

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, ÁCs Bátfai, Margaréta	2020. december 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	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?	8
2.3. Változók értékének felcserélése	10
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	12
2.6. Helló, Google!	14
2.7. A Monty Hall probléma	16
2.8. 100 éves a Brun téTEL	18
3. Helló, Chomsky!	23
3.1. Decimálisból unárisba átváltó Turing gép	23
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	24
3.3. Hivatkozási nyelv	26
3.4. Saját lexikális elemző	28
3.5. Leetspeak	29
3.6. A források olvasása	33
3.7. Logikus	34
3.8. Deklaráció	35

4. Helló, Caesar!	39
4.1. double ** háromszögmátrix	39
4.2. C EXOR titkosító	42
4.3. Java EXOR titkosító	45
4.4. C EXOR törő	47
4.5. Neurális OR, AND és EXOR kapu	51
4.6. Hiba-visszaterjesztéses perceptron	55
5. Helló, Mandelbrot!	58
5.1. A Mandelbrot halmaz	58
5.2. A Mandelbrot halmaz a std::complex osztállyal	64
5.3. Biomorfok	68
5.4. A Mandelbrot halmaz CUDA megvalósítása	73
5.5. Mandelbrot nagyító és utazó C++ nyelven	78
5.6. Mandelbrot nagyító és utazó Java nyelven	90
6. Helló, Welch!	103
6.1. Első osztályom	103
6.2. LZW	107
6.3. Fabejárás	112
6.4. Tag a gyökér	115
6.5. Mutató a gyökér	115
6.6. Mozgató szemantika	115
7. Helló, Conway!	123
7.1. Hangyszimulációk	123
7.2. Java életjáték	130
7.3. Qt C++ életjáték	140
7.4. BrainB Benchmark	150
8. Helló, Schwarzenegger!	162
8.1. Szoftmax Py MNIST	162
8.2. Mély MNIST	166
8.3. Minecraft-MALMÖ	172

9. Helló, Chaitin!	173
9.1. Iteratív és rekurzív faktoriális Lisp-ben	173
9.2. Gimp Scheme Script-fu: króm effekt	174
9.3. Gimp Scheme Script-fu: név mandala	182
10. Helló, Gutenberg!	187
10.1. Programozási alapfogalmak	187
10.2. Programozás bevezetés	188
10.3. Programozás	188
III. Második felvonás	190
11. Helló, Berners-Lee!	192
11.1. C++ és Java összehasonlítása	192
11.2. Gyors prototípus-fejlesztés Python és Java nyelven	193
12. Helló, Arroway!	196
12.1. OO szemlélet	196
12.2. Homokatózó	198
12.3. Gagyi	208
12.4. Yoda	210
12.5. Kódolás from scratch	211
12.6. EPAM: Java Object metódusok	214
12.7. EPAM: Eljárásorientál vs Objektumorientált	214
12.8. EPAM: Objektum példányosítás programozási mintákkal	214
13. Helló, Liskov!	215
13.1. Liskov helyettesítés sértése	215
13.2. Szülő-gyerek	218
13.3. Anti OO	219
13.4. <i>deprecated - Hello, Android!</i>	219
13.5. Hello, Android!	220
13.6. <i>Hello, SMNIST for Humans!</i>	220
13.7. Ciklomatikus komplexitás	220
13.8. EPAM: Interfész evolúció Java-ban	221
13.9. EPAM: Liskov féle helyettesíthetőség elve, öröklődés	222
13.10. EPAM: Interfész, Osztály, Absztrakt Osztály	222

14. Helló, Mandelbrot!	224
14.1. Reverse engineering UML osztálydiagram	224
14.2. Forward engineering UML osztálydiagram	225
14.3. Egy esettan	225
14.4. BPMN	227
14.5. <i>BPEL Helló, Világ! - egy visszhang folyamat</i>	227
14.6. TeX UML	227
14.7. EPAM: Neptun tantárgyfelvétel modellezése UML-ben	227
14.8. EPAM: Neptun tantárgyfelvétel UML diagram implementálása	228
14.9. EPAM: OO modellezés	228
15. Helló, Chomsky!	229
15.1. Encoding	229
15.2. OOCWC lexer	229
15.3. <i>l334d1c45</i>	232
15.4. <i>Full screen</i>	234
15.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció	234
15.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció	235
15.7. Perceptron osztály	235
16. Helló, Stroustrup!	242
16.1. JDK osztályok	242
16.2. Másoló-mozgató szemantika	244
16.3. Hibásan implementált RSA törése	246
16.4. Változó argumentumszámú ctor	251
16.5. Összefoglaló	254
16.6. EPAM: It's gone. Or is it?	255
16.7. EPAM: Kind of equal	256
16.8. EPAM: Java GC	256
17. Helló, Gödel!	257
17.1. Gengszterek	257
17.2. C++11 Custom Allocator	258
17.3. STL map érték szerinti rendezése	260
17.4. Alternatív Tabella rendezése	261

17.5. Prolog családfa	263
17.6. GIMP Scheme hack	265
17.7. EPAM: Mátrix szorzás Stream API-val	265
17.8. EPAM: LinkedList vs ArrayList	266
17.9. EPAM: Refactoring	267
18. Helló,...!	268
18.1. FUTURE tevékenység editor	268
18.2. OOCWC Boost ASIO hálózatkezelése	268
18.3. SamuCam	269
18.4. BrainB	269
18.5. OSM térképre rajzolása	270
18.6. EPAM: XML feldolgozás	271
18.7. EPAM: ASCII Art	271
18.8. EPAM: Titkos üzenet, száll a gépben!	271
19. Helló, Lauda!	272
19.1. Port scan	272
19.2. AOP	273
19.3. <i>Android Játék</i>	276
19.4. JUnit teszt	276
19.5. OSCl	277
19.6. <i>OSCI2</i>	277
19.7. <i>OSCI3</i>	277
19.8. EPAM: DI	278
19.9. EPAM: JSON szerializáció	278
19.10EPAM: Kivételkezelés	278
20. Helló, Calvin!	279
20.1. MNIST	279
20.2. Deep MNIST	280
20.3. <i>CIFAR-10</i>	283
20.4. Android telefonra a TF objektum detektálója	283
20.5. SMNIST for Machines	283
20.6. Minecraft MALMO-s példa	284
20.7. EPAM: Reaktív programozás	285
20.8. EPAM: Back To The Future	285
20.9. EPAM: AOP	286

IV. Irodalomjegyzék	287
20.10Általános	288
20.11C	288
20.12C++	288
20.13Lisp	288

DRAFT

Ábrák jegyzéke

2.1. A B_2 konstans közelítése	22
4.1. A double ** háromszögmátrix a memóriában	41
5.1. A Mandelbrot halmaz a komplex síkon	58
6.1. C++ Run	105
6.2. Java Run	107
6.3. Futtatás	112
6.4. Futtatás	113
6.5. Futtatás	114
7.1. Terminálba való futtatás	124
7.2. UML ábra	125
7.3. Hangya szimuláció kapcsolók nélkül	129
7.4. Hangya szimuláció a recommended kapcsolókkal	130
7.5. Terminál futtatás 1	138
7.6. Terminál futtatás 2	139
7.7. Program futás közben	140
7.8. Terminál	149
7.9. ÉLetjáték	150
7.10. Futtatás	160
7.11. Gameplay	160
7.12. Eredmény txt	161
8.1. Futtatás	164
8.2. Eredmény	165
8.3. Terhelés	166
8.4.	171

8.5.	171
8.6.	Terhelés	172
9.1.	Script Fu futtatás	174
9.2.	Chrome írás	178
9.3.	Chrome border	181
9.4.	Mandala	186

DRAFT

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

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...

Egy mag 100 százalékban:

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

    return 0;
}
```

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

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

    return 0;
}
```

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 "infny-f.c"
```

Egy mag 0 százalékban:

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

    return 0;
}
```

Minden mag 100 százalékban:

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

A `gcc infny-f.c -o infny-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	infty-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:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

A feladatnak nincs megoldása mert: ha a program tartalmaz végtelen ciklus akkor lefagy = true és akkor lefagy2 is true lesz ha meg a programban nincs végtelen ciklus akkor lefagy = false és akkor lefagy 2 egy végtelen ciklussá alakul át.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ha fel szeretnénk két változónak értékét cserálni, akkor a legegyszerűbb megoldásnak egy segédváltozó bevezetése tűnhet. De ennél sokkal jobb módszerek vannak. Az egyik ilyen megoldás az hogy változó értékét összeadjuk majd ebből az összegből kivonjuk a régi változók értékét.

```
int a = 1;
int b = 2;
a = a+b;
b = a-b;
a = a-b;
```

De nem csak matematikai módszerekkel tudunk két változót felcseréni hanem exorral is. Ennek a tipusnak annyi lenne a lényege hogy a számokat kettes számrendszerben tároljuk vagyis 0-ból és 1-esekből áll. Xor művelet minden esetben 1-et ad vissza kivéve ha a két változónknak az értéke megegyezik.

```
int a = 1;
int b = 2;
a = a^b;
b = a^b;
a = a^b;
```

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés násználata 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:

Tanulságok, tapasztalatok, magyarázat...

A labdapattogtatás gyakorlatilag annyiból áll hogy a terminálban egy karakter pattog az ablak jelenlegi méretében. Az initscr feladata az hogy beolvassa az ablak méretét. Ezután létrehozzuk változókat hogy eltároljuk a labda helyzetét x és y, lépésközöt a xnov és ynov-ban, ablak mérete mx és my. Mivel initscr-vel olvassuk be az ablak méretét így ha pattogás közbe változtatunk az ablak nagyságán azt érzékelni fogja a program.

```
WINDOW *ablak;
ablak = initscr ();
```

```
int x = 0;
int y = 0;

int xnov = 1;
int ynov = 1;

int mx;
int my;
```

Miután végeztünk a változók létrehozásával.Csinálunk egy végtelciklust ebbe fogjuk pattogtatni a labdát.Ehhez használnunk kell getmaxy() függvényt melynek átadjuk az ablakban eltárolt értékeket.A mwprintf() függvény fogja a "labdánkat" a megfelelő kordinátákba pattogtatni.Most már tudjuk az ablak tulajdonságait.Az x és y változó folyamatosan növekedni fog +1-el és így érjük el a pattogást a terminálban.A labda mozgásának a sebességét a usleep() függvénnyel tudjuk megadni minnél nagyobb annál lassabb.Ám nem kell félni a nagyobb számoktól mert a usleep mikroszekundumba számol tehát ha azt akarjuk hogy a labdánk 1/s sebességet menjen egymilliót kell megadni.

```
for ( ; ; ) {

    getmaxyx ( ablak, my , mx );

    mvprintw ( y, x, "O" );

    refresh ();
    usleep ( 100000 );
```

Ennél a résznél fog különdözni az if-es és a nem ifes megoldás.Elsőnek az if-est nézve

```
if ( x>=mx-1 ) { // jobb oldalt ellenőrzi
    xnov = xnov * -1;
}
if ( x<=0 ) { // bal oldalt ellenőrzi
    xnov = xnov * -1;
}
if ( y<=0 ) { // tetejét ellenőrzi
    ynov = ynov * -1;
}
if ( y>=my-1 ) { // alját ellenőrzi
    ynov = ynov * -1;
}
```

Tehát ha valamelyik érték eléri a határát beszorozzuk minusz egyel és így elérjük hogy visszafele pattogjon.A nem if-es megoldásnál maradékos osztást használunk hogy a labda egy bizonyos értéknél visszapattanjon.Ennek a megoldásnak annyi lenne a lényege hogy a szám értékét fogja visszaadni amit osztunk addig amíg a két számnak az értéke vagyis az ablak szélességevel és magaságával.

```
getmaxyx(ablak, my, mx);
xj = (xj - 1) % mx;
xk = (xk + 1) % mx;
yj = (yj - 1) % my;
```

```
yk = (yk + 1) % my;
//clear ();
mvprintw (abs (yj + (my - yk)),
abs (xj + (mx - xk)), "X");
refresh ();
usleep (100000);
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU, <https://youtu.be/KRZlt1ZJ3qk>.

Megoldás forrása: bhaxi.com/thematic_tutorials/bhaxi_textbook_IgyNeveldaProgramozod/Turing/bogomips.c

Tanulságok, tapasztalatok, magyarázat...

```
#include <stdio.h>

int main()
{
    int word = 1; //kettes számrendszerben: 00000000 ←
                  00000000 00000000 00000001
    int length = 0;

    do
    {
        length++;
    }
    while (word<<=1); //itt léptetjük eggyel:00000000 ←
                      00000000 00000000 00000010
                      // újra: 00000000 00000000 00000000 ←
                      00000100
                      //...

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

    return 0;
}
```

A típus méretét, illetve hogy hány bitet foglal, bitshifteléssel könnyen meghatározhatjuk. A while ciklusunkban minden shiftelünk egyet balra a biteken és addig növeljük a length változót, amíg csupa 0 bitet nem fog tartalmazni a word változónk.

Ahogy a feladatban már említve lett a BogoMIPS hát akkor ő következik. Azért foglalkozunk ezzel, mert ennek a while fejléce ugyan azt a módszert használja amit az előbb csináltunk.

```
while (loops_per_sec <= 1)
{
    ;
}
```

Először deklarálunk 2 változót, az első a `loops_per_sec`, melynek kezdetkor az értékét egyre állítjuk. A bitshiftelés hatására ebbe 2 hatványokat fogunk tárolni. A `ticks` pedig a CPU időt fogja tárolni.

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

    ...
}
```

A while ciklus addig tart, ameddig a `loops_per_sec` le nem nullázódik. A ciklusba belépve, minden iterációban, az aktuális CPU időt, eltároljuk a `ticks` változóban. Ezután pedig meghívjuk a `delay` függvényt.

```
void delay (unsigned long long loops)
{
    unsigned long long i;
    for ( i=0; i<loops; i++);
}
```

Ez a függvény egy hosszú egész számot kér paraméterként, ezek után a for ciklus megy végig 0-tól paraméter-1-ig. Utána a while ciklusban újra lekérjük az aktuális processzor időt és kivonjuk belőle a kezdeti étéket. Így megtudjuk, hogy mennyi ideig tartott a cpu-nak befejezni a `delay` függvényben lévő for ciklust. Ezt egészen addig ismételjük, ameddig nem teljesül az if-ben lévő feltétel.

```
while (loops_per_sec <= 1 )
{
    ...

    if (ticks >= CLOCKS_PER_SEC)
    {
        loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC; // ←
        loops_per_sec/ticks = ???/CLOCKS_PER_SEC

        printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec/500000,
               (loops_per_sec/5000) % 100);
        return 0;
    }
}
```

A CLOCKS_PER_SEC szabvány értéke 1.000.000, vagyis akkor teljesül az if-ben lévő feltétel ha a szabvány értéket eléri vagy meghaladja a processzor idő. Majd kiszámoljuk mennyi CLOCKS_PER_SEC idő alatt milyen hosszú ciklust képes végrahajtani a gép. A eredmény megadásához használhatjuk a log függvényt a következő módosítások után.

```
printf("ok - %lld %f BogoMIPS\n", loops_per_sec, log( ←
    loops_per_sec));
```

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A PageRank képlete: $PR(A) = (1-d) + d \left(\frac{PR(T_1)}{C(T_1)} + \dots + \frac{PR(T_n)}{C(T_n)} \right)$ A: az aktuális oldal T: oldalak amik az a-ra hoz vannak linkelve PR(A): a oldal pagerank száma PR(Tn): azok oldalak pagerank száma amik az a-ra mutatnak C(Tn): t oldalon az összes link ami a-ra mutatnak d: 0 és 1 közé esik

Elsőnek is a kapcsolatokat a mátrixba tároljuk. Sorok és oszlopok metszetébe láthatjuk milyen kapcsolat van az oldalak között. Ezt a mátrixot adjuk a pagerank() -nak függvénynek. A tömbben tároljuk az oldalak értékét, és a PR-ben tároljuk el a mátrixszorzás eredményét. A mátrixszorzást a L és PRv tömbökkel hajtjuk végre. Ezek a tömbök összeszorzásával kapunk egy 4x1 oszlopvektort ami a 4 oldalunk pagerankja lesz.

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

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

    int i;

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

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

    int i;
    double osszeg=0;

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

    return sqrt(osszeg);
}
```

```
}

void
pagerank(double T[4][4]){
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
    double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0 };

    int i, j;

    for(;;) {

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

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

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

    kiir (PR, 4);
}

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

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

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

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

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

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

printf("\n");

return 0;
}
```

```
laci@laci-GL503VD: ~/programok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
collect2: error: ld returned 1 exit status
laci@laci-GL503VD:~/programok$ ^C
laci@laci-GL503VD:~/programok$ gcc pagerank.c -o pagerank -lm
laci@laci-GL503VD:~/programok$ ./pagerank

Az eredeti mátrix értékeivel történő futás:
0.090909
0.545455
0.272727
0.090909

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

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

laci@laci-GL503VD:~/programok$
```

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall probléma alapja három ajtó, mely mindegyike rejt valamit: két ajtó mögött egy-egy roncs autó van, egy ajtó mögött pedig egy vadiúj autó. Ez annyit jelent, hogy két ajtó értéktelen és egy ajtó pedig értékes ajándékot rejt. Azért, hogy könnyebben tudunk beszélni az egészről, mondjuk azt, hogy két ajtó mögött nincs és egy ajtó alatt pedig van ajándék.

```
# An illustration written in R for the Monty Hall Problem
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
# https://bhaxor.blog.hu/2019/01/03/ ←
# erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan
#
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {
  if(kiserlet[i]==jatekos[i]){
    mibol=setdiff(c(1,2,3), kiserlet[i])
  }else{
    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))
  }
  musorvezeto[i] = mibol[sample(1:length(mibol),1)]
}

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

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

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
> nemvaltoztatesnyer= which(kiserlet==jatekos)  
> valtoztat=vector(length = kiserletek_szama)  
>  
> for (i in 1:kiserletek_szama) {  
+  
+    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
+    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]  
+  
+}  
>  
> valtoztatesnyer = which(kiserlet==valtoztat)  
>  
>  
> sprintf("Kiserletek szama: %i", kiserletek_szama)  
[1] "Kiserletek szama: 10000000"  
> length(nemvaltoztatesnyer)  
[1] 3333913  
> length(valtoztatesnyer)  
[1] 6666087  
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)  
[1] 0.5001304  
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)  
[1] 10000000  
>
```

2.8. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

A természetes számok építőelemei a prímszámok. Abban az értelemben, hogy minden természetes szám előállítható prímszámok szorzataként. Például $12=2\cdot2\cdot3$, vagy például $33=3\cdot11$.

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

Két egymást követő páratlan prím között a legkisebb távolság a 2, a legnagyobb távolság viszont bármilyen nagy lehet! Ez utóbbit könnyű bebizonyítani. Legyen n egy tetszőlegesen nagy szám. Akkor szorozzuk össze $n+1$ -ig a számokat, azaz számoljuk ki az $1\cdot2\cdot3\cdots\cdot(n-1)\cdot n\cdot(n+1)$ szorzatot, aminek a neve $(n+1)$ faktoriális, jele $(n+1)!$.

Majd vizsgáljuk meg az a sorozatot:

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

- $(n+1)!+2=1\cdot2\cdot3\cdots\cdot(n-1)\cdot n\cdot(n+1)+2$, azaz $2^{\text{valamennyi}}+2$, 2 többszöröse, így ami osztható kettővel
- $(n+1)!+3=1\cdot2\cdot3\cdots\cdot(n-1)\cdot n\cdot(n+1)+3$, azaz $3^{\text{valamennyi}}+3$, ami osztható hárommal
- ...
- $(n+1)!+(n-1)=1\cdot2\cdot3\cdots\cdot(n-1)\cdot n\cdot(n+1)+(n-1)$, azaz $(n-1)^{\text{valamennyi}}+(n-1)$, ami osztható $(n-1)$ -el
- $(n+1)!+n=1\cdot2\cdot3\cdots\cdot(n-1)\cdot n\cdot(n+1)+n$, azaz $n^{\text{valamennyi}}+n$, ami osztható n -el
- $(n+1)!+(n+1)=1\cdot2\cdot3\cdots\cdot(n-1)\cdot n\cdot(n+1)+(n-1)$, azaz $(n+1)^{\text{valamennyi}}+(n+1)$, ami osztható $(n+1)$ -el

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

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

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

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

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
```

```
#  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU General Public License for more details.  
#  
# You should have received a copy of the GNU General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>  
  
library(matlab)  
  
stp <- function(x) {  
  
    primes = primes(x)  
    diff = primes[2:length(primes)]-primes[1:length(primes)-1]  
    idx = which(diff==2)  
    t1primes = primes[idx]  
    t2primes = primes[idx]+2  
    rt1plust2 = 1/t1primes+1/t2primes  
    return(sum(rt1plust2))  
}  
  
x=seq(13, 1000000, by=10000)  
y=sapply(x, FUN = stp)  
plot(x,y,type="b")
```

Soronként értelemezzük ezt a programot:

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

Kiszámolja a megadott száigmig a prímeket.

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

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

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

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

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

```
[1] 2 3 5
```

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

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

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

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

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

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

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

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

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

```
> 1/t1primes+1/t2primes  
[1] 0.5333333 0.3428571 0.1678322
```

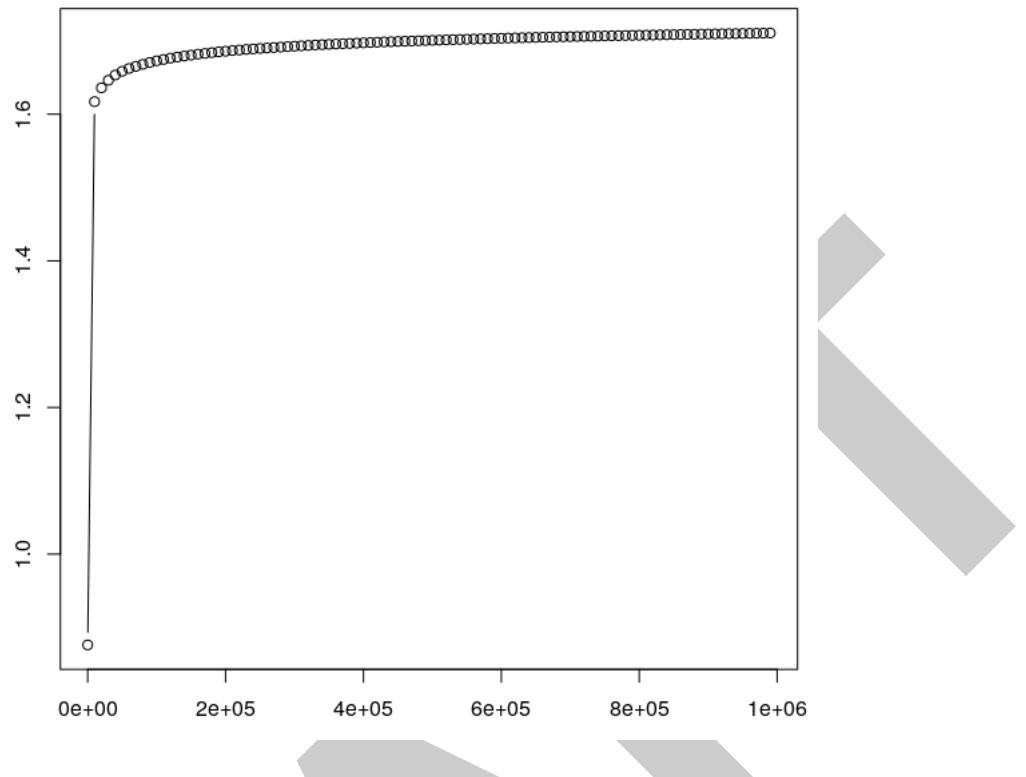
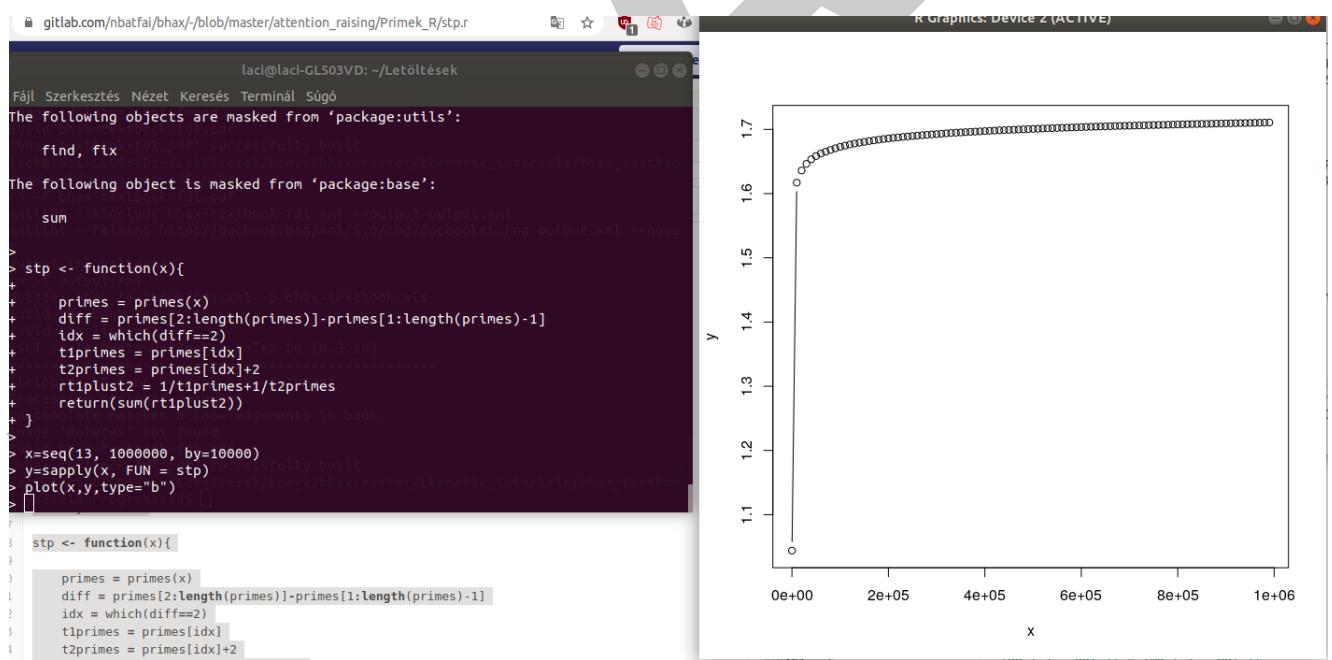
Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvényel.

```
sum(rt1plust2)
```

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

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

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

2.1. ábra. A B_2 konstans közelítése

Werkfilm

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

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: az első előadás [27 fólia](#).

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!

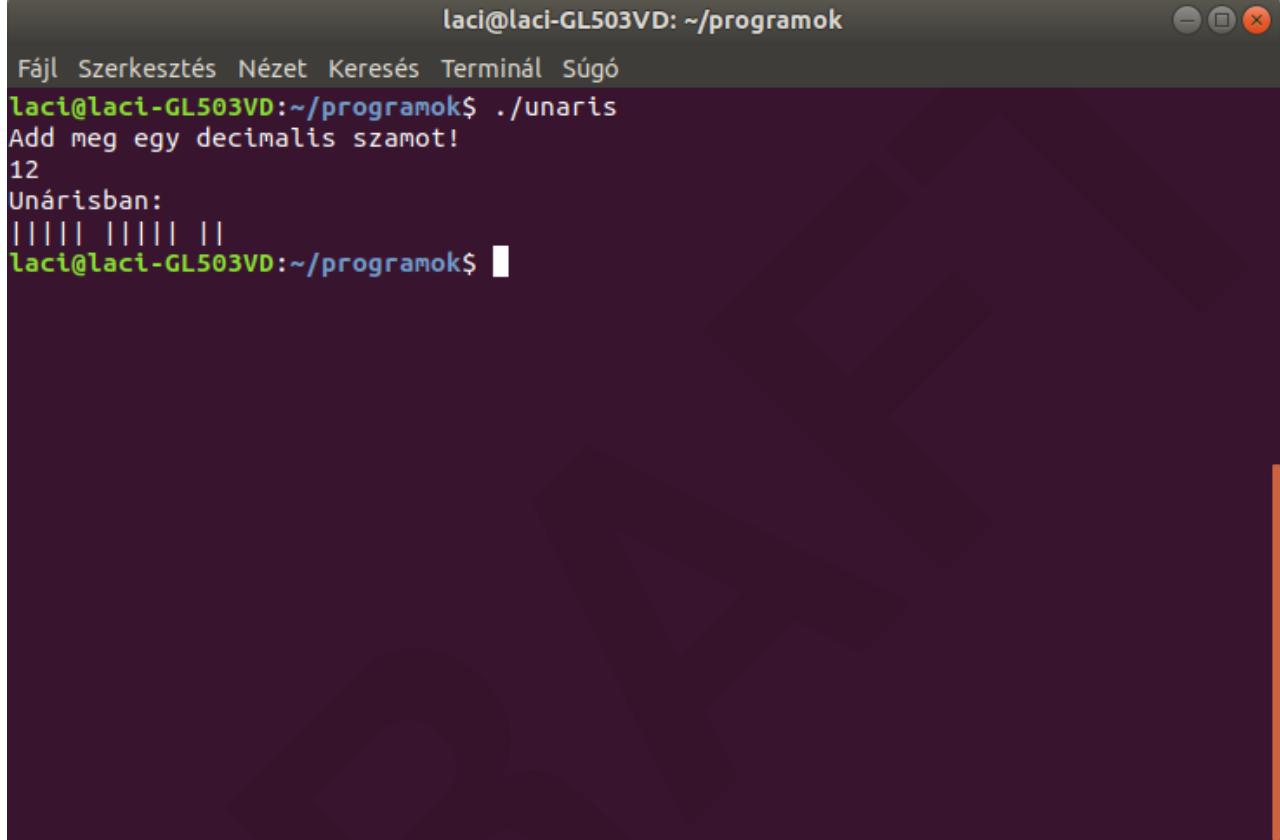
A unáris számrendszer a legegyszerűbb az összes számrendszer közül. Gyakorlatilag ugyanaz, mintha az ujjaink segítségével számolnánk, tehát a számokat vonalakkal jelöljük. A szám ábrázolása pont annyi darab vonalból áll, amennyi a szám. A könnyebb olvashatóság érdekében minden ötödik után rakhunk helyközt vagy bármilyen karaktert amit tetszik.

A forrásként megadott program lényegében egy átváltó, mely függőleges vonalakat ír ki a bemenettől függően. Mivel ez egy C++ program, ezért ennek a fordításához a g++-t érdemes használni, a szintaxisa teljesen megegyezik a gcc-nél megszokottakkal. Ha futtatjuk ezt kell látnunk.

