

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 Â©s Erdei, Péter	2019. május 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
0.0.5 - 0.0.9	2019-02-19 - 2019-05-05	Heti csokrok elkészítése, error correction, pontosítások	nbatfai & erpeti97
0.1.0	2019-05-07	Végső simitások, ellenőrzés	erpeti97

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?	7
2.3. Változók értékének felcserélése	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun téTEL	11
2.8. A Monty Hall probléma	12
3. Helló, Chomsky!	13
3.1. Decimálisból unárisba átváltó Turing gép	13
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	13
3.3. Hivatalos nyelv	14
3.4. Saját lexikális elemző	16
3.5. l33t.l	16
3.6. A források olvasása	17
3.7. Logikus	18
3.8. Deklaráció	19

4. Helló, Caesar!	21
4.1. int *** háromszögmátrix avagy "legyen egy csillaggal kevesebb és double"	21
4.2. C EXOR titkosító	21
4.3. Java EXOR titkosító	22
4.4. C EXOR törő*	23
4.5. Neurális OR, AND és EXOR kapu	23
4.6. Hiba-visszaterjesztéses perceptron*	24
5. Helló, Mandelbrot!	25
5.1. A Mandelbrot halmaz	25
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	25
5.3. Biomorfok	26
5.4. A Mandelbrot halmaz CUDA megvalósítása	27
5.5. Mandelbrot nagyító és utazó C++ nyelven	28
5.6. Mandelbrot nagyító és utazó Java nyelven	28
6. Helló, Welch!	32
6.1. Első osztályom	32
6.2. LZW	35
6.3. Fabejárás	35
6.4. Tag a gyökér	38
6.5. Mutató a gyökér	39
6.6. Mozgató szemantika	40
7. Helló, Conway!	42
7.1. Hangyszimulációk	42
7.2. Java életjáték	43
7.3. Qt C++ életjáték	46
7.4. BrainB Benchmark	48
8. Helló, Schwarzenegger!	50
8.1. Szoftmax Py MNIST	50
8.2. Mély MNIST	52
8.3. Minecraft-MALMÖ	52

9. Helló, Chaitin!	53
9.1. Iteratív és rekurzív faktoriális Lisp-ben	53
9.2. Gimp Scheme Script-fu: króm effekt	53
9.3. Gimp Scheme Script-fu: név mandala	54
10. Helló, Gutenberg!	55
10.1. Programozási alapfogalmak(PICI)	55
10.2. Programozás bevezetés	57
10.3. Programozás	59
III. Második felvonás	61
11. Helló, Arroway!	63
11.1. A BPP algoritmus Java megvalósítása	63
11.2. Java osztályok a Pi-ben	63
IV. Irodalomjegyzék	64
11.3. Általános	65
11.4. C	65
11.5. C++	65
11.6. Lisp	65

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](#)]

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

1. fejezet

Vízió

Mi a programozás?

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

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 kódcsipelek! 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.

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.

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

Végtelen ciklus

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

Megoldás videó:

The screenshot shows a terminal window and a GitHub repository page side-by-side.

Terminal:

```
an7hr0x@an7hr0xPC:~/ProgJ/repo/bhax/thematic_tutorials/bhax_textbook/else
```

```
an7hr0x@an7hr0xPC:~/ProgJ/repo/bhax/thematic_tutorials/bhax_textbook/else$ uname -a
```

```
Broadcom Corporation BCM4709M [Cavium Networks Cavium Octeon] #1 SMP Tue Jul 10 10:45:24 UTC 2018 armv7l GNU/Linux
```

```
an7hr0x@an7hr0xPC:~/ProgJ/repo/bhax/thematic_tutorials/bhax_textbook/else$ gcc -v
```

```
an7hr0x@an7hr0xPC:~/ProgJ/repo/bhax/thematic_tutorials/bhax_textbook/else$ ./ciklusti
```

```
an7hr0x@an7hr0xPC:~/ProgJ/repo/bhax/thematic_tutorials/bhax_textbook/else$ ./ciklusti
```

GitHub Repository:

Chomsky/egyik lexesorr forrás
horcsik 4 days ago

Last commit Last update

Turing/BHAX

```
an7hr0x@an7hr0xPC:~
```

```
Top: 20:11:33 up 1:02, 0 users, load average: 0.76, 0.44, 0.15
```

```
Tasks: 200 total, 1 running, 156 sleeping, 0 stopped, 4 zombie
```

```
Kmem: 12.5 MiB, 0.0 MiB, 0.0 MiB, 0.0 MiB, 0.0 MiB, 0.0 MiB
```

```
Kib Mem: 16249468 total, 10495468 Free, 3488888 used, 2265120 buff/cache
```

```
Kib Swap: 12599896 total, 12599896 free, 0 used, 12599896 avail Mem
```

Task	Pid	User	State	Start Time	Time	Command
0	1	root	idle	0.000	0.000	/sbin/init
1	2	root	idle	0.000	0.000	sshd[1]
2	4	root	idle	0.000	0.000	dhclient[4]
3	7	root	idle	0.000	0.000	dhclient[7]
4	8	root	idle	0.000	0.000	dhclient[8]
5	9	root	idle	0.000	0.000	dhclient[9]
6	10	root	idle	0.000	0.000	dhclient[10]
7	11	root	idle	0.000	0.000	watchdog[11]
8	20	root	idle	0.000	0.000	sshd[20]
9	21	root	idle	0.000	0.000	sshd[21]
10	22	root	idle	0.000	0.000	sshd[22]
11	23	root	idle	0.000	0.000	sshd[23]
12	24	root	idle	0.000	0.000	sshd[24]
13	25	root	idle	0.000	0.000	sshd[25]
14	26	root	idle	0.000	0.000	sshd[26]
15	27	root	idle	0.000	0.000	sshd[27]
16	28	root	idle	0.000	0.000	sshd[28]
17	29	root	idle	0.000	0.000	sshd[29]
18	30	root	idle	0.000	0.000	sshd[30]
19	31	root	idle	0.000	0.000	sshd[31]
20	32	root	idle	0.000	0.000	sshd[32]
21	33	root	idle	0.000	0.000	sshd[33]
22	34	root	idle	0.000	0.000	sshd[34]
23	35	root	idle	0.000	0.000	sshd[35]
24	36	root	idle	0.000	0.000	sshd[36]
25	37	root	idle	0.000	0.000	sshd[37]
26	38	root	idle	0.000	0.000	sshd[38]
27	39	root	idle	0.000	0.000	sshd[39]
28	40	root	idle	0.000	0.000	sshd[40]
29	41	root	idle	0.000	0.000	sshd[41]
30	42	root	idle	0.000	0.000	sshd[42]
31	43	root	idle	0.000	0.000	sshd[43]
32	44	root	idle	0.000	0.000	sshd[44]
33	45	root	idle	0.000	0.000	sshd[45]
34	46	root	idle	0.000	0.000	sshd[46]
35	47	root	idle	0.000	0.000	sshd[47]
36	48	root	idle	0.000	0.000	sshd[48]
37	49	root	idle	0.000	0.000	sshd[49]
38	50	root	idle	0.000	0.000	sshd[50]
39	51	root	idle	0.000	0.000	sshd[51]
40	52	root	idle	0.000	0.000	sshd[52]
41	53	root	idle	0.000	0.000	sshd[53]
42	54	root	idle	0.000	0.000	sshd[54]
43	55	root	idle	0.000	0.000	sshd[55]
44	56	root	idle	0.000	0.000	sshd[56]
45	57	root	idle	0.000	0.000	sshd[57]
46	58	root	idle	0.000	0.000	sshd[58]
47	59	root	idle	0.000	0.000	sshd[59]
48	60	root	idle	0.000	0.000	sshd[60]
49	61	root	idle	0.000	0.000	sshd[61]
50	62	root	idle	0.000	0.000	sshd[62]
51	63	root	idle	0.000	0.000	sshd[63]
52	64	root	idle	0.000	0.000	sshd[64]
53	65	root	idle	0.000	0.000	sshd[65]
54	66	root	idle	0.000	0.000	sshd[66]
55	67	root	idle	0.000	0.000	sshd[67]
56	68	root	idle	0.000	0.000	sshd[68]
57	69	root	idle	0.000	0.000	sshd[69]
58	70	root	idle	0.000	0.000	sshd[70]
59	71	root	idle	0.000	0.000	sshd[71]
60	72	root	idle	0.000	0.000	sshd[72]
61	73	root	idle	0.000	0.000	sshd[73]
62	74	root	idle	0.000	0.000	sshd[74]
63	75	root	idle	0.000	0.000	sshd[75]
64	76	root	idle	0.000	0.000	sshd[76]
65	77	root	idle	0.000	0.000	sshd[77]
66	78	root	idle	0.000	0.000	sshd[78]
67	79	root	idle	0.000	0.000	sshd[79]
68	80	root	idle	0.000	0.000	sshd[80]
69	81	root	idle	0.000	0.000	sshd[81]
70	82	root	idle	0.000	0.000	sshd[82]
71	83	root	idle	0.000	0.000	sshd[83]
72	84	root	idle	0.000	0.000	sshd[84]
73	85	root	idle	0.000	0.000	sshd[85]
74	86	root	idle	0.000	0.000	sshd[86]
75	87	root	idle	0.000	0.000	sshd[87]
76	88	root	idle	0.000	0.000	sshd[88]
77	89	root	idle	0.000	0.000	sshd[89]
78	90	root	idle	0.000	0.000	sshd[90]
79	91	root	idle	0.000	0.000	sshd[91]
80	92	root	idle	0.000	0.000	sshd[92]
81	93	root	idle	0.000	0.000	sshd[93]
82	94	root	idle	0.000	0.000	sshd[94]
83	95	root	idle	0.000	0.000	sshd[95]
84	96	root	idle	0.000	0.000	sshd[96]
85	97	root	idle	0.000	0.000	sshd[97]
86	98	root	idle	0.000	0.000	sshd[98]
87	99	root	idle	0.000	0.000	sshd[99]
88	100	root	idle	0.000	0.000	sshd[100]
89	101	root	idle	0.000	0.000	sshd[101]
90	102	root	idle	0.000	0.000	sshd[102]
91	103	root	idle	0.000	0.000	sshd[103]
92	104	root	idle	0.000	0.000	sshd[104]
93	105	root	idle	0.000	0.000	sshd[105]
94	106	root	idle	0.000	0.000	sshd[106]
95	107	root	idle	0.000	0.000	sshd[107]
96	108	root	idle	0.000	0.000	sshd[108]
97	109	root	idle	0.000	0.000	sshd[109]
98	110	root	idle	0.000	0.000	sshd[110]
99	111	root	idle	0.000	0.000	sshd[111]
100	112	root	idle	0.000	0.000	sshd[112]
101	113	root	idle	0.000	0.000	sshd[113]
102	114	root	idle	0.000	0.000	sshd[114]
103	115	root	idle	0.000	0.000	sshd[115]
104	116	root	idle	0.000	0.000	sshd[116]
105	117	root	idle	0.000	0.000	sshd[117]
106	118	root	idle	0.000	0.000	sshd[118]
107	119	root	idle	0.000	0.000	sshd[119]
108	120	root	idle	0.000	0.000	sshd[120]
109	121	root	idle	0.000	0.000	sshd[121]
110	122	root	idle	0.000	0.000	sshd[122]
111	123	root	idle	0.000	0.000	sshd[123]
112	124	root	idle	0.000	0.000	sshd[124]
113	125	root	idle	0.000	0.000	sshd[125]
114	126	root	idle	0.000	0.000	sshd[126]
115	127	root	idle	0.000	0.000	sshd[127]
116	128	root	idle	0.000	0.000	sshd[128]
117	129	root	idle	0.000	0.000	sshd[129]
118	130	root	idle	0.000	0.000	sshd[130]
119	131	root	idle	0.000	0.000	sshd[131]
120	132	root	idle	0.000	0.000	sshd[132]
121	133	root	idle	0.000	0.000	sshd[133]
122	134	root	idle	0.000	0.000	sshd[134]
123	135	root	idle	0.000	0.000	sshd[135]
124	136	root	idle	0.000	0.000	sshd[136]
125	137	root	idle	0.000	0.000	sshd[137]
126	138	root	idle	0.000	0.000	sshd[138]
127	139	root	idle	0.000	0.000	sshd[139]
128	140	root	idle	0.000	0.000	sshd[140]
129	141	root	idle	0.000	0.000	sshd[141]
130	142	root	idle	0.000	0.000	sshd[142]
131	143	root	idle	0.000	0.000	sshd[143]
132	144	root	idle	0.000	0.000	sshd[144]
133	145	root	idle	0.000	0.000	sshd[145]
134	146	root	idle	0.000	0.000	sshd[146]
135	147	root	idle	0.000	0.000	sshd[147]
136	148	root	idle	0.000	0.000	sshd[148]
137	149	root	idle	0.000	0.000	sshd[149]
138	150	root	idle	0.000	0.000	sshd[150]
139	151	root	idle	0.000	0.000	sshd[151]
140	152	root	idle	0.000	0.000	sshd[152]
141	153	root	idle	0.000	0.000	sshd[153]
142	154	root	idle	0.000	0.000	sshd[154]
143	155	root	idle	0.000	0.000	sshd[155]
144	156	root	idle	0.000	0.000	sshd[156]
145	157	root	idle	0.000	0.000	sshd[157]
146	158	root	idle	0.000	0.000	sshd[158]
147	159	root	idle	0.000	0.000	sshd[159]
148	160	root	idle	0.000	0.000	sshd[160]
149	161	root	idle	0.000	0.000	sshd[161]
150	162	root	idle	0.000	0.000	sshd[162]
151	163	root	idle	0.000	0.000	sshd[163]
152	164	root	idle	0.000	0.000	sshd[164]
153	165	root	idle	0.000	0.000	sshd[165]
154	166	root	idle	0.000	0.000	sshd[166]
155	167	root	idle	0.000	0.000	sshd[167]
156	168	root	idle	0.000	0.000	sshd[168]
157	169	root	idle	0.000	0.000	sshd[169]
158	170	root	idle	0.000	0.000	sshd[170]
159	171	root	idle	0.000	0.000	sshd[171]
160	172	root	idle	0.000	0.000	sshd[172]
161	173	root	idle	0.000	0.000	sshd[173]
162	174	root	idle	0.000	0.000	sshd[174]
163	175	root	idle	0.000	0.000	sshd[175]
164	176	root	idle	0.000	0.000	sshd[176]
165	177	root	idle	0.000	0.000	sshd[177]
166	178	root	idle	0.000	0.000	sshd[178]
167	179	root	idle	0.000	0.000	sshd[179]
168	180	root	idle	0.000	0.000	sshd[180]
169	181	root	idle	0.000	0.000	sshd[181]
170	182	root	idle	0.000	0.000	sshd[182]
171	183	root	idle	0.000	0.000	sshd[183]
172	184	root	idle	0.000	0.000	sshd[184]
173	185	root	idle	0.000	0.000	sshd[185]
174	186	root	idle	0.000	0.000	sshd[186]
175	187	root	idle	0.000	0.000	sshd[187]
176	188	root	idle	0.000	0.000	sshd[188]
177	189	root	idle	0.000	0.000	sshd[189]
178	190	root	idle	0.000	0.000	sshd[190]
179	191	root	idle	0.000	0.000	sshd[191]
180	192	root	idle	0.000	0.000	sshd[192]
181	193	root	idle	0.000	0.000	sshd[193]
182	194	root	idle	0.000	0.000	sshd[194]
183	195	root	idle	0.000	0.000	sshd[195]
184	196	root	idle	0.000	0.000	sshd[196]
185	197	root	idle	0.000	0.000	sshd[197]
186	198	root	idle	0.000	0.000	sshd[198]
187	199	root	idle	0.000	0.000	sshd[199]
188	200	root	idle	0.000	0.000	sshd[200]
189	201	root	idle	0.000	0.000	sshd[201]
190	202	root</td				

