

Univerzális programozás

Így neveld a programozód!

Ed. BHAX, DEBRECEN,
2020. március 2, v. 0.0.5

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, 2020, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

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

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

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

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

DRAFT

COLLABORATORS

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai Norbert, Bátfai Mátyás, Bátfai Nándor, Bátfai Margaréta, és Tutor Tünde	2020. május 5.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2020-03-02	Az Chomsky/ $a^n b^n c^n$ és Caesar/EXOR csokor feladatok kiírásának aktualizálása (a heti előadás és laborgyakorlatok támogatására).	nbatfai

Ajánlás

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

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

DRAFT

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	10
2.3. Változók értékének felcserélése	12
2.4. Labdapattogás	14
2.5. Szóhossz és Linus Tovald -féle BogoMIPS	17
2.6. Hello Google!	18
2.7. Monty Hall probléma	21
2.8. 100 éves a Brun Tétel	29
2.9. MALMÖ folytonos csiga feladat	30
3. Helló, Chomsky!	33
3.1. 3.1 Decimálisból unárisba átváltó Turing gép	33
3.2. 3.2 Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	33
3.3. 3.3 Hivatkozási nyelv	34
3.4. 3.4 Saját lexikális elemző	35
3.5. 3.5 Leetspeak	35

3.6. 3.6 A források olvasása	38
3.7. 3.7 Logikus	39
3.8. 3.8 Deklaráció	39
3.9. 3.9 MALMÖ Discrete csiga	43
4. Helló, Caesar!	44
4.1. 4.1 Alsó háromszögmátrix	44
4.2. 4.2 C EXOR titkosító	46
4.3. 4.3 Java EXOR titkosító	47
4.4. 4.4 C EXOR törő	48
4.5. 4.5 Neurális OR, AND és EXOR kapu	51
4.6. 4.6 Hiba-visszaterjesztéses perceptron	53
4.7. 4.7 Steve látása	54
5. Helló, Mandelbrot!	55
5.1. 5.1 A Mandelbrot halmaz	55
5.2. 5.2 A Mandelbrot halmaz a <code>std::complex</code> osztálytal	58
5.3. 5.3 Biomorfok	61
5.4. 5.4 A Mandelbrot halmaz CUDA megvalósítása	66
5.5. 5.5 Mandelbrot nagyító és utazó C++ nyelven	68
5.6. 5.6 Mandelbrot nagyító és utazó Java nyelven	68
5.7. 5.7 Steve felszalad a láváig	69
6. Helló, Welch!	70
6.1. 6.1 Első osztályom	70
6.2. 6.2 LZW	71
6.3. 6.3 Fabejárás	72
6.4. 6.4 Tag a gyökér	75
6.5. 6.5 Mutató a gyökér	75
6.6. 6.6 Mozgató szemantika	76
6.7. 6.7 Steve szemüvege	78
7. Helló, Conway!	79
7.1. 8.1 Hangyszimulációk	79
7.2. 8.2 Java életjáték	81
7.3. 8.3 Qt C++ életjáték	82
7.4. 8.4 BrainB Benchmark	87
7.5. 8.5 Malmö 19 RF	88

8. Helló, Schwarzenegger!	89
8.1. Szoftmax Py MNIST	89
8.2. Mély MNIST	93
8.3. Minecraft-MALMÖ	94
8.4. Minecraft-MALMÖ	95
9. Helló, Chaitin!	96
9.1. Iteratív és rekurzív faktoriális Lisp-ben	96
9.2. Malmö kód továbbfejlesztése	99
10. Hello, Gutenberg!	106
10.1. 10.1. Programozási alapfogalmak	106
10.2. Keringhan.Ritchie: A C programozási nyelv	106
10.3. 10.4. Python bevezetés	107
10.4. 10.2. Szoftverfejlesztés C++ nyelven	108
III. Második felvonás	110
11. Helló, Arroway!	112
11.1. A BPP algoritmus Java megvalósítása	112
11.2. Java osztályok a Pi-ben	112
IV. Irodalomjegyzék	113
11.3. Általános	114
11.4. C	114
11.5. C++	114
11.6. Lisp	114

Ábrák jegyzéke

4.1. A double ** háromszögmátrix a memóriában	46
5.1. A Mandelbrot halmaz a komplex síkon	56

DRAFT

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: <https://youtu.be/lvmi6tyz-nI>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c, bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-w.c.

Számos módon hozhatunk és hozunk létre végtelen ciklusokat. Vannak esetek, amikor ez a célunk, például egy szerverfolyamat fusson folyamatosan és van amikor egy bug, mert ott lesz végtelen ciklus, ahol nem akartunk. Saját péláinkban ilyen amikor a PageRank algoritmus rázza az 1 liter vizet az internetben, de az iteráció csak nem akar konvergálni...

A végtelen ciklusok léjtjogosultsága eksőre kicsit meglepő lehet a kezdő programozóknak, miért is akarnánk, hogy egy program ne álljon le? Pedig egyébként sokszor nagyon hasznosak. Én egyébként szoktam gyerekekkel foglalkozni, és már velük is volt szó végtelen ciklusokról. A Microsoft Kodu-ban dolgozunk általában, átlag 6 éves korosztállyal és egy akadálypályás játéket kellett készítenünk. A cél az volt, hogy a karakterünkkel át kelljen kelni a szemben oda-vissza mozgó biciklisek között. A biciklisek kódja volt itt a kérdéses, megadott útvonalon kellett oda-vissza haladniuk addig amíg a játék tart. A kulcsó itt amíg a játék tart, persze nem olyan végtelen ciklusról volt szó amivel mi foglalkozunk, de szerintem ez a példa szemléletes. Tehát el kellett érünk, hogy a biciklisek ne hagyják abba mozgásukat. A gyerekekkel ezt a feladatot úgy oldottuk meg, hogy a feltétel megalkotásához 2 blokkot használtunk fel, az egyik a "lát" a másik pedig a "bármí". Tehát ameddig akármit lát a biciklis mozogni fog az útvonalán, és ugye valamit az adott környezetben mindig lát tehát nem fog a cilusból kilépni. Egyébként a képen látszik, hogy itt "ha" feltételeket szabunk, de a labdapattogtatós feladatban majd láthatjuk milyen egyszerű ezekből ciklust készíteni.



Egy mag 100 százalékban:

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

    return 0;
}
```

vagy az olvashatóbb, de a programozók és fordítók (szabványok) között kevésbé hordozható

```
int
#include <stdbool.h>
main ()
{
    while(true);

    return 0;
}
```

The screenshot shows two terminal windows side-by-side. Both windows have the title 'tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/bhx programok/Turing'. The left window displays a command-line session where the user runs 'gcc infny-f.c' and 'gcc infny-w.c' respectively. The right window shows the system's resource usage with the 'top' command. The top panel of the 'top' window provides a summary of system load (Tasks: 104, 652 thr; 2 running), memory usage (Mem: 7.99G/47.1G), and system statistics (Load average: 0.49 0.41 0.45, Uptime: 07:34:53). Below this is a detailed process list with columns for PID, USER, PRI, NI, VIRT, RES, SHR, %CPU, %MEM, TIME+, and Command. Numerous processes are listed, including multiple instances of 'firefox' and 'pulseaudio'. The bottom of the 'top' window has a series of keyboard shortcut keys: F1, F2, F3, F4, F5, F6, F7, F8, F9, F10.

Az első végtelen ciklus egy processzor magot használ 100%-ban. Ez úgy lehetséges hogy itt egy while ciklust használunk ami folyamatosan vizsgálja a megadott feltételt, ehhez 1 CPU magot használ viszont annak a teljes teljesítményét muszáj kihasználnia hogy folyamatosan figyelje a ciklus feltételét. Ugyanez a helyzet igazából egy végtelen for ciklus esetén is, megjegyezném, hogy az assembly kódjuk megegyezik.

Azért érdemes a `for (;;)` hagyományos formát használni, mert ez minden C szabvánnyal lefordul, másrészről a többi programozó azonnal látja, hogy az a végtelen ciklus szándékunk szerint végtelen és nem szoftverhiba. Mert ugye, ha a `while`-al trükközünk egy nem triviális 1 vagy `true` feltétellel, akkor ott egy másik, a forrást olvasó programozó nem látja azonnal a szándékunkat.

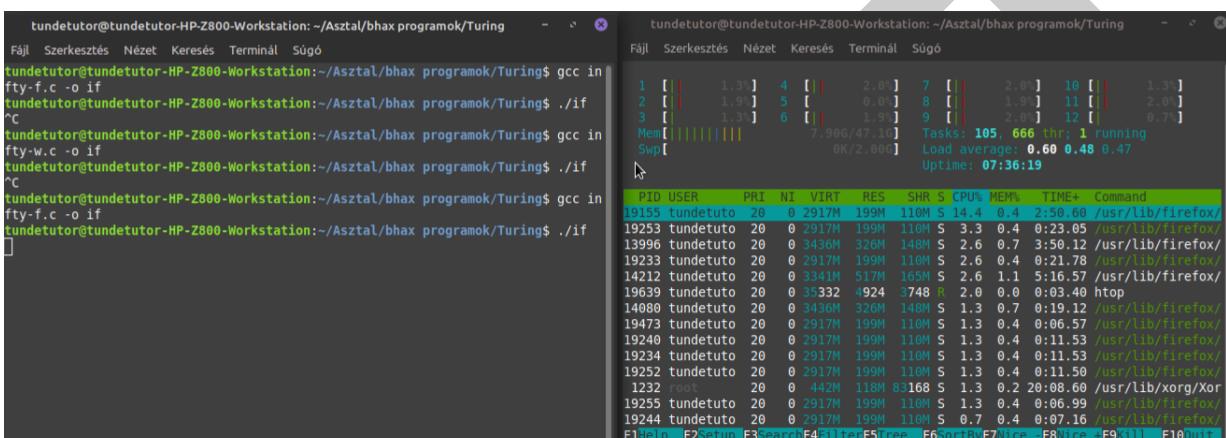
Egyébként a fordító a `for`-os és `while`-os ciklusból ugyanazt az assembly kódot fordítja:

```
$ gcc -S -o infny-f.S infny-f.c
$ gcc -S -o infny-w.S infny-w.c
$ diff infny-w.S infny-f.S
1c1
< .file "infny-w.c"
```

```
---
> .file "infty-f.c"
```

Egy mag 0 százalékban:

A második ciklus egy cpu magot közel 0%-ban használ. Ehhez használtunk egy for ciklust, alapvetőleg minden kettőtől 1 magot használ 100%-ban. Ahhoz hogy a cpu mag 0%-ot használjon használnunk kell a sleep(1) függvényt ami a számolást késlelteti ezért CPU rá van kényszerülve a késleltetésre, viszont így sem tudjuk elérni hogy pontosan 0%-on menjen, inkább közelít a 0-hoz, a gyakorlat azt mutatja hogy így kb 1%-on megy. A képen egyébként ha megnézzük, az 5. CPU magnál látunk egy 0.0 értéket, tehát hozzá került a kérdéses ciklus.



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

    return 0;
}
```

Minden mag 100 százalékban:

A harmadik ciklusunk minden cpu magot 100%-osan használ. Ehhez szükségünk van egy for ciklusra és az omp.h nevű fájlra. Ennek segítségével párhuzamosan engedi használni egy feladatra a processzunk minden magját. Alapvetőleg ha egy ilyen ciklust futtatunk egy magot használ ki 100%-ban, a tesztjeim során azt vettem észre, hogy éppen melyik magot használja az váltakozhat. Fordítani és futtatni egyébként a következőképpen tudjuk ezt a programot:

```
gcc -fopenmp filenev.c -o fajlnev
```

Nos így néz ki ha lefutott. Nagyon kíváncsi voltam egyébként ennek a programnak a futására, mivel 2 CPU-t használok és érdekelte hogy így is működik-e dolog, viszont ahogy láthatjuk működött.

```
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ gcc infy-f.c -o if
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ ./if
^C
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ gcc infy-w.c -o if
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ ./if
^C
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ gcc infy-f.c -o if
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ ./if
^C
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ gcc vgetelen_omp.c -o vo
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ ./vo
^C
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ gcc -fopenmp vgetelen_omp.c -o vo
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ ./vo
^C
```

Fájl Szerkesztés Nézet Keresés Terminál Súgó

Fájl Szerkesztés Nézet Keresés Terminál Súgó

PID	USER	PRI	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	Command
19922	tundetuto	20	0	98M	912	816	R	118%	0.0	1:06.67	/vo
19924	tundetuto	20	0	98M	912	816	R	99.7	0.0	0:05.59	/vo
19927	tundetuto	20	0	98M	912	816	R	99.7	0.0	0:05.59	/vo
19929	tundetuto	20	0	98M	912	816	R	99.7	0.0	0:05.59	/vo
19931	tundetuto	20	0	98M	912	816	R	99.7	0.0	0:05.59	/vo
19930	tundetuto	20	0	98M	912	816	R	99.7	0.0	0:05.58	/vo
19932	tundetuto	20	0	98M	912	816	R	99.7	0.0	0:05.56	/vo
19923	tundetuto	20	0	98M	912	816	R	97.8	0.0	0:05.50	/vo
19933	tundetuto	20	0	98M	912	816	R	96.4	0.0	0:05.48	/vo
19155	tundetuto	20	0	2917M	199M	110M	S	5.9	0.4	3:11.03	/usr/lib/firefox/
13996	tundetuto	20	0	3468M	348M	162M	S	2.0	0.7	4:01.43	/usr/lib/firefox/

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

A **gcc infy-f.c -o infy-f -fopenmp** parancssorral készítve a futtathatót, majd futtatva, közben egy másik terminálban a **top** parancsot kiadva tanulmányozzuk, mennyi CPU-t használunk:

```
top - 20:09:06 up 3:35, 1 user, load average: 5,68, 2,91, 1,38
Tasks: 329 total, 2 running, 256 sleeping, 0 stopped, 1 zombie
%Cpu0 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu6 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu7 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem :16373532 total,11701240 free, 2254256 used, 2418036 buff/cache
KiB Swap:16724988 total,16724988 free, 0 used. 13751608 avail Mem

 PID USER      PR  NI      VIRT      RES      SHR S %CPU %MEM     TIME+ COMMAND
 5850 batfai    20    0    68360     932      836 R 798,3  0,0   8:14.23 infy-f
```



Werkfilm

- <https://youtu.be/lvmi6tyz-nl>

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

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

Megoldás videó:

A példában említett „Lefagy” függvénynek bármely programról el kell tudnia dönteni hogy tartalmaz-e végtelen ciklust. Ez nem feltétlen valósulhat meg, pl.: ha tartalmaz végtelen ciklust akkor "true" értékkel kell visszatérnie viszont ha nem tartalmaz akkor elindít egy végtelen ciklust. Ekkor ha a T1000 program „nem lefagyó” akkor mindenképp lefagy mert ha a „Lefagy” függvény nem "true" értékkel tér vissza elindítja a végtelen ciklust. Tehát ilyen esetben saját magáról nem tudja eldöntenи hogy van-e benne végtelen ciklus avagy sem. Azaz gyakorlatilag egy paradoxon jelenséggel állunk szemben. Megjegyzés: Néhány fejlesztői környezet jelzi ha a programunkban van végtelen ciklus de ezek is többnyire csak a legegyszerűbbekre tudnak szűrni szintaktika alapján pl.: ha a programunk tartalmaz `while(true)`, `while(1)`, vagy `for(; ;)`, ciklusokat. Amikor először próbáltam az első feladathoz írni ilyen ciklust nekem a "Geany" fejlesztői környezetben való futtatáskor néhány másodperc után kilépett a ciklusból.

Megoldás forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a Lefagy függvényt, amely tetszőleges programról el tudja döntenи 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(;;);
    }

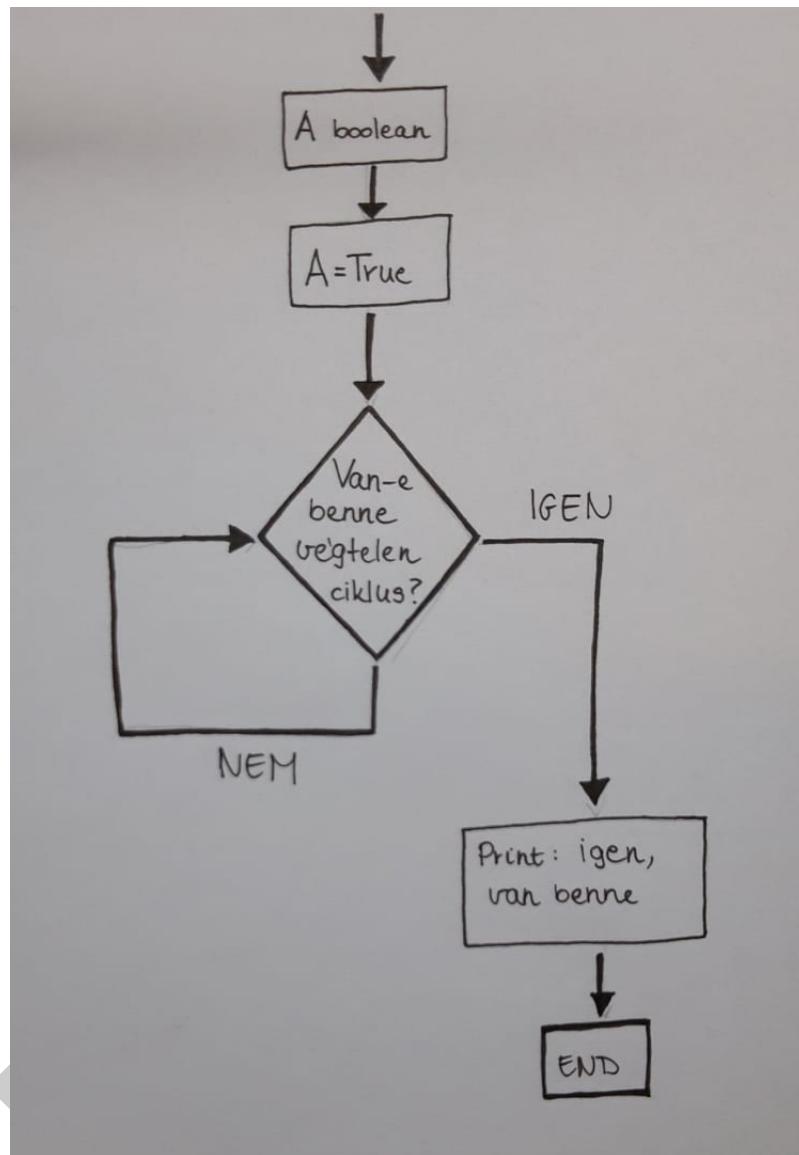
    main(Input Q)
    {
        Lefagy2(Q)
    }
}
```

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.

A feladattal kapcsolatban készítettem egy folyamat ábrát. Ha nem igaz hogy tartalmaz végtelen ciklust, akkor visszatérünk újra ehhez az elágazáshoz, tehát lefagy a program. Amennyiben tartalmaz, kiírja tehát ellentmondásos.



2.3. Változók értékének felcserélése

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

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

Megoldás forrása: ugyanott.

Logikai utasítások nélkül a változók cseréjének legegyszerűbb módja segédváltozóval megcserélni az értékeket de egyéb megoldások is vannak pl.: az összeadásos vagy kivonásos. Logikai utasítással is meg tudjuk ezt tenni, a legjobb példa erre a bitenkénti kizáró vagy (XOR). Az én programom 3 fajta változócsereit tud végrehajtani, és minden futtatáskor véletlenszerűen választott módszerrel teszi ezt meg de nyílván az eredmény mindenkorban ugyanaz lesz.

```
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_text
Fájl Szerkesztés Nézet Keresés Terminál Súgó
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_text
book_IgyNeveldaProgramozod/Turing$ gcc valtozocsere.c -o vcs
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_text
book_IgyNeveldaProgramozod/Turing$ ./vcs
1Minden futattasaskor egy random kivalasztott modeszert hasznalok.
Irj be 2 megcserelendo valtozot!
2 7
a = 2 es b = 7
A valtozokat az osszeadasos modszerrel cserelem meg.
Ekkor a valtozocsere utan: a = 7 es b = 2.
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_text
book_IgyNeveldaProgramozod/Turing$ ./vcs
2Minden futattasaskor egy random kivalasztott modeszert hasznalok.
Irj be 2 megcserelendo valtozot!
6 8
a = 6 es b = 8
A valtozokat az kivonásos modszerrel cserelem meg.
Ekkor a valtozocsere utan: a = 8 es b = 6.
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_text
book_IgyNeveldaProgramozod/Turing$ ./vcs
2Minden futattasaskor egy random kivalasztott modeszert hasznalok.
Irj be 2 megcserelendo valtozot!
18 3
a = 18 es b = 3
A valtozokat az kivonásos modszerrel cserelem meg.
Ekkor a valtozocsere utan: a = 3 es b = 18.
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorialbhax_text
book_IgyNeveldaProgramozod/Turing$ ./vcs
0Minden futattasaskor egy random kivalasztott modeszert hasznalok.
Irj be 2 megcserelendo valtozot!
18 3
a = 18 es b = 3
A valtozokat segedvaltozo hasznalatalaval cserelem meg.
Ekkor a valtozocsere utan: a = 3 es b = 18.
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_text
book_IgyNeveldaProgramozod/Turing$
```

Így néz ki a teljes program:

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

int main()
{
    srand(time(NULL));
    int r = rand() % 3;
    printf("%d", r);
    int a;
    int b;
    int seged;
    printf("Minden futattasaskor egy random kivalasztott modeszert ←
          hasznalok.\nIrj be 2 megcserelendo valtozot!\n");
    scanf("%d%d", &a, &b);
    if( r == 0)
    {
        printf("a = %d es b = % d\nA valtozokat segedvaltozo hasznalatalaval ←
              cserelem meg.", a, b);
        seged = a;
        a = b;
        b = seged;
        printf("\nEkkor a valtozocsere utan: a = %d es b = %d.\n", a, b);
    }
    else if( r == 1)
    {
```

```
printf("a = %d es b = % d\nA valtozokat az osszeadasos modszerrel ←
      cserelem meg.", a, b);
a = a + b;
b = a - b;
a = a - b;
printf("\nEkkor a valtozocsere utan: a = %d es b = %d.\n", a, b);
}
else
{
    printf("a = %d es b = % d\nA valtozokat az kivonasos modszerrel ←
          cserelem meg.", a, b);
    a = a - b;
    b = a + b;
    a = b - a;
    printf("\nEkkor a valtozocsere utan: a = %d es b = %d.\n", a, b);
}
return 0;
}
```

A segédváltozós módszer használata:

```
seged = a;
a = b;
b = seged;
```

A összeadásos módszer használata:

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

A kivonásos módszer használata:

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

2.4. Labdapattogás

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

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

Megoldás forrása: ugyanott.

A labdapattogató feladat lényege hogy a képernyőt úgy tudjuk használni mint egy koordináta rendszert (mivel az is). A kiválasztott karakterünket, akár az O betűt megadott pályán tudjuk, elindítani, itt fontos hogy kicsit elcsúsztatva kezdjük el a pattogtatást hogy be tudja járni az egész képernyőt. Ha pl x egyenlő

2, y egyenlő 2 értékeken indítjuk el akkor csak a képernyő egy átlóját tudja bejárni mivel csak azon az útvonalon fog oda-vissza pattogni. A képernyőnket mátrixként is lehet értelmezni.

