük ezt a programot:

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

Kiszámolja a megadott számig a prímeket.

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

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

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

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

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

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

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

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

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

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

Az $1/t1primes$ a `t1primes` 3,5,11 értékéből az alábbi reciprokokat képzi:

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

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

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

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

```
> 1/t1primes+1/t2primes
[1] 0.53333333 0.34285711 0.1678322
```

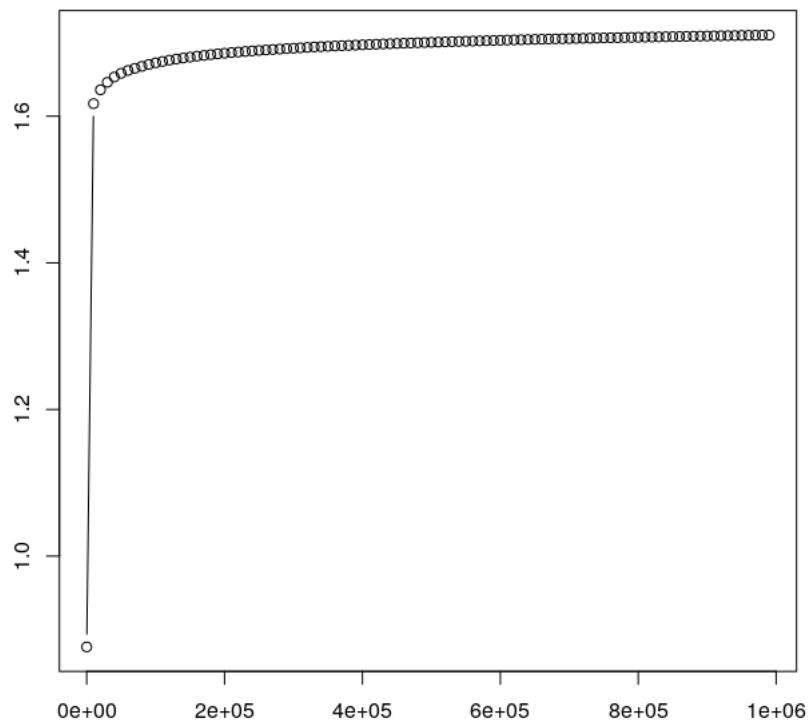
Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvényel.

```
sum(rt1plust2)
```

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

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

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



2.8. ábra. A B_2 konstans közelítése

Werkfilm

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

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#id511324>

Forráskód linkje: <https://gitlab.com/tamas.simon12/bhax/blob/master/Forrasok/3.Fejezet/decim.c>

A Turing gép működését Alan Turing angol matematikus dolgozta ki 1936-ban. A Turing gép a mai számítógépek legyszártípusított modelje.

Fordítás: **gcc decim.c -o decim**

Futtatás: **./decim**

Az alábbi program decimális(tízes) számrendszerből alakít unitárius azaz egyes számrendszerbe. Az unitárius számrendszer a legegyszerűbb. Az átalakiítás úgy történik hogy a programnak megadott számot, például 12, 12 "l" karakterrel írja le, ötöseel tördelve a következőképpen: ||||| ||||| ||.

```
#include <stdio.h>

int
main()
{
    int a, db=0;
    printf("Adjon meg egy decimalis szamot!\n");
    scanf("%d", &a);
    printf("A megadott szam unarisba atvaltva:\n");
    for (int i = 0; i < a; i++)
    {
        printf(" | ");
        db++;
        if (db % 5 == 0)
        {
            printf("   ");
        }
    }
}
```

```
    }
    printf("\n");
    return 0;
}
```

```
simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc decim.c -o decim
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./decim
Adjon meg egy decimalis szamot!
12
A megadott szam unarisba atváltva:
||||| ||||| ||
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ 
```

3.1. ábra.

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

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

Megoldás videó:

Megoldás forrása:

Forráskód linkje:

A generatív nyelvtan egy olyan rendszer ami egy algoritmus segítségével legenerál egy nyelvet. Kell egy kezdeti érték majd a szabalyokat alkalmazzuk.

S, X? Y legyenek változók
a, b, c legyenek konstansok

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

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

aYbbcc (aY - aaX)

aaXbbcc (Xb - bX)
aabXbcc (Xb - bX)
aabbXcc (Xc - Ybcc)
aabbYbcc (bY - Yb)
aabYbbccc (bY - Yb)
aaYbbbccc (aY - aa)
aaabbbcc

A, B, C változók
a, b, c konstansok
 $A = aAB$, $A = aC$, $CB = bCc$, $cB = Bc$, $C = bc$

A (A - aAB)
aAB (A - aC)
aaCB (CB - bCc)
aabCc (C - bc)
aabbcc

A (A - aAB)
aAB (A - aAB)
aaABB (A - aAB)
aaaABB (A - aC)
aaaaCBBB (CB - bCc)
aaaabCccBB (cB - Bc)
aaaabCBcB (cB - Bc)
aaaabCBBc (CB - bCc)
aaaabbCccBc (cB - Bc)
aaaabbCBcc (CB - bCc)
aaaabbbCccc (C - bc)
aaaabbbbcccc

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

A C89-e szabvány 1989-ben jött létre. Ezt a változatot sokan "ANSI C"-ként említik.

A C99 szabvány újításai:

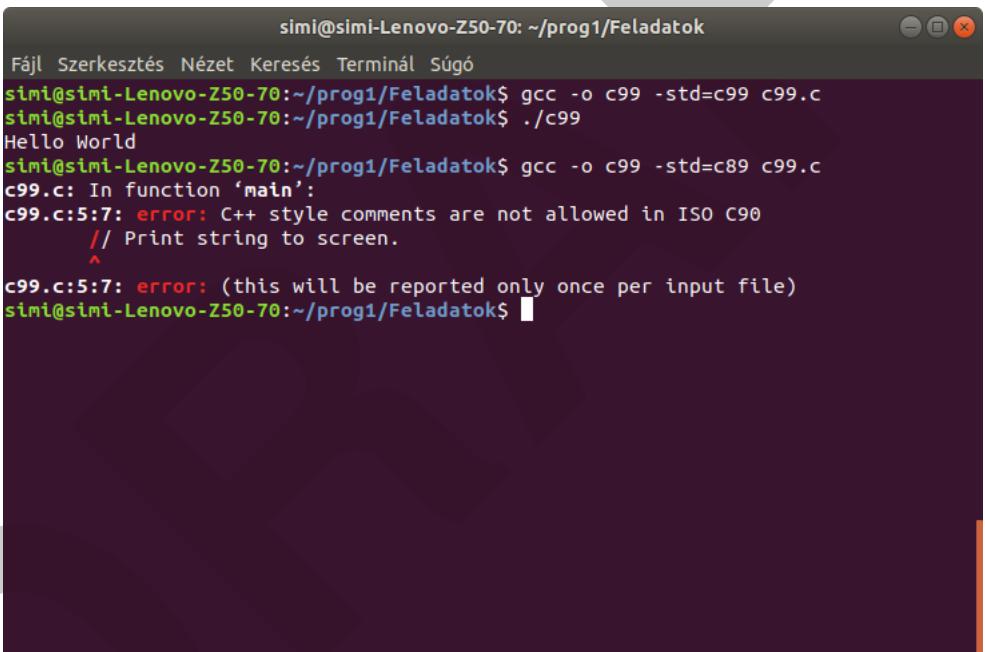
- Új adattípusok: **long long int, bool, complex**

- Egysoros komment //, amit a C++-ból vett át,
- Új függvények: **sprintf()**
- Új header fájlok: stdbool.h, complex.h, inttzpes.h, tgmath.h
- Változó definiálás helyére vonatkozó szabályok enyhítése

Forráskód linkje: [..//bhax/blob/master/Forrasok/3.Fejezet/c99.c](#)

```
#include <stdio.h>

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



The screenshot shows a terminal window with a dark background and light-colored text. The title bar reads "simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok". The terminal prompt is "simi@simi-Lenovo-Z50-70:~/prog1/Feladatok\$". The user runs the command "gcc -o c99 -std=c99 c99.c", followed by "./c99", which outputs "Hello World". Then, they run "gcc -o c99 -std=c89 c99.c", which results in an error message: "c99.c: In function 'main': c99.c:5:7: error: C++ style comments are not allowed in ISO C90 // Print string to screen. ^ c99.c:5:7: error: (this will be reported only once per input file)".

3.2. ábra.

Forráskód linkje: [..//bhax/blob/master/Forrasok/3.Fejezet/c992.c](#)

```
#include <stdio.h>
void main ()
{
    for(int i=0;i<5;++i)
        printf ("%d\n",i);
}
```



```
simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc -o c992 -std=c99 c992.c
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./c992
0
1
2
3
4
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc -o c992 -std=c89 c992.c
c992.c: In function 'main':
c992.c:4:2: error: 'for' loop initial declarations are only allowed in C99 or C11 mode
  for(int i=0;i<5;++i)
  ^
c992.c:4:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 to compile your code
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$
```

3.3. ábra.

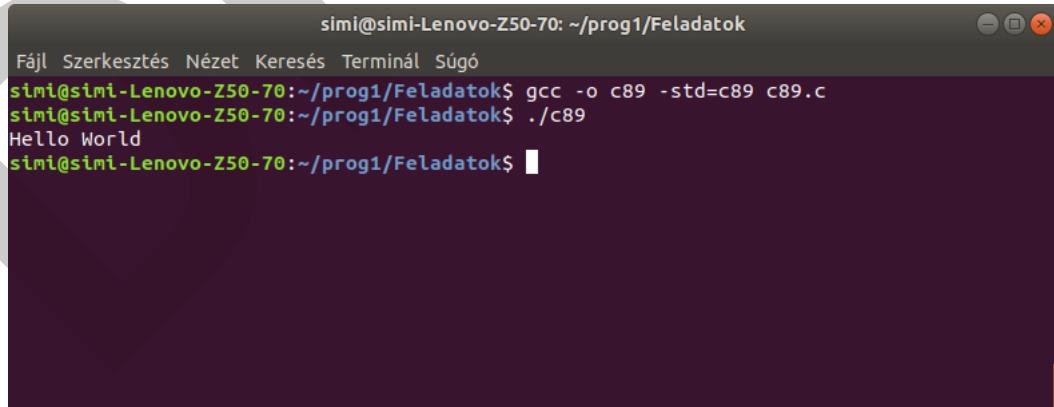
Mint a fenti képeken is látható a C99-es szabvány szerinti fordítás tudja értelmezni a "://" egysoros kommentet és a **for** ciklusba ágyazott deklarálást, míg a C89-es szabvány az első példában, a C++-os kommentek nem megengedettek hibát adja. A második példában ez ilyen deklarálás c89-es szabvány szerint nem megengedett.

Az előző két példát az alábbi módon átírva a c89-es szabvány szerinti fordítás is működni fog.

Forráskód linkje: [/bhax/blob/master/Forrasok/3.Fejezet/c89.c](#)

```
#include <stdio.h>

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



```
simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc -o c89 -std=c89 c89.c
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./c89
Hello World
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$
```

3.4. ábra.

Forráskód linkje: [./bhax/blob/master/Forrasok/3.Fejezet/c892.c](#)

```
#include <stdio.h>
void main ()
{
    int i=0;
for(i=0;i<5;++i)
    printf ("%d\n",i);
}
```

```
sími@ sími-Lenovo-Z50-70: ~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
sími@ sími-Lenovo-Z50-70:~/prog1/Feladatok$ gcc -o c892 -std=c89 c892.c
sími@ sími-Lenovo-Z50-70:~/prog1/Feladatok$ ./c892
0
1
2
3
4
sími@ sími-Lenovo-Z50-70:~/prog1/Feladatok$
```

3.5. ábra.

3.4. Saját lexikális elemző

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

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

Megoldás forrása:

Forráskód linkje: [..../bhax/blob/master/Forrasok/3.Fejezet/lexik.l](https://bhax.blob/master/Forrasok/3.Fejezet/lexik.l)

Az .l fájlt a **lex -o fájlnév.c fájlnév.l** fordítjuk c programra.

A c programot a **gcc fájlnév.c -o fájlnév -lfl** fordítjuk. A -lfl kapcsoló segít abban hogy a programban lévő függvényeket ismerje fel a fordító.

Majd a **./fájlnév** parancsal futtatjuk. A programot a **Ctrl+D**-vel le állítva megkpajuk a végeredményt.

```
% {
#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;
}
```

A program 3 részből áll, tekintsük ezeket részenként.

Az első rész:

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

A "%" közötti részt a lexer be teszi a c programba. Ebbem a részben deklaráljuk a realnumbert 0-ként, ebben számoljuk meg hogy hány zámot olvas be a program. A digit-et definiájuk és a mi esetünkben a [0-9] karktercsoportot adjuk meg neki.

Az második rész:

Itt határozzuk meg a forítási szabályokat.

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

```

A "*" azt jelenti hogy az előtte lévő, a mi esetünkben {digit}, számjegyből bármennyi lehet, akár 0 db is. A "." előtti "\" levédő abban segíti hogy a "." az tizedes pont legyen. Ez után a "+" azt mutatja hogy legalább egy új számjegy van. A (\.{digit}+) utánni kérdőjel segít eldöntani hogy van-e a szánka tizedes része vagy nincs.

Ha találunk ilyen számot növeljük a realnumber változót, majd kiiratjuk a számot a %s-el előbb sztringként, majd a %f-el számként az atof függvényel segítségével.

Az harmadik rész:

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

Az yylex()-el hívjuk a lexikális elemzést, majd a végén a printf-el kiiratom a talált valós számok számát.

3.5. Leetspeak

Lexelj össze egy l33t cipher!

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

Megoldás forrása:

A leetspeak egy olyan írásmód ami a latin ABC betüit ASCII karakterekkel helyettesíti.

Forráskód linkje: [./bhax/blob/master/Forrasok/3.Fejezet/leed.l](#)

Fordítás(létrehozzuk az l fájlból a c-t): lex -o fájlnév.c fájlnév.l

Fordítás(fordítjuk a c programot): gcc fájlnév.c -o fájlnév -lfl.

Futtatás:./fájlnév

```
% {
    #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', {"i", "1", "|", "!"}}, {'j', {"j", "7", "_|", "_/"}}, {'k', {"k", "|<", "1<", "|{"}}, {'l', {"l", "1", "|", "|_"}}, {'m', {"m", "44", "(V)", "|\\//|"}}, {'n', {"n", "|\\|", "/\\/", "/V"}}, {'o', {"o", "0", "()", "[]"}}, {'p', {"p", "/o", "|D", "|o"}}, {'q', {"q", "9", "O_", "(,)"}} , {'r', {"r", "12", "12", "|2"}}, {'s', {"s", "5", "$", "$"}}, {'t', {"t", "7", "7", "'|'"}} , {'u', {"u", "|_|", "(_)", "[_]"}}, {'v', {"v", "\\\/", "\\\/", "\\\/"}} , {'w', {"w", "vv", "\\\/\\", "(/\\)"}},
```

```
{'x', {"x", "%", " ") ("") ("} },
{'y', {"y", "", "", ""} },
{'z', {"z", "2", "7_", ">_"} },

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

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

%}
%%%
. {

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

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

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

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

            found = 1;
            break;
        }
    }

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

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

Tanulságok, tapasztalatok, magyarázat...

3.6. ábra.

A program ugyan úgy három részből áll mint a lexikális elemzőfeladat.

A program első részében deklaráljuk a szükségs header fájlokat. Deklaráljuk L337SIZE-t ami meghatározza az input hosszát, majd a ciper strukturát is, ami megkepja a karaktert és a négy lehetséges sztringet amit megkaphat. A l337d1c7 nevű tömb fogja tárolni a latin betükből álló bemenetet és a hozzájuk tartozó lehetőségeket.

```
% {
#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  
};  
  
}%
```

A második részben a for cikluson belüli i változóval végig megyünk az inputon, majd minden betűt kis betűre alakít és hozzá rendel egy random számot 0 és 100 között, amit az r változóban tárolunk. Ez a véletlenszerűen generált szám lesz az ami alapján eldől, hogy a négy lehetséges csere érték közül melyiket helyettesítse. Ha a szám kisebb mint 91 akkor az első, ha kisebb mint 95 a második, ha kisebb mint 98 akkor a harmadik érték, ha a szám 99 vagy 100 akkor a negyedik lehetőséget válaaztj és írja ki.

```
%%  
. {  
  
    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 program utolsó részében az yylex()-el indítjunk függvényhívást és az input átalakítását.

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

3.6. A források olvasása

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

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

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

Bugok

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

i.

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

ii. A for cikluson addig megy mig az i változó kisebb mint min 5.

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

iii. A for ciklus addig megy mig az i változó kisebb mint min 5, ugyan úgy mint az előző estben.

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

iv. A for ciklus addig megy mig az i változó kisebb mint min 5, és a tömb i-edik eleme megkapja i értékét.

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

v. A for ciklus addig megy mig az i változó kisebb mint az n értéke és a d+1 általi mutató egyenlő az s+1 általi mutatóval.

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

vi. Először kiiratjuk az f értéke az "a" és "a+1" pontban, majd ugyan ez a függvény értéke az "a+1" és "a" pontban.

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

vii. Először kiiratjuk az f értéke "a" pontban, majd az "a" értékekét.

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

viii. Először kiiratjuk az f értéke az "a" számnak a lefoglalt memóriacímből, majd az "a" értékekét.

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

Megoldás forrása:

Megoldás videó:

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

Minden x esetén létezik y, úgy hogy x kisebb mint y, és y prím.

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

Minden x esetén létezik y, úgy hogy x kisebb mint y, y prím szám és $y+2$ is.

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

Létezik y minden x esetén, x prím, ha x kisebb mint y.

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

létezik y minden x esetén, ha y kisebb mint x akkor x nem prím.

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

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

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

Mit vezetnek be a programba a következő nevek?

- egész

```
int a;
```

- egészre mutató mutató

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

- egész referenciaja

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

- egészek tömbje

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

- egészek tömbjének referenciaja (nem az első elemé)

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

- egészre mutató mutatók tömbje

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

- egészre mutató mutatót visszaadó függvény

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

- egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

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

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

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

Megoldás video:

Megoldás forrása:

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összahasonlítása azt mutatja meg, hogy miért érdemes a **typedef** használata: [.../bhax/blob/master/Forrasok/3.Fejezet/fptr.c](#), [.../bhax/blob/master/Forrasok/3.Fejezet/fptr2.c](#).

```
#include <stdio.h>

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

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

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

```
int
main ()
{
    int (*f) (int, int);

    f = sum;

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

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

    g = sumormul;

    f = *g (42);

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

    return 0;
}
```

```
#include <stdio.h>

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

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

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

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

int
main ()
{
```

```
F f = sum;  
printf ("%d\n", f (2, 3));  
G g = sumormul;  
f = *g (42);  
printf ("%d\n", f (2, 3));  
return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

DRAFT

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

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

Megoldás forrása:

Forráskód linkje: [./bhax/blob/master/Forrasok/4.%20Fejezet/tm.c](https://bhanu.s3.amazonaws.com/banax/blob/master/Forrasok/4.%20Fejezet/tm.c)

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

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

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

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

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

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

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

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

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

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

free (tm);

return 0;
}
```

A program elején deklaráljuk az `nr` változót, ami az 5-ös értéket kapja, ez lesz a mátrix sorainak a száma. Ez után deklaráljuk a `tm` változót, aminek lefoglaljuk a 8 bájtnyi táthelyet, majd ki iratjuk a memóriacímét. A `malloc` függvény memóriát foglal és egy pontert ad vissza amikor mondjuk meg mire mutasson. A `malloc` a `double *` helyigényét adja vissza ami 40 bájt. (`nr * sizeof (double *)`) Egy `if`-fel megnézzük hogy le fogalma-e a helyet neki, ha igen a program tovább ha `NULL`-t kap program le áll.

A első két for ciklusokban létrehozzuk a háromszögmátrixot, a harmadikkal ki iratjuk.

A program utolsós részében a `free` függvényel felszabadítjuk `tm` által foglalt memóriát.

Fordítás: **gcc fájlnév.c -o fájlnév**

Futtatás: **./fájlnév**

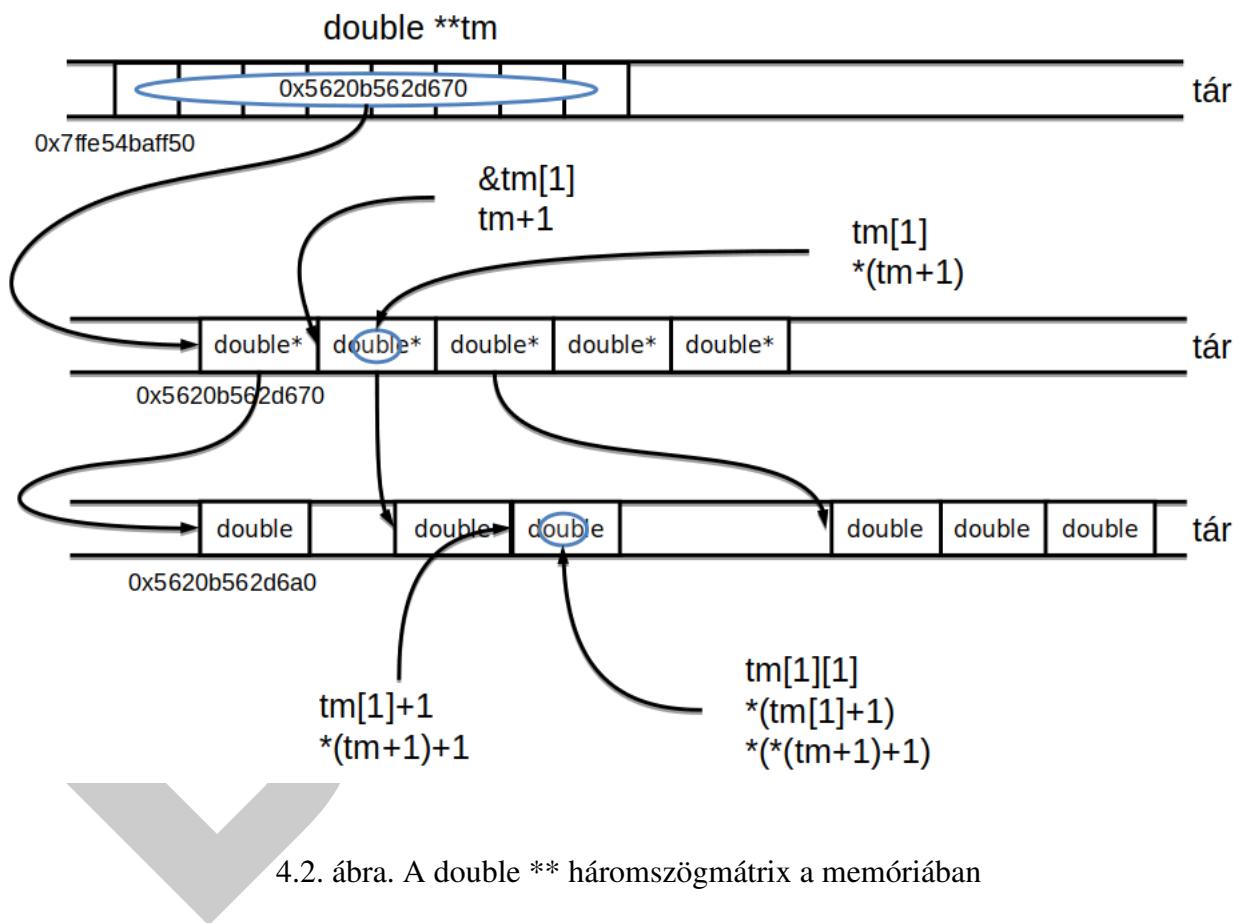
simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok

Fájl Szerkesztés Nézet Keresés Terminál Súgó

```
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc tm.c -o tm
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./tm
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,
```

simi@simi-Lenovo-Z50-70:~/prog1/Feladatok\$

4.1. ábra.