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

int main()
{
    int b;
    int szamlalo = 0;
    cout<<"Add meg egy decimalis szamot!\n";
    cin >> b;
    cout<<"Unárisban:\n";
    for (int i = 0; i < b; ++i)
    {
        cout<<"1";
        ++szamlalo;
        if (szamlalo % 5 == 0)
            cout<< " ";
```

```
    }
    cout<<' \n';
    return 0;
}
```



```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ ./unaris
Add meg egy decimalis szamot!
12
Unárisban:
| | | | | | | |
laci@laci-GL503VD:~/programok$
```

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

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

Megoldás videó:

Megoldás forrása: az első előadás [30-32 fólia](#).

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!

A generatív grammatika Noam Chomsky nevéhez kötődik. Úgy vizsgálja a nyelvtant mint az ismeret alapját, "hiszen ha nem lenne nyelvtanunk, akkor a tudást nem tudnánk se megörökíteni, se továbbadni. Nézete azt a vállotta, hogy a tudás és az ismeret többnyire öröklött (generációról generációra terjed), vagyis univerzális (gondolván a gyerekekre, akik könnyedén elsajátítják anyanyelvüket). A generatív grammatikának négy fontos része van: nemterminális jelek, terminális jelek, helyettesítési/képzési szabályok és mondat/kezdo szimbólumok, illetve három nyelvtan fajtája: környezetfüggetlen és reguláris. Nézzünk meg két példát környezetfüggetlen leírásra (a nyilak jelölik majd a képzési szabályokat)

1. példa

S, X, Y "változók"

a, b, c "konstansok"

S → abc, S → aXbc, Xb → bX, Xc → Ybcc, bY → Yb, aY → aaX, aY → aa
S-ből indulunk ki

S (S → aXbc)

aXbc (Xb → bX)

abXc (Xc → Ybcc)

abYbcc (bY → Yb)

aYbbcc (aY → aa)

aabbcc

S (S → aXbc)

aXbc (Xb → bX)

abXc (Xc → Ybcc)

abYbcc (bY → Yb)

aYbbcc (aY → aaX)

aaXbbcc (Xb → bX)

aabXbcc (Xb → bX)

aabbXcc (Xc → Ybcc)

aabbYbcc (bY → Yb)

aabYbbccc (bY → Yb)

aaYbbbccc (aY → aa)

aaabbccc

2. példa

A, B, C "változók"

a, b, c "konstansok"

A → aAB, A → aC, CB → bCc, cB → Bc, C → bc

S-ből indulunk ki

A (A → aAB)

aAB (A → aC)

aaCB (CB → bCc)

aabCc (C → bc)

aabbcc

A (A → aAB)

aAB (A → aAB)

aaABB (A → aAB)

aaaABBB (A → aC)

aaaaACBBB (CB → bCc)

```
aaaabCcBB (cB -> Bc)
aaaabCBcB (cB -> Bc)
aaaabCBBc (CB -> bCc)
aaaabbCcBc (cB -> Bc)
aaaabbCBcc (CB -> bCc)
aaaabbbCccc (C -> bc)
aaaabbbbcccc
```

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.

Megoldás videó:

Megoldás forrása: az első előadás [63-65 fólia](#).

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!

A C utasítás fogalma BNF-ben való definiálása:

```
<utasítás> ::= <kifejezés> | <összetett_utasítás> | <feltételes_utasítás> | ←
    <while_utasítás> | <do_utasítás> | <for_utasítás> | <switch_utasítás> | ←
    <break_utasítás> | <continue_utasítás> | <return_utasítás> | <←
    goto_utasítás> | <cimke_utasítás> | <>nulla_utasítás>

<kifejezés> ::= <értékadás> | <függvényhívás>
<értékadás> ::= <változó><szám>
<változó> ::= <betű>{<betű>}
<betű> ::= a-z
<szám> ::= <számjegy>{<számjegy>}
<számjegy> ::= 0|1|2|3|4|5|6|7|8|9
<függvényhívás> ::= <típus><függvénynév>
<típus> ::= <betű>{<betű>}
<függvénynév> ::= <betű>{<betű>}

<összetett_utasítás> ::= <deklarációlista> | <utasításlista>
<deklarációlista> ::= <deklaráció>{<deklaráció>}
<deklaráció> ::= <típus><változó>
<utasításlista> ::= <utasítás>{<utasítás>

<feltételes_utasítás> ::= if<kifejezés><utasítás> | if<kifejezés><utasítás> ←
    else<utasítás>

<break_utasítás> ::= break
```

```
<while_utasítás> ::= while<kifejezés><utasítás> | while<kifejezés><utasítás <-
    ><break_utasítás>

<do_utasítás> ::= do<utasítás>while<kifejezés> | do<utasítás>while< <-
    kifejezés><break_utasítás>

<for_utasítás> ::= for([<kifejezés>] [<kifejezés>] [<kifejezés>])<utasítás> | <-
    for([<kifejezés>] [<kifejezés>] [<kifejezés>])<utasítás><break_utasítás>

<switch_utasítás> ::= switch<kifejezés><utasítás> | switch<kifejezés>< <-
    utasítás><case><kifejezés><default> | switch<kifejezés><utasítás>< <-
    break_utasítás>

<continue_utasítás> ::= continue

<return_utasítás> ::= return | return<kifejezés>

<goto_utasítás> ::= goto<azonosító>

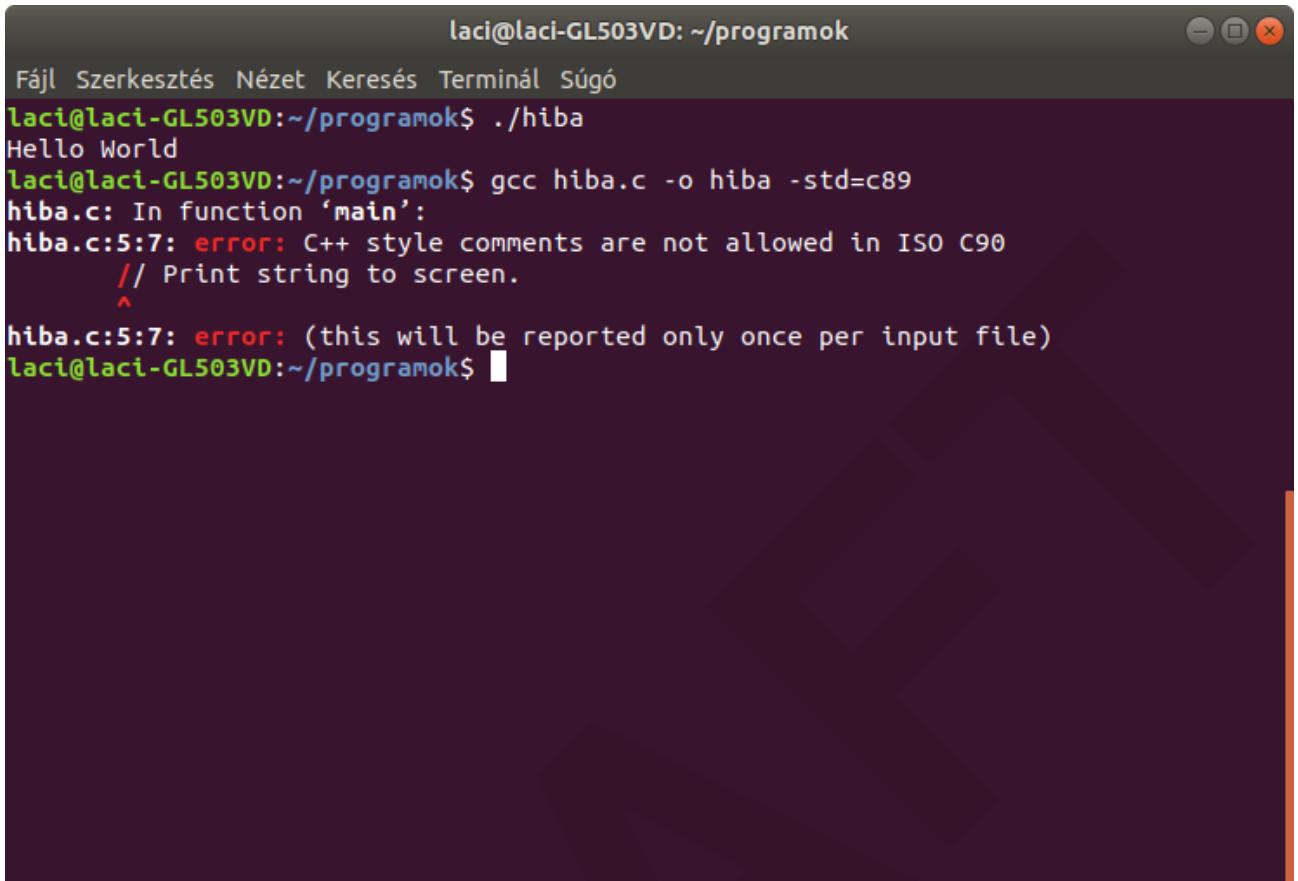
<cimke_utasítás> ::= <azonosító>
<azonosító> ::= <cimke>
<cimke> ::= <betű>{<betű>}

<>nulla_utasítás> ::=
```

Program nyelveket nem hiába nevezzük "nyelvnek" mert olyan mint egy normális nyelv fejlődik új funkciókat tulajdonságokat kap mint ebbe a feladatban is.c89 és a c99 verziók is ilyenek.Ez a példaprogram azért nem fut le c89-be mert c99-ben implementálták a kommenteket így c89 nem ismeri fel így hátról hibaüzenetet dob be.

```
[  
#include <stdio.h>  
  
int main ()  
{  
    // Print string to screen.  
    printf ("Hello World\n");  
    return 0;  
}
```

```
for (int i = 0; i < 10; i++) {  
    x=i*i;  
    printf("%d", x);  
}
```



The screenshot shows a terminal window titled "laci@laci-GL503VD: ~/programok". The terminal displays the following command and its output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ ./hiba
Hello World
laci@laci-GL503VD:~/programok$ gcc hiba.c -o hiba -std=c89
hiba.c: In function 'main':
hiba.c:5:7: error: C++ style comments are not allowed in ISO C90
    // Print string to screen.
          ^
hiba.c:5:7: error: (this will be reported only once per input file)
laci@laci-GL503VD:~/programok$
```

3.4. Saját lexikális elemző

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

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

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

Tanulságok, tapasztalatok, magyarázat...

A forrásból kapott program egy lex program melyel egy lexiális elemzőt lehet készíteni. Szóvegfájlokkal olvassa be a lexikális szabályokat, és egy c forráskódot készít melyet gcc-vel tudunk fordítani. A mi lex forráskódunk 3 részből áll ezeket a részeket %%-al vannak elválasztva egymástól. Az első az a definíciós rész ahol bármilyen c-s forrást lehet használni itt lehet a header-et meghívni. A kapcsos zároljelekbe levő kódot ahogy beírtuk úgy másolja be a c programunkba. Az elején behívjuk a szokásos include-dal deklaráljuk a studio header fájlt ezután létrehozunk egy realnumber nevezetű int-et melyet majd arra fogjuk használni hogy tárolja hány számot olvas be a program. Ezek után jönnek a definíciók és a mi helyzetünkben a digit nevű definícióval a szöglletes zárójelben való 0-9 ig való számokat vesszük.

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

```
%}  
digit [0-9]
```

A második részünk a fordítási szabályoknak van fenntartva. Ez 2 részből áll reguláris kifejezésekkel és az azokhoz tartozó c utasításokhoz.

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

A harmadik rész az már a tényleges program, ahol meghívjuk az előbb létrehozott yylex függvényt.

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

The screenshot shows a terminal window titled "laci@laci-GL503VD: ~/programok". The terminal output is as follows:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
laci@laci-GL503VD:~/programok$ lex -o realnumber.c realnumber.l  
laci@laci-GL503VD:~/programok$ gcc realnumber.c -o realnumber -lfl  
laci@laci-GL503VD:~/programok$ ./realnumber  
dsdasd323fdsfsd  
dsdasd[realnum=323 323.000000]fdsfsd  
dsds112ffffbvbvf  
dsds[realnum=112 112.000000]ffffbvbvf  
dasdasdaddmkdmvfkfmfvme3dgmkd1fkmdfkdmf6  
dasdasdaddmkdmvfkfmfvme[realnum=3 3.000000]dgmkd[realnum=1 1.000000]fkmdfkdmf[real  
num=6 6.000000]  
^C  
laci@laci-GL503VD:~/programok$
```

3.5. Leetspeak

Lexelj össze egy l33t cipher-t!

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

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

Mi előtt átnéznénk a forráskódot találjuk ki hogy mi is az a leet nyelv.A leet nyelv arre lett létrehozva hogy a szavakban lévő betűket számokra vagy speciális karakterekre cseréljék le. Az 3.4-es feladatban megnéztük hogy hogy épül fel egy lexer és hogy hogy lehet használni. A program elején mint az előző feladatban importálva vannak a header fájlok.

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

A második részben találkozunk egy #define-al amivel azt tudjuk elérni ha a programban valahol hivatkozunk a l337size-ra akkor mellette lévő értékkel fogja helyettesíteni.A következő részben láthatjuk a cipher struktúrát amely egy char-c-ből és egy 4 elemű tömbre mutató char *-ből áll.Ezek alapján hozzuk létre a l337d1c7 [] tömböt .Ez a tömb tárolja az egyes betűket és a hozzá tartozó helyettesítő karaktereket.

```
% {
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))

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

{'a', {"4", "4", "@", "/-\\"}}, {'b', {"b", "8", "13", "|{}"}}, {'c', {"c", "(", "<", "{}"}}, {'d', {"d", "|)", "[", "|{}"}}, {'e', {"3", "3", "3", "3"}}, {'f', {"f", "|=", "ph", "|#"}}, {'g', {"g", "6", "[", "+"}}, {'h', {"h", "4", "|-", "[-"]}}, {'i', {"1", "1", "|", "!"}}, {'j', {"j", "7", "_|", "_/"}}, {'k', {"k", "|<", "1<", "|{"}}, {'l', {"l", "1", "|", "|_"}}, {'m', {"m", "44", "(V)", "|\\//|"}}, {'n', {"n", "|\\|", "/\\/", "/V"}}, {'o', {"0", "0", "()", "[]"}}, {'p', {"p", "/o", "|D", "|o"}}, {'q', {"q", "9", "O_", "(,)"}}, {'r', {"r", "12", "12", "|2"}}, {'s', {"s", "5", "$", "$"}}, {'t', {"t", "7", "7", "'|/'"}}, {'u', {"u", "|_|", "(_)", "[_]"}}, {'v', {"v", "\\\/", "\\\/", "\\\/"}}}, {'w', {"w", "VV", "\\\/\\", "(/\\)"}},
```

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

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

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

%}
```

A harmadik részben a for ciklusunkba lekérjük a l337size értékét ami a l337dlc7[] tömb oszta a chiper struktúra méretével. Mivel 1337dlc7[] egy struktúrált tömb ezért 1337dlc7[i].c ként tudunk hivatkozni a tömb egy bizonyos elemére. A tolower az átalakítja a nagybetűs bemenetet kisbetűkre. Ezek után pörgetünk egy random számot 1-100 között. Ettől a számtól fog eldőlni hogy a karakterünket melyik másik karakterre írja át a programunk pl:ha a gép 92-öt sorsol akkor a char *leet[0] tömb első elemére cserélünk. Az int found változónak az a feladat hogy jelzni hogy a tömbünkbe megtalálható-e a karakterünk ha nem akkor visszaadjuk úgy ahogy van.

```
%{
%%
. {

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

    {
        if(1337dlc7[i].c == tolower(*yytext))
        {

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

            if(r<91)
                printf("%s", 1337dlc7[i].leet[0]);
            else if(r<95)
                printf("%s", 1337dlc7[i].leet[1]);
            else if(r<98)
                printf("%s", 1337dlc7[i].leet[2]);
            else
                printf("%s", 1337dlc7[i].leet[3]);
```

```
    found = 1;
    break;
}

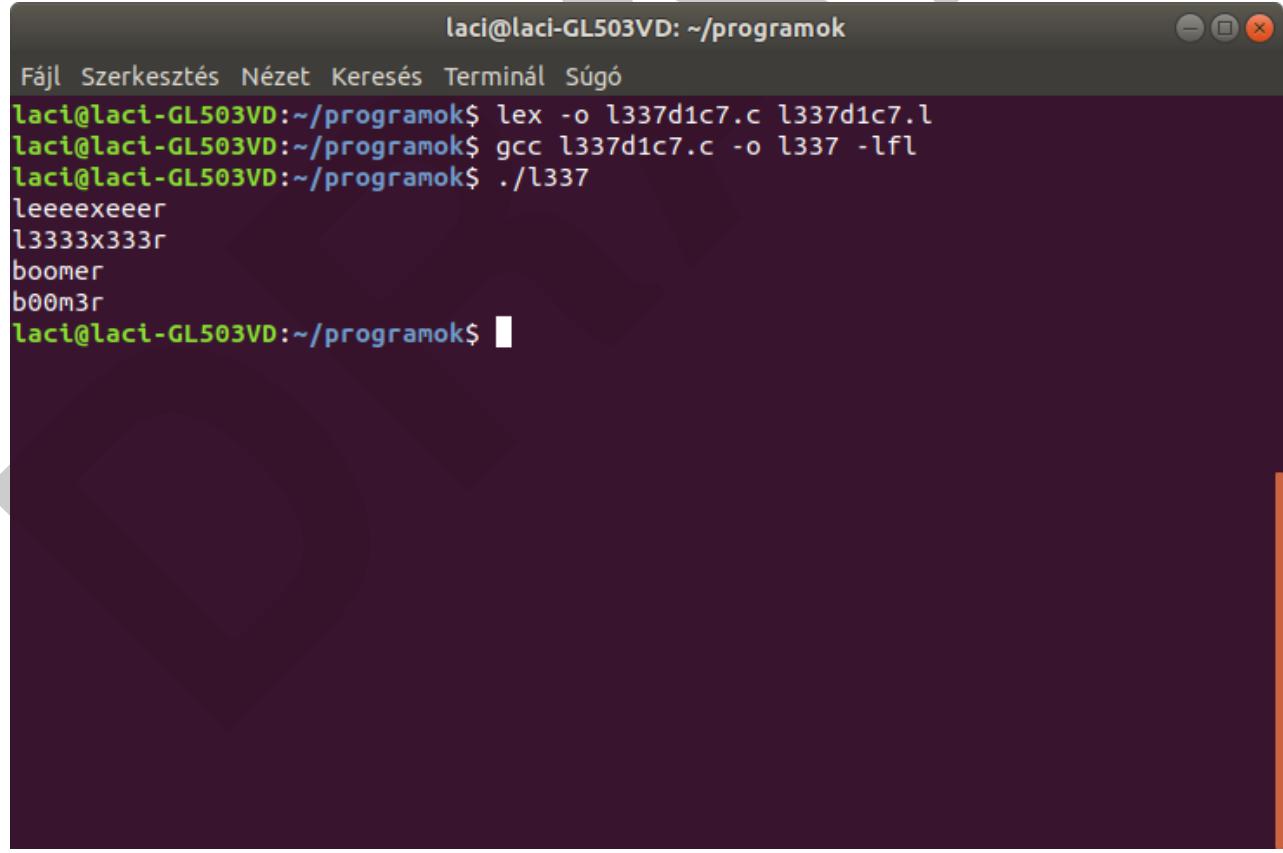
}

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

}
```

A legutolsó részben elindítjuk a lexelést.

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



The screenshot shows a terminal window titled "laci@laci-GL503VD: ~/programok". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ lex -o l337d1c7.c l337d1c7.l
laci@laci-GL503VD:~/programok$ gcc l337d1c7.c -o l337 -lfl
laci@laci-GL503VD:~/programok$ ./l337
leeeexeeeer
l3333x333r
boomer
b00m3r
laci@laci-GL503VD:~/programok$
```

3.6. A források olvasása

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

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

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

Bugok

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

i.

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

Ez azt jelenti, hogyha eddig nem volt figyelmen kívül hagyva a SIGINT jel, akkor a jelkezelő függvény kezelje. Ellenkező esetben hagyjuk figyelmen kívül.

ii.

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

Ez egy teljesen alap for ciklus ahol első iterációba az i az 0 majd ellenőrizzük hogy i kisebb mint 5 és minden körrrel növeljük az i-t 1-el

iii.

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

Gyakorlatilag ez a for ciklus megegyezik a felette lévővel de nem mindegy melyik oldalra tesszük a ++-okat mert ha bal oldalt tesszük akkor megváltoztatjuk a változó értékét de az i++-al nem változik a változó értéke

```
i = 1;
j = ++i;
(i az 2, j az 2)
```

```
i = 1;
j = i++;
(i az 1, j az 2)
```

iv.

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

Ez egy bugos program mert egyszerre hozzuk létre az i-t és hivatkozunk a tomb[i]-re és ez nem fog lefutni.

v.

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

Ez a program is bugos mivel értékkadó operátort használunk ezért a `&&` jobb oldalán nem egy logikai operandus áll

vi.
vi.

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

Ez a kódcsipet is hibás mivel két int-et adunk de nincs kiértékelési sorrendjük

vii.
vii.

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

Ez a programrész nincs baj kiírjuk az a értékét utánna meg a módosított a-t

viii.
viii.

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

Ennek a kódcsipetnek is kiértékelési problémája van.

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

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 (\forall y \text{ prim})) \leftrightarrow
```

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

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

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

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

1. minden számnál van nagyobb prím szám.
2. minden számnál létezik nagyobb ikerprím számpáros.
3. van olyan szám amelynél minden prím szám kisebb.
4. van olyan szám aminél bármely nagyobb szám nem prím.

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Mit vezetnek be a programba a következő nevek?

- egész

```
int a;
```

- egészre mutató mutató

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

- egész referenciaja

```
int &c = a;
```

- egészek tömbje

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

- egészek tömbjének referenciaja (nem az első elemé)

```
int (&tr)[2] = c;
```

- egészre mutató mutatók tömbje

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

- egészre mutató mutatót visszaadó függvény

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

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int *(*1)();
```

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

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

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

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

Vezesd be egy programba (forduljon le) a következőket:

- int a;

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

Megoldás videó:

Megoldás forrása:

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összehasonlítása 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...

DRAFT

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c](https://bhax.org/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c)

Elsőnek tiszátnunk kell mi is az a hárömszögmátrix.Háromszögmátrixnak 2 tulajdonsága van az első hogy négyzetes,tehát a sorai és oszlopai száma megegyeznek a másidik tulajdonsága az hogy a főátlója alatt csupa nulla szerepel a mi mátrixunk az egy alsó mátrix.Most már tudjuk mit várunk el a programtól nézzünk is bele a forrásba.

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

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

Először is include-áljuk a megsokott header file-okat. Ezek után létrehozunk egy int típusú nr nevű változót aminek az értéke 5 lesz. A későbbiekben ez fogja majd meghatározni, hogy a kirajzolásnál hány sort fog a program a futtatás után megjeleníteni. Még itt deklaráljuk a **tm pointert is.

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

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

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

Először kiíratjuk a tm memóriacímét majd ezek után az if-en belül látható egy malloc nevezetű függvény ami annyit csinál, hogy helyet foglal a memóriában jelen esetben 40 bájtot, mert az nr értéke 5 a double 8 így ha összeszorozzuk a kettőt, akkor megkapjuk a 40 bájtot. A malloc alapból egy void típusat ad vissza vagyis ha típuskényszerítést alkalmazunk, akkor bármilyen tetszőleges típusat vissza tud adni. Jelen esetben ezt is használtuk, ezért látható itt a double ** mert ezzel elérjük azt hogy a visszatérés típusa egy double ** legyen. Az if még annyit tesz, hogy leellenőrzi, hogy a malloc sikeresen lefoglalta-e a helyet a memóriában és hogy sikeresen vissza adta-e a double ** mutatót. Ha ez nem sikerült, akkor egyenlő a NULL-al, vagyis nem mutat seholvá és kilép a programból, de ha sikerült akkor kiírjuk a memóriacímét.

```

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

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

```

A forciklussal végigmegyünk az 5 soron és minden egyik sorban újra megtörténik a memória foglalás. Például nézzük az alábbi esetet amikor is az i=2: A malloc a tm[2]-nek most $(2+1 * 8)$ bájtot foglal, vagyis a harmadik sorban három 8 bájtnyi helyet foglal le. Ez az egész egy if feltételeként szerepel, melyben ismét ellenőrizzük, hogy sikeres volt-e a helyfoglalás. Ha sikeresen jártunk, akkor kiíratjuk a tm[0] memóriacímét.

```

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 ("%d, ", tm[i][j]);
    printf ("\n");
}

```

A következő forciklusokkal létrehozzuk az alsó háromszögmátrixot. A for cikluson belül értéket adunk a harmadik sori int-eknek. Az i-vel megyünk a 4-ig, vagyis nr-1-ig, j-vel pedig mindenkor 0-tól i-ig. Az i jelöli a sorok számát, a j pedig az oszlopokat. Mátrix minden eleméhez a sorszám*(szorszám+1)/2+oszlopszám, és ezzel megkapjuk a feladat legelején felvázolt mátrixot, amit a következő for-ban már csak elemenként kiíratunk.

```

tm[3][0] = 42;
(* (tm + 3)) [1] = 43; // mi van, ha itt hiányzik a külső ←
()
* (tm[3] + 2) = 44;
* (* (tm + 3) + 3) = 45;
for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
}

```

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

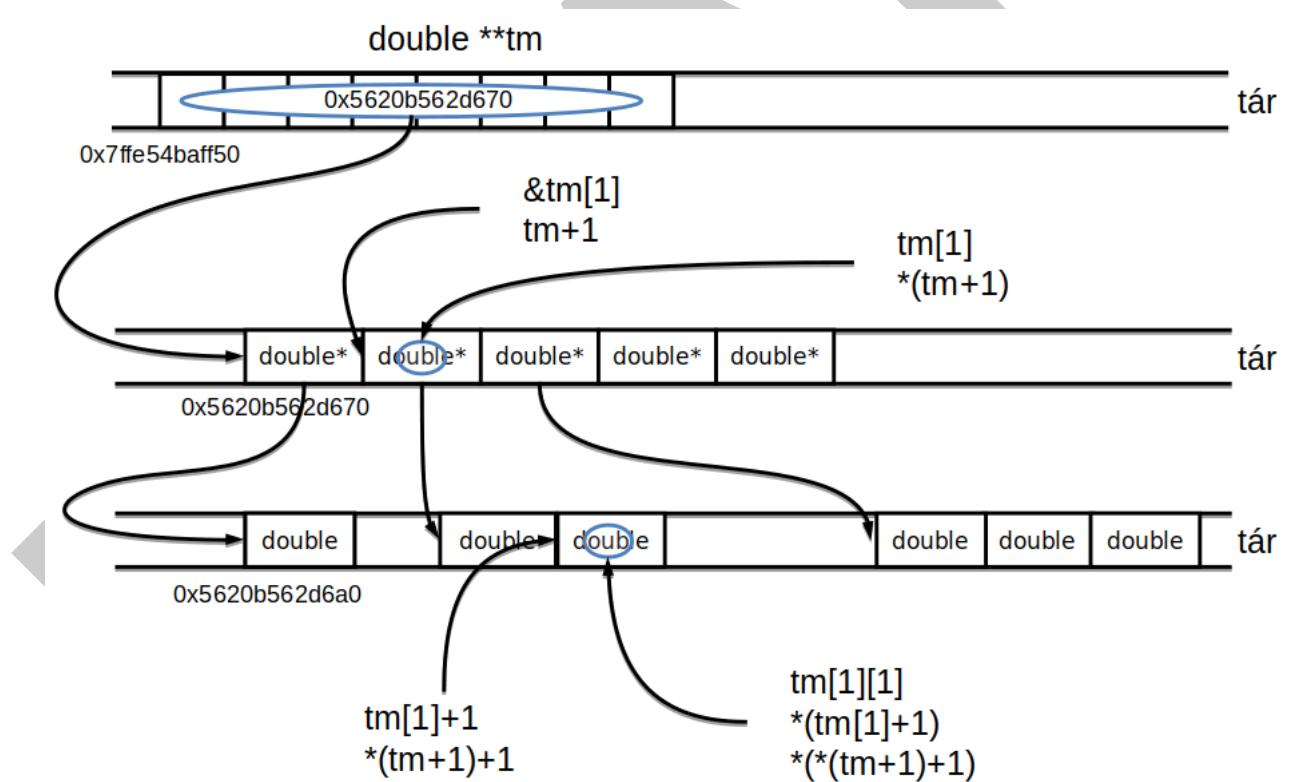
A tm 3. sorának első elemének értékét 42-re módosítjuk. Utána a harmadik sor második elemének az értékét változtatjuk, majd a harmadik sor harmadik elemét, végül pedig a harmadik sor negyedik elemét. A második lehetőségnél felmerül a kérdés, hogy elhagyható-e a külső zárójel. Elhagyható viszont, így nem a harmadik sorba lesz a módosítás, hanem a 4. sor első eleménél, mivel $*(tm + 3)[1]$ azzal ekvivalens, hogy $*(tm+4)$. Az értékek megváltoztatása után újra kiíratjuk az alsó háromszögmátrixunkat a forciklus segítségével.

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

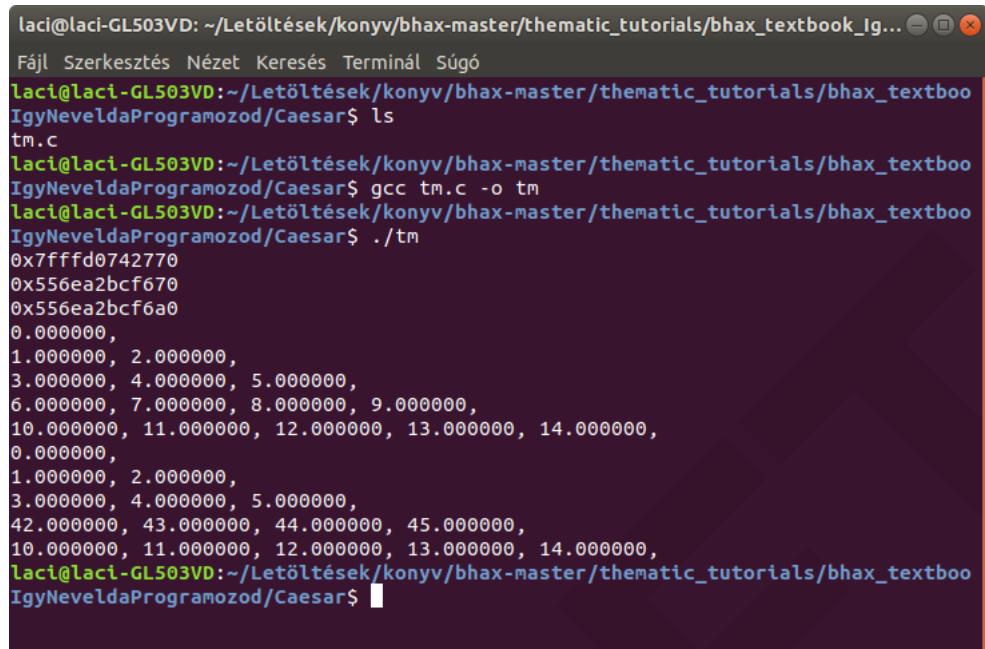
free (tm);

return 0;
}
```

A program utolsó részében a free függvény segítségével felszabadítjuk az egyes sorokban illetve az egész tm által foglalt memóriát.