Megoldás forrása: [Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/ciklus1.c](#), [Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/ciklus2.c](#), [Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/ciklus3.c](#)

Egy szálat több féle képpen lehet végtelen ciklusokkal 100%-on futtatni, én a példában a while(1); végtelen ciklust használom, de lehetne for(;;) ciklussal is csinálni mint ahogy a következő példám is mutatni fogja.

```
#include <stdio.h>

int main (){

    while(1);
```

```
    return 0;  
}
```

Ebben a példában egy szálat a for(;;) végtelen ciklus segítségével 0%-on fogom "pörgetni", mégpedig úgy, hogy a ciklusba beleteszünk egy sleep(1) parancsot, amely azt eredményezi hogy a ciklus végtelenszer 1mp-re sleep-be teszi a szálat, így 0% on marad amíg fut a programunk.

```
#include <unistd.h>  
int main () {  
  
    for (;;)   
        sleep(1);  
  
    return 0;  
}
```

Végül, OpenMP használatával elérhetjük azt is hogy minden szálunkat 100%-on futassuk, mivel az OpenMP szálankénti program futásra is használható.

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

Azt, hogy mely szálak hány %-on futnak, a **top** parancs segítségével ellenőrizhetjük.

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 dönteneni, 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.: Tehát ha létezne ilyen program akkor 50% ban tökéletesen működik, mert ha nem fagy le, akkor kiírja hogy nem fagyott le a program tehát azt hogy true. Viszont, ha a program lefagyós, akkor nem fog kiírni semmit, így nem kapunk false visszajelzést. De a feltételezések ből kiindulva ha nem kapunk outputot, se true-t se false-t akkor jön az a feltevés, hogy végtelen ciklusunk van, mert nem fut le a programunk.

Változók értékének felcserélése

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

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

Megoldás forrása: Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/valtozo1.c, Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/valtozo2.c

Tanulságok, tapasztalatok, magyarázat: A változók értékének felcserélésére sok módszer van talán a két legegyszerűbb, és amit mindenhol alapvetőként mutatnak a segédváltozó használatával és azok használata nélkül történő változó érték felcserélés. A segédváltozó a kettő közül is talán a könnyebbik, csak "felesleges" változót vezetünk be ennek a módszernek a használatával, és még a memóriában is több helyet foglal igy, ami a mai korban kis programokban nem probléma, mivel van bőven erőforrás. A segédváltozó nélküli módszerben először az egyik változó értékét hozzáadjuk a másik változóhoz, majd a kapott összegből kivonjuk a változatlan változó értékét és így az egyik csere meg is történt, majd végső lépésként kivonjuk a kapott összegből a kicserélt változó értékét, és ezek után fel is lett cserélve mindkettő változó.

```
#include <stdio.h>

int main() {

    int a = 11;
    int b = 22;

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

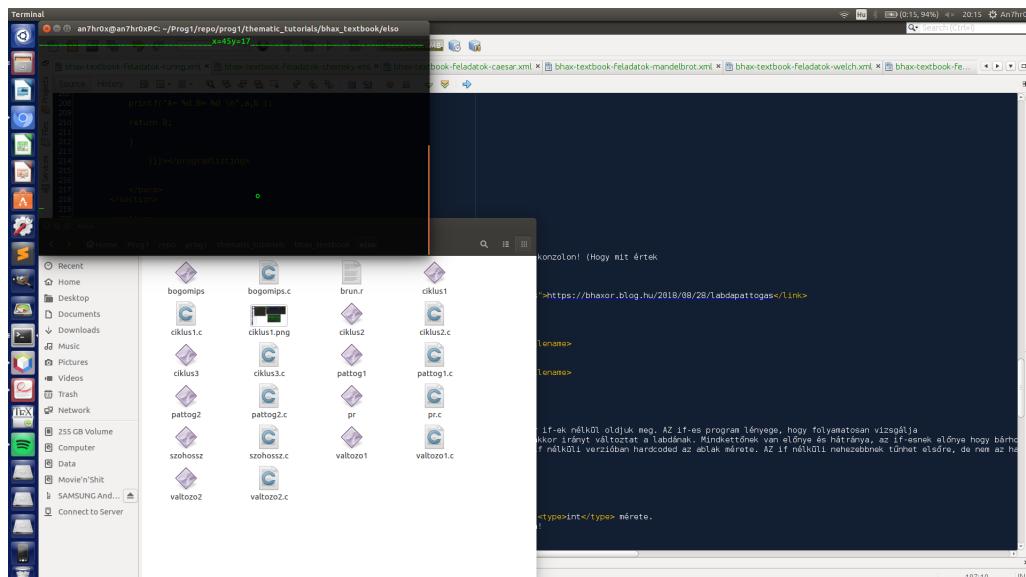
    printf("A= %d B= %d \n",a,b );
}
```

```
    return 0;  
  
}  
  
}
```

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 videókon.)

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



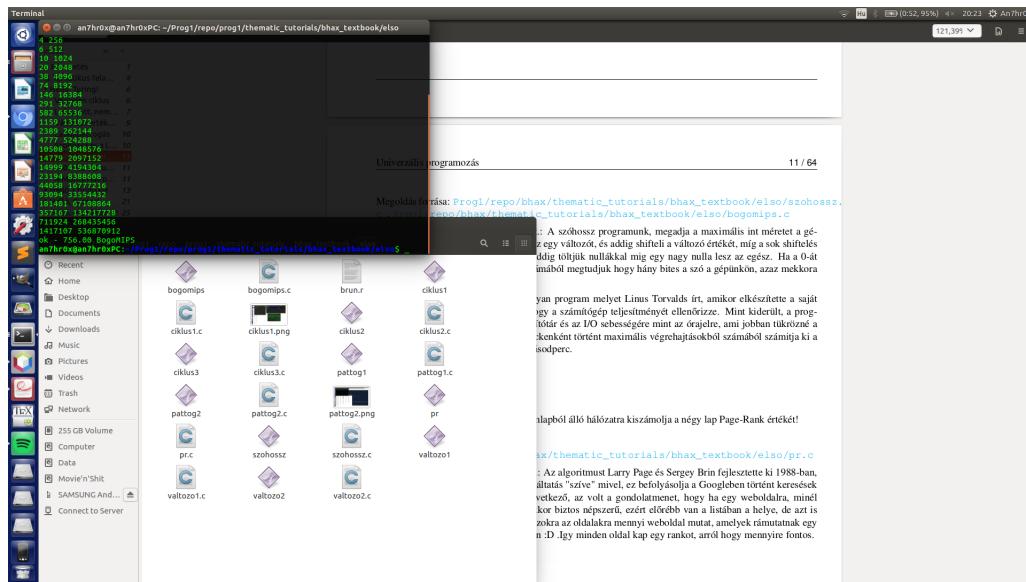
Megoldás forrása: [Progl/repo/bhax/thematic_tutorials/bhax_textbook/elso/pattog1.c](https://github.com/bhaxor/Progl/blob/master/repo/bhax/thematic_tutorials/bhax_textbook/elso/pattog1.c), [Progl/repo/bhax/thematic_tutorials/bhax_textbook/elso/pattog2.c](https://github.com/bhaxor/Progl/blob/master/repo/bhax/thematic_tutorials/bhax_textbook/elso/pattog2.c)

Tanulságok, tapasztalatok, magyarázat.: Két módszert fogunk megnézni, először az if-eket használót, és egy másikat amikor if-ek nélkül oldjuk meg. AZ if-es program lényege, hogy folyamatosan vizsgálja 4 if utasítás hogy a labda elérte-e az ablakunk valamelyik szélét, és ha elérte akkor irányt változtat a labdának. Mindkettennek van előnye és hátránya, az if-esnek előnye hogy bárhol nincs hardcode az ablak mérete. AZ if nélküli nehezebbnek tűnhet elsőre, de nem az ha átfutjuk az agunkban hogyan is tér el az if-estől.

Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás video:



Megoldás forrása: [Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/szohossz.c](https://github.com/bhax-prog1/prog1/blob/main/repo/bhax/thematic_tutorials/bhax_textbook/elszo/szohossz.c), [Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/bogomips.c](https://github.com/bhax-prog1/prog1/blob/main/repo/bhax/thematic_tutorials/bhax_textbook/elszo/bogomips.c)

Tanulságok, tapasztalatok, magyarázat: A szóhossz programunk, megadja a maximális int méretet a gépünkön azzal a módszerrel hogy felvesz egy változót, és addig shifteli a változó értékét, míg a sok shiftelés miatt az értéke nem lesz nulla, tehát addig töltjük nullákkal míg egy nagy nulla lesz az egész. Ha a 0-át elérte az értéke, akkor a shiftelések számából megtudjuk hogy hány bites a szó a gépünkön, azaz mekkora az int mérete.

Következő a BogoMIPS, mely egy olyan program melyet Linus Torvalds írt, amikor elkészítette a saját kerneljét. A feladata a programnak hogy a számítógép teljesítményét ellenőrizze. Mint kiderült, a program sajnos jobban hagyakszik a gyorsítótár és az I/O sebességére mint az órajelre, ami jobban tükrözne a tényleges teljesítményt. A program a tickenként történt maximális végrehajtásokból számából számítja ki a teljesítményt, melyre az időkorlát 1 másodperc.

Helló, Google!

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

Megoldás videó:

Megoldás forrása: [Prog1/repo/bhax/thematic_tutorials/bhax_textbook/elszo/pr.c](https://github.com/bhax-prog1/prog1/blob/main/repo/bhax/thematic_tutorials/bhax_textbook/elszo/pr.c)

Tanulságok, tapasztalatok, magyarázat: Az algoritmust Larry Page és Sergey Brin fejlesztette ki 1998-ban, ez az algoritmus a Google keresőszolgáltatás "szíve" mivel, ez befolyásolja a Googleben történt keresések során a találat listát. Működése a következő, az volt a gondolatmenet, hogy ha egy weboldalra, minél több másik weboldalról mutat link, akkor biztos népszerű, ezért előrébb van a listában a helye, de azt is figyelembe veszi az algoritmus, hogy azokra az oldalakra mennyi weboldal mutat, amelyek rámutatnak egy másikra, lényeg hogy rekurzió van jelen. Igy minden oldal kap egy rankot, arról hogy mennyire fontos.

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

Brun tétele, miszerint az ikerprímek (olyan prímpár, amelyek különbsége 2) reciprokösszege egy Brun-konstans néven ismert (B2-tal jelölt) véges értékhez konvergál. A tételt Viggo Brun bizonyította be 1919-ben, és jelentős fontossággal bír a szita eljárások terén. Érdekesség, hogy a Google a Nortel szabvány licitálásakor, a 1.902.160.540 konstans szerinti áron vették meg. Brun téTEL az ikerprímek egyik alapköve a matematikában.

A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall problémáról először talán a Numberphile youtubere csatornáján hallottam jó pár éve, azótais érdekesnek találom, és egyben lenyűgözőnek ezt a problémát. A probléma alapfeltevése, hogy egy játék showban vagyunk, ahol a műsorvezető felajánlja hogy válasszunk a 3 ajtó közül egyet. 2 ajtó mögött egy kecske van, és a maradék egy mögött meg egy új autó. Az első választás teljesen mindegy a probléma szempontjából, mert 1/3 esélyünk van a nyerésre. Miután kiválasztottuk, a műsorvezető kinyit egy ajtót ami mögött biztosan az egyik kecske van, és megkérde tőlünk, szeretnénk-e a döntésünket megváltoztatni, és a másik ajtót választani. Sokan úgy gondolják hogy még mindig 1/3 az esélye hogy nyerünk, de csak akkor, ha nem váltunk. Ha ajtót váltunk, már 2/3-lesz, tehát nagyobbak az esélyeink és érdemes váltani mindenkor. Az R szimuláció pont ezt a problémát modellez.