4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Forráskód linkje: [..../bhax/blob/master/Forrasok/4.%20Fejezet/exortitkosito.c](#)

Az exor titkosító az exor művelet segítségevel titkosít. A program a beolvasott szöveg és a kulcs bitjein hajtja végre a xor műveletet. Így az olvasható szövegből egy sor értelmezhetetlen szimbólum kombinációt kapunk titkos szövegnek.

```
#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);
    }
}
```

Fordítás: **gcc fájlnév.c -o fájlnév**

Futtatás: **./fájlnév kulcs <tiszta.txt> titkos.szöveg** (a kulcs bármi lehet)

```

Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc exortitkosito.c -o exortitkosito
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./exortitkosito 56789012 <tiszta.txt> titkos.sz
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ more titkos.sz
t[&]JVSTAFL[&][&]C[&]EM[&]>\[&]YOVV[&]SAOBEYRSX[&] ]M[&]UL[&]JWVCSX[&]KU\F[&]R_&K[&][BVRBP[&]M[&]@T8G[&]
[P[&]A[&]RV]CKWG[&]CQ
K[&]K[&]KAT^&[&]Y[&]_YG[&]LXRS[&]J^IP^&&D[&]UUDD[&]KD[&]ZXSWUZD\^&WJD[&]RO[&]XZ[&]A[&]2[&]C]RDVUL^Z@T[&]X[&]SE
&&JV^&[&]WPA[&]ZX^LV
B^&[&]SPA[&]KG[&]RKCD[&]WSY[&]HPQLRS[&]VBC[&]E[&]M[&]XJ[&]GSYWZQ[&]PBC]UU[&]Y[_VD[&]LD_&&]G[&]T^
LTRV[&]ACYS[&]T
J\ ]&&UYB[&]D[&]JX\AW\LP[P[&]PAJ[&]UP]RL[&]8AWD[&]D[&]FY\YM[&]YY\SV[PF[&]WVD[&]YA\[TF[&]T[&]Z[&]BUW^SD
[&]P[&]SQWKT[&]A\
BS[&]A[&]BLMQEXG[&][Z]G[&]2PCK[B[&]C[&]W[YTY[&]Z[&]^&RYC[&]YKAR[&]TQQEG[&]J[&]EWZUBAO[&]TU[K[&]VTX
X^&[&]CL[&]P[&]CPTX
[&]u\&[&]P[&]GBRJ[&]WAXRS[&]K[&]ZRR[&]JVCSX[&]B_KPLRL\&[&]VJX^RAFYE[&]BVGXSYLL]P[&]_Z[&]Z^K[&
:G[&]ZWC[&]FPD[&]UU
EW[&]K[&]TDVU[&]EWG[&]LXT[&]F[&]WWI^&&C[&]C[&]GW[&]C_]U[&]NGSZL[&]B[&]YS[&]X\&&Y^VL[&]RT@UE[&]W[&]P^P\_
```

4.3. ábra.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#exor_titkosito

Forráskód linkje: <..//bhax/blob/master/Forrasok/4.%20Fejezet/exortitkosito.java>

A következő java program egy titkosító és törő is egyben. A program ugyan úgy mint az előző C programban is a kizártó vagy művelet segítségével titkosít, itt jön létre az szimbólumokból álló szöveg és tör, mikor a szöveget éjra olvashatóvá alakítja.

A program olvassa a bemenetet bitenként majd a neki megfelelő kulcs bittel végrehajtja a kizártó vagy műveletet, majd tovább lépteti a kulcsIndex-et. Az eredmény a buffer tömbjébe kerül, ez lesz kiiratva program végén.

Ahhoz hogy titkos szöveget feltörjük újra lefuttatjuk a programot, ekkor a titkos szövegbeli szimbólumok bitjaiet és a neki megfelelő kulcs bitet ossze xorozva a szöveg "életre kel", vagyis a latin ABC betűit kapjuk vissza.

```

public class exortitkosito {

    public exortitkosito(String kulcssSzöveg,
        java.io.InputStream bejövőCsatorna,
        java.io.OutputStream kimenőCsatorna)
        throws java.io.IOException {

        byte [] kulcs = kulcssSzöveg.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 exortitkosito(args[0], System.in, System.out);  
  
    } catch(java.io.IOException e) {  
  
        e.printStackTrace();  
  
    }  
  
}  
}
```

A program futtatása

javac fájlnév.java

java fájlnév kulcs > titkositott.txt - a titkosítandó szöveg beolvasása, a kulcs bármi lehet, egyeznie kell majd a töréshez használt kulccsal

more titkositott.txt - a titkos szöveg

java fájlnév kulcs < titkositott.txt - dekódoljuk a titkosított szöveget, visszakapva az eredetit

```
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ javac exortitkosito.java
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ java exortitkosito alma > titkositott.sz
Ez titkositva lesz!
Titkositva, bizony!
^Csimi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ more titkositott.sz
[Binary data]
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ java exortitkosito alma < titkositott.sz
Ez titkositva lesz!
Titkositva, bizony!
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$
```

4.4. ábra.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Forráskód linkje: [./bhax/blob/master/Forrasok/4.%20Fejezet/exortoro.c](#)

```
#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 szoveg valszag 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;
}
```

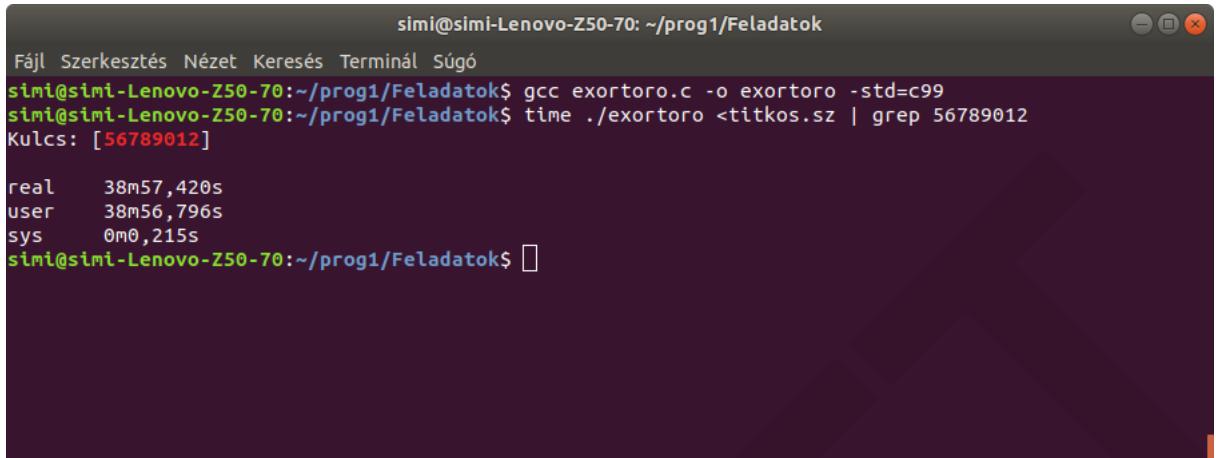
A programban a tiszta_lehet függvény megtáprálja megpróbálja megtippelni, hogy a kapott szöveg tartalmazza-e a leggyakoribb magyar szavakat (hogy, nem, az, ha), mivel ezek használata nélkül nem igazán alkotható értelmes magyar nyelvű szöveg. Az átlag szóhossz vizsgálatával csökkentjük a potenciális töréseket.

Az exor_toro függvény a benne lévő exor függvény segítségével elvégzi a kizárt vagyos műveletet a titkos szöveg és kulcs között, kihasználva az exor művelet azon tulajdonságát hogy ha valamit kétszer exorozunk az eredmény egyenlő az eredeti értékkel. E tulajdonságot felhasználva állítja vissza az erdei szöveget.

A mainnel belül a program elő állítja a kulcsot majd elvégzi az exor műveletet.

Fordítás: **gcc fájlnév.c -o fájlnév -std=c99**

Futtatás: **./fájlnév <titkos.szöveg |grep kulcs** (a kulcsnak meg kell egyeznie a titkosítási kulccsal)



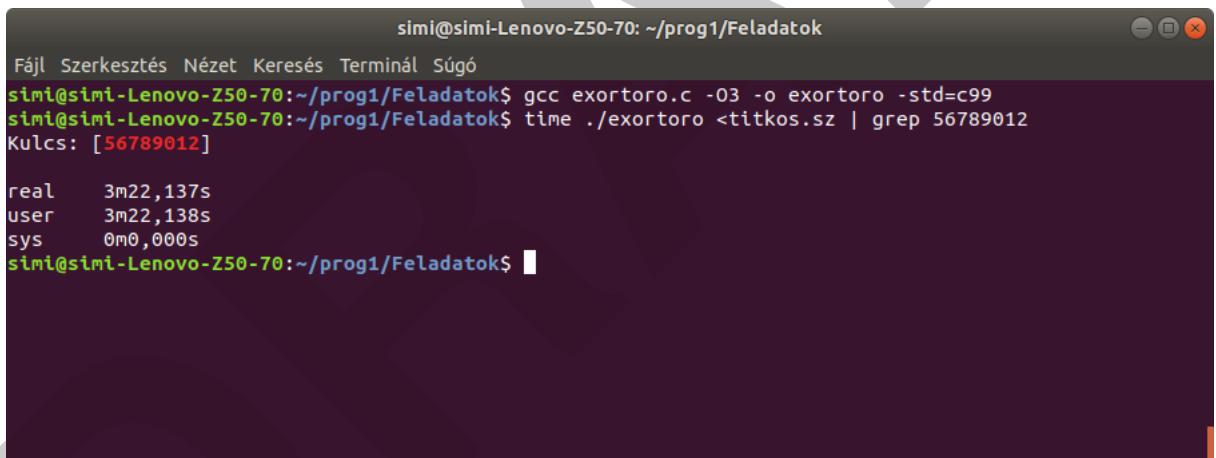
```
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc exortoro.c -o exortoro -std=c99
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ time ./exortoro <titkos.sz | grep 56789012
Kulcs: [56789012]

real    38m57,420s
user    38m56,796s
sys     0m0,215s
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ 
```

4.5. ábra.

Fordítás: **gcc fájlnév.c -O3 -o fájlnév -std=c99**

Futtatás: **./fájlnév <titkos.szöveg |grep 56789012**



```
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc exortoro.c -O3 -o exortoro -std=c99
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ time ./exortoro <titkos.sz | grep 56789012
Kulcs: [56789012]

real    3m22,137s
user    3m22,138s
sys     0m0,000s
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ 
```

4.6. ábra.

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Forráskód linkje: <..//bhax/blob/master/Forrasok/4.%20Fejezet/nn.r>

Az alábbi feladatokban megtanítjuk egy neuronnak az OR, AND és EXOR műveleteket, mindezeket R-ben.

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
# https://youtu.be/Koyw6IH5ScQ

library(neuralnet)

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

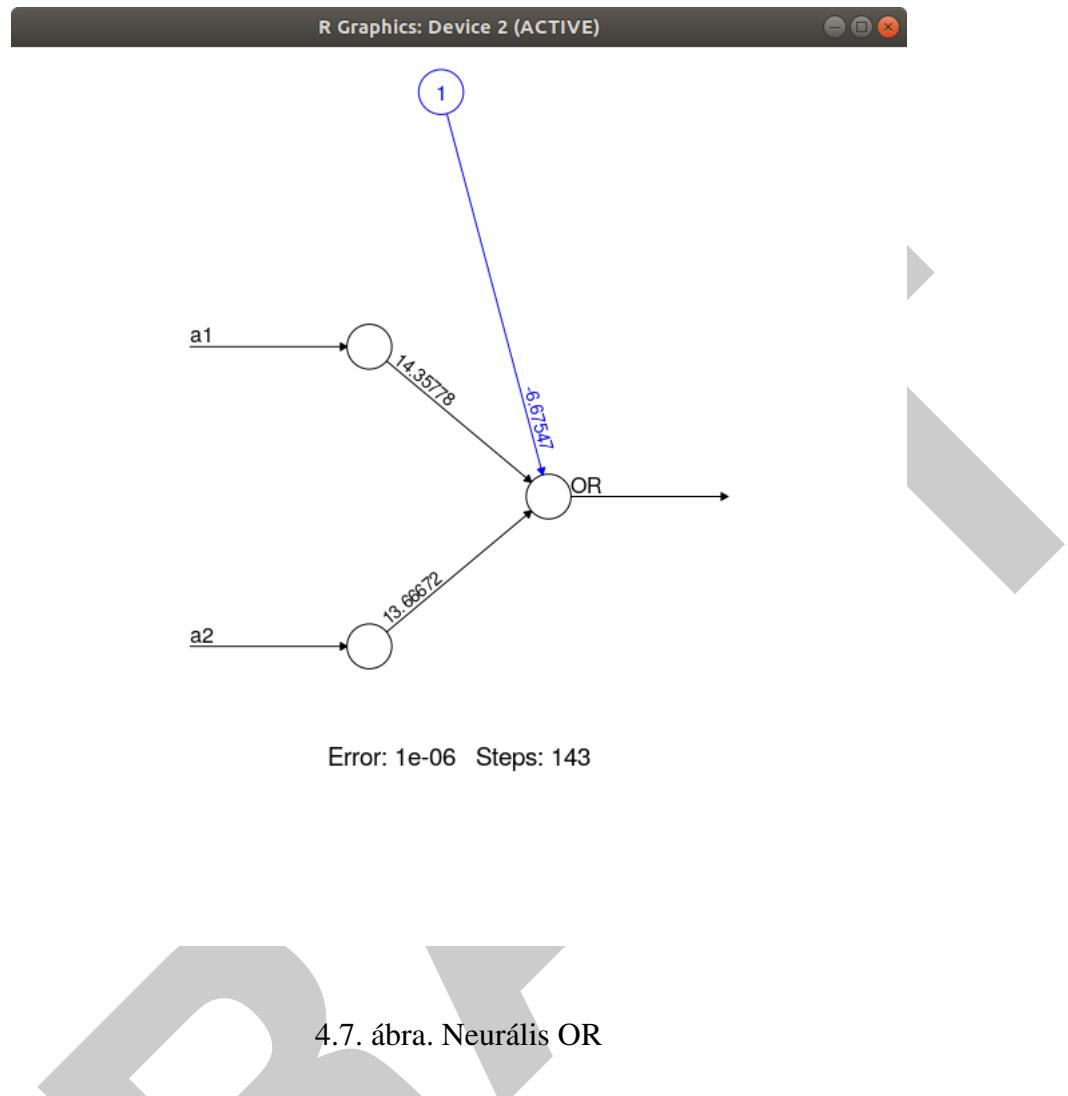
or.data <- data.frame(a1, a2, OR)

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

plot(nn.or)

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

OR, azaz ligikai megengedő vagy. Megadunk két értéket(a1,a2) és azt hogy a vagy műveletet végre hajtva milyen eredmény kell ki jöjjön(OR). Az eredményt 143 lépésből kapjuk.



```

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)

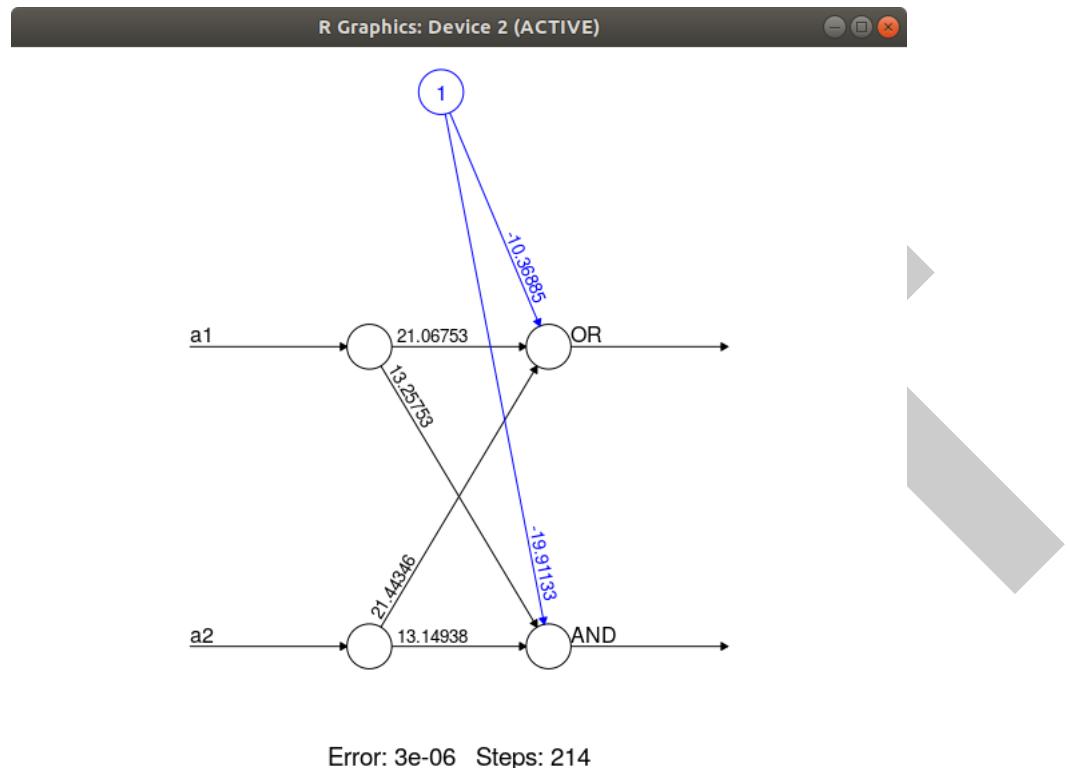
nn.operand <- neuralnet(OR+AND~a1+a2, operand.data, hidden=0, linear.output= FALSE,
                         stepmax = 1e+07, threshold = 0.000001)

plot(nn.operand)

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

```

Ha az OR mellett felhasználjuk azt AND, vagyis logikai és műveletet, ugyan úgy megadjuk a két értéket és azt hogy milyen eredményt akarunk kapni. Az eredményt 214 lépésben kapjuk.



4.8. ábra. Neurális AND

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

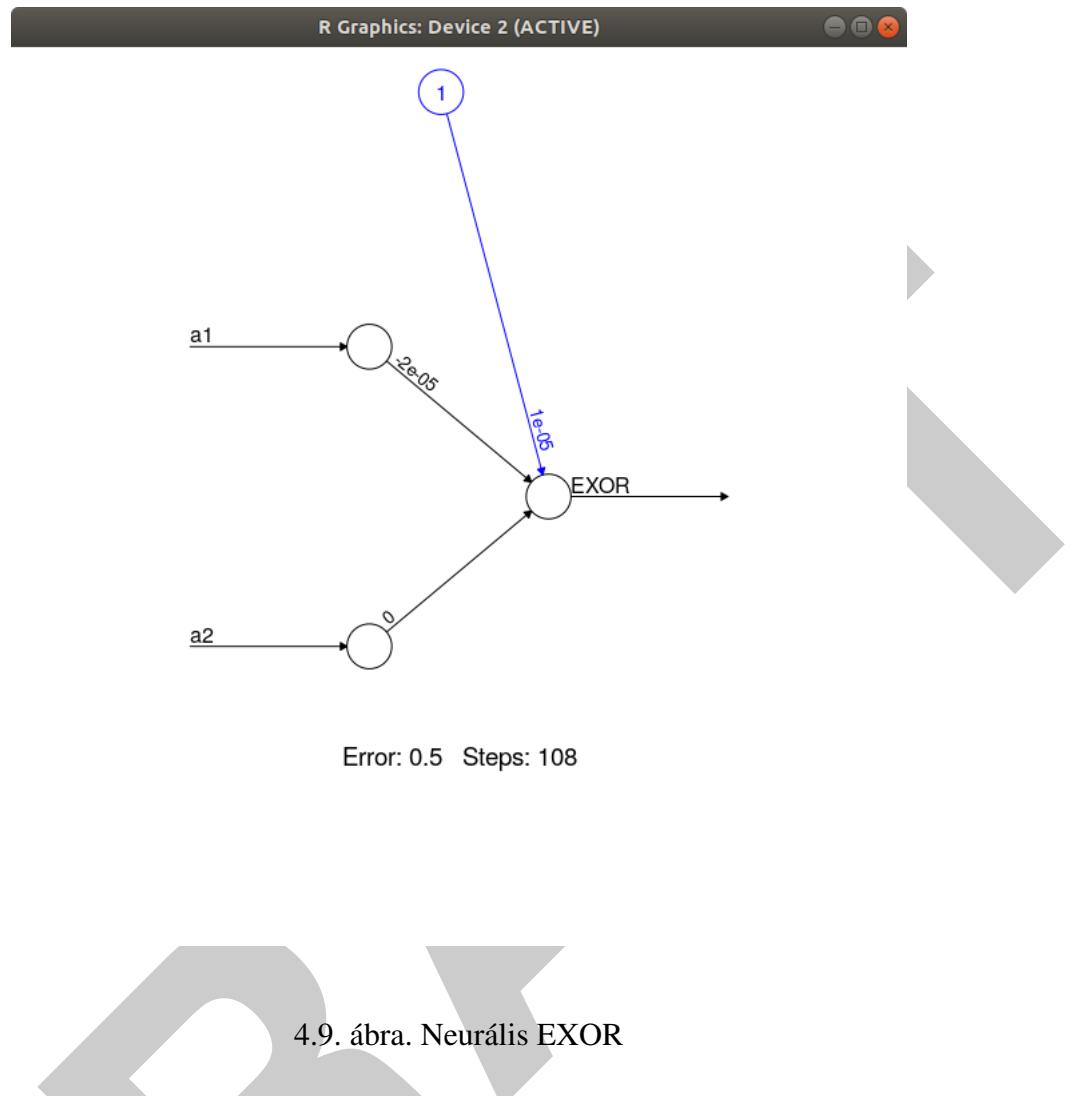
exor.data <- data.frame(a1, a2, EXOR)

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, azaz logikai kizáró vagy. Ugyan úgy mint az előző két feladatban itt is megadjuk az értékeket, és hibát tapasztunk, mivel 108 lépés alatt, 0,5 a hibák aránya. Ezért szükség van rejtett rétegekre, ezt mutatja a következő példa.