4.1. ábra. A `double **` háromszögmátrix a memóriában



```
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textbook_lg... Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textbook_lg... IgyNeveldaProgramozod/Caesar$ ls
tm.c
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textbook_lg... IgyNeveldaProgramozod/Caesar$ gcc tm.c -o tm
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textbook_lg... IgyNeveldaProgramozod/Caesar$ ./tm
0x7fff0742770
0x556ea2bcf670
0x556ea2bcf6a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/thematic_tutorials/bhax_textbook_lg... IgyNeveldaProgramozod/Caesar$
```

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

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 használj azt, dolgozd fel, javítsd, adj hozzá értéket!

Az EXOR titkosító lényegében a logikai vagyra, azaz a XOR műveletre utal, mely bitenként összehasonlítja a két operandust, és minden 1-et ad vissza, kivéve, amikor az összehasonlított 2 Bit megegyezik, mert akkor nullát. Tehát 2 operandusra van szükségünk, ez jelen esetben a titkosítandó bemenet, és a titkosításhoz használt kulcs. Ideális esetben a kettő mérete megegyezik, így garantálható, hogy szinte feltörhetetlen kódot kapunk, mivel túl sokáig tart annak megfejtése. Viszont ha a kulcs rövidebb, mint a titkosítandó szöveg, akkor a kulcs elkezd ismétlődni, ami biztonsági kockázatot rejt magában.

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

Először a kulcs és a buffer méretének maximumát konstansban tároljuk el, ezek nem módosíthatóak.

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

A main függvénynek argumentumokat adunk, argc-vel adjuk át az argumentumok számát, és az argumentumokra mutatók pedig az argv tömbben tároljuk el.

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

A main() belül deklarálunk két tömböt, egyikbe a kulcsot tároljuk, a másikban pedig a beolvasott karaktereket, mind a kettőnek a mérete korlátozott.

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

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

Létrehozunk számlálókat, melyek segítségével bejárjuk majd a kulcs tömböt, és számoljuk a beolvasott bajtokat. A kulcs méretét a strlen() függvényel kapjuk meg, amely visszadja a másodjára megadott érték hosszát. Ezután a strncpy() függvényel átmásoljuk az argv[1]-ben tárolt sztringet karakterenként a kulcs tömbe, lényegében mindegyikhez visszaad egy pointert. A MAX_KULCS-csal pedig meghatározzuk, hogy mennyi karaktert másoljon át.

```
while ((olvasott_bajtok = read (0, (void *) buffer, ←
    BUFFER_MERET)) )
{
    for (int i = 0; i < olvasott_bajtok; ++i)
    {
        buffer[i] = buffer[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }

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

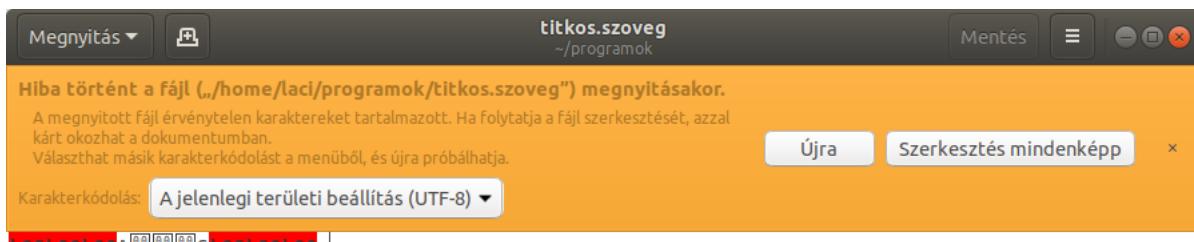
A while ciklus feltétele addig lesz igaz, ameddig a read parancs beolvassa a megadott mennyiségi bajtokat. A read 3 argumentumot kap: az első az, hogy honnan olvassuk be a bajtokat, jelen esetben a standard inputról olvasunk, a beolvasott bajtokat a buffer-ben tároljuk egészen addig, ameddig el nem érjük a megadott mennyiséget, amit BUFFER_MERET definiál. Ezután pedig végigmegyünk elemenként a bufferben eltárolt karaktereken és össze EXOR-ozzuk a kulcs tömb megfelelő elemével, majd inkrementáljuk a kulcs_index-et 1-el, mely egészen addig nő, ameddig el nem érjük a kulcs_meret-et, ekkor lenullázódik. Végezetül pedig kiírjuk a buffer tartalmát a standard outputra.

```
laci@laci-GL503VD:~/programok$ gcc exor.c -o exor
laci@laci-GL503VD:~/programok$ ./exor asd <asd.szoveg > titkos.szoveg
laci@laci-GL503VD:~/programok$
```

Megnyitás ▾ asd.szoveg ~/programok Mentés

asd asd das|

Egyeszerű szöveg ▾ Tabulátorszélesség: 8 ▾ 1. sor, 12. oszlop ▾ BESZ



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

Tanulságok, tapasztalatok, magyarázat...

A legfontosabb dolog hogy a java az egy objektum orientált programozási nyelv azaz létre tudunk hozni objektumokat másnéven class-okat amikkel bizonyos utasításokat végezhetünk.Javában a classok nem különülnek ell mint c++-ba hanem main() függvénybe számítanak.A jexor függvény meghívjuk a input és ostream függvényeket hogy tudjuk mahd olvasni a bajtokat és ha ez nem sikerül throws hibát fog kiírni.Ezek után byte-okból álló tömböt hozunk létre kulcs és buffer néven. A getBytes() függvényel olvassuk be a kulcsot a kulcs tömbbe. A buffer tömbnek foglaltunk 256 bajt-nyi helyet a memóriába.Majd definíáljuk a kulcs tmb bejárásához, és a beolvasott bajtok számlálására.A while ciklus addig fog menni amíg nem olvassuk be a buffert vagy már nem tudunk többet beolvasni.Majd a beágyazott for ciklussal elemenként összeszorozzuk a buffer tartalmát a kulcsal és növeljük a kulcsindexet a % operátorral.Ennek következtében ha elérjük a kulcs hosszát, akkor lenullázódik.

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

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

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

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

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

    }

    kimenőCsatorna.write(buffer, 0, olvasottBájtok);

}

}
```

Ebben a részben csak meghívjuk ez elején deklarált jexort és futtatjuk a try catch pedig ha valami baj történik hibát fog kidobni.Ha nem adunk meg kulcsot akkor hibát fogunk kapni.

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

    try {

        new jexor(args[0], System.in, System.out);

    } catch(java.io.IOException e) {

        e.printStackTrace();

    }

}
```

The terminal window shows the following commands and their outputs:

```
laci@laci-GL503VD:~/programok$ java jexor.java
error: class found on application class path: jexor
laci@laci-GL503VD:~/programok$ javac jexor.java
laci@laci-GL503VD:~/programok$ java jexor 00000001 <asd.szöveg > titkos.txt
laci@laci-GL503VD:~/programok$
```

The Microsoft Word document shows the following content in the 'titkos.txt' file:

QCT⁰⁰QCT⁰⁰TQC:

Below the document, the status bar displays:

Egyszerű szöveg ▾ Tabulátorszélesség: 8 ▾ 1. sor, 13. oszlop ▾ BESZ

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE

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

Mint az exorban is itt is meghatározunk bizonyos konstansokat, és includoljuk a header fájlokat.

```
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;
}
```

Az atlagos_szohossz függvényel kiszámoljuk a bemenünk átlagos szóhosszát, odaadunk neki egy tömböt és a tömbnek a méretét. Majd a for ciklussal körbejárjuk a tömböt és minden elem után hozzáadunk 1-et az sz-hez.Return-ként pedig a tömb méretét/sz értéket adjuk.

```
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");

}
```

Ebbe tiszta_lehet függvényben azt akarnánk ellenőrizni hogy a bemenetként adott szövegünk tiszta szöveg e. Ezt úgy ellenőrzük hogy megnézzük hogy tartalmazza-e a szövegünk a leggyakoribb magyar szavakat. Ám mi van ha szövegünk nem tartalmazza a kért szavakat de mégis tiszta szöveg kap nem tudja visszafejteni.

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

```
int kulcs_index = 0;

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

}
```

Az exor függvény ugyan azt csinálja mint az exor titkosító mivel ha valamit 2x exoro-zunk akkor visszakapjuk a tiszta szöveget. Az argumentumban kap egy lehetséges kulcsot és annak a méretét meg a titkos szöveget a méretével.

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

    return tiszta_lehet (titkos, titkos_meret);
}
```

Az exor_tores meghívja az előtte definiált függvényeket és vagy 0-t vagy 1-et ad vissza ami attól függ hogy a szövegünk tiszta vagy nem.

```
int
main (void)
{
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;
```

Most a mainben elsőnek deklarálunk egy kulcs tömböt és egy titkos tömbet melyeknek a mérete a program elején megadott konstansok. Még definiálunk egy mutatót, mely a titkos tömbre mutat és létrehozunk egy számlálót az olvasott_bajtok néven.

```
// 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;
```

Ezzel a while ciklussal addig olvassuk a bájtokat amíg a buffer tele nem lesz.

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

A for ciklussal kinullázzuk a buffer megmaradt helyeit és előállítjuk az összes létező kulcsot.

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

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

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

return 0;
}
```

Itt a végén lefuttatjuk az összes kulcsot és meghívjuk az exor_tores ←
függvényt és ha igazat ad
kiirajta a szöveget és a használt kulcsot

4.5. Neurális OR, AND és EXOR kapu

R

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

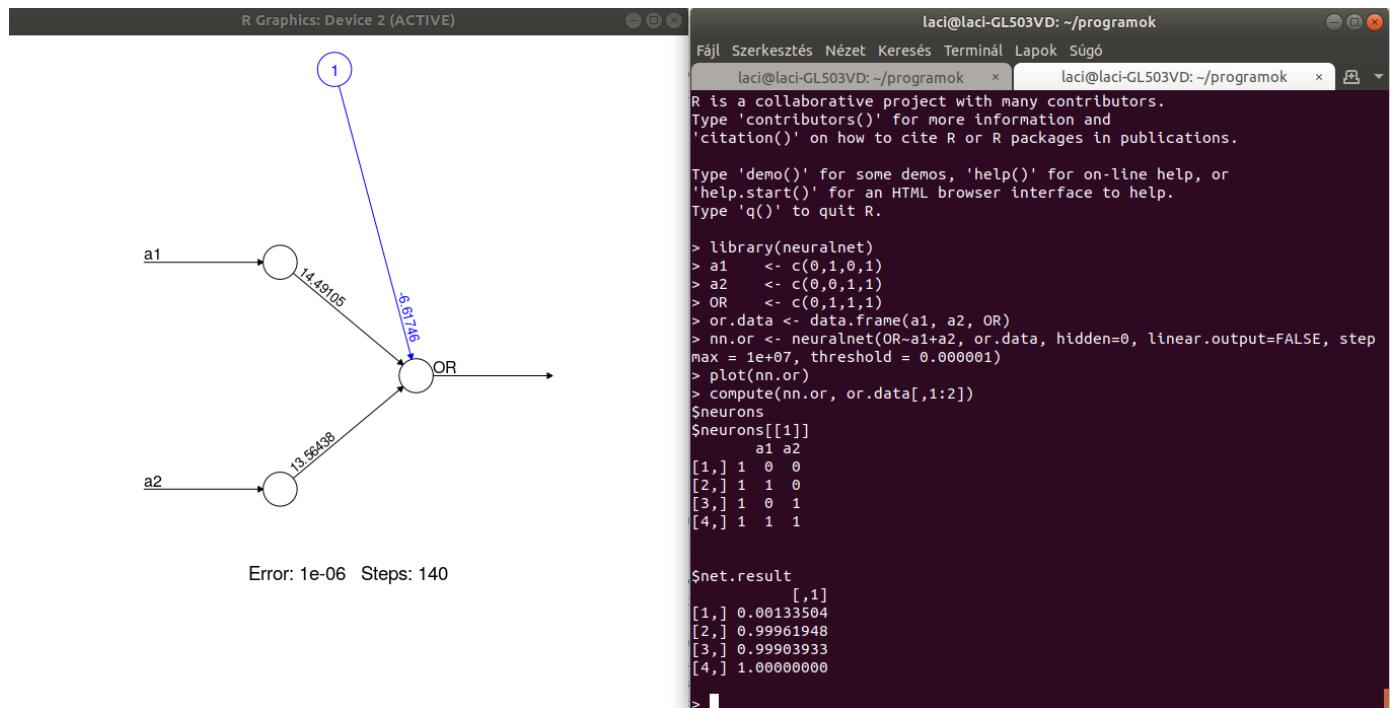
Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R

Tanulságok, tapasztalatok, magyarázat...

OR

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

Az a1 és az a2-ben megadtunk 0-k és 1-sek sorozatát majd az OR-ra az "a1 or a2" művelet eredményét, majd ebből adatot csinálunk az or.data változóba. Az nn.or változóba használjuk a neuralnet függvényt, amely kiszámolja nekünk a neutrális hálót. Átadjuk neki az adatokat (amit az előbb előállítottunk az a1, a2 és az OR-ból), tehát megtanítjuk neki ezt az OR műveletet, majd saját magát továbbfejlesztve beállítja a súlyokat úgy hogy megtanulja a dolgok menetét. Fontos hogy a neuralnet függvényt nem a logikai műveletet végzi el, hanem tényleges tanulás után probálja megmondani az eredményt nekünk. A plot függvénytel kirajzoltatjuk az első csatolt képen látható ábrát, ami illusztrálja a számításokat. Itt látható az ismétlések száma, hogy hányszor végezte a műveletet (steps) és hogy ezekből hányszor kapott rossz eredményt (errors, nagyon kicsi). A compute parancccsal beadjuk neki az adatokat és számításra utasítjuk a függvényünket, tehát kikérdezzük tőle azt amit megtanítottunk neki az elején. És ahogy a második kép mutatja ügyesen megtanulta és vissza is adta a helyes eredményeket.



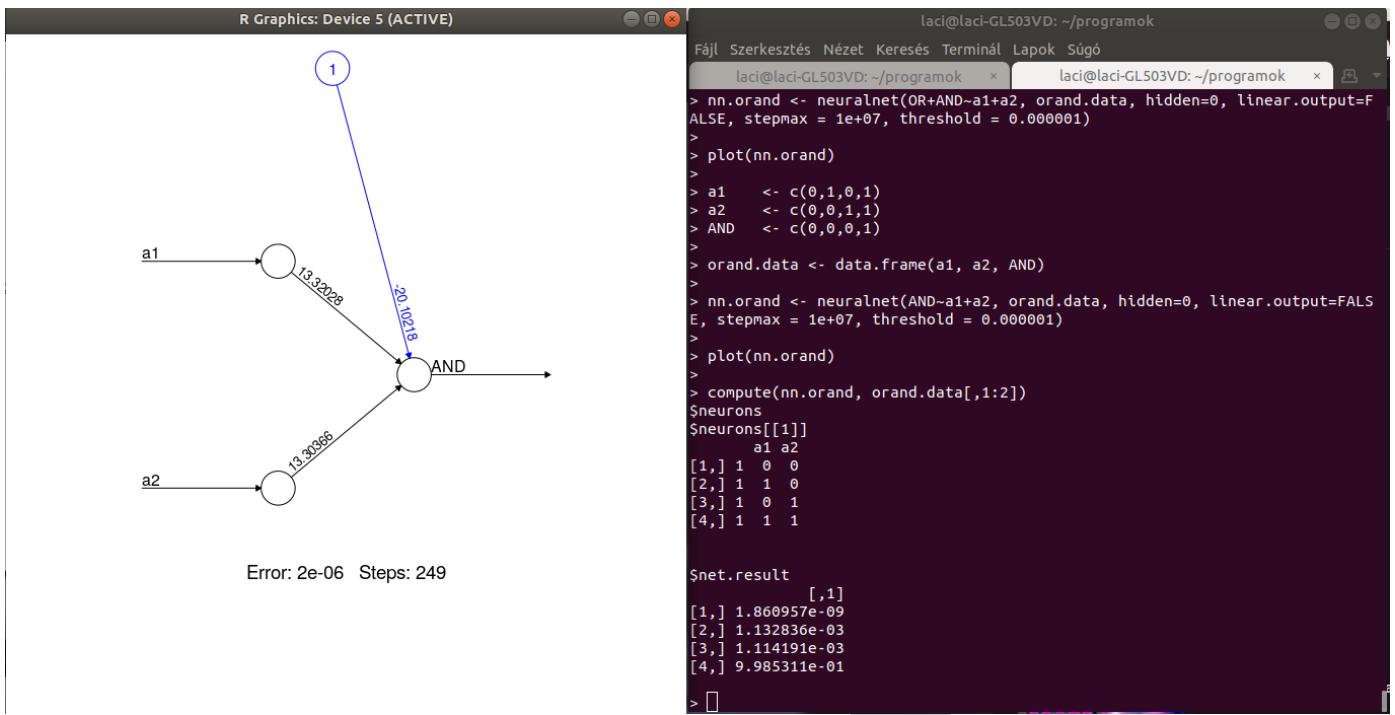
AND

```

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

```

Ugyan ez folyik le az end műveletnél is.



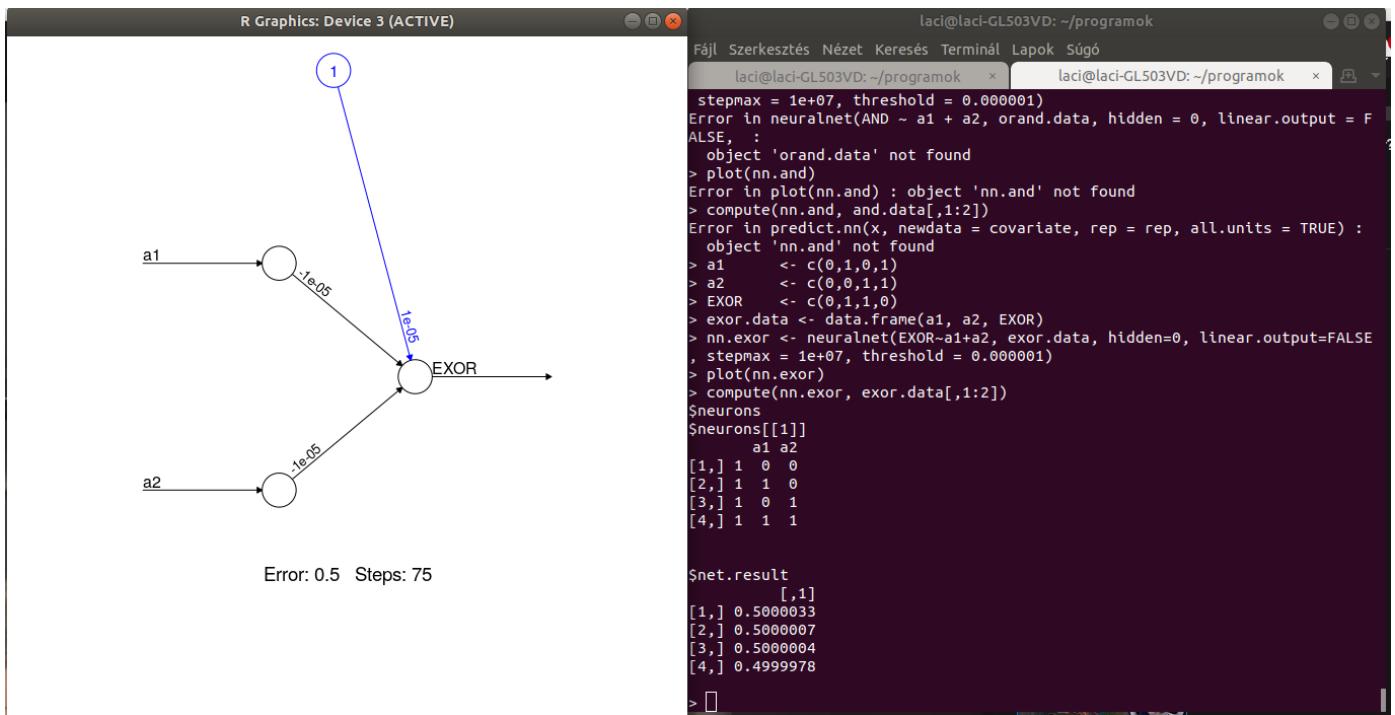
EXOR

```

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

```

Ezután pedig jönne az EXOR, viszont ez már nem annyira egyszerű. Amikor régen ezt a technológiát kitalálták, és az EXOR nem működött, sokan elpártoltak tőle. Majd a kor nagy matematikusai megfejtették, hogy nem lehetetlen feladat, csak egy apróságra van szükség, létre kell hozni a rejtett neuronokat, melyek segítik a tanulást.

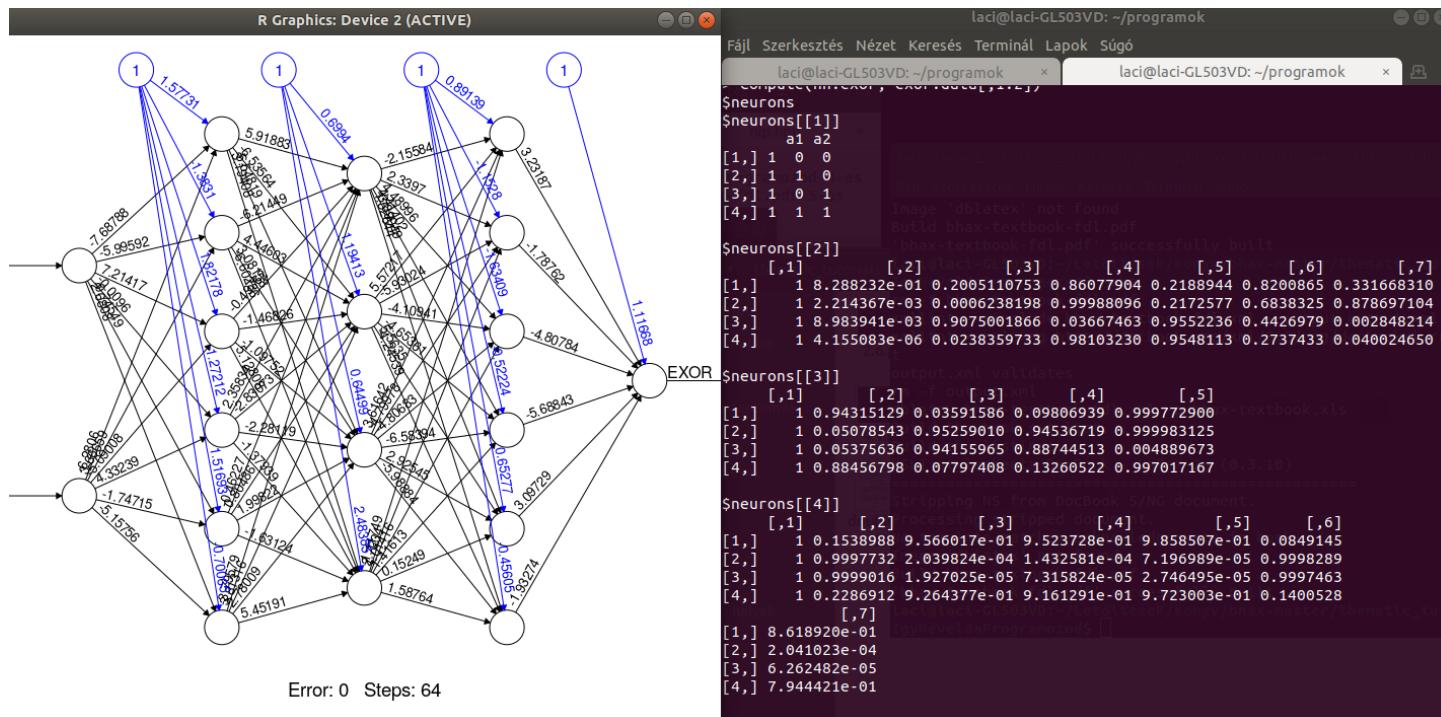


EXOR neuronokkal

```

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])
  
```

Most hogy 6 4 6 neuron beállítást adtunk meg a programunk hibarátája lement 0.5-ről 0-ra esett.



4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

Tanulságok, tapasztalatok, magyarázat...

```

#include <iostream>
#include "perceptron.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);
}

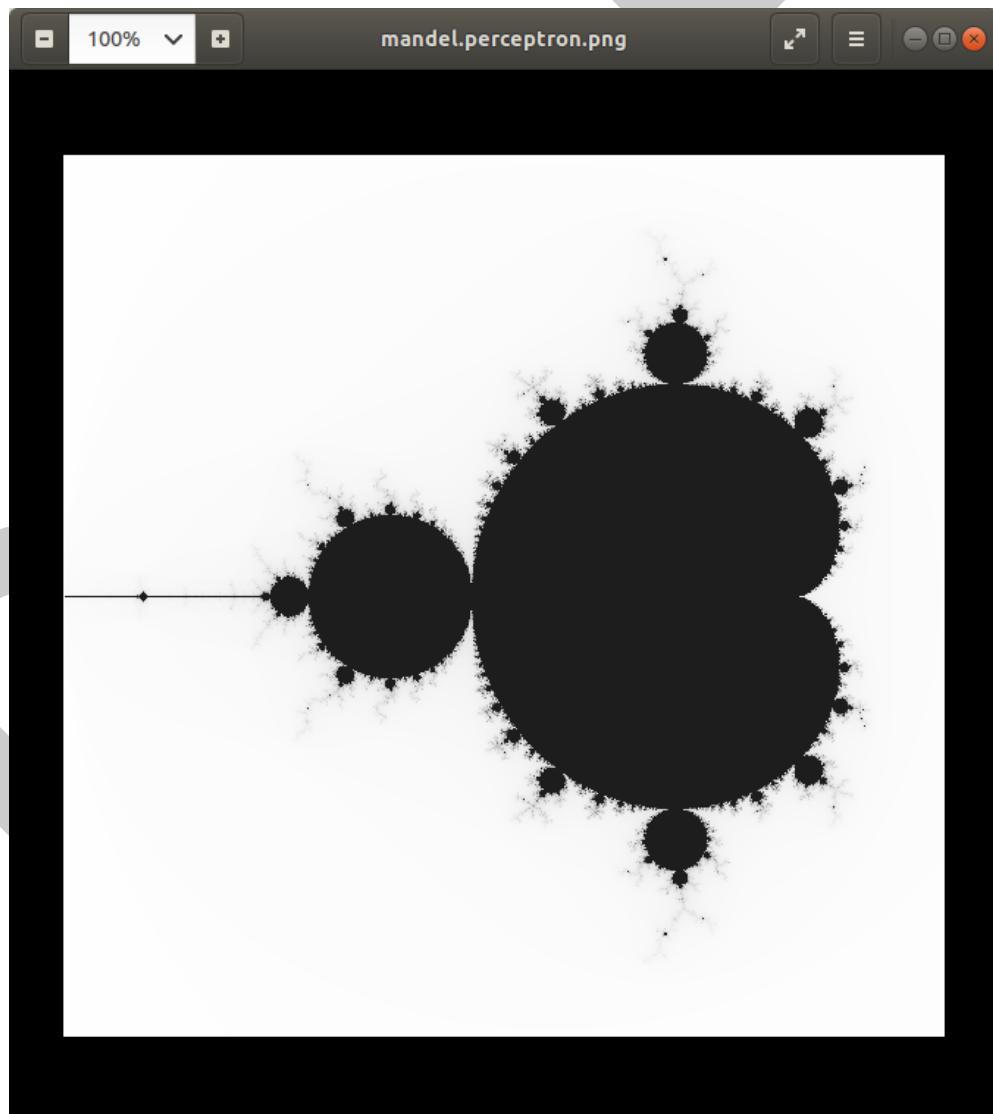
```

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

    delete p;
    delete [] image;

}
```

A fentebb említett perceptron.cpp programot a mainben meg is hívjuk header fájlként, átláthatóbbá téve a főprogramot. A fő számítások viszont a perceptron.hpp-ben vannak, a mainben az ott deklarált Perceptron osztályt hívjuk segítségül meg az eredmény kiszámolásának céljából. A mainben a png.hpp header fájl segítségével létrehozunk egy új png kiterjesztésű képet, úgyanolyan szélességgel és magasséggel mint a mandelbrotos kép volt. A két egymásbaágazódó for ciklus segítségével végigmegyünk a kép minden pixelén és az előzőekben lementett mandel_perceptron.png pixeleinek piros (red) komponenseit rámásoljuk a most létrehozott kép pixeleire. A program végén pedig kiíratjuk ezt a percceptron értéket a value változó segítségével.