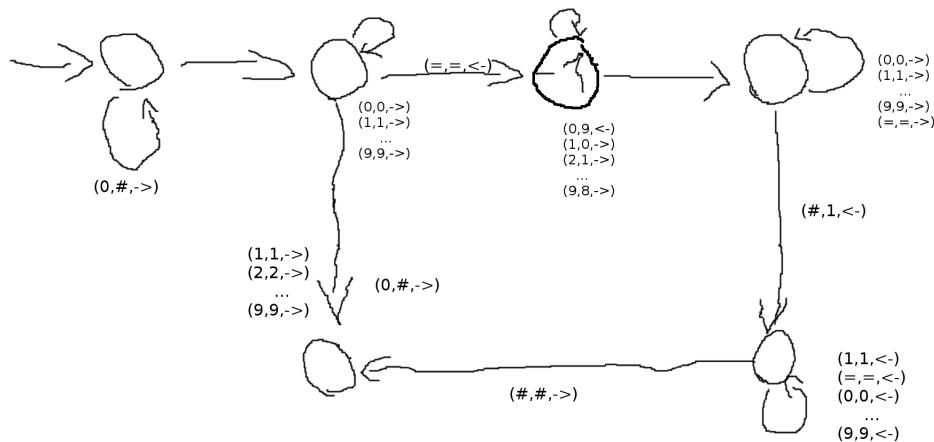
3. fejezet

Helló, Chomsky!

Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás video:



Megoldás forrása: Progl/repo/bhax/thematic_tutorials/bhax_textbook/masodik/unaris_unar.c

Tanulságok, tapasztalatok, magyarázat... Az unáris számrendszer szimbólumokon alapszik, legegyszerűbb számrendszer, és a kézenszámolás is tekinthető unáris számrendszernek. A számokat a szimbólumok darabszámaival ábrázoljuk, mint például a kezeinken az ujjak, vagy a programomban az 1-esekkel. A program, bekér egy számot, és for ciklus segítségével annyi 1-est ábrázol amennyi a szám. A turning gépes automata megtekinthető a rajzomon, ez az automata ha lefut, akkor egy decimális számot, unárisba átvált.

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:

Passzolt feladat.

Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiál 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: Prog1/repo/bhax/thematic_tutorials/bhax_textbook/masodik/verz/verz.c

Tanulságok, tapasztalatok, magyarázat... A könyv 13 féle utasítás típust tárgyal, ezek a következők: A kifejezéses utasítás, Aa összetett utasítás vagy blokk, A feltételes,do,while,for,switch,break,continue,return,goto címkézett, és a nulla utasítás.

A kifejezéses utasítás legtöbbször értékkedások vagy függvényhívások, alakjuk a

kifejezés;

As összetett utasítás vagy a blokkot ott használjuk ahol elvileg csak egy utasítás lenne elhelyezhető de ha szeretnénk több utasítást is használni akkor segít nekünk ez a típus. Alakjai:

összetett_utasítás:

{deklaraciólistaopc utasításlistaopc}

deklaraciólista:

deklaráció

deklaráció deklaraciólista

-utasításlista

-utasítás

-utasítás utasításlista

if (kifejezés)

 utasítás

if (kifejezés)

 utasítás

else

 (utasítás)

A feltételes utasításnál a gép minden esetben kiértékeli a kifejezést, és ha az érték nem nulla akkor az első utasítás kerül végrehajtásra, ha nulla, akkor meg a második kifejezés kerül végrehajtásra. Az else C ben úgy működik hogy utolsó else nélküli if-hez kapcsolódik.

A while utasítás addig ismétlődik amíg az értéke nem nulla, az érték vizsgálat minden végrehajtás előtt történik. Alakja:

```
while (kifejezés)
    utasítás
```

A do utasítás ugyanaz mint a while, csak az értékvizsgálat mindenkor az utasítás végrehajtás után történik. Alakja:

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

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

A for utasítás a következő a soron, az első kifejezés a ciklust inicializálja, a második azt a vizsgálatot határozza meg mely minden iterációt megelőz, és a vezérlés kilép a programból ha nullával válik egyenlővé, a harmadik kifejezés gyakrabban az egyet iterációk után történő inkrementálást határozza meg. Bármely kifejezés elhagyható.

A switch utasítás a kifejezés értékétől függően más utasításra vált át a vezérlés. Az eredménynek int típusnak kell lennie. A switchen belül bármelyik utasítás címkézhető egy vagy több

```
case állandó_kifejezés;
```

taggal. Kilépni a switchből a break utasítással lehet. Alakja:

```
switch (kifejezés)
utasítás
```

A break utasítás hatására befejeződik a breaket körülvevő legbelőle álló while, do, for vagy switch.

```
break;
```

A continue utasítás hatására az utasítást körülvevő legbelőle álló while, do, for utasítás a ciklusfolytatónak részére adódik át, vagyis a ciklus végére.

```
continue;
```

A return a függvény hívójához tér vissza.

```
return;
```

A goto utasítás a feltétel nélküli vezérlés átadásnál használatos

```
goto azonosító;
```

A címkézett utasítás arra jó hogy, bármely utasítást megelőzhetik az

```
azonosító;
```

előtagok, amelyek címkeként szolgálnak, ezeket használja a goto utasítás.

A nulla utasítás alakja a

;

hordozhat címkét vagy üres ciklus törzset is képezhet.

A két fontos eltérés a verziók között, ami még nincs a régebbiben, hogy nincsenek // -típusú kommentek, és a for cikluson belül nem lehet deklarálni változót. Lásd.: forrásban

Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása: [Prog1/repo/bhax/thematic_tutorials/bhax_textbook/masodik/lex/real.c](https://github.com/prog1/repo/bhax/thematic_tutorials/bhax_textbook/masodik/lex/real.c)

Tanulságok, tapasztalatok, magyarázat...

```
% {  
#include <stdio.h>  
#include <string.h>  
int valos_szamok = 0;  
%}  
szam [0-9]  
%%  
{szam}*(\.\{szam}+) ? {++valos_szamok;  
    printf("[valossz=%s %f]", yytext, atof(yytext));}  
%%  
int main()  
{  
    yylex ();  
    printf("Valós számok darabszáma: %d\n", valos_szamok);  
    return 0;  
}
```

A lexer, más néven lexikális analízis egy olyan folyamat amely egy inputon bemenő karaktereket tokenekre alakítják át. A forrás feladatomban lévő lexer, arra van kialakítva, hogy a 0-9-ig terjedő számokat megkeresse. Ebből áll a forrásom eleje is, azt mutatjuk meg a programnak hogy azokat keresse. Miután lefut a programunk, kiírja az inputon beérkezett valós számok darabszámát.

I33t.I

Lexelj össze egy l33t ciphert!

Megoldás videó: [BN L33t Stream](#)

Megoldás forrása: [leet.c](#)

Tanulságok, tapasztalatok, magyarázat... A lexereket felhasználva és egy kicsit módosítva az előző példánkból, összerakhatunk egy l33t cipher-t is. A cipherek egy kódolásra illetve dekódolásra alkalmas lexikális algoritmusok. A l33t(értsd. leet) egy olyan online kreált nyelv, melynek alapja lehet bármely nyelv, a l33t viszont a betűket kicseréli hasonló karakterekre, melyeket szintén lehet olvasni, és értelmezni őket, mint az eredeti nyelvet. Példa lehet a 'e' betű és a '3' cseréje, vagy a 't' és a '7'-es cseréje. A leet cipher hasonló alapokon működik mint a lexer, csak például kap egy tömböt amely tartalmazza a betűket és azoknak a helyettesítéseit, hogy mire lehet kicserálni, mert egy betűnek több leet megfelelője is lehet.

A források olvasása

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

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

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



Bugok

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

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

II. Az i változó nincs deklarálva és ha lenne deklarálva, akkor a ciklus elszámolna 4 ig, majd megáll, az ++i inkrementáció miatt.

III. Ugyanaz mint az előző, csak utólag inkrementálódik, és szintén nincs az int deklarálva.

IV. Egy tömböt próbálunk változó inkrementálással feltölteni, az 'i' itt sincs deklarálva.

V. Az i változót addig növeljük míg nem lesz egyenlő n-1-el és minden közben teljesül a (*d++ = *s++) feltétel. Itt, a *s++-el a pointert inkrementáljuk, nem azt amire mutat, és ezt fogjuk átadni a *d++ szintén előbb növelt pointernek. Bug, hogy az ÉS logikai művelet jobb oldalán nem boolean van.

VI. Az 'f' függvényel két decimális számot szeretnénk kiíratni, a probléma ott adódik, hogy utána az argumentumok sorrendjének hiánya miatt definiálatlanok, és módosíthatják egymást.

VII. Szintén két decimális szám kiíratásáról van szó, csak az egyik szám az egy 'f' függvény argumentuma ként van megadva, a másik meg csak maga a számként.

VIII. Itt is két szám kiíratásáról van szó, az elsőnél f függvénynek megadjuk az 'a' változónak a címét majd a függvény az erre a változóra mutató értéket fogja inkrementálni, a másik szám meg maga az 'a' változó értéke.

Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

I. Végtelen sok prímszám van.

II. Végtelen sok iker-prímszám van.

III. Véges sok prímszám van.

IV. Véges sok prímszám van.

Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

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

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

Megoldás videó:

Megoldás forrása: [deklarator.cpp](#)

Tanulságok, tapasztalatok, magyarázat... Forrásban megtalálhatóak a bevezetések.

```
an7hr0x@an7hr0xPC:~/Progl/repo/bhax/thematic_tutorials/bhax_textbook/ ←
 masodik/dekla$./dekla
5
5
0x601070
0
1
2
3
4
0x601080
```

## 4. fejezet

# Helló, Caesar!

### **int \*\*\* háromszögmátrix avagy "legyen egy csillaggal kevesebb és double"**

Tehát, hogyan íratunk ki egy háromszögmátrixot konzolra double, double-re mutató mutató és double-re mutató mutatóra használatával.

Megoldás videó:

Megoldás forrása: [Csill.c](#)

Tanulságok, tapasztalatok, magyarázat...:

A programunk úgy épül fel, hogy először megadjuk hogy hány sora legyen a mátrixnak, és deklarálunk egy mutatót amely mutató double tipusú lesz. Ez a mutató fog majd nekünk segíteni a mátrix soraiban lévő elemekre hivatkozásban. A malloc függvény használatával lefoglalunk minden egyes sornak egy-egy mutatót a memóriában. A visszaadott void tipusú mutatót double re kényszerítjük, ha valami hiba lenne eközben és NULL pointerünk szólette, akkor kilépünk a programból. Majd minden sorban foglalunk helyet a memóriában egy alsó háromszög mátrixnak megfelelő double-nek. Ezek már nem mutatók lesznek, hanem konkrét értékek, itt is ha valami hiba lépne fel, akkor kilépünk a programból. A sorokat és a soronkénti oszlopokat 0-tól kezdve egyesével feltöjtük lebegőpontos számokkal. Végül, végiglépkedünk az egészen mégegyszer, és a korábban lefoglalt memória területeket felszabadítjuk hogy ne legyen memória leak .

## C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: [Exor Kódoló](#)

Tanulságok, tapasztalatok, magyarázat... A program működésének egy része a forrásomban el van magyarázva már, de itt is a lényeget megírom. Az exor kódoló, egy kizárvagy-os logikai műveleten alapuló kódolás, mely egy megadott kulcs alapján kódol. Használata igen egyszerű, futtatásnál, első argumentumnak megadjuk a használni kívánt kulcsot, majd a titkosítandó szöveges fájlt, és adunk egy kimeneti fájl nevet, majd futtatjuk, és meg is kaptuk a számunkra olvashatatlan kódolt szöveget.

## Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: [Exor Kódoló in Java](#)

Tanulságok, tapasztalatok, magyarázat...