Viszont nézzük meg a C programot! Itt az if-eket úgy kerüljük me, hogy for ciklusokat használunk az if helyett, mégpedig úgy, hogy nem adunk kezdőértéket a ciklusváltozónak és nem is növeljük azt, ez egy gyönyörű for ciklusnak álcázott if függvény. És itt van a teljes kód is:

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

//gcc bouncy0.c -o bouncy0 -lncurses

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

    int x = 0;
    int y = 0;

    int delX = 1;
    int delY = 1;

    int mx;
    int my;

    while(true)
    {
        printf("\033[0;32m");

        getmaxyx(ablak, my, mx);
        mvprintw(y, x, "O");

        refresh();
        usleep(100000);

        clear();

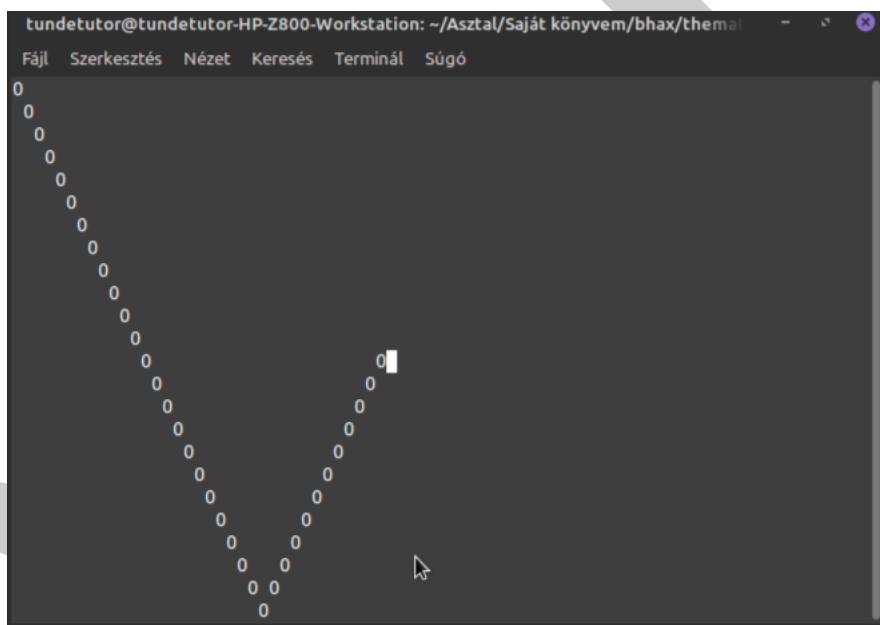
        x = x + delX;
        y = y + delY;

        for(;x <= 0;)
        {
            delX = delX * -1;
            break;
        }
        for(;x >= mx-1;)
        {
            delX = delX * -1;
        }
    }
}
```

```
        break;
    }
    for(;y <= 0;)
    {
        delY = delY * -1;
        break;
    }
    for(;y >= my-1;)
    {
        delY = delY * -1;
        break;
    }
}

return 0;
}
```

Képen annyira nem látszik mi történik, szóval a képhez kiszedtem a kódöt így látszik szépen milyen utat jár be a labdánk.



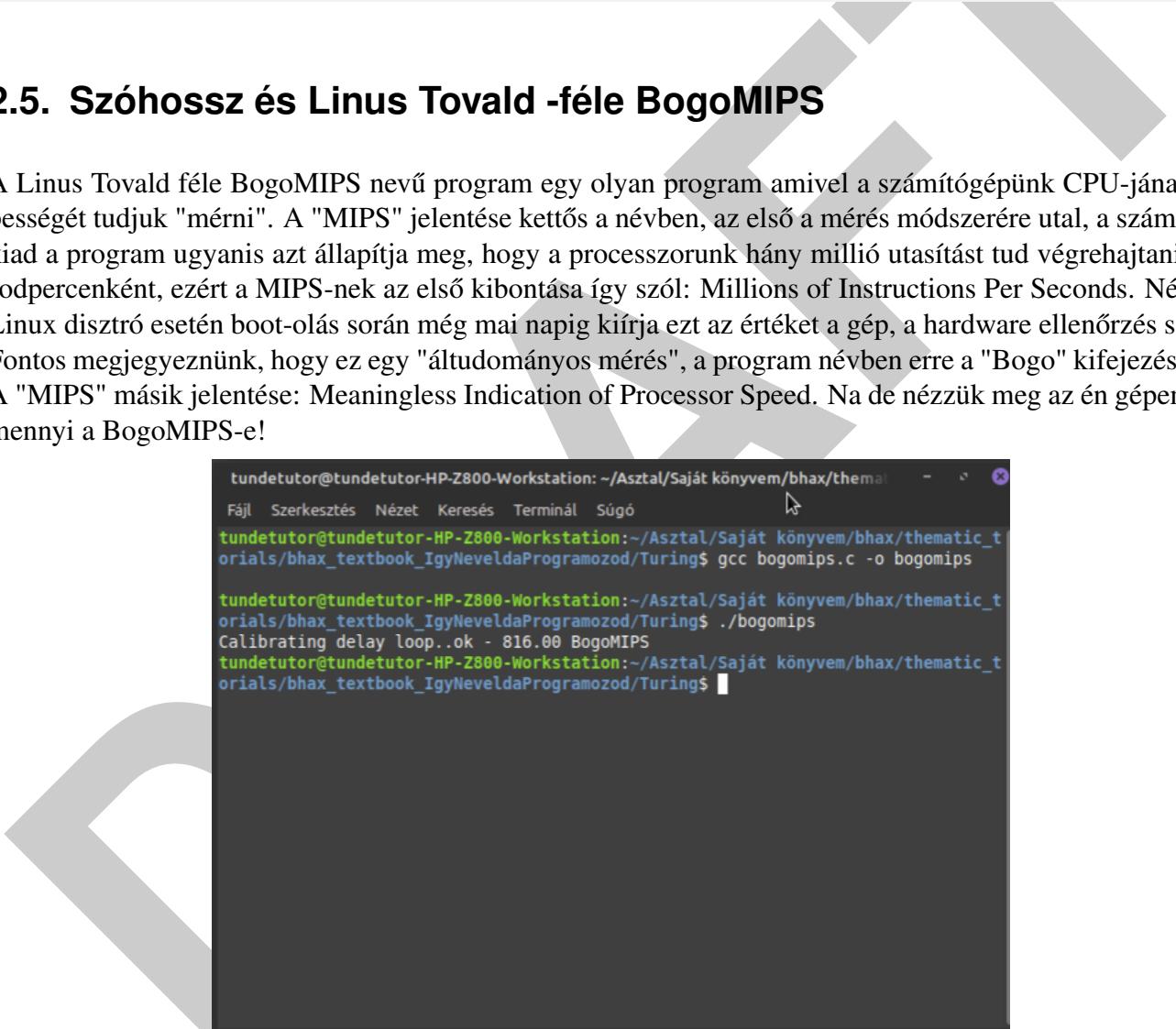
Vizsont azt is tudjuk, hogy a for és while ciklusokkal ügyesen lehet variálgatni, szóval a for ciklusokat kicseréltem while ciklusokra és így is tökéletesen lefutott a program. Az alábbi kódcsipetben látható a while ciklusos megoldás is, illetve kommentben mellettük a for ciklus testvérei a while-jainknak.

```
while( !(x <=0) )           //for(;x <= 0;)
{
    delX = delX * -1;
    break;
}
while( !(x >= mx-1) )         //for(;x >= mx-1;)
{
    delX = delX * -1;
    break;
}
```

```
while( !(y <= 0) )           //for(;y <= 0;)
{
    delay = delay * -1;
    break;
}
while( !(y >= my-1) )         //for(;y >= my-1; )
{
    delay = delay * -1;
    break;
}
}
```

2.5. Szóhossz és Linus Tovald -félé BogoMIPS

A Linus Tovald féle BogoMIPS nevű program egy olyan program amivel a számítógépünk CPU-jának sebességét tudjuk "mérni". A "MIPS" jelentése kettős a névben, az első a mérés módszerére utal, a szám amit kiad a program ugyanis azt állapítja meg, hogy a processzorunk hány millió utasítást tud végrehajtani másodpercenként, ezért a MIPS-nek az első kibontása így szól: Millions of Instructions Per Seconds. Néhány Linux disztró esetén boot-olás során még mai napig kiírja ezt az értéket a gép, a hardware ellenőrzés során. Fontos megjegyeznünk, hogy ez egy "áltudományos mérés", a program névben erre a "Bogo" kifejezés utal. A "MIPS" másik jelentése: Meaningless Indication of Processor Speed. Na de nézzük meg az én gépemnek mennyi a BogoMIPS-e!



```
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing$ gcc bogomips.c -o bogomips
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing$ ./bogomips
Calibrating delay loop...ok - 816.00 BogoMIPS
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing$
```

Láthatjuk, hogy ez egy 816-os érték. Megjegyezném, hogy itthon van mégégy hasonló workstation, abban jelenleg 2 db Xeon X5660-as processzort találunk, amikor még csak egy olyan volt benne akkor azon is végrehajtottam ezt a mérést és szintén 800 körül értéket kaptam. Az én gépem most szintén 2 db nagyon hasonló X5650-s processzorral van megáldva, szerintem nem ugyanaz a számítási kapacitás tehát találó a "Bogo" jelző. Viszont térdünk rá a szóhosszra!

Ennél a feladatnál a BogoMIPS program while ciklusának fejét kell felhasználnunk ami: **while(loops_per_sec <= 1)** formájú. Itt az történik a rövidke programon belül, hogy belül, hogy a bitshifttel, addig shifteljük

balra a biteket amíg el nem érünk a végéig. A **szo** változónk értéke most éppen 1, tehát áll valamennyi 0-ból a baloldalán, ha pedig jobb oldalról olvassuk akkor 1 db 1-essel kezdődik, ha megtudjuk mennyi 0 áll az 0-es előtt megtudjuk hány biten tárolódik, ugye ehhez kell a bitshift.

```
#include <stdio.h>

int main()
{
    int szohossz = 0, szo = 1;

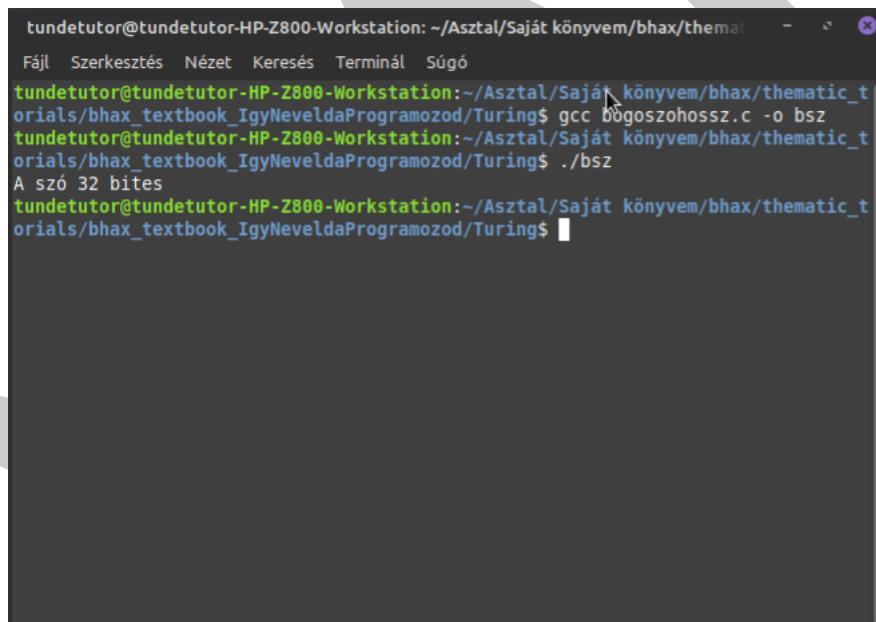
    do
    {

        szohossz++;

    } while(szo <= 1);

    printf("A szó %d bites\n", szohossz);

    return 0;
}
```



```
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_torials/bhax_textbook_IgyNeveldaProgramozod/Turing$ gcc b0goszohossz.c -o bsz
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_torials/bhax_textbook_IgyNeveldaProgramozod/Turing$ ./bsz
A szó 32 bites
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_torials/bhax_textbook_IgyNeveldaProgramozod/Turing$
```

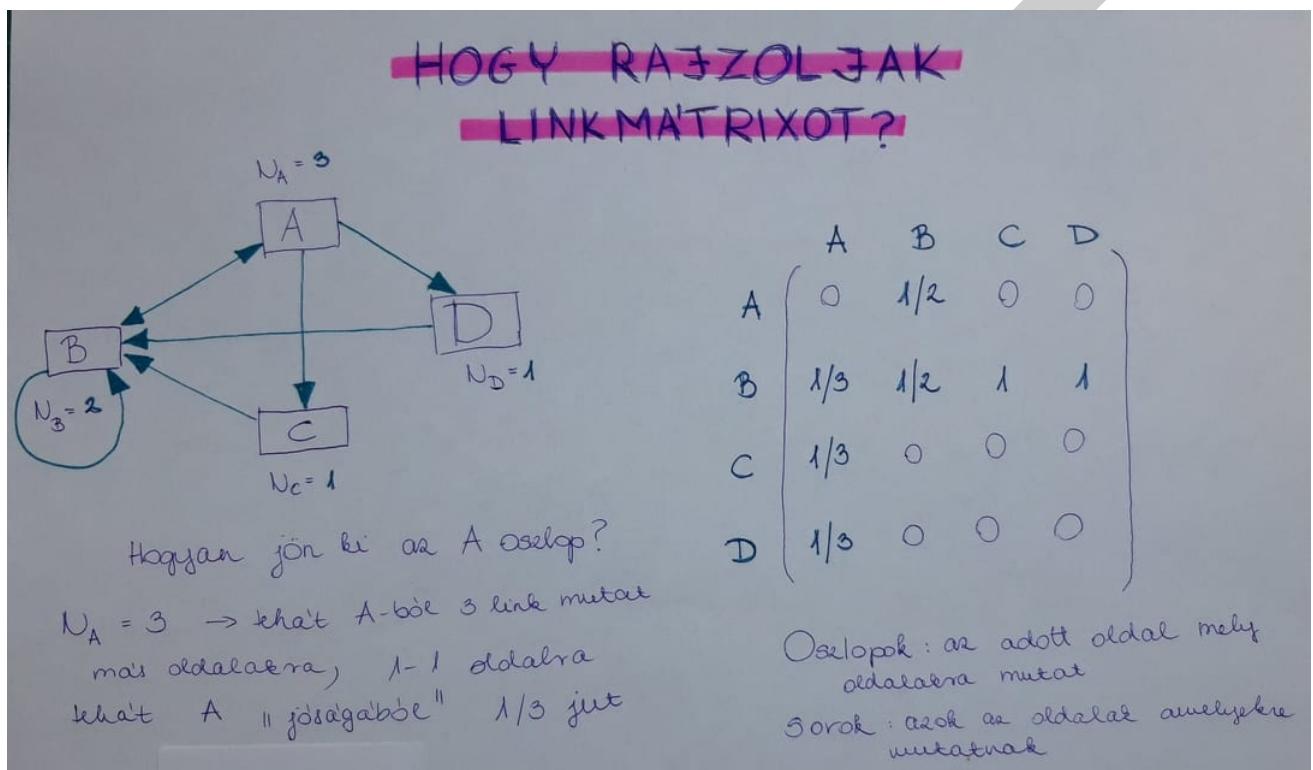
Fordítás és futtatás után megtudtuk, hogy 32 bites az 1-esünk. tehát áll 1 db 1-estől ami előtt tőle balra 31 db 0 szerepel.

2.6. Hello Google!

Ha csak nem más kereső motort használunk, akkor naponta nagyon sok alkalommal veszsük elő a jól ismert Google-t és keresünk rá valami kifejezésre. Amikor keresünk, reménykedünk benne, hogy a szükséges információ szerepelni fog az első pár találatban. De gondolkodtunk-e már rajta, hogy hogyan is talája meg a kereső motor a szémünkra leghasznosabb oldalakat?

Igazából ez is mint nagyon sok más dolog az emberi tényezőtől függ elsősorban, mégpedig ez most azt jelenti, hogy akik bizonyos weboldalakat szerkesztenek, mely más oldalakat linkelnek be a sajátjukra. Az alapja az algoritmusnak, hogy sorba kell rendezni a találatokat, aszerint kell sorbarendezni őket, hogy az úgynevezett "PageRank" értékük mekkora, a PageRank pedig az adott weboldal minőségére utaló tulajdon-ság.

Ehhez a feladathoz én készítettem egy ábrát, az alapja a Bátfai Norbert által készített diasor, ezt be is linkelem a kép alá. Csak kicsit átírtam rajta a jelöléseket, picit könnyebben emészhetővé próbáltam alakítani.



Forrás: [Bátfai Norbert diasora](#)

Ez a rendszer gyakorlatilag egy 4 db honlapból álló gráf. A honlapokat nevezzük el A, B, C és D-nek. Mindegyikük rendelkezik egy "N" nevű tulajdonsággal, ami egy számmal írható le, ez a szám azt jelenti, hogy az adott oldalból hány db link fut ki. Az ábrán látható 4 honlap esetén $N(A)=3$, $N(B)=2$, $N(C)=1$ és $N(D)=1$. Belőük képeznünk kell egy linkmátrixot. A linkmátrix is fel van rajzolva, oszlopfoltonos módon a legegyszerűbb felrajzolnunk, most csak az A oszlopot írom le részletesen, utána a többi már magától értetődő lesz. Az A oszlop esetében nézzük az $N(A)$ értéket, ez ugye 3, tehát A PageRank értékét egyenlően el kell osztanunk azon 3 honlap között amelyekre mutat, most minden PageRank legyen először 1, ekkor a mátrixban A oszlopában A-hoz 0 kerül mivel magára nem mutat, a többi 3 (B, C és D) oldal esetében pedig 1/3-mmal kell számolnunk, tehát A a saját "jóságát" egyenlően osztja el.

A programban a main függvényen belül találjuk az elkészült linkmátrixot.

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

```
{0.0, 0.0, 1.0/3.0, 0.0}  
};  
  
double PR[4] = {0.0, 0.0, 0.0, 0.0};  
double PRv[4] = {1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0};
```

Vegyük észre, hogy végtelen for ciklussal találkozunk, ugyanis ezeket a PageRank számításokat a szumma miatt a végtelenségig lehet számítani, minél többször fut le a ciklus annál pontosabb PageRank értéket kapunk a számítások végén. Ebből akkor tud kilépni, ha a **distance** kisebb a megadott 0.00001 értéknél, ami ugye egy nevezetes azonosság gyöke lesz, ekkor kapja meg a breaket a for ciklusunk.

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

Az itt található dupla for ciklusra azért van szükség, mert mátrix-szal dolgozunk.

```
for(int i = 0; i < 4; i++)  
{  
    double tmp = 0.0;  
    for(int j = 0; j < 4; j++)  
    {  
        tmp += L[i][j] * PR[j];  
    }  
    PRv[i] = tmp;  
}  
if(distance(PR, PRv, 4) < 0.00001)  
{  
    break;  
}  
}  
kiir (PR, 4);  
return 0;  
}
```

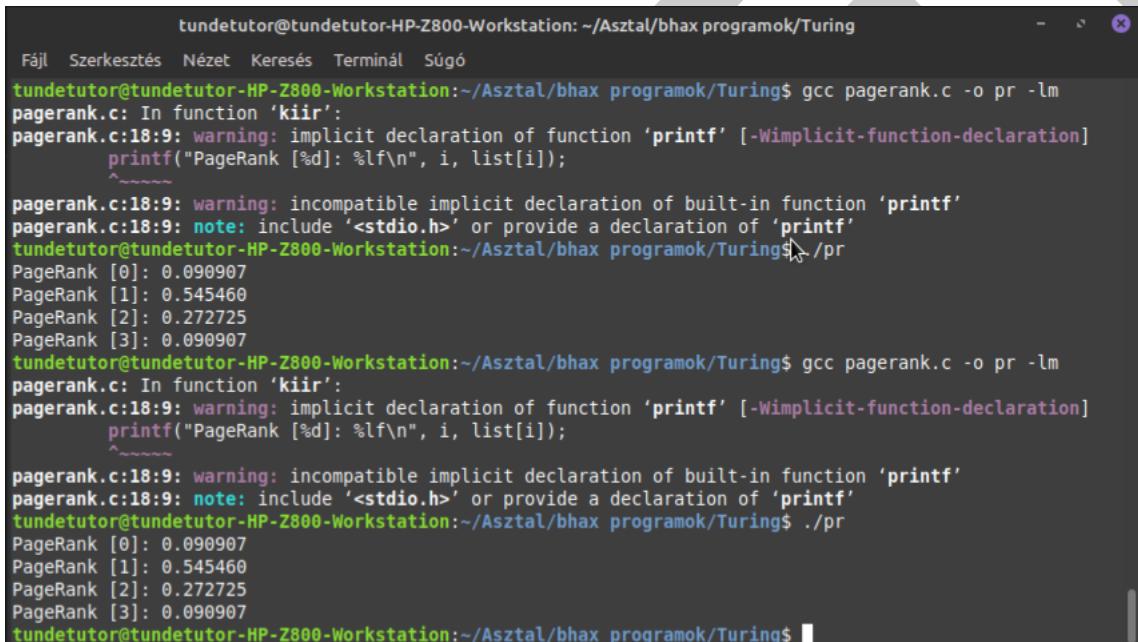
Itt láthatjuk a **distance** függvény működését. A végén gyökön kell vonnunk az összegből, nálam azért van ilyen furcsán megoldva mert elsőre nem jöttem rá hogy fordításhoz a **sqrt** függvny miatt kellene a -lm kapcsoló, aztán gondoltam 1/2-edik hatványra emelek így ez a kicsit matekos bonyolultabb megoldás maradt benne.

```
#include <stdlib.h>  
#include <math.h>  
  
double distance (double PR[], double PRv[], int db)  
{  
    float pwr = 0.5;  
    double osszeg = 0.0;  
    for(int i = 0; i < db; i++)
```

```
{  
    osszeg += (PRv[i] - PR[i]) * ( PRv[i] - PR[i] );  
}  
return pow(osszeg, pwr); //sqrt (osszeg);  
}
```

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

Lefutattva a program a következő outputokat adja:



```
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ gcc pagerank.c -o pr -lm  
pagerank.c: In function 'kiir':  
pagerank.c:18:9: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]  
    printf("PageRank [%d]: %lf\n", i, list[i]);  
          ^~~~~~  
pagerank.c:18:9: warning: incompatible implicit declaration of built-in function 'printf'  
pagerank.c:18:9: note: include '<stdio.h>' or provide a declaration of 'printf'  
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ ./pr  
PageRank [0]: 0.090907  
PageRank [1]: 0.545460  
PageRank [2]: 0.272725  
PageRank [3]: 0.090907  
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ gcc pagerank.c -o pr -lm  
pagerank.c: In function 'kiir':  
pagerank.c:18:9: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]  
    printf("PageRank [%d]: %lf\n", i, list[i]);  
          ^~~~~~  
pagerank.c:18:9: warning: incompatible implicit declaration of built-in function 'printf'  
pagerank.c:18:9: note: include '<stdio.h>' or provide a declaration of 'printf'  
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$ ./pr  
PageRank [0]: 0.090907  
PageRank [1]: 0.545460  
PageRank [2]: 0.272725  
PageRank [3]: 0.090907  
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/bhax programok/Turing$
```

2.7. Monty Hall probléma

A Monty Hall probléma egy amerikai TV műsorból ered, az alap szituáció az, hogy adott 3 db ajtó, melyek közül az egyik mögött nyereményt találunk. Választanunk kell ezek közül egyet, ez lesz a tippünk, 1/3, azaz 33,33% esélyünk van a megfelő ajtót kiválasztani igaz? Monty Hall, a műsorvezető viszont picit megkavarja a dolgokat. Felfedi mi van az egyik -szükségszerűen nem nyertes és nem általunk választott-ajtó mögött, majd ezután megkérdezi a játékost, hogy szeretné-e megváltoztatni a döntését. Itt van a csavar, a műsor nézői arra lettek figyelmesek, hogy a döntésüköt megváltoztató játékosok 2/3 része nyert. Ez akkor azt jelenti, hogy 1 db ajtóra, a 3 közül, 2/3 nyerési esély jut? Igen! Pontosan emiatt nevezhetjük a Monty Hall problémát matematikai paradoxonnak.

A feladat leírásában megtalálhatjuk Bátfai Norbert R nyelvű szimulációs programját. Ebben a programban hasonlóan az én 2. programomhoz, a gép sorsolja ki az eseteket, a nyertes és a választott ajtókat. Egyébként

mint aogy láthatjuk, alapból 1000000 esettel dolgozunk, de kisebb nagyobb esetszámokat is vizsgálhatunk a kódot átírva.

Mielőtt nekilátunk R programokat futtatni, ha még nem volt, akkor most telepítsük a szükséges dolgokat! A paracsok amiket használnunk kell:

```
sudo apt-get install r-base
```

Futtatni pedig így tudjuk az R nyelvű programokat:

```
Rscript filenec.r
```

A fő különbség az én kódомmal szemben, hogy az R nyelvű program előre generálja le az összes esetet, és ezeket vektorban tárolja majd később kiértékeli, míg az én programom legenerál egy esetet és egyből ki és értékeli.

```
kiserletek_szama=1000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

  if(kiserlet[i]==jatekos[i]){

    mibol=setdiff(c(1,2,3), kiserlet[i])

  }else{

    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))

  }

  musorvezeto[i] = mibol[sample(1:length(mibol),1)]
}

}
```

Egy másik különbség, hogy ebben a programban kiszámoljuk hány esetben nyer a játékos ha változtat **length(valtoztatesnyer)** és ha nem változat a játékos **length(nemvaltoztatesnyer)**, míg az én programom csak azt mondja meg, hogy ha változtatunk a döntésen minden esetben akkor hány alkalommal lehet nyerni **X** esetből.

```
nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

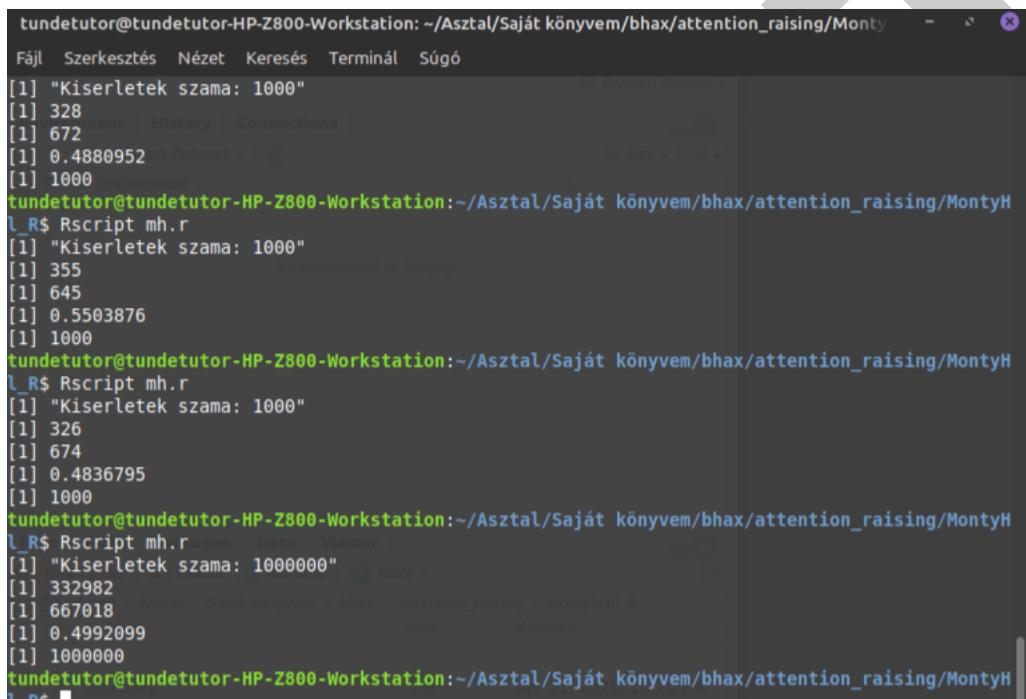
  holvált = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvált[sample(1:length(holvált),1)]
}

}
```

```
valtoztatesnyer = which(kiserlet==valtoztat)

sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Mindkét program esetében megjelenik ez a 2/3 nyerési arány, tehát szerintem mind a két program pontosan dolgozik. Lássuk egy printsceent!



```
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/attention_raising/MontyH
Fájl Szerkesztés Nézet Keresés Terminál Súgó
[1] "Kiserletek szama: 1000"
[1] 328
[1] 672
[1] 0.4880952
[1] 1000
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/attention_raising/MontyH
[R]$ Rscript mh.r
[1] "Kiserletek szama: 1000"
[1] 355
[1] 645
[1] 0.5503876
[1] 1000
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/attention_raising/MontyH
[R]$ Rscript mh.r
[1] "Kiserletek szama: 1000"
[1] 326
[1] 674
[1] 0.4836795
[1] 1000
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/attention_raising/MontyH
[R]$ Rscript mh.r
[1] "Kiserletek szama: 1000000"
[1] 332982
[1] 667018
[1] 0.4992099
[1] 1000000
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/attention_raising/MontyH
[R]$
```

Mivel nekem ez a feladat különösen tetszett, ezért C nyelven is megvalósítottam, mégpedig 2 változatban. Azzal kezdem amelyikből kiindultam. Ebben a programban anyereményjátékot le lehet játszani, a felhasználó választhat ajtót, és "Monty Hall" pedig felfed egyet majd a felhasználó dönthet arról, hogy másikat választ-e. Nézzük meg a kódomat!