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

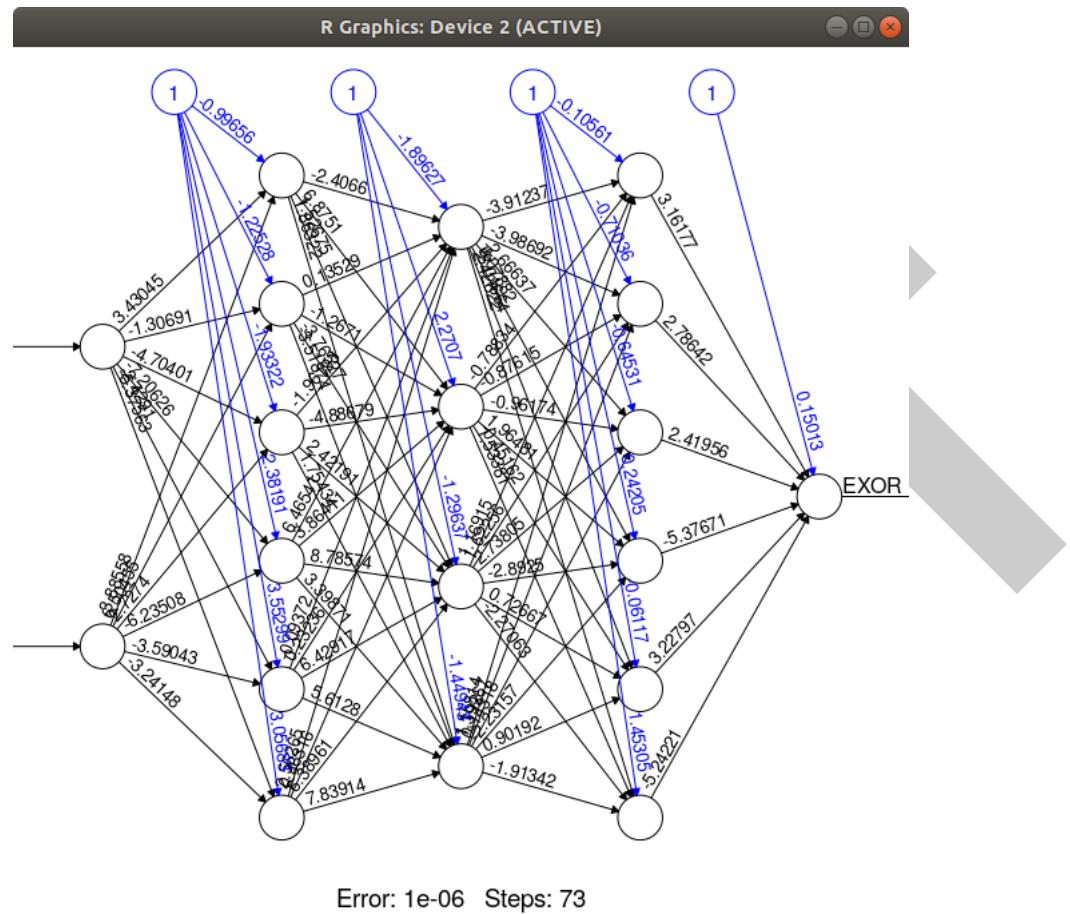
exor.data <- data.frame(a1, a2, EXOR)

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)

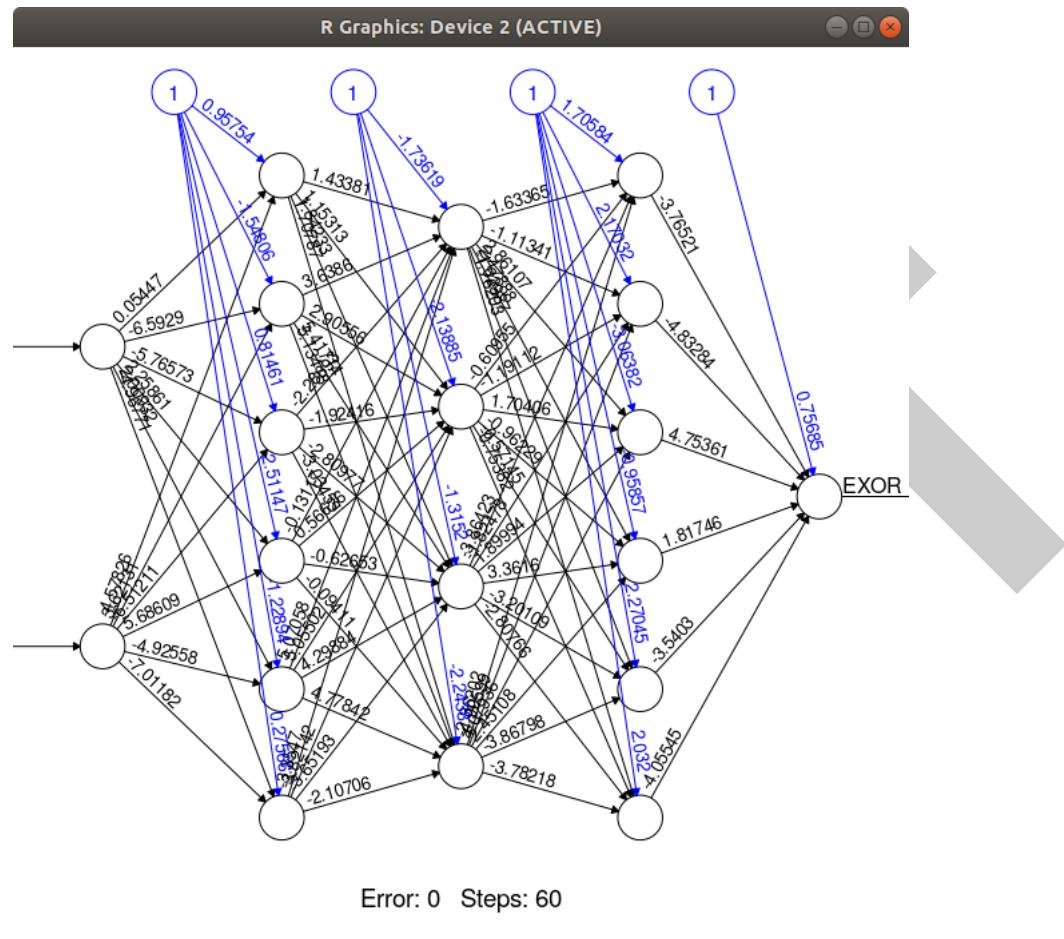
compute(nn.exor, exor.data[,1:2])
```

A hozzá adott rétegek segítségével 73 lépés alatt kapn k eredményt.



4.10. ábra. Neurális EXOR

Mivel ezek szimulációk megesik olyan is hogy 60 lépés alatt kaptunk helyes eredményt, 0 hibával.



4.11. ábra. Neurális EXOR

4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

Forráskód linkje: [..../bhax/blob/master/Forrasok/4.%20Fejezet/mlp.hpp](https://github.com/bhax/blob/master/Forrasok/4.%20Fejezet/mlp.hpp)

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

Futtatás **./perc mandel.png**

A perceptron egy algoritmus a gépi tanulásban.

A main függvényen belül a size változóban tároljuk a megadott kép méretét, majd létrehozunk egy új perceptron aminek 3 rétege lesz, az első a kép pixeleinek száma. A második réteg 256, a harmadik 1. A perceptron a pixelek piros árnyalatait tárolja. A perceptron a számolások elvégzése után a standart kimenetre írja az eredményt.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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

Forráskód linkje: [./bhax/blob/master/Forrasok/5.%20Fejezet/mandelpngt.c++](https://github.com/bhax/blob/master/Forrasok/5.%20Fejezet/mandelpngt.c++)

Fordítás: **g++ mandelpngt.c++ -lpng16 -O3 -o mandelpngt**

Futtatás: **./mandelpngt t.png**

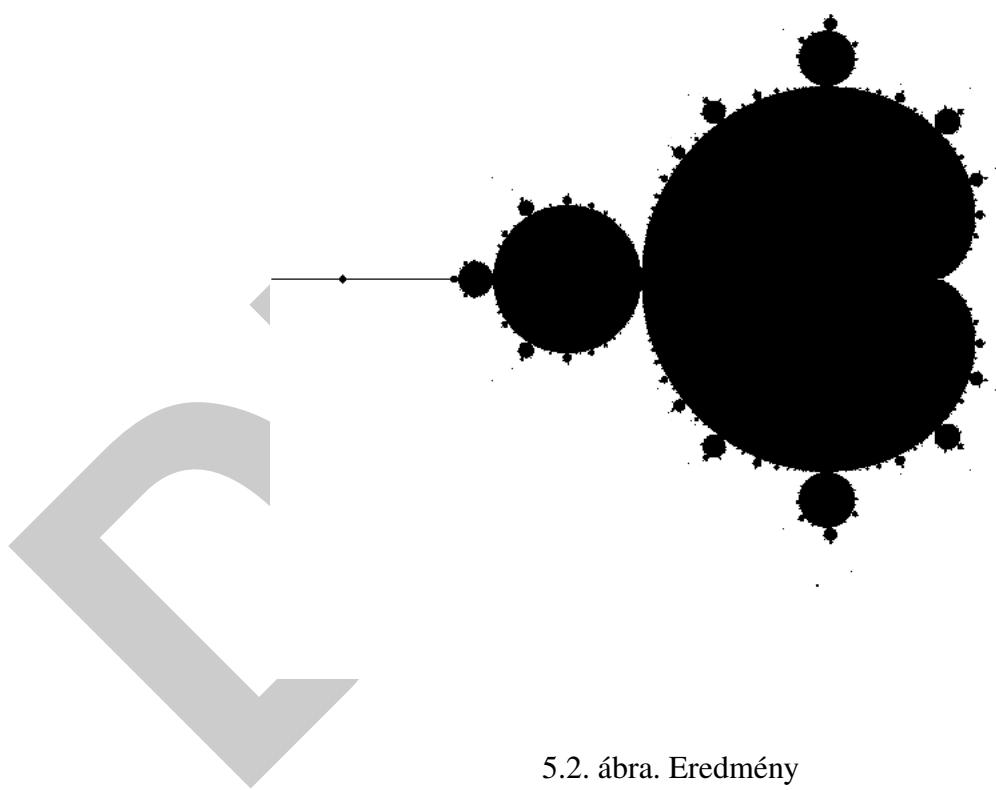
Tutoráltam: Butcován György

A program futtatáshoz szükségünk van a png++ headerre, amit terminálból a **sudo apt-get install libpng++-dev** parancs segítségével tudunk telepíteni.

A program futtásakor megadott, ebben az esetben t.png fájlnév lesz az amibe beírjuk a kapott eredményt. Létrehozunk egy j magasságú és k szélességű ponthalmazt. majd vizsgáljuk c változó eleme a Mandelbrot halmaznak. A mandel függvényben számolunk egy átlag időt is(amit ki is iratunk), hogy mennyi időbe telik létrehozni a Mandelbrot halmazt. A main függvényben az if segítségével megnézzük hogy a futtatás során megadtunk-e egy fájlnevet, ha ez hiányzik a "Hasznalat: ./mandelpng fajlnev" hiba üzenetet kapjuk. Ha a képalkotás befejeződött megkapjuk a "mentve" üzenetet.

```
simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok/mandelbrot
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok/mandelbrot$ g++ mandelpngt.c++ -lpng16 -O3 -o mandelpngt
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok/mandelbrot$ ./mandelpng t.png
980
9.80213 sec
t.png mentve
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok/mandelbrot$
```

5.1. ábra. Eltet idő



5.2. ábra. Eredmény

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

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rátcsot és kiszámoljuk, hogy a rácspontjai mely komplex számoknak

felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1} = z_n^2 + c$, ($0 \leq n$) képlet alapján úgy, hogy a c az éppen vizsgált rácspont. A z_0 az origó. Alkalmazva a képletet a

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

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

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

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

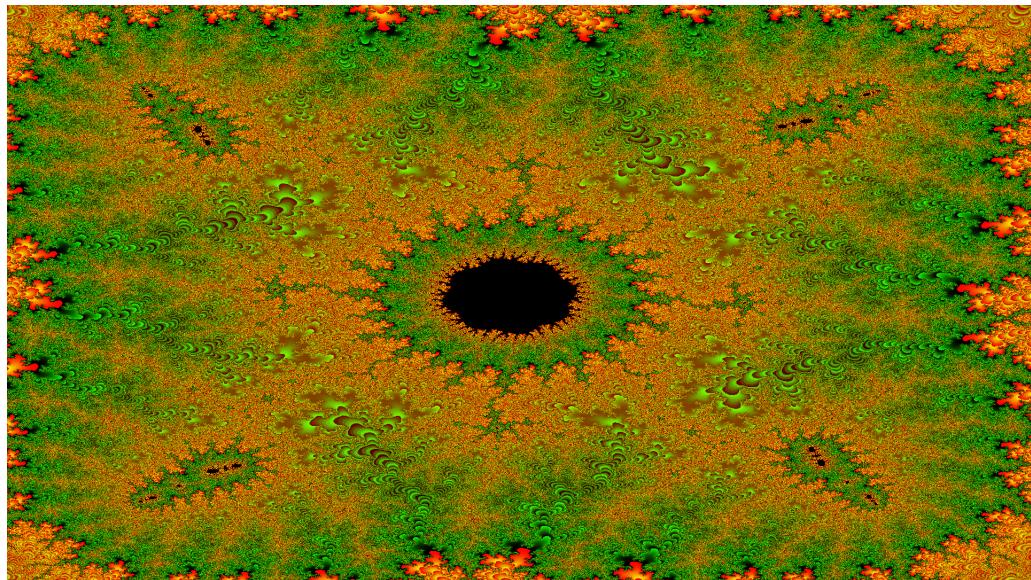
Megoldás videó: <https://youtu.be/gvaqijHIRUs>

Megoldás forrása:

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhax/attention_raising/Mandelbrot/3.1.2.cpp](#) nevű állománya.

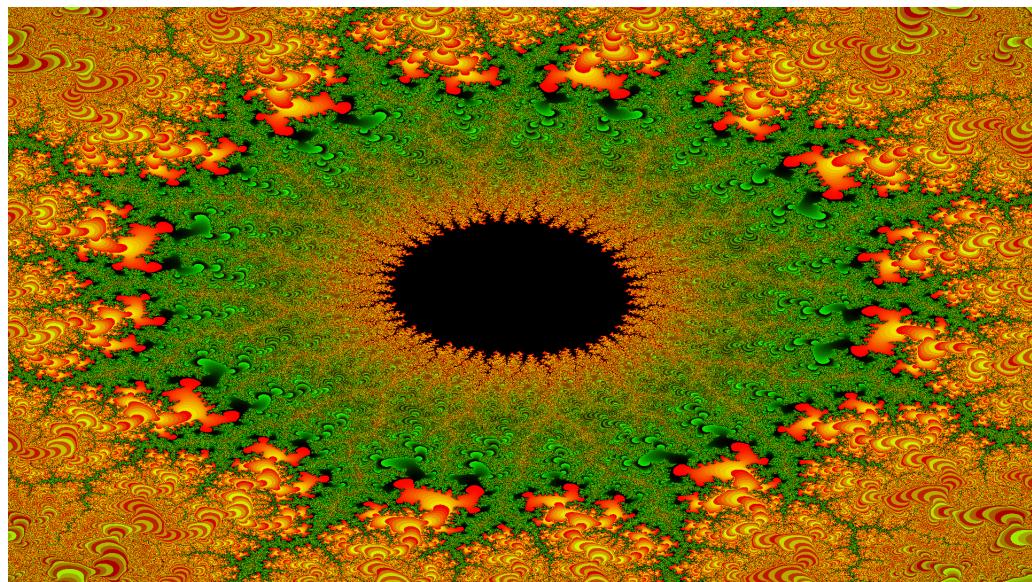
Forráskód linkje: [./bhax/blob/master/Forrasok/5.%20Fejezet/3.1.2.cpp](#)

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ←
// -0.01947381057309366392260585598705802112818 ←
// -0.0194738105725413418456426484226540196687 ←
// 0.7985057569338268601555341774655971676111 ←
// 0.798505756934379196110285192844457924366
```



5.3. ábra. A Mandelbrot halmaz a komplex síkon

```
// ./3.1.2 mandel.png 1920 1080 1020 ←
0.4127655418209589255340574709407519549131 ←
0.4127655418245818053080142817634623497725 ←
0.2135387051768746491386963270997512154281 ←
0.2135387051804975289126531379224616102874
// Nyomtatás:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1 --left-footer=" ←
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
```



5.4. ábra. A Mandelbrot halmaz a komplex síkon

Includeoljuk complex headert, így már tudunk komplex számokkal is dolgozni.

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

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

int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
```

```
double a = -1.9;
double b = 0.7;
double c = -1.3;
double d = 1.3;

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

Az if fel vizsgáljuk, hogy a program futtatásához megadtuk-e a megfelelő számú argumentumot. Ha igen akkor a változók(szelesség, magasság,iteraciosHatar, a, b, c, d) rendre átveszik a nekik adott adatot. Ha a program nem kapja meg a futtatáshoz szükséges 9 argumentumot a helyes futtatáshoz ad segítséget. A program további része ugyan azon számításokat hajtja végre mint az előző feladat, majd megszine pluszba futás során %-ossan jelezzük hogy épp hol tartunk a képalkotásba..

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

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

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

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

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

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

        reC = a + k * dx;
```

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

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

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

    ++iteracio;
}

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

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

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

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbqRzY76E>

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

Forráskód linkje: [./bhax/blob/master/Forrasok/5.%20Fejezet/3.1.3.cpp](#)

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

A különbség a **Mandelbrot halmaz** és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

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

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

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

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

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

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

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

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

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
```

```
// Futtatas:  
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10  
// Nyomtatás:  
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ↫  
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↫  
color  
//  
// BHAX Biomorphs  
// Copyright (C) 2019  
// Norbert Batfai, batfai.norbert@inf.unideb.hu  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.  
//  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <https://www.gnu.org/licenses/>.  
//  
// Version history  
//  
// https://youtu.be/IJMbqRzY76E  
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↫  
Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf  
//  
  
#include <iostream>  
#include "png++/png.hpp"  
#include <complex>  
  
int  
main ( int argc, char *argv[] )  
{  
  
    int szelesseg = 1920;  
    int magassag = 1080;  
    int iteraciosHatar = 255;  
    double xmin = -1.9;  
    double xmax = 0.7;  
    double ymin = -1.3;  
    double ymax = 1.3;  
    double reC = .285, imC = 0;  
    double R = 10.0;
```

Ha a megadott argumentum száma 12 akkor az, atof parancs segítségével a változóknak átadjuk a nekik

megfelelő értékeket a parancssori argumentumból

```
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] );

}
```

Ha a megadott argumentum száma 12 nem akkor hibaüzenetet küldünk a felhasználónak a helyes futtatásról

```
else
{
    std::cout << "Használat: ./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 << "Számítás\n";
```

A létrehozott képnek végig megyünk sorain és oszlopain, és ezekben a pontokban a cc változónak megfelelő színüre festjük a pixelt a képen.

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

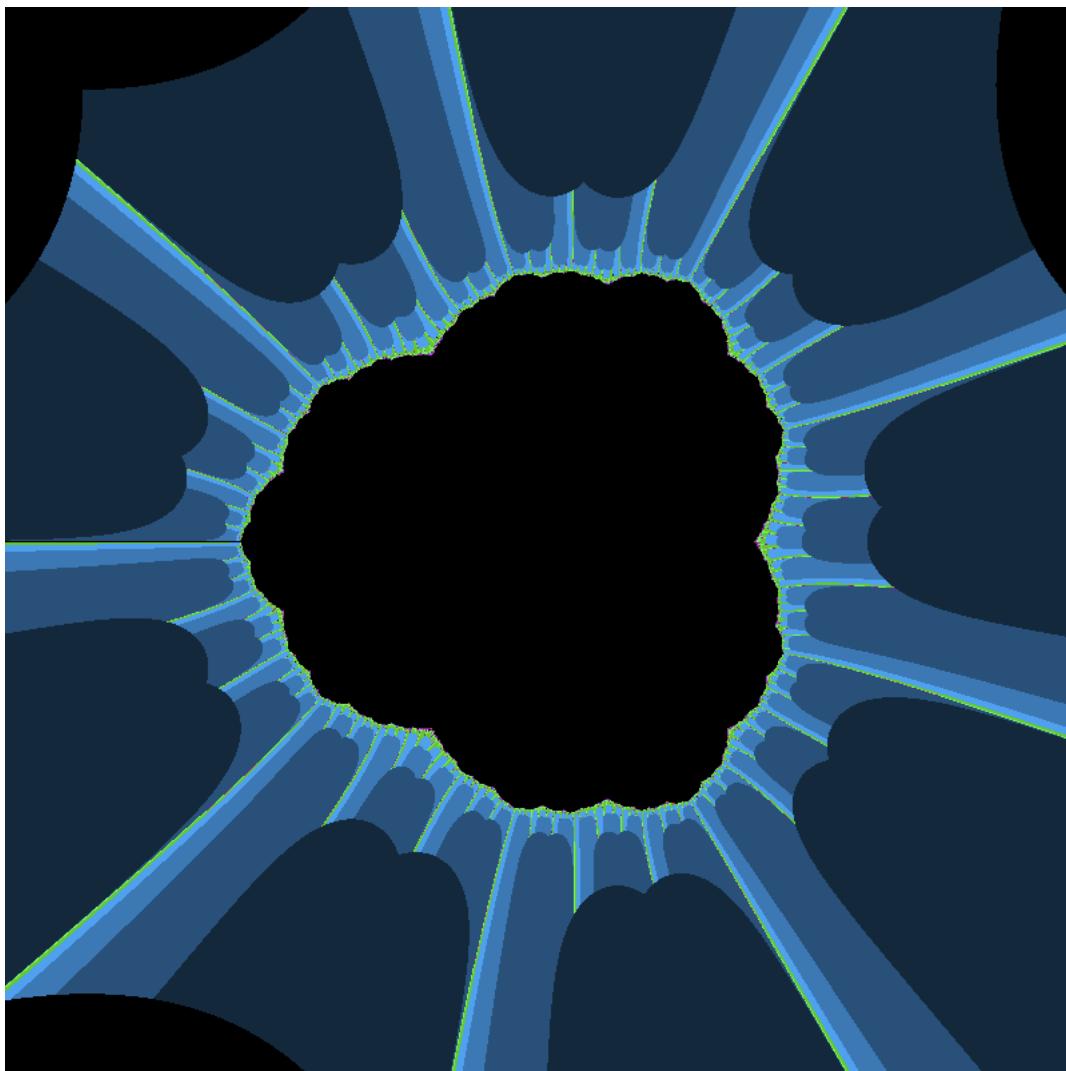
    for ( int x = 0; x < szelesseg; ++x )

        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;  
}
```

Ha sikeresen lefut a program kiiratjuk a kép nevét és hogy sikeresen megalkottuk a megadott értékek alapján a képet.

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



5.5. ábra. Eredmény

A program nagyban hasonlít a fenti kettőhöz, kezdetben vizsgáljuk hogy megvan e a megfelelő számú argumentum, majd végig megyünk a ponthalmazunk sorain és oszlopain, megszínezzük a pixeleket, majd kiiratjuk , jelen esetben a "bmorf.png" fájlba. Egyetlen különböző a már feljebb is említett iteráció beli eltérés, míg az előző két programban a c -nek változott az értéke, ebben a programban a cc egy konstans. mivel

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu nevű állománya.

Forráskód linkje: [..../bhax/blob/master/Forrasok/5.%20Fejezet/mandelpngc_60x60_100.cu](#)

Valami technikai ok miatt nem sikerült munkára bírni a garfikus kártyát így felhasznánák egy passzolási lehetőséget.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás videó: Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal.

Megoldás forrása:

Forráskód linkje: [./bhax/blob/master/Forrasok/5.%20Fejezet/mandelbrotnagyito.cpp](#)

```
// Verzio: 3.1.1.cpp
// Forditas:
// g++ 3.1.1.cpp `libpng-config --ldflags` -O3 -o 3.1.1
// Futtatas:
// ./3.1.1 mandel.png 1920 1080 2040 ←
// -0.0194738105730936639226058559870580211281 ←
// -0.0194738105725413418456426484226540196687 ←
// 0.7985057569338268601555341774655971676111 ←
// 0.798505756934379196110285192844457924366
// ./3.1.1 mandel.png 1920 1080 1020 ←
// 0.41276554182095892553405747094075195491310.41276554182458180530801428176346234
// 0.2135387051768746491386963270997512154281 ←
// 0.2135387051804975289126531379224616102874

#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;
```

Meghatározzuk a kéünk szélességét, magasságát és az iterációs határt. Az atof parancs segítségével az a, b, c, d változóknak átadjuk az argumentumban megadott értékeket.

```
if ( argc == 9 )
{
```

```
szelesseg = atoi ( argv[2] );
magassag = atoi ( argv[3] );
iteraciosHatar = atoi ( argv[4] );
a = atof ( argv[5] );
b = atof ( argv[6] );
c = atof ( argv[7] );
d = atof ( argv[8] );
}
else
{
    std::cout << "Hasznalat: ./3.1.1 fajlnev szelesseg magassag n a b c d" <-
        << std::endl;
    std::cout << "Most az alapbeallitasokkal futtatjuk " << szelesseg << "
"
<< magassag << " "
<< iteraciosHatar << " "
<< a << " "
<< b << " "
<< c << " "
<< d << " " << std::endl;
//return -1;
}
```

Megalkotjuk a képet aminek pixeleit a for ciklusok során kiszínezünk.

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

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

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

Végig megyünk a j hosszúságú és k szélességű soron, c változóba elmentjük a (reC, imC) háló csomópont-jának megfelelő komplex számot.

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

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

        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 < iteracionsHatar )
{
    z_n = z_n * z_n + c;

    ++iteracio;
}

iteracio %= 256;
```

Beállítjuk a pixeleknek megfelelő szint, a kép pixeleinek színezési folyamtát a program futása alatt szárazkossan jelezzük.