Először létrehozunk egy public classt vagy objectet melyben dolgozni fogunk. Utána megcsináljuk a konstruktort , mellyel majd átadjuk az objektumnak a kapott értékekkel. A paraméterként átadott kulcsot tároljuk egy stringbe, majd az input és output csatornát létrehozzuk. Egy byte tömbben tároljuk a megadott kulcs bájtjait és a buffer méretét. Innen hasonlít a C változathoz, csak azzal a különbséggel hogy a mainbe teszünk még egy try catch ágat hogy az esetleges hibákat megfogjuk.

```
public class ExorTitkosító {

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

 byte [] kulcs = kulcsSzöveg.getBytes();
 byte [] buffer = new byte[256];
 int kulcsIndex = 0;
 int olvasottBájtok = 0;
 while((olvasottBájtok =
 bejövőCsatorna.read(buffer)) != -1) {

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

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

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

 public static void main(String[] args) {

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

```
 e.printStackTrace();

}

}
}
```

## C EXOR törő\*

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

Megoldás videó:

Megoldás forrása: [Exor Kódtörő](#)

Tanulságok, tapasztalatok, magyarázat...

A programunk felépítése a következő. Először a program elején kiszámítjuk az átlagos szóhosszt. Összeszámoljuk a forrásunkban szóközöket és elosztjuk vele az összes karakter számát. Ezekután, biztosak lehetünk abban, hogy a szövegünk tartalmaz gyakori magyar szavakat, mint például: "az", "ha", "nem", "hogy". Az előbb kiszámolt átlagos szóhossz segítségével csökkenthetjük a törések számát is emiatt. Majd bájtonként végigmegyünk az EXOR-al. A % használatával, a kulcsot aktuálisként tartjuk akkor is, ha a szó hosszabb mint a kulcs. Ezek után, while ciklussal megyünk amíg van bemenet, és egymásba ágyazott for ciklusokkal kipróbáljuk és előállítjuk az összes kulcsot, ha ez megtörtént, újra exorozunk, ennek köszönhetően nem fog kelleni második buffer.

## Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat... A következő logikai kapuk szimulált ideghálóval fognak kiértékelődni. OR:

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

AND:

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

EXOR:

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

## Hiba-visszaterjesztéses perceptron\*

C++

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

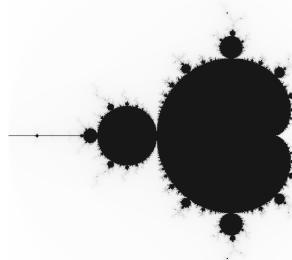
Először definiáljuk a kezdeti súlyokat, majd az input végigmegy a hálózaton de a súlyok változatlanok maradnak. Majd az elkészült kimenetet összevetjük a tényleges kimeneti jellet. Majd a hibát visszaküldjük a neuronokon keresztül és most változtatunk a súlyokon, úgy, hogy a hibákat amennyire csak tudjuk lecsökkenentsük. Azt, hogy milyen módon változtassuk a súlyokat, abban segít a hiba-visszaterjesztéses algoritmus a kimeneti rétegekből a rejttekbe. Elsőnek a kimeneti rétegen meghatározzuk a hibák számát, és az így kapott értékeket visszaterjesztjük a kimeneti réteg előttre, a rejttet rétegekre. A kapott hibákat egymás után, korábbi rétegekre terjesztjük vissza, így a hiba skálázódik, a jelenlegi meg az őt megelőző súlyok értékeinek függvényében. Ez addig folytatódik, míg meg nem kapjuk a bemeneti réteget.

## 5. fejezet

# Helló, Mandelbrot!

### A Mandelbrot halmaz

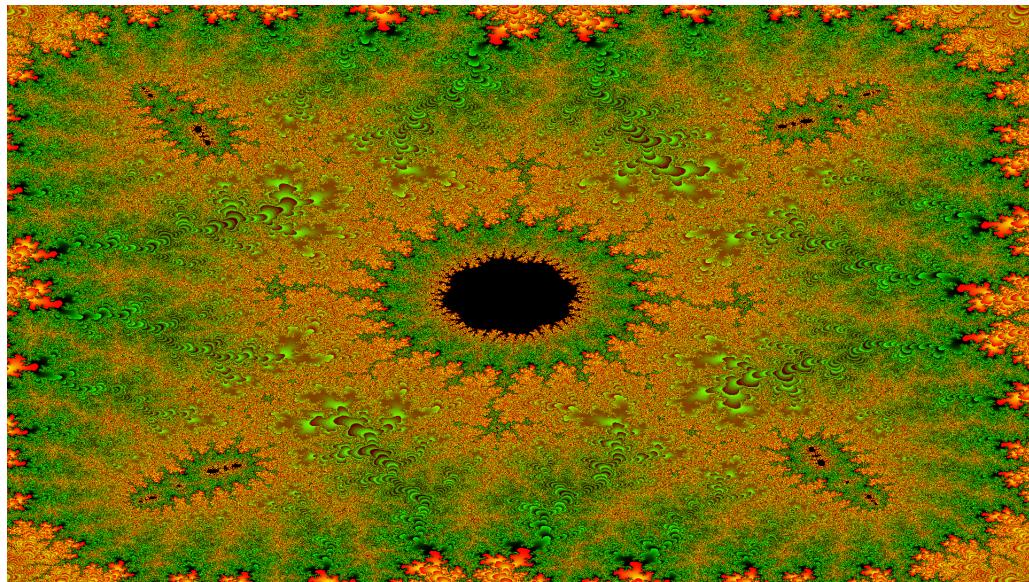
Megoldás forrása: [Mandel](#)



A Mandelbrot halmaz a komplex számsíkon egy halmaz, melyet Benoit Mandelbrot talált meg. Célja nehezen értelmezhető kérdésekre válasz adása volt, például hogy a -9 et mely két szám összeszorzásával kapjuk meg. Az ikonikus képet úgy kapjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy például 800x800 pixeles rácsot és kiszámítjuk hogy a rács pontjai mely komplex számoknak felelnek meg. A rács összes pontját megvizsgáljuk a  $z_{n+1} = z_n^2 + c$ , ( $0 \leq n$ ) képlet alapján. A képletben a  $z$ -ket vizsgáljuk meg azon szempont szerin hogy kivezet e a 2 sugarú körből vagy nem. Ha nem lép ki akkor feketére színezzük hogy a  $c$  rácspont a halmaz része. Ez alapján működik a programunk is, persze ennél bonyolultabb a valóságban. Fordításhoz szükségünk lesz a png könyvtárra, és meg is kapjuk a kész png képünket a halmazról, mely kép bizonyos dolgok átirásával a forrásunkban szinezhető, és formázható.

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

Megoldás videó:

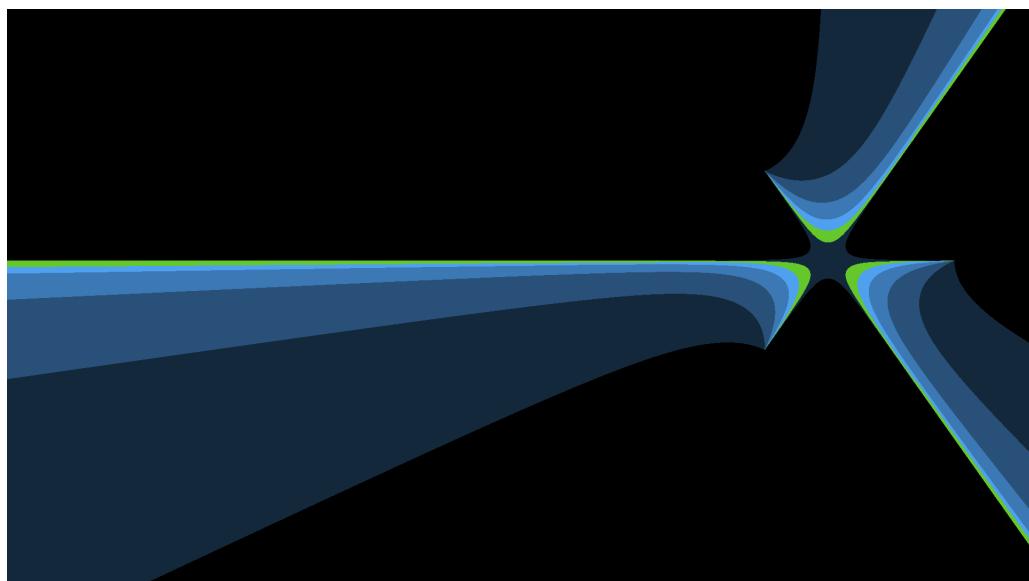


Megoldás forrása: [std::complex Mandel](#)

Hogy komplex számokkal tudjunk számolni, importálni kell a komplex osztályt c++-ban. A programunk hasonlít az előzőhez, ugyan az a lényege és működési elve(mármint Mandelbrot képlete). A programunkban for ciklusokkal végig megyünk a háló oszlopain és sorain. A reC és imC a háló rácspontjait számoljuk ki és a rácspontokhoz tartozó komplex számokat pedig a complex osztály segítségével számítjuk ki. Ha az iterációs határ elérése miatt kell kilépnünk a while ciklusból, akkor ebben a pontban az iterációkonkvergens és a Mandelbrot halmaz elemének vesszük, és feketére színezzük be. Futtatáskor meg kel adni még a png könyvtár használatát, a kép nevét, meg hogy milyen számításokkal dolgozzon a programunk.

## Biomorfok

Megoldás videó: <https://youtu.be/IJMbgrzY76E>



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

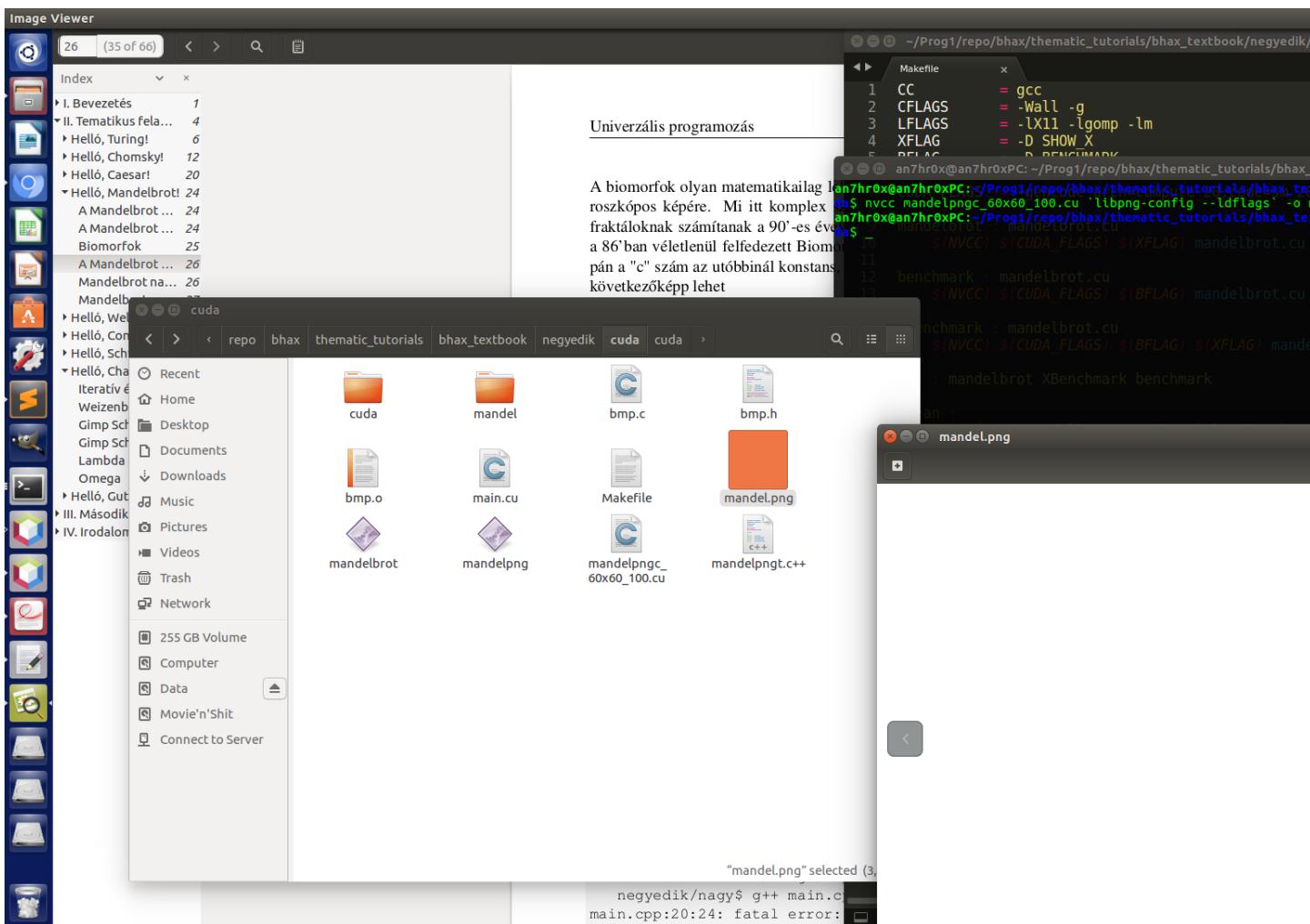
Tanulságok, tapasztalatok, magyarázat...

A biomorfok olyan matematikailag leképzett fraktál alakzatok, melyek hasonlitanak egy organizmus mikroszkópos képére. Mi itt komplex számsíkon vett fraktálokra gondolunk ennél a feladatnál. Híresebb fraktáloknak számítanak a 90'-es években felfedezett Júlia-halmazok, a jól ismert Mandelbrot Halmaz és a 86'ban véletlenül felfedezett Biomorfok. A Mandelbrot és a Júlia halma közti különbség csekély, csupán a "c" szám az utóbbinál konstans, Mandelnél meg változik egy iterációban. Futtatni a programunkat, a következőképp lehet

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

## A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:



Megoldás forrása:

A CUDA technológia az NVIDIA tulajdona és fejlesztése, a videókártyát használja a számításokhoz. A CUDA segítségével a mandelbrot halmazt kb 70x gyorsabban tudnánk generálni. A használathoz NVIDIA-s videókártya kell csak. Nekem minden hibátlanul is működött kivéve a végeredményt mert az egy fehér kép lett. Próbáltam több forrásból beszerezni a CUDA-s mandelbrot halmaz forrását, de mindegyiknél valami error dobott ki és jelenleg is még próbálkozás fázisban tart a kép felélesztése.

# Mandelbrot nagyító és utazó C++ nyelven

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

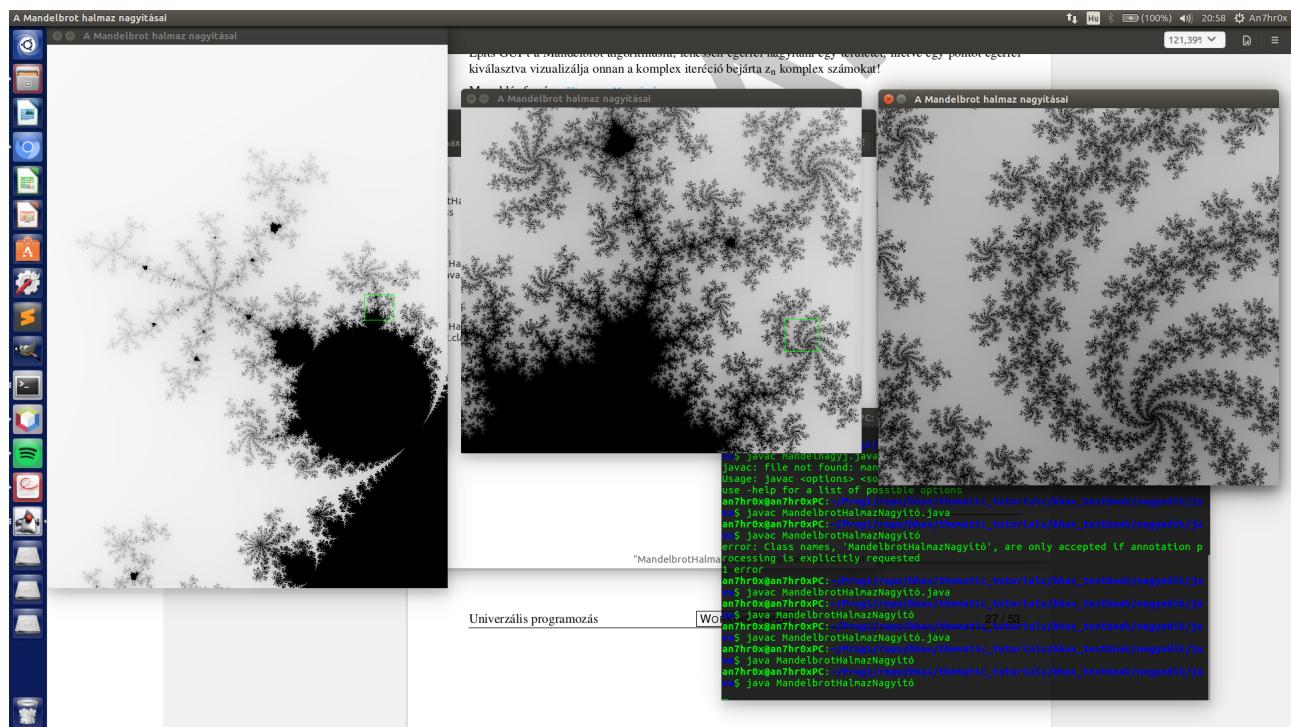
Megoldás forrása: Utazo Mandel

## Megoldás videó:

Az utazó / nagyító lényege a következő. Először is veszünk egy alap leképzett mandelbrot halmazt, és azt teszi lehetővé a programunk, hogy ha abba bele nagyítunk esetleg mozgatjuk is a képet, folyamatosan generálja le a folytatását, így egy végtelen fraktált kapva. A programom lefordulásánál egy hibába akadtam, melynek a megjavításán még jelenleg is dolgozok.