Mindössze egy main függvényből áll a kód, viszont így is elég hosszú, kb 280 sorról beszélünk, vázolom, hogy miért. Először is deklaráltam, a váltózóimat, kisorsolunk egy random számot, 0, 1 vagy 2 lehet az értéke, ez alapján van kisorsolva, hogy melyik a nyertes ajtó. A **valasztott** változóban pedig tároljuk melyik ajtót választja a felhasználónk. A továbbiakban a döntése alapján ágazik el a programunk először 3 fő ágra, ezek közül én az első esetet mutatom be, a többi is nagyon hasonlóan működik.

```
int main()
{
    int ajtol;
    int ajto2;
    int ajto3;
    int valasztott;
    int r;
    int valasz;
    int nyertesjatekok = 0;
```

```
srand(time(NULL));
r = rand() % 3;
//r = 0;

printf("Valassz egy ajtot!\n");
scanf("%d", &valasztott);
printf("\nMost felfedem az egyik ajtot: \n");
```

Tehát, az I. számú eset, amikor a random generálás során az 1-es ajtó lett a nyertes, ezt az első fő estet további 3 esetre kell bontanunk, a felhasználó ajtóbálasztásától függően. Daraboljuk a kódot ily módon!

A felhasználó az első nyertes ajtót választotta, felfedjük a 3. számú ajtót, amit ugye nem választott és nem is nyertes, majd rábízzuk, a döntést, hogy újat választ-e. ekkor még 2 esetre bomlik ez az eset is. Amikor eleve a nyertes ajtót választja a játékos az az eset amikor ha újat választ akkor veszít, az összes eset 1/3 része ugye.

```
if( r == 0) //I. eset, az első ajtó a nyertes
{
    ajto1 = 1;
    ajto2 = 0;
    ajto3 = 0;

    if( valasztott == 1) //I./1 eset, elve a nyertes ajtót ←
        valasztottuk
    {
        printf("A 3-mas szamu ajto nem nyertes.\nSzeretned a dontesed ←
            megvaltoztatni? (1 <-- Y/ 0 <-- N)\n");
        scanf("%d", &valasz);

        if(valasz == 1) //ekkor az eleve nyertesként választott ajtó ←
            helyett egy vesztes ajtót választunk
        {
            valasztott = 2;
            printf("A valasztott ajtoban egy bena Porsce van. Porbald ←
                ujra!");
        }

        else if (valasz == 0) //maradunk a nyertes választásnál
        {
            printf("Nyertél! A nyeremenyed egy kecske!\n");
            nyertesjatekok = nyertesjatekok + 1;
        }
    }
    else
    {
        printf("Invalid bemenet");
    }
}
```

A második és 3 esetben nem nyertes ajtót választott a játékosunk, ha ilyenkor felfedjük a másik nem nyertes

ajtót akkor ha megváltoztatja a tippjét mindenképp nyerni fog, ez történik az **else if** és az **else** ágban is. Ugye így már látszik ez a 2/3-os arány? A II. és III. számú fő esetben is ugyanilyen szituáció alakul ki, tehát a nyerési esélyek csak ezzel a példával szemléltethetőek.

```
else if(valasztott == 2) //I./2 eset, elve a vesztes ajtót ←
    választottuk, az 1-es a nyertes
{
    printf("A 3-mas szamu ajto nem nyertes.\nSzeretned a dontesed ←
        megvaltoztatni? (1 <-- Y/ 0 <-- N)\n"); //felfedem a 3mas ←
        ajtót
    scanf("%d", &valasz);

    if(valasz == 1) //ekkor a korábban választott vesztes ajtó ←
        helyett egy nyertes ajtót választunk
    {
        valasztott = 1;
        printf("Nyertél! A nyeremenyed egy kecske!\n");
        nyertesjatekok = nyertesjatekok + 1;
    }

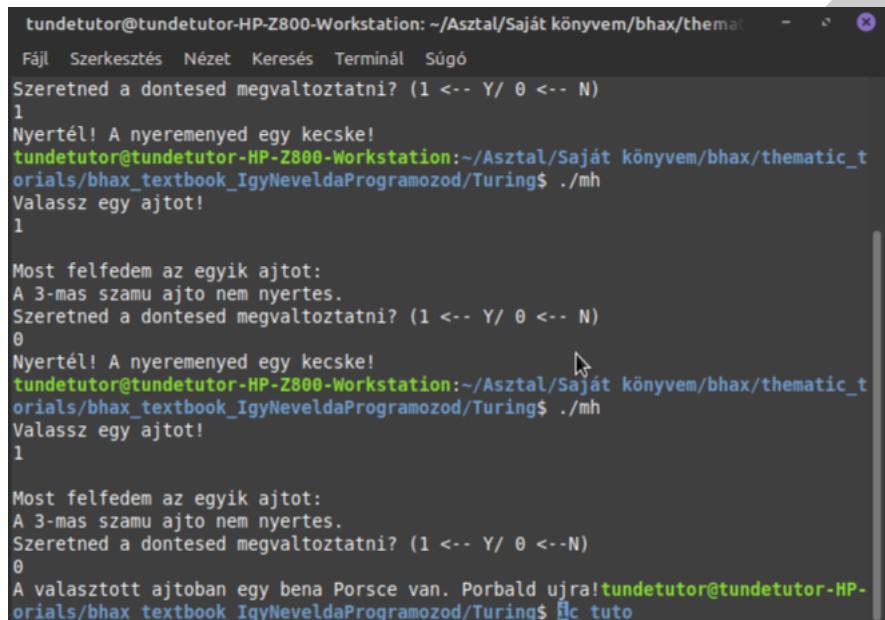
    else if (valasz == 0) //maradunk a nyertes választásnál
    {
        printf("A valasztott ajtoban egy bena Porsce van. Porbald ←
            ujra!");
    }
    else
    {
        printf("Invalid bemenet");
    }
}
else //I.3 vesztes ajót választottunk, a 3-masat
{
    printf("A 2-es szamu ajto nem nyertes.\nSzeretned a dontesed ←
        megvaltoztatni? (1 <-- Y/ 0 <-- N)\n");
    scanf("%d", &valasz);

    if(valasz == 1) //ekkor a korábban választott vesztes ajtó ←
        helyett egy nyertes ajtót választunk
    {
        valasztott = 1;
        printf("Nyertél! A nyeremenyed egy kecske!\n");
        nyertesjatekok = nyertesjatekok + 1;
    }

    else if (valasz == 0) //maradunk a nyertes választásnál
    {
        printf("A valasztott ajtoban egy bena Porsce van. Porbald ←
            ujra!");
    }
    else
```

```
{  
    printf("Invalid bemenet");  
}  
}  
}
```

Így néz ki a program ha futtatjuk, ez egy kis terminálos játék.



A screenshot of a terminal window titled "tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_themes/bhax_textbook_IgyNeveldaProgramozod/Turing\$./mh". The window shows a game loop where the user is prompted to guess a number between 1 and N. The user inputs 1, 0, and 1 respectively, each time winning a prize ("A nyereményed egy kecske!"). The user then inputs 0, which is incorrect ("A 3-mas szamu ajto nem nyertes."). Finally, the user inputs 1 again, which is also incorrect ("A valasztott ajtoban egy bena Porsce van. Porbald ujra!").

```
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_themes/bhax_textbook_IgyNeveldaProgramozod/Turing$ ./mh  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
Szeretned a dontesed megvaltoztatni? (1 <-- Y/ 0 <-- N)  
1  
Nyertél! A nyereményed egy kecske!  
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_themes/bhax_textbook_IgyNeveldaProgramozod/Turing$ ./mh  
Valassz egy ajtot!  
1  
  
Most felfedem az egyik ajtot:  
A 3-mas szamu ajto nem nyertes.  
Szeretned a dontesed megvaltoztatni? (1 <-- Y/ 0 <-- N)  
0  
Nyertél! A nyereményed egy kecske!  
tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_themes/bhax_textbook_IgyNeveldaProgramozod/Turing$ ./mh  
Valassz egy ajtot!  
1  
  
Most felfedem az egyik ajtot:  
A 3-mas szamu ajto nem nyertes.  
Szeretned a dontesed megvaltoztatni? (1 <-- Y/ 0 <-- N)  
0  
A valasztott ajtoban egy bena Porsce van. Porbald ujra!tundetutor@tundetutor-HP-Z800-Workstation:~/Asztal/Saját könyvem/bhax/thematic_themes/bhax_textbook_IgyNeveldaProgramozod/Turing$ ./mh_tuto
```

Viszont írtam egy olyan programot is ami úgy dolgozza fel az eseteket, mintha minden játék során a játékos új ajtót választana, az előző program tapasztalati és az R szimulációs program adta az ötletet ennek megvalósításához. Tehát elég minden ágban azt vizsgálni, hogy mi történik a döntés megváltoztatásakor. Nézzük a **main()** függvényt! Itt bekérünk egy vizsgálni kívánt esetszámot, és létrehozzuk a **nyertes** változót. A for cikluson belül meghívjuk a **mh(r, tipp)** függvényt, a kisorsolt nyerő ajtóval és a szintén random sorsolt választott ajtóval, ha a függvény visszatérési értéke 1, 1-gel növeljük a **nyertes** változót.

```
int main()  
{  
    int esetszam = 0;  
    int nyertes = 0;  
    int i;  
    int r;  
    int tipp;  
    srand(time(NULL));  
  
    printf("A program azt szimulalja, hogy minden esetben megvaltoztatjuk ←  
          az eredeti dontesunket.\n");  
    printf("Ird be az altalad vizsgálni kivánt esetszámot!\n");  
    scanf("%d", & esetszam);  
    printf("Esetszam: %d\n", esetszam);  
  
    for(i = 0; i < esetszam; i++)  
    {  
        //kisorsolom az ajtót
```

```
r = rand() % 3;
//printf("r: %d\n", r);

//srand(time(NULL)); //kisorsolom a játékos tippjét
tipp = rand() % 3;
//printf("tipp: %d\n", tipp);

if( mh(r, tipp) == 1 )
{
    nyertes = nyertes + 1;
}
printf("Nyertesjatekok szama: %d", nyertes);
}
```

Itt pedig láthatjuk a **mh()** függvényt, az I., II. és III. fő esetekkel illetve annak aleseteivel dolgozik, tehát azzal, hogy melyik a nyertes ajtó és melyiket "választotta" a játékos. Ha az I/1.-es esetet vizsgáljuk, az 1. a nyertes ajtó és azt választotta a játékos, ha ekkor megváltoztatja a döntését -márpedig megváltoztatja- akkor nem nyert tehát a függvény 0 értékkel tér vissza.

```
int mh(int ajto, int valasztott)
{
    if( ajto == 0) //I. eset, az első ajtó a nyertes
    {
        if( valasztott == 0) //I./1 eset, elve a nyertes ajtót ←
            valasztottuk
        {
            return 0;
        }
        else if(valasztott == 1) //I./2 eset, elve a vesztes ajtót ←
            valasztottuk, az 1-es a nyertes
        {
            return 1;
        }
        else //I.3 vesztes ajót választottunk, a 3-masat
        {
            return 1;
        }
    }

    else if( ajto == 1)
    {
        if( valasztott == 0) //I./1 eset, elve vesztes ajtót valasztottuk
        {
            return 1;
        }
        else if(valasztott == 1) //I./2 eset, eleve a nyertes ajtót ←
            valasztottuk, az 1-es a nyertes
    }
}
```

```
{  
    return 0;  
}  
else //I.3 vesztes ajót választottunk, a 3-masat  
{  
    return 1;  
}  
}  
  
else  
{  
    if( valasztott == 0) //I./1 eset, elve a vesztes ajtót ←  
        választottuk  
{  
        return 1;  
}  
    else if(valasztott == 1) //I./2 eset, elve a vesztes ajtót ←  
        választottuk, az 1-es a nyertes  
{  
        return 1;  
}  
    else //I.3 nyertes ajót választottunk, a 3-masat  
{  
    return 0;  
}  
}  
}
```

Az mh3.c nevű programom pedig így néz ki lefuttatva.

```
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ gcc mh3.c -o mh3  
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ ./mh3  
Esetszam: 10000000  
Nyertesjatekok szama: 6641tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ gcc mh3.c -o mh3  
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ ./mh3  
Esetszam: 10000000  
Nyertesjatekok szama: 666950tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ gcc mh3.c -o mh3  
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ ./mh3  
Esetszam: 10000000  
Nyertesjatekok szama: 6667215tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ gcc mh3.c -o mh3  
tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ ./mh3  
Esetszam: 10000000  
Nyertesjatekok szama: 6666491tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ ./mh3  
Esetszam: 10000000  
Nyertesjatekok szama: 6666952tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ ./mh3  
Esetszam: 10000000  
Nyertesjatekok szama: 6666774tundetutor@tundetutor-HP-Z800-Workstation: ~/Asztal/Saját könyvem/bhax/thematic_textbook_IgyNeveldaProgramozod/Turing$ ./mh3  
Esetszam: 10000000
```

2.8. 100 éves a Brun Tétel

1919-ben Viggo Brun felfedezett egy speciális számot, egy konstanst, amit Brun-konstansnak emlegetünk. Mi teszi ezt a számot olyan különlegessé? Ahhoz, hogy ezt megértsük az alapjainál próbálon megragadni a témát.

A csak 1-gyel és önmagukkal osztható prímszámokról szerintem a könyv olvasója már biztosan hallott, általános iskolában is tanuljuk matematika órán, azt is tudjuk már, hogy a prímszámok halmazának számossága végtelen, viszont kevsebbet hallottak az ikerprímekről, vagy a négyes prímekről, de most az ikerprímeket vizsgáljuk meg átfogóbban. Az ikerprímek halmaza a prímszámok halmazának egy részhalmaza, olyan speciális számpárosokat tartalmaz, melyekben minden két szám prím és különbségük pontosan 2. Ilyen ikerprímek pl.: 3 és 5, 5 és 7 vagy a 17 és 19. Ezekkel kapcsolatban még nincs bizonyítva, hogy számuk véges vagy végtelen lenne, még ma is a matematika egyik nagy kérdése. Viszont ezekkel a különleges számokkal kapcsolatban amit már biztosan tudunk, hogy közük van a Brun-konstanshoz.

A Brun-konstans azért érdekes az ikerprímekkel kapcsolatban mivel ha sorra vesszük az ikerprímeink reciprokösszegeit észre kell vennünk, hogy egy számhoz közelítenek az értékek, a Brun konstanshoz, azaz ahhoz konvergálnak a reciprok összegek.

A feladathoz kaptunk egy R nyelven írt kódot, ez egy Bátfai Norbert által írt kód. A program egy grafikonon kirajzolja nekünk, hogy az ikerprímek reciprok összeg értéke, hogyan konvergál a Brun-Konstanshoz, ami megközelítőleg 1.90216, tehát valahol a 2 körül találunk meg a számegyenesen.

Mielőtt elemezzük és futattuk a kódot megjegyezzm, hogy bizonyos könyvtárak szükségesek a futtatásához. Én telepítettem már korábban egy R Studio-t ami felajánlotta a hiányzó könyvtárak telepítését, így nem kellett manuálisan megcsinálnom.

Itt látható a rövidke kód ami számol nekünk. Ha megfigyeljük, a program a **matlab** könyvtár hívásával kezdődik, erről írtam, hogy ennek a telepítése szükséges lesz a futtatáshoz. Ezt követően létrehozzuk az **stp** függvényt ami a számításokat fogja végezni.

Az **stp** függvényben először megkeressük a prímszámokat, majd a szomszédos prímek közti különbséget vizsgáljuk. Látható hogy az **idx** nevű változóba gyűjtjük azoknak a prímeknek az indexét, melyek között a különbség 2 volt, tehát ikerprímek. A megtalált ikerprímeknek ezután a reciprokát képezzük majd ezeket a számokat összeadjuk. Az **stp** függvény ezután az ikerprímek reciprokok összegének szummájával tér vissza. Ezzel megvizsgáltuk a program legfontosabb részét, már csak a grafikon kirajzoltatása van hátra.

```
library(matlab)

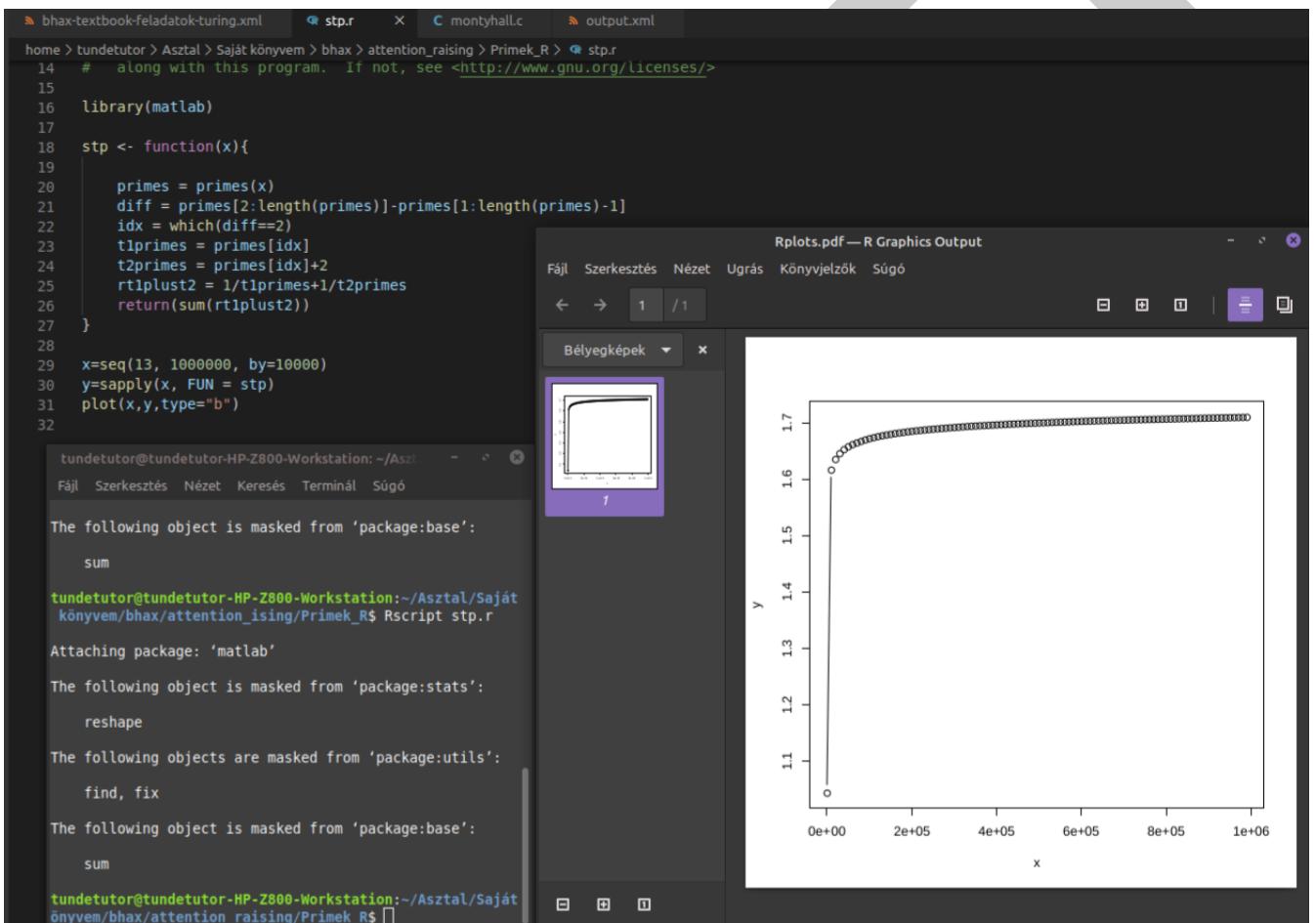
stp <- function(x) {

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

A kirajzoltatáshoz használunk egy **x**-et és egy **y**-t. Az **x**-ben egy sorozatot veszünk 13-tól egészen 1000000-ig, majd az **y** pedig megkapja az **stp** visszatérési értékét, ezzel megalkotva a koordináta rendszert. Aki használt már korábban Matlabot annak a **plot** ismerős lehet, matlabban ezzel a függvénytel tudjuk ábrázolni azokat amiket szeretnénk, és mivel meghívtuk a megfelelő könyvtárat, most is ezt használhatjuk.

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

A futtatás után egy pdf-et kapunk amely ábrázolja hogy az ikerprímek reciprokösszegei hogyan konvergálnak a Brun-konstanshoz.



2.9. MALMÖ | folytonos csiga feladat

Ehhez a feladathoz 2 megoldásom társul. az egyik Python nyelven a másik pedig C++-ban. Viszont mind a kettő folytonos mozgási parancsokkal dolgozik.

Nézzük először a Python kódöt. Itt az történik, hogy Steve elkezd csiga vonalban egyre fentebb és fentebb menni az arénában, úgy hogy tőle a fal mindig balra van, tehát a sarkokban jobbra kell fordulnia. Én ebben a megoldásban 2 egymásba ágyazott for ciklust használtam, a külső 128-szor tudna maximum végrehajtóni,

ez cvsak azért maradt annyi mert egy nagy számot akartam írni, hogy ne hagyja abba a mozgását véletlenül sem. Ez a for ciklus azért felelős, hogy a szinteken fentebb tudjon menni.

A belső for ciklus minden alkalommal fut le, azért mert az arénának 4 oldala van és azt szeretnénk, hogy Steve minden szinten körbeérjen. Ebben a ciklusban minden **.87 * i** ideig végzi a mozgást, vagyis az előre haladását. Ez ennél a programnál egyfajta "varázsszám" volt, hogy a folytonos mozgási parancsokkal nagyjából megfelő hosszakat mozogjon szintenként. Ezután jobbra fordul egyet. Ha a belső for ciklus 4 alkalommal már lefutott, azaz Steve egy szintet bezárta, akkor ugrik egyet, hogy a következő szinten folytassa a mozgását.

```
def run(self):
    world_state = self.agent_host.getWorldState()
    i = 1
    j = 1
    # Loop until mission ends:
    while world_state.is_mission_running:
        print("--- nb4tf4i arena ----- \n ←")
        for i in range(128):
            for j in range(4):
                self.agent_host.sendCommand( "move 1" )
                time.sleep(.87 * i)
                self.agent_host.sendCommand( "turn 1" )
                time.sleep(.5)
                self.agent_host.sendCommand( "turn 0" )
                time.sleep(.5)
                self.agent_host.sendCommand( "jump 1" )
                time.sleep(.5)
                self.agent_host.sendCommand( "jump 0" )
                time.sleep(.5)

    world_state = self.agent_host.getWorldState()
```

Megoldás videó: [Folytonos csiga python-ban](#)

A videón látszik, hogy ugyan valóban csiga vonalban megy felfelé az arénában Steve, elég szerencsétlenül mozog. Ez a folytonos mozgásnak tudható be, ugyanis így nem tud olyan precízen mozogni mint diszkrét parancsokkal, ilyenkor a "turn 1" sem azt jelenti, hogy 90 fokkal forduljon jobbra hanem inkább "fordul jobbra valamerre".

A C++ verzió egy sokkal egyszerűbb kód, itt Steve csak forog és nem is az arénában van szemmel láthatóan. Itt annyit mondtunk Steve-nek, hogy 0.5 másodpercig menjen előre majd ugyanennyi ideig forduljon jobbra. Csak ezt a **do-while** ciklust vágtam ide, ugyanis itt a Steve-nek a mozgási parancsokat. Szerintem python-ban érdemesebb próbálkozni a Malmövel, C++-ban több hibalehetőségünk van.

```
do {
    agent_host.sendCommand("move 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds(500));

    agent_host.sendCommand("turn 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds(500));
```

```
world_state = agent_host.getWorldState();
for( boost::shared_ptr<TimestampedString> error : world_state.Boost_
    errors )
    cout << "Error: " << error->text << endl;
} while (world_state.is_mission_running);
```

Megoldás videó: [Folytonos csiga C++-ban](#)

A videó 3 feladatmegoldást tartalmaz. ezek közül a kapcsolódó az első lesz. Megjegyzem, hogy a legoptimálisabb beállításokkal sem tudtam lerenderelni nem szemcsésre a videót, de szerintem a feladatok lényegi része így is látszik.

DRAFT

3. fejezet

Helló, Chomsky!

3.1. 3.1 Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Ebben a feladatban decimálisból unáris számrendszerbe váltó Turing gépről van szó, ahhoz hogy tudjuk mi ez tudnunk kell melyek a decimális illetve unáris számrendszerök. A decimális a jól ismert tízes számrendszer, melyben 10 számjegyből (0-9) tudjuk leképezni a számokat. Általános iskolából emlékezhetünk arra, hogy az ilyen tízes számrendszerbeli számokat oly módon írtuk fel hogy milyen számok vannak egyes helyiértékeken, pl.: egyes, tízes, százas, ezres. stb. Az adott helyiértéken álló számjegyet meg kell szoroznunk a helyiértékkel, majd ezekből összeget képzünk. Tehát pl.: $a = 321 = 3 \cdot 100 + 2 \cdot 10 + 1 \cdot 1$. Az egyes számrendszer olyan számrendszer melyben a számokat 1 számjeggyel az 1-essel tudjuk leírni mégpedig oly módon, hogy amennyi maga a szám egymás után annyi egyest irunk pl.: $3 = 111$. A gép a két számrendszer között úgy vált át hogy a decimális számból addig von ki 1-eseket míg az 0 nem lesz.

Ha ezt az átváltást program segítségével szeretnénk lemodellezni megoldás lehet ha mondjuk egy while ciklust használnunk amelynek kilépései feltétele hogy a decimális számunk 0 legyen. A ciklus törzsében minden lefutáskor levonunk a decimális számból egyet és egy 1-es számjegyet hozzáfüzünk pl egy (a program idnításakor még üres) txt állományhoz. Ekkor a ciklus végére a txt állományban keletkezett adat az eredeti decimális számunk unáris alakjának felel meg

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

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

Megoldás videó:

A generatív grammatika olyan nyeltani szabályrendszert jelent melyben véges számú szóból és véges számú szabályból végtelen számú mondatot lehet alkotni.

Sok olyan jelenség van amit nem lehet környezetfüggetlen nyelveken leírni, ilyen pl.: a természetes nyelvek, a primszámok halmaza. A környezetfüggő nyelvek zártak a halmazműveletekre az uniós műveletet leszámítva. S ($S \rightarrow aXbc$) $aXbc$ ($Xb \rightarrow bX$) $abXc$ ($Xc \rightarrow Ybcc$) $abYbcc$ ($bY \rightarrow Yb$) $aYbbcc$ ($aY \rightarrow aaX$)

aaXbcc (Xb → bX) aabXbcc (Xb → bX) aabbXcc (Xc → Ybcc) aabbYbcc (bY → Yb) aabYbbccc (bY → Yb) aaYbbbcc (aY → aa) aaabbbccc

A (A → aAB) aAB (A → aAB) aaABB (A → aAB) aaaABBB (A → aC) aaaaCBBB (CB → bCc) aaa-abCcBB (cB → Bc) aaaabCBcB (cB → Bc) aaaabCBBc (CB → bCc) aaaabbCcBc (cB → Bc) aaaabbCBcc (CB → bCc) aaaabbbCccc (C → bc) aaaabbbcccc

3.3. 3.3 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.

A BNF környezetfüggetlen szintaxisokat leíró szintaxis.

A C nyelvi utasítás fogalmának szintaktikai definíciója BNF szintaxisban:

```
<utasítás> ::=  
    <összetett_utasítás>  
    <kifejezés>; (értékkedás pl, num=10)  
    if(<kifejezés>) <utasítás>  
    else if(<kifejezés>) <utasítás>  
    else <utasítás>  
    switch (<kifejezés>)  
        <egész_konstans_kifejezés : <utasítás>  
        goto <azonosító>;  
        <azonosító> : <utasítás>  
        break; continue; return<kifejezés>;  
        or(<kifejezés1><kifejezés2><kifejezés3>) <utasítás>  
        while(<kifejezés>) <utasítás>  
        do <utasítás> while<kifejezés>  
    ; (üres utasítás, pl FORTRAN continue-ja)
```

Példa olyan programra ami a C89-es szabvánnyal nem fordul de C99-cel igen. Ebben a programban cikluson belül deklárálok változót ami a C99-es szabványtól lehetséges. Ha C89-cel próbáljuk meg fordítani akkor a fordító javasolja is a C99 hozzávalatát.