5. fejezet

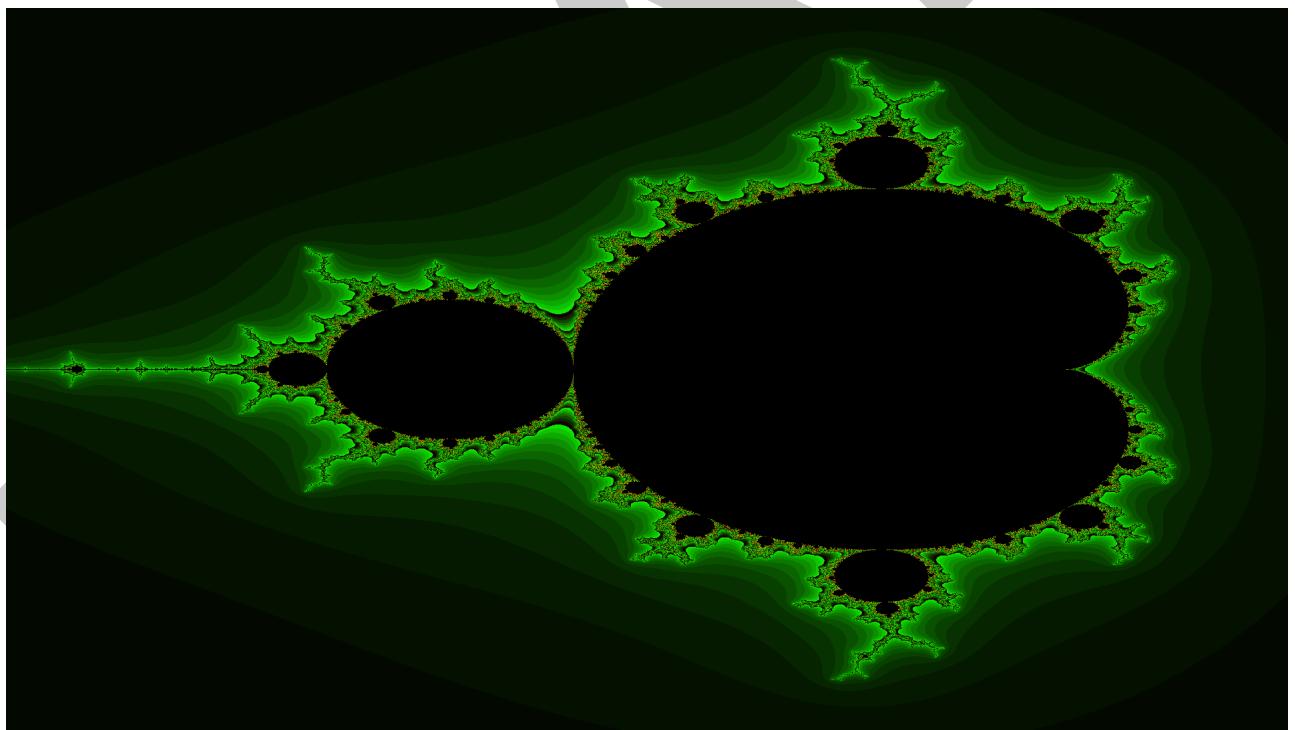
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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



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 rácspont. A z_0 az origó. Alkalmazva a képletet a

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

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

A kép kreálásához szükségünk lesz a png++ header fájlra amit a sudo apt -get install libpng++-dev parancsal tudjk feltelepíteni és a g++ fordítónál szükségünk van a -lpng kapcsolóra.

```
// mandelpngt.c++
// 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/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063\_01\_parhuzamos\_prog\_linux
//
```

```
// https://youtu.be/gvaqijHlRUs
//
#include <iostream>
#include "png++/png.hpp"
#include <sys/types.h>
```

Itt meghívjuk az általános header fájlokat itt az érdekesség a png++ és a sys/times a png++ azért importáljuk hogy tudjunk képet generálni és sys/times pedig hogy letödjük mérni a futási időt a programnak.

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

Létrehozunk 2 konstant értéket.

```
[  
void  
mandel (int kepadat[MERET] [MERET]) {  
  
    // Mérünk időt (PP 64)  
    clock_t delta = clock ();  
    // Mérünk időt (PP 66)  
    struct tms tmsbuf1, tmsbuf2;  
    times (&tmsbuf1);
```

Itt mérjük az időt az elsőbe csak egy 4 jegyű számként a másodikba pedig 4 tizedes jegy pontosságig másodpercben.

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

Ezen a rész adjuk meg a mandelbrot halmaz tulajdonságait.

```
[  
    // a számítás  
    float dx = (b - a) / szelesseg;  
    float dy = (d - c) / magassag;  
    float reC, imC, reZ, imZ, ujreZ, ujimZ;  
    // Hány iterációt csináltunk?  
    int iteracio = 0;  
    // Végigzongorázzuk a szélesség x magasság rátcsot:  
    for (int j = 0; j < magassag; ++j)  
    {  
        //sor = j;  
        for (int k = 0; k < szelesseg; ++k)  
        {  
            // c = (reC, imC) a rátcs csomópontjainak  
            // megfelelő komplex szám  
            reC = a + k * dx;
```

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

    ++iteracio;

}

kepadat[j][k] = iteracio;
}
}

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

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

Itt folynak le számítások. Magát az értékeket 2 változóban kell tárolnunk mivel a programunk nem képes komplex számok feldolgozására. Miután feltöltöttük a tömbünket képadatokkal kiiratjuk a lefutási időt amiket a program elején hoztunk létre.

```
[

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

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

```
}

int kepadat [MERET] [MERET];

mandel (kepadat);

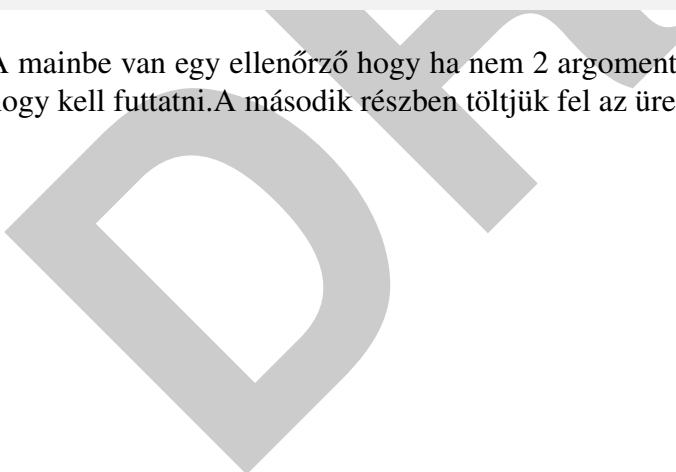
png::image < png::rgb_pixel > kep (MERET, MERET);

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

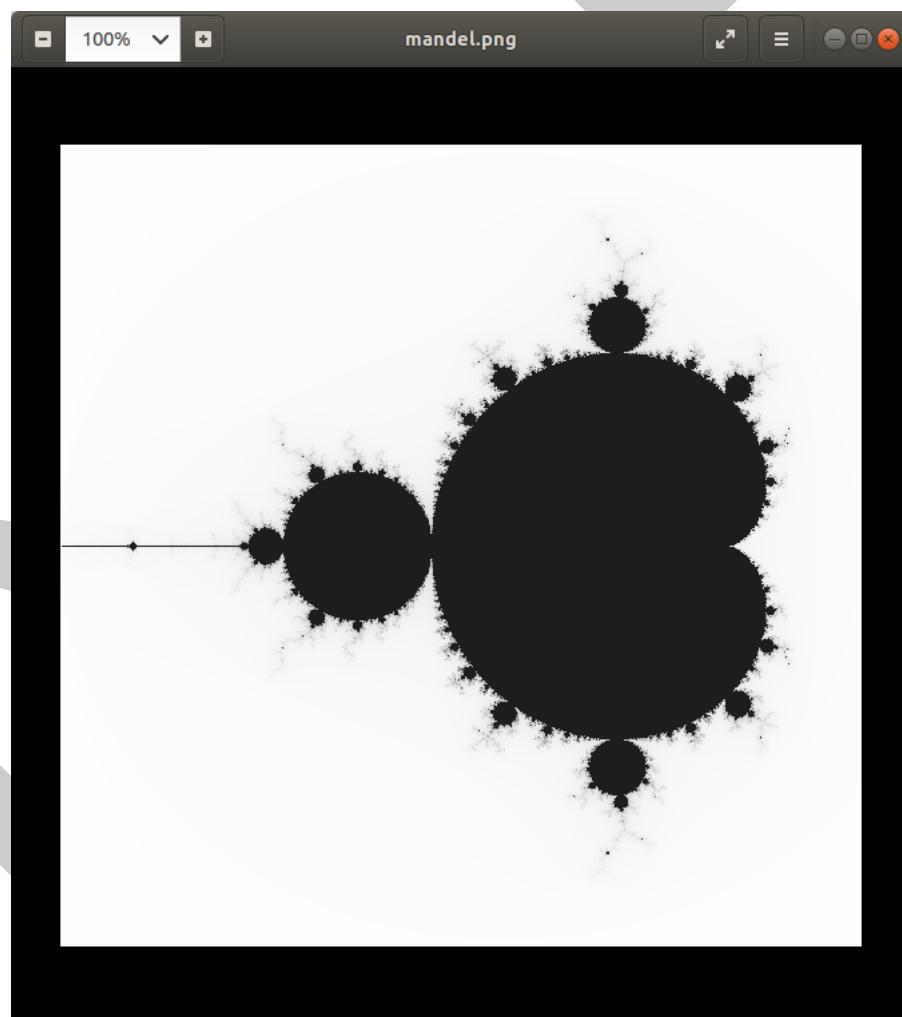
kep.write (argv[1]);
std::cout << argv[1] << " mentve" << std::endl;

}
```

A mainbe van egy ellenőrző hogy ha nem 2 argomentumot kap a consoloból akkor hibát ír ki és kiprinteli hogy kell futtatni.A második részben töltjük fel az üres png-et a kiszámolt pixel színekkel.



```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ touch mandel2019.cpp
laci@laci-GL503VD:~/programok$ g++ mandel2019.cpp -o mandel2019 -lpng
laci@laci-GL503VD:~/programok$ ./mandel2019 mandel2019.png
1614
16.1451 sec
mandel2019.png mentve
laci@laci-GL503VD:~/programok$
```



5.2. A Mandelbrot halmaz a std::complex osztályal

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

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

Megoldás forrása:

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

```
// 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
//
//
// 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>
```

Mint minden elején include-oljuk a header fájlokat. Ami itt újnak tűnhet az a complex header ez annyit tesz lehetővé hogy a programunk képes legyen komplex számok feldolgozására így nem kell 2 változóként mentenünk a számokat hanem csak elég egybe.

```
[  
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;  
    }  
}
```

Mint az előző feladatban itt megtaláljuk a mandelbrot halmazunk értékeit csak annyi a különbség hogy a felhasználó megadhatja az attribútum értékeit de nem muszály mert ha nem adunk meg semmit akkor az alamértelmezett számokkal fut le. Meg persze van egy ellenőrző hogy ha több vagy kevesebb argumentumot ad meg a felhasználó akkor hibát dob ki és ki írja hogyan kéne futtatni.

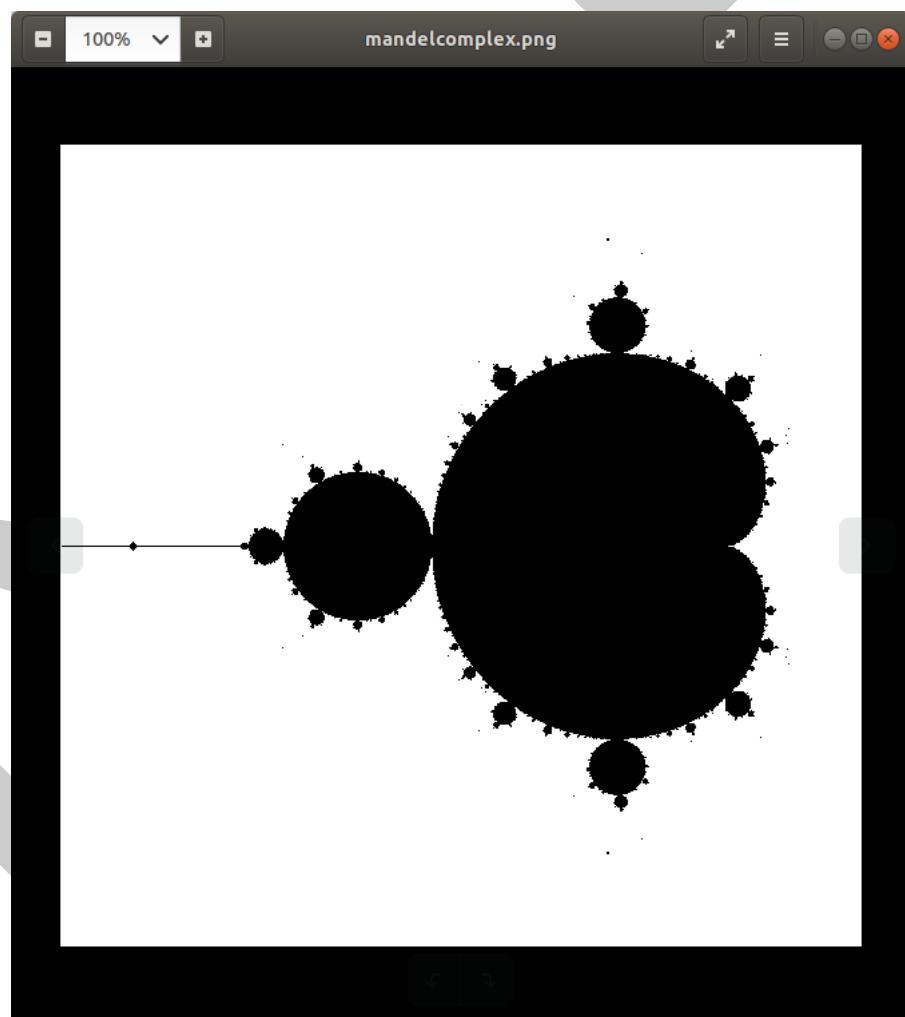
```
[  
    png::image < png::rgb_pixel > kep ( szelesseg, magassag );  
  
    double dx = ( b - a ) / szelesseg;  
    double dy = ( d - c ) / magassag;  
    double reC, imC, reZ, imZ;  
    int iteracio = 0;
```

Itt hozzuk létre az öres png-t és létrehozzuk az iteracio számlálót.

```
[ 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;  
}
```

Ebben a részben használjuk elsőnek a double-ket amik tartalmazzák a szám valódi alakját és az imaginárius alakját. Ennek köszönve tudjuk létrehozni a a c és a z_n változókat. Ezután számoljuk ki a c esetén a z_n-eket, és ha elérjük az iterációs határt akkor, tudhatjuk, hogy az iteráció konvergens.

```
laci@laci-GL503VD: ~/programok
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ g++ mandelpngt.cpp -o mandelcomplex -lpng
laci@laci-GL503VD:~/programok$ ./mandelcomplex mandelcomplex.png
1581
15.8174 sec
mandelcomplex.png mentve
laci@laci-GL503VD:~/programok$
```



5.3. Biomorfok

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

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 < iteraciosHatar; ++i)
```

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

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

```
// Verzio: 3.1.3.cpp  
// Forditas:  
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3  
// Futtatas:  
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10  
// Nyomtatás:  
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ←  
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←  
// color  
//  
// BHAX Biomorphs  
// Copyright (C) 2019  
// Norbert Batfai, batfai.norbert@inf.unideb.hu  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.  
//  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <https://www.gnu.org/licenses/>.  
//  
// Version history  
//  
// https://youtu.be/IJMbqRzY76E  
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9\_Iss5\_2305--2315\_Biomorphs\_via\_modified\_iterations.pdf  
//
```

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

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

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

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

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

A programunk eleje megegyezik a mandelbrotos programunkal csak annyiban tér el hogy a felhasználótól kérük be a cc konstans értékét és a különböszámot. Van 10 db parancssorban argumentumunk amit a felhasználónk tud megadni de ha nem ad meg semmit akkor az alapértelmezett értékeket használja fel a program. Az else akkor fog kiírórni ha nem megfelelően adjuk meg az értékeket ekkor a program is le áll.

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

double dx = ( xmax - xmin ) / szelesseg;
double dy = ( ymax - ymin ) / magassag;
```

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

Ebben a részben hozzunk létre egy üres pnt-t, a lépésközt a rácsok között a cc számmal fogjuk számolni.

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

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

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

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

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

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

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

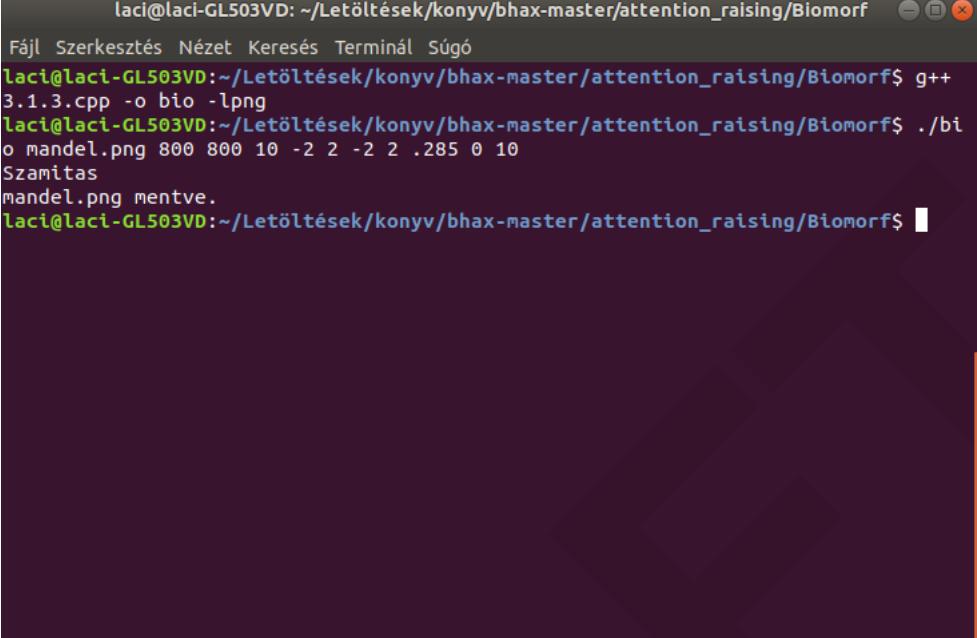
    int szazalek = ( double ) y / ( double ) magassag * 100.0;
    std::cout << "\r" << szazalek << "%" << std::flush;
}

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

A két egymásba lévő for ciklussal végigmegyünk a rácspontokon és egy harmadik for ciklussal kiszámoljuk a függvényértéket, addig amíg el nem érjük az iterációs határt.

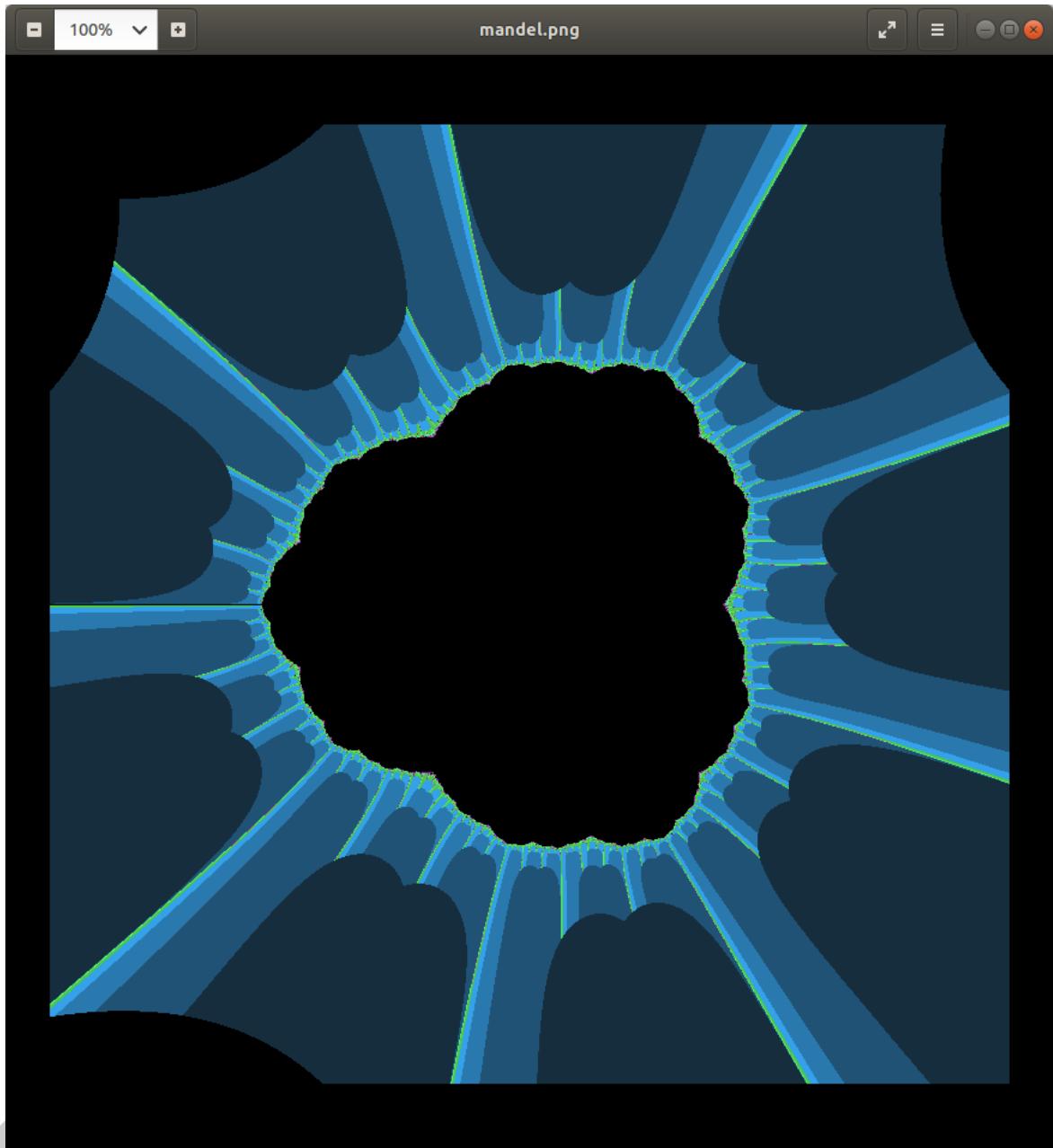
```
if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
```

Ez volt a bugos kódcsipet amit Clifford Pickover programja tartalmazott.



```
laci@laci-GL503VD: ~/Letöltések/konyv/bhax-master/attention_raising/Biomorf
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/Biomorf$ g++
3.1.3.cpp -o bio -lpng
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/Biomorf$ ./bi
o mandel.png 800 800 10 -2 2 -2 2 .285 0 10
Szamitas
mandel.png mentve.
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/Biomorf$
```

DRAFT



5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: [bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu](https://github.com/bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu) nevű állománya.

```
[  
// mandelpngc_60x60_100.cu  
// 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/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
//

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

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

#define MERET 600
#define ITER_HAT 32000

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

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

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

    // c = (reC, imC) a rács csomópontjainak
```

```
// megfelelő komplex szám
reC = a + k * dx;
imC = d - j * dy;
// z_0 = 0 = (reZ, imZ)
reZ = 0.0;
imZ = 0.0;
iteracio = 0;
// z_{n+1} = z_n * z_n + c iterációk
// számítása, amíg |z_n| < 2 vagy még
// nem értük el a 255 iterációt, ha
// viszont elértek, akkor úgy vesszük,
// hogy a kiinduláci c komplex számra
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;

}
return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{
    int j = blockIdx.x;
    int k = blockIdx.y;

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

}
 */

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

    int j = blockIdx.x * 10 + tj;
```

```
int k = blockIdx.y * 10 + tk;
kepadat[j + k * MERET] = mandel(j, k);
}

void
cudamandel (int kepadat [MERET] [MERET])
{
    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

    // dim3 grid (MERET, MERET);
    // mandelkernel <<< grid, 1 >>> (device_kepadat);

    dim3 grid (MERET / 10, MERET / 10);
    dim3 tgrid (10, 10);
    mandelkernel <<< grid, tgrid >>> (device_kepadat);

    cudaMemcpy (kepadat, device_kepadat,
               MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
    cudaFree (device_kepadat);
}

int
main (int argc, char *argv[])
{
    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

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

    int kepadat [MERET] [MERET];

    cudamandel (kepadat);

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

    for (int j = 0; j < MERET; ++j)
    {
```

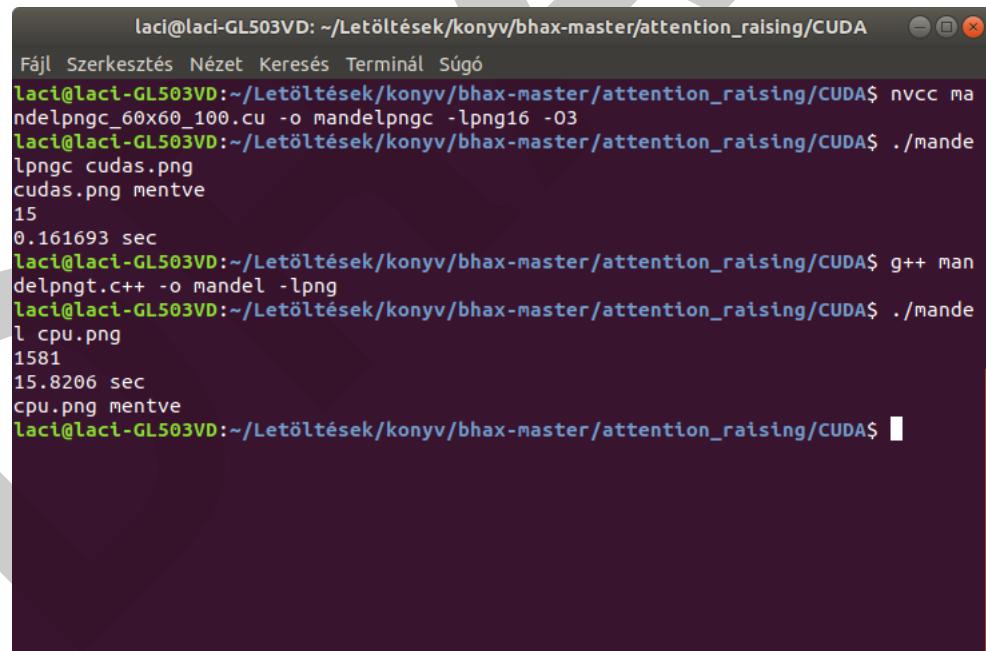
```
//sor = j;
for (int k = 0; k < MERET; ++k)
{
    kep.set_pixel (k, j,
        png::rgb_pixel (255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT));
}
kep.write (argv[1]);

std::cout << argv[1] << " mentve" << std::endl;

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

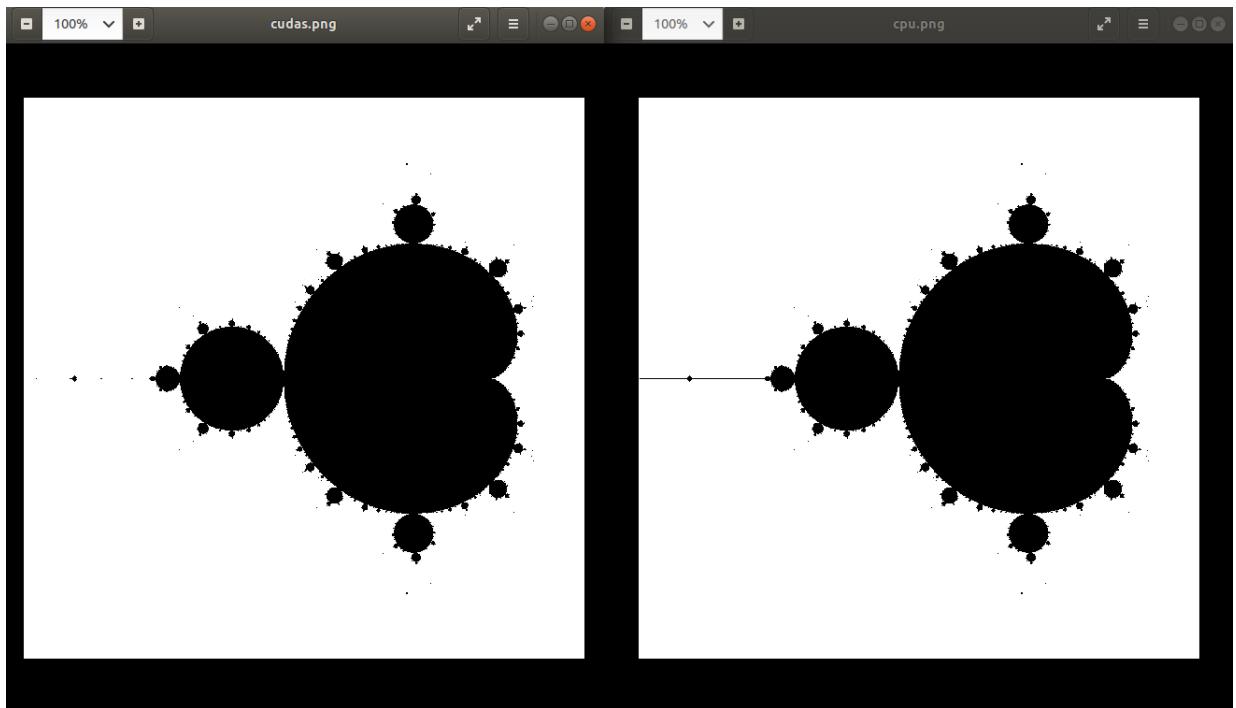
delta = clock () - delta;
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;

}
```



A terminal window titled 'laci@laci-GL503VD: ~/Letöltések/konyv/bhax-master/attention_raising/CUDA' displays the execution of a CUDA kernel and a CPU version of the Mandelbrot set generation. The CUDA execution (using nvcc) takes approximately 0.161693 seconds to generate a 60x60 image. The CPU execution (using g++) takes approximately 15.8206 seconds to generate a similar image. Both versions output the generated image to 'cudas.png'.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/CUDA$ nvcc mandelpngc_60x60_100.cu -o mandelpngc -lpng16 -O3
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/CUDA$ ./mandelpngc cudas.png
cudas.png mentve
15
0.161693 sec
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/CUDA$ g++ mandelpngt.cpp -o mandel -lpng
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/CUDA$ ./mandel cpu.png
1581
15.8206 sec
cpu.png mentve
laci@laci-GL503VD:~/Letöltések/konyv/bhax-master/attention_raising/CUDA$
```



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:

A program a QT GUI-t használja, ennek segítségével tudjuk elkészíteni a Mandelbrot halmazt beutazó programunkat. Ez a GUI az egyik legertékkelőbb grafikus interfésze a C++-nak, rengeteg tutorial van róla fent a neten.

Fordítás: Az szükséges 4 fájlnak egy mappában kell lennie. A mappában futtatni kell a qmake -project parancsot. Ez létre fog hozni egy *.pro fájlt. Ebbe a fájlba be kell írni a következőt: QT += widgets sort. Ezután futtatni kell a qmake *.pro. Ezután lesz a mappában egy Makefile, ezt kell majd használni. Kiadjuk a make parancsot, mely létrehoz egy bináris fájlt. Ezt pedig a szokásos módon futtatjuk. Ahhoz, hogy részletesebb képet kapj a ránagyított területről, az "n" billentyűt kell lenyomnod, mely kiszámlálja a z-ket a megadott területen. Ahogy folyamatosan nagyítjuk, észre vehetjük, hogy a újra meg újra Mandelbrot halmazokat kapunk.