```
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/ ←
 negyedik/nagy$ g++ main.cpp -o main
main.cpp:20:24: fatal error: QApplication: No such file or directory
compilation terminated.
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/ ←
 negyedik/nagy$
```

# Mandelbrot nagyító és utazó Java nyelven



A java mandelbrothalmaz nagyító nagyon hasonlit a c++-os megoldáshoz, itt is kapunk egy ablakot melyben bennevan a halmazunk, és a balklikkel tudjuk kijelölni hogy hova és mekkora területre nagyítsunk rá, miután kiválasztottuk megmutatja egy új képen azt a területet. Amikor fut a programunk még tudunk olyat is csinálni hogy ha az s billentyűt megnyomjuk akkor a jelenleg megjelenített halmazról csinál nekünk egy screenshotot, továbbá ha az n gombot nyomjuk meg, akkor a számítások iterációs határát is tudjuk növelni, ezzel növelődik a pontosság.

```
public class MandelbrothalmazNagyító extends Mandelbrothalmaz {
 private int x, y;
 private int mx, my;

 public MandelbrothalmazNagyító(double a, double b, double c, double d,
 int szélesség, int iterációsHatár) {
 super(a, b, c, d, szélesség, iterációsHatár);
 setTitle("A Mandelbrot halmaz nagyításai");

 addMouseListener(new java.awt.event.MouseAdapter() {
 public void mousePressed(java.awt.event.MouseEvent m) {
 x = m.getX();
 y = m.getY();
 mx = 0;
 my = 0;
 repaint();
 }
 public void mouseReleased(java.awt.event.MouseEvent m) {
 double dx = (MandelbrothalmazNagyító.this.b
 - MandelbrothalmazNagyító.this.a)
 /MandelbrothalmazNagyító.this.szélesség;
 double dy = (MandelbrothalmazNagyító.this.d
 - MandelbrothalmazNagyító.this.c)
 /MandelbrothalmazNagyító.this.magasság;

 new MandelbrothalmazNagyító(MandelbrothalmazNagyító.this.a +
 x*dx,
 MandelbrothalmazNagyító.this.a+x*dx+mx*dx,
 MandelbrothalmazNagyító.this.d-y*dy-my*dy,
 MandelbrothalmazNagyító.this.d-y*dy,
 600,
 MandelbrothalmazNagyító.this.iterációsHatár);
 }
 });
 addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
 public void mouseDragged(java.awt.event.MouseEvent m) {
 mx = m.getX() - x;
 my = m.getY() - y;
 }
 });
 }
}
```

```
 repaint();
 }
}

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);
 if(számításFut) {
 g.setColor(java.awt.Color.RED);
 g.drawLine(0, sor, getWidth(), sor);
 }
 g.setColor(java.awt.Color.GREEN);
 g.drawRect(x, y, mx, my);
 g.dispose();

 StringBuffer sb = new StringBuffer();
 sb = sb.delete(0, sb.length());
 sb.append("MandelbrotHalmazNagyítás_");
 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 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);
}

g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
}

public static void main(String[] args) {

 new MandelbrothHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
```

## 6. fejezet

# Helló, Welch!

### 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 video:

Megoldás forrása: [CPP Polar](#), [Java Polar](#)

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!

C++-ban megvalósítva az eredmény:

```
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/otodik ↵
 /polar$ g++ polar.cpp -o polar
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/otodik ↵
 /polar$./polar
0.691468
-0.388269
0.254064
0.460803
1.17808
0.0145214
-0.0130577
-1.57915
-1.17233
-0.187721
```

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

```
{
public:
 PolarGen ()
 {
 nincsTarolt = true;
 std::srand (std::time (NULL));
 }
 ~PolarGen ()
 {
 }
 double 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;
 }
 }
private:
 bool nincsTarolt;
 double tarolt;
};
int main (int argc, char **argv)
{
 PolarGen pg;
 for (int i = 0; i < 10; ++i)
 std::cout << pg.kovetkezo () << std::endl;
 return 0;
}
```

És javaban megvalósítva:

```
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());
 }
}
```

```
}
```

## LZW

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

Megoldás videó:

Megoldás forrása: [Binfa](#)

A programunk egy olyan algoritmus mely a bemenő bináris adatokat fa struktúrában ábrázolja, az algoritmus megalkotója Abraham Lempel, Jacob Ziv, és Terry Welch. Ez az algoritmus veszteségmentes tömörítést eredményez, melyet sok kiterjesztés is használ például, a .gif, vagy tömörítő programok mint a gzip, zip. Miután az input adatok bekerülésekor a program a gyökértől kiindulva addig követjük a fa ágait, amíg egy olyan részsztringhez nem érünk mely már nem található meg a fában. Ekkor a részsztring utolsó karakterével ami pont feldolgozás alatt van bővítjük a fát. A következő bemenetnél egy új részsztringet újra a gyökértől végigfuttatjuk. Így kapjuk meg az LZW Bináris fánkat.

```
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/otodik ↵
/z3a7c$./z3 <old

almafa

-----1 (4)
-----1 (3)
-----1 (2)
---/(1)
melyseg=4
```

## Fabejárás

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

Megoldás videó:

Megoldás forrása: [Binfa Inorder](#), [Binfa Preorder](#), [Binfa Postorder](#)

A fákat 3 módon tudjuk bejárni, preorder inorder és posztorder. Ezek azt jelentik hogy ha egy nem üres fán végig akarunk menni hogy indulunk el. Az inorder bejárásnál a fánk bal részfájával kezdünk, majd jön a gyökér és végül a jobb részfa. Preordernél ez úgy néz ki hogy gyökér, bal oldal és jobb oldal, majd postordernél bal oldal, jobb oldal és végül a gyökér. A LZWBinfánk alapból inorder bejárású, melynek a bejárását a következő forráskód részleténél lehet megszabni.

```
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;
}
}
```

Preordernél a következő a helyzet:

```
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;
 }
}
```

Postordernél végül így néz ki a kiir függvényünk:

```
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;
}
}
```

```
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/ ←
otodik/pip$./z3i <old
```

```
01011101011010110101111
```

```
-----1(4)
-----1(3)
-----0(4)
----1(2)
-----1(5)
-----1(4)
-----0(5)
-----0(3)
---/(1)
-----1(3)
----0(2)
```

```
melyseg=5an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/ ←
bhax_textbook/ot ./z3pre <old
```

```
01011101011010110101111
```

```
---/(1)
-----0(2)
-----1(3)
----1(2)
-----0(3)
-----1(4)
-----0(5)
-----1(5)
----1(3)
-----0(4)
-----1(4)
```

```
melyseg=5an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/ ←
bhax_textbook/ot ./z3pos <old
```

```
01011101011010110101111
```

```
-----1 (3)
-----0 (2)
-----0 (5)
-----1 (5)
-----1 (4)
-----0 (3)
-----0 (4)
-----1 (4)
-----1 (3)
-----1 (2)
---/(1)
```

## Tag a gyökér

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

Megoldás videó:

Megoldás forrása: [Binfa CPP](#)

Itt az osztályban a fa gyökere nem egy pointer, hanem egy object. A fa a mutató, ami mindenkorán a jelenleg létrejövő fa csomópontjára mutat. A LZWBInFa konstruktőr ráállítja a fa pointert a gyökerére, a dekonstruktőrben pedig felszabadítjuk a gyökérnek a gyerekeit. << operátort túlterheljük, és az inputot ezek alapján tesszük be a fába. Ha az input karakter 0, akkor megnézzük hogy a jelenlegi csomópuntunknak létezik-e 0-ás gyereke, vagyis a fa pointer éppen rá mutat-e. Ha nincs neki, akkor példányosítjuk a 0-betűt és új csomópontot hozunk létre. Ezek után a jelenlegi csomópontunk 0-ás gyermekét ráállítjuk az új csomópontra, majd a fával visszaállunk a gyökerre. Ha van 0-ás gyerek, akkor pedig a fa pointert ráállítjuk. Ha 1-es karakter van, akkor ugyan ez történik.

```
class LZWBInFa
{
public:
 LZWBInFa () :fa (&gyoker)
 {
 }
 ~LZWBInFa ()
 {
 szabadit (gyoker.egyesGyermek ());
 szabadit (gyoker.nullasGyermek ());
 }
 void operator<< (char b)
 {
```

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

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

 fa = fa->nullasGyermek ();
 }
}

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

Futtatáskor a helyes argumentum megadásnál kell egy bemenő és egy kimenő fájl, melyből olvasunk és melybe írunk.

```
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/ ←
 otodik/z3a7$./binnfa old -o new
an7hr0x@an7hr0xPC:~/Prog1/repo/bhax/thematic_tutorials/bhax_textbook/otodik ←
 /z3a7$
```

## Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: [Binfa CPP muti](#)

Itt az eltérés az előzőhez képest, hogy a fa pointert ráállítjuk a gyökérre a konstruktorból. De most a gyökér is pointer lesz, ezért ahol az előző forrásban referenciaként adtuk át a gyökeret a fa pointernek, itt referencia nélkül tesszük. De még példányosítjuk is a konstruktorból a gyökeret, helyet is foglalunk neki a memóriában és erre állítjuk rá a fát. És végezetül felszabadításkor a pont helyett nyilat használunk ha a mutató mutatóit kell elérni.