```
int main()  
{  
    for(int i = 0; i < 20; i++)  
    {  
        printf("Cikluson belüli deklarálás.");  
    }  
}
```

3.4. 3.4 Saját lexikális elemző

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

A lexer programok szöveg elemző programok amivel szövegeket lehet tetszőleges szempontból elemezni illetve átalakítani. A példaprogram a beirt szövegen valós számokat keres. A program 3 fő részre bontható (%% az elválasztójel), az első részben meghívjuk a szükséges header fájlokat és deklarálhatunk változókat. A második részben definiáljuk hogy mit is keresünk a szövegen, illetve mit csinálunk vele, azaz szabályokat alkotunk. Ebben a programban valós számokat keresünk, tehát olyan számokat amelyek tetszőleges számú számjegyből állhatnak vagy tetszőleges számú számjegyből és 1 db pontból majd újabb tetszőleges számú számjegyből állnak. A megtalált valós számokat a program a már átalakított szövegen kiemeli. minden megtalált valós szám után növeli a realnumbers nevű változót is, tehát számolja is a megtalált számokat. A program utolsó része a main függvény ahol meghivatjuk a lexert és saját utasításokkal egészíthetjük ki a programot.

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.l

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

Tanulságok, tapasztalatok, magyarázat...

3.5. 3.5 Leetspeak

Lexelj össze egy l33t cipher!

Az előző programhoz hasonlóan ez is egy lexer program. Ez úgy működik, hogy az adott szövegen ki tudja cserélni a karaktereket egy az eredetihez hasonló karakterre (9-est egy g betüre). Ezt olyan módon teszi meg hogy az angol abc betüihez illetve 0-9-ig a számokhoz társít 4 kinézetre hasonló karaktert (ha nincs 4 hasonló akkor pl 2 hasonlót kétszer társítunk hozzá). A program a beolvasást kisbetüsíti tehát mindegy hogy a bemenetre kis vagy nagy betük érkeznek felismeri az adott karaktert. A kicsérélést random

módon teszi, 0 és 100 között sorsol egy random szémet, ha 91-nál kisebb akkor az első társított karakterre cseréli a megtaláltat, tehát erre nagyobb esély van mint a többire. A második karakterre 4% esély van, a harmadikra 3% és a negyedikre 2%.

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/1337d1c7.1

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

#define L337SIZE (sizeof 1337d1c7 / sizeof (struct cipher))

struct cipher {
    char c;
    char *leet[4];
} 1337d1c7 [] = {

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

```
{'4', {"h", "h", "A", "A"}},  
{'5', {"S", "S", "S", "S"}},  
{'6', {"b", "b", "G", "G"}},  
{'7', {"T", "T", "j", "j"}},  
{'8', {"X", "X", "X", "X"}},  
{'9', {"g", "g", "j", "j"}}

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

%}
%%

. {

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

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

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

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

            found = 1;
            break;
        }

    }

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

    }
%%

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

Tanulságok, tapasztalatok, magyarázat...

3.6. 3.6 A források olvasása

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?

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

2. for(i=0; i<5; ++i)

3. for(i=0; i<5; i++)

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

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

6. printf("%d %d", f(a, ++a), f(++a, a));

7. printf("%d %d", f(a), a);

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

1. Akkor és csak akkor kezelje a 'jelkezelo' függvény a SIGINT jelet, ha az nincs ignorálva.
2. Egy for ciklus ami a ötször hajtja végre a hozzá rendelt utasításokat. Preorder módon először az i-t növeli és csak aztán végzi el az utasításokat.
3. Egy for ciklus ami a ötször hajtja végre a hozzá rendelt utasításokat. Postorder módon először elvégzi az utasításokat és csak azután növeli az i-t.
4. Egy for ciklus ami a ötször hajtja végre a hozzá rendelt utasításokat. Viszont a tomb[] első öt értékét lecseréli az aktuális i értékre. Tehát 0,1,2,3,4.
5. WIP
6. A standard outputra kiiratjuk az f() függvény visszatérési értékét, decimális számban.
7. A standard outputra kiiratjuk az f() függvény visszatérési értékét 'a'-ra és magát az 'a'-t is, decimális számban.
8. A standard outputra kiiratjuk az f() függvény visszatérési értékét 'a'-ra (mivel annak a memória címére mutatunk) és magát az 'a'-t is, decimális számban

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. 3.7 Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

- A primszámok száma végtelen.
- Az ikerprimek száma végtelen.
- A primszámok száma véges.
- A primszámok száma véges. (itt egy az előzővel ekvivalens formula volt)

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

Tanulságok, tapasztalatok, magyarázat...

3.8. 3.8 Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

```
int main()
{
    //egész
    int num = 32;

    //egészre mutató mutató
    int* toNum = &num;

    //egészek tömbje
    int numArray[4] = {0,1,2,3};

    //egészek tömbjének referenciája
    int (&numArrayRef)[4] = numArray;

    //egészre mutató mutatók tömbje
    int* pointerArray[4];

    //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

    return 0;
}
```

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:

- a mint egész tipusú változó.
- a memória címére mutató mutató.
- Egy egészre mutató mutatót r névvel, ami az a értékét mint mutatócím tartalmazza.
- Öt elemű tömb, mely egészekből áll.
- Egy 5 elemű, egészeket tartalmazó tömbre mutató mutató, mely a c tömbre mutat.
- Öt elemű egészekre mutató mutatókból álló tömb.
- Egéssel visszatérő, paraméter nélküli függvényre mutató mutató.

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

```
#include <stdio.h>

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

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

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

```
        return sum;

}

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

    f = sum;

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

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

    g = sumormul;

    f = *g (42);

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

    return 0;
}
```

```
#include <stdio.h>

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

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

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

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

```
int
main ()
{
    F f = sum;

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

    G g = sumormul;

    f = *g (42);

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

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

3.9. 3.9 MALMÖ | Discrete csiga

A Turing feladatcsokorban hasonló feladattal találkozhattunk de ott diszkrét helyett folytonos mozgást alkalmaztunk. A két mozgás között jelentős különbségek vannak, míg folytonos mozgásnál a "move 1" parancs mint kapcsoló működik ami miatt addig mozog amíg a "time.sleep()" engedi, addig diszkrét mozgásnál a "move 1" utasításként működik tehát 1 blokknyit mozog előre a "time.sleep()"-ben megadott idő alatt. Azt hogy csigában haladjon mi úgy oldottuk meg, hogy először az legsó szinten lévő virágot kibányássza majd egy szinttel ugorjon fentebb és ezután kezdj el a csigát. A csigához 2 db for ciklust használtunk, a belső for ciklusban minden az oldalnyi mozgáshoz elegendő lépésmennyiséget teszi a külső for ciklus pedig azt teszi lehetővé hogy adott szinten az adott oldalhosszot 4x tegye meg, ennek elején minden fentebb ugrik egy szintet.

Megoldás videó: [vidi itt, katt](#)

4. fejezet

Helló, Caesar!

4.1. 4.1 Alsó háromszögmátrix

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

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

Először is mik azok a háromszögmátrixok? A háromszögmátrixok speciális kvadratikus mátrixok, általában 2 fajta háromszögmátrixot különböztetünk meg, van alsó pés felső háromszögmátrix. A felső háromszögmátrix jellemzője hogy a főátló alatt csupa 0 érték található, míg az alsó háromszögmátrix ennek fordítottja, annál a főátló fölött található csupa 0 érték.

A példaprogram egy 5x5-ös alsóháromszögmátrix-ot ábrázol. A 8. sorban tálunk egy if függvényt, ez egy hibakeresés és helyfoglalás egyben. A feltételében egy malloc-kal helyet foglalunk a mátrixunknak illetve megvizsgáljuk, hogy azok száma 0.e mert amennyiben a feltétel igaz nem beszélehtünk semmiképp háromszögmátrixról, és egy return -1, gibát kapunk vissza. Tehát ha pl a "nr" változó ami meghatározza hányszor hanyas lesz a mátrix, 0 akkor hibát ad vissza a program. Gyakori hogy a helyfoglalást egy if-en belül oldják meg, mint a hibakezelés miatt. Ezután egy preorder for cikluson belül helyet foglalunk a mátrixunk elemeinek ismét malloc-ot használva. A for cikluson belül pedig újabb hibakezeléssel találkozunk. A size ugye a mátrix beli elemek számát jelöli, ez sem lehet nulla, ha mégés az akkor return -1 és hibát ad vissza a program. Ha a program idáig helyesen lefutott és a helyfoglalással minden rendben volt, ezután 2 for cikluson belül feltölti a mátrixot. Az egyik for ciklus a mátrix sorain halad végig a másik pedig annak oszlopain. Alapból egy képlet segítségével az értékes elemek értéke az a szám ahányadik értékes elemről van szó. Itt megjegyezném hogy az első értékes elem egy 1 mivel a C sok más programnyelvhez hasonlóan 0-tól számol. Ezután ismét az előző 2 for ciklust felhasznáva a program kíratja azok elemeit.

A program egyébként 2 db mátrixot ír ki a képernyőnkre. A második mátrixban bizonyos elemeknek konkrét értéket adva (42, 43, 44, 45). Ezt az értékadást minden elem esetében más módon teszi meg ezzel példákat adva a mutatók helyes használatára, ugyanis egy mátrix beli elemre több módon is hivatkozhatunk. Az első és egyben legegyszerűbb ha konkrétan megmondjuk mely elemről van szó "tm[s][o]". Ha a mutatókat is ki szeretnénk használni akkor több megoldás is helyes. Lehet mutatni a mátrixon belül sorra vagy oszlopra és a másik értéket konkrétan megadni vagy lejóhet a sorokhoz és az oszlopokhoz is mutatót használni. Ezután ismét 2 for ciklus segítségével kíratjuk magát a mátrixot.

A program utolsófontos része pedig a mátrixnak és annak elemeinek lefoglalt hely felszabadítása, ezt a `free()` függnyénnel tehetjük meg.

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

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

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

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

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

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

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

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

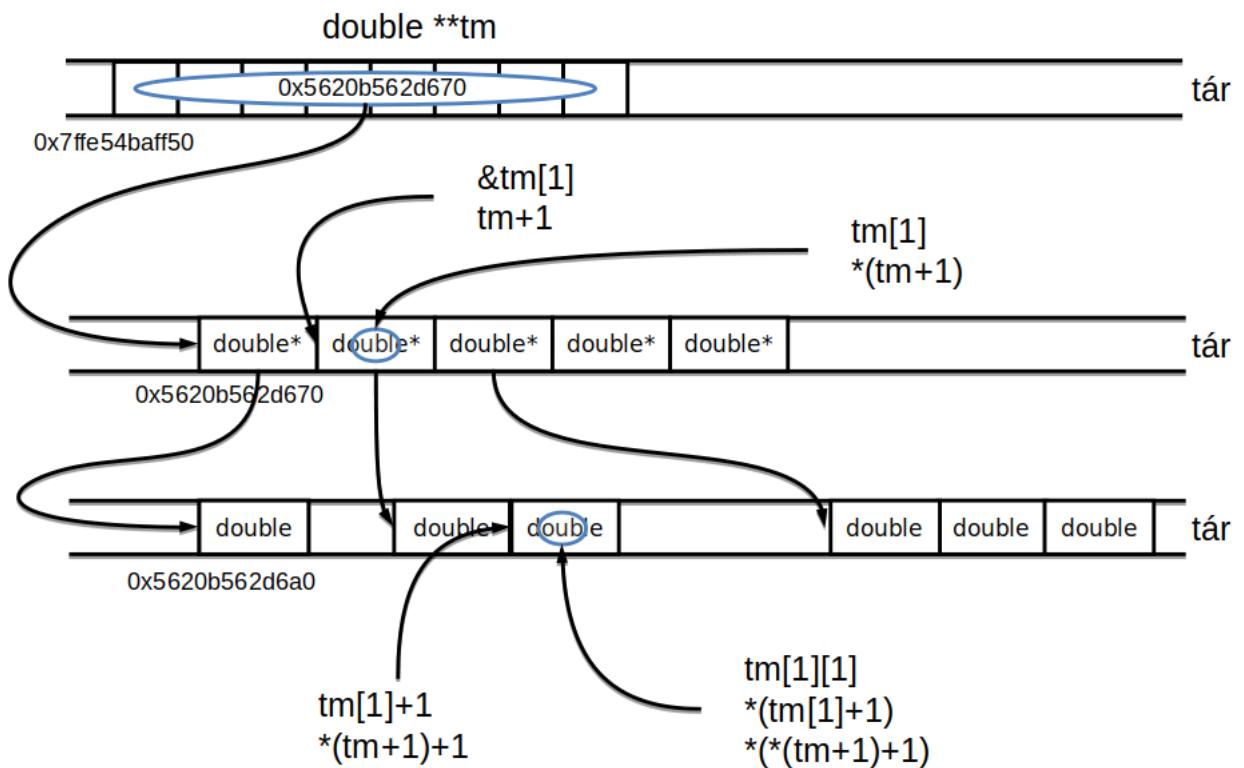
    for (int i = 0; i < nr; ++i)
        free (tm[i]);
```

```

    free (tm);

    return 0;
}

```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

4.2. 4.2 C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: egy részletes feldolgozása az [e.c és t.c forrásoknak](#).

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

Az EXOR titkosító lényege hogy megadunk egy valahánybetűs kulcsot és egy szöveget. ezeket összeXOR-ozzuk. Ekkor az eredet szövegből egy furcsa bitsorozatot kapunk, a szövegünk titkosítva is van. Ha ezt a titkosított szöveget ismételten összeExorozzuk a kulccsal, akkor visszakapjuk az eredeti szövegünket. Hogyan lehetséges ez? Az XOR, azaz a kizárá vagy logikai művelet alkalmas pl karakterek bitenkénti cseréjére is,

itt gyakorlatilag vaz történik, hogy a szöveg bitjeit "összemossa" a kulcséval és egy felismerhetetlen bitsorozatot kapunk. Nyílván amikor ezt visszafele csináljuk ugyanaz történik és megkapjuk a tiszta szövegünket. Erről a javanál írok részletesebben.

4.3. 4.3 Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

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

A java-ban megírt program rövid magyarázata:

Először is, a programnak szüksége van a kulcsra, amit egy stringben tárol el illetve kell egy 256 byte-os buffer a bemenetünknek. Ezek után egy while ciklus segítségével addig olvassa a bemenetet amíg az van, majd ezen belül egy for ciklust találunk. A for ciklus annyiszor hajtódik végre ahány beolvastott byte-ünk, azaz karakterünk van, ezena for cikluson belül pedig a bufferbe beolvastott szövegből byte-onként összeEXOR-ozza a bufferben lévő dolgokat a kulcs egyik elemével.

```
import java.io.InputStream;
import java.io.OutputStream;

public class main
{
    public static void encode (String key, InputStream in, OutputStream out ←
        ) throws java.io.IOException
    {
        byte[] kulcs = key.getBytes();
        byte[] buffer = new byte[256];
        int kulcsIndex = 0;
        int readBytes = 0;

        while((readBytes = in.read(buffer)) != -1)
        {
            for(int i=0; i<readBytes; i++)
            {
                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]); //itt ←
                csinálja az összeEXOR-ozást
                kulcsIndex = (kulcsIndex + 1) % key.length();
            }

            out.write(inputBuffer, 0, readBytes);
        }
    }

    public static main (String[] args)
    {
        if(args[0] != "")
```

```
{  
    try  
    {  
        encode(args[0], System.in, System.out); //  
    }  
    catch(java.io.IOException e)  
    {  
        e.printStackTrace();  
    }  
}  
else  
{  
    System.out.println("Please provide a key!");  
    System.out.println("java main <key>");  
}  
  
}  
}
```

4.4. 4.4 C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Adott volt az előző feladatban a t.c program ami arra való hogy ugye XOR titkosított szövegeket törjünk vele. Labor feladat volt ezt kivitelezni is, amit sikerült is megoldanom pár esetben. Nagyon felbosszantott hogy nem tudtam mindenki szövegét törni majd utána jártam a dolognak hogy mi volt a baj...

Előzmények: Az XOR törő program alapvetőleg a brute force technikát alkalmazza, azaz próbálunk ki minden lehetséges kulcs kombinációt. Nyílván ez nem működhet mindenféle jelszótörésre meg egyéb ha-xorkodáshoz mivel ez az egyik leggyakoribb törési mechanizmus és gyakran pl a nemes egyszerűséggel ettől le vannak véve úgy hogy nem próbálkozhatunk mondjuk 5-nél több alkalommal. Na de itt nem eza helyzet szóval törjük meg azt a titkosítást... Alapból a program 8 karakteres kulcsokat tud törni, akiknek a szövegét én kinéztem nekik zömében 4 karakterből álló kulcsuk volt. Oké. A következő lépés az volt hogy a segítségként megadott karaktereket egy tömbbe vettem hogy ne kelljen a teljes ABC-n végészaladnia a programnak, illetve a for ciklusokból is kiszedtem 4 db-ot. Leteszteltem a saját szövegemben, a program készen állt a törésre. Ekkor meglepetésemre sorra nem törte a mások által titkoított szövegeket. Kérdezősködtem és a csoporttársaim nagyrésze ugyanígy járt. De ami igazán érdekes volt, hogy az XOR törőink ugyanazokat a szövegeket törték és ugyanazokat nem. Végül utána jártunk és az volt a baj hogy a törő a magyar szövegeket tudja hatékonyan törni így meg kellett elégedni azzal hogy azt a pár szöveget helyesen és viszonylag gyorsan törte a program.

Itt pedig az az XOR törő van mellékelve ami 4 karakteres kulcsot tör, amely a g, s, e és p karakterekből áll:

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256
```

```
#define KULCS_MERET 4
#define _GNU_SOURCE

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

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

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

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

```
}

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

    return tiszta_lehet (titkos, titkos_meret);
}

int
main (void)
{
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char betuk[5] = {'g', 's', 'e', 'p', '\0'};
    char *p = titkos;
    int olvasott_bajtok;

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

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

        // osszes kulcs eloallitasa
        for (int ii = 0; ii <= 4; ++ii)
            for (int ji = 0; ji <= 4; ++ji)
                for (int ki = 0; ki <= 4; ++ki)
                    for (int pi = 0; pi <= 4; ++pi)
                    {
                        kulcs[0] = betuk[ii];
                        kulcs[1] = betuk[ji];
                        kulcs[2] = betuk[ki];
                        kulcs[3] = betuk[pi];

                        if (exor_tores (kulcs, KULCS_MERET, titkos, p - ←
                                      titkos))
                            printf
                                ("Kulcs: [%c%c%c%c]\nTiszta szoveg: [%s]\n",

```

```
        ii, ji, ki, pi, titkos);

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

return 0;
}
```

4.5. 4.5 Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

Egy nagyon jó szemléltető videó: <https://www.youtube.com/watch?v=qv6UVOQ0F44>

A neurális háló egy nagyon érdekes de nem könnyű téma. Itt az első bekezdésben röviden bemutatnám mi-ről is van szó. Egy neurális háló 2 főbb dologból áll: neuronokból és rétegekből. A neuronok lényege hogy a kapott adatokat továbbítani vagy módosítani legyenek képesek. A neuronoknak van egy bemeneti oldala, itt a bejövő adatokat szummázza illetve van egy kimeneti oldala. A valódi idegsjekben a "bemeneti" adatok axonokon keresztül érkeznek, az axonok által összekötött idegsejtek pedig a neuron hálózatot alakítják ki, ehhez hasonlóan működnek a programozásbeli neurális halózatok is, ezeket gráffal tudjuk szemléltetni. A neuronok tehát adatokat kapnak és azokat továbbítják. Ez a kapcsolat alakítja ki a rétegeket. 3 fő rétegből szoktak állni a neurális hálózatok, ezek a bemeneti, a rejttett és a kimeneti réteg. Rejtett rétegből lehet több, vagy akár egy sem, minél több rejttett rétegből áll egy hálózat annál komplexebb feladatokat képes végrehajtani. A rejttett réteget lehet műveleti rétegnek is nevezni, ahogy ezt sejteni lehet itt a kapott adatokkal a neuronok műveleteket végeznek, ezek leggyakrabban a legalapabb logikai műveletek, azaz OR, AND, XOR stb. Léteznek úgy nevezett deep learning neural network-ök, ezeknek a lényege az hogy a neurális hálózatunkhoz, egy rewar systemes. jutalmazási rendszert adunk, ezek a hálózatok sok lefutás, generációt keresztül tanulnak, a folyamatosan érkező adatokat feldolgozzák. Minél pontosabb reward systemet alakítunk ki annál hatékonyabb lesz a tanulási folyamata.

Na de térjünk is rá a neurális OR, AND és XOR kapukra. Ezekhez 2 db input szükséges, az egyik lgyen A a másik pedig B. Az input módosítás nélül adja tovább az adatokat. Az OR és az And esetén nincs rejttett rétegünk, ezek elemi logikai műveletek. Ahhoz hogy a neurális hálónkat ezek elvégzésére megtanítsuk eggyel data.frame táblára van szükségünk, illetve a neuralnet() függvényre. Ez azt jelenti, hogy a data-frame-ben megadjuk neki az inputokat és a művelet elvégzése ztáni eredményt, példát mutatunk be.

Az AND és az OR műveleteket, rejttett réteg bélkül is el tudja végezni, akár a kettőt is egyszerre, de az XOR, azaz a kizárt vagy esetén más a helyzet. Ha ítéletlogikai formulaként felírjuk láthatjuk hogy egy sokkal összetettebb formuláról van szó, még attól AND egy elemi konjunkciónak felel meg az OR pedig ugye elemi diszjunkcióknak. Az XOR-hoz 3 rejttett rétre lesz szükségünk.

```
library(neuralnet)
```

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

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

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

plot(nn.or)

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

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

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

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

plot(nn.inand)

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

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

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)

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

4.6. 4.6 Hiba-visszaterjesztéses perceptron

C++

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

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

Tanulságok, tapasztalatok, magyarázat...

Ez a program egy elég érdekes program. Én elsősorban a megoldás videóban található programról szeretnék írni. A program a mandelbrot halmazról kirazoltatott png képpel dolgozik. Ebben a png képben fekete és fehér pixelek vannak, itt fontos megjegyezni hogy a fehér pixelek tartalmaznak piros színt míg a feketék nem. A program elején meg van hívva az mlp.hpp. Az mlp kibontva MUlti Layer Perceptron, ez adja a programhoz a 3 rétegű neurális hálót, ettől lesz képes tanulni. A 9. sorban van deklaráció a kép mérete azaz size, ami a szélesség*magasság, azaz a tartalmazott pixelek száma. A program további részében a 13. sorig a png-nek való helyfoglalás történik, a képet mint egy mátrixot kell elképzelünk, ugye van szélessége (oszlopok) és magassága (sorok). Ezután létrehoz egy p, Perceptron típusú mutatót, ait az mlp beli osztály miatt lehet meg, illetve egy double típusút, ami az image, azaz maga a png kép mint tömb, elemeire, tehát a pixelekre mutat. Ezt követően a dupla forcilusban feltölti az image mátrixot a pixelekkel, hasonló módon mint a háromszögmátrix esetében és megvizsgálja a piros színű pixeleket. A program végén a foglalt helyeket szabaddá tesszük.

Ezt a programot még sokat kell tanulmányozno de nagyon rajta vagyok az ügyön, a neurális háló működéséről pedig az előző feladatban írtam tehát ne,m szeretném ismételni magam.

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

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

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

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

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

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

    double value = (*p)(image);

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

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

4.7. 4.7 Steve látása

Megoldás videó: <https://www.youtube.com/watch?v=DX8dI04rWtk>

Steve 2 módon láthat. Az egyik a lineOfSight, azaz azt a blokkot látja amin a kereszt van. A másik mód pedig egy körülötte lévő cuboidot használ. Ezt az xml módosításával kiterjeszhetjük az alap 3x3x3-masról akár 7x7x7-esre is. Itt fontos arra ügyelni hogy mely általa érzékelt blokkokkal szeretnénk dolgozni, ugyanis ezek egy tömbben vannak eltárolva és meg iOS vannak számozva. Ezeknek a cuboidoknak a közepén van mindenig Steve. Ezekre a tömbbeli elemekre tudunk hivatkozni tehát ki tudjuk íratni a tartalmukat, vagy egy if függvényel külön tudunk kiíratást csinálni hogy mondja el Steve hol és mit lát.

5. fejezet

Helló, Mandelbrot!