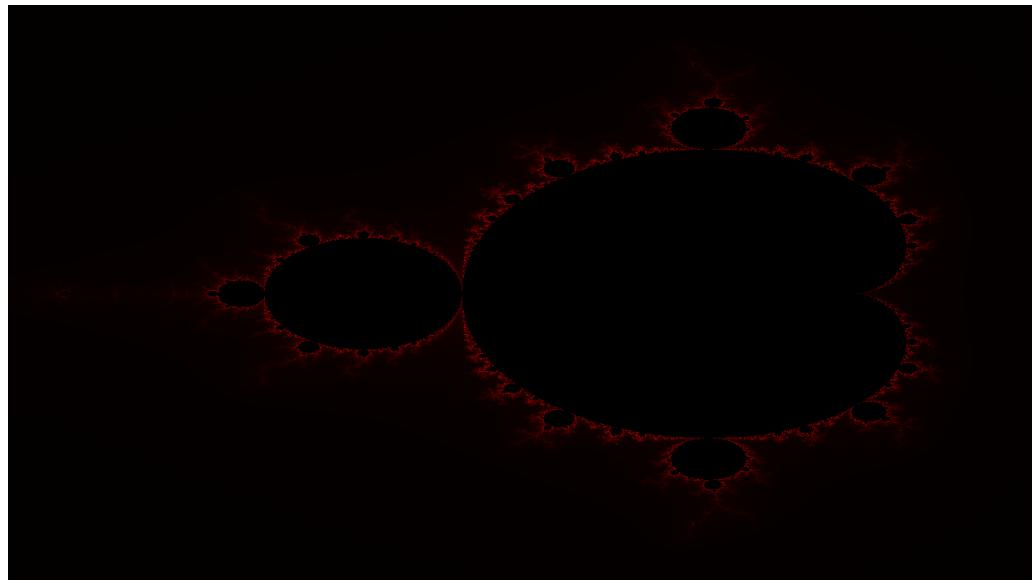
```
    kep.set_pixel ( k, j, png::rgb_pixel ( iteracio%255, 0, 0 ) );

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

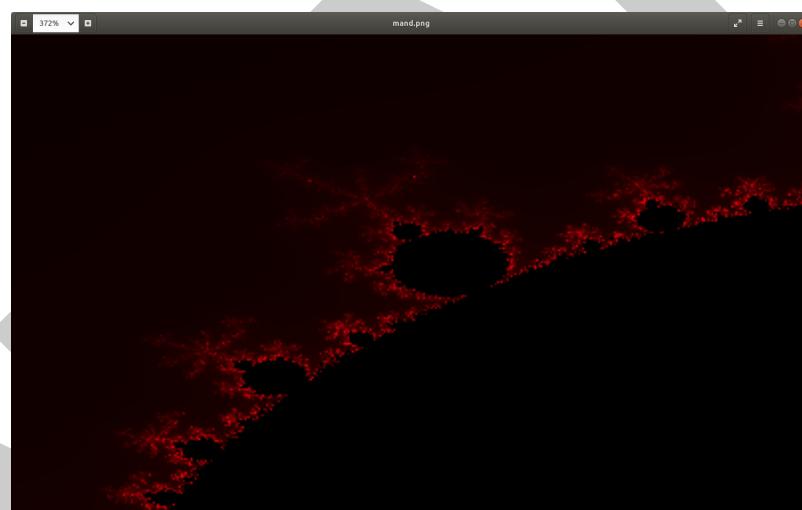
Kiiratjuk a képünk nevét és hogy sikeresen elmentettük.

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

}
```



5.6. ábra. Nagyítás előtt



5.7. ábra.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJnssY>, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_ajandekprogramok/

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

Forráskód linkje: [..../bhax/blob/master/Forrasok/5.%20Fejezet/MandelbrotHalmaz.java](https://bhax/blob/master/Forrasok/5.%20Fejezet/MandelbrotHalmaz.java)

[..../bhax/blob/master/Forrasok/5.%20Fejezet/MandelbrotHalmazNagy%C3%ADt%C3%B3.java](https://bhax/blob/master/Forrasok/5.%20Fejezet/MandelbrotHalmazNagy%C3%ADt%C3%B3.java)

Az alábbi program az előző C++ programnak a javaban írt változata.

Ebben a programban is megadjuk a vizsgált sík méreteit, az iterációs határt. Ezekből a méretekkel létre hozzuk a képet aminek a pixeleit majd színezzük. Az "n" billentyű lenyomásával pontosabb számítást végezünk. Az 'm' gomb lenyomásával pontosabb számítást végezünk de közben sokkal magasabba vesszük az iterációs határt. Ki színezzük a pixeleket és ki rajzoljuk a kapott képet. A pixeleket újra számolunk miután nagyítunk a képen. A program futása soran lehetőségünk van screenshotra is.

```
public class MandelbrotHalmaz extends java.awt.Frame implements Runnable {

    protected double a, b, c, d;
    protected int szélesség, magasság;
    protected java.awt.image.BufferedImage kép;
    protected int iterációsHatár = 255;
    protected boolean számításFut = false;
    protected int sor = 0;
    protected static int pillanatfelvételSzámláló = 0;

    public MandelbrotHalmaz(double a, double b, double c, double d,
                           int szélesség, int iterációsHatár) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
        this.szélesség = szélesség;
        this.iterációsHatár = iterációsHatár;
        this.magasság = (int)(szélesség * ((d-c)/(b-a)));
        kép = new java.awt.image.BufferedImage(szélesség, magasság,
                                              java.awt.image.BufferedImage.TYPE_INT_RGB);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
            }
        });
        addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
                    pillanatfelvétel();
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    if(számításFut == false) {
                        MandelbrotHalmaz.this.iterációsHatár += 256;
                        számításFut = true;
                        new Thread(MandelbrotHalmaz.this).start();
                    }
                } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_M) {
                    if(számításFut == false) {
                        MandelbrotHalmaz.this.iterációsHatár += 10*256;
                        számításFut = true;
                        new Thread(MandelbrotHalmaz.this).start();
                    }
                }
            }
        });
    }
}
```

```
        }
    }
}) ;
setTitle("A Mandelbrot halmaz");
setResizable(false);
setSize(szélesség, magasság);
setVisible(true);
számításFut = true;
new Thread(this).start();
}

public void paint(java.awt.Graphics g) {
    g.drawImage(kép, 0, 0, this);
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
}
public void update(java.awt.Graphics g) {
    paint(g);
}
public void pillanatfelvétel() {
    java.awt.image.BufferedImage mentKép =
        new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKép.getGraphics();
    g.drawImage(kép, 0, 0, this);
    g.setColor(java.awt.Color.BLUE);
    g.drawString("a=" + a, 10, 15);
    g.drawString("b=" + b, 10, 30);
    g.drawString("c=" + c, 10, 45);
    g.drawString("d=" + d, 10, 60);
    g.drawString("n=" + iterációsHatár, 10, 75);
    g.dispose();
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrotHalmaz_");
    sb.append(++pillanatfelvételszámláló);
    sb.append("_");
    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
    sb.append("_");
    sb.append(d);
    sb.append(".png");
    try {
        javax.imageio.ImageIO.write(mentKép, "png",
            new java.io.File(sb.toString()));
    }
}
```

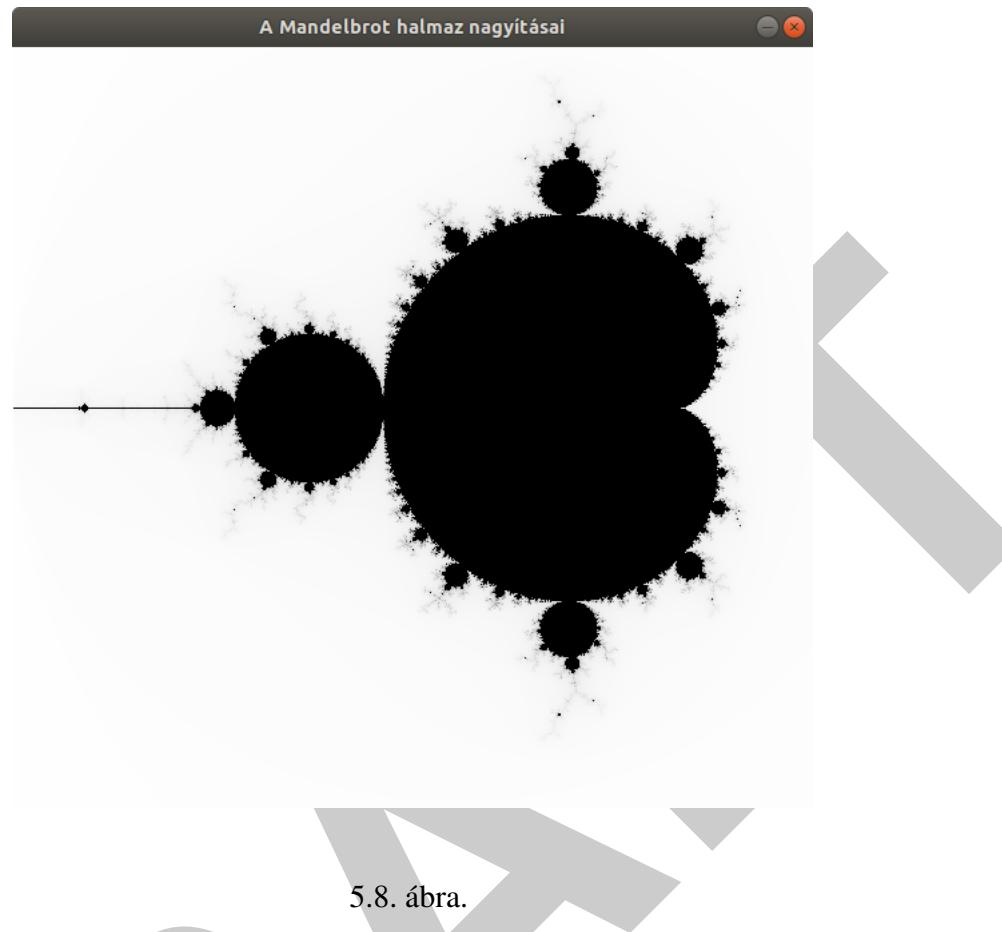
```
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }

    public void run() {
        double dx = (b-a)/szélesség;
        double dy = (d-c)/magasság;
        double reC, imC, reZ, imZ, ujreZ, ujimZ;
        int rgb;
        int iteráció = 0;
        for(int j=0; j<magasság; ++j) {
            sor = j;
            for(int k=0; k<szélesség; ++k) {
                reC = a+k*dx;
                imC = d-j*dy;
                reZ = 0;
                imZ = 0;
                iteráció = 0;

                while(reZ*reZ + imZ*imZ < 4 && iteráció < iterációsHatár) {
                    ujreZ = reZ*reZ - imZ*imZ + reC;
                    ujimZ = 2*reZ*imZ + imC;
                    reZ = ujreZ;
                    imZ = ujimZ;

                    ++iteráció;
                }
                iteráció %= 256;
                rgb = (255-iteráció) |
                    ((255-iteráció) << 8) |
                    ((255-iteráció) << 16);
                kép.setRGB(k, j, rgb);
            }
            repaint();
        }
        számításFut = false;
    }

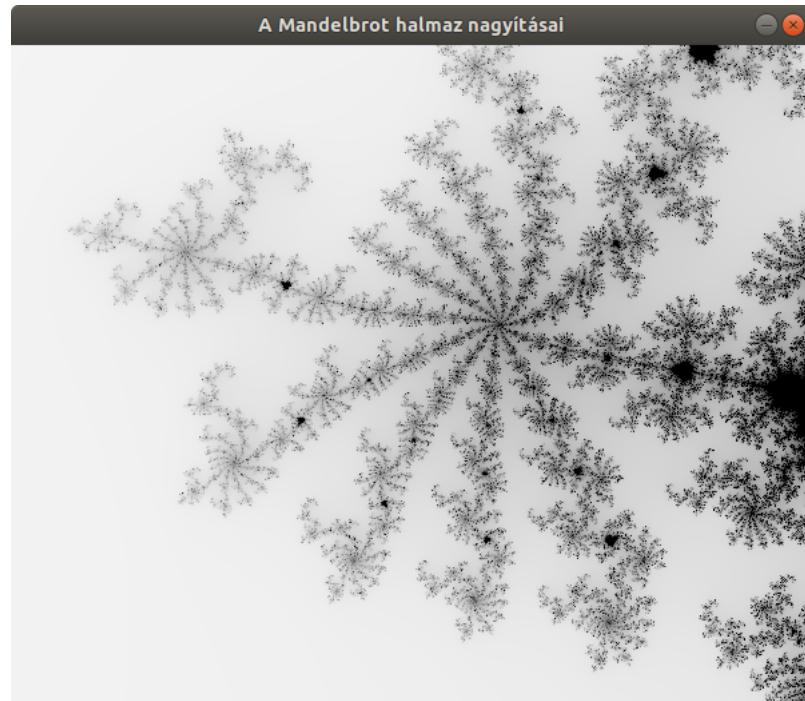
    public static void main(String[] args) {
        new MandelbrotHalmaz(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```



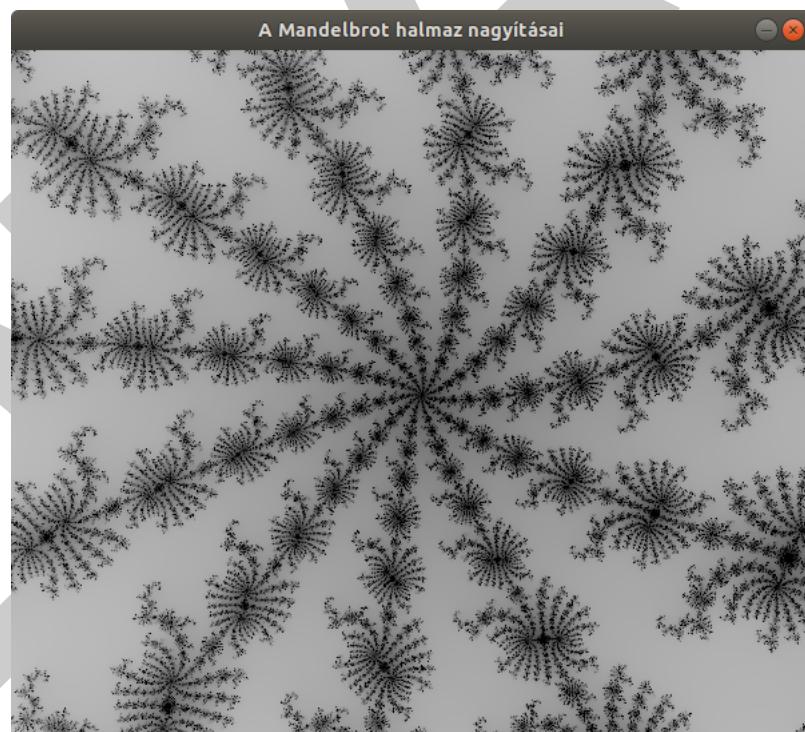
5.8. ábra.



5.9. ábra.



5.10. ábra.



5.11. ábra.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

A polárgenerációs algoritmust felhasználva képez random számokat.

Fordítás **g++ Random.cpp -o Random**

Futtatás **./Random**

Forráskód linkje: [..//bhax/blob/master/Forrasok/6.Fejezet/Pol%C3%A1rGener%C3%A1tor.java](#)

Létrehozuk egy random nevű osztályt, ahol található egy konstruktur és egy logikai változó amiben tároljuk hogy volt e korábbi értke elmentve.

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

//Random osztály
class Random {

public:
    Random(); //konstruktur
    ~Random() {} //destruktor
```

```
    double get(); //random lekérés

private:

    bool exist; //van-e korábbi érték
    double value; //random értéke

};

Random::Random() { //a konstruktor kifejtése
exist = false;
std::srand (std::time(NULL)); //random inicializálás
};

double Random::get() { //random lekérő függvény kifejtése
```

Ha eddig nem volt elmentett értek akkor az algoritmus generál egyet.

```
if (!exist)
{
    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 * u2 - 1;
        w = v1 * v1 + v2 * v2;
    }
    while (w > 1);

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

    value = r * v2;
    exist = !exist;

    return r * v1;
}
```

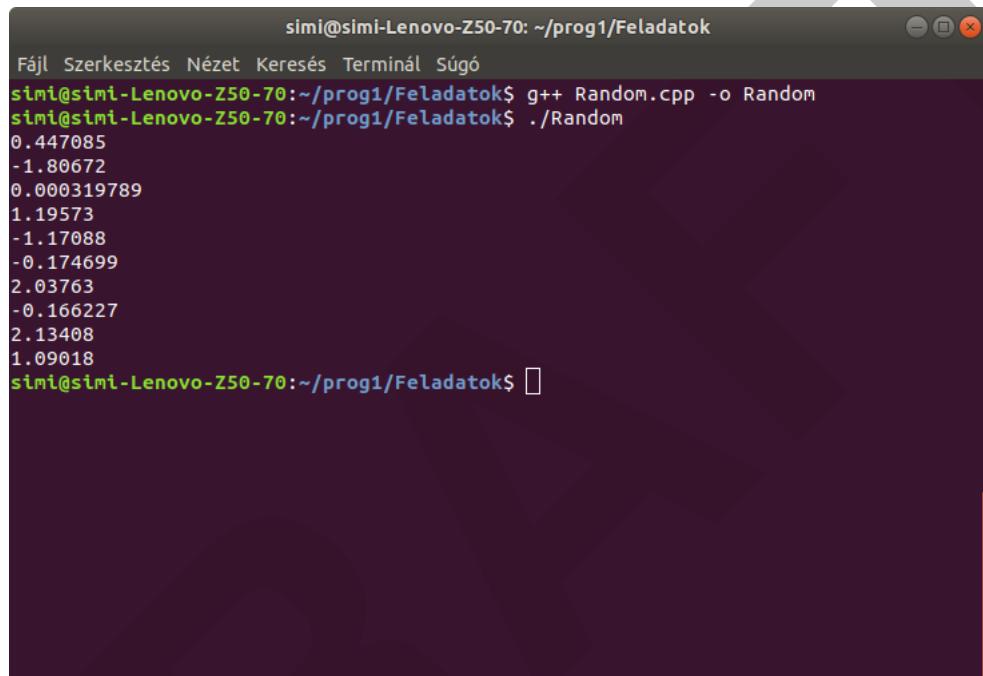
Ha létezett mentek érték akkor azt adja vissza.

```
else
{
    exist = !exist;
    return value;
};
```

10 random szám generálás

```
int main()
{
    Random rnd;

    for (int i = 0; i < 10; ++i) std::cout << rnd.get() << std::endl;
}
```



The screenshot shows a terminal window with a dark background and light-colored text. The title bar reads "simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok". The command entered was "g++ Random.cpp -o Random" followed by "./Random". The output displays ten random floating-point numbers between -1.80672 and 0.447085.

```
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ g++ Random.cpp -o Random
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./Random
0.447085
-1.80672
0.000319789
1.19573
-1.17088
-0.174699
2.03763
-0.166227
2.13408
1.09018
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$
```

6.1. ábra.

Forráskód linkje: <https://gitlab.com/tamas.simon12/bhax/blob/master/Forrasok/6.Fejezet/Random.cpp>

Fordítás **javac Polárgnerátor.java**

Futtatás **./Polárgenerátor**

Az előző program átírva java nyelvre. FVan egy logikai változó amiben tároltuk hogy volt e korábbi mentett érték illetve egy double változó amibe elmentjük majd a kapott értéket.

```
public class PolárGenerátor {

    boolean nincsTárolt = true;
    double tárolt;

    public PolárGenerátor() {

        nincsTárolt = true;
```

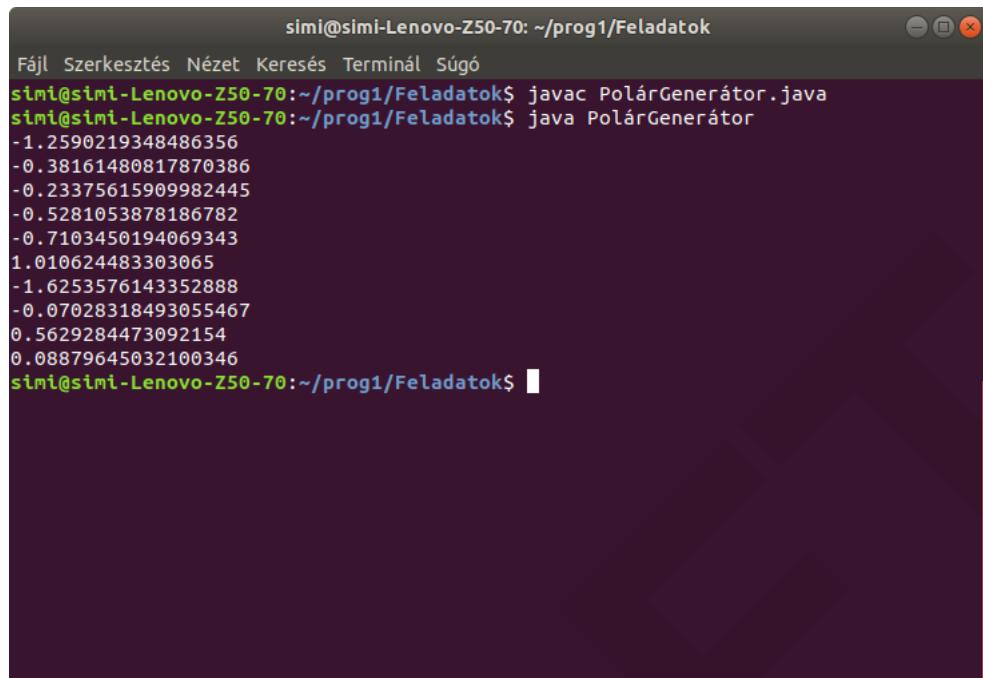
```
}
```

Ha eddig nem volt elmentett értek akkor az algoritmus generál egyet.

```
public double következő() {  
  
    if(nincsTárolt) {  
  
        double u1, u2, v1, v2, w;  
        do {  
            u1 = Math.random();  
            u2 = Math.random();  
  
            v1 = 2*u1 - 1;  
            v2 = 2*u2 - 1;  
  
            w = v1*v1 + v2*v2;  
  
        } while(w > 1);  
  
        double r = Math.sqrt((-2*Math.log(w))/w);  
  
        tárolt = r*v2;  
        nincsTárolt = !nincsTárolt;  
  
        return r*v1;  
  
    } else {  
        nincsTárolt = !nincsTárolt;  
        return tárolt;  
    }  
}  
  
public static void main(String[] args) {  
  
    PolárGenerátor g = new PolárGenerátor();
```

10 random szám generálás

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



A screenshot of a terminal window titled "simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok". The window shows the following command-line session:

```
javac PolárGenerátor.java
java PolárGenerátor
-1.2590219348486356
-0.38161480817870386
-0.23375615909982445
-0.5281053878186782
-0.7103450194069343
1.010624483303065
-1.6253576143352888
-0.07028318493055467
0.5629284473092154
0.08879645032100346
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$
```

6.2. ábra.

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

Forráskód linkje: [./bhax/blob/master/Forrasok/6.Fejezet/lzw.c](#)

Fordítás: **gcc lzw.c -o lzw**

Futtatás: **./lzw < t.txt**

Az LZW algoritmus egy bináris fát épít a gyökértől kezdve a következő képpen:

Ha az olvasott bit 0 és az aktuális csomópontnak nincs nullás gyermeké, létrehozunk egyet, majd visszatérünk a gyökérhez. Ha van nullás gyermek, akkor rálépünk arra a gyermekre és azt vizsgáljuk aktuális csomópontként.

Ha az olvasott bit 1 és az aktuális csomópontnak nincs egyes gyermeké, létrehozunk egyet, majd visszatérünk a gyökérhez. Ha van egyes gyermek, akkor rálépünk arra a gyermekre és azt vizsgáljuk aktuális csomópontként.

A binfa structúrájának létrehozása.

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

typedef struct binfa
```

```
{  
    int ertek;  
    struct binfa *bal_nulla; //mutató a gyökérre (ebben a programban gyökér= ←  
        elem)  
    struct binfa *jobb_egy; //mutató a gyökérre  
  
} 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);  
    }  
}
```

Ha létezett 0-s gyerek rá léptetjük a mutatók, ha nem létezett létre hozzuk és rá állítjuk a mutatót.

```
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;
}
}
```

Ha létezett 1-s gyerek rá léptetjük a mutatók, ha nem létezett létre hozzuk és rá állítjuk a mutatót.

```
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;
}
}
}
```

Kiiratjuk a gyökeret és a fa mélységét

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

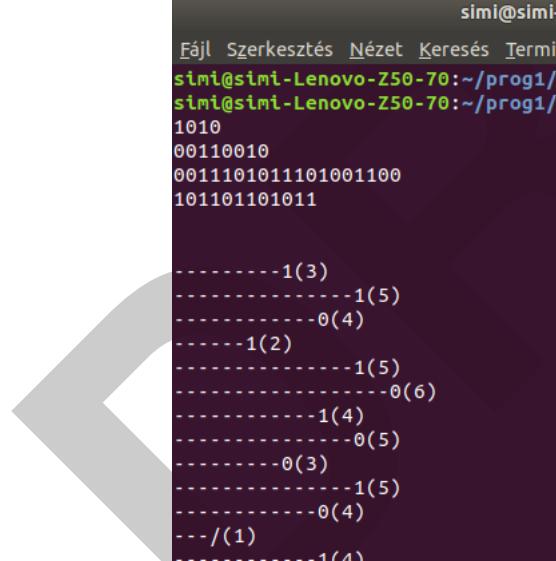
static int melyseg = 0;
int max_melyseg = 0;
```

A fa kiirása inorder módon, következ feladatban látható preorder illetve posztorder módon is.

```
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);
    }
}
```



```
simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ gcc lzw.c -o lzw
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./lzw <t.txt
1010
00110010
0011101011101001100
1011011010111

-----1(3)
-----1(5)
-----0(4)
----1(2)
-----1(5)
-----0(6)
----1(4)
-----0(5)
----0(3)
-----1(5)
-----0(4)
--/(1)
-----1(4)
-----1(3)
----0(2)
-----1(4)
-----0(3)
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$
```

6.3. ábra.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

A bináris fát 3 féle képpen lehet be járni: inorder , preorder, posztorder módon.

Inorder bejárás

Ennél a bejárásnál először a fa bal oldalat, utánna a gyökeret és végül a fa jobb oldalát dolgozzuk fel.