```
public:

LZWBinFa ()
{
 gyoker = new Csomopont();
 fa = gyoker;
}
~LZWBinFa ()
{

 szabadit (gyoker->egyesGyermek());
 szabadit (gyoker->>nullasGyermek());
 delete gyoker;
}
```

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

## Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: [Binfa CPP cctor](#)

```
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;
}
```

```
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);
}
```

```
Csomopont * masol (Csomopont * elem, Csomopont * regifa) {
 Csomopont * ujelem = NULL;
 if (elem != NULL) {
 ujelem = new Csomopont (elem->getBetu());
 ujelem->ujEgyesGyermek (masol (elem->egyesGyermek (), ←
 regifa));
 ujelem->ujNullasGyermek (masol (elem->>nullasGyermek (), ←
 regifa));
 if (regifa == elem)
 fa = ujelem;
 }
 return ujelem;
}
```

```
LZWBinFa binFa3 = std::move (binFa);
kiFile << "depth = " << binFa3.getMelyseg () << std::endl;
kiFile << "mean = " << binFa3.getAtlag () << std::endl;
kiFile << "var = " << binFa3.getSzoras () << std::endl;
```

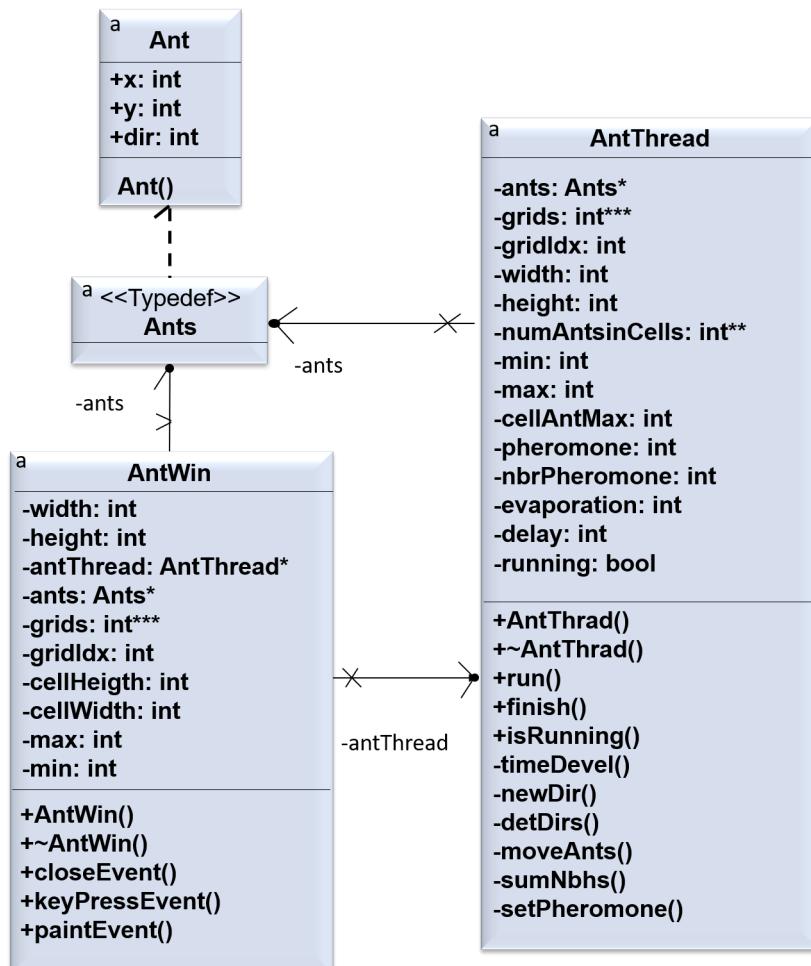
## 7. fejezet

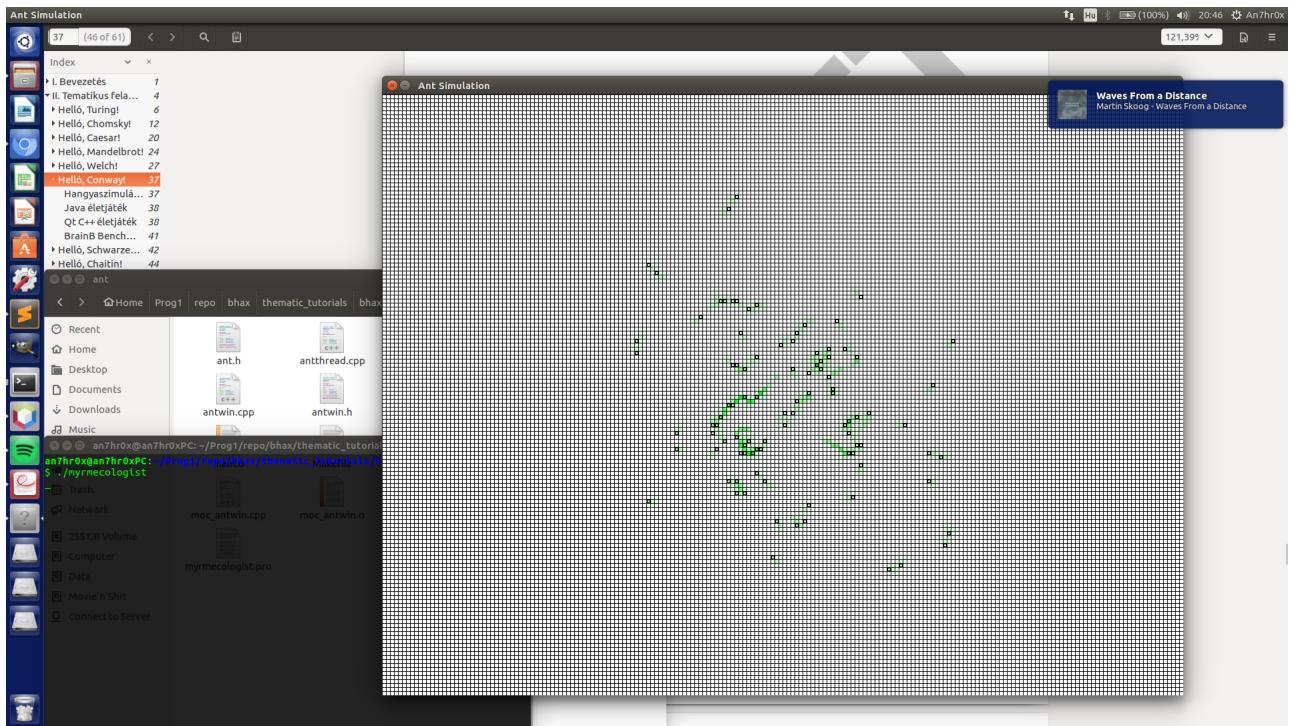
# Helló, Conway!

## Hangyaszimulációk

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

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





Megoldás forrása: [Hangya](#)

Ha futtatjuk a programunkat, akkor egy ablakot kapunk mely megfelel terráriumban lévő hangya kolónia szimulációjának. A hangyák mozognak, feromont bocsájtanak ki, és élik az életüket a programunkban a megadott feltételek alapján. Az UML osztálydiagram értelmezése a következő. Az ANt osztályban a hangyáknak a tulajdonságaik vannak meghatározva, irányai az x és y koordinátái, a dir meg mint direction az iránytukat jelenti. Ha az osztálydiagramban valami előtt '+' vagy '-' jel van az azt jelenti hogy ha '+' akkor publikus osztály és bárhol elérhető, osztályán kívül is, ha '-' akkor pedig private, és csak a saját osztályában érhető el. Ebből következik, hogy az ants egy vektor, amiben a hangyák vannak. Az AntWin private részében vannak az ablak, és a sejtek méreteire irányuló beállítások. Az antThread pedig egy ablaknak a szála mely a sejtszámításokat végzi. Grids a rácsokat jelenti, a gridIdx pedig két rácspont közül tárol egyet. A min és max a feromon min 1 és max 255 értékét jelenti. Függvényként megtalálható a konstruktor sé destruktur is. A closeEvent() kikapcsolásért szolgál, a keyPressEvent() a gombnyomást, és a paintEvent() a hangyák színéért felelős. Az AntThread hasonlit az AntWin re. Az evaporation a párolágás mértéke.

## Java életjáték

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

Megoldás videó:

Megoldás forrása: [Sejt Kávé](#)

Tanulságok, tapasztalatok, magyarázat...

A program törénete megegyezik a C++-os változatéval, mivel ugyan az csak más kódban implementálva. Ugyan úgy van élettér, melyek cellákra tagolódnak, és a cellákban egy sejt található, melyek meghalnak ha nincs minimum 2 élő szomszédja, és mindaddig halott marad amíg 3 élő szomszédja van, különben élni fog. Ezeknek a szabályoknak az együttese a kódban itt található:

```
public void időFejlődés() {

 boolean [][] rácsElőtte = rácsok[rácsIndex];
 boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];

 for(int i=0; i<rácsElőtte.length; ++i) { // sorok
 for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok

 int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);

 if(rácsElőtte[i][j] == ÉLŐ) {
 /* Elő élő marad, ha kettő vagy három élő
 szomszedja van, különben halott lesz. */
 if(élők==2 || élők==3)
 rácsUtána[i][j] = ÉLŐ;
 else
 rácsUtána[i][j] = HALOTT;
 } else {
 /* Halott halott marad, ha három élő
 szomszedja van, különben élő lesz. */
 if(élők==3)
 rácsUtána[i][j] = ÉLŐ;
 else
 rácsUtána[i][j] = HALOTT;
 }
 }
 }
 rácsIndex = (rácsIndex+1)%2;
}
}
```

A sikló a sejtterben lévő élőlény. Mely egy adott irányba megy, és másolja is magát. Ezek egy kettő dimenziós tömbbe vannak elhelyezve, az x érték a befoglaló téglalal felső sarkának az oszlopának fele meg, az y pedig a sorának.

```
public void sikló(boolean [][] rács, int x, int y) {

 rács[y+ 0][x+ 2] = ÉLŐ;
 rács[y+ 1][x+ 1] = ÉLŐ;
 rács[y+ 2][x+ 1] = ÉLŐ;
 rács[y+ 2][x+ 2] = ÉLŐ;
 rács[y+ 2][x+ 3] = ÉLŐ;
}
```

A siklóagyú egy előre megadott irányba lövi ki a siklókat.

```
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;
```

```

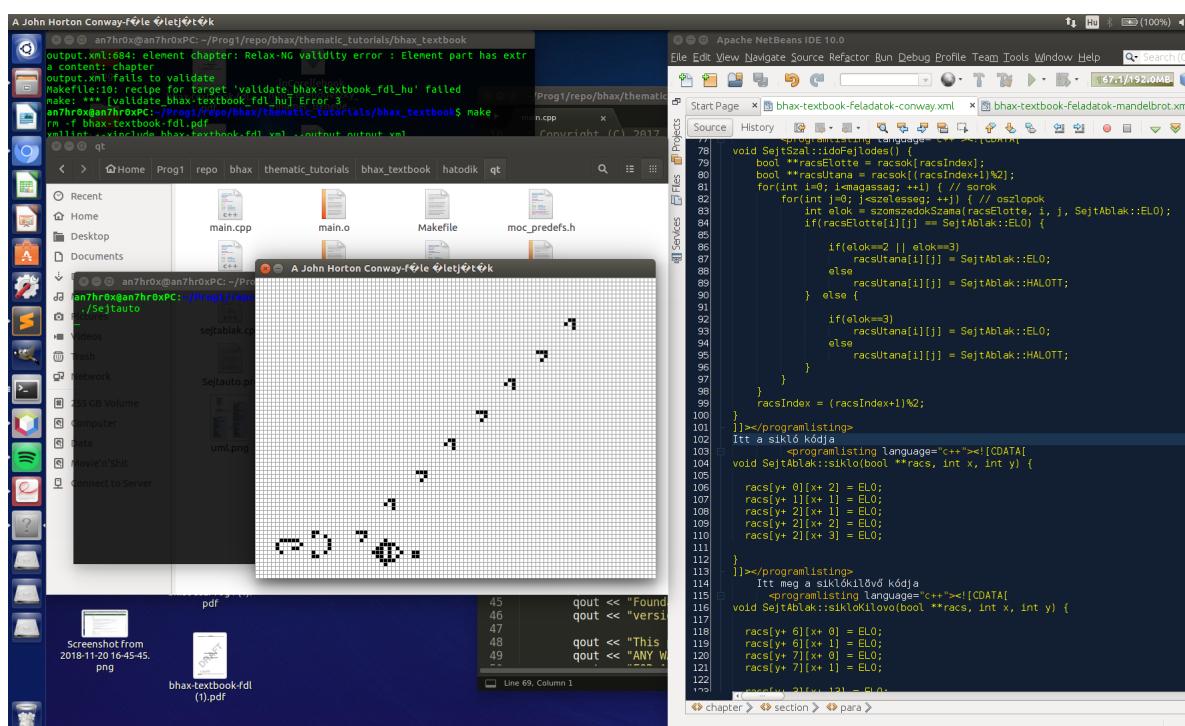
rács[y+ 10][x+ 25] = ÉLŐ;
rács[y+ 11][x+ 25] = ÉLŐ;
}

```

A program egy objectként van megírva, majd a main függvényben példányosítva van.(paramétere az oszlop és sor mérete)

## Qt C++ életjáték

Most Qt C++-ban!



Megoldás video:

Megoldás forrása: [The Game Of \(Thrones\) Life](#)

Tanulságok, tapasztalatok, magyarázat... Az algoritmust John Conway a Cambridge egyetem matematikusa találta ki, és a program híres lett 1970-ben. A program egy sejt automatizációs szimuláció, mely előre megadott szabályok szerint működik, és a kis négyzetek melyek a sejteknek felelnek meg, ezek szabályok szerint szaporodnak, mozognak, halnak meg.

Először is a szabályokat implementáljuk

```

void SejtSzal::idoFejlodes() {
 bool **racsElotte = racsok[racsIndex];
 bool **racsUtana = racsok[(racsIndex+1)%2];
 for(int i=0; i<magassag; ++i) { // sorok
 for(int j=0; j<szelesség; ++j) { // oszlopok
 int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);
 if(racsElotte[i][j] == SejtAblak::ELO) {

```

```
 if(elok==2 || elok==3)
 racsUtana[i][j] = SejtAblak::ELO;
 else
 racsUtana[i][j] = SejtAblak::HALOTT;
 } else {

 if(elok==3)
 racsUtana[i][j] = SejtAblak::ELO;
 else
 racsUtana[i][j] = SejtAblak::HALOTT;
 }
}
racsIndex = (racsIndex+1)%2;
}
```

Itt a sikló kódja

```
void SejtAblak::siklo(bool **racs, int x, int y) {

 racs[y+ 0][x+ 2] = ELO;
 racs[y+ 1][x+ 1] = ELO;
 racs[y+ 2][x+ 1] = ELO;
 racs[y+ 2][x+ 2] = ELO;
 racs[y+ 2][x+ 3] = ELO;

}
```

Itt meg a siklókilővő kódja

```
void SejtAblak::sikloKilovo(bool **racs, int x, int y) {

 racs[y+ 6][x+ 0] = ELO;
 racs[y+ 6][x+ 1] = ELO;
 racs[y+ 7][x+ 0] = ELO;
 racs[y+ 7][x+ 1] = ELO;

 racs[y+ 3][x+ 13] = ELO;

 racs[y+ 4][x+ 12] = ELO;
 racs[y+ 4][x+ 14] = ELO;

 racs[y+ 5][x+ 11] = ELO;
 racs[y+ 5][x+ 15] = ELO;
 racs[y+ 5][x+ 16] = ELO;
 racs[y+ 5][x+ 25] = ELO;

 racs[y+ 6][x+ 11] = ELO;
 racs[y+ 6][x+ 15] = ELO;
 racs[y+ 6][x+ 16] = ELO;
 racs[y+ 6][x+ 22] = ELO;
```