5.1. 5.1 A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

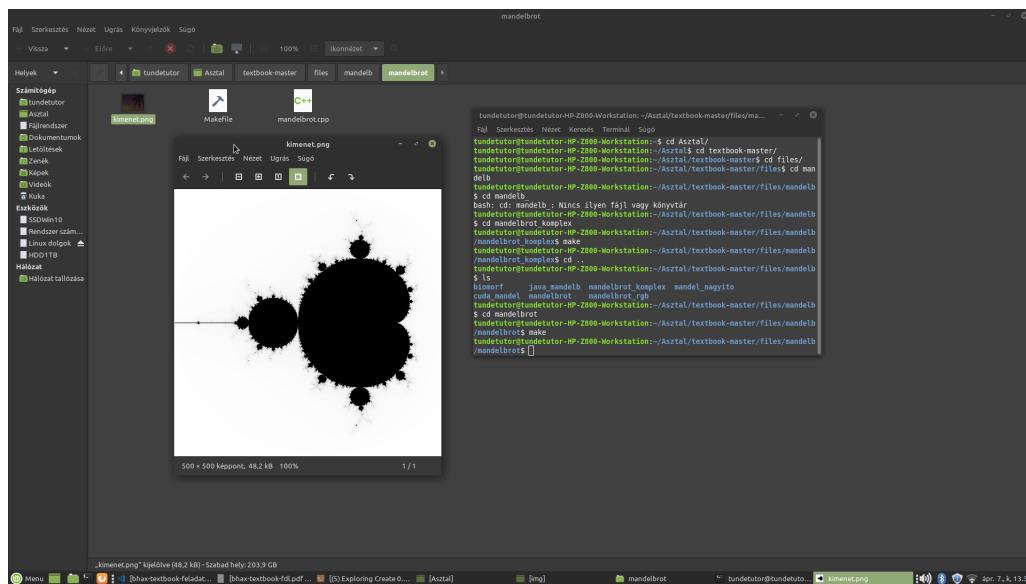
Megoldás videó: <https://youtu.be/gvaqijHIRUs>

Megoldás forrása: bhax/attention_raising/CUDA/mandelpngt.cpp nevű állománya.

The screenshot shows a Visual Studio Code interface. On the left, the sidebar displays extensions: MARKETPLACE, including `ext:cu`, `vscode-cudacpp 0.1.1`, `CUDA C++ language support for Visu...`, `kriegalex`, `vscode-clangd 0.0.21`, `Clang Language Server`, and `LLVM Extensions`. The main editor area contains a C++ file named `mandelbrot_komplex.cpp` with code for generating a Mandelbrot set image. A preview window titled "kimenet.png" shows the resulting fractal image, which is a black and white Mandelbrot set. The status bar at the bottom shows file tabs for `mandelbrot_komplex.c...` and `mandelbrot_komplex`, along with other system information.

5.1. ábra. A Mandelbrot halmaz a komplex síkon

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



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

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

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

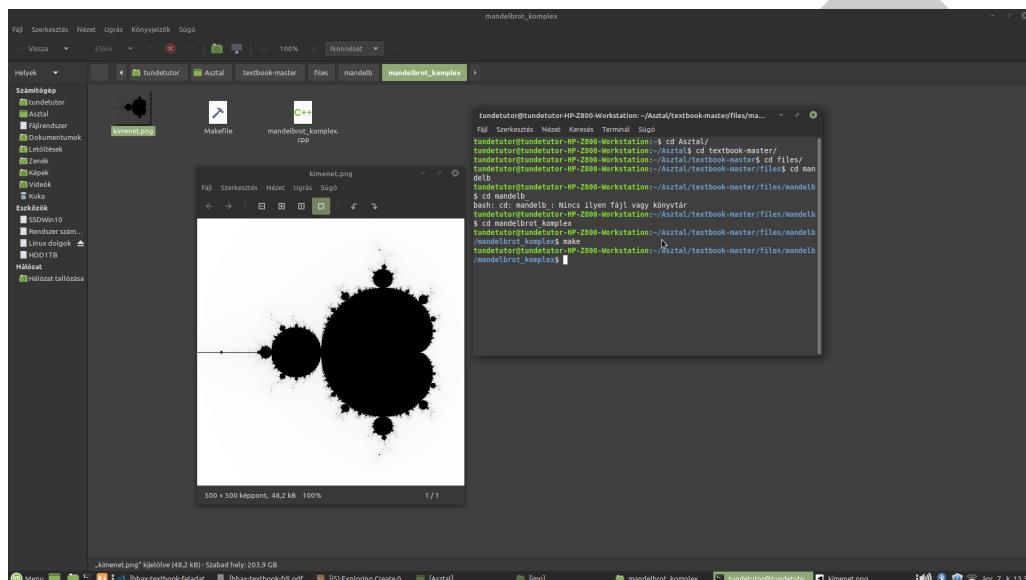
Ez végül is egy C alapú C++ program lett, mivel ahhoz hogy ténylegesen elkészüljön a png, ahhoz szükség van gy png könyvtárra viszont ez C-hez nem igazán található. A program úgy működik, hogy egy struktúrát használunk ahhoz, hogy komplex számokkal dolgozhassunk. A mandelbrot halmazt úgy tudjuk megjeleníteni, hogy a png-t mint egy mátrixot képzeljük el, és minden fog tartozni egy 2-256 közötti szám, ez határozza meg hogy az adott pixel milyen színű legyen. Ezt a számot úgy fogjuk megkapni hogy a cikluson belüli iteráció száma lesz az, azaz hogy az adott értékkel ki tudunk-e kerülni a ciklusból vagy sem. Ha nem sikerül, az iteráció száma 256 lesz, tehát fekete pixelt kapunk, ebben az esetben arról van szó, hogy az adott érték nem tudott kikerülni a while ciklusból, tehát nem tudta elhagyni magát a mandelbrot halmazt sem, azaz annak eleme. Ha ez a szám 0, akkor fehér pixelt kapunk. A png-ben megjelennek szürke pixelek is, ezek a 2 érték között valahol vannak. Összességében ezért van az, hogy a kapott png fekete és fehér

színekből áll. A png-t még lehetne cifrázni aszerint, hogy milyen "gyorsan" lépnek ki egy-egy értékek a while ciklusból, ilyen módon generálják a sok színes mandelbrot halmazt ábrázoló képet is.

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

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

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



Az előző feladathoz képest itt a legnagyobb különbség az az, hogy itt kihasználjuk a C++ előnyeit a C-vel szemben. Mint ahogy azt már említtettem az előző egy C alapú program volt, viszont ehhez már használtunk C++ könyvtákat ami nagyban segíti a programozók dolgát. Ilyen pl a komplex számokhoz tartozó könyvtár, itt már nem kellett struktúráz használnunk a komplex számok leírásához, a C++ ezt tudja magától is, illetve itt van külön függvénye a négyzetre emelésnek is. Ezeken kívül ugyanazt az algoritmust használjuk a mandelbrot halmaz megvalósításához mint az előző feladatban, tehát lényegi változás nincs, csak a részletekről lenne szó.

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

```
// Verzio: 3.1.2.cpp  
// Forditas:  
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2  
// Futtatas:  
// ./3.1.2 mandel.png 1920 1080 2040 ←  
-0.01947381057309366392260585598705802112818 ←  
-0.0194738105725413418456426484226540196687 ←  
0.7985057569338268601555341774655971676111 ←  
0.798505756934379196110285192844457924366  
// ./3.1.2 mandel.png 1920 1080 1020 ←  
0.4127655418209589255340574709407519549131 ←  
0.4127655418245818053080142817634623497725 ←  
0.2135387051768746491386963270997512154281 ←  
0.2135387051804975289126531379224616102874
```

```
// Nyomtatás:  
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --line-numbers=1 --left-footer=" ↵  
    BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵  
    color  
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf  
//  
//  
// Copyright (C) 2019  
// Norbert Bátfai, batfai.norbert@inf.unideb.hu  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.  
//  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <https://www.gnu.org/licenses/>.
```

```
#include <iostream>  
#include "png++/png.hpp"  
#include <complex>  
  
int  
main ( int argc, char *argv[] )  
{  
  
    int szelesseg = 1920;  
    int magassag = 1080;  
    int iteraciosHatar = 255;  
    double a = -1.9;  
    double b = 0.7;  
    double c = -1.3;  
    double d = 1.3;  
  
    if ( argc == 9 )  
    {  
        szelesseg = atoi ( argv[2] );  
        magassag = atoi ( argv[3] );  
        iteraciosHatar = atoi ( argv[4] );  
        a = atof ( argv[5] );  
        b = atof ( argv[6] );  
        c = atof ( argv[7] );  
        d = atof ( argv[8] );  
    }  
}
```

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

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

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

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

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

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

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

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

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

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

            ++iteracio;
        }

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

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

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

}
```

5.3. 5.3 Biomorfok

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

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

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

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

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

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

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

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

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

            ++iteracio;
        }
}
```

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

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

```
for ( int k = 0; k < szelesseg; ++k )
{
    double rez = a + k * dx;
    double imZ = d - j * dy;
    std::complex<double> z_n ( rez, imZ );

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

A biomorfos feladat szerintem nagyon érdekes volt. A biomorfokról érdemes tudni, hogy a Julia halamzok, Ezek a Mandelbrot halmaz részei, és számuk végtelen, ugye egy biomorfban a konstansunk nem változik, emiatt van hogy a Mandelbrot halmaz részei és számuk is végtelen mivel végtelen kontsans létezik. A program makefile-jában tudunk a kirajzolt képen módosítani, ezt több módon is próbáltam. Az r-t lecseréltem 10-ről 5-re, ez a halmazra való ráközelítésen módosított, a minimum és maximum x és y-ok módosításával pedig a halmaz fekete része körüli "ráközelítést" lehetett módosítani, tehát ha ide nagyon piaci értékeket írunk, gyakorlatilag semmi sem fog látszódni a halmazunkból. Ezen kívül a felbontást nagyon megemeltem, hogy a halmaz szélén a színes részek jobban kirajzolódjanak szerintem ugyanus az benne a legösszetettebb textúra és ez a legérdekesebb is egyben. Csatolok néhány képet a a kirajzoltatott png-kről, úgy hogy látszódjanak a makefile-ban megadott értékek is.

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ↫
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↫
// color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
```

```
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9\_Iss5\_2305--2315\_Biomorphs\_via\_modified\_iterations.pdf
//

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );
    }
}
```

```
}

else
{
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesség magasság n a b c ←
        d reC imC R" << std::endl;
    return -1;
}

png::image<png::rgb_pixel> kep ( szelesség, magasság );

double dx = ( xmax - xmin ) / szelesség;
double dy = ( ymax - ymin ) / magasság;

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

std::cout << "Számítás\n";

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

    for ( int x = 0; x < szelesség; ++x )
    {

        double reZ = xmin + x * dx;
        double imZ = ymax - y * dy;
        std::complex<double> z_n ( reZ, imZ );

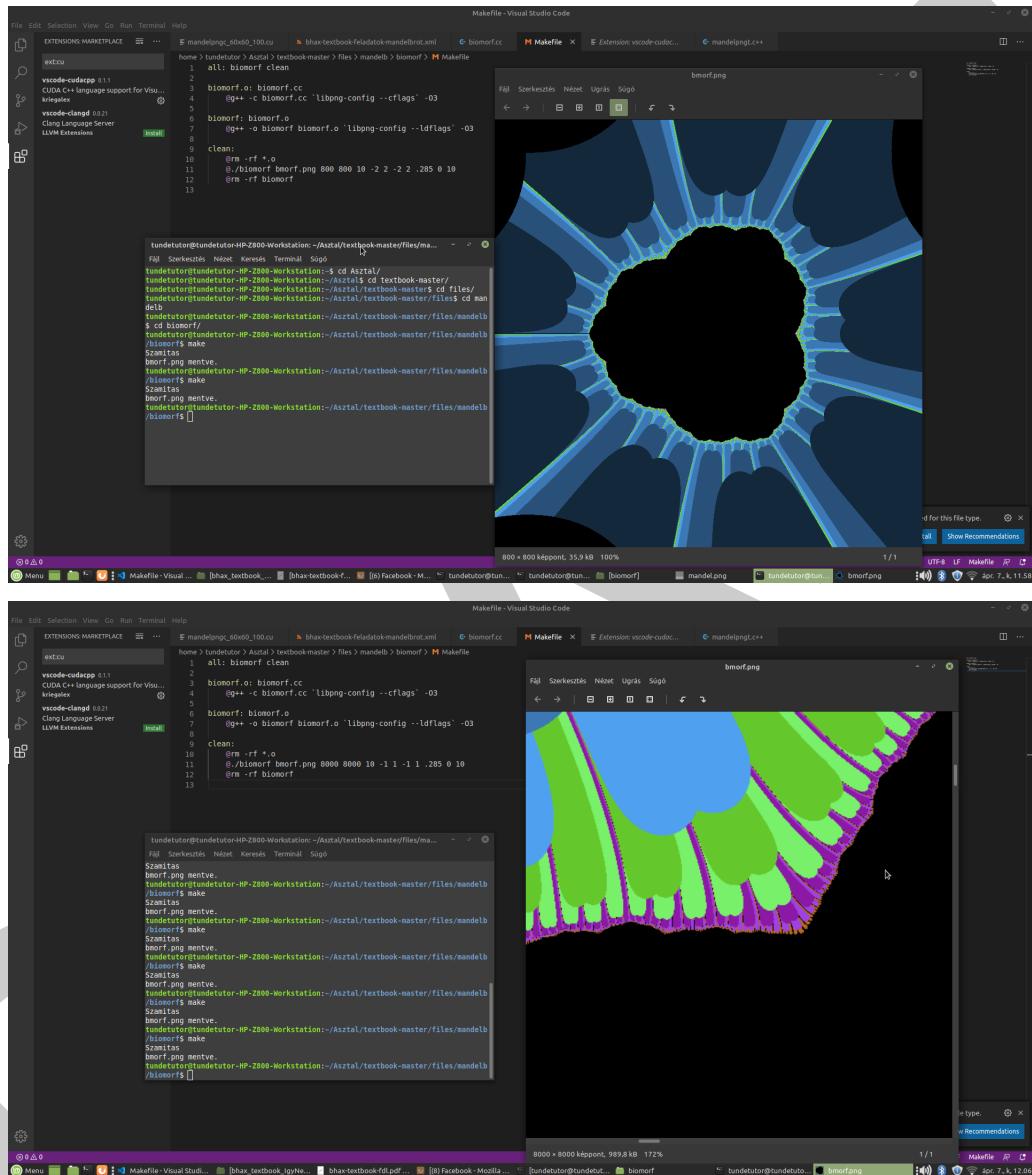
        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {

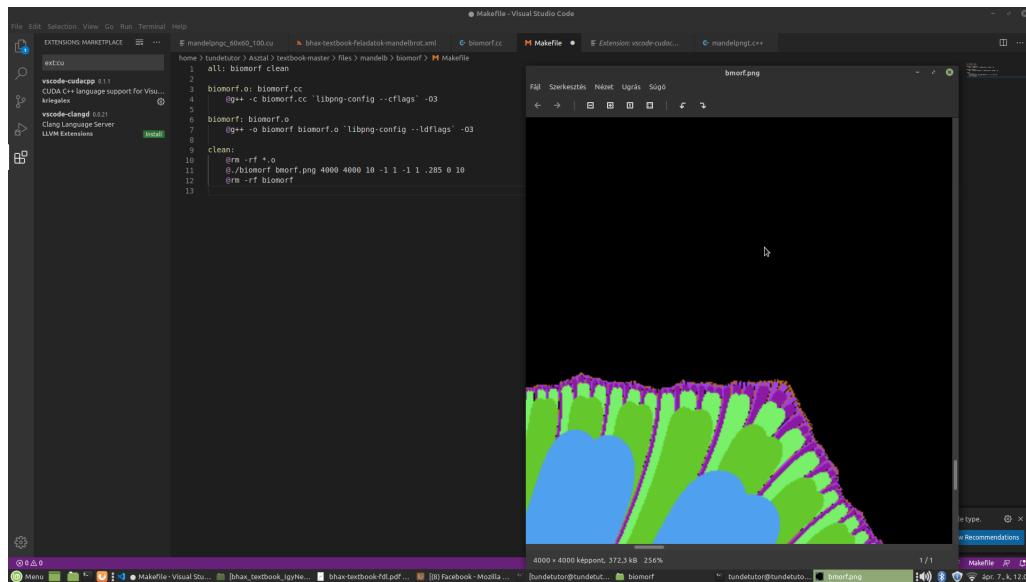
            z_n = std::pow(z_n, 3) + cc;
            //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
            if (std::real ( z_n ) > R || std::imag ( z_n ) > R)
            {
                iteracio = i;
                break;
            }
        }

        kep.set_pixel ( x, y,
                        png::rgb_pixel ( (iteracio*20)%255, (iteracio ←
                            *40)%255, (iteracio*60)%255 ) );
    }
}

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

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





5.4. 5.4 A Mandelbrot halmaz CUDA megvalósítása

Ebben a feladatban ugyanazzal a mandelbrot halmaz algoritmussal dolgozunk mint az eddigi feladatokban is viszont itt van egy nagy különböző az eddigiekhez képest. Itt a feladat lényege hogy a CPU helyett a videókártya számítási kapacitását használjuk ki. Ahhoz hogy ezt meg tudjuk csinálni mindenki szükségünk van egy Nvidia kártyára mivel a CUDA az Nvidia saját fejlesztése.

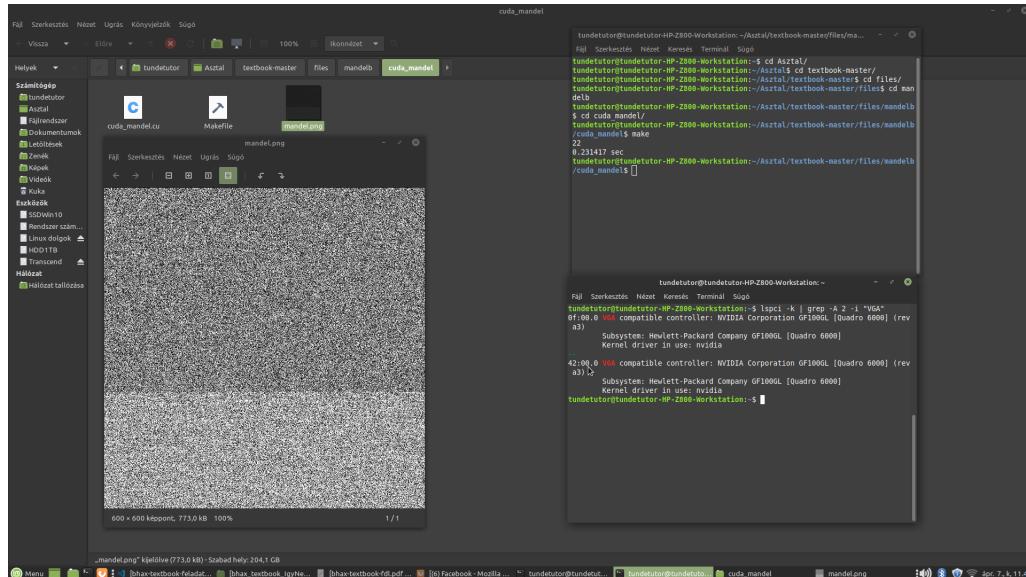
A CUDA lényege, hogy az Nvidia kártyák rendelkeznek bizonyos mennyiségű CUDA magokkal, ezekkel párhuzamosan tudunk egyszerre több számítást is végezni. A CPU is rendelkezik magokkal amellyel párhuzamos számításokat végezhetünk viszont nem annyival mint egy Nvidia kártya és máshogyan is működnek, egy ilyen program esetében a CPU csak egy magot használ. Ahhoz hogy rávegyük a videókártyánkat a számolásra mindenki számításra mindenki telepítenünk kell az Nvidia Cuda Toolkitet. CUDA illesztőprogramokból egyébként találhatunk többet is, ugyanis mostanában kezd egyre jobban elterjedni hogy a deep learning miatt, ami nagy számítási kapacitás igényű folyamat. Véleményem szerint a CUDA a következő évkben eléggé elterjedté válhat ennek köszönhetően. Viszont most nekünk az alap Nvidia Cuda Toolkit tökéletesen megfelel.

Az elképzelés az lenne hogy a png minden pixelét más szál, más CUDA mag tudja kiszámolni. Ha belegen dolunk ez hatalmas újítás ahhoz képest mintha ugyanezt a CPU-val szeretnénk megvalósítani. Nyílván ez a gyakorlatban nem teljesen így van hogy minden pixelre jut egy szál, de a CUDA-val ehhez már közelítünk, gondolunk bele hogy ha egy erősebb GPU-val rendelkezünk amiben közel 1200 cuda mag van valószínűleg a számítás ideje jóval lecsökken a CPU-hoz képest. Azt is érdemes megemlítenünk, hogy hiába van sok magos, sokszálas erős processzorunk ha egy folyamat csak 1 szálat fog használni, nem tudjuk kihasználni az egész teljesítményét még a GPU esetén igen.

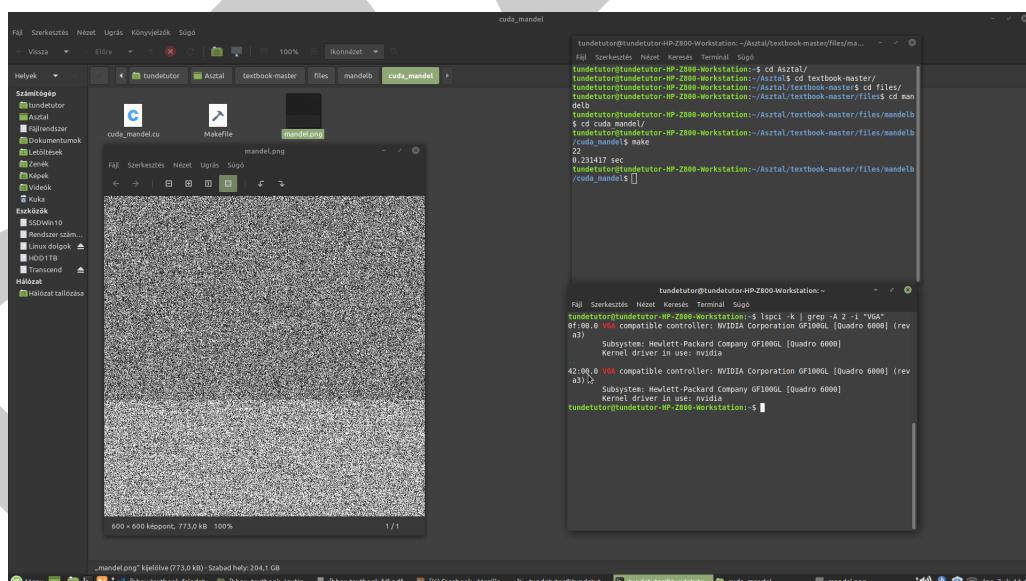
Én a CUDA-t a Blenderben való Cycles rendermotorral történő rendereléshez szoktam is használni. A Blender Cycles-e is ha nem rendelkezünk GPU-val processzorból fog számolni. A renderelés során egy ilyen technológia komoly napokat is megspórolhat nekünk, itt azért már érzi az ember, hogy valóban jelentősége van ennek. Egyébként 3D renderelés esetén hasonló dolog történik, mint a mandelbrot halmaznál, csak ott nem ezt az algoritmust használjuk, hanem a 3D felületet alkotó háromszögek normálvektoraiból lehet meghatározni egy-egy pixel milyen színű legyen.

Nálam megcsinál egy png-t viszont az közel sem a mandelbrot halmaz amit kiad. Az a sejtésem hogy a hardware konfigom a baj, én 2 db Nvidia Quadro kártyát használok, ezek kifejezettem 3D grafikai folya-

matokhoz vannak fejlesztve, viszont még nem sikerült SLI hídba kötnöm őket és szerintem vagy ez okozza a gondot vagy az hogy ezek nem mai kártyák és driver probléma van.



Akárhányszor futattam mindenkor csak hanygafocit rajzol a program. Szerencsére találtam itthon egy másik gépet is (ez egyébként apám) szóval arra is telepítettük a libpng-t majd futtattuk a programot. Abban a gépen egyébként egy Nvidia GTX 1060-as GPU-t találunk, megfelelő drivekkal és persze a CUDA Toolkit-et is már régen feltelepítettük rá. Csodák csodájára ez szépen kirajzolta a halmazt, természetesen ugyanazt a programot futtatva ami nekem ezeket a furcsa zajos képeket adta. Nyilvánvaló, hogy valami probléma van az iterációk számítása során. Amikor elhoztam a gépemet ezekkel a GPU-kkal tudtam, hogy kompromisszumokat kell kötnöm, 10 éves technológiáról beszélünk.



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

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

5.5. 5.5 Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása:

Ebben a feladatban olyan programot kellett készíteni amivel már valódi GUI-t (Graphical User Interface) is tudunk használni. Ahhoz, hogy egy ilyet lehessen készíteni szükségünk van QT library-re. Ezzel egyszerű GUI-kat lehet létrehozni, a nagyító utazó programot az egérrel lehet használni, illetve kap saját programablakot. Ez nagyon nagy újítás az eddigi csak terminálban futó programjainkhoz képest, és úgy érezhetjük, hogy ez egy mérföldkő. Viszont ehhez telepítenünk kell a QT-t.

5.6. 5.6 Mandelbrot nagyító és utazó Java nyelven

Itt az előző programunk java átíratával van dolgunk. Ez szerencsére nem okozott az előzőhöz hasonló gondot, csak a jdk 8-at kellett telepítenünk,

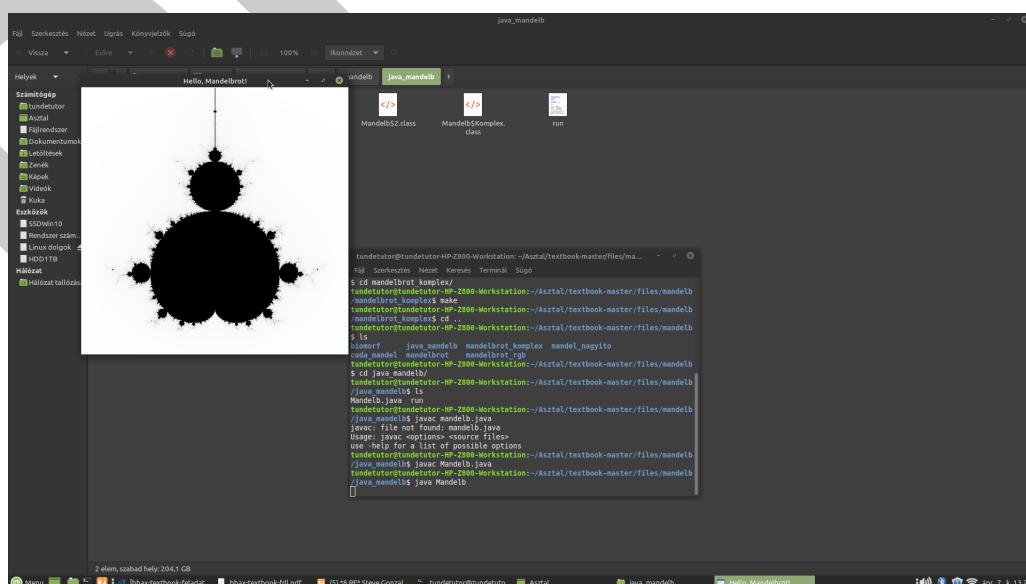
```
sudo apt-get install openjdk-8-jdk
```

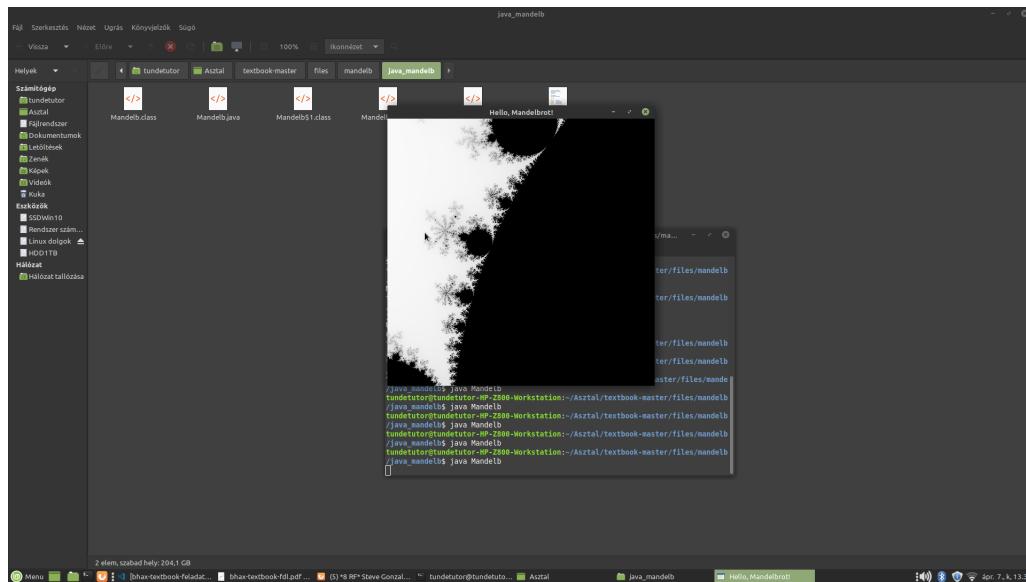
ez volt a csomagtárolóban is, tehát egy parancssal ezt a gondot le is tudtuk. A futtatás sem okozott különösebb problémát. A futtatás így nézett ki:

```
javac mandelb.java
```

```
java mandelb
```

A C++ program fordítása és futtatása is elég macerás volt egyébként, ahhoz képest a a java-val nagyon könnyű dolgunk volt. Az eredményt pedig néhány screenshot formájában csatalom ide. Vicces volt kipróbálni hogy a nagyító meddig nagyít, egy idő után sajnos elértem azt hogy elpixeleződött a kép.





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

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

5.7. 5.7 Steve felszalad a láváig

Ebben a programban Steve első dolga, hogy elszadjon a láváig, ezt move, jumpmove párosokkal teszi egy while cikluson belül aminek a kilépései feltételében szerepel, az általa érzékelt kockákból ha az előtte és fölötté előtte lévő kockákban "flowing_lava" vagy "lava szerepel". Ha ez megtörténik Steve leszalad és elkezdi csigában gyűjteni a virágokat. A program lefutása az alábbi videóban látható.

Megoldás videó: <https://www.youtube.com/watch?v=zO6cNp8L4-Q>

6. fejezet

Helló, Welch!

6.1. 6.1 Első osztályom

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

Megoldás videó:

A következő java kód Bátfai Norbert Tanár Úr kódja, a polártranszformációs algoritmus segítségével véletlenszám generálást tudunk végrehajtani. Ez is, a fejezetben objektum orientált programokkal foglalkozunk. A java alapvetőleg objektum orientált nyelv, ahhoz hogy tudjuk a programot futtatni, ugyanúgy kell elneveznünk a .java fájlt is mint ami a benne létrehozott osztály neve. Nézzük is meg ez a java program hogyan fut.