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

Preorder bejárás

Ebben az esetben a bejárást gyökérrel kezdjük, ez után feldolgozzuk a fa bal, majd végül a jobb oldalát.

```
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 : elem ←
                ->ertek,
                melyseg);
        kiir (elem->bal nulla);
        kiir (elem->jobb_egy);
        --melyseg;
    }
}
```

```
    }  
}
```

Posztorder kiiratás

Posztorderr bejárás esetén előbb a bal majd a jobb oldlát dolgozzuk fel a fának és csak legutóljára a gyökeret.

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

6.4. 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 videó:

Megoldás forrása:

Forráskód linkje: [..//bhax/blob/master/Forrasok/6.Fejezet/tag.cpp](#)

A z3a7.cpp programban a gyoker az alábbi módon van deklarálva, az LZWBInFa protected részében. Ebben az esetben a gyökér része a binfának.

```
protected:  
Csomopont gyoker;
```

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

Forráskód linkje: [..../bhax/blob/master/Forrasok/6.Fejezet/mutato.cpp](#)

Ahhoz hogy a gyökér mutató legyen nem kell mast tenni, mint az előbb is említett protected részben "Csomopont gyoker"-et átírni az alábbi módon:

```
protected:  
    Csomopont *gyoker;
```

Így a gyoker típusa egy csomópontra mutató pointer.

```
LZWBinFa ()  
{  
    gyoker = new Csomopont;  
    fa = gyoker;  
}  
~LZWBinFa ()  
{  
    szabadit (gyoker->egyesGyermek ());  
    szabadit (gyoker->>nullasGyermek ());  
    delete gyoker;  
}
```

A fa komstrukturban a gyökrnek a "new" segítségével létrehozunk egy új csomópontot, majd ennek a címét át adjuk a fának. A szabadít függvényben a gyokeret ráállítjuk a gyerekek csomópontára mert ez után töröljük a gyökeret, ha ezt nem teszük szegmentálási hibába ütközünk a program futása közben.

Ezek után csak annyi maradt hogy ahol referencia oprátorokat töröljük a gyökér elől.

```
kiir (&gyoker, os); // itt még csomópot  
kiir (gyoker, os); // itt már mutató
```

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

Tutorált: György Dóra

Fordítás **g++ mozgat.cpp -o mozgat**

Futtatás **./mozgat t.txt -o kifajl**

Forráskód linkje: [..../bhax/blob/master/Forrasok/6.Fejezet/z3a9.cpp](#)

```
class LZWBinFa {
public:

    LZWBinFa () :fa ( &gyoker ) {

    }

    ~LZWBinFa () {
        std::cout << "LZWBinFa dtor" << std::endl;
        szabadit ( gyoker.egyesGyermek () );
        szabadit ( gyoker.nullasGyermek () );
    }
}
```

Az előző feladathoz az LZWBinFa osztály publikus részehez hozzá adjuk az alábbi programrészleteket:

Első sorban létrehozzuk a másoló operátort ami a "gyoker.ujEgyesGyerek"-re, másolja a régi fa egyes gyerekét, a "gyoker.ujNullasGyermek"-re pedig a régi fa nullás gyerekét. Ha a régi fa pointere a régi gyükérnek a referencia címére mutat akkor a fa megkapja a gyökér referencia címére.

```
LZWBinFa ( const LZWBinFa & regi ) {
    std::cout << "LZWBinFa copy ctor" << std::endl;

    gyoker.ujEgyesGyermek ( masol ( regi.gyoker.egyesGyermek (), regi <-
        .fa ) );
    gyoker.ujNullasGyermek ( masol ( regi.gyoker.nullasGyermek (), <-
        regi.fa ) );

    if ( regi.fa == & ( regi.gyoker ) )
        fa = &gyoker;

}
```

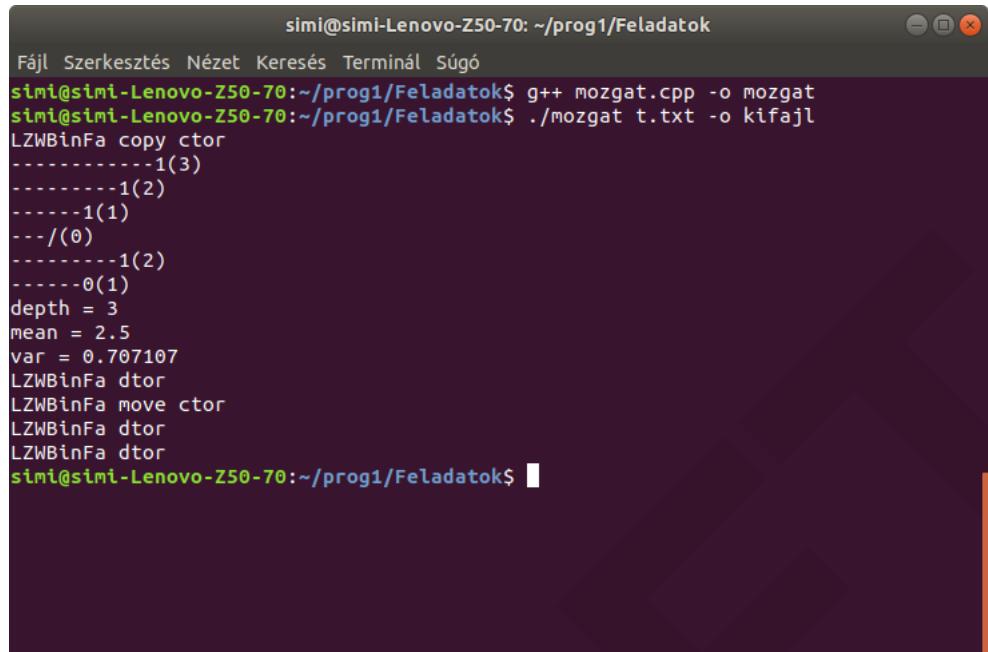
Ezek után a régi és az új fánka a gyökérre mutató pointereit felcseráljuk, majd a csere után a régi fa poinereit rá állítjuk a "nullptr"-re.

```
LZWBinFa ( LZWBInFa && regi ) {
    std::cout << "LZWBinFa move ctor" << std::endl;

    gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
    gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

    regi.gyoker.ujEgyesGyermek ( nullptr );
    regi.gyoker.ujNullasGyermek ( nullptr );

}
```



A screenshot of a terminal window titled "simi@simi-Lenovo-Z50-70: ~/prog1/Feladatok". The window shows the following command-line session:

```
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ g++ mozgat.cpp -o mozgat
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ ./mozgat t.txt -o kifajl
LZWBinFa copy ctor
-----1(3)
-----1(2)
----1(1)
---/(0)
-----1(2)
----0(1)
depth = 3
mean = 2.5
var = 0.707107
LZWBinFa dtor
LZWBinFa move ctor
LZWBinFa dtor
LZWBinFa dtor
simi@simi-Lenovo-Z50-70:~/prog1/Feladatok$ █
```

6.4. ábra.

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az alábbi forráskódok linkjei: [./bhax/tree/master/Forrasok/7.%20Fejezet/Myrmecologist](https://bhaxor.github.io/Forrasok/7.%20Fejezet/Myrmecologist)

Ehhez a feladathoz is szükség lesz az OpenCV programra.

A proramot az alábbi parancsokkal hozzuk létre és futtatjuk:

qmake myrmecologist.pro

make

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

Az ANT header fájlba megadunk egy x, y és dir változót. A dir a grand függvény visszatérési értékének a 8-al való osztási maradékát kapja eredményül.

```
#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
```

```
#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
```

```
#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;
                }
                sum += grid[s][o];
            }
}

int AntThread::getAnts ( int *row, int *col, int *dir )
{
    int i, j, o;
    double maxSum = 0.0;
    int maxIdx = -1;

    for ( i = 0; i < numAnts; ++i ) {
        if ( ants->at(i).y < 0 ) continue;
        if ( ants->at(i).x < 0 ) continue;
        if ( ants->at(i).y > height ) continue;
        if ( ants->at(i).x > width ) continue;

        o = ants->at(i).x + ants->at(i).y * width;
        sumNbhs ( grid, ants->at(i).y, ants->at(i).x, ants->at(i).dir, o );
        if ( sumNbhs > maxSum ) {
            maxSum = sumNbhs;
            maxIdx = i;
        }
    }

    if ( maxIdx > -1 ) {
        *row = ants->at(maxIdx).y;
        *col = ants->at(maxIdx).x;
        *dir = ants->at(maxIdx).dir;
    }
}
```

```
        } 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;
```

```
}
```

Az AntWin osztályban jelenik meg az ablak szélessége magassága és a hangyák száma.

Megadjuk hogy a "p" billentyűvel szünetelhetjük a szimulációt, a "q" vagy "ESC" bombbal pedig kilépünk belőle.

```
#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
```

Az ablak címének az "Ant Simulation"-t adjuk meg. A megjeleített hangyákat sötét zöldel jelöljük, ahol pedig elhaladt ott egyre világosabb zöldel jelöljük.

```
#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();
}
```

A mainben meghívjuk az eddig deklarált függvényeket.

```
#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", "szelessseg" }, "Oszlopok (cellakban ←
        ) szama.", "szelessseg", "200" );
    QCommandLineOption magas_opt ( { "m", "magassag" }, "Sorok (cellakban) ←
        szama.", "magassag", "150" );
    QCommandLineOption hangyszam_opt ( { "n", "hangyszam" }, "Hangyak szama. ←
        ", "hangyszam", "100" );
    QCommandLineOption sebesseg_opt ( { "t", "sebesseg" }, "2 lepes kozotti ←
        ido (millisek-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 );

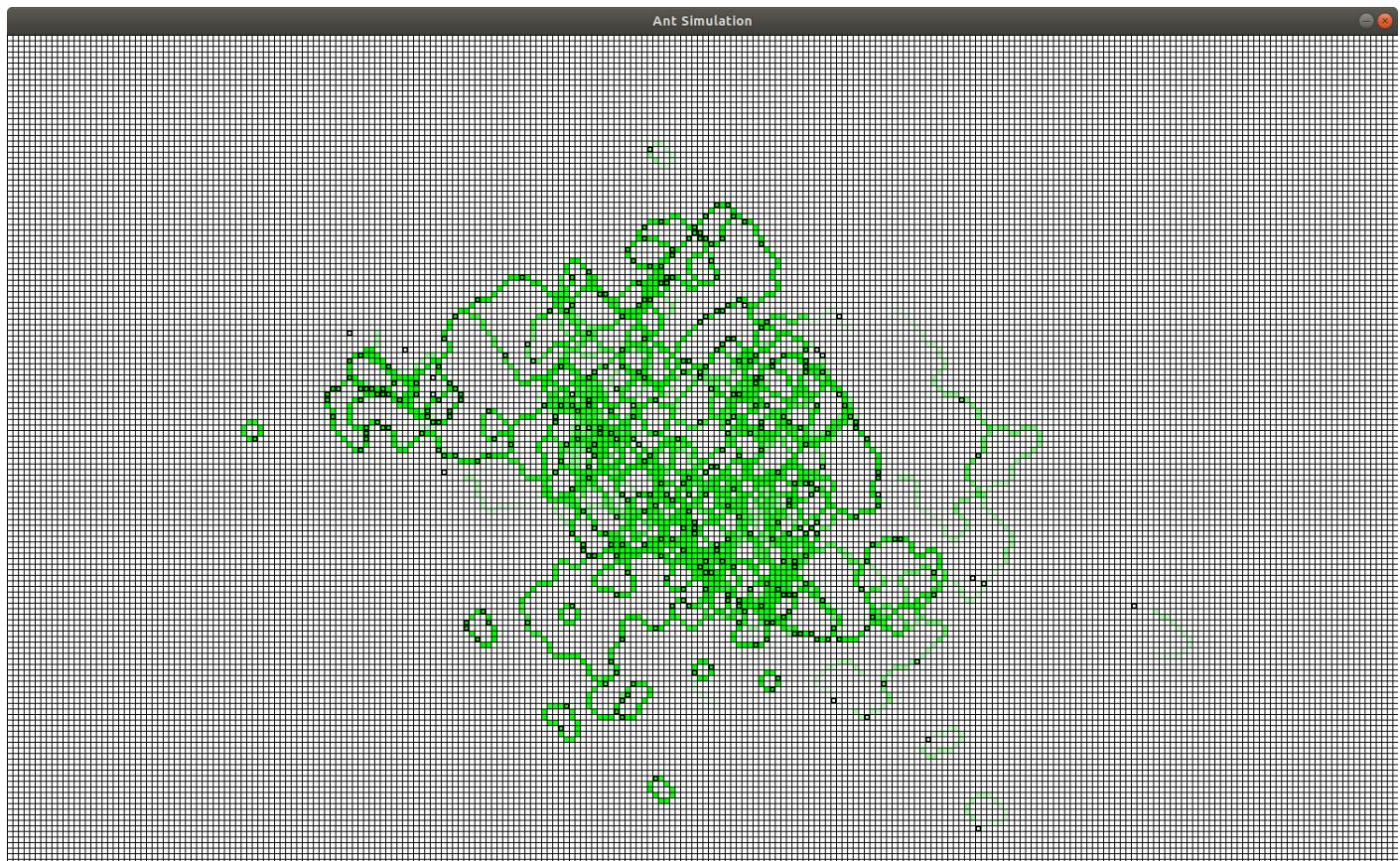
QString szeles = parser.value ( szeles_opt );
QString magas = parser.value ( magas_opt );
QString n = parser.value ( hangyaszam_opt );
QString t = parser.value ( sebesseg_opt );
QString parolgas = parser.value ( parolgas_opt );
QString feromon = parser.value ( feromon_opt );
QString szomszed = parser.value ( 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() );

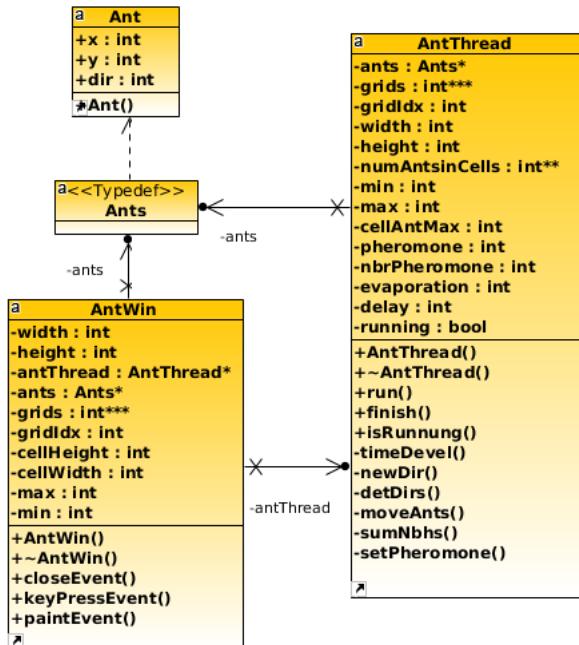
AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), feromon.toInt(), szomszed.toInt(), parolgas.toInt(),
           alapertek.toInt(), mincella.toInt(), maxcella.toInt(),
           cellameret.toInt() );

w.show();
```

```
    return a.exec();  
}
```



7.1. ábra.



7.2. ábra.

7.2. Java életjáték

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

Megoldás video:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Forráskód linkje: [..../bhax/blob/master/Forrasok/7.%20Fejezet/Java%20%C3%A9letj%C3%A1t%C3%A9k-%C3%A9k-Sejtautomata.java](http://bhax/blob/master/Forrasok/7.%20Fejezet/Java%20%C3%A9letj%C3%A1t%C3%A9k-%C3%A9k-Sejtautomata.java)

Fordítás: `javac Sejtautomata.java`

Futtatás: `java Sejtautomata`

John Horton Conway a Cambridge egyetem matematikusa találta ki az életjátékot. A cella egy négyzetrácsos mezőből áll, ahol sejtek vannak, ezek elő és halott állapotok között váltakoznak. A játék kimenetele a kezdő állapotól függ, attól hogy hogyan hagyheyezük el a játék elején a kezdő sejteket. Egy sejttel egy kör alatt három dolog történhet: elpusztul, ha 2-nél kevesebb vagy 3-nál több szomszédja van, túléli, ha van 2 vagy 3 szomszédja, vagy új sejt születhet minden olyan cellában aminek pontosan 3 szomszédja van. A mi programunk a sikló kilövőt hozza létre.

```

/** DIGIT 2005, Javat tanítok
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class Sejtautomata extends java.awt.Frame implements Runnable {
  
```

```
public static final boolean ÉLŐ = true;
public static final boolean HALOTT = false;
protected boolean[][][] rácsok = new boolean[2][][];
protected boolean[][] rács;
protected int rácsIndex = 0;
protected int cellaSzélesség = 20;
protected int cellaMagasság = 20;
protected int szélesség = 20;
protected int magasság = 10;
protected int várakozás = 1000;
private java.awt.Robot robot;
private boolean pillanatfelvétel = false;
private static int pillanatfelvételszámláló = 0;
```

Deklaráljuk a változóinkat,boolean változó segítségével döntjük majd el egy sejtről hogy elő, ekkor igaz értékű, vagy halott, ekkor hamis. Deklarálunk egy kétdimenziós rácsot, amiben majd fut a játék. Eltároljuk a cella adatait(cellaSzélesség, cellaMagasság) és a sejttér adatait(szélesség, magasság), a várakozás pedig a két képfrissítés közötti idő.

```
public Sejtautomata(int szélesség, int magasság) {
    this.szélesség = szélesség;
    this.magasság = magasság;
    rácsok[0] = new boolean[magasság][szélesség];
    rácsok[1] = new boolean[magasság][szélesség];
    rácsIndex = 0;
    rács = rácsok[rácsIndex];
    for(int i=0; i<rács.length; ++i)
        for(int j=0; j<rács[0].length; ++j)
            rács[i][j] = HALOTT;
    siklóKilövő(rács, 5, 60);
    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
            setVisible(false);
            System.exit(0);
        }
    });
}
```

Itt lére hozzuk a Sejtautomata objektumot. Létrehozzuk a rächálót, amin a for ciklussal végig menve minden sejet halottra állítunk, majd létrehozzuk a siklókilövőt a rács 5.-ik oszlopának 60.ik sorába.

```
addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent e) {
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            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();
}
});
```

A programot futás közben befolyásolhatjuk bizonyos billentyűk lenyomásával. A K gombott lenyomva kicsinyíteni tudjuk az ablakot, a cella magasságát és szélsséget megfelezzük és ezt állítjuk be új méretnek. Az N billentyűt megnyomva nagyíthatunk, ebben az esetben a cella adatait megduplázzuk, és duplázott méretet állítjuk be. Az S billentyűvel képernyőfelvételt tudunk készíteni. A G-vel gyorsítani tudjuk a játék menetét, megfelezve a várakozási időt. Az L billentyű a G ellentéte ezzel lassítani lehet a körök közti időt.

```
addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});

addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});
```

Egér kattintással új sejteket tudunk létre hozni, x és y tárolja az egér pozícióját. Az egér bal gombát nyomva tárta egy egész sorozat új ssejtet tudunk létre hozni.

```
cellaSzélesség = 10;
cellaMagasság = 10;
try {
    robot = new java.awt.Robot(
        java.awt.GraphicsEnvironment.
        getLocalGraphicsEnvironment().
        getDefaultScreenDevice());
} catch(java.awt.AWTException e) {
    e.printStackTrace();
}
```

```
setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
          magasság*cellaMagasság);
setVisible(true);
new Thread(this).start();
}
```

Meghatározzuk a cella szélességét és magasságát, megadjuk a létrehozott ablak címét , "Setautomata", majd láthatóvá tesszük a létrehozott ablakot. Eddig tartottak az előkészületek, innentől indul a játék.

```
public void paint(java.awt.Graphics g) {
    boolean [][] rács = rácsok[rácsIndex];
    for(int i=0; i<rács.length; ++i) {           // végig lépked a ←
        sorokon
            for(int j=0; j<rács[0].length; ++j) {   // s az oszlopok
                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);
                g.setColor(java.awt.Color.LIGHT_GRAY);
                g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                            cellaSzélesség, cellaMagasság);
            }
    }
}
```

A két for ciklussal megyünk a rács minden celláján, ha a rács adott cellája amin vagyunk élő akkor a cellát feketére, ha halott akkor fehérre szineyyük.

```
if(pillanatfelvétel) {
    pillanatfelvétel = false;
    pillanatfelvétel(robot.createScreenCapture
        (new java.awt.Rectangle
            (getLocation().x, getLocation().y,
             szélesség*cellaSzélesség,
             magasság*cellaMagasság)));
}
}
```

Képernyőfelvétel készítése

```
public int szomszédokSzáma(boolean [][] rács,
    int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 0;
    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 = 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;
}
```

Ebben a függvényben vizsgáljk a szomszédokat, a sejt a 0. pozícióban van vele nem foglalkozunk most. A sejt tér szélein lévő "levágott" szomszédok vele szemben lévő oldalon a vele egy szintben lévő sejteket tekintjük. A függvény eredményül a szomszédok számát adja.

```
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) {
                szomszedja van, különben halott lesz. */
                if(élők==2 || élők==3)
                    rácsUtána[i][j] = ÉLŐ;
                else
                    rácsUtána[i][j] = HALOTT;
            } else {
                if(élők==3)
                    rácsUtána[i][j] = ÉLŐ;
                else
                    rácsUtána[i][j] = HALOTT;
            }
        }
    }
    rácsIndex = (rácsIndex+1)%2;
}

public void run() {
```

```
while(true) {  
    try {  
        Thread.sleep(várakozás);  
    } catch (InterruptedException e) {}  
  
    időFejlődés();  
    repaint();  
}
```

Az időFejlődés függvény dönti el hogy mi történjen egy sejttel attól függően hány szomszédja van. Két for ciklussal végig megyünk a cellákon, minden lépés után meghívva a szomszédokSzáma függvényt, hogy megtudjuk hány élő szomszédja van a sejtnek és ezt a számot eltároljuk a "élők" változóba. Ha a sejtünk élő volt, élő marad ha van 2 vagy 3 szomszédja ellenkező esetben halott lesz. Ha a sejtünk halott abban az esetben lesz élő ha pontosan 3 szomszédja van,különben halott marad.

```
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;  
}
```

Az elhelyezett sikló átlósan jobbra halad, a fent megadott minta alapján cellákat előre téve halad. Az x és y jelöli a sikló bal felső részét.

```
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;

}
```

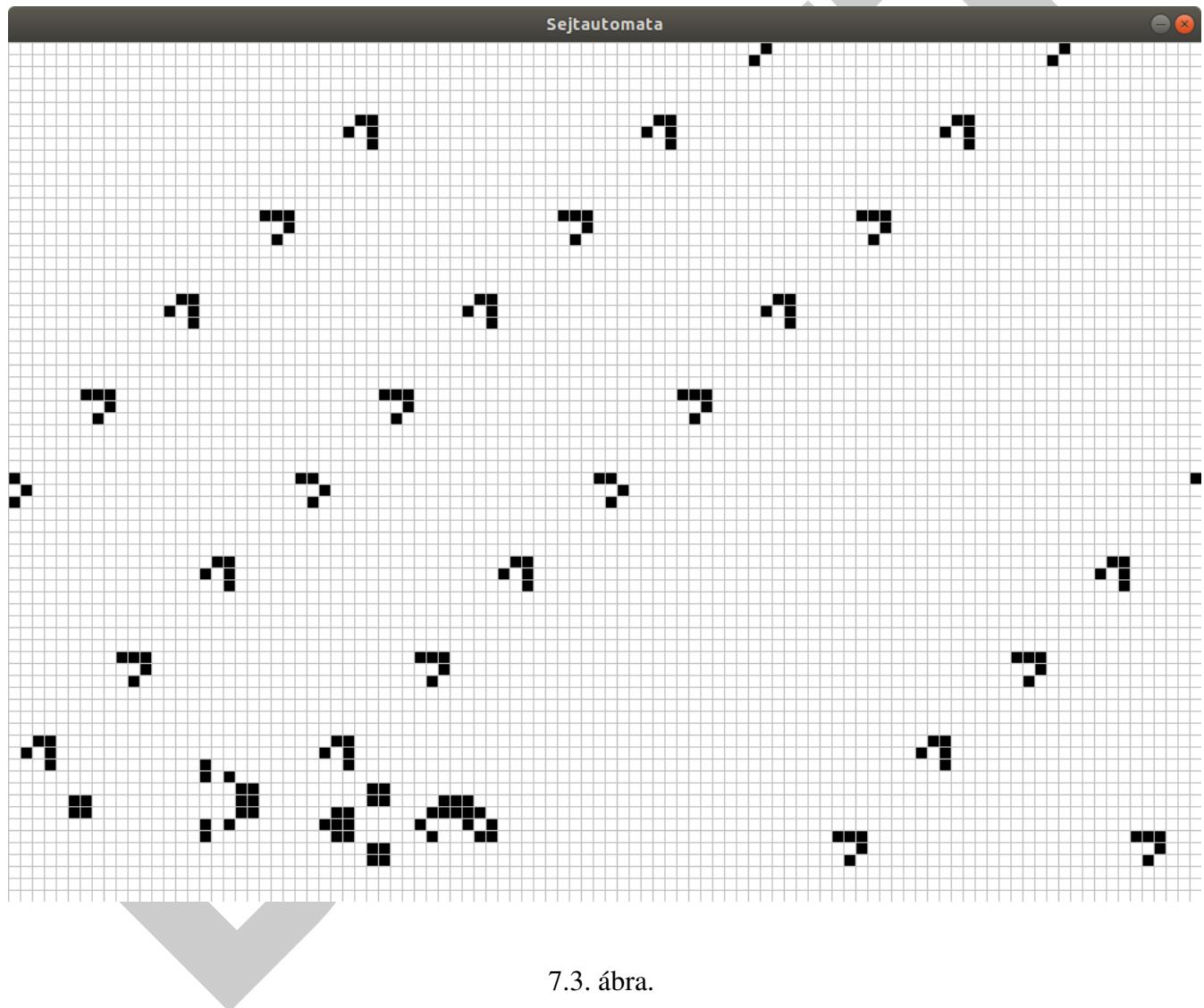
A fentebb megadott minta lesz az amin haladni fog a ki lőtt sikló, ezeket a cellákat allítjuk majd élőre.

```
public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételszámláló);
    sb.append(".png");
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
```

A készített képernyőképek "Sejtautomata" névvel lesznek elmentve png formátumban.

```
public void update(java.awt.Graphics g) {  
    paint(g);  
}  
  
public static void main(String[] args) {  
    new Sejtautomata(100, 75);  
}  
}
```

A program main részében meghatározza a Sejtautomata sorainak és oszlopainak a számát.