```
racs[y+ 6][x+ 23] = ELO;
racs[y+ 6][x+ 24] = ELO;
racs[y+ 6][x+ 25] = ELO;

racs[y+ 7][x+ 11] = ELO;
racs[y+ 7][x+ 15] = ELO;
racs[y+ 7][x+ 16] = ELO;
racs[y+ 7][x+ 21] = ELO;
racs[y+ 7][x+ 22] = ELO;
racs[y+ 7][x+ 23] = ELO;
racs[y+ 7][x+ 24] = ELO;

racs[y+ 8][x+ 12] = ELO;
racs[y+ 8][x+ 14] = ELO;
racs[y+ 8][x+ 21] = ELO;
racs[y+ 8][x+ 24] = ELO;
racs[y+ 8][x+ 34] = ELO;
racs[y+ 8][x+ 35] = ELO;

racs[y+ 9][x+ 13] = ELO;
racs[y+ 9][x+ 21] = ELO;
racs[y+ 9][x+ 22] = ELO;
racs[y+ 9][x+ 23] = ELO;
racs[y+ 9][x+ 24] = ELO;
racs[y+ 9][x+ 34] = ELO;
racs[y+ 9][x+ 35] = ELO;

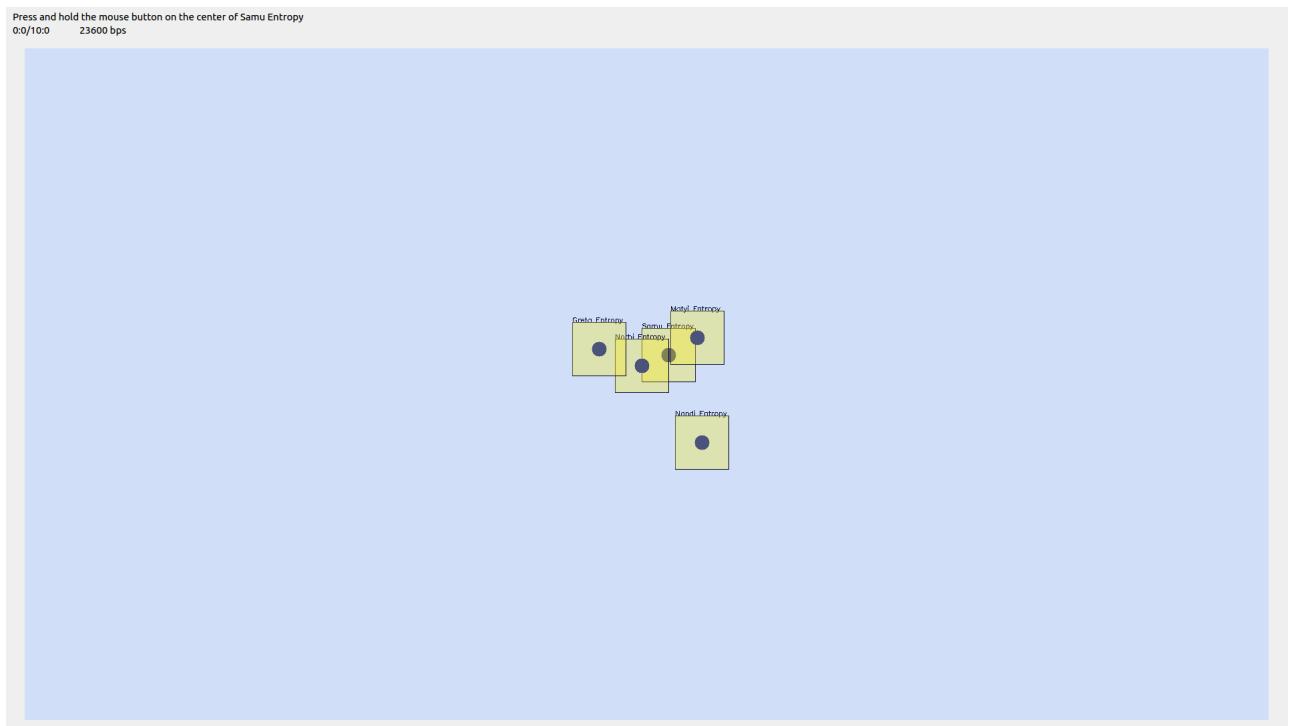
racs[y+ 10][x+ 22] = ELO;
racs[y+ 10][x+ 23] = ELO;
racs[y+ 10][x+ 24] = ELO;
racs[y+ 10][x+ 25] = ELO;

racs[y+ 11][x+ 25] = ELO;

}
```

## BrainB Benchmark

Megoldás video:



Megoldás forrása: [The Game Of \(Thrones\) Life](#)

Tanulságok, tapasztalatok, magyarázat... A programunk az esportolók tesztelésére lett kifejlesztve. A működésének az elve, hogy a képernyőn megjelennek négyzetek, benne körökkel. Ha kattintunk egyet elindul a tesztelés, és a feladata az "játékosnak" hogy a körben bennetartsa az egerét amik mozognak, sőt idővel több megjelenik. Ha túl gyorsnak bizonyul a játék, és kicsúszik a játékos a területből akkor lessul egy picit.

## 8. fejezet

# Helló, Schwarzenegger!

### 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://progater.blog.hu/2016/11/13/hello\\_samu\\_a\\_tensorflow-bol](https://progater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol)

Tanulságok, tapasztalatok, magyarázat...

A TensorFlow egy szoftverkönyvtár, célja a gépi tanulási algoritmusok megvalósítása. Nagyon sokrétfű sok mindenre használható eszköz a TensorFlow, például kézirás felismerésre is, mely tekinthető a TensorFlow "Hello World" programjának is. A programunk lényege hogy egy 28x28 pixeles png képeken lévő számjegyeket akarunk felismertetni a géppel. A modellünk alapja a következő részen nyugszik.

```
import argparse

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

import matplotlib.pyplot

FLAGS = None

def readimg():
 file = tf.read_file("sajat8a.png")
 img = tf.image.decode_png(file)
 return img

def main(_):
 mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

 x = tf.placeholder(tf.float32, [None, 784])
 W = tf.Variable(tf.zeros([784, 10]))
 b = tf.Variable(tf.zeros([10]))
```

```
y = tf.matmul(x, W) + b

y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, ↵
 y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(↵
 cross_entropy)

sess = tf.InteractiveSession()

tf.initialize_all_variables().run()
```

Miután meg lett adva hogy meglehet adni a helyes számot melyet fel kell ismernie meg a többi változót, elkezdjük tesztelni a hálózatot. Pontosság értékét kiiratjuk, kiiratjuk a teszt képet is manuális "human" ellenörzésre. Ezek után a gép kiirja hogy Ő mit érzékel.

```
print("-- A halozat tanitása")
for i in range(1000):
 batch_xs, batch_ys = mnist.train.next_batch(100)
 sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
 if i % 100 == 0:
 print(i/10, "%")
 print("-----")

Test trained model
print("-- A halozat tesztelese")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("-- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.test. ←
 images,
 y_: mnist.test.labels}))
print("-----")

print("-- A MNIST 42. tesztképenek felismerése, mutatom a számot, a ←
 továbbipeshez csukd be az ablakat")

img = mnist.test.images[42]
image = img

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm ←
 .binary)
matplotlib.pyplot.savefig("4.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")
```

```
print("<!-- A sajat kezi 8-asom felismerese, mutatom a szamot, a ←
 továbblepeshez csukd be az ablakat")\n\nimg = readimg()
image = img.eval()
image = image.reshape(28*28)\n\nmatplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm ←
 .binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()\n\nclassification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})\n\nprint("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")\n\nif __name__ == '__main__':
 parser = argparse.ArgumentParser()
 parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ ←
 mnist/input_data',
 help='Directory for storing input data')
 FLAGS = parser.parse_args()
 tf.app.run()</pre>
```

## Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Passzolt feladat

## Minecraft-MALMÖ

Megoldás videó:

Megoldás forrása:

Passzolási lehetőséggel éltem.

## 9. fejezet

# Helló, Chaitin!

### Iteratív és rekurzív faktoriális Lisp-ben

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

A Lisp programozási nyelv különbsége a többihez képest hogy lengyel írás módszert alkalmaz, azaz a műveletet előre írjuk és utána az operandusokat. A programunkban először a függvényt a definennal definiálni kell.

```
> (define (fact n) (do ((i 1 (+ 1 i)) (num 1 (* i num))) ((> i n) num))) ←
 //i-t 1 től növeljük, és a numot szorozzuk i-vel, addig amíg i nem lesz ←
 nagyobb mint n értéke.
fact
> (fact 5)
120
```

A rekurziv változatban található egy if felétel amely vizsgálja hogy n kisebb e mint 1, ha nem akkor n-t addig szorozzuk meg n-1-el amíg már nem hivja meg magát többször, tehát kisebb lesz mint egy.