Main.cpp

```
// main.cpp
#include <QApplication>
#include "frakablak.h"

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

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

/*
FrakAblak w1,
w2(-.08292191725019529, -.082921917244591272,
    -.9662079988595939, -.9662079988551173, 600, 3000),
w3(-.08292191724880625, -.0829219172470933,
    -.9662079988581493, -.9662079988563615, 600, 4000),
w4(.14388310361318304, .14388310362702217,
    .6523089200729396, .6523089200854384, 600, 38655);
w1.show();
w2.show();
w3.show();
w4.show();
*/
return a.exec();
}
```

Frakablak.cpp

```
// frakablak.cpp
//
// Mandelbrot halmaz nagyító

#include "frakablak.h"

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

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

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

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

```
}
```

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

```
void FrakAblak::paintEvent(QPaintEvent*)
{
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);
        if(!zX.empty()) //excuse me
        {
            for(int i=0; i<zX.size(); i++)
            {
                qpainter.drawLine(zX[i], zY[i], zX2[i], zY2[i]);
            }
        }
    }
    qpainter.end();
}
```

```
void FrakAblak::mousePressEvent(QMouseEvent* event)
{
    if (event->button() == Qt::LeftButton)
    {

        // A nagyítandó kijelölt területet bal felső sarka:
        x = event->x();
        y = event->y();
        mx = 0;
        my = 0;
    }
    else if(event->button() == Qt::RightButton)
    {
        double dx = (b-a)/szelesseg;
        double dy = (d-c)/magassag;
        double reC, imC, reZ, imZ, ujreZ, ujimZ;

        int iteracio = 0;

        reC = a+event->x()*dx;
        imC = d-event->y()*dy;

        reZ = 0;
```

```
imZ = 0;
iteracio = 0;

while(reZ*reZ + imZ*imZ < 4 && iteracio < 255) {
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ*reZ - imZ*imZ + reC;
    ujimZ = 2*reZ*imZ + imC;
    zX.push_back((int)((reZ - a)/dx));
    zY.push_back((int)((d - imZ)/dy));
    zX2.push_back((int)((ujreZ - a)/dx));
    zY2.push_back((int)((d - ujimZ)/dy));
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;
}

update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event) {

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

update();
}

void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {

if(szamitasFut)
    return;

szamitasFut = true;

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

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

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

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

update();
}

void FrakAblak::screenshot()
{
    QPainter draw(fraktal);
    draw.setPen(QColor(255,255,0));
    QString a_s = "a="+QString::number(a);
    QString b_s = "b="+QString::number(b);
    QString c_s = "c="+QString::number(c);
    QString d_s = "d="+QString::number(d);
    QString n_s = "n="+QString::number(iteraciosHatar);

    draw.drawText(10,15,a_s);
    draw.drawText(10,30,b_s);
    draw.drawText(10,45,c_s);
    draw.drawText(10,60,d_s);
    draw.drawText(10,75,n_s);

}

void FrakAblak::keyPressEvent (QKeyEvent *event)
{

    if(szamitasFut)
        return;

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

        delete mandelbrot;
        mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
            iteraciosHatar, this);
        mandelbrot->start();
    }
    else if (event->key() == Qt::Key_S) {
        screenshot();
        fraktal -> save("image.png");
    }
}

void FrakAblak::vissza(int magassag, int *sor, int meret)
{
```

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

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

Frakablak.h

```
#ifndef FRAKABLAK_H
#define FRAKABLAK_H

#include <QMainWindow>
#include <QImage>
#include <QPainter>
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"

class FrakSzal;

class FrakAblak : public QMainWindow
{
    Q_OBJECT

public:
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
               double d = 1.35, int szelesseg = 600,
               int iteraciosHatar = 255, QWidget *parent = 0);
    ~FrakAblak();
    void vissza(int magassag, int * sor, int meret) ;
    void vissza(void) ;
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;

protected:
    void paintEvent(QPaintEvent*) ;
```

```
void mousePressEvent (QMouseEvent*);  
void mouseMoveEvent (QMouseEvent*);  
void mouseReleaseEvent (QMouseEvent*);  
void keyPressEvent (QKeyEvent*);  
void screenshot();  
  
private:  
    QImage* fraktal;  
    FrakSzal* mandelbrot;  
    bool szamitasFut;  
    // A nagyítandó kijelölt területet bal felső sarka.  
    int x, y;  
    // A nagyítandó kijelölt terület szélessége és magassága.  
    int mx, my;  
    std::vector<int> zX, zY, zX2, zY2;  
};  
  
#endif // FRAKABLAK_H
```

Frakszal.cpp

```
// frakszal.cpp  
//  
// Mandelbrot halmaz rajzoló  
  
#include "frakszal.h"  
  
FrakSzal::FrakSzal(double a, double b, double c, double d,  
                    int szelesseg, int magassag, int iteraciosHatar, ←  
                    FrakAblak *frakAblak)  
{  
    this->a = a;  
    this->b = b;  
    this->c = c;  
    this->d = d;  
    this->szelesseg = szelesseg;  
    this->iteraciosHatar = iteraciosHatar;  
    this->frakAblak = frakAblak;  
    this->magassag = magassag;  
  
    egySor = new int[szelesseg];  
}  
  
FrakSzal::~FrakSzal()  
{  
    delete[] egySor;  
}  
  
void FrakSzal::run()  
{
```

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

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

            rez = ujrez;
            imZ = ujimZ;

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

        iteracio %= 256;
    }
    //a színezést viszont már majd a FrakAblak osztályban lesz
    egySor[k] = iteracio;
}
```

```
// Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.  
frakAblak->vissza(j, egySor, szelesseg);  
}  
frakAblak->vissza();  
}
```

Fraksal.h

```
#ifndef FRAKSZAL_H  
#define FRAKSZAL_H  
  
#include <QThread>  
#include <math.h>  
#include "frakablak.h"  
  
class FrakAblak;  
  
class FrakSzal : public QThread  
{  
    Q_OBJECT  
  
public:  
    FrakSzal(double a, double b, double c, double d,  
              int szelesseg, int magassag, int iteraciosHatar, FrakAblak * ←  
              frakAblak);  
    ~FrakSzal();  
    void run();  
  
protected:  
    // A komplex sík vizsgált tartománya [a,b]x[c,d].  
    double a, b, c, d;  
    // A komplex sík vizsgált tartományára feszített  
    // háló szélessége és magassága.  
    int szelesseg, magassag;  
    // Max. hány lépésig vizsgáljuk a  $z_{n+1} = z_n \cdot z_n + c$  iterációt?  
    // (tk. most a nagyítási pontosság)  
    int iteraciosHatar;  
    // Kinek számolok?  
    FrakAblak* frakAblak;  
    // Soronként küldöm is neki vissza a kiszámoltakat.  
    int* egySor;  
};  
  
#endif // FRAKSZAL_
```

```
laci@laci-GL503VD: ~/programok/mandel zoom cpp
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/mandel zoom cpp$ qmake -project
laci@laci-GL503VD:~/programok/mandel zoom cpp$
```

```
laci@laci-GL503VD: ~/programok/mandel zoom cpp
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/mandel zoom cpp$ qmake mandel\ zoom\ cpp.pro
Info: creating stash file /home/laci/programok/mandel zoom cpp/.qmake.stash
laci@laci-GL503VD:~/programok/mandel zoom cpp$ make
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_DEPRECATED_WARNINGS -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/x86_64-linux-gnu/qt5 -isystem /usr/include/x86_64-linux-gnu/qt5/QtWidgets -isystem /usr/include/x86_64-linux-gnu/qt5/QtGui -isystem /usr/include/x86_64-linux-gnu/qt5/QtCore -I. -isystem /usr/include/libdrm -I/usr/lib/x86_64-linux-gnu/qt5/mkspecs/linux-g++ -o frakablak.o frakablak.cpp
frakablak.cpp: In member function ‘virtual void FrakAblak::paintEvent(QPaintEvent*)’:
frakablak.cpp:44:27: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    for(int i=0; i<zX.size(); i++)
                           ^
frakablak.cpp: In member function ‘virtual void FrakAblak::mouseReleaseEvent(QMouseEvent*)’:
frakablak.cpp:108:48: warning: unused parameter ‘event’ [-Wunused-parameter]
    void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {
                           ^
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_DEPRECATED_WARNINGS -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/x86_64-linux-gnu/qt5 -isystem /usr/include/x86_64-linux-gnu/qt5/QtWidgets -isystem /usr/include/x86_64-linux-gnu/qt5/QtGui -isystem /usr/include/x86_64-linux-gnu/qt5/QtCore -I. -isystem /usr/include/libdrm -I/usr/lib/x86_64-linux-gnu/qt5/mkspecs/linux-g++ -o frakablak.o frakablak.cpp
```

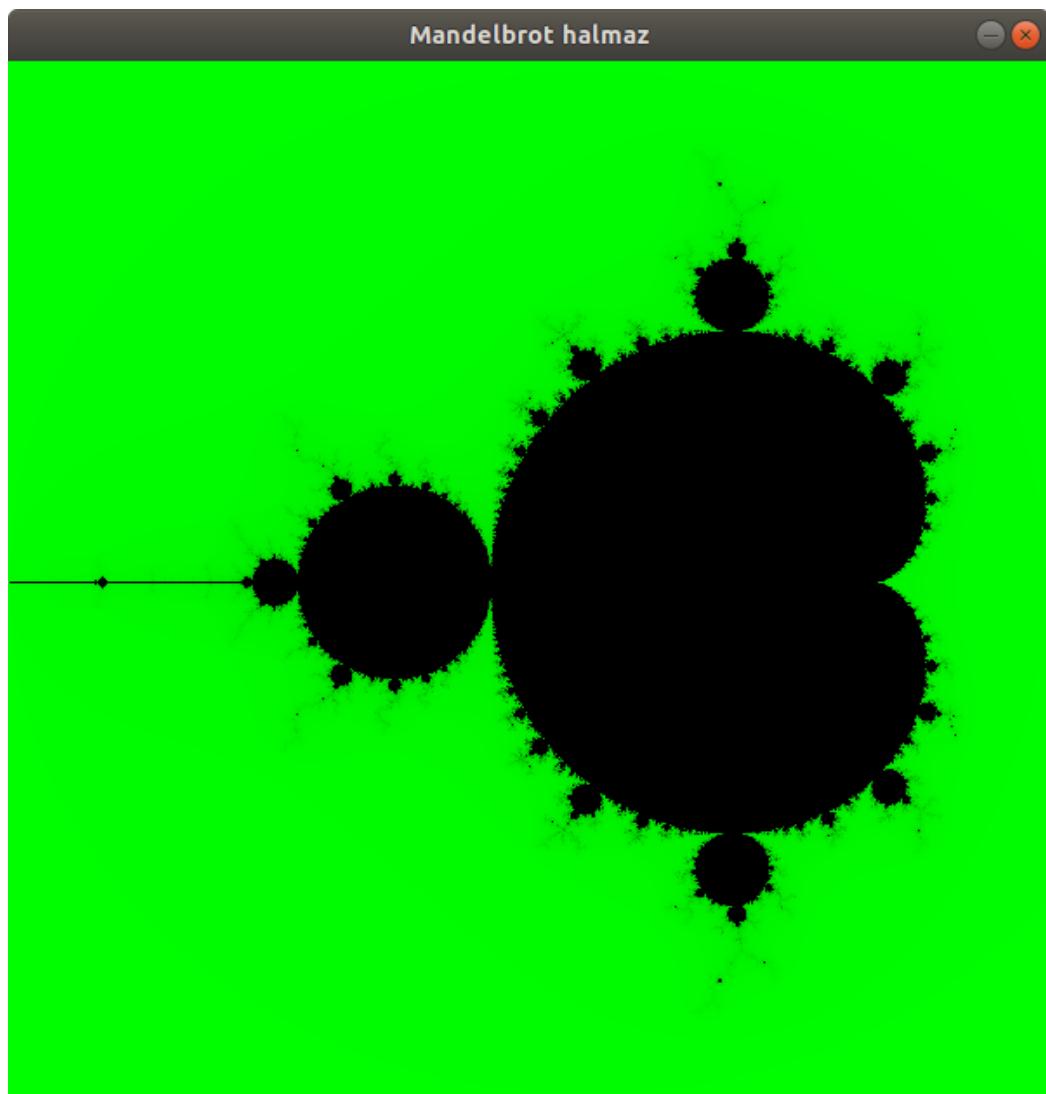
```
laci@laci-GL503VD: ~/programok/mandel zoom.cpp
Fájl Szerkesztés Nézet Keresés Terminál Súgó
GNU nano 2.9.3                         mandel zoom.cpp.pro

#####
# Automatically generated by qmake (3.1) Thu Apr 2 14:57:25 2020
#####

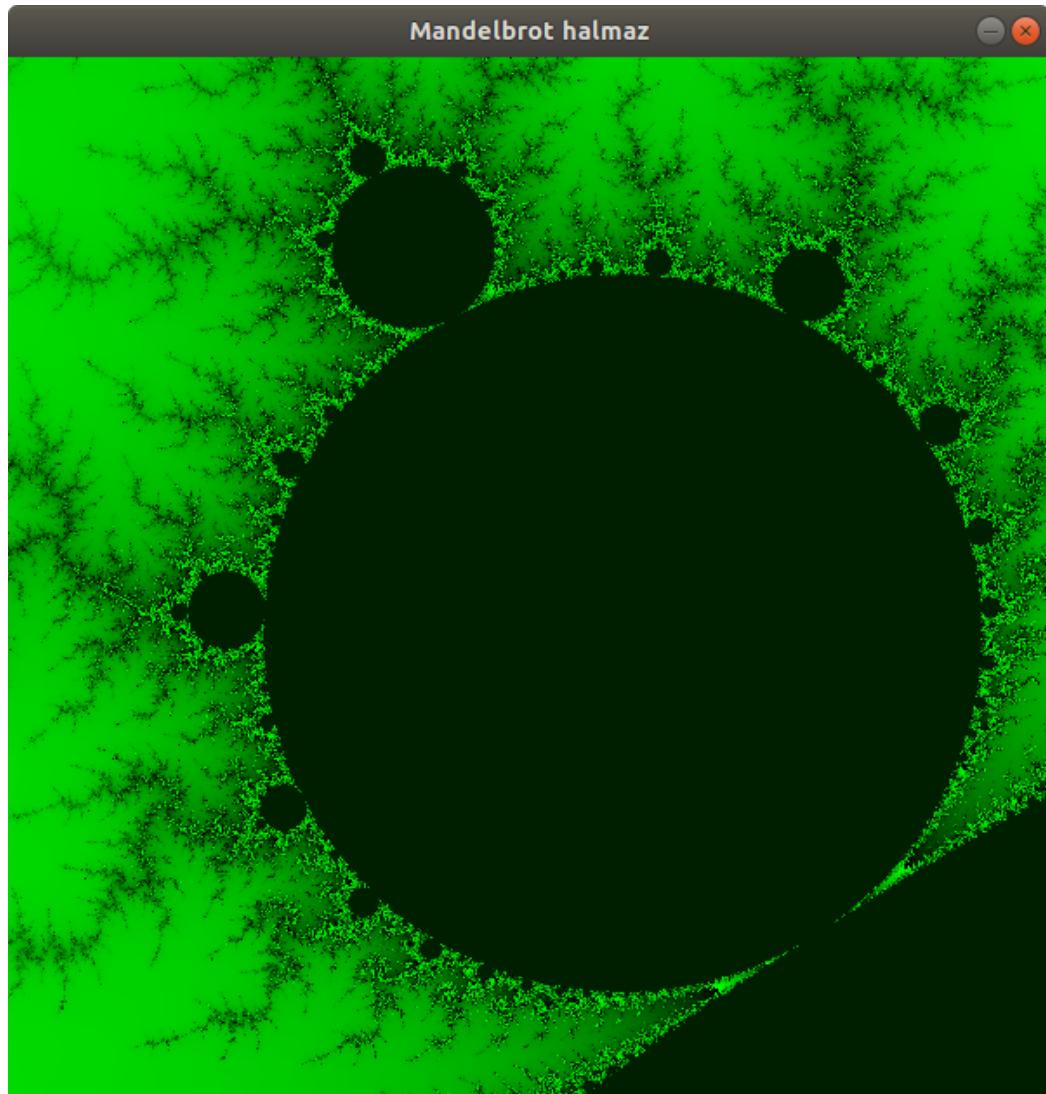
TEMPLATE = app
TARGET = "mandel zoom.cpp"
INCLUDEPATH += .

# The following define makes your compiler warn you if you use any
# feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS
QT += widgets
# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version o$
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs de$ [ 22 sor beolvasva ]
^G Súgó      ^O Kiírás     ^W Keresés   ^K Kivágás   ^J Sorkizárá$ C Pozíció
^X Kilépés   ^R Beolvasás  ^\ Csere      ^U Beilleszté^T Helyes-e? ^ Ugrás sorra
```

```
laci@laci-GL503VD: ~/programok/mandel zoom.cpp
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/mandel zoom.cpp$ ./mandel\ zoom\ cpp
```



DRAFT



5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Uj3B6IJnssY>, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmaz_nagyito_uttasozo_javaval/

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

MandelbrotHalmazNagyító.java

```
/*
 * MandelbrotHalmazNagyító.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 */
```

```
* @version 0.0.1
*/
import java.util.*;
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /** A nagyítandó kijelölt területet bal felső sarka. */
    private int x, y;
    /** A nagyítandó kijelölt terület szélessége és magassága. */
    private int mx, my;
    private List<Integer> zX = new ArrayList<>();
    private List<Integer> zX2 = new ArrayList<>();
    private List<Integer> zY = new ArrayList<>();
    private List<Integer> zY2 = new ArrayList<>();

    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nyígtani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     *
     * @param a           a [a,b]x[c,d] tartomány a koordinátája.
     * @param b           a [a,b]x[c,d] tartomány b koordinátája.
     * @param c           a [a,b]x[c,d] tartomány c koordinátája.
     * @param d           a [a,b]x[c,d] tartomány d koordinátája.
     * @param szélesség   a halmazt tartalmazó tömb szélessége.
     * @param iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
                                     int szélesség, int iterációsHatár) {
        // Az ős osztály konstruktörának hívása
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        // Egér kattintó események feldolgozása:
        addMouseListener(new java.awt.event.MouseAdapter() {
            // Egér kattintással jelöljük ki a nagyítandó területet
            // bal felső sarkát:
            public void mousePressed(java.awt.event.MouseEvent m) {
                if (m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
                    // A nagyítandó kijelölt területet bal felső sarka:
                    x = m.getX();
                    y = m.getY();
                    mx = 0;
                    my = 0;
                }
                else if(m.getButton() == java.awt.event.MouseEvent.BUTTON3) {
                    double dx = (b-a)/szélesség;
                    double dy = (d-c)/szélesség;
                    double reC, imC, reZ, imZ, ujreZ, ujimZ;

                    int iteracio = 0;
```

```
    reC = a+m.getX()*dx;
    imC = d-m.getY()*dy;

    reZ = 0;
    imZ = 0;
    iteracio = 0;

    while(reZ*reZ + imZ*imZ < 4 && iteracio < 255) {
        // z_{n+1} = z_n * z_n + c
        ujreZ = reZ*reZ - imZ*imZ + reC;
        ujimZ = 2*reZ*imZ + imC;
        zX.add((int)((reZ - a)/dx));
        zY.add( (int)((d - imZ)/dy));
        zX2.add((int)((ujreZ - a)/dx));
        zY2.add((int)((d - ujimZ)/dy));
        reZ = ujreZ;
        imZ = ujimZ;

        ++iteracio;
    }
    repaint();
}
// Vonszolva kijelölünk egy területet...
// Ha felengedjük, akkor a kijelölt terület
// újraszámítása indul:
public void mouseReleased(java.awt.event.MouseEvent m) {
    double dx = (MandelbrotHalmazNagyító.this.b
                  - MandelbrotHalmazNagyító.this.a)
                /MandelbrotHalmazNagyító.this.szélesség;
    double dy = (MandelbrotHalmazNagyító.this.d
                  - MandelbrotHalmazNagyító.this.c)
                /MandelbrotHalmazNagyító.this.magasság;
    // Az új Mandelbrot nagyító objektum elkészítése:
    if(m.getButton() == java.awt.event.MouseEvent.BUTTON3) {
        return;
    }
    dispose();
    new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+
                                  x*dx,
                                  MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
                                  MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
                                  MandelbrotHalmazNagyító.this.d-y*dy,
                                  600,
                                  MandelbrotHalmazNagyító.this.iterációsHatár);
}
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
```

```
public void mouseDragged(java.awt.event.MouseEvent m) {
    // A nagyítandó kijelölt terület szélessége és magassága:
    mx = m.getX() - x;
    my = mx;
    repaint();
}
});
}
/** 
 * Pillanatfelvétellek készítése.
 */
public void pillanatfelvétel() {
    // Az elmentendő kép elkészítése:
    java.awt.image.BufferedImage mentKép =
        new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKép.getGraphics();
    g.drawImage(kép, 0, 0, this);
    g.setColor(java.awt.Color.YELLOW);
    g.drawString("a=" + a, 10, 15);
    g.drawString("b=" + b, 10, 30);
    g.drawString("c=" + c, 10, 45);
    g.drawString("d=" + d, 10, 60);
    g.drawString("n=" + iterációsHatár, 10, 75);
    if (számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    g.setColor(java.awt.Color.WHITE);
    g.drawRect(x, y, mx, my);
    if (!zX.isEmpty())
    {
        for (int i = 0; i<zX.size(); ++i){
            g.drawLine(zX.get(i), zY.get(i), zX2.get(i), zY2.get(i));
        }
    }
    g.dispose();
    // A pillanatfelvétel képfájl nevének képzése:
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrothalmazNagyitas_");
    sb.append(++pillanatfelvételszámLálo);
    sb.append("_");
    // A fájl nevébe belelevesszük, hogy melyik tartományban
    // találtuk a halmazt:
    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
}
```

```
sb.append("_");
sb.append(d);
sb.append(".png");
// png formátumú képet mentünk
try {
    javax.imageio.ImageIO.write(mentKép, "png",
        new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
    e.printStackTrace();
}
}
/***
 * A nagyítandó kijelölt területet jelző négyzet kirajzolása.
 */
public void paint(java.awt.Graphics g) {
    // A Mandelbrot halmaz kirajzolása
    g.drawImage(kép, 0, 0, this);
    // Ha éppen fut a számítás, akkor egy vörös
    // vonallal jelöljük, hogy melyik sorban tart:
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    // A jelző négyzet kirajzolása:
    g.setColor(java.awt.Color.WHITE);
    g.drawRect(x, y, mx, my);
    if (!zX.isEmpty())
    {
        for (int i = 0; i<zX.size(); ++i){
            g.drawLine(zX.get(i), zY.get(i), zX2.get(i), zY2.get(i));
        }
    }
}
/***
 * Példányosít egy Mandelbrot halmazt nagyító obektumot.
 */
public static void main(String[] args) {
    // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
    // tartományában keressük egy 600x600-as hálóval és az
    // aktuális nagyítási pontossággal:
    new MandelbrothalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
}
```

MandelbrotHalmaz.java

```
/*
 * MandelbrotHalmaz.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
```

```
*  
*/  
/**  
 * A Mandelbrot halmazt kiszámoló és kirajzoló osztály.  
 *  
 * @author Bátfa Norbert, nbatfai@inf.unideb.hu  
 * @version 0.0.1  
 */  
import java.awt.Color;  
public class MandelbrotHalmaz extends java.awt.Frame implements Runnable {  
    /** A komplex sík vizsgált tartománya [a,b]x[c,d]. */  
    protected double a, b, c, d;  
    /** A komplex sík vizsgált tartományára feszített  
     * háló szélessége és magassága. */  
    protected int szélesség, magasság;  
    /** A komplex sík vizsgált tartományára feszített hálónak megfelelő kép ←  
     */  
    protected java.awt.image.BufferedImage kép;  
    /** Max. hány lépésig vizsgáljuk a  $z_{n+1} = z_n \cdot z_n + c$  iterációt?  
     * (tk. most a nagyítási pontosság) */  
    protected int iterációsHatár = 255;  
    /** Jelzi, hogy éppen megy-e a számítás? */  
    protected boolean számításFut = false;  
    /** Jelzi az ablakban, hogy éppen melyik sort számoljuk. */  
    protected int sor = 0;  
    /** A pillanatfelvételek számozásához. */  
    protected static int pillanatfelvételSzámláló = 0;  
    /**  
     * Létrehoz egy a Mandelbrot halmazt a komplex sík  
     * [a,b]x[c,d] tartománya felett kiszámoló  
     * <code>MandelbrotHalmaz</code> objektumot.  
     *  
     * @param a a [a,b]x[c,d] tartomány a koordinátája.  
     * @param b a [a,b]x[c,d] tartomány b koordinátája.  
     * @param c a [a,b]x[c,d] tartomány c koordinátája.  
     * @param d a [a,b]x[c,d] tartomány d koordinátája.  
     * @param szélesség a halmazt tartalmazó tömb szélessége.  
     * @param iterációsHatár a számítás pontossága.  
     */  
    public MandelbrotHalmaz(double a, double b, double c, double d,  
        int szélesség, int iterációsHatár) {  
        this.a = a;  
        this.b = b;  
        this.c = c;  
        this.d = d;  
        this.szélesség = szélesség;  
        this.iterációsHatár = iterációsHatár;  
        // a magasság az  $(b-a) / (d-c) = szélesség / magasság$   
        // arányból kiszámolva az alábbi lesz:  
        this.magasság = (int)(szélesség * ((d-c)/(b-a)));  
    }
```

```
// a kép, amire rárajzoljuk majd a halmazt
kép = new java.awt.image.BufferedImage(szélesség, magasság,
                                         java.awt.image.BufferedImage.TYPE_INT_RGB);
// Az ablak bezárásakor kilépünk a programból.
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent e) {
        setVisible(false);
        System.exit(0);
    }
});
// A billentyűzetről érkező események feldolgozása
addKeyListener(new java.awt.event.KeyAdapter() {
    // Az 's', 'n' és 'm' gombok lenyomását figyeljük
    public void keyPressed(java.awt.event.KeyEvent e) {
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel();
        // Az 'n' gomb benyomásával pontosabb számítást végzünk.
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            if(számításFut == false) {
                MandelbrotHalmaz.this.iterációsHatár += 256;
                // A számítás újra indul:
                számításFut = true;
                new Thread(MandelbrotHalmaz.this).start();
            }
            // Az 'm' gomb benyomásával pontosabb számítást végzünk,
            // de közben sokkal magasabbra vesszük az iterációs
            // határt, mint az 'n' használata esetén
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_M) {
            if(számításFut == false) {
                MandelbrotHalmaz.this.iterációsHatár += 10*256;
                // A számítás újra indul:
                számításFut = true;
                new Thread(MandelbrotHalmaz.this).start();
            }
        }
    }
});
// Ablak tulajdonságai
setTitle("A Mandelbrot halmaz");
setResizable(false);
setSize(szélesség, magasság);
setVisible(true);
// A számítás indul:
számításFut = true;
new Thread(this).start();
}
/**
 * A halmaz aktuális állapotának kirajzolása.
 */
public void paint(java.awt.Graphics g) {
```

```
// A Mandelbrot halmaz kirajzolása
g.drawImage(kép, 0, 0, this);
// Ha éppen fut a számítás, akkor egy vörös
// vonallal jelöljük, hogy melyik sorban tart:
if(számításFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
    paint(g);
}
/**
 * Pillanatfelvételek készítése.
 */
public void pillanatfelvétel() {
    // Az elmentendő kép elkészítése:
    java.awt.image.BufferedImage mentKép =
        new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKép.getGraphics();
    g.drawImage(kép, 0, 0, this);
    g.setColor(java.awt.Color.YELLOW);
    g.drawString("a=" + a, 10, 15);
    g.drawString("b=" + b, 10, 30);
    g.drawString("c=" + c, 10, 45);
    g.drawString("d=" + d, 10, 60);
    g.drawString("n=" + iterációsHatár, 10, 75);
    g.dispose();
    // A pillanatfelvétel képfájl nevének képzése:
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrotHalmaz_");
    sb.append(++pillanatfelvételszámláló);
    sb.append("_");
    // A fájl nevébe belelevesszük, hogy melyik tartományban
    // találtuk a halmazt:
    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
    sb.append("_");
    sb.append(d);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(mentKép, "png",
```

```
        new java.io.File(sb.toString())));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
/***
 * A Mandelbrot halmaz számítási algoritmusa.
 * Az algoritmus részletes ismertetését lásd például a
 * [BARNESLEY KÖNYV] (M. Barnsley: Fractals everywhere,
 * Academic Press, Boston, 1986) hivatkozásban vagy
 * ismeretterjesztő szinten a [CSÁSZÁR KÖNYV] hivatkozásban.
 */
public void run() {
    // A [a,b]x[c,d] tartományon milyen sűrű a
    // megadott szélesség, magasság háló:
    double dx = (b-a)/szélesség;
    double dy = (d-c)/magasság;
    double reC, imC, reZ, imZ, ujreZ, ujimZ;
    int rgb;
    // Hány iterációt csináltunk?
    int iteráció = 0;
    // Végigzongorázzuk a szélesség x magasság hálót:
    for(int j=0; j<magasság; ++j) {
        sor = j;
        for(int k=0; k<szélesség; ++k) {
            // c = (reC, imC) a háló rácspontjainak
            // megfelelő komplex szám
            reC = a+k*dx;
            imC = d-j*dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteráció = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértek, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while(reZ*reZ + imZ*imZ < 4 && iteráció < iterációsHatár) {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ*reZ - imZ*imZ + reC;
                ujimZ = 2*reZ*imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;

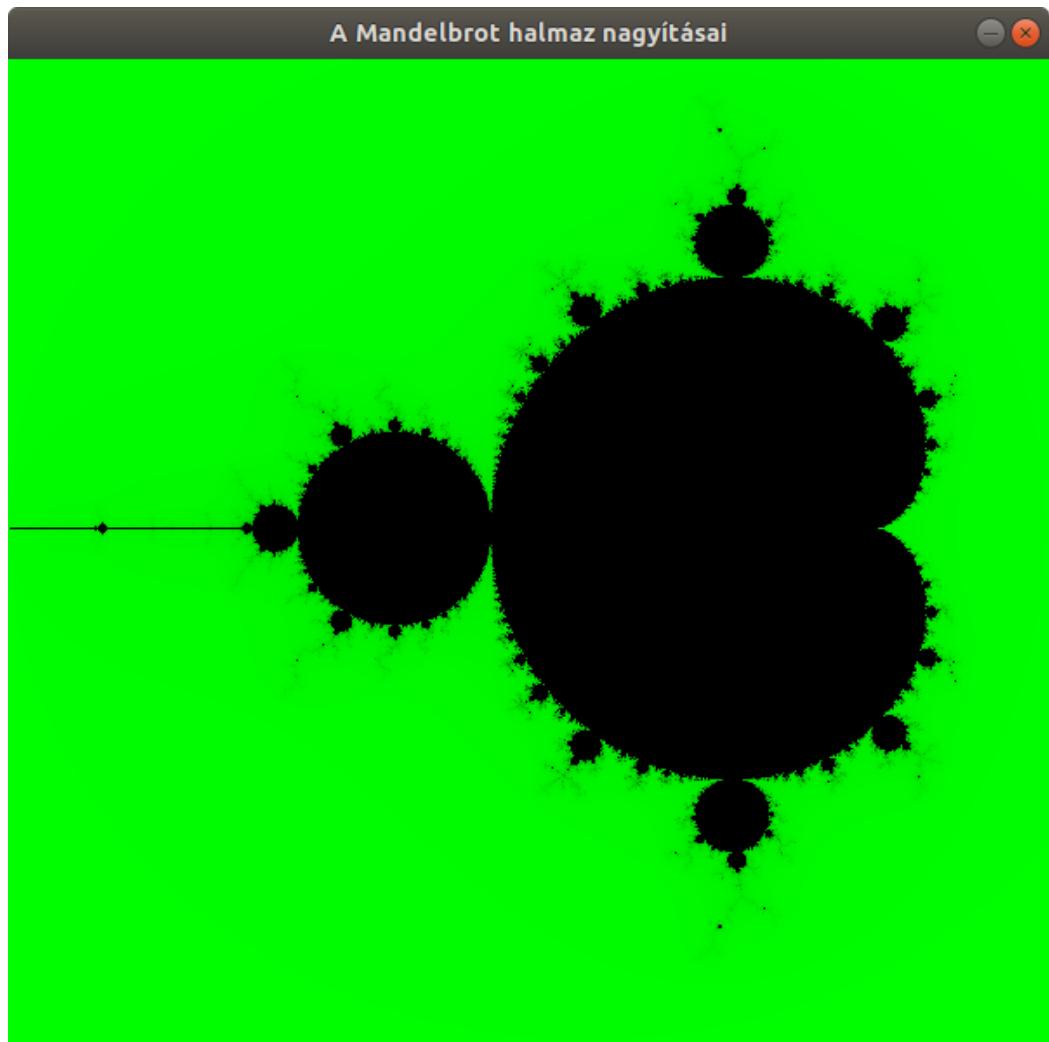
                ++iteráció;
            }
        }
    }
}
```

```
// ha a < 4 feltétel nem teljesült és a
// iteráció < iterációsHatár sérülésével lépett ki, azaz
// feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c
// sorozat konvergens, azaz iteráció = iterációsHatár
// ekkor az iteráció %= 256 egyenlő 255, mert az esetleges
// nagyítások során az iteráció = valahány * 256 + 255
// iteráció %= 256;
// így a halmaz elemeire 255-255 értéket használjuk,
// azaz (Red=0,Green=0,Blue=0) fekete színnel:
Color bg = new Color(0, 255-iteráció, 0);
rgb = bg.getRGB();
// rajzoljuk a képre az éppen vizsgált pontot:
kép.setRGB(k, j, rgb);
}
repaint();
}
számításFut = false;
}
/**
 * Példányosít egy Mandelbrot halmazt kiszámoló obektumot.
 */
public static void main(String[] args) {
    // A halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35] tartományában
    // keressük egy 400x400-as hálóval:
    new MandelbrotHalmaz(-2.0, .7, -1.35, 1.35, 600, 255);
}
}
```