7.3. ábra.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Forráskód linkje: [./bhax/tree/master/Forrasok/7.%20Fejezet/C++%20%C3%A9letj%C3%A1t%C3%A1l%C3%A9k](https://github.com/bhax/tree/master/Forrasok/7.%20Fejezet/C++%20%C3%A9letj%C3%A1t%C3%A1l%C3%A9k)

A proramot az alábbi parancsokkal hozzuk létre és futtatjuk:

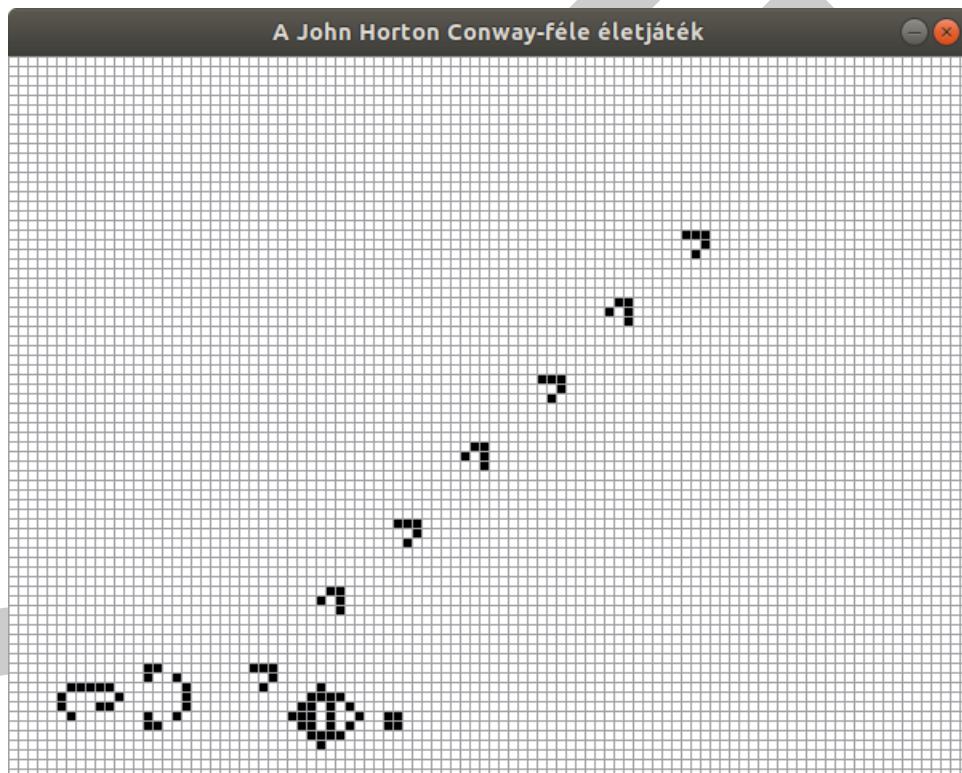
qmake-qt4 -project

qmake-qt4 Sejtauto.pro

make

./Sejtauto

A C++ program ugyan azt az eredményt adja mint az előbb végig nézett java program. Szintén ugyan azt a sikló kiövős játékot dolgozza fel. Ahhoz hogy a C++-ban írt porgram fusson telepíteni kell az OpenCV-t ami egy hosszú folyamat.



7.4. ábra.

7.4. BrainB Benchmark

Megoldás video:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Tutoriáltam volt: György Dóra

Forráskód linkje: [./bhax/tree/master/Forrasok/7.%20Fejezet/BrainB](#)

A proramot az alábbi parancsokkal hozzuk létre és futtatjuk:

qmake BrainB.pro

make

./BrainB

A BrainB-s program fő célja a tehetségkutatás az e-sport terén. A program a sportolók reakció képességét vizsgája. A program futása során a jobb egér gombot nyomva tartva követnünk kell Samu Entropy karakterét. Ha kellő ideig követtük újabb játékos jelenik meg ezzel tesztve hogy hany új karakter megjelöléséi tudjuk követni a saját (Samu) karaktert. Ha viszont sok ideig "szemelöl veszítjük" azaz nem a megfelelő karakteren tartjuk a kurzrt akkor eltűnnék a játékosok mindenadvig még újra a saját karakterünket nem követjük.

A program az OpenCV programját felhasználva jeleníti meg a grafikus felületet.

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>
```

A Hero classban alkotjuk meg karakterünk akit követni fogunk. Ay x és y tárolja a karakter körüli téglalap szélességét és hosszát. Color lesz a színe, agility a sebessége, name pedig a neve.

```
class Hero;
typedef std::vector<Hero> Heroes;

class Hero
{
public:
    int x;
    int y;
    int color;
    int agility;
    int cond {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 () { }
```

A move függvény mozgatja a karaktert, nehezítve a követését.

```
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 };

    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;

}

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;
    }
}

}
```

Itt jönnek létre a további játékosok, itt rajzoljuk meg őket. Az OpenCV segít nekünk újra abban hogy megjelenítsük a játékosokat. A játék akkor ér véget mikor az entime léri a 10 percet.

```
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
```

Az alábbi kódban található az a rész amiben deklaráljuk hogy hova menjtük el az kapott eredményeket. Megnyitjuk a fájlt ami tartalmazza a játékosok neveit, a játék időt, illetve hogy hány megjelenített karakter játékos után vesztettük el Samut. Majd closeal bezárjuk a megnyitott fájlt.

```
#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
```

Ebben a részben határozzuk meg hogy hol elenjenk meg az új játékosok ha már elég ideje követjük Samut. A lényeg hogy közel spawnoljanak Samuhz mivel a játék lenyege hogy a "sűrűjében" hogy tufjuk követni saját játékosunk.

```
#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;

}
```

Az alábbi részben update-elődnek a karakterek, kiiratjuk az eredményt ha kiléünk a játékból. Hozzá adjuk az egér kurzor mozgását, mivel ezzel követjük Samut. Ezek mellett az S billentyűt lenyomva menteni tudjuk az eddigi eredményt. P gombbal pedig szünetetltetjük. A q és ESc kiléptet a játékból.

```
#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 );\n\n    if ( brainBThread->get_paused() ) {\n        QString pausedstr = "PAUSED (" + QString::number ( ←\n            brainBThread->get_nofPaused() ) + ")";\n\n        qpainter.drawText ( 210, 40, pausedstr );\n    }\n\n    qpainter.end();\n}\n\nvoid BrainBWin::mousePressEvent ( QMouseEvent *event )\n{\n    brainBThread->set_paused ( false );\n}\n\nvoid BrainBWin::mouseReleaseEvent ( QMouseEvent *event )\n{\n    //brainBThread->set_paused(true);\n}\n\nvoid BrainBWin::mouseMoveEvent ( QMouseEvent *event )\n{\n    start = true;\n\n    mouse_x = event->pos().x() -xs - 60;\n    //mouse_y = event->pos().y() - yshift - 60;\n    mouse_y = event->pos().y() - ys - 60;\n}\n\nvoid BrainBWin::keyPressEvent ( QKeyEvent *event )\n{\n    if ( event->key() == Qt::Key_S ) {\n        save ( brainBThread->getT() );\n    } else if ( event->key() == Qt::Key_P ) {\n        brainBThread->pause();\n    } else if ( event->key() == Qt::Key_Q || event->key() == Qt::Key_Escape ) {\n        close();\n    }\n}
```

```
BrainBWin::~BrainBWin()
{
}
```

qouttal kiiratjuk licenszeket, illetve a végleges eredményt.

```
#include <QApplication>
#include <QTextStream>
#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();
}
```

A BrainB.pro egybe gyűjti azoknak a programoknak a neveit amiket a programunk a futás alatt felhasznál.

```
QT += widgets core
CONFIG += c++11 c++14 c++17
QMAKE_CXXFLAGS += -fopenmp
LIBS += -fopenmp
LIBS += `pkg-config --libs opencv`

TEMPLATE = app
TARGET = BrainB
INCLUDEPATH += .

HEADERS += BrainBThread.h BrainBWin.h
SOURCES += BrainBThread.cpp BrainBWin.cpp main.cpp
```



7.5. ábra.

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa...
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Tanulságok, tapasztalatok, magyarázat...

2. passz

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3. passz

8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndl8>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. passz

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

5. passz lehetőség

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

6. passz lehetőség (SMNIST játék 10 szint miatt 5+2)

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

7. passz lehetőség (SMNIST játék 10 szint miatt 5+2)

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

3 szintet különmöztetünk meg a számítógépek programozásában: gépi, assembli szintű, magas. A magas szintű nyelven megírt programot forrásprogramnak, vagy forrásszövegnek nevezzük. Fordító programmal tudunk gépi níelvű programra fordítani forrásprogramból. A fordító a következő lépéseket végzi: lexikális elemzés(feldarabol), szintaktikai elemzés(ellenőrzés), szemantikai elemzés, kódgenerálás, majd afutó program működését a futtató rendszer felügyeli. Az interpreteres fordítási technika nem készít tárgyprogramot, utasításonként elemzi a programot, értelmezi és végrehajtja. minden programnyelvnek megvan a saját szabványa, definiálva vannak a szintaktikai és a szemantikai szabályok.

Imperatív nyelvek, algoritmikus nyelvek, ezek működtetik a processzort. A program utasítások sorozata. Az algoritmus a változók értékeit alakítja.

Deklaratív nyelvek, nem algoritmikus nyelvek. A programozó csak a problémát adja meg. A programozónak nincs lehetisége memóriaműveletekre.

Máselvő (egyéb) nyelvek, incs egységes jellemzőjük. Általában tagadják valamelyik imperatív jellemzít.

Jelölésrendszerek: terminális, nem terminális, alternatíva, opción, iteráció.

Az adattípus egy absztrakt programozási eszköz, ami komponenseként jelenik meg. Az adattípust a tartomány (amilyen típust felvehet), a művelet (amit a tartomány elemein elvégezhet) és a reprezentáció (az értékek tárban való megjelenése) határoz meg. minden nyelv rendelkezik beépített típusokkal, és a programozó is definiálhat típusokat. Saját típust úgy tudunk létrehozni, hogy megadjuk a tartományát, a műveleteit és a reprezentációját. Az adattípusoknak két nagy csoportjuk van: skalár (egyszerű), strukturált (összetett).

Az egész típus belső ábrázolása fixpontos, a valósé lebegőpontos. A numerikus típusok (egész és valós) a numerikus és hasonlító műveletek hajthatók végre. A karakteres típus elemei karakterek, a karakterlánc cé sztring. Logikai típust. Ennek tartománya a hamis és igaz értékekkel áll. A felsorolásos típust saját típusként kell létrehozni.

Összetett típus a tömb és a rekord. A tömb elemei azonos típusuak, lehetnek egy vagy többdimenziósak. A tömb elemeire indexek segítségével tudunk hivatkozni. A rekord tartományának elemei értékcsoporthoz, melyek elemei különböző típusuak lehetnek. minden mezőnek van neve és típusa. A C-ban és a Pascal-ban a rekord típusa egyszintű (nincsenek almezők), de a mezők típusa lehet összetett. A rekord nevével az összes mezőre tudunk hivatkozni, ha csak egyre akarunk akkor: eszköznév.mezőnév.

A mutató típus egyszerő típus. A mutató típus tartományának elemei tárcímek, így valósítjuk meg az indirekt címezést (minden mutató egy címre mutat). Van egy speciális eleme, a NULL, ami nem mutat sehova.

A nevesített konstansnak három komponense van: név, típus, érték. Mindig deklarálni kell. Szerepe egyrészt, hogy a sokszor előforduló értékeket "beszélő" névvel látjuk el, így könnyebben tudunk utalni rá a szövegben, másrészt ha az értékét meg akarjuk változtatni, akkor ezt elég egy helyen megtenni.

A változónak négy komponense van: név, cím, attribútumok, érték. A név az azonosító, ez jelenik meg a programban. Az attribútumok a változó futás közbeni viselkedését határozzák meg. Hárrom féle deklarációt különböztetünk meg: explicit(a programozó végzi, a változó teljes nevéhez ad meg attributumot), implicit (A programozó végzi, betűkhöz rendel attribútumokat egy külön deklarációs utasításban) és autómatikus (a fordítóprogram rendel attribútumot azokhoz a változókhoz, amelyek nincsenek explicit módon deklarálva). Egy változóhoz cím rendelhető: startikus tárkiosztás: A futás előtt eldő a változó címe, és a futás alatt az nem változik. Dinamikus tárkiosztás: A cím hozzárendelését a futtató rendszer végzi, a futás során változhat.

A C típurendzere: aritmetikai típusok (integrált, valós), származtatott típusok (tömb, függvény, mutató, struktúra, union), és void típusok.

A kifejezéseknek két komponensük van, érték és típus. Egy kifejezés a követzőkből áll: operandusok, operátorok, kerek zárójelek. egy operátor lehet: egyoperandusú (unáris), kétoperandusú (bináris), vagy háromoperandusú (ternáris) operátorokról.

Kétoperandusú operátorok esetén az operandusok és az operátor sorrendje lehet: prefix, az operátor az operandusok előtt áll (* 3 5), infix, az operátor az operandusok között áll (3 * 5), postfix, az operátor az operandusok mögött áll (3 5 *).

A műveletek végrehajtási sorrendje mehet balról jobbra, vagy jobbról balra felírától függően.

A utasításonak két nagy csoportjuk van: a deklarációs és a végrehajtható utasítások. A deklarációs utasítások mögött nem áll tárgykód. A végrehajtható utasításokból generálja a fordítóprogram a tárgykódot.

Értékkadó utasítás feladata beállítani vagy módosítani egy (esetleg több) változó értékkomponensét a program futásának bármely pillanatában

Az üres utasítás hatására a processzor egy üres gépi utasítást hajt végre.

Az ugró utasítás segítségével a program egy adott pontjáról egy adott címkével ellátott végrehajtható utasításra adhatjuk át a vezérlést.

A kétfelirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján két tevékenység közül válasszunk, illetve egy adott tevékenységet végrehajtsunk vagy sem.

A többfelirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján egymást kölcsönösen kizáró akárhány tevékenység közül egyet végrehajtsunk.

A ciklusszervező utasítások lehetivé teszik, hogy a program egy adott pontján egy bizonyos tevékenységet akárhányszor megismételjünk. Egy ciklus általános felépítése: fej, mag, vég. Az ismétlésre vonatkozó információk vagy a fejben vagy a végben szerepelnek.

Feltételes ciklusnál az ismétlődést egy feltétel igaz vagy hamis értéke szabályozza. Szemantikájuk alapján beszélünk kezdőfeltételes és végfeltételes ciklusról.

Eliírt lépésszámú ciklusfajtánál az ismétlődésre vonatkozó információk a fejben vannak. minden esetben tartozik hozzá egy ciklusváltozó.

A felsorolásos ciklus az elírt lépésszámú ciklus explicit módon megadott értékeket vesz fel, és minden felvett érték mellet lefut a mag.

A végtelen ciklus az a ciklusfajta, ahol sem a fejben, sem a végben nincs információ az ismétlődésre vonatkozóan.

Az összetett ciklus az előző négy ciklusfajta kombinációból áll össze. A ciklusfejben tetszőlegesen sok ismétlődésre vonatkozó információ sorolható föl

A continue átugorja a ciklus magjának hátralévi utasításait.

A break a ciklust szabályosan befejezteti, illetileg kilép a többszörös elágaztató utasításból.

A return szabályosan befejezteti a függvényt és visszaadja a vezérlést a hívónak.

A blokk olyan programegység, amely csak másik programegység belsejében helyezkedhet el, külső szinten nem állhat. Formálisan a blokknak van kezdete, törzse és vége. A kezdetét és a végén egy alapszó jelzi. A törzsben lehetnek deklarációs és végrehajtható utasítások.

A hatáskör szinonimája a láthatóság. A program szövegének azon részét, ahol az adott név ugyanazt a programozási eszközöt. Egy programegységen belül nevet a programegység lokális nevének nevezik. Kétféle hatáskörkezelést ismerünk, a statikus és a dinamikus hatáskörkezelést: A statikus hatáskörkezelés fordítási időben történik, a fordítóprogram végzi. Statikus hatáskörkezelés esetén egy lokális név hatásköre az a programegység, amelyben deklárták és minden olyan programegység, amelyet ez az adott programegység tartalmaz.

Az I/O platform-, operációs rendszer-, implementációfüggő. Az I/O az az eszközrendszer a programnyelvben, amely a perifériákkal történő kommunikációért felelős. Az I/O során adatok mozognak a tár és a periféria között.

C-ben I/O eszközrendszer nem része a nyelvnek, könyvtári függvények állnak rendelkezésre. Az I/O függvények minimálisan egy karakter vagy karaktercsoport, illetve egy bájt vagy bájtcsoport írását és olvasását teszik lehetővé.

A kivételek olyan események, amelyek megszakítást okoznak. A kivételkezelés az a tevékenység, amelyet a program végez. Egyes kivételek figyelése letiltatható vagy engedélyezhető. A kivételeknek általában van neve (eseményhez kapcsolódó üzenet) és kódja (egész szám).

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Annak a módja, hogy egy programozási nyelvet elsajátítsunk, nem más mint hogy programokat írunk az adott nyelven. Kezdjük a legegyszerűbb programmal a Figyelem, emberek! szavak kiírásával. Fordítjuk: cc figyel.c, majd futtatjuk: ./a.out. A C programok tetszőleges nevű függvényeket tartalmaznak amelyek a számítási műveleteket határozzák meg. Speciális függvény a main (főprogram) mely minden programban elő kell forduljon. Itt hívjuk meg az előre megírt függvényeket, vagy itt írjuk meg. A program olyan sorrendben végzi el az utasításokat ahogy azok a main-be vannak. A függvény neve után szereplő () az argumentum listát tartalmazza, a {} pedig az utasítás listát (lehetnek üresek is). A printf("Figyelem emberek!\n") függvényhívás a kimenetbe írja a Figyelem emberek szöveget, a végére pedig egy sortörést.

Egysoros kommentet a // -el, többsorosat pedig a /* */ jelek segítségével hozunk létre. Az int az egész, a float a lebegő, char a karakter, típusú váltózókat jelöli. Sorok végét ; -vel zárjuk le. A while ciklus működése: akkor áll le, amikor a feltétel hamissá válik. A printf -ben lévő % -al meg tudjuk adni, hogy a kiírandó szám milyen típusú lesz.

A for ciklus sokban hasonlít a while -hoz, akkor lép ki a ciklusból, ha a feltétel hamissá válik.

A #define segítségével a program elején szimbolikus állandót vagy változót adunk egy karakterlánchoz.

A getchar() függvény híváskor karaktert olvas be, a putchar() hívásakor pedig karaktert ír a kimenetre.

10.3. Programozás

[BMECPP]

Bevezetés: A C++ lehetővé teszi az objektmoriemtált és generikus programozást. A C++ -t Bjarne Stroustrup fejlesztette ki. A C++ a C nyelvre alapszik. A C a C++ részhalmazának tekinthető.

C -ben a függvényhívások esetén a paraméterlistát üresen is hagyhatjuk. A C++ esetén a üres paraméterlisták megadunk egy void paramétert (a zárójelek közé).

C++ -ban kétfajta main függvénymegadás van: int main(), illetve int main (int argc, char* argv[]), ahol az argc a parancssorban megadott parancs argumentumainak száma, az argv pedig maga a parancs argumentumait kapja meg.

A bool típus a logikai igaz vagy hamis (true vagy false) értékeket ad. A C -ben ez a bool még nincs, helyette az int vagy az enum típust használjuk erre a célra.

A C++ már header fájlok terén is bővült, illetve a C -ben lévő wchar típus a C++ -ban beépített típus lett.

A C++ -ban megjelent újítás, ami az hogy cílusok feltételei megadása közben illetve a függvények paramétereinek magadásakor is definiálhatunk változókat.

C++ -ban lehetőségünk van úgyanolyan nevű függvényeket létrehozni, abban az esetben ha az argumentumai különböznek.

Ha van egy olyan függvénydekláráció, hogy void f(int i) és a mainben f(a) -val hívjuk meg akkor az i szimbólisan az a lemasolt értékére fog hivathozni. A következő példában az átadott érték az a változó címe lesz, egy pointer, melyre a függvényben a pi szimbólummal hivatkozunk, a * operátorral tudunk hozzáérni az a változóhoz és változtatni is azon. Létezik cím- és értékszerinti paraméterátadás.

Az osztály alapelve az egységbázis, mivel egy adott dolog struktúrájának leírását foglalja magába. Az osztályoknak lehetnek példányai, önálló egyedi amiket objektumoknak nevezünk. Az osztályok esetén tudjuk biztosítani, hogy az objektumok tulajdonságaihoz a program többi része ne férjen hozzá, ne tudja változtani azokat, ezt a védekezést adatrejtésnek hívjuk. A példákon keresztül láthatjuk hogy a struktúrák esetében nemcsak tagváltozói hanem tagfüggvényei is lehetnek. A this poniterrel tagváltozókra tudunk hivatkozni.

Adatrejtés esetén a private kulcsszót hívjuk segítségül, melyet a struktúra vagy osztály definíciójába írunk. A private: után felsorolt tagváltozók és függvények csak az osztályon belül lesznek láthatóak majd. A C++ esetében érdemesebb az ostályt használni a struktúra helyett.

A konstruktur olyan specilis tagfüggvény melynek neve megegyezik az osztály nevével és automatikusan meghívódik amikor példányosítjuk az osztályt. Ha nem írunk mi magunk konstruktort az osztályba, attól

még van ott egy alapértelmezett konstruktor ami nem csinál semmit. Tehát a konstruktor végzi el az inicializálást, ezzel ellentétben pedig van a destruktör ami pedig a felszabadításra szolgál. A destruktort a hullám jellel való kezdésről könnyű felismerni, melyet az osztály neve követ. A destruktör akkor hívódik meg amikor megszűnik az objektum.

A dinamikus memoriakezelésre C-ben a malloc és a free függvények állnak rendelkezésünkre. C++-ban viszont már nem is függvényeket hanem operátorokat használunk ezek helyett, ez pedig a new és a delete. A new a lefoglalt típusra mutató pointerrel tér vissza, használat után ezt a delete-el szabadítjuk fel. Tömbök lefoglalása és felszabadítása esetén úgyanígy kell eljárni hozzájárulásra a tömb "jelet": []. Dinamikus adattagok térolása esetén megvalósítható a FIFO, ahol ha őt elemet adunk hozzá akkor meg kell növelnünk a dinamikus területet, amikor meg kiveszünk, akkor egyel csökkentjük. A másolókonstruktur esetén az új objektumot egy már meglévő alapján inicializáljuk, másolatot akarunk látrehozni.

A sablonok alatt olyan osztálysablonokat és függvénysablonokat értünk, amelyek deklarációja esetén bizonyos elemeket paraméterként kezelünk. A paramétereket explicit vagy implicit módon adhatjuk meg. A függvénysablonok deklarálását a template kulcsszóval kezdjük, majd kacsacsőrök között felsoroljuk a sablonparamétereiket (pl class T vagy name T vagy type T) vesszővel elválasztva.

A hagyományos hibakezelés továbbfejlesztése a kivételkezelések, melyekkel sokkal átláthatóbbá tudjuk tenni a programunkat. C++-ban a kivételkezelést try-catch blokkal oldjuk meg (példa a 190-es oldal).

DRAFT

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Berners-Lee!

11.1. Java 2 útikalauz programozóknak 5.0 I-II. és C++ összehasonlítás

A Java és C++ között vannak hasonlóságok és eltérések. Sok C++ nyelven írt kifejezés szintaktikailag helyes Javaban is, de ami hasonlít az nem minden azonos. A C++ nyelv bővebb a Java nyelvnél, azomban az szabványos osztálykönyvtárak révén a Java nyelvet szélesebb alkalmazásban lehet használni. Támogatja a grafikus felhasználói felület programozását, hálózati programozást, virtuális gépek köötti osztott objektumelérést és még sok más is. Mindez elérhető C++ nyelven is csak ott külső könyvtárat kell segítségül hívni.

A C++ nyelven írt forráskódok hordozhatók, amik az új gépen újra fordítva és szerkesztve futni fog, a bináris kód viszont nem, mert az tartalmaz a helyi operációs rendszerre és hardverre vonatkozó feltételeket. Ezzel ellentétben a Java hordozható bináris formában is.

A C++ nyelv olyan bináris kódot igyekszik elő állítani a leghatékonyabban működik futási szempontból az adott könyezetben. A Javaban viszont minden függvény virtuális. A osztályok, objektumok azonoítása és a változok inicializálása is futási időben történik és folyamatosan építünk a kivétkelésre.

A Java kevésbé függ a platformtól, kevesebbet bíz az implementációra mint a C vagy C++. Javaban az argumentumok kiértékelése balról jobbra történik és pontosan a meghatározott sorrendben történik. A Java fordító olyan dolgokat is ellenőriz amire a C vagy C++ fordító csak warningot, azaz figyelmeztést.

A legnagyobb különbözet a két nyelv között az objektummodellek közötti adódnak. A C++ nyelvben az objektumok a memoriában egymást követő bájtsorozatként fogadja el, ezeket tudjuk manipulálni a mutókkal ha figyelmen kívül hagyjuk a fordító program általi figyeletet. Ezzel ellentétben a Java mivel virtuálisan fut nem fér közvetlenül hozzá a memoriához. A Java az osztályokat class-fájlá fordul, amit és indításkor ezeket ellenőrzi. A class-fájl formátuma szabványos.

A Java megkülböztet objektumokat, amiket hivatkozásokon keresztül érünk el és primitív típusokat. Az objektumhivatkozások és primitív adatok lehetnek statikus vagy automatikus adatterületen de az objektumok nem. A Java az objektumokat egy véletlenszerű terüleuten tárolja. A Java nem tartalmaz destruktur meghívó mechanizmusn a C++-al ellenértében.

C++-ban az objektum élettartama pontosan ismert, ezeket mi határozzuk meg a new és delete operátorokkal, míg Javaban nincs lehetősége a programozónak a memória felszabadításra. Javaban nincs érvénytelen

hivatkozás, amíg ez C++ alatt előfordulhat ha az objektumot már töröltük. Java nyelven ameddig van referenciaink a objektumra addig biztos létezik.

11.2. Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)

A Python nyelvet 1990-ban alkották meg amely egy magas szintű, dinamikus, objektumorientált és platformfüggetlen nyelv. Leginkább prototípusok készítésére és tesztelésére használják. Egy Python komponens együtt tud működni más nyelven írt modulokkal. A Python egy szkriptnyelv. Támogat magas szintű típusokat is mint például a listák és szótárak, így könnyítve a fejlesztéseket.

A Pythonban nincs szükség a fordítási fázisra, úgy mint C, C++ vagy Java nyelven, elegendő megdani a Python forrást amit az értelmező autómatikusan futtat. A Python értelmezőprogram támogatja a következő platformokat: Windows, MacOS X, Unix, S60, iPhone, windows CE. A Python egyszerű használni, azonos jellegű alkalmazásokat egyszerűbb ezen a nyelven megírni mint C/C++ vagy Java nyelven.

A Python tartalmaz saját Python nyelven írt kódkönyvtárat. A Pythonban írt programok sokkal tömörebbek, mint Java, C, vagy C++ nyelven írtak, mivel összetett kifejezéseket írunk le egy rövid állításban, nincs szükség nyitó és záró jelekre és nem kell definiálni a változókat.

A programnak behúzás alapú a szyntaxa. Nincs szükség a kapcsos zárójelkre, ezeket "váltja fel" a behúzás. Első utasítás nem lehet behúzott. Pythonban a sor végéig tart az utasítás, nincs szökség a ';' jelre. A sor végi '\' -el tudjuk új sorba folytatni az utasítást. Megjegyzéseket a # jellel lehet tenni.

A értelmező minden sort tokenekre bont, ezek lehetnek: azonosító, kulcsszó, operátor, delimiter literál.

Pythonban minden adat objektum. Az adattípusok lehetnek: számok, sztringek, ennesek, listák és szótárak. Pythonban a változók alatt az egyes objektumokra mutató referenciát értjük és ezekre tudunk hivatkozni. Pythonba is létezik lokális és globalis változó.

A print segítségével írhatunk a konzolra. Használható a más nyelvekben is ismert 'if'. Támogatja a ciklusok kezelését.

A 'def' kulcsszóval definiálhatjuk a függvényeket. A függvények rendelkeznek paraméterekkel, amiknek adhatunk alapértelmezett értéket is. Függvényhíváskor az argumentumokat megadhatjuk a megszokott módon ahogy a függvényhívásban van vagy közvetlenül az egyes argumentumoknak is adhatunk értékt.

A Python támogatja az objektumorientált fejlesztést. Definiálhatunk osztályokat amelyek példányai az objektumok. Az osztályok örökölhetnek más osztályoktól is.

Fejlesztér érdekében a Python sok szabványos modult tartalmaz, ehhez adodnak a mobilkészülékekhez létrehoztak, mint az 'appuifw' ami támofatja a felhasználói felület kezelését. A 'messaging' ami az SMS és MMS üzenetek kezelését segíti. A 'camera' modul a kamerával kapcsolatos dolgokban segít.

A Python is támogatja a kivételkezelést. A 'try' kulcsszó után kerül a kód rész ahol a kivételek helyzet elő állhat. Az 'except' blokk amelyre hiba esetén kerül a vezérlé.

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 UDPROG repó: source/labor/polargen)

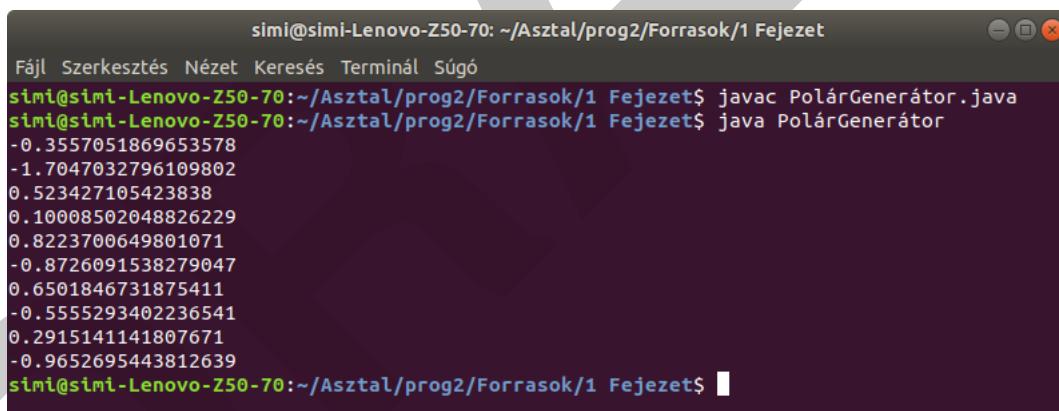
Megoldás forrása: [./Forrasok/12.Fejezet/PolárGenerátor.java](#)