```
public class PolarGenerator {
    boolean nincsTarolt = true;
    double tarolt;
    public PolarGenerator() {
        nincsTarolt = true;
    }
    public double kovetkezo() {
        if(nincsTarolt) {
            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2*u1 - 1;
                v2 = 2*u2 - 1;
                w = v1*v1 + v2*v2;
                if(w > 1) {
                    u1 = Math.sqrt(w)*cos(2*pi*r);
                    u2 = Math.sqrt(w)*sin(2*pi*r);
                }
            } while(u1*u1 + u2*u2 > 1);
            nincsTarolt = false;
            tarolt = Math.sqrt(u1*u1 + u2*u2);
        }
        return tarolt;
    }
}
```

```
        } while(w > 1);

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

        tarolt = r*v2;
        nincsTarolt = !nincsTarolt;

        return r*v1;

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

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

Fordítás és futtatás után a következő gyönyörű eredményeket kaptuk:

```
0.500010678804642
0.7466265624656746
1.0803216647807399
-0.32064099460970763
-2.5477451034196923
0.6811730798994344
-0.44975361547907916
-0.9422083605110528
0.49015177151970096
2.058535110772562
```

6.2. 6.2 LZW

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

A feladatcsokorban ez volt szerintem a legfontosabb feladat és a többi is főleg erre épített. Ezt a programot, Bátfau Norbert Tanár Úr videói alapján írtam meg. Most ezt a bináris fát a nulláról kezük el írni, ez a "from scratch" fánk. Viszont felmerül a kérdés, mi is maga a bináris fa?

A bináris fák adatstrukturák. A félév során az Adatszerkezetek és algoritmusok nevű tantárgy is sokat foglalkozik ezzel a fajta adatstruktúrával. A bináris fákban különböző adattípusokat tudunk tárolni. Az egyik legegyszerűbb talán amiben csak egész számokat tárolunk, ezen keresztül bemutatom a működését. Ha

adott egy egész számokat tartalmazó rendezett tömbünk, annak elemeit bináris fában is tudjuk tárolni. Ezzel lényegesen lerövidíthetjük az adathalmazunkban történő keresés idejét. A bináris fáknak van 1 db gyökere, és 2 db gyermekje a gyökérnek, egy jobb és egy baloldali. Ha az egész számos példánál maradunk, akkor a gyökér a tömbként ábrázolt adathalmazban a középső indexű elem, a bal oldali gyermekje a gyökérnél kisebb értékek között a középső elem, a jobboldalinál pedig az annál nagyobbak közti középső elem és így tovább minden gyerekkel és azok gyermekivel. Tehát a bináris fánk szintenként duplázódik elemszámot tekintve, ez azt jelenti, hogy egy levélemet (olyan csomópont amely nem rendelkezik gyermekkel) könnyedén elérhetővé válik. Ha van plkb 2500 adatunk, látható, hogy egy keresett elem maximum 11 lépésből elérhető, ez adja az algoritmus gyorsaságát.

A programunkban tárgyalta bináris fa első sorban 0 és 1 értékkal dolgozik, ezeket úgy rendezzük el a fában, hogy ha a vizsgált elem 0 akkor balra kerül, ha 1 akkor pedig jobbra. A bemenő adatokat vizsgáljuk és ha nem létezik még olyan csomópont mint a bemenő adat akkor létrehozunk egyet és a mutatót visszaállítjuk a gyökérre, ha már létezik akkor pedig ráállítjuk a mutatót. Ilyen módon tudjuk magát a fát felépíteni.

A videóban ami alapján én is írtam a programot, Bátfai Norbert z alábbi bitsorral tesztelte a programját, így én is úgy döntöttem azzal tesztelem, hogy biztos legyek annak működésében. Szerencsére jól lefutott a program és ezt a kimenetet kaptam:

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

6.3. 6.3 Fabejárás

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

Megoldás videó:

Ehhez a feladathoz is a "from sracth" programot használtam. Eza kód alapvetőleg inorder bejárást használ, azaz először a baloldalt vizsgálja, aztán az adott csomópont gyökerét majd annak a jobb oldali gyermekét, tehát ha inorder kiíratást végezünk magának a fának a gyökere középen fog megjelenni. Nézzünk is meg egy kódcsipetet és az ilyen módonbejárt fa kimetetét is.

```
template <typename ValueType>
void BinTree<ValueType>::print(Node *node, std::ostream & os)
{
    if (node)
    {
        ++depth;
```

```
print(node -> leftChild(), os);

for(int i = 0; i < depth; ++i)
{
    os << "___";
}

os << node -> getValue() << ' ' << depth << ' ' << node -> getCount() << ' ' << std::endl;

print(node -> rightChild(), os);
--depth;
}
}
```

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

A következő bejárás a preorder bejárás lesz, ez azt jelenti, hogy először minden gyökeret vizsgáljuk, azt követően a előzör a baloldalt, majd a jobb oldalt. Itt is megmutatom a kódcsipetet és a kimenetet is, ebben az esetben a teljes fa gyökere lesz az első kiíratott elem.

```
template <typename ValueType>
void BinTree<ValueType>::print(Node *node, std::ostream & os)
{
    if(node)
    {
        ++depth;

        for(int i = 0; i < depth; ++i)
        {
            os << "___";
        }

        os << node -> getValue() << ' ' << depth << ' ' << node -> getCount() << ' ' << std::endl;

        print(node -> leftChild(), os);
    }
}
```

```
    print(node -> rightChild(), os);
    --depth;
}
}
```

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

Végül nézzük meg a postorder bejárást is. Ennél a bejárásnál az előzőhez képest annyi különbség van, hogy a yökeret vizsgáljuk utolára, de a balról-jobbra bejárás megmarad. Ekkor az utolsó kiíratott elem lesz magának a fának a gyökere.

```
template <typename ValueType>
void BinTree<ValueType>::print(Node *node, std::ostream & os)
{
    if(node)
    {
        ++depth;

        print(node -> leftChild(), os);

        print(node -> rightChild(), os);

        for(int i = 0; i < depth; ++i)
        {
            os << "----";
        }

        os << node -> getValue() << ' ' << depth << ' ' << node -> getCount() << ' ' << std::endl;

        --depth;
    }
}
```

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

6.4. 6.4 Tag a gyökér

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

Megoldás videó:

Ehhez a feladathoz egy másik bináris fa programot használtam, a z3a7.cpp névvel elátottat, ebben a programban a gyökér alapból tagja a fának, tehát kompozícióban van azzal. Ezt az alábbi apró kódcsipetben megtekinthetjük. És a következő feladatban pedig megmutatom, hogyan lehet ugyanezt a programot úgy módosítani, hogy a gyökérből mutató váljon.

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

Egyébként ezt és a következő programot is lefutattam a COVID19 vírus genomjával, mint bemenet és ugyanazt kaptam eredményül. Ezt csak mint érdekesség gondoltam megjegyzem, persze ennek a kimenetét nem csatolnám mivel annál egy 41 mélységű fáról beszélünk.

6.5. 6.5 Mutató a gyökér

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

Megoldás videó:

Az általunk használt "from scratch" fás programban a gyökér eleve mutató ezért kerestem egy násik forrást ahol nem az, ez a z3a7.cpp, és úgy gondoltam érdekesebb ha ebben a feladatban ebből a forrásból készítünk olyat ahol a gyökér mutató, mintha megmutatnám azt amelyiket alapból úgy írtuk.

```
LZWBinFa ()  
{  
    gyoker = new Csomopont ('/');  
    fa = gyoker;  
}  
~LZWBinFa ()  
{  
    szabadit (gyoker->egyesGyermekek());  
    szabadit (gyoker->>nullasGyermekek());  
}
```

Itt ebben a kódcsipetben látszik, hogy a gyökérből az előzőhöz képest mutatót készítettünk, a gyökér is már új csomópont. Persze a kód többi részét is kelelt módosítanunk, de szerintem itt az LZWBinFa osztályban látszik a legjobba a különbség.

6.6. 6.6 Mozgató szemantika

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

Megoldás videó:

Ehhez a feladathoz azt a programot fejlesztük tovább amit a 2. feladathoz használtunk. Itt mindenki szóba kell hoznunk, hogy ha a fát szeretnénk mozgatni, akkor ezt csak úgy tehetjük meg a BinTree ősosztályba beágyazott Node osztálytal együtt rekurzívan hajtjuk végre. Egyébként készítetünk másoló és másoló értékkadó konstruktort is de most azok kódcsipetét nem teszem bele a könyvbe, lehet a későbbiek során még belekerül, de most a mozgató szemantikán volt a hangsúly. Nézzünk egy kimenetet is, mivel látható a kódcsipetben hogy a mozgató konstruktorba tesztelés cljából tettünk kiíratást is.

```
BinTree & operator=(const BinTree & old)  
{  
    std::cout << "BT copy assign ctor, masolo ertekadas" << std::endl;  
  
    BinTree tmp{old};  
    std::swap(*this, tmp);  
    return *this;  
}  
BinTree(BinTree && old){  
    std::cout << "BT move ctor, mozgato " << std::endl;  
  
    root = nullptr;
```

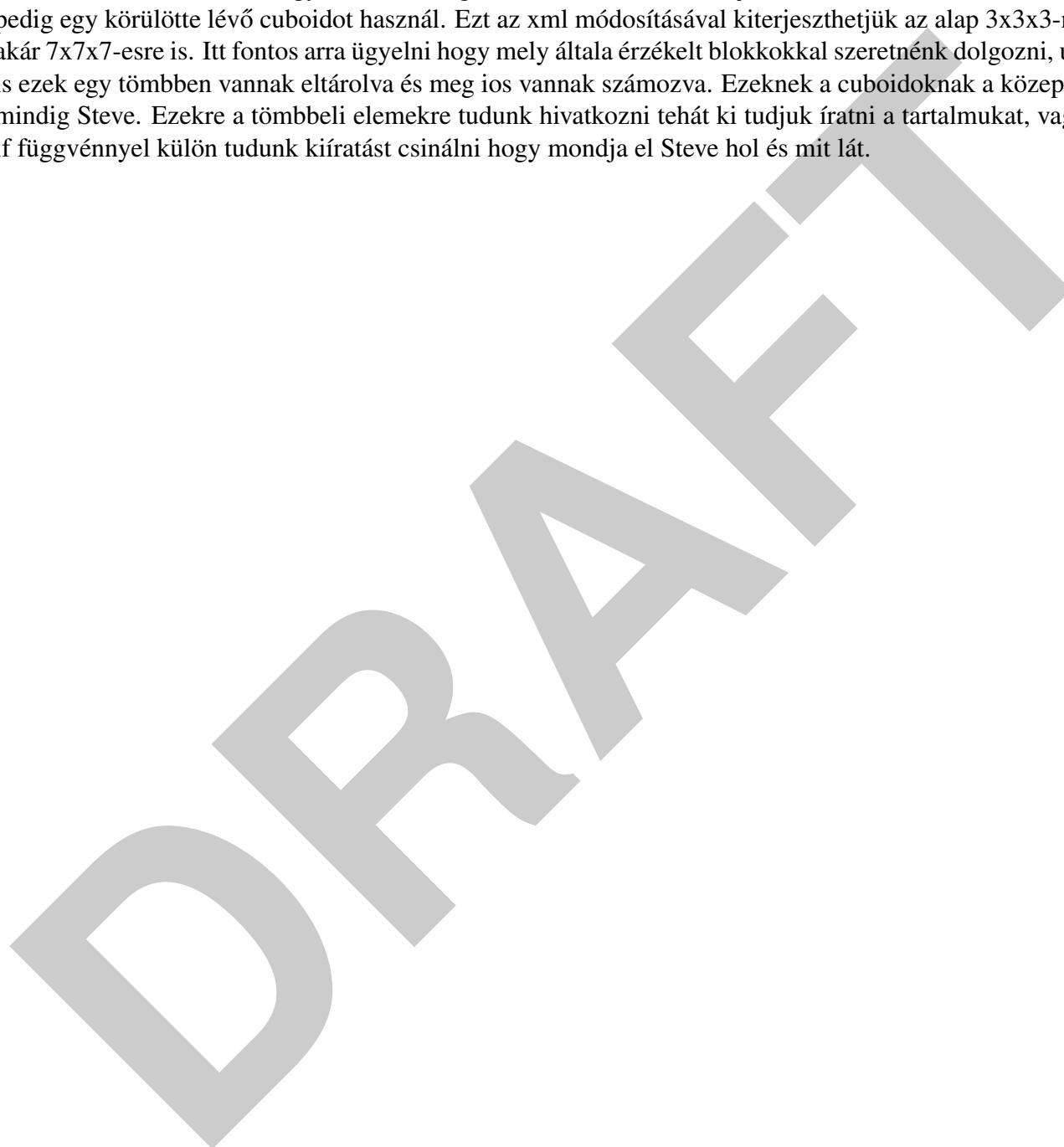
```
*this = std::move(old);  
  
}  
BinTree & operator=(BinTree && old) {  
    std::cout << "BT move assign ctor, mozgato ertekekadas" << std::endl;  
  
    std::swap(old.root, root);  
    std::swap(old.treep, treep);  
  
    return *this;  
}
```

```
BT ctor  
-----2 3 0  
-----5 2 0  
-----7 3 0  
---8 1 0  
-----9 2 0  
  
BT ctor  
-----0 3 0  
-----0 2 0  
---/ 1 0  
BT copy ctor, masolo konstruktor  
BT ctor  
***  
BT copy assign ctor, masolo ertekekadas  
BT copy ctor, masolo konstruktor  
BT move ctor, mozgato  
BT move assign ctor, mozgato ertekekadas  
BT move assign ctor, mozgato ertekekadas  
BT move assign ctor, mozgato ertekekadas  
BT dtor  
BT dtor  
***  
BT move ctor, mozgato  
BT move assign ctor, mozgato ertekekadas  
BT dtor  
BT dtor  
BT dtor  
BT dtor  
BT dtor
```

6.7. 6.7 Steve szemüvege

Megoldás videó: <https://www.youtube.com/watch?v=DX8dI04rWtk>

Steve 2 módon láthat. Az egyik a lineOfSight, azaz azt a blokkot látja amin a kereszt van. A másik mód pedig egy körülötte lévő cuboidot használ. Ezt az xml módosításával kiterjeszhetjük az alap 3x3x3-masról akár 7x7x7-esre is. Itt fontos arra ügyelni hogy mely általa érzékelt blokkokkal szeretnénk dolgozni, ugyanis ezek egy tömbben vannak eltárolva és meg vannak számozva. Ezeknek a cuboidoknak a közepén van minden Steve. Ezekre a tömbbeli elemekre tudunk hivatkozni tehát ki tudjuk íratni a tartalmukat, vagy egy if függvényel külön tudunk kiíratást csinálni hogy mondja el Steve hol és mit lát.



7. fejezet

Helló, Conway!

7.1. 8.1 Hangyaszimulációk

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

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

Megoldás forrása:

Ebben a feladatban egy hangyaszimulációs programot vizsgálunk. Ahhoz, hogy megértsük működésének alapjait, ismernünk kell a valódi hangyák viselkedését is. Nekem egy rövid ideig, volt egy kb 30 hangyból álló apró hangyakolóniám, sajnos a felelőtlenségem miatt elpusztultak (nem megfelő ételt adtam nekik) de addig is picit beleláttam, hogy hogyan élnek együtt. A hangyák úgynevezett kolónitúdattal élnek, egy-egy példánynak esélye sincs egyedül túlélni, nagyon szorosan egymásra vannak utalva, gyakorlatilag a kolóniára úgy kell tekintenünk mint egy állatra, mikor ettettem a hangyáimat akkor is ezt a szempontot kellett szem előtt tartani, hogy úgy adjak nekik élelmet, hogy az a teljes bolynak elég legyen. A valóságtól való első számú különbség a szimulációs programban, hogy itt a hangyabolyban nincs királynő, pedig egyébként ténylegesen ő a hangyaboly legfontosabb eleme, tehát ha ő elpusztul a boly nem tud túlélni, elsősorban amiatt hogy csak a királynő tud petéket lerakni, de egyébként sem tudnak királynő nélkül elni.

```
File Edit Selection View Go Run Terminal Help
home > tundelutor > Asztal > JátékokMappa > hangya > antwin.cpp > Antwin.cpp > moc_antwin.cpp > moc_antwin.h
75
76     grid = grids[gridIndex];
77
78     for ( int i=0; i<height; ++i ) {
79         for ( int j=0; j<width; ++j ) {
80
81             double rel = 255.0/max;
82
83             qpainter.fillRect ( j*cellWidth, i*cellHeight,
84                                 cellWidth, cellHeight,
85                                 qColor ( 255 - grid[i][j]*rel, // 255 - grid[i][j]*rel )
86
87             );
88
89             if ( grid[i][j] != min )
90             {
91                 qpainter.setPen (
92                     QPen (
93                         qColor ( 155 - grid[i][j]*rel,
94                                 95 - grid[i][j]*rel, 155 ),
95                         1 )
96                 );
97
98                 qpainter.drawRect ( j*cellWidth, i*cellHeight,
99                                     cellWidth, cellHeight );
100
101
102
103
104                 qpainter.setPen (
105                     QPen (
106                         qColor ( 150,150,150 ), // 0,0,0
107                         1 )
108
109                 );
110
111                 qpainter.drawRect ( j*cellWidth, i*cellHeight,
112                                     cellWidth, cellHeight );
113
114             }
115
116             for ( auto h : *ants ) {
117                 qpainter.setPen ( QPen ( Qt::black, 1 ) );
118
119                 qpainter.drawRect ( h.x*cellWidth, h.y*cellHeight+1,
120                                     cellWidth-2, cellHeight-2 );
121
122         }
```

Maga program egyébként 5 db C++ kódból és 4 db headerból áll. Nem fogom minden részletesen bemutatni, mert nekem is feleslegesen sok munka lenne és az olvasót sem szeretném untatni, szóval a számonra érdekesebb részkről fogok írni. Mivel GUI-cal szeretnénk megjeleníteni a programot, ezrt ismét a Qt-t hívtuk segítségül, végül sikerült ezt a Linux Mint-emen is megoldani és a qmake make parancsok majd a "myrmecologist" futtatása után szépe kirajzolódott a hangyabolyom. Az ablakban a következőket láthatjuk: a pici mozgó négyzetek nyílván a hanygáink, ezek zöld szálakat húznak maguk után, ezek a maguk mögött hagyott feromon nyomok, ezek alapján tudnak megfelően mozogni, és láthatóan egy nagy rácshálót, ami az egész programablakot beteríti. Ez a rács szerintem rendkívül zavarja a szemet és az ember nem szívesen nézi így perceken át a hangyákat szóval én azt világosszürkére állítottam, látható az alábbi, antwin.cpp-ből kódcsipetben, a QPaintEvent felel ugyanis a megjelenítésért.

```
qpainter.drawRect ( j*cellWidth, i*cellHeight,  
                    cellWidth, cellHeight );  
  
    }  
}
```

Az antwin.cpp-ben ebben a kódrészletben van egyébként nagyon sok dolog ami a GUI megjelenítésért felel. Mos más dolgunk nem igazán volt a Qt-vel, nincs mouseMove Event stb. Mivel ez egy szimuláció, nyilván nem interaktív.

A programunkban tehát jelenleg csak dolgozókkal találkozunk és az ő mozgásukat tudjuk megfigyelni. Ha elindítottuk a programot, akkor láthatjuk, hogy különböző pontokon jelennek meg a hangyáink, ahhoz hogy így induljon el a program és ne egy pontban jelenjen meg az összes hangya, a main.cpp-ben a qsrand függvény segítségére volt szükség, erről teszek is ide egy kódcsipetet. Fontos ugye, hogy ne egy helyen jöjjenek létre.

```
qsrand ( QDateTime::currentMSecsSinceEpoch() );
```

7.2. 8.2 Java életjáték

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

Megoldás videó:

Megoldás forrása:

A játékot John Horton Conway, Cambridge-i matematikus alkotta meg. A játék egy sejtautomata. A lényege az hogy, szimulálja a sejtek életét, egyszerű szabályok meghatározásával. Maga a "játékmenet" egyébként elég passzív, a játkosnak annyi dolga van, hogy az első generációni sejtet ő helyezi el a rácshálón majd figyeli mi történik velük.

Mivel a következő feladat is ugyanerről az életjátékról szól C++ megvalósításban arra gondoltam, hogy a játékról magáról itt írok bővebben, a következő feladatban pedig inkább a kódot vizsgáljuk. Szóval ismerjük is meg a szabályokat!

A sejtek a rácshálóban egy 8 rácspontnyi távolságban tudnak "érzékeéni". Ez azért fontos mert egy sejt önmagában elpusztul, szüksége van más sejtekre is a környezetében, ezeknek a száma 2 és 3 lehet, a közvetlen szomszedságában. Tehát 1 sejt 1 szomszéddal elpusztul de ha 3-nál több szomszédja van, akkor már "túlszaporodtak" és megint elpusztul az a sejt amelynek több szomszédja van. Új sejt születik akkor ha egy sejtnak pontosan 3 szomszédja van. A játék körökre, generációra bontható, és minden generáció változásait vizsgálhatjuk. A játék szabályai ilyen egyszerűek mégis nagyon összetett dolgokkal találkozhatunk.