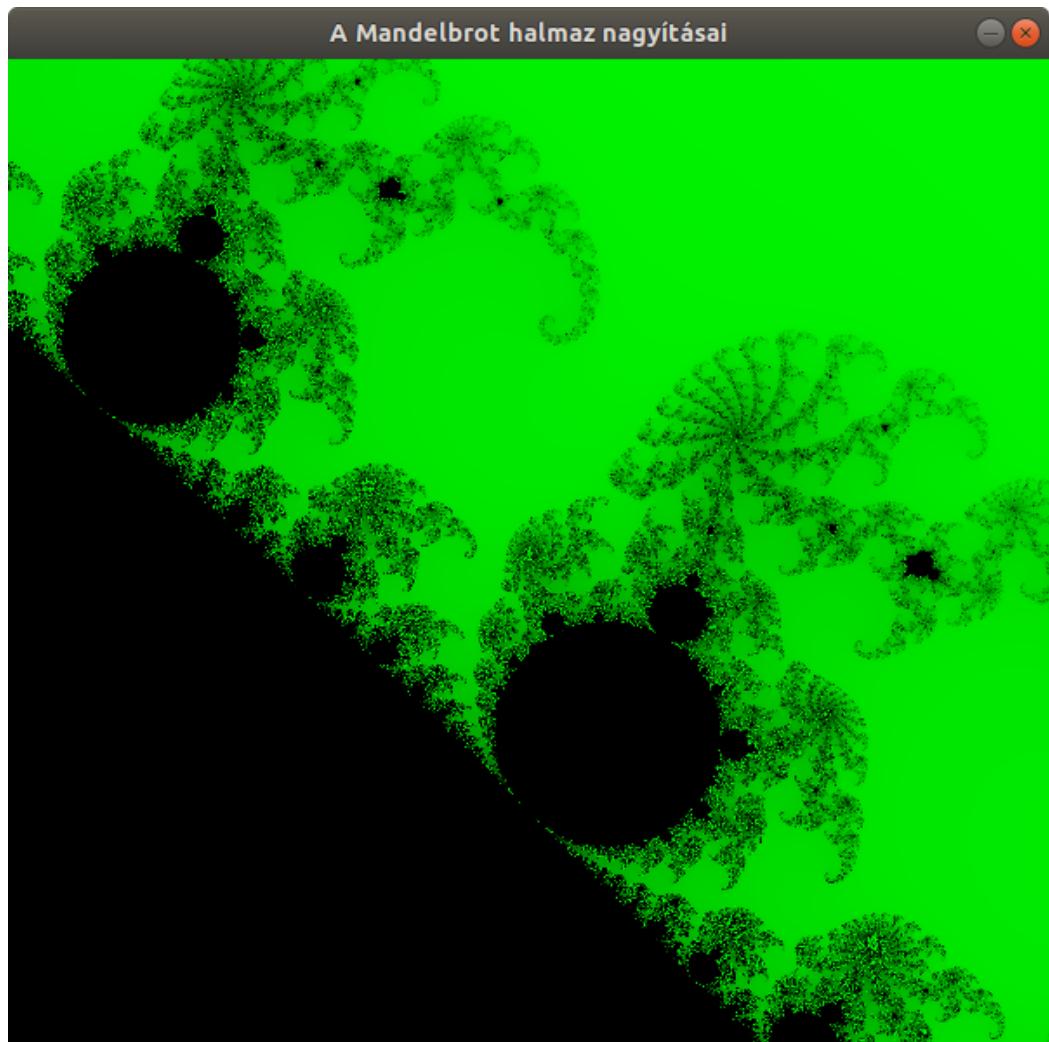


```
laci@laci-GL503VD: ~/programok/mandel_zoom_java
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/mandel_zoom_java$ javac MandelbrotHalmazNagyító.java
laci@laci-GL503VD:~/programok/mandel_zoom_java$ java MandelbrotHalmazNagyító
```

DRY



DRAFT



Dísz

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

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

//Random osztály
class PolarGen {

public:
    PolarGen(); //konstruktor
    ~PolarGen(){} //destruktur
    double kovetkezo(); //random lekérés

private:
    bool nincsTarolt;
    double tarolt; //random értéke
```

```
};
```

Include-oljuk a szükséges osztályokat ami újnak tűnhetnek azok a ctime és ctime az elsőbe a nevéből eredően matematikai műveleteket importálunk a ctime-al le lehet kérni a rendszeridőt meg dátum formátumot átlehet alakítani string formátumba. Most hogy átnéztük a header-öket lássunk is bele mi a public és a private között a különbség .A public-ba írt tagok elérhetőek a class-on kívül is ellentéte a private meg csak a classon belül érhetőek el. A konstruktor az egy olyan függvény ami csak akkor megy végbe amikor létrehozzunk a PolarGen típusú objektumokat. Ez az állítás a destruktora is igaz csak ez a program végén fut le.

```
PolarGen::PolarGen() { //a konstruktor kifejtése
    nincsTarolt = false;
    std::srand (std::time(NULL)); //random inicializálás
}
```

Ebbe a konstruktorba adunk egy alap értéket a nincsTarolt és meghívjuk a srand() függvényt ami generál egy random számot.

```
double PolarGen::kovetkezo() { //random lekérő függvény kifejtése
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;

        do{
            u1 = std::rand () / (RAND_MAX + 1.0); //innentől jön az algoritmus
            u2 = std::rand () / (RAND_MAX + 1.0);
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);

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

        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

        return r * v1; //idáig tart az algoritmus
    }

    else
    {
        nincsTarolt = !nincsTarolt; //ha van korábbi random érték, akkor azt ←
            adja vissza
        return tarolt;
    }
};
```

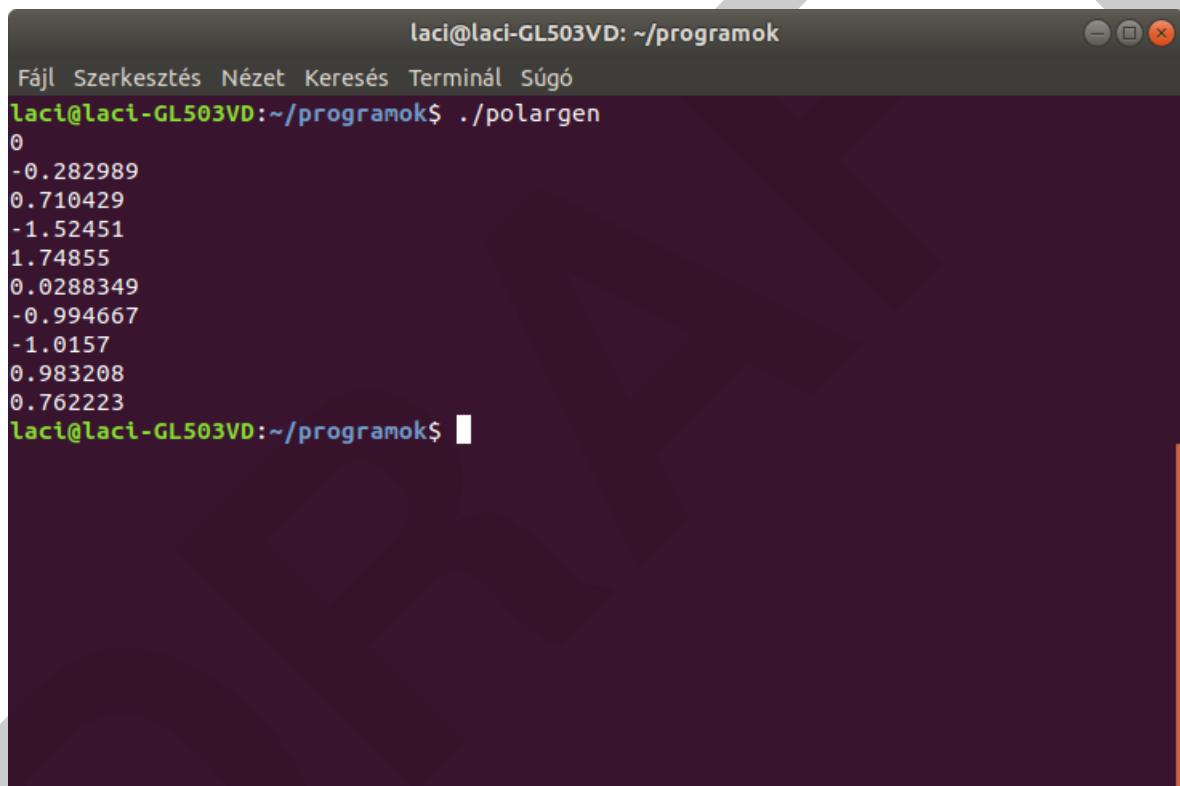
A kovetkezo() függvényünkbe van egy algoritmus mellyel ellenőrizzük hogy van-e tárolt random számunk ha nincs létrehozzunk 2-őt az egyiket visszaadjuk a másikat eltároljuk.

```
int main()
{
    PolarGen rnd;

    for (int i = 0; i < 10; ++i) std::cout << rnd.kovetkezo() << std::endl; ←
        //10 random szám generálása

}
```

A mainbe létrehozzuk a PolarGen típusú változót és létrehozunk 10 random számot.



```
laci@laci-GL503VD:~/programok$ ./polargen
0
-0.282989
0.710429
-1.52451
1.74855
0.0288349
-0.994667
-1.0157
0.983208
0.762223
laci@laci-GL503VD:~/programok$
```

6.1. ábra. C++ Run

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

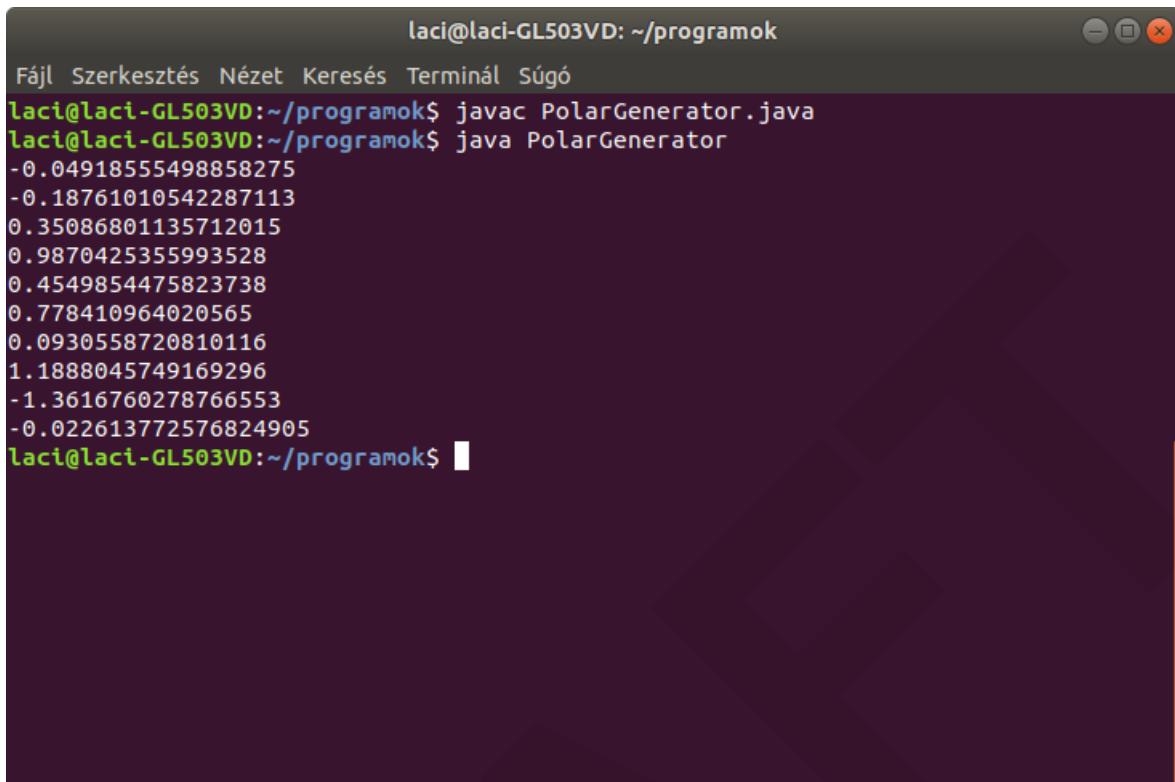
    public PolarGenerator()
    {
        nincsTarolt = true;
    }
```

```
public double kovetkezo()
{
    if(nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do{
            u1 = Math.random();
            u2 = Math.random();
            v1 = 2* u1 -1;
            v2 = 2* u2 -1;
            w = v1*v1 + v2*v2;
        } while (w>1);

        double r = Math.sqrt((-2 * Math.log(w) / w));
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
        return r * v1;
    }
    else
    {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
}

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





A screenshot of a terminal window titled "laci@laci-GL503VD: ~/programok". The window shows the following command-line session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok$ javac PolarGenerator.java
laci@laci-GL503VD:~/programok$ java PolarGenerator
-0.04918555498858275
-0.18761010542287113
0.35086801135712015
0.9870425355993528
0.4549854475823738
0.778410964020565
0.0930558720810116
1.1888045749169296
-1.3616760278766553
-0.022613772576824905
laci@laci-GL503VD:~/programok$
```

6.2. ábra. Java Run

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

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

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

A header-öket nem is magyarázom mert már ismerjük őket. Elsőnek amit látunk létrehozunk egy struktúrát binfa néven ami tartalmaz 3 értéket a bal és jobb gyermekét és a gyermekekre mutató mutatót.

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

Az uh_elem() függvény az foglal egy helyet a BINFA tipusú változónak majd a return-be visszaadunk egy üres területre mutató pointert.

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

Deklarálunk néhány statisztikára használható függvényt.

```
int  
main (int argc, char **argv)  
{  
    char b;  
  
    BINFA_PTR gyoker = uj_elem ();  
    gyoker->ertek = '/';  
    gyoker->bal nulla = gyoker->jobb_egy = NULL;  
    BINFA_PTR fa = gyoker;
```

Itt létrehozzuk a gyökeret a gyökér értéket per jelre állítjuk és a bal és jobb gyereket null-ra állítjuk mivel még üres a fánk és vegül a fa mutatót a gyökérre állítjuk.

```
while (read (0, (void *) &b, 1))  
{  
//     write (1, &b, 1);  
    if (b == '0')  
    {  
        if (fa->bal nulla == NULL)  
        {  
            fa->bal nulla = uj_elem ();  
            fa->bal nulla->ertek = 0;  
            fa->bal nulla->bal nulla = fa->bal nulla->jobb_egy = NULL;  
            fa = gyoker;  
        }  
    }
```

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

A while ciklusban hozzuk létre a binfákat. A standard inputól fogadjuk a bemenetet bitenként. Ha a bemenet 0, akkor megnézzük hogy van-e nullás gyermeké ha nincs a akkor kreálunk egyet és a fa mutatót visszaállítjuk a gyökérre.Ha a bemenet nem 0 akkor ellenőrizzük hogy van jobb oldali gyermeké ha nincs létrehozzuk és ismét a fa mutatót visszaállítjuk a gyökérre de ha van akkor a mutató a jobb oldali gyermekre fog mutatni.

```
printf ("\n");
kiir (gyoker);

extern int max_melyseg, atlagosszeg, melyseg, atlagdb;
extern double szorasosszeg, atlag;

printf ("melyseg=%d\n", max_melyseg-1);

/* Átlagos ághossz kiszámítása */
atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
ratlag (gyoker);
// atlag = atlagosszeg / atlagdb;
// (int) / (int) "elromlik", ezért casoljuk
// K&R tudatlansági védelem miatt a sok () :)
atlag = ((double)atlagosszeg) / atlagdb;

/* Ághosszak szórásának kiszámítása */
atlagosszeg = 0;
```

```
melyseg = 0;
atlagdb = 0;
szorasosszeg = 0.0;

rszoras (gyoker);

double szoras = 0.0;

if (atlagdb - 1 > 0)
    szoras = sqrt( szorasosszeg / (atlagdb - 1));
else
    szoras = sqrt (szorasosszeg);

printf ("altag=%f\nszoras=%f\n", atlag, szoras);

szabadit (gyoker);
}
```

A mainbe létrehozunk egy pár változót amibe statisztikai adatokat fogunk tárolni.

```
int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

void
ratlag (BINFA_PTR fa)
{

if (fa != NULL)
{
    ++melyseg;
    ratlag (fa->jobb_egy);
    ratlag (fa->bal_nulla);
    --melyseg;

    if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
    {

        ++atlagdb;
        atlagosszeg += melyseg;
    }
}
}
```

Itt a fa átlagját számoljuk ki.

```
// a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
// http://sourceforge.net/projects/javacska/
```

```
// az atlag() hivasakor is inicializálni kell őket, a
// a rekurzív bejáras használja
double szorasosszeg = 0.0, atlag = 0.0;

void
rszoras (BINFA_PTR fa)
{

    if (fa != NULL)
    {
        ++melyseg;
        rszoras (fa->jobb_egy);
        rszoras (fa->bal nulla);
        --melyseg;

        if (fa->jobb_egy == NULL && fa->bal nulla == NULL)
        {

            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}
```

Itt meg a szórást.

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

void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
        kiir (elem->jobb_egy);
        // ez a postorder bejáráshez képest
        // 1-el nagyobb mélység, ezért -1
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertekek < 2 ? '0' + elem->ertekek : elem->ertekek ↔
                ,
                melyseg-1);
        kiir (elem->bal nulla);
        --melyseg;
    }
}
```

}

A kiír segítségével fogjuk a fánkat kiírni standard outputra most inorder bejárást fogunk használni.

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

Szabadít rész a programunk destruktora.A lefoglalt tárterületet a free() függvényel szabadítjuk fel.De mi előtt a szülőtől eltávolítjuk előtte a gyerekeket kell persz.

```
laci@laci-GL503VD: ~/programok/oldbinfa
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/oldbinfa$ gcc alap.c -o alap
laci@laci-GL503VD:~/programok/oldbinfa$ ./alap <t.txt
0100010
1010001
2323525671
346319653

-----1(7)
-----1(6)
-----1(5)
-----1(4)
-----1(3)
-----1(2)
---/(1)
-----1(4)
-----1(3)
-----0(4)
---0(2)
-----1(4)
-----0(3)
melyseg=7laci@laci-GL503VD:~/programok/oldbinfa$
```

6.3. ábra. Futtatás

6.3. Fabejárás

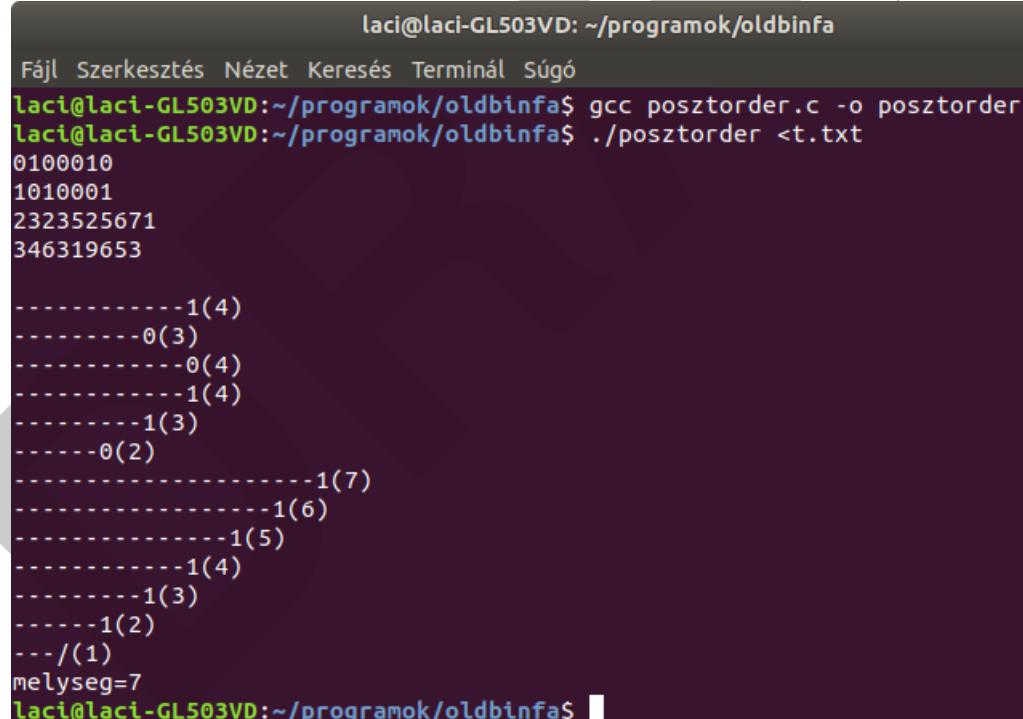
Járd be az előző (inorder bejárású) fát pre- és posztorder is!

Megoldás videó:

Megoldás forrása:

Posztorder Bejárás

```
void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
        kiir (elem->jobb_egy);
        kiir (elem-> bal_nulla)
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' : elem->ertek ↔
                , melyseg-1);
        --melyseg;
    }
}
```



The screenshot shows a terminal window titled "laci@laci-GL503VD: ~/programok/oldbinfa". The terminal displays the following output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/oldbinfa$ gcc posztorder.c -o posztorder
laci@laci-GL503VD:~/programok/oldbinfa$ ./posztorder <t.txt
0100010
1010001
2323525671
346319653

-----1(4)
----0(3)
----0(4)
----1(4)
---1(3)
--0(2)
-----1(7)
-----1(6)
-----1(5)
-----1(4)
-----1(3)
-----1(2)
---/(1)
melyseg=7
laci@laci-GL503VD:~/programok/oldbinfa$
```

6.4. ábra. Futtatás

Preorder Bejárás

```
void
kiir (BINFA_PTR elem)
{
if (elem != NULL)
{
++melyseg;
if (melyseg > max_melyseg)
max_melyseg = melyseg;
for (int i = 0; i < melyseg; ++i)
printf ("---");
printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek --
,
melyseg-1);
kiir (elem->jobb_egy);
// ez a postorder bejáráshoz képest
// 1-el nagyobb mélység, ezért -1
kiir (elem->bal nulla);
--melyseg;
}
}
```

```
laci@laci-GL503VD: ~/programok/oldbinfa
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/oldbinfa$ gcc preorder.c -o preorder
laci@laci-GL503VD:~/programok/oldbinfa$ ./preorder <t.txt
0100010
1010001
2323525671
346319653

---/(1)
-----0(2)
-----0(3)
-----1(4)
-----1(3)
-----0(4)
-----1(4)
-----1(2)
-----1(3)
-----1(4)
-----1(5)
-----1(6)
-----1(7)
melyseg=7
laci@laci-GL503VD:~/programok/oldbinfa$
```

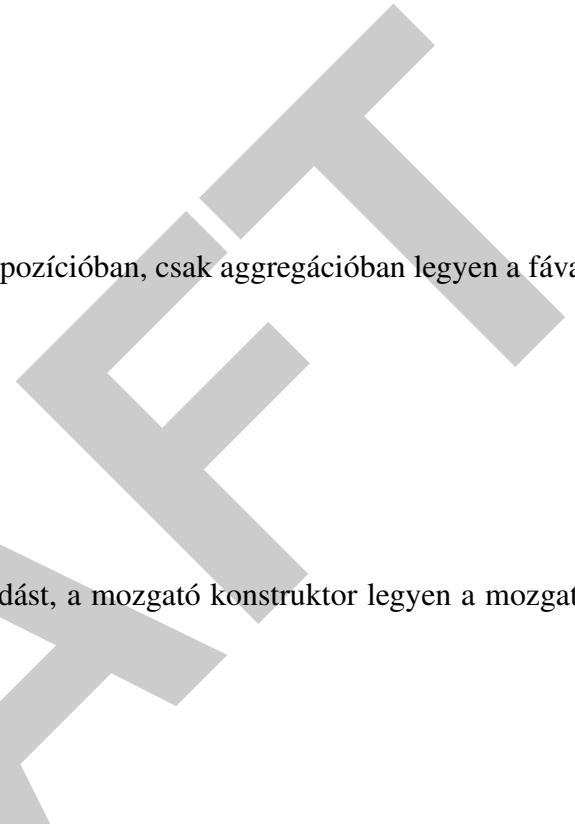
6.5. ábra. Futtatás

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:



6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

6.4-6.6

```
#include <iostream>

template <typename ValueType>
class BinTree {

protected:
    class Node {

private:
    ValueType value;
    Node *left;
    Node *right;
    int count{0};

    // TODO rule of five
    Node(const Node &);
    Node & operator=(const Node &);
    Node(Node &&);
    Node & operator=(Node &&);

public:
```

```

Node(ValueType value): value(value), left(nullptr), right(nullptr) ←
{
    ValueType getValue(){return value;}
    Node * leftChild(){return left;}
    Node * rightChild(){return right;}
    void leftChild(Node * node){left = node;}
    void rightChild(Node * node){right = node;}
    int getCount(){return count;}
    void incCount(){++count;}
};

Node *root;
Node *treep;
int depth{0};

```

Elsőnek includeoljuk az iostream-et utána létrehozunk egy template osztályt aminek adunk egy ValueType tipus paramétert amire a nodokba lévő értékekre mutatunk. Protected részbe csinálunk egy node osztályt. Private részben definiáljuk a ValueType-unkat value-re létrehozunk a bal és jobb gyermekre pointereket és létrehozunk egy számlálót "programozó pukkasztó" stílusban. TODO résznél felvesszük a Node másoló konstruktort ami kap egy konstants referenciát és ezt eljátszik mégegyszer a jobb gyermek referenciaira is. A public elsőnek megírjuk a node konstuktorát aminek megadjuk a value értékét tipust a bal és jobb gyermeket. Csinálunk egy node gyökér mutatót és egy treep mutatót plusz egy mélység számlálót.

```

private:
    // TODO rule of five
    BinTree(const BinTree &);
    BinTree & operator=(const BinTree &);
    BinTree(const BinTree &&);
    BinTree & operator=(const BinTree &&);

public:
    BinTree(Node *root = nullptr, Node *treep = nullptr): root(root), treep ←
        (treep) {}
    ~BinTree() {
        deltree(root);
    }
    BinTree & operator<<(ValueType value);
    void print(){print(root, std::cout);}
    void print(Node *node, std::ostream & os);
    void deltree(Node *node);

};

```

Privátban újra elvégezzük amit részleteztem csak a BinTree -re. Publicban BinTree nek megadunk 2 alapértelmezett értéket és root-ba berakom a root pointert és így járok el a treep-el is. Példűnyosítjuk a BinTree-t és a destrukturát.

```

template <typename ValueType>
class ZLWTtree : public BinTree<ValueType> {

```

```
public:  
    ZLWTree(): BinTree<ValueType>(new typename BinTree<ValueType>::Node ←  
        ('/')) {  
        this->treep = this->root;  
    }  
    ZLWTree & operator<<(ValueType value);  
};
```

Itt megadjuk hogy a fát / jellet legyen feldarabolva

```
template <typename ValueType>  
BinTree<ValueType> & BinTree<ValueType>::operator<<(ValueType value)  
{  
    if(!treep) {  
  
        root = treep = new Node(value);  
  
    } else if (treep->getValue() == value) {  
  
        treep->incCount();  
  
    } else if (treep->getValue() > value) {  
  
        if(!treep->leftChild()) {  
  
            treep->leftChild(new Node(value));  
  
        } else {  
  
            treep = treep->leftChild();  
            *this << value;  
        }  
  
    } else if (treep->getValue() < value) {  
  
        if(!treep->rightChild()) {  
  
            treep->rightChild(new Node(value));  
  
        } else {  
  
            treep = treep->rightChild();  
            *this << value;  
        }  
  
    }  
}
```

```
treep = root;  
  
return *this;  
}
```

Itt van a binfa feltöltésa ami röviden megnézi hogy van e a node-nak bal vagy jobb gyereke ha nincs csinálha meg van akkor továbbléünk.

```
template <typename ValueType>  
ZLWTree<ValueType> & ZLWTree<ValueType>::operator<< (ValueType value)  
{  
  
    if(value == '0') {  
  
        if(!this->treep->leftChild()) {  
  
            typename BinTree<ValueType>::Node * node = new typename BinTree <=   
                <ValueType>::Node (value);  
            this->treep->leftChild(node);  
            this->treep = this->root;  
  
        } else {  
  
            this->treep = this->treep->leftChild();  
        }  
  
    } else {  
  
        if(!this->treep->rightChild()) {  
  
            typename BinTree<ValueType>::Node * node = new typename BinTree <=   
                <ValueType>::Node (value);  
            this->treep->rightChild(node);  
            this->treep = this->root;  
  
        } else {  
  
            this->treep = this->treep->rightChild();  
        }  
  
    }  
  
    return *this;  
}
```

Itt ismét a gyermeket létezését vizsgáljuk.

```
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;
}

}

```

Itt nézzük meg a fánknak a depth-jét.

```

template <typename ValueType>
void BinTree<ValueType>::deltree(Node *node)
{
    if (node)
    {
        deltreenode->leftChild());
        deltreenode->rightChild());

        delete node;
    }
}

```

Itt van a destrukturunk aminek a feladat hogy a binfat és a gyeremekeit likvidálja amivel memóriát szabadít fel.

```

int main(int argc, char** argv, char ** env)
{
    BinTree<int> bt;

    bt << 8 << 9 << 5 << 2 << 7;

    bt.print();

    std::cout << std::endl;

    ZLWTtree<char> zt;

    zt  ↵
        << '0' << '1' << '1' << '1' << '1' << '0' << '0' << '1' << '0' << '0' << '1' << '0' << '0' << '1' << '0' << '1' << '0'

```

```
zt.print();  
}
```

Main-nünkben meg csak tolunk egy pár tesztet hogy működik e amit írtunk.