```
> (define (fact n) (if (< n 1) 1 (* n(fact(- n 1)))))
fact
> (fact 5)
120
```

### Gimp Scheme Script-fu: króm effekt

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

Megoldás videó: [https://youtu.be/OKdAkI\\_c7Sc](https://youtu.be/OKdAkI_c7Sc)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Chrome](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome)  
Passzolt feladat.

## 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](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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala)  
Passzolt feladat.

# 10. fejezet

## Helló, Gutenberg!

### Programozási alapfogalmak(PICI)

#### 2.Hét

Megismertet minket a programozási nyelvek szintjeivel, Megnevezi pontosan a magasszintű nyelven írt programot. Ismerteti velünk a szintaktika és a szemantika fogalmát. Processzorok gép nyelvének létezéséről is szó esik, mint ahogy a fordító és értelmező programokról is. Megmagyarázza mi a fordítóprogram és mi az értelmező program. Hivatkozási nyelvről is ismerteti az alapvető információkat. Következő rész a programnyelvek osztályozása, beszél az imperatív nyelvekről, deklaratív nyelvekről, és a másnyelvű nyelvekről. Végül a jegyzet jelöléseiről esik szó, meg arról hogy miket tárgyal majd a jegyzet.

#### 3.Hét

Először az adattípusokat vesszük. Fogalma egy absztrakt programozási eszköz, amely minden más, konkret programozási eszköz egy komponenseként jelenik meg. Az adattípus neve egy azonosító. Léteznek típusos és nem típusos nyelvek, melyek vagy ismerik vagy nem az adattípusokat. Egy adattípus 3 dolgot határoz meg. Szó esik a tarományokról, a műveletekről és a reprezentációkról. Léteznek standard típusok. Definiálni is lehet egyes nyelvknél típusokat , melyekhez szükséges megadni a tartományt a műveleteket és a reprezentációt. Reprezentációt kevés nyelvben lehet megadni(pl.Ada). Az adattípusok önállóak, és egymástól különböznek, de létre lehet hozni altípusokat, mely egy szűkebb tartománya egy típusnak, így a műveletei és a reprezentációja változatlan. Az adattípusoknak 2 nagy csoportja van, a skalár vagy másnéven az egyszerű adattípus, mely atomi értékeket tartalmaz, ezek lehetnek a literálok egy program szövegében. A másik nagy típus a struktúrált vagy összetett adattípus, ezeknek a tarományainak az elemei is valamilyen típussal rendelkeznek, és nem atomiak. Ezek általában valamilyen absztrakt adatszerkezet programnyelvi megfelelői. (2.4.1.\Egyszerű típusok) Minden nyelvben létezik az egész típus/típusok. Néhány nyelv még ismeri az előjel nélküli típust. Az egész fixpontosan van belsőleg ábrázolva, a valós típusok viszont lebegőpontosak. Az egész és valós típusokra a közösen numerikus típusokként hivatkozunk. Léteznek karakteres típusok, melynek elemei karakterek, vagy például a karakterlánc és a sztring elemei a karaktersorozatok. Léteznek még ezeken kívül logikai típusok(tartománya igaz hamis értékekkel áll), felsorolás típus, sorszámozott típus és a sorzámozott típus altípusa ként lehet származtani az intervallum típust. (2.4.2)\Összetett típusok)Két legfontosabb típus itt a tömb, és a rekord. A tömböt meghatározza a dimenziók száma, az index készletének típusa és tartománya, és az elemeinek a típusa. A C nyelv nem ismeri a többdimenziós tömböket. Ha a tömb nevére hivatkozunk, akkor egy érték csoportra hivatkozunk vele(kivéve C). Elemek sorrendjét a reprezentáció határozza meg. Az értékcsoporthoz egyes elemeire az indexek segítségével hivatkozunk. A PICI könyv ezek után kitér a tömb típussal kapcsolatban kérdésekre, és azok megválaszolására,

melyek fontosak ahhoz hogy a tömb típust helyesen használjuk. A következő amire kitér, az a rekord típus, melynek tarományának elemei olyan értékcsoporthoz, amelyeknek az elemei különböző típusúak lehetnek. Az egyes elemeket mezőknek nevezzük, minden mezőnek önálló neve és egy saját típusa van. C ben a rekord statikus típus, más nyelvknél (pl. Ada) viszont eltérhet. Kitér ősnyelvek rekordtípusaira a könyv még. És az újabb nyelvek rekordtípusának egyszintűségére is. Egyes mezőkre külön minősített névvel tudunk hivatkozni, ennek alakja

eszköznév .mezónév

, erre azért van szükség mert a mezők nevei nem minden egyediek. A rekord típus alapvető szerepet játszik az input-outputnál. (2.4.3.\Mutató típus) A mutató típus egy lényegében egyszerű típus, különlegessége hogy tartományának elemei tárcímek. Ezzel valósítható meg az indirekt címzés. Értéke egy tárbeli cím, így nem egy értékre, hanem egy tár adott területére, és az azon lévő értékre. Van egy speciális eleme, a NULL cím. (2.5\Nevesített konstans) 3 komponensből áll, a névből, típusból, értékből. A nevesített konstanst deklarálni kell. az értéke deklarációjánál eldől és később nem változtatható meg. Szerepe egyrészt, hogy sokszor előforduló értékeket "beszélő" nevekkel látunk el, így könnyebb rá utalni a szövegben. És a másik, talán fontosabb, hogy ha egy programban átakarjuk írni az értékét, nem kell minden előfordulásánál, hanem elég csak egy helyen, a deklarációs utasításban. A PICI kiér itt is fontos kérdésekre és válaszokra. (2.6\A változó) 4 komponensből állnak, ezek a következők: név, attribútumok, cím, érték. A név az egy egyedi azonosító. Az attribútumok olyan jellemzők mely a változó futás közbeni viselkedését határozzák meg, és a változóhoz deklaráció segítségével rendelődnek. (Itt szó esik az explicit, implicit, automatikus deklarációról). A változó címköponense a tárnak azt a részét határozza meg, ahol a változó értéke elhelyezkedik. Egy változóhoz cím rendelhető több módon, lehet statikus tárkiosztás, dinamikus tárkiosztás, és programozó által vezérelt tárkiosztás. MIndhárom esetben a programozónak meg kell tudnia szüntetni a címköponenst. Változó értékköponensének meghatározására rendelkezésre áll az értékkadó utasítás. Szó esik az inputról, kezdőérték adásról. (2.7\Alapelemek az egyes nyelvekben) C, bemutatja az aritmetikai típusokat és a származtatott típusokat, meg a void típust. Ábrázolásokról, deklarációkról esik szó, és a megadásuknak a formájáról.

#### 4.hét

(3\Kifejezések) Kifejezés felépítése, pre-in-post-fix fogalma. Kifejezés kiértékelés sorrendje, menete. Zárójelzések menete. A típuskényszerítés meg a típusegyenértékűségről esik szó. Konstans kifejezés. A C egy alapvetően kifejezésorientált nyelv. Tömbökről esik szó az elején. A C kifejezésfogalmának rekurzív definícióját is bemutatja (kifejezés, elsődleges kifejezés, balérték, ). C precedencia táblázatot ismerteti és mik találhatóak benne. Megmutatja hogy milyen jelölésekkel milyen operátorokat értelmezünk (pl: +, -, !).

#### 5.Hét

(4\Utasítások) Az utasításokról bővebben irtam a Utasítások című labor feladatomban, a PICI könyv is nagyjából ugyan azt tárgyalja. Szó esik a két nagy csoportról (deklarációs, végrehajtható). A végrehajtható utasítások a következők: értékkadó, üres, ugró, elágaztató, ciklusszervező, hívó, vezérlésátadó, I/O, egyéb. Ezeket a könyv egyesével is értelmezi, és ír róluk egy kissébb összefoglalást.

#### 6.Hét

(5\A programok szerkezete) Az eljárásorientált programnyelvekben a program szövege többé-kevésbé független, szuverén részekre, ún. programegységekre tagolható. Kérdésekkel és válaszokkal indul a fejezet, melyek belátást nyújtanak a programok működésébe. Alprogramok a következő, fogalma és használatukra tér ki, felépülése formálisan (fej vagy specifikáció, törzs vagy implementáció, vég) és komponensei (név, formális paraméterlista, törzs, környezet). Az alprogramoknak két fajtája van: eljárás és függvény.

Eztkövetően a hívásokra mutat példát. Alprogramok esetén típust paraméterként átadni nem lehet. A C például csak egyetlen paraméterátadási módot ismer. Ezek után kitért Ada,PL/I

### 7.hét

Paraméter kiértékelés, paraméter átadás. Paraméter átadási módok(érték,cím,eredmény,érték-eredmény,név,szöveg) szerinti átadásokról beszélhetünk). Végén itt is kitért a frotranra.

### 8.hét

Blokk fogalma, formalitása. ELhezése is említésre kerül, aktivizációja és befejezése is. Nem minden eljárásorientált nyelv ismeri. Hatáskör, elmondja a fogalmát, a név hatáskörét. Lokális név fogalma. Szintekről is szó esik, képes ábrával bemutatva. Mindig befelé terjed a hatáskör.

### 9.hét

Input output fogalom meghatározása. Lehet input állomány, output állomány, input-output állomány. Az előbbiek fő lényege az adatmozgatás(ennek létezik 3 formája is). Szó esik az állományokkal való dolgozás során történő lépésekéről is. Ezek után szót ejt még az implicit állományokról. Végső pontja a fejezetnek a különböző programnyelvek közötti különbségek, és megoldások.

### 11.hét

Kivételkezelés lényege hogy ha érzékeli a kivétel kezelő hogy történt egy adott esemény, akkor ez a programrész máshogy reagál, és csinál valamit. Kivételeknek van neve, ezért azonosíthatóak. Kivételek figyelelse tiltható és engedélyezhető is. Később feltesz pár kérdést is a könyv, melyek a kivételalkotásról szól, hatáskörükiről, és egyéb elméleti kérdések.

### 13.Hét

Említi a Neumann architektúrán felépült gépek lényegét, ismerteti a szál / process fogalmát. A processek kezelése operácoós rendszer szinten történik. Következnek ezek után a párhuzamos programozás nyelvi alapfogalmai, melyben kitért a kommunikációra, szinkronizációra, konkurenciára, kölcsönös kizáráusra. És végül elmondja a könyv hogy a párhuzamos programozás megvalósításához a programnyelvnek mivel kell rendelkeznie.

## Programozás bevezetés

[KERNIGHANRITCHIE]

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

A kifejezéses utasítás legtöbbször értékadások vagy függvényhívások, alakjuk a

kifejezés;

As összetett utasítás vagy a blokkot ott használjuk ahol elvileg csak egy utasítás lenne elhelyezhető de ha szeretnénk több utasítást is használni akkor segít nekünk ez a típus. Alakjai:

összetett\_utasítás:

```
{deklaraciólistaopc utasításlistaopc}
```

deklaraciólista:

```
deklaráció
```

```
deklaráció deklarációlista
 -utasításlista
 -utasítás
 -utasítás utasításlista
```

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

A feltételes utasításnál a gép minden esetben kiértékeli a kifejezést, és ha az érték nem nulla akkor az első utasítás kerül végrehajásra, ha nulla, akkor meg a második kifejezés kerül végrehajtásra. Az else C ben úgy működik hogy utolsó else nélküli if-hez kapcsolódik.

A while utasítás addig ismétlődik míg az értéke nem nulla, az érték vizsgálat minden végrehajtás előtt történik. Alakja:

```
while (kifejezés)
 utasítás
```

A do utasítás ugyan az mint a while, csak az értékvizsgálat minden az utasítás végrehajtás után történik. Alakja:

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

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

A for utasítás a következő a soron, az első kifejezés a ciklust inicializálja, a második azt a vizsgálatot határozza meg mely minden iterációt megelőz, és a vezérlés kilép a programból ha nullával válik egyenlővé, a harmadik kifejezés gyakrabban az egyet iterációk után történő inkrementálást határozza meg. Bármely kifejezés elhagyható.

A switch utasítás a kifejezés értékétől függően más utasításra vált át a vezérlés. Az eredménynek int típusnak kell lennie. A switchen belül bármelyik utasítás címkezhető egy vagy több

```
case állandó_kifejezés;
```

taggal. Kilépni a switchből a break utasítással lehet. Alakja:

```
switch (kifejezés)
utasítás
```

A break utasítás hatására befejeződik a breaket körülvevő legbelőle legelső while, do, for vagy switch.

```
break;
```

A continue utasítás hatására az utasítást körülvevő legbelőző while, do, for utasítás a ciklusfolytató részére adódik át, vagyis a ciklus végére.

```
continue;
```

A return a függvény hívójához tér vissza.

```
return;
```

A goto utasítás a feltétel nélküli vezérlés átadásnál használatos

```
goto azonosító;
```

A címkézett utasítás arra jó hogy, bármely utasítást megelőzhetik az

```
azonosító;
```

előtagok, amelyek címkeként szolgálnak, ezeket használja a goto utasítás.

A nulla utasítás alakja a

```
;
```

hordozhat címkét vagy üres ciklustörzset is képezhet.

A két fontos eltérés a verziók között, ami még nincs a régebbiben, hogy nincsenek // -típusú kommentek, és a for cikluson belül nem lehet deklarálni változót. Lásd.: forrásban

## Programozás

[BMECPP]

5.Hét

A fejezet a C++ nem objektumorientált újdonságaival kezdődik. Megmutatja különböző függvényeknek a C és C++ beli változatát, és a különbségeket ezek között elmagyarázza(pl.: main). Bool típus bevezetése. Többájtós sztringek de C stílusúkról esik szó. Változódeklaráció de mint utasítás, így létrehozhatunk ott változót ahol szükségünk van rá. Függvénynevek túlterhelése a következő. C ben nem lehet két azonos nevű függvény mert az azonosítja őket, de C++ ban már lehet, mert az azonosítás a függvénynév és az argumentumával történik, és ez elnevezéseknel előnyös. Alapértelmezett függvényargumentumok, tehát ha érték nélkül adunk meg egy argumentumot akkor az alapértelmezett értékével kerül meghívásra(pl. CreateWindow 100 as x,y értéke). Paraméterátadás referenciátipussal, itt is C programból indulunk ki példaként, mert itt kizárolag érték szerinti paraméterátadás történik csak. Itt szó esik még a pointerekről. Majd bemutatja hogy hogy alakíthatunk át egy C példát C++ ra referenciátipussal.

6.hét

Megismertet az objektumorientáltság alapjaival, hátteret biztosít, és informatika történelmet mesél. Bevezeti az objektumokat, öröklés fogalmát és a tipus támogatást tipus váltásokat. Példát is hoz koordinátarendszer beli számításokra, ezzel a strukturáltságot mutatja meg(itt szó esik az attribútumokról és a metódusról is). Tagváltozókat mutatja meg, tagfüggvényeket és ezeket példában is megmutatja, és kitért a C-hez képest található különbségre. Következő volt a data hiding, itt kitér a programot írók problémájára. Ezek

után szó esik a példányosításról objektumról és hoz pár példát is. Következő a konstruktor és destruktur, itt a sebezhetőségről esik szó, és az alapértelmezett inicializálásról, kitér még a konstruktorok részleteire melyek érdekesek és sokat használatosak. DEstruktorra csak röviden tér ki, megmutatja mivel kezdődik, és leirja hogy a felszabadítás a lényege. A következő rész a dinamikus memóriakezelés, itt is a C re tér ki először példával, majd bemutatj a C++ os megoldást. FOntos szót ejteti a másoló konstruktorról is, mivel azt is bemutatja, de mi azt már jól ismerjük a LZWBinafa alapján. Következő fejezet volt a Friend függvények és osztályok, utánuk jött a tagváltozók inicializálása rész, melyre már itértünk picit a konstruktoroknál. Beágyazott definíciók megismertetése, és C++ hoz kapcsolódása, rengeteg példával.

## 7.hét

Operátorok és túlterhelésük. C++ előzményekkel kezdődik, és beszél az operátorokról általában is. Fügveny szintaxis és a túlterhelés a következő, itt is sok sok példa és kódcsipetke.

## 9.Hét

Először kitér szabványos adatfolyamokra, másnéven streamekre, szót ejt az istream,ostream ről, melyek a be és kimeneti adatfolyamok osztályai. Objektumokat is ismertet, mintpéldául a cin,cout,cerr... Eztkövetően sok példát hoz a bemenetre, sok szám szöveg és egyéb bekérésekre,ezekkel műveletekre, és különböző kiiratásokra. Később rátér a manipulátorokra,ezekkel a streameket lehet manipulálni, például a cout << flush. Az előre definiált manipulátorok megtalálhatóak a iomanip névtérben. Kitér megint a C ben használt megoldásokra, és utána megmutatja a C++ os megoldást. Végső soron az állománykezelést veszi, melyek az if,of,fstream névre hallgatnak. Ismét C-s példával kezd. KItér a konstruktorokra és destruktorkra újra. Jelzőbit használat különféle fájl megnyitásokra.

## 11.hét

Kivételkezelésről esik szó, először hoz egy példát hogy elmagyarázza az értelmi lényegét. Hozott egy problémát amit egy kódcsipetben meg is old. Következenek ezek után az alapok, ha például kivétel van akkor a kivételkezelőre ugorjon. Példát ad arra hogy ha számot szeretnénk bekérni inputba, és nem azt kapunk akkor kezelhetjük kivételként erre a példára láthatunk is egy kódcsipetet melyben nem nulla számot kér, és ha 0 át kapna mégis akkor dob egy exceptiont. Majd később egy táblázatban is összefoglalja az előnyeit annak ha használunk kivételkezelést. Majd még hoz sok sok példát sok féle különböző kivételekre. Szó esik az egymásba ágyazott try-catch ről is. Végül megmutatja a stack rewindot, melyre hoz egy példát, és elmagyarázza hogy ha f2 kivételel dob, akkor mi történik. Egy láncszerű esemény fut le, ha exceptiont kap el.

## **III. rész**

# **Második felvonás**

**Bátf41 Haxor Stream**

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

## 11. fejezet

### Helló, Arroway!

#### A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

#### Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

---

## IV. rész

### Irodalomjegyzék

## Általános

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

## C

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

## C++

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

## Lisp

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

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