```
public class PolárGenerátor {  
  
    boolean nincsTárolt = true;  
    double tárolt;  
  
    public PolárGenerátor() {  
        nincsTárolt = true;  
    }  
  
    public double következő() {  
        if (nincsTárolt) {  
            double u1, u2, v1, v2, w;  
            do {  
                u1 = Math.random();  
                u2 = Math.random();  
                v1 = 2 * u1 - 1;  
                v2 = 2 * u2 - 1;  
                w = v1 * v1 + v2 * v2;  
            } while (w>1);  
            double r = Math.sqrt((-2 * Math.log(w)) / w);  
            tárolt = r * v2;  
            nincsTárolt = !nincsTárolt;  
        }  
    }  
}
```

```
        return r * v1;
    } else {
        nincsTárolt = !nincsTárolt;
        return tárolt;
    }
}

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

A programunkban található nincsTárolt nevű változó értéke ha false akkor van már értéke a tárolt nevű lebegőpontos változonak vagyis generáltunk már számot, ha true akkor még nem kapott értéket. A következő nevű metódusban állítjuk elő a random számokat egy matematika képletet segítségül hívva. A program abban az esetben ha még nincs előállított számunk, a nincsTarolt értékről hamis akkor a metódus le generál két számot, az egyiket elmenti a tarolt változóba a másikat visszaadja. Ha már volt tárolt számunk az else ág a nincsTarolt logikai értékét negáljára állítja. A main függvényben a kiiratás történik, a mi esetünkben 10 számot generál a program.



```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/1 Fejezet
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ javac PolárGenerátor.java
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ java PolárGenerátor
-0.3557051869653578
-1.7047032796109802
0.523427105423838
0.10008502048826229
0.8223700649801071
-0.8726091538279047
0.6501846731875411
-0.5555293402236541
0.2915141141807671
-0.9652695443812639
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$
```

12.1. ábra. Eredmény

Ugyan ezt a feladatot láthatjuk C++ban, az első ami feltűnik az hogy a program terjedelmesebb és nehezebben átlátható, mint a fenti Java.

Megoldás forrása: [./Forrasok/12.Fejezet/Polargen.cpp](#)

```
#ifndef POLARGEN__H
#define POLARGEN__H

#include <cstdlib>
#include <cmath>
```

```
#include <ctime>

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

private:
    bool nincsTarolt;
    double tarolt;

};

#endif
```

Fenetbb a program header része, #ifdnef-fel leelenőrizzük hogy létezik e már ilyen nevű header fájl, ha nem akkor a #define parancsal létrehozzuk. Ezek mllet a már Javabol ismert változókat deklaráljuk és értéket adunk nekik.

```
#include "polargen.h"

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 * u2 - 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;
}
}
```

Ebben a programban használjuk fel az előbb létrehozott header fájlt. A kovetkezo nevű metodus a header fájlban deklarált PolarGen tulajdonságait veszi át és ugyan úgy működik mint a már be mutattott Java.

```
#include <iostream>
#include "polargen.h"

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

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

    return 0;
}
```

A főprogramunkban meghíjuk a már megírt függvényünket és segítségével elő állítjuk az értékeket.

A következő képen egy OpenJDK, OracleJDKban írt kódrész látható. Itt párhuzamosítva van, több szálon fogja meghívni ugyan azt a metódust ugyan arra az objektumra, ezt jelzi a synchronized szó. A StrictMath-re azért van szükség mert a porgramban használunk gyökvonást és logaritmus függvényt és mindenellet sokkal pontosabb.

```

 * @return the next pseudorandom, Gaussian ("normally") distributed
 *         {@code double} value with mean {@code 0.0} and
 *         standard deviation {@code 1.0} from this random number
 *         generator's sequence
 */
synchronized public double nextGaussian() {
    // See Knuth, ACP, Section 3.4.1 Algorithm C.
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1; // between -1 and 1
            v2 = 2 * nextDouble() - 1; // between -1 and 1
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}

```

12.2. ábra. Eredmény

12.2. Homokózó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden másról 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 Servlethez és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

12.3. „Gagyi”

Az ismert formális „while ($x \leq t \&& x \geq t \&& t \neq x$);” tesztkérdéstípusra adj a szokásosnál (miszerint x , t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referencia) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x , t értékkal meg nem! A példát építsd a JDK Integer.java forrására 3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás videó: [./Forrasok/12.Fejezet/Gagyi](#)

-128-tól 128-ig a JDK létrehozza az Integer objektumokat a poolba. Ha az érték nem ebből a tartomániból van, akkor a két értéknek külön objektum lesz létre hozva és nincs vége a ciklusnak mivel $t \neq x$.

```

public class Gagyil {
    public static void main (String[] args)
    {

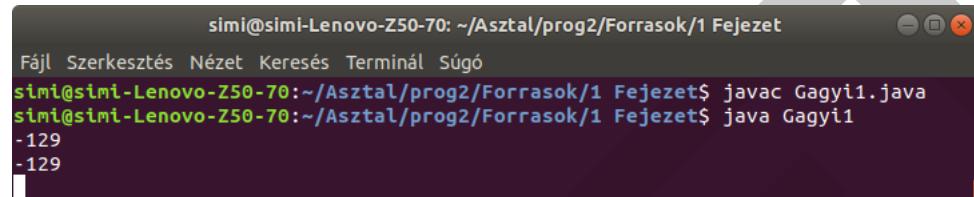
        Integer x = -129;
        Integer t = -129;
    }
}

```

```
System.out.println (x);
System.out.println (t);

while (x <= t && x >= t && t != x);

}
```



```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/1 Fejezet
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ javac Gagy1.java
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ java Gagy1
-129
-129
```

12.3. ábra. Eredmény

Java nyelven ha kis Integer számokkal dolgozunk erre van egy ugynevezett pool, amikre nem feltétlenül kell új objektumokat létrehozni hanem innen gyorsan kivhető egy kész objektum. Ha a poolbol ki vett Integer memóriacímeit akarjuk összehasonlítani, akkor ezek egyenlőek lesznek. A poolban -128 és 127 közötti értékek vannak.

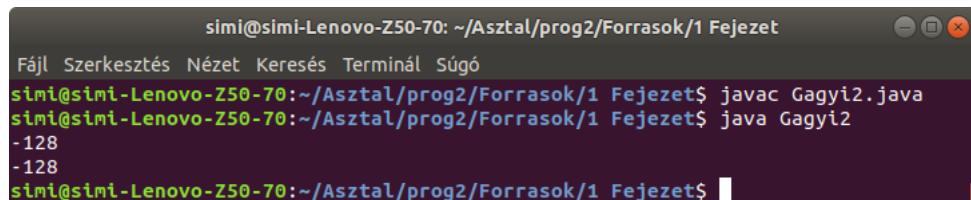
```
public class Gagy1 {
    public static void main (String[] args)
    {

        Integer x = -128;
        Integer t = -128;

        System.out.println (x);
        System.out.println (t);

        while (x <= t && x >= t && t != x);

    }
}
```



```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/1 Fejezet$ javac Gagyi2.java
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ java Gagyi2
-128
-128
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$
```

12.4. ábra. Eredmény

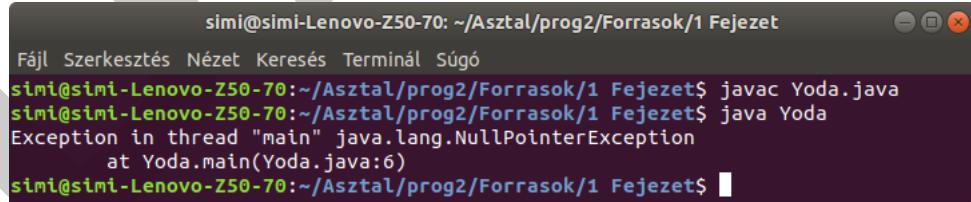
12.4. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-t leáll, ha nem követjük a Yoda conditions-t! https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás videó: [../Forrasok/12.Fejezet/Yoda.java](#)

Az s.equals("bármi") nem követi a Yoda conditionst. Ha az s sztringnek a null értéket adjuk akkor NullPointerException t kapunk mert az nem Yoda condition és leáll a program. Ha s értéke a bármi karakter sor lenne a program egyezik válszt adna. A Yoda condition nem követi a kifejezések megszokott sorrendjét.

```
public class Yoda
{
    public static void main (String[] args)
    {
        String s = null;
        if(s.equals("bármi"))
        {
            System.out.println ("egyezik");
        }
    }
}
```

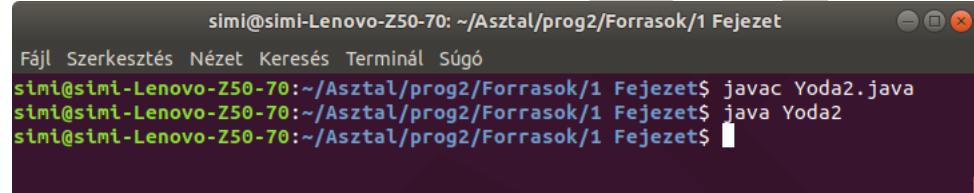


```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/1 Fejezet$ javac Yoda.java
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ java Yoda
Exception in thread "main" java.lang.NullPointerException
        at Yoda.main(Yoda.java:6)
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$
```

12.5. ábra. Eredmény

Ha a "bármi"==s formában adnánk meg akkor e képpen követné hibaüzenet nélkül le fut a program., viszont nem írja ki semmit mivel a bármi továbbra se lesz egyenlő a null kezdeti értékkel.

```
public class Yoda2
{
    public static void main (String[]args)
    {
        String s = null;
        if("bármí"==s)
        {
            System.out.println ("egyezik");
        }
    }
}
```



The screenshot shows a terminal window titled "simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/1 Fejezet". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ javac Yoda2.java
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ java Yoda2
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$
```

12.6. ábra. Eredmény

12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/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).

..
./Forrasok/12.Fejezet/PiBBP.java

A BBP algoritmus Bailey-Borwein-Plouffe nevéhez köthető, aki 1995ben fejlesztette ezt ki és rá egy evre publikálta. A program kiszámolja pi-nek az n-edik számjegyét hexadecimálisan. A programban a $\{16^d \text{Pi}\}$ értéke egyenlő a $\{4 * \{16^d S1\} - 2 * \{16^d S4\} - \{16^d S5\} - \{16^d S6\}\}$ törtrészével. A main részben kiiratjuk hogy Pi 10000001 számjegye mi lesz.

```
/*
 * PiBBP.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A BBP (Bailey-Borwein-Plouffe) algoritmust a Pi hexa
```

```
* jegyeinek számolását végző osztály. A könnyebb olvahatóság
* kedvéért a változó és metódus neveket megpróbáltuk az algoritmust
* bemutató [BBP ALGORITMUS] David H. Bailey: The BBP Algorithm for Pi.
* cikk jelöléseihez.
*
* @author Bátfa Norbert, nbatfai@inf.unideb.hu
* @version 0.0.1
*/
public class PiBBP {
    /** A Pi hexa kifejtésében a d+1. hexa jegytől néhány hexa jegy.*/
    String d16PiHexaJegyek;
    /**
     * Létrehoz egy <code>PiBBP</code>, a BBP algoritmust a Pi-hez
     * alkalmazó objektumot. A [BBP ALGORITMUS] David H. Bailey: The
     * BBP Algorithm for Pi. alapján a
     * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
     * kiszámítása, a {} a törtrészt jelöli.
     *
     * @param d a Pi hexa kifejtésében a d+1. hexa jegytől
     * számoljuk a hexa jegyeket
     */
    public PiBBP(int d) {

        double d16Pi = 0.0d;

        double d16S1t = d16Sj(d, 1);
        double d16S4t = d16Sj(d, 4);
        double d16S5t = d16Sj(d, 5);
        double d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

        Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};

        while(d16Pi != 0.0d) {

            int jegy = (int)StrictMath.floor(16.0d*d16Pi);

            if(jegy<10)
                sb.append(jegy);
            else
                sb.append(hexaJegyek[jegy-10]);

            d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
        }
    }
}
```

```
d16PiHexaJegyek = sb.toString();  
}  
/**  
 * BBP algoritmus a Pi-hez, a [BBP ALGORITMUS] David H. Bailey: The  
 * BBP Algorithm for Pi. alapján a {16^d Sj} részlet kiszámítása.  
 *  
 * @param d a d+1. hexa jegytől számoljuk a hexa jegyeket  
 * @param j Sj indexe  
 */  
public double d16Sj(int d, int j) {  
  
    double d16Sj = 0.0d;  
  
    for(int k=0; k<=d; ++k)  
        d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);  
  
    /* (bekapcsolva a sorozat elején az első utáni jegyekben növeli pl.  
     * a pontosságot.)  
    for(int k=d+1; k<=2*d; ++k)  
        d16Sj += StrictMath.pow(16.0d, d-k) / (double)(8*k + j);  
    */  
  
    return d16Sj - StrictMath.floor(d16Sj);  
}  
/**  
 * Bináris hatványozás mod k, a 16^n mod k kiszámítása.  
 *  
 * @param n kitevő  
 * @param k modulus  
 */  
public long n16modk(int n, int k) {  
  
    int t = 1;  
    while(t <= n)  
        t *= 2;  
  
    long r = 1;  
  
    while(true) {  
  
        if(n >= t) {  
            r = (16*r) % k;  
            n = n - t;  
        }  
  
        t = t/2;  
  
        if(t < 1)  
            break;  
    }  
}
```

```
r = (r*r) % k;

}

return r;
}
/** 
 * A kiszámolt néhány hexa jegy visszaadása. A használt lebegőpontos
 * aritmentia függvényében mondjuk az első 6 pontos peldául
 * d=1000000 esetén.
 *
 * @return String a kiszámolt néhány hexa jegy
 */
public String toString() {

    return d16PiHexaJegyek;
}
/** Példányosít egy BBP algoritmust implementáló obektumot.*/
public static void main(String args[]) {
    System.out.println(new PiBBP(1000000));
}
}
```

```
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ javac PiBBP.java
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$ java PiBBP
6C65E5308
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/1 Fejezet$
```

12.7. ábra. Eredmény

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megséríti 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 UDPORG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Megoldás videó:

Megoldás forrása:

A Liskov elv, röviden LSP azt mondja ha T-nek az altípusa S, akkor ahol T-t használtuk ott S-t is felhasználhatjuk. Vagyis ha az ősosztály példányai helyett, annak valamelyik gyerek osztályát használom a program ugyan úgy fog működni. Az alábbi két kód erre fog példát mutatni.

Megoldás forrása: [./Forrasok/13.%20Fejezet/liskovsert.cpp](#)

```
class Madar {  
public:  
    virtual void repul() {};  
};  
  
class Program {  
public:  
    void fgv ( Madar &madar ) {  
        madar.repul();  
    }  
};  
  
class Sas : public Madar  
{};  
  
class Pingvin : public Madar  
{};
```

```
int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin );

}
```

Kezdetben definiálunk egy Madár osztályt, ez a definíció szerinti T, ahol meghatározunk egy repül függvényt. Ezt követi a Program osztály ami a Madár osztályal közösen lesz a definíció szerinti S, itt azt mondjuk hogy minden madár repül. Deklaráljuk a Sas és Pingvin osztályokat, ezek az osztályok Madár típusúak lesznek, aminek egy tulaldonsága az hogy a madarak repülnek. Így azt mondjuk a pingvinnek is hogy repüljön, aki ugye nem tud, ezzel sérül a Liskov elv.

Megoldás forrása: [..../Forrasok/13.%20Fejezet/liskovrafigyel.cpp](#)

```
class Madar {
//public;
//void repul() {};
};

class Program {
public:
    void fgv ( Madar &madar ) {
    }
};

class RepuloMadar : public Madar {
public:
    virtual void repul() {};
};

class Sas : public RepuloMadar{
};

class Pingvin : public Madar {
};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );
```

```
Sas sas;  
program.fgv ( sas );  
  
Pingvin pingvin;  
program.fgv ( pingvin );  
  