A játék során kialakulhatnak úgynevezett stabil alakzatok, ilyen például egy egyszerű négyzet amely 4 db sejtből áll, ekkor minden sejtnek 3 szomszédja van tehát túlélik a kört. Az egyik leghíresebb azakzat az a "sikló" amely átlósan tud mozogni is, iletve a vízszintesen mozgó űrhajó. Ezek olyan alakzatok amelyek idővel "mozgásuk során" önmagukba alakulnak vissza. Csatolok is róluk ide egy képet, arról mikor futattam ezt a programt:

Sejtautomata.java - Visual Studio Code

```
home tundekor@Asztal: HézoldMapa> javac Sejtautomata.java & antbuild.cgi & antwin
```

```
Fájl Szerkesztő Nézet Keresési Terminal Súgó
```

```
tundekor@tundekor-HP-Z800-Workstation:~/Asztal/JátzszoMapa$ javac Sejtautomata.java Sejtautomata error: class names, 'Sejtautomata', are only accepted if annotation processing is explicitly requested 1 error
```

```
tundekor@tundekor-HP-Z800-Workstation:~/Asztal/JátzszoMapa$ cd ..
```

```
tundekor@tundekor-HP-Z800-Workstation:~/Asztal/JátzszoMapa$ javac Sejtautomata.java Sejtautomata error: class names, 'Sejtautomata', are only accepted if annotation processing is explicitly requested 1 error
```

```
tundekor@tundekor-HP-Z800-Workstation:~/Asztal/JátzszoMapa$ cd ..
```

```
tundekor@tundekor-HP-Z800-Workstation:~/Asztal/JátzszoMapa$ javac Sejtautomata.java Sejtautomata
```

```
Sejtautomata
```

```
L1. Col 1 Space 4 LF Java Apr 27, 6:13 AM
```

```
File Edit Selection View Go Run Terminal Help
```

```
bxh_textbook-fejlesztői.companymail~/tutasi Sejtautomata.java X antbuild.cgi antwin
```

```
13 private int szellesség; 14 private boolean pillaantevetel = false; 15 private int pillaanteveteltszámoló = 0;
```

```
16 public Sejtautomata(int szellesség, int magasság) { 17     this.szellesség = szellesség; 18     this.magasság = magasság; 19     rások[0] = new boolean[magasság][szellesség]; 20     rások[1] = new boolean[magasság][szellesség]; 21     rácslIndex = 0; 22     ráscs = rások[rácslIndex]; 23     for (int i = 0; i < rások[0].length; ++i) 24         for (int j = 0; j < rások[0].length; ++j) 25             ráscs[i][j] = MALOTT; 26     sikerLövööracs = 5, 60; 27     addMouseListener(new java.awt.event.WindowAdapter() { 28         public void windowClosing(java.awt.event.WindowEvent e) { 29             setVisible(false); 30             System.exit(0); 31         } 32     }); 33     addKeyListener(new java.awt.event.KeyAdapter() { 34         public void keyPressed(java.awt.event.KeyEvent e) { 35             if(e.getKeyCode() == java.awt.event.KeyEvent.VK_I) { 36                 if(e.getExtendedKeyCode() == java.awt.event.KeyEvent.VK_N) { 37                     cellaSzélesítés -= 2; 38                     cellaMagasság += 2; 39                     setSzeleseztotta(this.szellesség*cellaSzélesség, 40                     Sejtautomata.this.nagasság*cellaMagasság, 41                     validate()); 42                 } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) { 43                     cellaSzélesség += 2; 44                     cellaMagasság -= 2; 45                     setSzeleseztotta(this.szellesség*cellaSzélesség, 46                     Sejtautomata.this.nagasság*cellaMagasság, 47                     validate()); 48                 } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S) { 49                     pillaantevetel = !pillantásrelejtve; 50                     if(e.getExtendedKeyCode() == java.awt.event.KeyEvent.VK_0) 51                         varázkozás /= 2; 52                     else if(e.getExtendedKeyCode() == java.awt.event.KeyEvent.VK_L) 53                         varázkozás *= 2; 54                     repakolt(); 55                 } 56             } 57         } 58         addMouseListener(new java.awt.event.MouseAdapter() { 59             public void mousePressed(java.awt.event.MouseEvent m) {
```

```
15.0.2
```

```
Matematika Ugyanúval Balazs_Gyula [Szerzők] Conway GoJava [bxh_textbook] bxh_textbook_i... bxh_textbook_ja... tundekor@tundekor... Sejtautomata
```

Fontosnak tartom megjegyezni, hogy olyan lakzatok is kirajzolódhatnak a játék futása során amelyek természetes jelenségekre hasonlítanak. Ilyen pl.: egy furcsa tintapaca szerű nagyjából összefüggő alak, az ilyen jellegű mintákat sok helyen tudják hasznosítani, ilyen lehet mondjuk a barlagok generálása is, aminél ezt egyébként nagyon gyakran alkalmazzák is.

Na és ha már java, nem hagyhatom ki hogy megmutassam miket találtam, a youtube-on. Több minecraftos megvalósítása is létezik ennek a játéknak, mégpedig úgy, hogy a 3D-s térben zajlanak ezek az események.

Minecraft 3D <https://www.youtube.com/watch?v=wNypW-aSCmE>

7.3. 8.3 Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása:

Ez a Qt C++ megvalósítás egy egészen összetett, objetumorientált program. Először a sejtablak.h-val kezdeném. Ebben a programban, rögzítjük a változókat, szélességet, magasságot, illetve meghatározzuk, hogy

eg ysejt lehet élő vagy halott sejt.

```
class SejtAblak : public QMainWindow
{
    Q_OBJECT

public:
    SejtAblak(int szelesseg = 100, int magassag = 75, QWidget *parent = 0);

    ~SejtAblak();
    // Egy sejt lehet elo
    static const bool ELO = true;
    // vagy halott
    static const bool HALOTT = false;
    void vissza(int racsIndex);
```

Eztán találunk egy protected részt, itt van egy ***racsok mutató, ez azért, kell mert egyszerre 2 db mátrixra szeretnénk rámutatni, ez amiatt kell, mert amikor lépünk generációt, nem kerülhet oda élő sejt ahol halott van. Ezen kívül ami még fontos, hogy itt találjuk a paintEventet, és a void siklo illetve a void sikloKilovo-t is. A sikloKilovo azért fontos mert ebben a verzióban nem a felhasználóra van bízva a játék kezdete, hanem előre defeniált.

```
protected:

    bool ***racskok;

    bool **racs;

    int racsIndex;

    int cellaSzelesseg;
    int cellaMagassag;

    int szelesseg;
    int magassag;
    void paintEvent(QPaintEvent* );
    void siklo(bool **racs, int x, int y);
    void sikloKilovo(bool **racs, int x, int y);
```

Tekintsük meg a sejtablak.cpp-t is! Itt több függvényt is találunk, az első a sejtAblak, itt hozzuk létre magát a programablakot, szélességgel, magassággal, a cellák szélességével és magasságával együtt, illetve ebben létrehozzuk a mátrixainkat is. Illetve leszögezzük, hogy a kiinduló programban minden cella halott. Ezután pedig start függvény tudja indítani a programot.

Az alábbi kódcsipetben pedig megtaláljuk a paintEvent-et. Itt is mátrix-szal dolgozunk.

```
void SejtAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);

    // Az aktualis
    bool **racs = racsok[racsIndex];
    // racsot rajzoljuk ki:
    for(int i=0; i<magassag; ++i) { // vegig lepked a sorokon
        for(int j=0; j<szelesseg; ++j) { // s az oszlopok
            // Sejt cella kirajzolasa
            if(racs[i][j] == ELO) //feltölti az adott színnel
                qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                                    cellaSzelesseg, cellaMagassag, Qt::black);
            else
                qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                                  cellaSzelesseg, cellaMagassag, Qt::white);
            qpainter.setPen(QPen(Qt::gray, 1));

            qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
                              cellaSzelesseg, cellaMagassag);
        }
    }

    qpainter.end();
}
```

Itt ebben a kódcsipetben pedig a folyamatos frissülésért felelős függvényt találjuk:

```
void SejtAblak::vissza(int racsIndex) //frissít
{
    this->racsIndex = racsIndex;
    update(); // ő meghívja a paint eventet
}
```

Ebben a részben pedig az egész sejtszal.cpp-t találjuk. Ennek érdekessége, hogy az előző feladatban említett szabályok vannak leírva C++ nyelven.

```
#include "sejtszal.h"

SejtSzal::SejtSzal(bool ***racsok, int szelesseg, int magassag, int ←
                   varakozas, SejtAblak *sejtAblak)
{
    this->racsok = racsok;
    this->szelesseg = szelesseg;
```

```
this->magassag = magassag;
this->varakozas = varakozas;
this->sejtAblak = sejtAblak;

    racsIndex = 0;
}

/***
 * Az kerdezett allapotban levo nyolcszomszedok szama.
 *
 * @param   racs     a sejtter racs
 * @param   sor      a racs vizsgalt sora
 * @param   oszlop   a racs vizsgalt oszlopa
 * @param   allapor  a nyolcszomszedok vizsgalt allapota
 * @return int a kerdezett allapotbeli nyolcszomszedok szama.
 */
int SejtSzal::szomszedokSzama(bool **racs,
                                int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
    // A nyolcszomszedok vegigzongorazasa:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgalt sejtet magat kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejtterbol szelenek szomszedai
                // a szembe oldalakon ("periodikus hatarfeltetel")
                int o = oszlop + j;
                if(o < 0)
                    o = szelesseg-1;
                else if(o >= szelesseg)
                    o = 0;

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

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

/***
 * A sejtter idobeli fejlodese a John H. Conway fele
 * eletjatek sejtautomata szabalyai alapjan tortenik.
 * A szabalyok reszletes ismerteteset lasd peldaul a
 * [MATEK JATEK] hivatkozasban (Csakany Bela: Diszkret
```

```
* matematikai jatekok. Polygon, Szeged 1998. 171. oldal.)  
*/  
void SejtSzal::idoFejlodes() {  
  
    bool **racsElotte = racsok[racsIndex];  
    bool **racsUtana = racsok[(racsIndex+1)%2];  
  
    for(int i=0; i<magassag; ++i) { // sorok  
        for(int j=0; j<szelesség; ++j) { // oszlopok  
  
            int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);  
  
            if(racsElotte[i][j] == SejtAblak::ELO) {  
                /* Elo elo marad, ha ketto vagy harom elo  
                szomszedja van, kulonben halott lesz. */  
                if(elok==2 || elok==3)  
                    racsUtana[i][j] = SejtAblak::ELO;  
                else  
                    racsUtana[i][j] = SejtAblak::HALOTT;  
            } else {  
                /* Halott halott marad, ha harom elo  
                szomszedja van, kulonben elo lesz. */  
                if(elok==3)  
                    racsUtana[i][j] = SejtAblak::ELO;  
                else  
                    racsUtana[i][j] = SejtAblak::HALOTT;  
            }  
        }  
    }  
    racsIndex = (racsIndex+1)%2;  
}  
  
/** A sejtter idobelő fejlodese. */  
void SejtSzal::run()  
{  
    while(true) {  
        QThread::msleep(varakozas);  
        idoFejlodes();  
        sejtAblak->vissza(racsIndex);  
    }  
}  
  
SejtSzal::~SejtSzal()  
{  
}
```

7.4. 8.4 BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Aki, a félév elején bekapcsolódott a TP vadászatba, annak ismerős lesz ez a program. Ez a BrainB program Bátfai Norbert Tanár Úr illetve más fejlesztők által megírt program. Maga a program egy játék, ami gyakorlatilag egy pszichológiai teszt. A lényeg a következő: Van egy felületünk, amin félre átlátszó négyzetek jelennek meg, ezek közül van egy kibálasztott különleges négyzet, amit Samu entrophy-nak nevezünk. A játék 10 perces, 10 per után leáll magától. A 10 perc leforgása alatt a játékosnak 1 feladata van, mégpedig, hogy a Samu entrophy-n tartsa az egerét lenyomott egérgombbal. Elsőre ez nem is tűnik olyan nehéz feladatnak viszont találkozunk nehezítő tényezőkkel. Ugyanolyan kinézetű négyzetek jelennek meg sorra mint a Samu entrophy, és mindegyikük apró mozgásokat végez. Említettem, hogy félre átlátszóak ezek a négyzetek tehát valamennyire lehet is követni a dolgot, de egy idő után szinte teljesen elveszítjük Samut. Pont emiatt érdekes teszt lehet, hogy a játék során a saját karakterünket és annak körülményeit nem mi irányítjuk. A játék 10 perc, se kevesebb sem több, tehát nem ér véget a játék ha eetleg levszettük Samut, ilyenkor meg kell keressünk! A program ezeket az elhaygyásokat is rögzíti és a végső kimenetében ezeket is számolja a teljesítményünk kiértékelésekor.

A program folyamatosan vizsgálja és frissíti azokat az adatokat amelyek a végeredmény meghatározásához kellenek. Ilyek pl.: Samu pozíciója illetve az egér pozíciója és az ezek közti távolság. Ezen kívül említettem már, hogy ha elhagytuk Samut akkor sincs minden veszve, a forráskódban található egy függvény, az "updateHeroes" itt tartja számon a program, hogy hányszor vesztettük el samu és a megtalálásokat is feljegyzi.

```
void BrainBWin::updateHeroes ( const QImage &image, const ↵
                                int &x, const int &y ) ↵
{
    if ( start && !brainBThread->get_paused() ) {
        int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) + ↵
                   ( this->mouse_y - y ) * ( this->mouse_y - y );
        if ( dist > 121 ) {
            ++nofLost;
            noffound = 0;
            if ( nofLost > 12 ) {
                if ( state == found && firstLost ) {
                    found2lost.push_back ( brainBThread ↵
                                           ->get_bps() );
                }
                firstLost = true;
                state = lost;
                nofLost = 0;
                //qDebug() << "LOST";
            }
        }
    }
}
```

```
//double mean = brainBThread->meanLost();  
//qDebug() << mean;  
  
brainBThread->decComp();  
}  
} else {  
    ++nofFound;  
    nofLost = 0;  
    if ( nofFound > 12 ) {  
  
        if ( state == lost && firstLost ) {  
            lost2found.push_back ( brainBThread->  
                ->get_bps() );  
        }  
  
        state = found;  
        nofFound = 0;  
        //qDebug() << "FOUND";  
        //double mean = brainBThread->meanFound();  
        //qDebug() << mean;  
  
        brainBThread->incComp();  
    }  
}  
  
}  
}  
pixmap = QPixmap::fromImage ( image );  
update();  
}
```

7.5. 8.5 Malmö 19 RF

A félév során volt egy feladat amiben, Bátfai Norbert Tanár Úr adott nekünk egy kódot amivel Steve 19 pipacs felszdeésére volt képes és ezt kellett úgy továbbfejlesztenünk, hogy több virágot tudjon szedni, erről fogom csatolni a videónkat.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

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

Tanulságok, tapasztalatok, magyarázat...

A könyvben már korábban is találkoztunk neurális hálót használó programokkal, viszint ebben a fejezetben piciit messzebb megyünk ezekkel. Azt ugye már egyszer említettem, hog a neurális hálók 3 alap rétegből épülnek fel, ezek a bemeneti, a műveleti és a kimeneti rétegek. A műveleti réteg pedig szintén több rétegből állhat. Ahhoz, hogy "egyszerűen" belepillanthassunk a neurális hálók világába rendelkezésünkre áll a Tensor Flow.

A gépi tanulás nagy számítás igényű folyamat ezért a Tensor Flow lehetővé teszi hogy egy feladatot egyszerre több CPU-val és/vagy GPU-val hajtsunk végre, és itt előbukkan a CUDA is ismét. A CUDA párhuzamos számításokat végző technológiája az NVIDIA-nak napjainkban egyre népszerűbbé látszik válni, és ez pontosan a már említett gépi tanulás miatt történik, ehhez ugyanis annyira jól lehet használni a videókártyákat, hogy az NVIDIA elkezdett nemrégiben kifejezetten mesterséges intelligenciához használatos videókártyákat is forgalmazni, ami a Tesla sorozat, illetve nemrégiben piacra dobtak egy asztali számítógépet is amit kifejezetten gépi tanuláshoz terveztek. Ennek a gépnek fontos megemlíteni, hogy saját NVIDIA operációs rendszere van illetve az NVIDIA fentart egy saját adatbankot a programok tanításához.

A MNIST egy kézzel írt arab számokról álló "adatbank", ez csupán arra szolgál, hogy a programunknak legyen elég bemnti adatait amit feldolgozhasson tanulás során. A Tensor Flow pedig nyílt forráskódú platform gépi tanuláshoz. Nagyon sok YouTube videót is találtam a témaiban és sokan egyébként Windows alatt dolgoznak vele. Nekem meggyült a bajom a Linux Mintemen ezekkel a Tensor Flow-s programokkal, tehát vagy virtuális gépen Ubuntu fogom őkt futtatni vagy Windowson.

Ahhoz hogy tudjuk futtatni a programunknak, mindenképp kell Python3-nak lennie a gépünkön, illetve a **pip3 install tensorflow** parancsal a Tensor Flow-t is telepítenünk kell.

A program egyébként úgy dolgozik, hogy először feldolgoz 60.000 képet, amin kézzel írott számjegyek szerepelnek. Ha ezzel elkészült akkor pedig 10.000 teszt képet dolgoz fel és megállapítja róluk, hogy

ezeken milyen számjegyek szerepelnek majd kapunk egy számot ami megmondja, hogy átlagosan milyen pontosan tudta megálapítani milyen számjegyek vannak a képeken.

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ↵
=====
#
# Norbert Batfai, 27 Nov 2016
# Some modifications and additions to the original code:
# https://github.com/tensorflow/tensorflow/blob/r0.11/tensorflow/examples/ ↵
#   tutorials/mnist/mnist_softmax.py
# See also http://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol
# ↵
=====

"""A very simple MNIST classifier.

See extensive documentation at
http://tensorflow.org/tutorials/mnist/beginners/index.md
"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse

# Import data
from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

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)

    # Create the model
    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

    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])

    # The raw formulation of cross-entropy,
    #
    # tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y))),
    #                 reduction_indices=[1]))
    #
    # can be numerically unstable.
    #
    # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
    # outputs of 'y', and then average across the batch.
    cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        labels=y_, logits=y))
    train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

    sess = tf.InteractiveSession()
    # Train
    tf.initialize_all_variables().run(session=sess)
    print("-- A halozat tanitasa")
    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("-- Pontossag: ", sess.run(accuracy, feed_dict={x: mnist.test}))
```

```
images,
y_: mnist.test.labels)))
print("-----")

print("-- A MNIST 42. tesztképenek felismerése, mutatom a számot, a ←
továbblepéshez 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 hálózat ennek ismeri fel: ", classification[0])
print("-----")

#print("-- A saját kezi 8-asom felismerése, mutatom a számot, a ←
#továbblepéshez csukd be az ablakat")
print("-- A MNIST 11. tesztképenek felismerése, mutatom a számot, a ←
továbblepéshez csukd be az ablakat")
# img = readimg()
# image = img.eval()
# image = image.reshape(28*28)
img = mnist.test.images[11]
image = img
matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm. ←
binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

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

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

if __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()
```

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ez a program, sokban hasonlít az előzőhez, mondhatni a "nagytestvére", tehát itt is megtörténik a 60.000 kép feldolgozása, azaz a tanulási fázis, amikor a MNIST-ből kapott inputokkal dolgozik. Ezután viszont láthatjuk az eltérést, ugyanis nem kezdi el a MNIST teszt képeiről megállapítani azokon milyen számok szerepelnek hanem egy általunk kiválasztott képen mondja meg, hogy azon milyen számot láthatunk. Még egy különbség, hogy ebben a programban rejtegett rétegből 5-öt találunk. A numpy-ra szükségünk lesz a képfeldolgozáshoz, tehát ezt mindekképp telepítsük. Amit említettem, hogy nekem Linus Minten ezt még nem sikerült lefuttatni az még mindig áll, tehát, most nem tudok még konkrét feladatmegoldást csatolni.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten, ←
    MaxPooling2D

from PIL import Image
import numpy as np
import sys

#tf.compat.v1.enable_eager_execution(
#    config=None, device_policy=None, execution_mode=None
#)

#physical_devices = tf.config.experimental.list_physical_devices('GPU')
#assert len(physical_devices) > 0, "Not enough GPU hardware devices ←
#available"
#tf.config.experimental.set_memory_growth(physical_devices[0], True)

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10, activation=tf.nn.softmax))
```

```
#tb_log_dir = "./cnn_tb"
#file_writer = tf.summary.create_file_writer(tb_log_dir)
#file_writer.set_as_default()
#tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_log_dir, ←
#    profile_batch=0)

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

#model.fit(x=x_train, y=y_train, epochs=10, callbacks=[tensorboard_callback ←
#    ])
model.fit(x=x_train, y=y_train, epochs=10)

model.evaluate(x_test, y_test)

input_image = np.array(Image.open(sys.argv[1]).getdata(0).resize((28, 28), ←
    0))

pred = model.predict(input_image.reshape(1, 28, 28, 1))

print(pred)

print("The number is = ", pred.argmax())
```

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A könyvben eddig már sok Minecraft Malmö feladattal találkoztunk, a félév során én is már sokat foglalkoztam vele szóval már tudok róla írni pár szót. A Malmö Projekt egy nyílt forráskódú Minecraft Mod, amit azért alkottak meg, hogy bevezetésként szolgáljon az ágensprogramozásba illetve a mesterséges intelligencia alapjaiba is betekintést nyerhetünk. Ha sikerült leszednünk a Microsoft github repójából, nagyon sok dolgot találunk benne. Fontos megemlíteni, hogy több nyelven is programozhatjuk, ezek közül én a Python-t és C++-t használtam már.

Ha csak kipróbálni szeretnénk, akkor én a Python nyelvet ajánlom, ehhez a prebulldelt verziót elég letöltenünk majd a launcClient.sh állományt futtatva a python kódjainkat egy másik terminálból egyszeerűen elindíthatjuk. Ha viszont mondjuk C++-t is szeretnénk használni én azt javaslom, hogy magunknak build-eljük forrásból ugyanis nekem másképp több hiba jelentkezett illetve több évfolyamtársam is ugyanúgy járt el mint én.

Fontos megjegyezni, hogy ha nincs túl erős gépünk, akkor nem biztos, hogy problémák nélkül fog futni a Malmö, tehát ha tudunk választani akkor eleve erősebb gépre ajánlom telepítni a biztos futás érdekében.

Itt szeretném még azt is megemlíteni, hogy ilyen szempontból kicsit instabil a dolog, többen tapasztaltunk olyat, hogy egy kód a képernyő felvétel során rosszabb teljesítménnyel futott mint amikor nem próbáltuk felvenni, de nem is kell feltétlenül csinálnunk hozzá valami extrát hogy Steve kicsit megmakacsolja magát. Aki ki akarja próbálni ne lepődjön meg rajta, ha ugyanaz a kód ami egyszer hibátlanul lefutott utána más, rosszabb eredményt produkál.

Én egyébként mindenkinél ajánlom, hogy ha tehet és ideje engedi próbálja ki! Szerintem nagyon sok siekrélményt is ad a tanuló programozóknak, ha látják lépésről lépésről, hogyan fut a kódjuk. illetve teljesen más élményt ad mint amikor a lefutott kódunk visszaad egy számot, sokkal "kézzelfoghatóbb" az egész.

8.4. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Itt az előző fejezetben tárgyalt kódunk javításáról lenne szó. Az előző kóddal Steve ugye 19 virágot tudott összeszedni, viszont itt egy javított verzió amivel 20-at tud. Erről kódról és Czanik András, akitől közösen dolgozunk a Malmö feladatokon, készítettünk egy videót amiben látható a kód futása illetve el is magyarázzuk, hogyan és mit változtattunk az eredeti kódon.

A linket még nem rövidítettem le és a benne lévő "and" jel miatt nem amkeli a pdf, de amint lerövítettem beszúrom ide.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

A Lisp nyelvvel én ebben a feladatban találkoztam először. Az általam eddig megismert programnyelvektől lényeges különbözik. De nézzük is meg miért lóg ki annyira a sorból!

A Lisp-et 1958-ban alkotta meg John McCarthy, ekkor vált felkapott témává először a mesterséges intelligencia kutatás és a Lisp nyelv szinte egybe olvadt annak fogalmával. A mesterséges intelligencia és a Lisp nyelv kapcsolata megalkotójához köthető, ugyanis John McCarthy aktívan részt vett az ezzel kapcsolatos kutatásokban is. Annak ellenére, hogy nem mai nyelv, napjainkban is van haszna, a GNU Emacs szövegszerkesztő is használja a Lisp-et.

Ami a Lisp nyelvet illeti, fő adatstruktúrája a láncolt lista, innen kapta a nevét is, angolul "List Processing", innen ered a "Lisp" mint rövidítés. gymásba ágyazott láncolt listákkal dolgozik, primitív szintaktika és egyszerű függvények jellemzik, ugyanis minden kifejezést zárójelek közé kell tennünk. Ennek előnye, hogy a számítógép számára könnyen értelmezhető, interpreter, a Python-hoz hasonló módon nem szükséges a Lisp nyelven írt programokat fordítani, értelmező nyelvről beszélünk. Ez azt jeleti a Lisp esetében, hogy a láncolt listákat először beolvassa majd kiértékeli és kiírat. Elsőre nagyon furcsa lehet a szintaktikája, főleg a prefix jelölést értem ez alatt, ami azt jelenti, hogy egy művelet megfogalmazása esetén az operátor előre kerül majd az operandusok azt követik. Ilyennel én először az elsőrendű logikai nyelvenél találkoztam, azok közül is én a Lisp-et az Ar, azaz az elemi aritmetika nyelvéhez tudom hasonlítani. Az Ar nyelv is alapvetőleg prefix, viszont be tudunk vezetni infix jelöléseket is. Amit pedig még mindenki meg szeretnék említeni a Lisp nyelvvel kapcsolatban, hogy nagyon könnyen fejleszthető, ami alatt azt értem, hogy könnyen hozhatunk létre új, saját függvényeket és operátorokat emiatt "személyre szabható" és így napjainkban sem mondhatjuk rá, hogy elavult nelvről beszélünk. Az Ar nyelv, ahogy említettem, számonra nagyon hasonló, az Ar nyelv is alapvetőleg egy db predikátumszimbólummal dolgozik, ami az egyenlőség fogalma és 3 db függvénszimbólummal, ami a rákövetkező függvény, a szorzás és az öszeadás, ezekből és 1 db konstans használatával az Ar nyelvben a teljes aritmetika felépíthető elemi lépésekre való lebontással, arról nem is beszélve, hogy a zárójelhasználat is nagyon hasonló.

Egy kis kitekintés az Ar nyelvre:

Ar nyelv

$$\langle \{t\}, \{=\}, \{s, +, *\}, \{o\} \rangle$$

$x, y, z \dots \rightarrow$ termékeszeti halmazok változók

$$(x = y) \Leftrightarrow = (x, y)$$

$1 \Leftrightarrow s(o)$	$s(s(o)) * s(s(o)) = s(s(s(o)))$
$2 \Leftrightarrow s(1)$	
$3 \Leftrightarrow s(2)$	$(x \neq y) \Leftrightarrow \neg (x = y)$

kisebb vagy egyenlő mint y :

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

Térjünk rá a faktoriális számító programra! Először is tekintsük meg a faktoriális matematikai háttérét, két képletet alkalmaztunk a programhoz, az első amivel foglalkozni szeretnénk az a rekurzív képlet. Itt az az alapgondolat, hogy $n! = n * (n - 1)!$, tehát, ha írunk egy faktoriális kiszámítására szolgáló függvényt, melynek paramétere n rekurzívan ($n - 1$)-gyel fogjuk meghívni. Szerintem ez volt az egyszerűbb feladat, és megmutatom hozzá a Lisp nyelvű kódot is.

Megjegyzés: mielőtt futtatni szeretnénk a programot, győződjünk meg róla hogy telepítve van a szükséges csomag, amennyiben nincs a

```
sudo apt-get install clisp
```

parancssal tudjuk telepíteni, illetve, ha már megírtuk a kódot, futtatási jogot kell adnunk neki.

```
(defun faktorialis_rekurziv (n)
  (if (= n 0)
      1
      (* n (faktorialis_rekurziv(- n 1)))) )
```

Láthatjuk, hogy a függvénybelül először azt vizsgáljuk, hogy az n egyenlő-e nullával, ennek az a jelentősége, hogy definíció szerint $0!$ az 1, célszerű ezzel kezdenünk. A függvény további részeiben a rekurzív hívást találjuk az előbb említett képlet szerint. Láttam az interneten olyan megoldást is, ahol az $n = 1$ esetére is külön feltételt vizsgáltak, de ebben a programban az teljesen feleslegessé válna a 0 miatt, ugyanis a rekurzív hívás során ha $n = 1$ akkor $(n - 1)!$ -sal szorzódik ami ugye 1 lesz, tehát $1 * 1$ -et kapunk.