A másoló és mozgató rész z3a18-ból

```
public:  
    BinTree(Node *root = nullptr, Node *treep=nullptr): root(root), treep( ←  
        treep){  
        std::cout << "BT ctor " << std::endl;  
    }  
    BinTree(const BinTree & old){  
        std::cout << "BT copy ctor " << std::endl;  
        root = cp(old.root,old.treep);  
    }  
  
    Node * cp(Node *node, Node *treep){  
        Node * newNode = nullptr;  
  
        if(node){  
            newNode = new Node(node->getValue());  
  
            newNode->leftChild(cp(node->leftChild(),treep));  
            newNode->rightChild(cp(node->rightChild(),treep));  
  
            if(node == treep){  
                this->treep= newNode;  
            }  
        }  
  
        return newNode;  
    }  
  
    BinTree & operator =(const BinTree & old){  
        std::cout << "BT copy assign " << std::endl;  
  
        BinTree tmp{old};  
        std::swap(*this, tmp);  
        return *this;  
    }  
    BinTree(BinTree && old){  
        std::cout << "BT move ctor " << std::endl;  
        root= nullptr;
```

```
*this = std::move(old);

}

BinTree & operator =(BinTree && old) {
    std::cout << "BT move assign " << std::endl;

    std::swap(old.root, root);
    std::swap(old.treep, treep);

    return *this;
}

~BinTree() {
    std::cout << "BT dtor " << std::endl;
    deltree(root);
}
```

A BinTree osztály public részbe kerül és eltávolítjuk a lekötéseket amiket z3a17-ben használtunk.

```
BinTree(Node *root = nullptr, Node *treep=nullptr): root(root), treep(←
    treep) {
    std::cout << "BT ctor " << std::endl;

}
```

Itt a node*root és a node*treep alapértelmezett értéket kapnak amik nullpointerek lesznek és a root megkapja a root-ot és a treep is megkapja az értékét. Ha ezek sikérülnek kiírjuk hogy BT ctor.

```
BinTree(const BinTree & old) {
    std::cout << "BT copy ctor " << std::endl;

    root = cp(old.root,old.treep);
}
```

Ez lesz itt a másoló konstruktorunk. Bintree bekéri a másolandó fát és az old-ot. Ezek után a rootnak odaadjuk a cp függvenyel az old gyökerünket ls treepet.

```
Node * cp(Node *node, Node *treep) {

    Node * newNode = nullptr;

    if(node) {
        newNode = new Node(node->getValue());

        newNode->leftChild(cp(node->leftChild(),treep));
        newNode->rightChild(cp(node->rightChild(),treep));
    }
}
```

```
    if(node == treep) {
        this->treep= newNode;
    }
}

return newNode;

}
```

A másoló függvény kéri a másolandó Node pointerjét és a treep mutatót. Ezután létrehozunk egy új node* pointert amiit a nullpointerre állítjuk. Ha van node-unk akkor új node egyenlő egy new node-ra amibe beletoljuk a node értékét. Ezek után az új node-unkba bal és jobb gyermekéibe beletoljuk a másolandó fa gyermekeit és treep-jét. Végül ha a node egyenlő a treep-el akkor a treep megkapja az új node-ot.

```
BinTree & operator =(const BinTree & old){
    std::cout << "BT copy assign "<< std::endl;

    BinTree tmp{old};
    std::swap(*this, tmp);
    return *this;

}
```

A másolás konstruktor hozzárendelése.

```
BinTree(BinTree && old){
    std::cout << "BT move ctor "<< std::endl;
    root= nullptr;

    *this = std::move(old);

}
```

A mozgató konstruktor. A root-ot nullpointerre mutatjuk kiirajtuk hogy bt move assisgn és felcseéljük az old.root, root-al és a old.treep-et treep-el majd vissza terunoljuk a this* pointert

```
BinTree & operator =(BinTree && old){
    std::cout << "BT move assign "<< std::endl;

    std::swap(old.root,root);
    std::swap(old.treep,treep);

    return *this;
}
```

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A program a hangyák feromonnal való kommunikációját szimulálja. A képernyőt felosztjuk cellákra és a cellákban levő hangyák megpróbálják elérni azt a társukat akinek a legerőbb a feromon szintje. A cellák feromon szintje folyamatosan csökken kivéve ha egy hangya belesétál egy feromon foltba mert akkor megnő a cella feromon tartalma.

The screenshot shows a terminal window titled 'laci@laci-GL503VD: ~/programok/hangya-sim'. The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/hangya-sim$ qmake -project
laci@laci-GL503VD:~/programok/hangya-sim$ qmake hangya-sim.pro
laci@laci-GL503VD:~/programok/hangya-sim$ make
make: Nothing to be done for 'first'.
laci@laci-GL503VD:~/programok/hangya-sim$ ./myrmecologist
```

7.1. ábra. Terminálba való futtatás

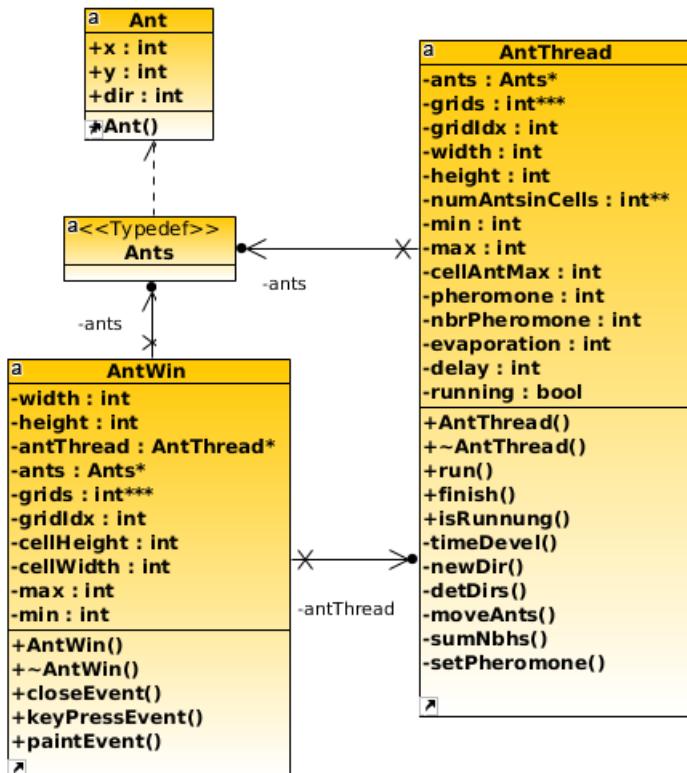
A szimulációnkon renegetek minden beállíthatunk kapcsolók segítségével. De persze nem kötelező megadni kapcsolókat mert vannak alapértelmezett értékek.

Lista a kapcsolókról:

- w cellák szélességét
- m cellák magasságát
- n a hangyák száma
- t a lépések gyakorisága (Milliszekundum)
- p feromon párolgás gyorsasága
- f hangya rálép egy feromon cellára mennyivel növelje annak értékét
- d feromon cellák kezdő értéke
- a maximális feromon érték
- i minimális feromon érték
- s hangya mennyi feromont hagyjon a szomszédos cellákba
- c egy cellába max hány hangy lehet

A main.cpp-be lévő futtatási javaslat

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



7.2. ábra. UML ábra

Na most hogy átnéztük a futtatást nézzünk is bele a forráskódba.

```

#ifndef ANT_H
#define ANT_H

class Ant
{
public:
    int x;
    int y;
    int dir;

    Ant(int x, int y) : x(x), y(y) {
        dir = qrand() % 8;
    }

};

typedef std::vector<Ant> Ants;

```

Ebben a részben a hangyák helyzetét határozzuk meg az x és y kordinátákkal. A helyzetén kívül a hangya irányát is itt számoljuk ki melyet a dir változóba tároljuk. Az utolsó sorban levő typefef -el az Ant tipusú elemekből álló vektorokat az Ants -el tudunk hivatkozni. A következő osztályunk az AntWin mely az antwin.h és antwin.cpp-ben találunk.

```
#ifndef ANTWIN_H
#define ANTWIN_H

#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCloseEvent>
#include "antthread.h"
#include "ant.h"

class AntWin : public QMainWindow
{
    Q_OBJECT

public:
    AntWin(int width = 100, int height = 75,
            int delay = 120, int numAnts = 100,
            int pheromone = 10, int nbhPheromon = 3,
            int evaporation = 2, int cellDef = 1,
            int min = 2, int max = 50,
            int cellAntMax = 4, QWidget *parent = 0);

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {

        antThread->finish();
        antThread->wait();
        event->accept();
    }

    void keyPressEvent ( QKeyEvent *event )
    {

        if ( event->key() == Qt::Key_P ) {
            antThread->pause();
        } else if ( event->key() == Qt::Key_Q
                    || event->key() == Qt::Key_Escape ) {
            close();
        }
    }

    virtual ~AntWin();
    void paintEvent(QPaintEvent*);
```

```
private:

    int ***grids;
    int **grid;
    int gridIdx;
    int cellWidth;
    int cellHeight;
    int width;
    int height;
    int max;
    int min;
    Ants* ants;

public slots :
    void step ( const int &);

};

#endif
```

A QMainWIndow osztály meghívásából tudjuk létrehozni a programunk ablakát. Mivel valamit ábrázolni is akarunk szükségünk van a QPainter osztályra. A QTString osztályal tudunk unicode stringeket tárolni, a QCloseEvent segítségével tudjuk a program bezárását szabályozni. Ez a header kapcsolja össze az Ant és az AntThread osztályokat. Szöval gyakorlatilag ebben a fájlban történik az ablak létrehozás és itt állítjuk be az egyes paramétereket plusz a program futtatását és szüneteltetését is itt vegezzük.

```
#ifndef ANTTHREAD_H
#define ANTTHREAD_H

#include <QThread>
#include "ant.h"

class AntThread : public QThread
{
    Q_OBJECT

public:
    AntThread(Ants * ants, int ***grids, int width, int height,
              int delay, int numAnts, int pheromone, int nbrPheromone,
              int evaporation, int min, int max, int cellAntMax);

    ~AntThread();

    void run();
    void finish()
    {
        running = false;
    }
}
```

```
void pause()
{
    paused = !paused;
}

bool isRunning()
{
    return running;
}

private:
    bool running {true};
    bool paused {false};
    Ants* ants;
    int** numAntsinCells;
    int min, max;
    int cellAntMax;
    int pheromone;
    int evaporation;
    int nbrPheromone;
    int ***grids;
    int width;
    int height;
    int gridIdx;
    int delay;

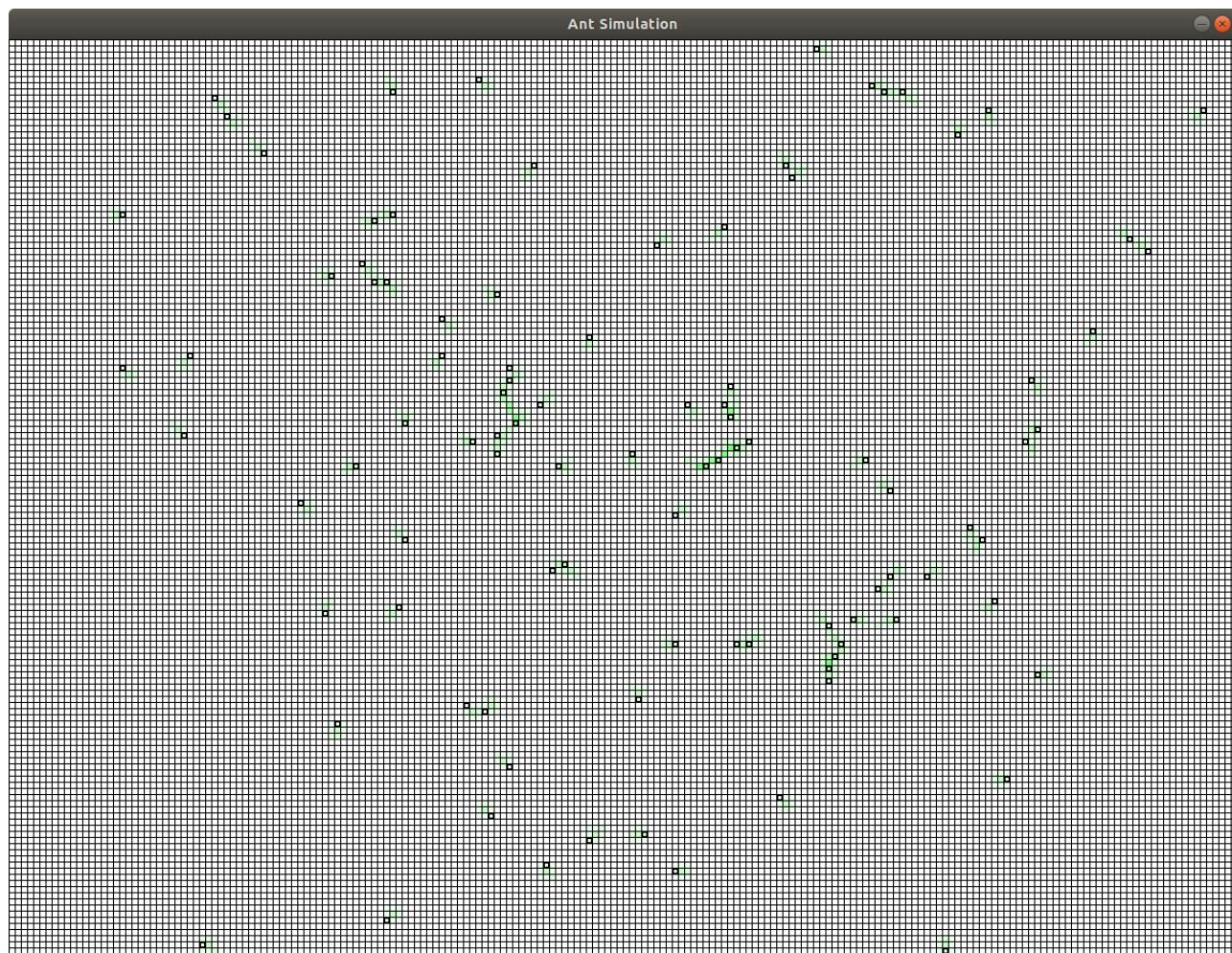
    void timeDevel();

    int newDir(int sor, int oszlop, int vsor, int voszlop);
    void detDirs(int irany, int& ifrom, int& ito, int& jfrom, int& jto );
    int moveAnts(int **grid, int row, int col, int& retrow, int& retcol, ←
        int);
    double sumNbhs(int **grid, int row, int col, int);
    void setPheromone(int **grid, int row, int col);

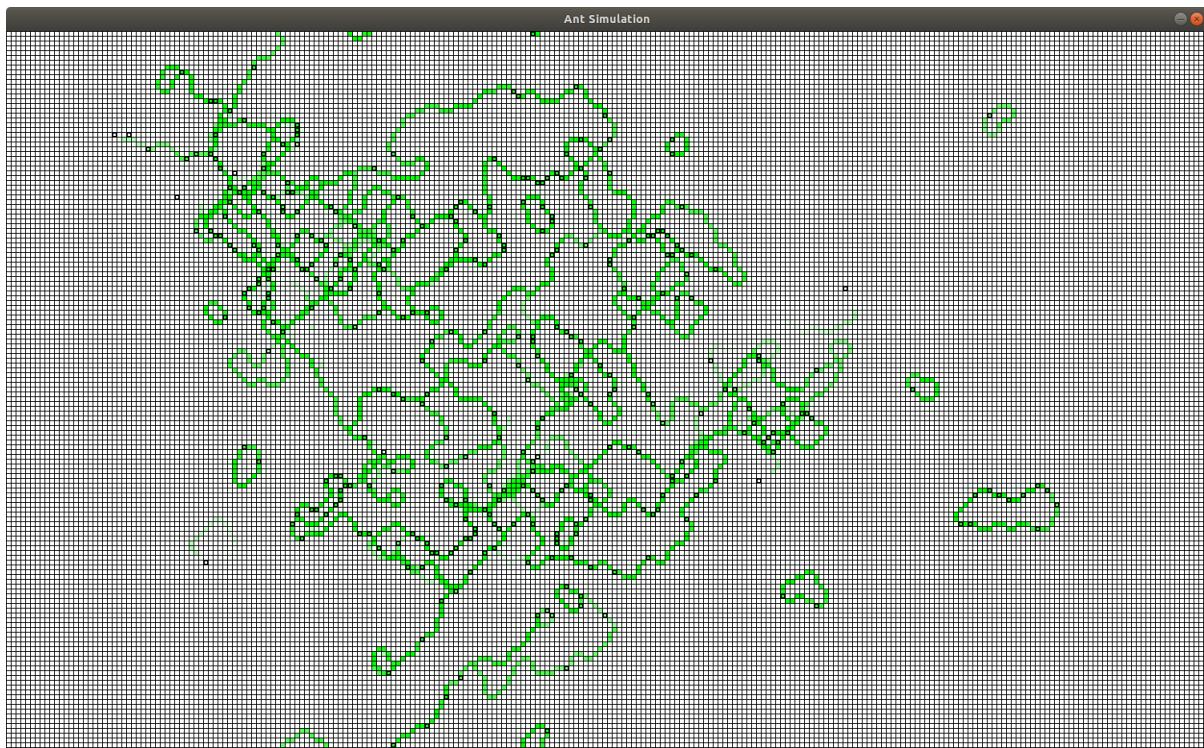
signals:
    void step ( const int &);

};
```

A számításokat az AntThread.h és AntThread.cpp -ben végezzük.A QThread osztályra szükségünk van hogy kezelje a program szálait. Az AntThread osztály megkapja azokat az értékeket amit az AntWin megkapott program elkezdi mozgatni a hangyákat.Kiszámolja hogy merre menjenek a feromon éréküket a cella max érték alapján állítja az egyes cellák feromonértékeit és magadja az AntWI osztálynak, hogy mely cellákat kell átfesteni.



7.3. ábra. Hangya szimuláció kapcsolók nélkül



7.4. ábra. Hangya szimuláció a recommended kapcsolókkal

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A négyzetrács mezőit celláknak, a korongokat sejteknek nevezzük. Egy cella környezete a hozzá legközelebb eső 8 mező (tehát a cellához képest „átlósan” elhelyezkedő cellákat is figyelembe vesszük, feltesszük hogy a négyzetrácsnak nincs széle). Egy sejt/cella szomszédai a környezetében lévő sejtek. A játék körökre osztott, a kezdő állapotban tetszőleges számú (egy vagy több) cellába sejteket helyezünk. Ezt követően a játékosnak nincs beleszólása a játékmenetbe. Egy sejttel (cellával) egy körben a következő három dolog történhet: A sejt túléli a kört, ha két vagy három szomszédja van. A sejt elpusztul, ha kettőnél kevesebb (elszigetelődés), vagy háromnál több (túlnépesedés) szomszédja van. Új sejt születik minden olyan cellában, melynek környezetében pontosan három sejt található.

```
public class Sejtautomata extends java.awt.Frame implements Runnable ←  
{  
    /** Egy sejt lehet élő */  
    public static final boolean ÉLŐ = true;  
    /** vagy halott */  
    public static final boolean HALOTT = false;  
    /** Két rácst használunk majd, az egyik a sejttér állapotát
```

```
* a t_n, a másik a t_n+1 időpillanatban jellemzi. */
protected boolean [][] [] rácsok = new boolean [2][][];
/** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
 * [2][][]-ból az első dimenziót használni, mert vagy az egyikre
 * állítjuk, vagy a másikra. */
protected boolean [] [] rács;
/** Megmutatja melyik rács az aktuális: [rácsIndex][][] */
protected int rácsIndex = 0;
/** Pixelben egy cella adatai. */
protected int cellaSzélesség = 20;
protected int cellaMagasság = 20;
/** A sejttér nagysága, azaz hányszor hány cella van? */
protected int szélesség = 20;
protected int magasság = 10;
/** A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
 * közötti valós idő. */
protected int várakozás = 1000;
// Pillanatfelvétel készítéséhez
private java.awt.Robot robot;
/** Készítsünk pillanatfelvételt? */
private boolean pillanatfelvétel = false;
/** A pillanatfelvételek számozásához. */
private static int pillanatfelvételSzámláló = 0;
/**
 * Létrehoz egy <code>Sejtautomata</code> objektumot.
 *
 * @param szélesség a sejttér szélessége.
 * @param magasság a sejttér szélessége.
 */
public Sejtautomata(int szélesség, int magasság) {
    this.szélesség = szélesség;
    this.magasság = magasság;
    // A két rács elkészítése
    rácsok[0] = new boolean[magasság][szélesség];
    rácsok[1] = new boolean[magasság][szélesség];
    rácsIndex = 0;
    rács = rácsok[rácsIndex];
    // A kiinduló rács minden cellája HALOTT
    for(int i=0; i<rács.length; ++i)
        for(int j=0; j<rács[0].length; ++j)
            rács[i][j] = HALOTT;
    // A kiinduló rácsra "élőlényeket" helyezünk
    //sikló(rács, 2, 2);
    siklóKilövő(rács, 5, 60);
    // Az ablak bezárásakor kilépünk a programból.
    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
            setVisible(false);
            System.exit(0);
        }
    }
```

```
});  
// A billentyűzetről érkező események feldolgozása  
addKeyListener(new java.awt.event.KeyAdapter() {  
    // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük  
    public void keyPressed(java.awt.event.KeyEvent e) {  
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {  
            // Felezük a cella méreteit:  
            cellaSzélesség /= 2;  
            cellaMagasság /= 2;  
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,  
                    Sejtautomata.this.magasság*cellaMagasság);  
            validate();  
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {  
            // Duplázzuk a cella méreteit:  
            cellaSzélesség *= 2;  
            cellaMagasság *= 2;  
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,  
                    Sejtautomata.this.magasság*cellaMagasság);  
            validate();  
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)  
            pillanatfelvétel = !pillanatfelvétel;  
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)  
            várakozás /= 2;  
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)  
            várakozás *= 2;  
        repaint();  
    }  
});  
// Egér kattintó események feldolgozása:  
addMouseListener(new java.awt.event.MouseAdapter() {  
    // Egér kattintással jelöljük ki a nagyítandó területet  
    // bal felső sarkát vagy ugyancsak egér kattintással  
    // vizsgáljuk egy adott pont iterációt:  
    public void mousePressed(java.awt.event.MouseEvent m) {  
        // Az egérmutató pozíciója  
        int x = m.getX()/cellaSzélesség;  
        int y = m.getY()/cellaMagasság;  
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];  
        repaint();  
    }  
});  
// Egér mozgás események feldolgozása:  
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {  
    // Vonszolással jelöljük ki a négyzetet:  
    public void mouseDragged(java.awt.event.MouseEvent m) {  
        int x = m.getX()/cellaSzélesség;  
        int y = m.getY()/cellaMagasság;  
        rácsok[rácsIndex][y][x] = ÉLŐ;  
        repaint();  
    }  
})
```

```
});  
// Cellaméretek kezdetben  
cellaSzélesség = 10;  
cellaMagasság = 10;  
// Pillanatfelvétel készítéséhez:  
try {  
    robot = new java.awt.Robot()  
        java.awt.GraphicsEnvironment.  
        getLocalGraphicsEnvironment().  
        getDefaultScreenDevice());  
} catch(java.awt.AWTException e) {  
    e.printStackTrace();  
}  
// A program ablakának adatai:  
setTitle("Sejtautomata");  
setResizable(false);  
setSize(szélesség*cellaSzélesség,  
       magasság*cellaMagasság);  
setVisible(true);  
// A sejttér életrekeltése:  
new Thread(this).start();  
}  
/** A sejttér kirajzolása. */  
public void paint(java.awt.Graphics g) {  
    // Az aktuális  
    boolean [][] rács = rácsok[rácsIndex];  
    // rácsot rajzoljuk ki:  
    for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon  
        for(int j=0; j<rács[0].length; ++j) { // s az oszlopok  
            // Sejt cella kirajzolása  
            if(rács[i][j] == ÉLŐ)  
                g.setColor(java.awt.Color.BLACK);  
            else  
                g.setColor(java.awt.Color.WHITE);  
            g.fillRect(j*cellaSzélesség, i*cellaMagasság,  
                      cellaSzélesség, cellaMagasság);  
            // Rács kirajzolása  
            g.setColor(java.awt.Color.LIGHT_GRAY);  
            g.drawRect(j*cellaSzélesség, i*cellaMagasság,  
                      cellaSzélesség, cellaMagasság);  
        }  
    }  
    // Készítünk pillanatfelvételt?  
    if(pillanatfelvétel) {  
        // a biztonság kedvéért egy kép készítése után  
        // kikapcsoljuk a pillanatfelvételt, hogy a  
        // programmal ismerkedő Olvasó ne írja tele a  
        // fájlrendszerét a pillanatfelvételekkel  
        pillanatfelvétel = false;  
        pillanatfelvétel(robot.createScreenCapture
```

```
(new java.awt.Rectangle
    (getLocation().x, getLocation().y,
     szélesség*cellaSzélesség,
     magasság*cellaMagasság)));
}

}

/***
 * Az kérdezett állapotban lévő nyolcszomszédok száma.
 *
 * @param rács a sejttér rács
 * @param sor a rács vizsgált sora
 * @param oszlop a rács vizsgált oszlopa
 * @param állapot a nyolcszomszédok vizsgált állapota
 * @return int a kérdezett állapotbeli nyolcszomszédok száma.
 */
public int szomszédochSzáma(boolean [][] rács,
    int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 0;
    // A nyolcszomszédok végigzongorázása:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgált sejtet magát kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejttérből szélénk szomszédai
                // a szembe oldalakon ("periódikus határfeltétel")
                int o = oszlop + j;
                if(o < 0)
                    o = szélesség-1;
                else if(o >= szélesség)
                    o = 0;

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

                if(rács[s][o] == állapot)
                    ++állapotúSzomszéd;
            }
    return állapotúSzomszéd;
}
/***
 * A sejttér időbeli fejlődése a John H. Conway féle
 * életjáték sejtautomata szabályai alapján történik.
 * A szabályok részletes ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 171. oldal.)
 */

```

```
public void időFejlődés() {  
  
    boolean [][] rácsElőtte = rácsok[rácsIndex];  
    boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];  
  
    for(int i=0; i<rácsElőtte.length; ++i) { // sorok  
        for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok  
  
            int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);  
  
            if(rácsElőtte[i][j] == ÉLŐ) {  
                /* Élő élő marad, ha kettő vagy három élő  
                szomszedja van, különben halott lesz. */  
                if(élők==2 || élők==3)  
                    rácsUtána[i][j] = ÉLŐ;  
                else  
                    rácsUtána[i][j] = HALOTT;  
            } else {  
                /* Halott halott marad, ha három élő  
                szomszedja van, különben élő lesz. */  
                if(élők==3)  
                    rácsUtána[i][j] = ÉLŐ;  
                else  
                    rácsUtána[i][j] = HALOTT;  
            }  
        }  
    }  
    rácsIndex = (rácsIndex+1)%2;  
}  
/** A sejttér időbeli fejlődése. */  
public void run() {  
  
    while(true) {  
        try {  
            Thread.sleep(várakozás);  
        } catch (InterruptedException e) {}  
  
        időFejlődés();  
        repaint();  
    }  
}  
/**  
 * A sejttérbe "élőlényeket" helyezünk, ez a "sikló".  
 * Adott irányban halad, másolja magát a sejttérben.  
 * Az élőlény ismertetését lásd például a  
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét  
 * matematikai játékok. Polygon, Szeged 1998. 172. oldal.)  
 *  
 * @param rács a sejttér ahová ezt az állatkát helyezzük  
 * @param x a befoglaló téglalal bal felső sarkának oszlopja
```

```
* @param      y      a befoglaló téglalap bal felső sarkának sora
*/
public void sikló(boolean [][] rács, int x, int y) {