}
```

Annak érdekébe, hogy az programunk a pingvineket ne akarja reptetni a Madar osztályban nem mondjuk azt minden madárra hogy tud repülni. Létrehozunk egy RepuloMadar osztályt és ezen belül határozzuk meg a repul() metódust. Így a sasnak megadjuk az új osztályunkat adjuk neki, hogy tudjanak repülni, a pingvinek pedig csak a Madar osztályba kerülnek. A főprogramban nincs szükség változtatásra, de a pingvineknek megse kell repülniük.

13.2. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetők! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

Megoldás videó:

Megoldás forrása:

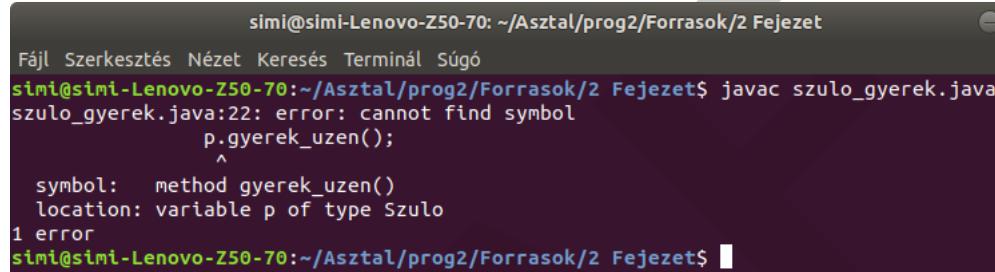
A következő Java, valamint C++ programokon keresztül mutatom be hogy az ősön keresztül csak az ős üzenetei küldhetők. Mindekkét program tartalmaz egy Szulo és Gyerek osztályt. A Gyerek a Szulo osztály leszármazottja, így a Gyerek a Szulo minden tulajdonságát örökli a saját tulajdonságai mellé. Ez által a Gyerek osztályon kereszül el tudjuk érni a Szulo-ben definiáltakat, de visszafele nem igaz, mivel a Szulo nem tudja a Gyerek azon új tuladonságait amelyeket nem öröklés útján tud. A főprogramban megprobálunk egy Szulo objektomat Gyerekként létrehozni hátha így ki tudjuk vele iratni a Gyerek üzenetét. De sikertelenül, erre adnak bizonyítékot a hibaüzenetek.

Megoldás forrása: [..//Forrasok/13.%20Fejezet/szulo_gyerek.java](#)

```
class Szulo  
{  
    public static void szulo_uzen()  
    {  
        System.out.println("Ez a szülő üzenete");  
    }  
}  
  
class Gyerek extends Szulo  
{  
    public static void gyerek_uzen()  
    {  
        System.out.println("Ez a gyerek üzenete");  
    }  
}
```

```
    }
}

public class szulo_gyerek
{
    public static void main(String[] args)
    {
        Szulo p = new Gyerek();
        p.gyerek_uzen();
    }
}
```



A screenshot of a terminal window titled "simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet". The command "javac szulo_gyerek.java" is run, resulting in the following error message:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ javac szulo_gyerek.java
szulo_gyerek.java:22: error: cannot find symbol
    p.gyerek_uzen();
           ^
      symbol:   method gyerek_uzen()
      location: variable p of type Szulo
1 error
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$
```

13.1. ábra. Eredmény

Megoldás forrása: [..../Forrasok/13.%20Fejezet/szulo_gyerek.cpp](#)

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

class Szulo {
public:
    void szulo_uzen()
    {
        cout << "Ez a szuló üzenete\n";
    }
};

class Gyerek : public Szulo {
    void gyerek_uzen()
    {
        cout << "Ez a gyerek üzenete\n";
    }
};

int main()
{
    Szulo * szulo = new Gyerek();
```

```
    szulo -> gyerek_uzen();  
}
```

```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ gcc szulo_gyerek.cpp -o szulo_gyerek  
szulo_gyerek.cpp: In function 'int main()':  
szulo_gyerek.cpp:22:11: error: 'class Szulo' has no member named 'gyerek_uzen'  
    szulo -> gyerek_uzen();  
               ^~~~~~  
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$
```

13.2. ábra. Eredmény

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>

Megoldás videó:

Megoldás forrása:

A következő 4 program ugyan azt a számolási feladato hajtja végre, külön magyarázatot nem irok mert a Java és C++ feladatok kidolgozása látható az első fejezet első feladatában. Mindegyik programot háromszorfuttatuk, futtatásról futtatásra növelve a for cikluson belül d vátozó értékét (ahogy az alábbi kódrészletekben látzik) úgy, hogy a elsőre 10^6 majd 10^7 végül 10^8 értéket kapja. A programot Ubuntu 18.04.3 LTS-en futtattam egy Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz alap órajelű gépen 8Gb rammal. Az értékek minden bizonnyal kisebbek is lehettek volna ha nem futottak volna más programok a háttérbe párhuzosan.

```
for (d = 1000000; d < 1000001; ++d)      // 10^6  
  
for (d = 10000000; d < 10000001; ++d)     10^7  
  
for (d = 100000000; d < 100000001; ++d)   10^8
```

Megoldás forrása: [./Forrasok/13.%20Fejezet/PiBBPBench.c](#)

```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ gcc PiBBPBench.c -o PiBBPBench -lm
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ ./PiBBPBench
6
2.047049
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ gcc PiBBPBench.c -o PiBBPBench -lm
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ ./PiBBPBench
7
24.180355
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ gcc PiBBPBench.c -o PiBBPBench -lm
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ ./PiBBPBench
12
276.738140
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$
```

13.3. ábra. C eredmény

Megoldás forrása: [..../Forrasok/13.%20Fejezet/PiBBPBench.cpp](#)

```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ g++ PiBBPBench.cpp -o PiBBPBench -lm
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ ./PiBBPBench
6
2.074407
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ g++ PiBBPBench.cpp -o PiBBPBench -lm
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ ./PiBBPBench
7
23.883579
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ g++ PiBBPBench.cpp -o PiBBPBench -lm
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ ./PiBBPBench
12
288.478542
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$
```

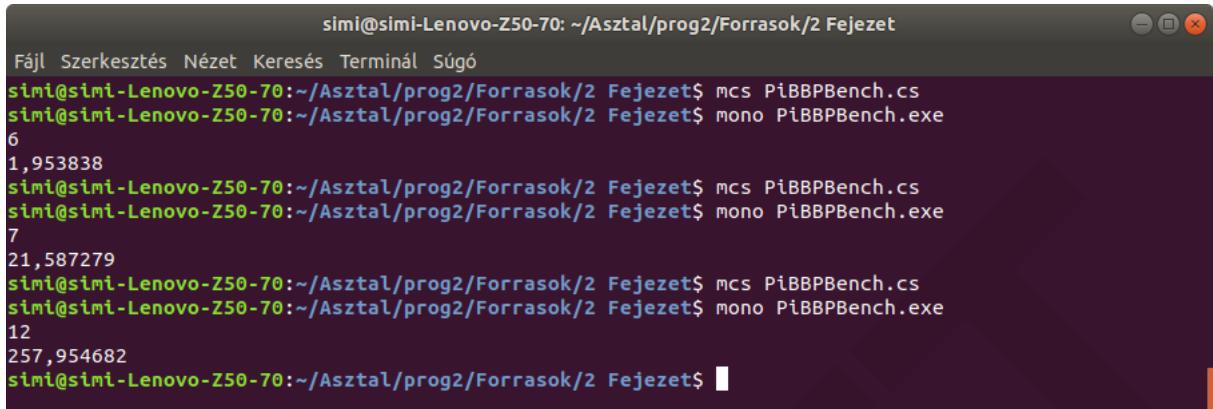
13.4. ábra. C++ eredmény

Megoldás forrása: [..../Forrasok/13.%20Fejezet/PiBBPBench.java](#)

```
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ javac PiBBPBench.java
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ java PiBBPBench
6
1.739
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ javac PiBBPBench.java
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ java PiBBPBench
7
20.558
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ javac PiBBPBench.java
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$ java PiBBPBench
12
242.175
simi@simi-Lenovo-Z50-70: ~/Asztal/prog2/Forrasok/2 Fejezet$
```

13.5. ábra. Java eredmény

Megoldás forrása: [./Forrasok/13.%20Fejezet/PiBBPBench.cs](#)



```
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ mcs PiBBPBench.cs
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ mono PiBBPBench.exe
6
1,953838
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ mcs PiBBPBench.cs
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ mono PiBBPBench.exe
7
21,587279
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ mcs PiBBPBench.cs
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$ mono PiBBPBench.exe
12
257,954682
simi@simi-Lenovo-Z50-70:~/Asztal/prog2/Forrasok/2 Fejezet$
```

13.6. ábra. C# eredmény

Itt látható az összesített eredmény, és mint látható a Java nyelven írt program a leggyorsabb.

	C	C++	Java	C#
10^6	2.047049	2.074407	1.739	1.953838
10^7	24.180355	23.883579	20.558	21.587279
10^8	276.73814	288.478542	242.175	257.954682

13.7. ábra. Összesített eredmény

13.4. deprecated - Hello, Android! (zöld feladat)

Élesszük fel a <https://github.com/nbatfai/SamuEntropy/tree/master/cs> projektjeit és vessünk össze néhány egymásra következőt, hogy hogyan változtak a források!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.5. 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.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.6. Hello, SMNIST for Humans! (piros feladat)

Fejleszd tovább az SMNIST for Humans projektet SMNIST for Anyone emberre szánt appá! Lásd az smnist2_kutatasi_jegyzokonyv.pdf-ben a részletesebb háttérét!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.7. 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)!

Megoldás videó:

Megoldás forrása:

A ciklomatikus komplexitást Thomas J. McCabe alkotta meg. A ciklomatikus komplexitás lényege hogy egy forráskódról megmonja annak komplexitását. A számítások a gráfelméletre alapszanak, a kód elágazásait a gráf pontjainak tekinti. és a köztük lévő élek alapján számol az alábbi képlettel.

$$M = E - N + 2P$$

E jelzi a gráfok eleinek száma, N a gráfban lévő csúcsok, P az összefüggő komponensek száma. A következő két kód funkciója csak annyi hogy ki számoljuk a komplexitásuk. Ennek kiszámításara Lizard nevű komplexitás számoló oldalt hívtam segítségül. Az oldalon nem kell mászt tenni, mint kiválsztani a program nyelvet, be másolni a programot aminek ki akarjuk számolni a komplexitását majd rá nyomni az Analyse gombra és meg is van az eredmény.

```
public class Days
{
    public static void main(String[] args)
    {
        int day = 5;
        if (day == 1)
        {
            System.out.println("Today is Monday");
        }
        else if (day == 2)
        {
            System.out.println("Today is Tuesday");
        }
        else if (day == 3)
        {
            System.out.println("Today is Wednesday");
        }
        else if (day == 4)
```

```
{  
    System.out.println("Today is Thursday");  
}  
else if (day == 5)  
{  
    System.out.println("Today is Friday");  
}  
else if (day == 6)  
{  
    System.out.println("Today is Saturday");  
}  
else if (day == 7)  
{  
    System.out.println("Today is Sunday");  
}  
}  
}
```

Try Lizard in Your Browser

.java

Analyse

```
else if (day == 6)  
{  
    System.out.println("Today is Saturday");  
}  
else if (day == 7)  
{  
    System.out.println("Today is Sunday");  
}  
}
```

Code analyzed successfully.

File Type .java

Token Count 147

NLOC 35

Function Name	NLOC	Complexity	Token #	Parameter #
Days::main	32	8	139	

13.8. ábra. Cikomatikus komplexitás Java

```
#include <iostream>  
using namespace std;  
  
int main()
```

```
{  
    int day = 5;  
    if (day == 1)  
    {  
        cout << "Today is Monday";  
    }  
    else if (day == 2)  
    {  
        cout << "Today is Tuesday";  
    }  
    else if (day == 3)  
    {  
        cout << "Today is Wednesday";  
    }  
    else if (day == 4)  
    {  
        cout << "Today is Thursday";  
    }  
    else if (day == 5)  
    {  
        cout << "Today is Friday";  
    }  
    else if (day == 6)  
    {  
        cout << "Today is Saturday";  
    }  
    else if (day == 7)  
    {  
        cout << "Today is Sunday";  
    }  
  
    return 0;  
}
```



Try Lizard in Your Browser

```
.cpp
{
    cout << "Today is Saturday";
}
else if (day == 7)
{
    cout << "Today is Sunday";
}

return 0;
}
```

Analyse

Code analyzed successfully.

File Type .cpp Token Count 117 NLOC 35

Function Name	NLOC	Complexity	Token #	Parameter #
main	33	8	110	

13.9. ábra. Cikomatikus komplexitás C++

14. fejezet

Helló, Mandelbrot!

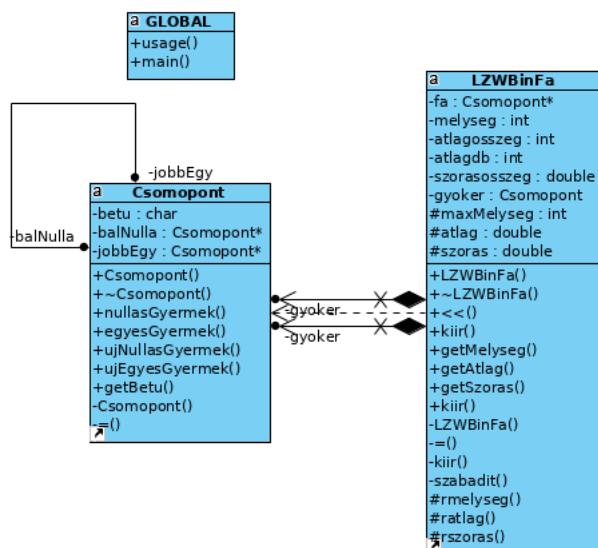
14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nIERIEOs, https://arato.inf.unideb.hu-batfai.norbert/UDPROG/deprecated/Prog1_6.pdf (28-32 fólia)

A feladatban szereplő forráskód:

Reverse engineering UML osztálydiagram azt jelenti hogy egy már létező kúdból állítunk elő osztálydiagrammot.

Osztálydiagramm generálásához sok program közül választhatunk, akár a fentiek közül is. Én viszont a Visual Paradigmot használtam, ez a programelérhető Windows é Linux operációs rendszerre egyaránt. Telepítéshez letöljük a programot a weboldalukról. Telepítés után a programban kiválasztjuk a Tool menüpontot, azon belül a Code fülre kattintunk, majd Instant Reverse. Itt kiválasztjuk hogy milyen nyelvű kúdbol csinálunk diagrammot. Jelen esetben az első védési feladatot használjuk, vagyis az LZWBinfa programot. Megadjuk az elérési útvonalat majd rá nyomunk az OK gombra és már el is kezdődik az osztálydiagramm készítés a C++ forrásból.

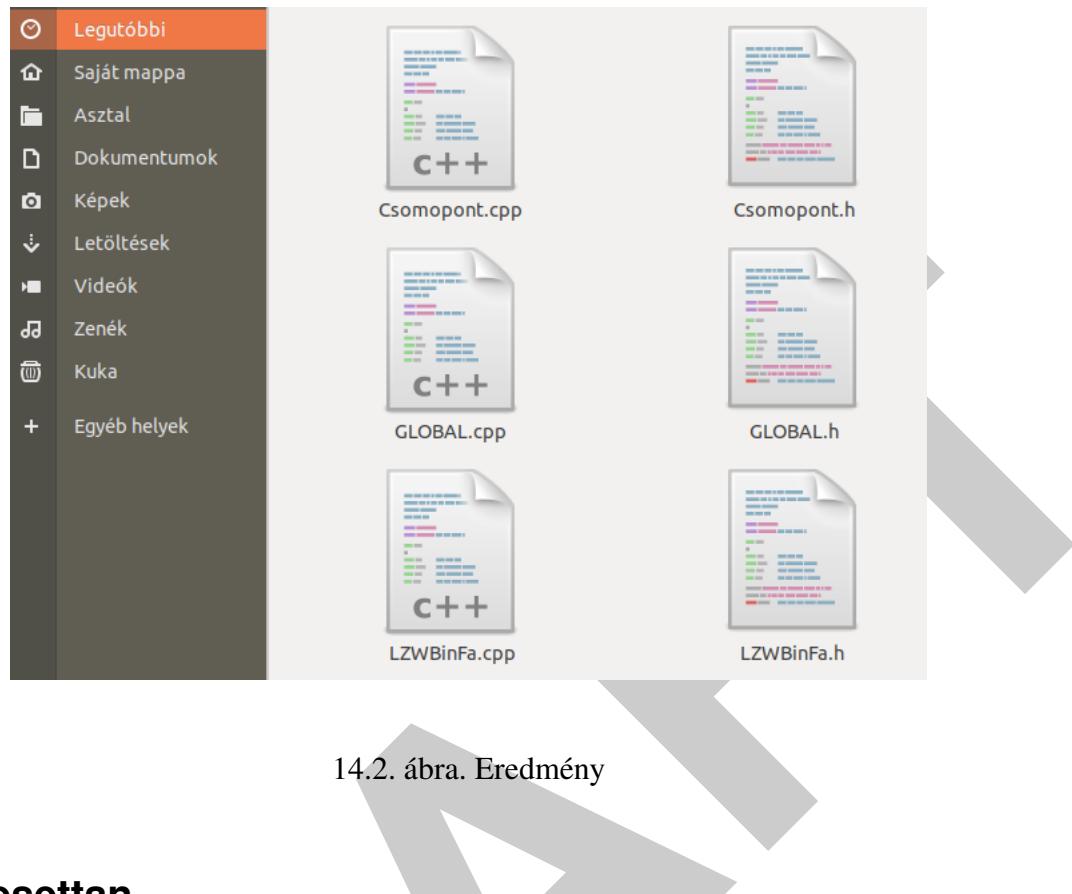


14.1. ábra. UML diagramm

14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Ebben a feladatban is a Visual Paradigm programot használtam. Itt a forward engineering modszert alkalmazzuk, ami annyit jelent hogy egy adott diagrammból kell előállítani a jozzá tartozó programot. A kezdet ugyan az, Tool menüpont, majd Code fül csak most Instant Generátorra lesz szükségünk, mivel ezzel tudunk UML diagrammot kóddá alakítani. Kiválasztjuk a nyelvet, megadjuk az output path-t, hogy hova mentse a generált kódokat majd Generate gombra kattintunk és készülnek is a kódok, amik az alábbiakban láthatók.



14.3. Egy esettan

A BME-s C++ tankönyv 14. fejezetét (427-444 elmélet, 445-469 az esettan) dolgozzuk fel!

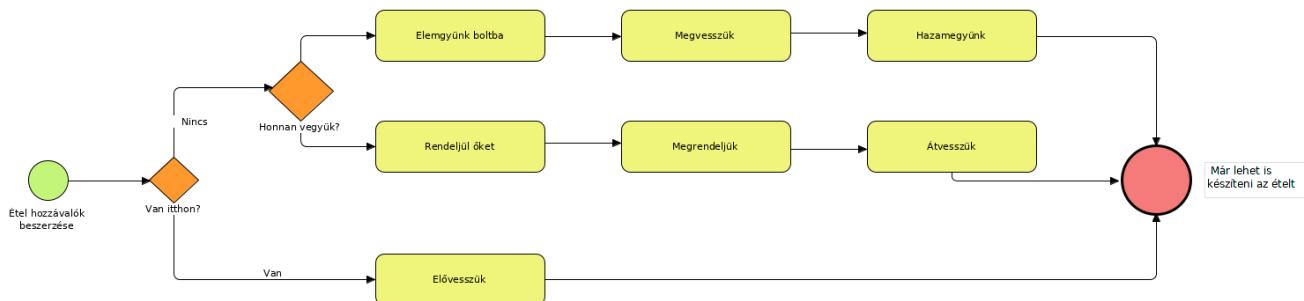
AZ UML (Unified Modeling Language) egy szabványos modellező nyelv a szotverfejlesztésben. Az UML leggyakoribb diagrammtípusa az osztálydiagramm, mely az osztályok, interfések és egyébb típusok kpcso-latát mutatja be. A diagrammok osszetevői dobozok, vonalak, ikonok és szöveges információk. UML-ben a legfontosabb elem az osztály, melyet egy háromba osztott télapalap jelöl. Felül van azosztály neve, aztán az osztály attributumai, alul pedig az osztály műveletei. Az utolsó két szakasz összefűlhető, de jobb az ha ott van. Egy attribútum szintaxisa: láthatóság név : típus multiplicitás = alapértelmezett_érték { tulajdon-ság }. A láthatóság lehet: + (public), - (private), # (protected), ~ (package). Ha a láthatóság után teszünk egy / jelet, akkor megadhatjuk a származtatott attribútumokat. A multiplicitás megadása: alsó határ..felső határ. Egy függvényt a követke- zőképpen tudunk leírin: láthatóság név (paraméterlista) : visszatéríté-si_érték { tulajdonságok }. Ezen belül a paraméterlista így alakulhat: irány(in, out, inout) név : típus = alapértelmezett_érték, legtöbbször az in irányt használjuk.

A kpcsolatok a modellben előként jelennek meg, így össze kötve a csomópontokat. Az asszociáció attributumait az asszociációs osztály jelöli, ezt egy szaggatott vonal jelöli. Kódban pedig két pointer jelöli amin keresztül az asszociáció bal és jobb oldali osztályát érjük el. Két speciális típusa van, az aggregáció (ez tartalmazást jelöl; jelölése: a tartalmazó oldalon egy üres rombusz) és a kompozíció (itt a tartalmazott objektum nem lehet megosztva, a tartalmazott és a tartalmazó együtt jönnek létre és együtt is szünnék meg; jelölése egy teli rombusszal történik).. Az UML modellező alkalmazások támogatják az átjárhatóságot, modellból képesek tudnak kódot generálni és fordítva. A kódgenerátorok minden fejlécet legenerálnak de a törzset üresen hagyják.

14.4. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/-Prog2_7.pdf (34-47 fólia)

A BPMN modell elkészítéséhez ismét a Visual Paradigm programot használtam. A menü sorból kiválasztjuk a Diagramm menüpontot, kiválasztjuk hogy Business Process Diagramm-ot akarunk csinálni és létrehozunk egy új projektet. A bemutatott feldat egy általános minden napi "probléma", mégpedig az hogy az ekészíteni kívánt ételhez hogy tujuk be szerezni a hozzávalókat, ha az szükséges. Ezt az alábbi módon ábrázoltam:



14.3. ábra. BPMN

Mint látható nem tűnik vészesnek ez a minden nap következő dolgunk, hogy ha éppen megvan otthon minden amire szükségünk van. A helyzet akkor bonyolódik (az ábra felső része), hogyha éppen nincsenek meg a hozzávalók. De még mindig van egy úgymond könnyebb megoldás amikor megrendenjük a hozzávalókat, igaz ez is mind időbe telik de nem szükséges elhagyni az otthonunk, ezek egy kényelmes megoldás. A "legrosszabb út" az hogyha nem rendelünk hanem mi magunk megyünk el a boltig.

14.5. BPEL Helló, Világ! - egy visszhang folyamat (zöld)

Egy visszhang folyamat megvalósítása az alábbi teljes „videó tutoriál” alapján: https://youtu.be/0OnIYWX2v_I

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.6. TeX UML

Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

15. fejezet

Helló, Chomsky!

15.1. Encoding

Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.2. OOCWC lexer

Izzítsuk be az OOCWC-t és vázoljuk a <https://github.com/nbatfai/robocar-emulator/blob/master/justine-rcemu/src/carlexer.ll> lexert és kapcsolását a programunk OO struktúrájába!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.3. I334d1c4⁵(zöld)

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet>(Ha ez első részben nem tettek meg, akkor írasd ki és magyarázd meg a használt struktúratömb memóriafoglalását!)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.4. Full screen(zöld)

Készítsünk egy teljes képernyős Java programot! Tipp: https://www.tankonyvtar.hu/en/tartalom/tkt/javatanitok-javat/ch03.html#labirintus_jatek

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.7. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! Lásd <https://youtu.be/XpBnR31BRJY>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16. fejezet

Helló, Stroustrup!

16.1. OO szemlélet

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.2. Homokózó

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.3. „Gagyi”

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.4. Yoda

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.5. Kódolás from scratch

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

17. fejezet

Helló, Gödel!

17.1. OO szemlélet

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.2. Homokózó

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.3. „Gagyi”

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.4. Yoda

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.5. Kódolás from scratch

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

18. fejezet

Helló, hetedikfejezet!

18.1. OO szemlélet

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.2. Homokózó

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.3. „Gagyi”

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.4. Yoda

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.5. Kódolás from scratch

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

19. fejezet

Helló, Schwarzenegger!

19.1. OO szemlélet

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.2. Homokózó

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.3. „Gagyi”

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.4. Yoda

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.5. Kódolás from scratch

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

20. fejezet

Helló, Calvin!

20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13/hello_sbol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.2. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a https://arato.inf.unideb.hu/batfai.norb_8_foliajat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.3. CIFAR-10 (zöld feladat)

Az alap feladat megoldása, +saját fotót is ismerjen fel, https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar_10_tf_tutorial_peldabol

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.4. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.5. SMNIST for Machines

Készíts saját modellt, vagy használj meglévőt, lásd: <https://arxiv.org/abs/1906.12213>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.6. SMNIST for Machines

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lásd pl.: <https://youtu.be/bAPSu3Rndl8>, https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innen_tol_mi, <https://bhaxor.blog.hu/2018/10/28/minecraft>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

DRAFT

20.7. Általános

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

20.8. C

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

20.9. C++

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

20.10. Lisp

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

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

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