A másik megoldás az iteratív képletet használja, ehhez létre kellett hoznunk egy **f** "lokális változót", majd a **dentimes** függvényvel n alkalommal megszorozza a szám rakkövetkezőjével, tehát az iteratív megoldás,

fordítva halad és minden f -et ad vissza.

A faktoriális **függvény** formális definíciója:

$$n! = \prod_{k=1}^n k \quad \text{ minden } n \geq 0 \text{ egész számra.}$$

Iteratív képlet a Wikipédiáról

A program futásáról készült videóm itt található.

```
(defun faktorialis_iterativ (n)
  (let ((f 1))
    (dotimes (i n)
      (setf f (* f (+ i 1))))
    f
  )
)
```

A programomat annyival még kiegészítettem, hogy az enyém bekéri az **n**-t, tehát nem a kódban kell meghatároznunk meddig szeretnénk a faktoriálist számolni.

Mivel a program rövid ezért ide beillesztem az egész forrást:

```
#!/usr/bin/clisp

(format t "Irj be egy szamot!~%")
(setq n (read))

(defun faktorialis_iterativ (n)
  (let ((f 1))
    (dotimes (i n)
      (setf f (* f (+ i 1))))
    f
  )
)

(defun faktorialis_rekurziv (n)
  (if (= n 0)
      1
      (* n (faktorialis_rekurziv(- n 1)))) )

(format t "Rekurziv szamitas:~%")

(loop for i from 0 to n
      do (format t "~D! = ~D~%" i (faktorialis_rekurziv i)))

(format t "Iterativ szamitas:~%")

(loop for i from 0 to n
      do (format t "~D! = ~D~%" i (faktorialis_iterativ i)))
```

A feladat megoldásához használtam [Besenczi Renátó videóját](#) illetve az alábbi oldalakról gyűjtöttem információkat: <http://ait.iit.uni-miskolc.hu/~dudas/MIEAok/MIea5.PDF> <http://nyelvek.inf.elte.hu/leirasok/Lisp/index.php?chapter=1>

9.2. Malmö kód továbbfejlesztése

Ez a program, az előző kettő fejezetben tárgyalt programokon alapszik, viszont itt már sok fejlesztést tettünk bele. Ezzel a kóddal neveztem én és Czanik András csapatban a RFH III. versenyre és a tabellán 16. helyre kerültünk vele. [A kvalifikációs videónk itt található.](#)

Illetve készítettem egy bemutatkozós, kódmagyarázós videót is [Ami itt elérhető.](#)

Steve átlagosan 27 virágot tud összegyűjteni, ez elég volt a kvalifikációhoz. A program lényege hogy Steve gyorsan felszalad a 30. szintre (ami virágszámban 28-at jelentne) és onnan kezdve csigavonalban halad lefelé úgy hogy az aréna fala tőle balra van. Amint virágot érzékel azt felveszi és jobbra ugrással az eggyel lentebbi szintre megy, ennek az a lényege, hogy olyan szinten amelyen már vett fel virágot ne időzzön tovább. Viszont nézzük meg a kódot!

Az előző kódokból átvett lényegi rész a **calcNbrIndex()** függvény. Ennek az a lényege, hogy ugye mint azt már tudjuk Steve egy cuboid-ban lát, viszont amikor elfordul egy sarokban, a cuboid nem fordul együtt vele ezért ugyanazokra a tömbbeli elemekre hivatkozva nem tudjuk elérni, hogy megfelelően reagáljon. Ez a függvény a **yaw** értékből, ami Steve elfordulását hivatott leírni, számolja ki, hogy a cuboid mely eleme lesz az amit keresünk adott elfordulás esetén.

```
def calcNbrIndex(self):
    if self.yaw >= 180-22.5 and self.yaw <= 180+22.5 :
        self.front_of_me_idx = 1
        self.front_of_me_idxr = 2
        self.front_of_me_idxl = 0
        self.right_of_me_idx = 5
        self.left_of_me_idx = 3
    elif self.yaw >= 180+22.5 and self.yaw <= 270-22.5 :
        self.front_of_me_idx = 2
        self.front_of_me_idxr = 5
        self.front_of_me_idxl = 1
        self.right_of_me_idx = 8
        self.left_of_me_idx = 0
    elif self.yaw >= 270-22.5 and self.yaw <= 270+22.5 :
        self.front_of_me_idx = 5
        self.front_of_me_idxr = 8
        self.front_of_me_idxl = 2
        self.right_of_me_idx = 7
        self.left_of_me_idx = 1
    elif self.yaw >= 270+22.5 and self.yaw <= 360-22.5 :
        self.front_of_me_idx = 8
        self.front_of_me_idxr = 7
        self.front_of_me_idxl = 5
```

```
    self.right_of_me_idx = 6
    self.left_of_me_idx = 2
elif self.yaw >= 360-22.5 or self.yaw <= 0+22.5 :
    self.front_of_me_idx = 7
    self.front_of_me_idxr = 6
    self.front_of_me_idxl = 8
    self.right_of_me_idx = 3
    self.left_of_me_idx = 5
elif self.yaw >= 0+22.5 and self.yaw <= 90-22.5 :
    self.front_of_me_idx = 6
    self.front_of_me_idxr = 3
    self.front_of_me_idxl = 7
    self.right_of_me_idx = 0
    self.left_of_me_idx = 8
elif self.yaw >= 90-22.5 and self.yaw <= 90+22.5 :
    self.front_of_me_idx = 3
    self.front_of_me_idxr = 0
    self.front_of_me_idxl = 6
    self.right_of_me_idx = 1
    self.left_of_me_idx = 7
elif self.yaw >= 90+22.5 and self.yaw <= 180-22.5 :
    self.front_of_me_idx = 0
    self.front_of_me_idxr = 1
    self.front_of_me_idxl = 3
    self.right_of_me_idx = 2
    self.left_of_me_idx = 6
else:
    print("There is great disturbance in the Force...")
```

Az alábbi rész a felszaladásért felelős for ciklust tartalmazza, illetve előkészítjük Steve-t a csiga vonalban történő mozgás megkezdésére. Fontos volt hogy beállítsuk a program elején, hogy folyamatosan lefelé nézzen, így könnyebben üti ki a virágokat, nem kell arra várunk, hogy rá is nézzen azokra.

```
i = 0
for i in range(30):
    self.agent_host.sendCommand( "jumpmove 1" )
    time.sleep(0.1)
    self.agent_host.sendCommand( "move 1" )
    time.sleep(0.1)

    self.agent_host.sendCommand( "turn 1" )
    time.sleep(0.2)
    self.agent_host.sendCommand( "look 1" )
    self.agent_host.sendCommand( "look 1" )
```

A program többi része úgy működik, hogy különböző feltételek teljesülését vizsgáljuk, ezeket bemutatom. Az alábbi kódcsipetben Steve a sarkokat figyeli, tudnia kell, hogy mikor forduljon jobbra egy szíten.

```
if nbr[self.front_of_me_idx+9] == "dirt" and nbr[self. ←  
left_of_me_idx+9] == "dirt":  
    self.agent_host.sendCommand( "turn 1" )  
    time.sleep(0.2)  
  
else:  
    print("There is no corner")
```

Tettünk a programba egy lávakerülésre használt részt, amennyiben Steve lávát érzékel, jobbra ugrik kettőt, az aréna belseje felé. Ezt a program futása során ritkán használta és az is a potenciálisan megszerezhető virágok elvesztésével járt, viszont meg tudta úszni a lávával való találkozást.

```
if nbr[self.left_of_me_idx+18]== "flowing_lava" or nbr[self. ←  
front_of_me_idx+9]== "flowing_lava" or nbr[self.front_of_me_idx ←  
+18]== "flowing_lava":  
    self.agent_host.sendCommand( "strafe 1" )  
    time.sleep(0.1)  
    self.agent_host.sendCommand( "strafe 1" )  
    time.sleep(0.1)
```

Az következő részt azért kellet a programba tenni, mert Steve ha virágot talál gyakran aföldet is kiüti alólá és egyben maga alól, "csapdába kerül" ilyenkor ki kell ugrania belőle. Kiegészítettük jobbra ugrásokkal is mivel ilyenkor gyakran egy szinttel lentebb kell folytatnia a mozgását.

```
if nbr[self.front_of_me_idx+9]== "dirt" and nbr[self.right_of_me_idx ←  
+9]== "dirt" and nbr[self.left_of_me_idx+9]== "dirt":  
    print("      IT'S A TRAAAAP")  
    self.agent_host.sendCommand( "jumpmove 1" )  
    time.sleep(0.1)  
  
if nbr[self.front_of_me_idx+9]== "dirt" and nbr[self.left_of_me_idx ←  
+9]== "air":  
    self.agent_host.sendCommand( "turn 1" )  
    time.sleep(0.2)  
    self.agent_host.sendCommand( "jumpstrafe 1" )  
    time.sleep(0.1)  
    self.agent_host.sendCommand( "jumpstrafe 1" )  
    time.sleep(0.1)
```

Elérkeztünk Steve virágszedéséhez, ha virágot talál meghívódik a **pickup()** függvény, azaz kiüti a virágot, itt találunk egy kevés késleltetést ami azért kell, hogy a kiütött virágot fel is tudja szedni, majd kettőt jobbra ugrik, tehát megy a következő szintre.

```
if nbr[self.front_of_me_idx+9]== "red_flower":  
    print("      VIRAGOT SZEDEK!!")  
    #self.agent_host.sendCommand( "move 1" )  
    #time.sleep(0.2)  
    self.pickUp()
```

```
time.sleep(0.2)
self.agent_host.sendCommand( "jumpstrafe 1" )
time.sleep(0.1)
self.agent_host.sendCommand( "jumpstrafe 1" )
time.sleep(0.1)
```

Ezekben a sorokban pedi az általános eseteket találjuk, amikor Steve-nek nincs más dolga mint menni és lépésenként ellenőrzni, hogy mi található a környezetében, hogy megfelően tudjon reagálni. Tehát az alap esetben egyet lép minden előre.

```
if nbr[self.front_of_me_idx+9] == "air" and nbr[self. ←
    front_of_me_idx] == "dirt":
    self.agent_host.sendCommand( "move 1" )
    time.sleep(0.05)

if nbr[self.front_of_me_idx+9] == "air" and nbr[self. ←
    front_of_me_idx] == "air":
    self.agent_host.sendCommand( "move 1" )
    time.sleep(0.05)
```

Ebben a feladatban [Szoboszlai István](#) volt a tutorom, egész pontosan ami a C++ megvalósítást illeti. Mivel a C++ megoldás szinte teljesen ugyanaz mint a Python-os ezért ott nem fogom taglalni a kódot csak beíllsztem ide annak lényegi részeit. És a C++ futásról készült videó pedig [itt](#) található.

```
for (int i = 0; i < 30; i++)
{
    agent_host.sendCommand("jumpmove 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds(100));
    agent_host.sendCommand("move 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds(100));
}
agent_host.sendCommand("turn 1");
boost::this_thread::sleep(boost::posix_time::milliseconds(200));
agent_host.sendCommand("look 1");
agent_host.sendCommand("look 1");

// main loop:
do {

    if(world_state.number_of_observations_since_last_state != 0)
    {
        std::stringstream ss;
        ss << world_state.observations.at(0).get() ->text;
        boost::property_tree::ptree pt;
        boost::property_tree::read_json(ss, pt);

        vector<std::string> nbr3x3;

        nbr3x3 = GetItems(world_state, ss, pt);
```

```
for(vector< boost::shared_ptr< TimestampedString>>::iterator it ←
    = world_state.observations.begin(); it !=world_state. ←
    observations.end(); ++it)
{
    boost::property_tree::ptree pt;
    istringstream iss((*it)->text);
    boost::property_tree::read_json(iss, pt);

    //string x =pt.get<string>("LineOfSight.type");
    //string LookingAt =pt.get<string>("XPos");
    //y = pt.get<double>("YPos");
    yaw = pt.get<double>("Yaw");
    //cout<<" Steve's Coords: "<<y<<" "<<yaw<<" "<<"RF:"<<virag;
}

calcNbrIndex();

//checking corners

if (nbr3x3[front_of_me_idx+9] == "dirt" && nbr3x3[ ←
    left_of_me_idx+9] == "dirt")
{
    agent_host.sendCommand("turn 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds ←
        (300));
}
else
    cout << "\nThere is no corner";

//checking lava
if (nbr3x3[left_of_me_idx+18] == "flowing_lava" || nbr3x3[ ←
    front_of_me_idx+9] == "flowing lava"
    || nbr3x3[front_of_me_idx+18] == "flowing_lava")
{
    agent_host.sendCommand("strafe 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds ←
        (100));
    agent_host.sendCommand("strafe 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds ←
        (100));
}

//checking traps
if (nbr3x3[front_of_me_idx+9] == "dirt" && nbr3x3[ ←
    right_of_me_idx+9] == "dirt"
    && nbr3x3[left_of_me_idx+9] == "dirt")
{
    cout << "\nIt's a TRAAAAP";
    agent_host.sendCommand("jumpmove 1");
    boost::this_thread::sleep(boost::posix_time::milliseconds ←
```

```
        (100));
    }

    if (nbr3x3[front_of_me_idx+9] == "dirt" && nbr3x3[ ←
        left_of_me_idx+9] == "air")
    {
        agent_host.sendCommand("turn 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (200));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (100));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (100));
    }

    //finding flower
    if (nbr3x3[front_of_me_idx+9] == "red_flower")
    {
        cout << "\nVIRÁGOT SZEDEK!!";
        agent_host.sendCommand("attack 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (230));
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (200));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (100));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (100));
    }

    if (nbr3x3[front_of_me_idx+9] == "air" && nbr3x3[ ←
        front_of_me_idx] == "dirt")
    {
        agent_host.sendCommand("move 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (50));
    }

    if (nbr3x3[front_of_me_idx+9] == "air" && nbr3x3[ ←
        front_of_me_idx] == "air")
    {
        agent_host.sendCommand("move 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ←
            (50));
    }
```

}

DRAFT

10. fejezet

Hello, Gutenberg!

10.1. 10.1. Programozási alapfogalmak

A programozási nyelveknek alapvetőleg 3 szintjét tudjuk megkülönböztetni: gépi nyelv, assemly szintű nyelv és a magas programozási nyelvek. A gépi nyelv jelentősége az hogy a CPU-k saját gépi nyelvvel rendelkeznek és csak azon a nyelven írt programokat tudják futtatni. Kezdő programozók tapasztalhatnak olyat ha pl .c kiterjesztésű fájlt próbálnak futtatni hogy szintaktikai hibát dob vissza mondjuk zárójel használatra, pontosan ezért van szükség fordítóprogramokra. Én linuxot használok így mondjuk gcc prog.c -o prog parancsral tudom előállítani a forráskódomból a futtatható tárgyprogramot. Amikor ezt tesszük a fordítóprogram először egy lexikális elemzést hajt végre majd ezt követően szintaktikai és szemantikai elemzést és az utolsó lépésekben kódot generál. Ezzel szemben az interpretens technika az utolsó lépést kihagyja és soronként elemzi és hajtja végre a megadott lépésekkel, ilyen technikával tudunk futtatni Python nyelven írt programokat. Assemly szintű nyelv: ezekről a köny csak említés szintjén ír, ezek a nyelvek a gépi kód és a magas szintű programozási nyelvek közti átmenetet képviselik. Magas szintű programozási nyelvek:

10.2. Keringhan.Ritchie: A C programozási nyelv

Az első Alapismeretek c. fejezetet olvastam el. Mivel én már korábban is olvastam a könyvből, illetve főleg példaprogramokon keresztül mutatja be a programozás alapjait, ezért számomra viszonylag kevés új dolog volt benne főleg említés szintjén számolok be az olvasott témakról.

Először egy klasszikus Hello Word! (A könyvben "Figyelem emberek!") program bemutatásával foglalkozik a könyv és vázolja is mire van szükségünk ahhoz hogy ezt futattni tudjuk, itt megemlíti, hogy valamilyen környezetben írjuk meg a forráskódot, az elengedhetetlen fordítást is említi illetve hogy ki kell találnunk, hogy a szöveget milyen képernyőn szeretnénk kiirva látni. Megjegyzem Linuxban mindez terminálból végrehajthatjuk, pl a nano szövegszerkesztő környezetben illetve a beépitett gcc-ven tudunk is fordítani. Bemutatásra kerül a jól ismert "main()" függvény, amibe a fő programunk kerül.

A következő pontban a változók kerülnek terítékre és azok deklarálása, ugye C-ben meg kell adnunk minden típusú változót szeretnénk létrehozni viszont ekkor még nem szükséges neki kezdőértéket adni mint Pythonban, ilyenkor változó tartalma gyakorlatilag memóriaszemét. A különböző típusú változók (a példában int és float) használatát és a while ciklus működését a köny egy Celsius-Farenheight változóprogramon keresztül mutatja be, illetve a változó típusokra való hivatkozás módját is szemlálteti (printf() függvény

használata). Ugyanezena példán keresztül bemutatja a for ciklus működését és szintaxisát is, hogy az olvasó láthassa ugyanúgy célra vezető lehet midnkét ciklus használata, viszont itt fontos megjegyezni, hogy probléma specifikus mikor melyiket érdemes használni. Ezen kívül megemlíti a könyv a szimbolikus állandókat melyekkel bizonyos objektumokra hivatkozhatunk a "#define" kifejezés segítségével.

A következő pont a karakterek ki és bevitelével foglalkozik a "getchar()" és "putchar()" függvények bemutatásán keresztül. A könyv is kiemeli, hogy ezeket a müveleteket a "printf()" és "scanf()" függvényekkel is megtehetjük, az hogy melyeket használjuk megint probála specifikus. Ebben a bekezdésben szó esik az ASCII karakterkészletről is. A megoldandó példaprogram itta a különböző karakterek megszámlálásáról szól az előző téma körben tárgyalt ciklusok segítségével. A következő példaprogram bemenetre érkező szavak számlálását várja el, a szavakat szóköz választja el és az adott szavak első karakterét kell felismernie a kész programozóknak ez alapján annyi szót számol ahány szókezdő karaktert talál. Ekkor felmerül az elágazás kérdése is. A könyv bemutatja az if/else szintaxisát és az "andandhelye" azaz "és", illetve a "||" azaz "vagy" operátor működését, amik elengedhetetlenek a különböző feltételek pontos megfogalmazásához.

A karakterek téma köré utána könyv a tömböket taglalja, ezeket C-ben szerintem egyszerü kezelní, szerencsére a számosztuk is 0-ról indul mint általában a magasabb szintű programozási nyelveknél megszokott. Fontosnak tartom megjegyezni hogy C-ben a stringek leírására gyakran a karaktertömbök használata a megoldás ami furcsa lehet annak aki korábban mondjuk C++-ban programozott.

Az utolsó téma kör a függvények meghívásával, paraméterinek megadásáról és általános használatukról szól. Le van irva hogyan tudunk visszatérésé értékekkel pontosan dolgozni, egyik függvényét átadni másik függvénynek, általános használatukat a könyv jól szemlélteti.

Összefoglalás: Az Alapismeretek c. fejezet valóban a legelemebb szinten mutatja be a C nyelv használatát. Az olvasás során felmerülő témaik majdnem mindegyikével találkoztam már korábban. Kezdő programozóknak ajánlanám, én is mikor elkezdtem C nyelven programozni ezt a könyvet kezdtettem el olvasgatni és valóban hasznos is volt.

10.3. 10.4. Python bevezetés

A Python programozási nyelv, amely 2018-ra a legnépszerűbb programozási nyelvvé vált, Guido van Rossum nevéhez fűződik aki azt 1990-ben alkotta meg. A kezdő programozóknak a python ideális nyelv számos előnye maitt, viszonylag könnyen elsajátítható, objektum orientált és platformfüggetlen is. A python olyan magas szintű programozási nyelv amely jobban hasonlit a természetes angol nyelvhez mint a többi magas szintű programozási nyelv, tömör, jól olvasható kódokat írhatunk és nincs szükség zárójelezésre sem. Fontos megjegyezni hogy alkalmazásfejlesztéshez is ideális, gyorsabban lehet fele dolgozni mint pl a C/C++ esetében mivel itt kimarad a fordítási fázis. A forráskódot az interpreternek köszönhetően azonnal futathatjuk és lényegesen gyorsabbá válik a programozási folyamat.

SZINTAXIS: Ha már a python nyelvről irok, fontos megemlíteni a szintaxist. A nyelv szintaktikájának egyik legfontosabb tulajdonsága hogy behúzás alapú. Pl.: ha irunk egy for ciklust a ciklusban lévő állítások egy behúzással (tab = 4 szóköz) benteb vannak mint a for ciklus feje. Erre emiatt a nyelv nagyon érzékeny is, ha nem megfelelően vannak a behúzásaink elrendezve egyből kapunk hibákat. A C/C++ nyelvekhez képest, amelyeket én is használtam korábban, talán a legszembetűnőbb különbség a ";" hiánya, illetve a ciklusok és függvények után kapcsos zárójelek helyett használatos ":". Ha kommentelni szeretnénk akkor tudunk soronként a "#" -el vagy több sort 3 db aposztróffal melyeket a komment elejére és végére kell illesztenünk.

A könyvben olvashattam a tipusokról és változókról. Ebben a programozási nyelvben is a szokásos tipusokat tudjuk használni (számok, azon belül egyszek tizedes/lebegőpontos törtek, stringek stb.) viszont jelentős különbség, hogy a változóinkat nem kell deklarálnunk, futás közben az interpreter a egadott kezdőérték alapján felismeri milyen tipusu változóval van dolga. Változók alatt az egyes objektumokra mutató referenciákat értjük. Vannak lokális és globális változók. Ha az adott egy függvényben vesszük fel akkor lokális változóról beszélünk, ha globálissá szeretnén tenni a függvény elején a változó felvételekor használnunk kell a "global" kulcsszót. Különböző változótipusok között szabad a konverzió, tehát tudunk stringől számot képezni stb. Szerencsére a nyelv sok beépített függvényt tartalmaz így a változóinkat könnyen tudjuk kezelni. A tömbök számozása 0-tól kezdődik.

Fontos megemlíteni a nyelv eszközeit. Az első az elágazás, itt is a jól ismert if/elif/else kulccsszavakat használjuk, működését tekintve olyan mint bármely más programozási nyelvben, a szntaxisában pedig a kettőspont használata az ami eltérést mutat más nyelvekhez képest. A ciklusoknál hasonló a helyzet, a while ciklusnál is a feltétel mögötti kettőspontot használjuk a kapcsos zárójelek helyett. Eltérel a for ciklusban van leginkább ugyanis annak 2 fajtájával találkozhatunk. Az első a megszokott for ciklus, szyntaxt illetően eltér de ezen kívül nincs különbség. A második fajta for cílus a "range()" függvényt használja, mely futtatása alatt egy listát ad vissza. A függvényeinkhez a "def" kulcsszót használhatjuk de itt is lányegbeli eltérést nem igazán tapasztalhatunk. Amikor az osztélyokról és objektumokról olvastam láttam magam előtt a "Class Steve:"-et és annak attribútumait. Mivel én korábban nem találkoztam objektum orientált programozással a gyakorlatban ezt még mindig tanulom és próbálom a gyakorlatba beépíteni de mindenkorban hasznos volt olvasnom erről is a könyvben.

10.4. 10.2. Szoftverfejlesztés C++ nyelven

A C++ egy magas szintű programozási nyelv általános célú frílhasználásra. Az első verziója 1983-ban jelent meg. jelentős különbség elődjéhez, a Chez Klpest, hogy a C++ objektum orientált. C-hez nagyon közel áll, egy C++ fordító jó esélyel tudja a C-t is fordítani, hiába vannak benne pl C kulccsszavak vagy éppen C header fájlok. A 2-6. oldalon a C nyelvhez képesti továbbfejlesztésről olvashatunk, meg is említi a könyv hogy az itt bemutatott kódokat már csak C++ fordítóval fordulnak.

Először a függvények ekrülnek megemlítésre. A C nyelvben egy üres paraméterlistájú függvényt tetszőleges számú paraméterrel hivhatunk meg, miközben C++-ban az üres paraméterlistára a "void" kifejezés utal. Ezen kívül jelentős különbség, hogy ha nem diniálunk visszatérési értéket C-ben alapértelmezettként int tipusú visszatérési értéket kapunk vissza miközben C++-ban fordítási hibát. A "main()" függvény estén nem kötelező a return utasítást használni mivel ha a kód sikeresen a fordul annak visszatérési értéke mindenkorban 0 lesz.

A bool tipust szükségszerű kiemelnem. Fontos különbség a két nyelv között, hogy C++-ban van bool tipusú változó amely "true" azaz igaz vagy "false" azaz hamis logikai értékkel rendelkezhet. C nyelvben ezt a logikai értéket int tipussal tudjuk kifejezni ami lehet 0 vagy 1.

A C-ben és a C++-ban is lehet használni több bájtos stringeket, A C++-szal ellentétben ami rednelkezik ezek kezeléséhez szükséges függvényekkel, C-ben ahoz hogy a wchar_t típiust tudjuk használni meg kell hivnunk vagy az stdlib.h, a z stddef.h vagy a wchar.h header fájlokat.

C++-ban mindenhol lehet változókat deklarálni ahol utasítást tudunk megadni. Ez azért hasznos mert ott tudjuk deklarálni ahol használni szeretnénk így átláthatóbb lesz a kódunk és kisebb esélyel feledkezünk meg a változóinkról. Pl.: ha egy for ciklust irunk annak fejében tudjuk deklarálni a ciklusváltozót, ekkor annak csak a ciklus törzsére lesz hatása.

A fordítás, futattás, nyomonkövetésről: Ez a bekezdés a könyvben számomra nem volt túl hasznos mivel itt a fejlesztői környezet fordításra és futtatásra való felhasználása volt kifejeve, ráadásul mint exe fájl Windows alatt. Én általában Linux rendszert használok és az elmentett .c vagy .cpp kiterjesztésű fájljaimat a rendszerbe beépített gcc és g++ fordítókkal tudom fordítani terminálon keresztül, tehát a hibaüzenetek is a terminál képernyőjén jelennek meg. Egy helyen tudom a sikeres fordítás után futtatni a kész futatható állományaimat. Szerintem ezzel a módszerrel fordítani mint ahogy az a könyvben van bemutatva de persze érdemes ismerni minden két módszert.

DRAFT

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

DRAFT

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

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