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

}

/**
 * A sejttérbe "élőlényeket" helyezünk, ez a "sikló ágyú".
 * Adott irányban siklókat lő ki.
 * Az élőlény ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban /Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 173. oldal./,
 * de itt az ábra hibás, egy oszloppal told még balra a
 * bal oldali 4 sejtes négyzetet. A helyes ágyú rajzát
 * lásd pl. az [ÉLET CIKK] hivatkozásban /Robert T.
 * Wainwright: Life is Universal./ (Megemlíthetjük, hogy
 * minden tartalmaz két felesleges sejtet is.)
 *
 * @param      rács      a sejttér ahová ezt az állatkát helyezzük
 * @param      x      a befoglaló téglalap bal felső sarkának oszlopa
 * @param      y      a befoglaló téglalap bal felső sarkának sora
*/
public void siklóKilövő(boolean [][] rács, int x, int y) {

    rács[y+ 6][x+ 0] = ÉLŐ;
    rács[y+ 6][x+ 1] = ÉLŐ;
    rács[y+ 7][x+ 0] = ÉLŐ;
    rács[y+ 7][x+ 1] = ÉLŐ;

    rács[y+ 3][x+ 13] = ÉLŐ;

    rács[y+ 4][x+ 12] = ÉLŐ;
    rács[y+ 4][x+ 14] = ÉLŐ;

    rács[y+ 5][x+ 11] = ÉLŐ;
    rács[y+ 5][x+ 15] = ÉLŐ;
    rács[y+ 5][x+ 16] = ÉLŐ;
    rács[y+ 5][x+ 25] = ÉLŐ;

    rács[y+ 6][x+ 11] = ÉLŐ;
    rács[y+ 6][x+ 15] = ÉLŐ;
    rács[y+ 6][x+ 16] = ÉLŐ;
    rács[y+ 6][x+ 22] = ÉLŐ;
    rács[y+ 6][x+ 23] = ÉLŐ;
    rács[y+ 6][x+ 24] = ÉLŐ;
```

```
rács[y+ 6][x+ 25] = ÉLŐ;

rács[y+ 7][x+ 11] = ÉLŐ;
rács[y+ 7][x+ 15] = ÉLŐ;
rács[y+ 7][x+ 16] = ÉLŐ;
rács[y+ 7][x+ 21] = ÉLŐ;
rács[y+ 7][x+ 22] = ÉLŐ;
rács[y+ 7][x+ 23] = ÉLŐ;
rács[y+ 7][x+ 24] = ÉLŐ;

rács[y+ 8][x+ 12] = ÉLŐ;
rács[y+ 8][x+ 14] = ÉLŐ;
rács[y+ 8][x+ 21] = ÉLŐ;
rács[y+ 8][x+ 24] = ÉLŐ;
rács[y+ 8][x+ 34] = ÉLŐ;
rács[y+ 8][x+ 35] = ÉLŐ;

rács[y+ 9][x+ 13] = ÉLŐ;
rács[y+ 9][x+ 21] = ÉLŐ;
rács[y+ 9][x+ 22] = ÉLŐ;
rács[y+ 9][x+ 23] = ÉLŐ;
rács[y+ 9][x+ 24] = ÉLŐ;
rács[y+ 9][x+ 34] = ÉLŐ;
rács[y+ 9][x+ 35] = ÉLŐ;

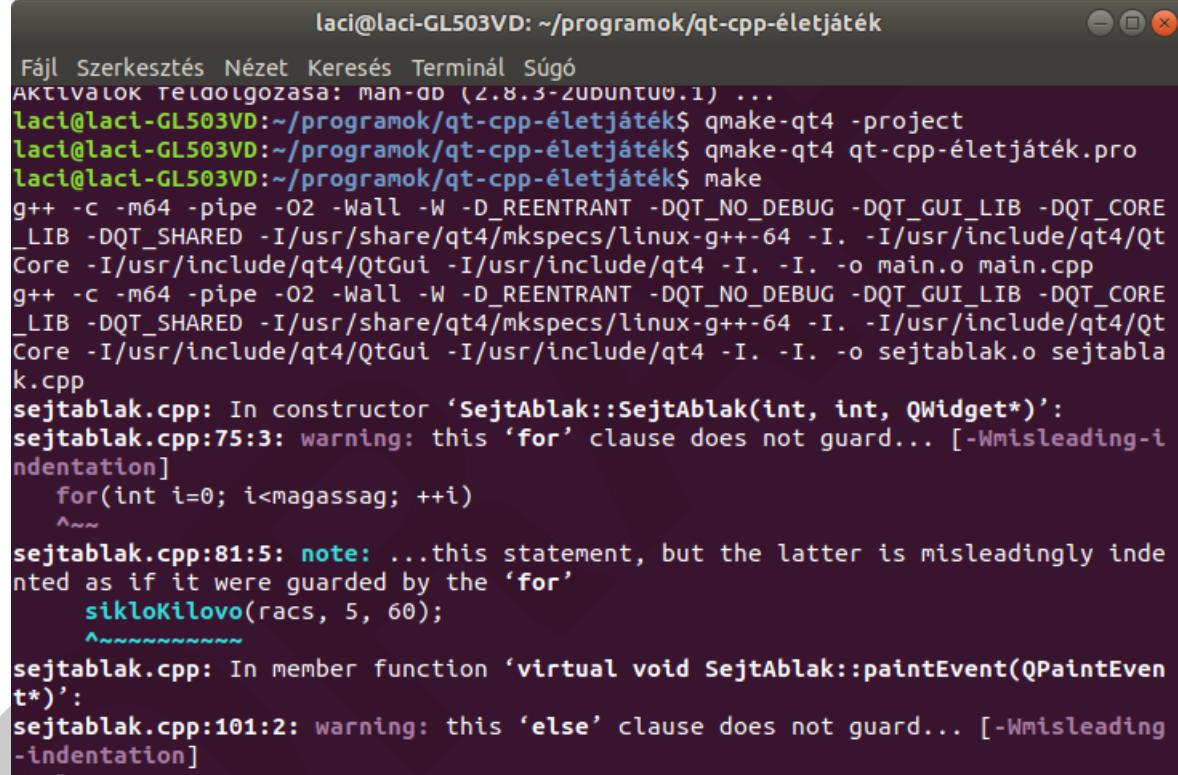
rács[y+ 10][x+ 22] = ÉLŐ;
rács[y+ 10][x+ 23] = ÉLŐ;
rács[y+ 10][x+ 24] = ÉLŐ;
rács[y+ 10][x+ 25] = ÉLŐ;

rács[y+ 11][x+ 25] = ÉLŐ;

}

/** Pillanatfelvételek készítése. */
public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    // A pillanatfelvétel kép fájlneve
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételszámláló);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
// Ne villogjon a felület (mert a "gyári" update()
```

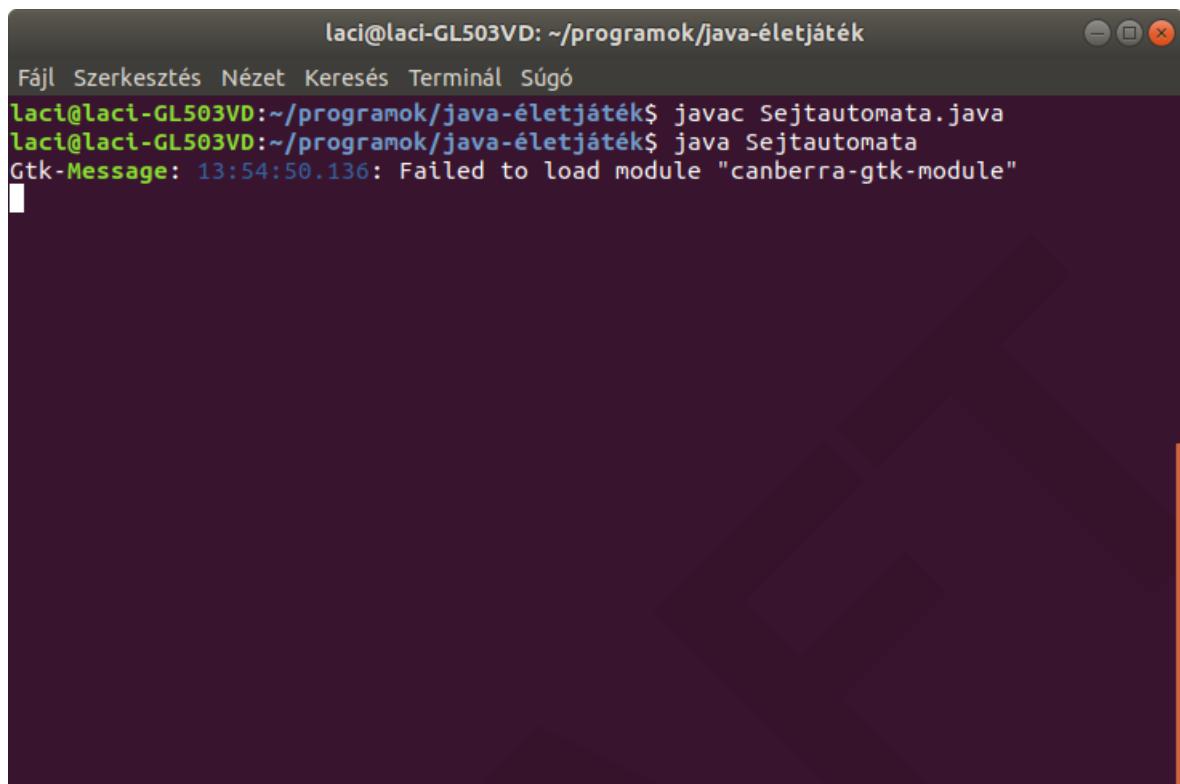
```
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
    paint(g);
}
/**
 * Példányosít egy Conway-féle életjáték szabályos
 * sejttér obektumot.
 */
public static void main(String[] args) {
    // 100 oszlop, 75 sor mérettel:
    new Sejtautomata(100, 75);
}
```



A terminal window titled "laci@laci-GL503VD: ~/programok/qt-cpp-életjáték". The window displays the following command-line output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
AKTÍVÁLOK TELAOLOGZASA: man-OO (2.8.3-ZUBUNTU0.1) ...
laci@laci-GL503VD:~/programok/qt-cpp-életjáték$ qmake-qt4 -project
laci@laci-GL503VD:~/programok/qt-cpp-életjáték$ qmake-qt4 qt-cpp-életjáték.pro
laci@laci-GL503VD:~/programok/qt-cpp-életjáték$ make
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -I. -o main.o main.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -I. -o sejtablak.o sejtablak.cpp
sejtablak.cpp: In constructor 'SejtAblak::SejtAblak(int, int, QWidget*)':
sejtablak.cpp:75:3: warning: this 'for' clause does not guard... [-Wmisleading-indentation]
    for(int i=0; i<magassag; ++i)
    ^
sejtablak.cpp:81:5: note: ...this statement, but the latter is misleadingly indented as if it were guarded by the 'for'
    sikloKilovo(racs, 5, 60);
    ^
sejtablak.cpp: In member function 'virtual void SejtAblak::paintEvent(QPaintEvent*)':
sejtablak.cpp:101:2: warning: this 'else' clause does not guard... [-Wmisleading-indentation]
    ^
```

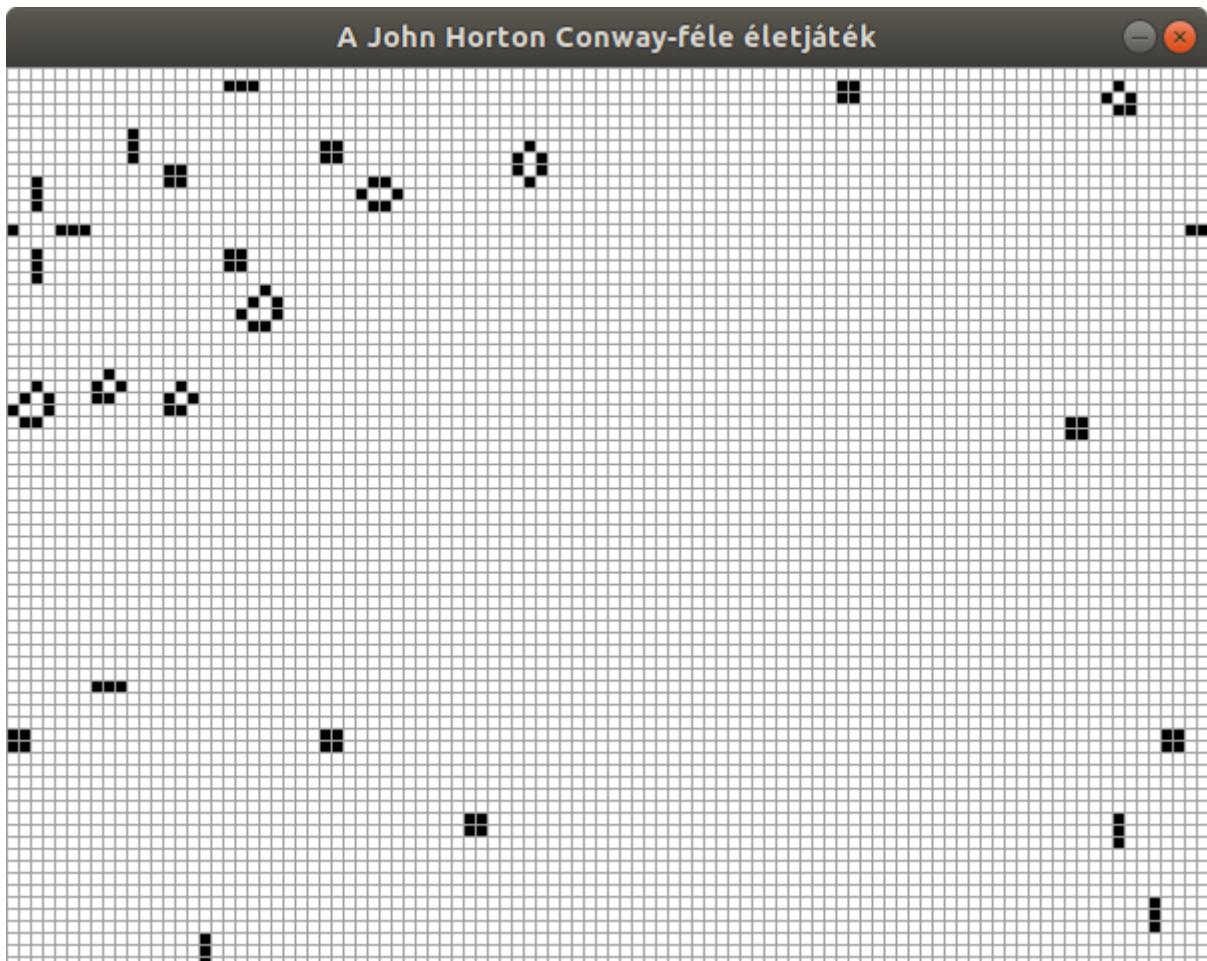
7.5. ábra. Terminál futtatás 1



The screenshot shows a terminal window titled "laci@laci-GL503VD: ~/programok/java-életjáték". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/java-életjáték$ javac Sejtautomata.java
laci@laci-GL503VD:~/programok/java-életjáték$ java Sejtautomata
Gtk-Message: 13:54:50.136: Failed to load module "canberra-gtk-module"
```

7.6. ábra. Terminál futtatás 2



7.7. ábra. Program futás közben

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Sejtszál.h Elsőnek includoljuk a QThread osztályt és a sejtablak.h-t Létrehozunk egy QThread osztályt melyet Sejtszál -nak nevezünk el. Publikusan tagben deklaráljuk a konstruktort és a destruktort és a run() függvényt. A protected részben létrehozzunk egy páros változót mint az előző feladatunkban és az utolsó sorban létrehozunk egy SejtAblakra mutatót mutatót.

```
#ifndef SEJTSZAL_H
#define SEJTSZAL_H

#include <QThread>
#include "sejtablak.h"
```

```
class SejtAblak;

class SejtSzal : public QThread
{
    Q_OBJECT

public:
    SejtSzal(bool ***racsok, int szelesseg, int magassag,
              int varakozas, SejtAblak *sejtAblak);
    ~SejtSzal();
    void run();

protected:
    bool ***racsok;
    int szelesseg, magassag;
    // Megmutatja melyik rács az aktuális: [rácsIndex] []
    int racsIndex;
    // A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
    // közötti valós idő.
    int varakozas;
    void idoFejlodes();
    int szomszedokSzama(bool **racs,
                         int sor, int oszlop, bool allapot);
    SejtAblak* sejtAblak;

};

#endif // SEJTSZAL_H
```

Sejtablak.h Hozzáadjuk a QtGui/QMainWIndow és a QPainter osztályokat ezeket már részleteztem az első feladatban valamint a sejszal.h-t utóbbiból deklaráljuk a SejtSzal classt. A QMainWIndow kiterjesztékként beállítjuk a SejtAblak osztályt. Publikusként létrehozunk 2 egy konstruktort és egy destruktort és deklaráljuk a vissza függvényt. Protectedben ismét változókat és boolean-eket vezetünk be ezeknek a feladatuk magába foglalja a nevük csak az utolsó 3 érdekes a paintEvent() ami a kirajzolásért lesz felelős a siklo() kirajzolja a siklót és a sikloKilovo() kirajzolja a siklokilovot.

```
#ifndef SEJTABLAK_H
#define SEJTABLAK_H

#include <QMainWindow>
#include <QPainter>
#include "sejtszal.h"

class SejtSzal;

class SejtAblak : public QMainWindow
{
    Q_OBJECT
```

```
public:  
    SejtAblak(int szelesseg = 100, int magassag = 75, QWidget *parent = 0);  
  
    ~SejtAblak();  
    // Egy sejt lehet élő  
    static const bool ELO = true;  
    // vagy halott  
    static const bool HALOTT = false;  
    void vissza(int racsIndex);  
  
protected:  
    // Két rácsot használunk majd, az egyik a sejttér állapotát  
    // a t_n, a másik a t_n+1 időpillanatban jellemzi.  
    bool ***racsok;  
    // Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a  
    // [2][][]-ból az első dimenziót használni, mert vagy az egyikre  
    // állítjuk, vagy a másikra.  
    bool **racs;  
    // Megmutatja melyik rács az aktuális: [räcsIndex][][]  
    int racsIndex;  
    // Pixelben egy cella adatai.  
    int cellaSzelesseg;  
    int cellaMagassag;  
    // A sejttér nagysága, azaz hányszor hány cella van?  
    int szelesseg;  
    int magassag;  
    void paintEvent(QPaintEvent*);  
    void siklo(bool **racs, int x, int y);  
    void sikloKilovo(bool **racs, int x, int y);  
  
private:  
    SejtSzal* eletjatek;  
  
};  
#endif // SEJTABLAK_H
```

Sejtszal.cpp A konstruktor csak bekéri az értékeket és a racsIndex-et 0-ra állítja. A szomszedokSzama() megadott állapotú elemek számát adja vissza az aktuális sejt szomszedságából. Az idoFejlodes() az aktuális rács alapján kiállítja a következő állapotot a következő rácsba. Végig megy az összes sejten és átnézi az összeset. Egy sejt csak akkor maradhat élő ha 2 vagy 3 szomszéda van és ha egy sejt halott csak akkor éledhet fel ha nincs 3 elő szomszéda miután ezeket elvégezte a rácsindexet lépteti. A run() egy végtelen ciklus mely elindítja el a folyamatot benne altatjuk a varakozas függvényt és mentjük az állapotot ha kapánk egy varakozást tudjuk mire kell visszaállítani a rácokat.

```
#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 kérdezett állapotban lévő nyolcszomszédok száma.
 *
 * @param rács a sejttér rács
 * @param sor a rács vizsgált sora
 * @param oszlop a rács vizsgált oszlopa
 * @param állapor a nyolcszomszédok vizsgált állapota
 * @return int a kérdezett állapotbeli nyolcszomszédok száma.
 */
int SejtSzal::szomszedokSzama(bool **racs,
                                int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
    // A nyolcszomszédok végigzongorázása:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgált sejtet magát kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejttérből szélénk szomszédai
                // a szembe oldalakon ("periódikus határfeltétel")
                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 sejttér időbeli fejlődése a John H. Conway féle
 * életjáték sejtautomata szabályai alapján történik.
 * A szabályok részletes ismertetését lásd például a
```

```
* [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
* matematikai játékok. Polygon, Szeged 1998. 171. oldal.)
*/
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) {
                /* Élő élő marad, ha kettő vagy három élő
                szomszedja van, különben halott lesz. */
                if(elok==2 || elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            } else {
                /* Halott halott marad, ha három élő
                szomszedja van, különben élő lesz. */
                if(elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            }
        }
    }
    racsIndex = (racsIndex+1)%2;
}

/** A sejttér időbeli fejlődése. */
void SejtSzal::run()
{
    while(true) {
        QThread::msleep(varakozas);
        idoFejlodes();
        sejtAblak->vissza(racsIndex);
    }
}

SejtSzal::~SejtSzal()
{
}
```

Sejtblak.cpp Megadja az ablak nevét és beállítja a szelesség és magasság változókat. A cellák méretét kódon belül kell megadni beállíthatjuk az ablak méretét is szintén kódon belül. létrehozunk 2 rácsot és az őket tároló objektumokat. A racsIndexet 0-ra állítjuk és feltöljük a rácsokat halott sejtekkel. Ezután meghívjuk a sikloKilovo() függvényt így kapunk egy siklókilövőt. Ezek után márcsak annyi dolgunk maradt, hogy példányosítunk egy SejtSzal-at a megfelelő paraméterekkel, aztán meghívjuk az így létrehozott objektum örökölt start() függvényét, ami meghívja a SejtSzal-unk run() metódusát, ezzel elindítva az életjátékot. A paintEvent() az végigmegy a rácsokon és minden élő sejet feketére fest egyábként meg fehére plusz rajzol egy téglalapot és beszínezi szürkére. A destruktur törli a threadet , rácsokat. rácsok tartalmát.

```
#include "sejtblak.h"

SejtAblak::SejtAblak(int szelessseg, int magassag, QWidget *parent)
: QMainWindow(parent)
{
    setWindowTitle("A John Horton Conway-féle életjáték");

    this->magassag = magassag;
    this->szelessseg = szelessseg;

    cellaSzelessseg = 6;
    cellaMagassag = 6;

    setFixedSize(QSize(szelessseg*cellaSzelessseg, magassag*cellaMagassag));

    racsok = new bool**[2];
    racsok[0] = new bool*[magassag];
    for(int i=0; i<magassag; ++i)
        racsok[0][i] = new bool [szelessseg];
    racsok[1] = new bool*[magassag];
    for(int i=0; i<magassag; ++i)
        racsok[1][i] = new bool [szelessseg];

    racsIndex = 0;
    racs = racsok[racsIndex];

    // A kiinduló racs minden cellája HALOTT
    for(int i=0; i<magassag; ++i)
        for(int j=0; j<szelessseg; ++j)
            racs[i][j] = HALOTT;
    // A kiinduló racsra "ELölényeket" helyezünk
    //siklo(racs, 2, 2);

    sikloKilovo(racs, 5, 60);

    eletjatek = new SejtSzal(racsok, szelessseg, magassag, 120, this);

    eletjatek->start();

}
```

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

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

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

    qpainter.end();
}

SejtAblak::~SejtAblak()
{
    delete eletjatek;

    for(int i=0; i<magassag; ++i) {
        delete[] racsok[0][i];
        delete[] racsok[1][i];
    }

    delete[] racsok[0];
    delete[] racsok[1];
    delete[] racsok;
}

void SejtAblak::vissza(int racsIndex)
{
    this->racsIndex = racsIndex;
    update();
}

/**
```

```
* A sejttérbe "ELOlényeket" helyezünk, ez a "sikló".  
* Adott irányban halad, másolja magát a sejttérben.  
* Az ELOlény ismertetését lásd például a  
* [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét  
* matematikai játékok. Polygon, Szeged 1998. 172. oldal.)  
*  
* @param racs a sejttér ahová ezt az állatkát helyezzük  
* @param x a befoglaló téglalap bal felső sarkának oszlopa  
* @param y a befoglaló téglalap bal felső sarkának sora  
*/  
void SejtAblak::siklo(bool **racs, int x, int y) {  
  
    racs[y+ 0][x+ 2] = ELO;  
    racs[y+ 1][x+ 1] = ELO;  
    racs[y+ 2][x+ 1] = ELO;  
    racs[y+ 2][x+ 2] = ELO;  
    racs[y+ 2][x+ 3] = ELO;  
  
}  
/**  
* A sejttérbe "ELOlényeket" helyezünk, ez a "sikló ágyú".  
* Adott irányban siklókat lö ki.  
* Az ELOlény ismertetését lásd például a  
* [MATEK JÁTÉK] hivatkozásban /Csákány Béla: Diszkrét  
* matematikai játékok. Polygon, Szeged 1998. 173. oldal./,  
* de itt az ábra hibás, egy oszloppal told még balra a  
* bal oldali 4 sejtes négyzetet. A helyes ágyú rajzát  
* lásd pl. az [ÉLET CIKK] hivatkozásban /Robert T.  
* Wainwright: Life is Universal./ (Megemlíthetjük, hogy  
* minden kettő tartalmaz két felesleges sejtet is.)  
*  
* @param racs a sejttér ahová ezt az állatkát helyezzük  
* @param x a befoglaló téglalap bal felső sarkának oszlopa  
* @param y a befoglaló téglalap bal felső sarkának sora  
*/  
void SejtAblak::sikloKilovo(bool **racs, int x, int y) {  
  
    racs[y+ 6][x+ 0] = ELO;  
    racs[y+ 6][x+ 1] = ELO;  
    racs[y+ 7][x+ 0] = ELO;  
    racs[y+ 7][x+ 1] = ELO;  
  
    racs[y+ 3][x+ 13] = ELO;  
  
    racs[y+ 4][x+ 12] = ELO;  
    racs[y+ 4][x+ 14] = ELO;  
  
    racs[y+ 5][x+ 11] = ELO;  
    racs[y+ 5][x+ 15] = ELO;  
    racs[y+ 5][x+ 16] = ELO;
```

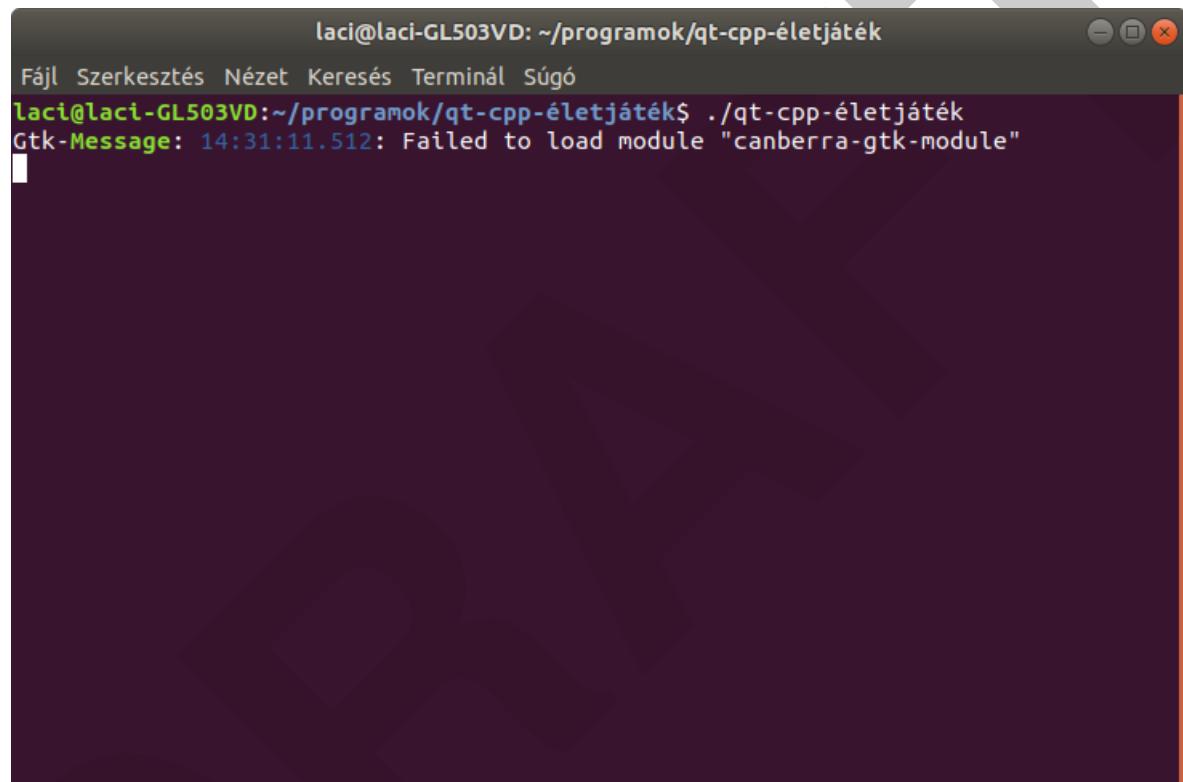
```
racs[y+ 5][x+ 25] = ELO;  
  
racs[y+ 6][x+ 11] = ELO;  
racs[y+ 6][x+ 15] = ELO;  
racs[y+ 6][x+ 16] = ELO;  
racs[y+ 6][x+ 22] = ELO;  
racs[y+ 6][x+ 23] = ELO;  
racs[y+ 6][x+ 24] = ELO;  
racs[y+ 6][x+ 25] = ELO;  
  
racs[y+ 7][x+ 11] = ELO;  
racs[y+ 7][x+ 15] = ELO;  
racs[y+ 7][x+ 16] = ELO;  
racs[y+ 7][x+ 21] = ELO;  
racs[y+ 7][x+ 22] = ELO;  
racs[y+ 7][x+ 23] = ELO;  
racs[y+ 7][x+ 24] = ELO;  
  
racs[y+ 8][x+ 12] = ELO;  
racs[y+ 8][x+ 14] = ELO;  
racs[y+ 8][x+ 21] = ELO;  
racs[y+ 8][x+ 24] = ELO;  
racs[y+ 8][x+ 34] = ELO;  
racs[y+ 8][x+ 35] = ELO;  
  
racs[y+ 9][x+ 13] = ELO;  
racs[y+ 9][x+ 21] = ELO;  
racs[y+ 9][x+ 22] = ELO;  
racs[y+ 9][x+ 23] = ELO;  
racs[y+ 9][x+ 24] = ELO;  
racs[y+ 9][x+ 34] = ELO;  
racs[y+ 9][x+ 35] = ELO;  
  
racs[y+ 10][x+ 22] = ELO;  
racs[y+ 10][x+ 23] = ELO;  
racs[y+ 10][x+ 24] = ELO;  
racs[y+ 10][x+ 25] = ELO;  
  
racs[y+ 11][x+ 25] = ELO;  
  
}
```

Main.cpp nem kap sok dolgot. Includoljuk QtGui-t QApplication könyvtárat és a sejtablak.h header. Meghívjuk a QApplication-t paraméternként argomentumkat adva neki. Meghívjuk a SejtAblak-ot paraméternként argomentumkat adva neki.

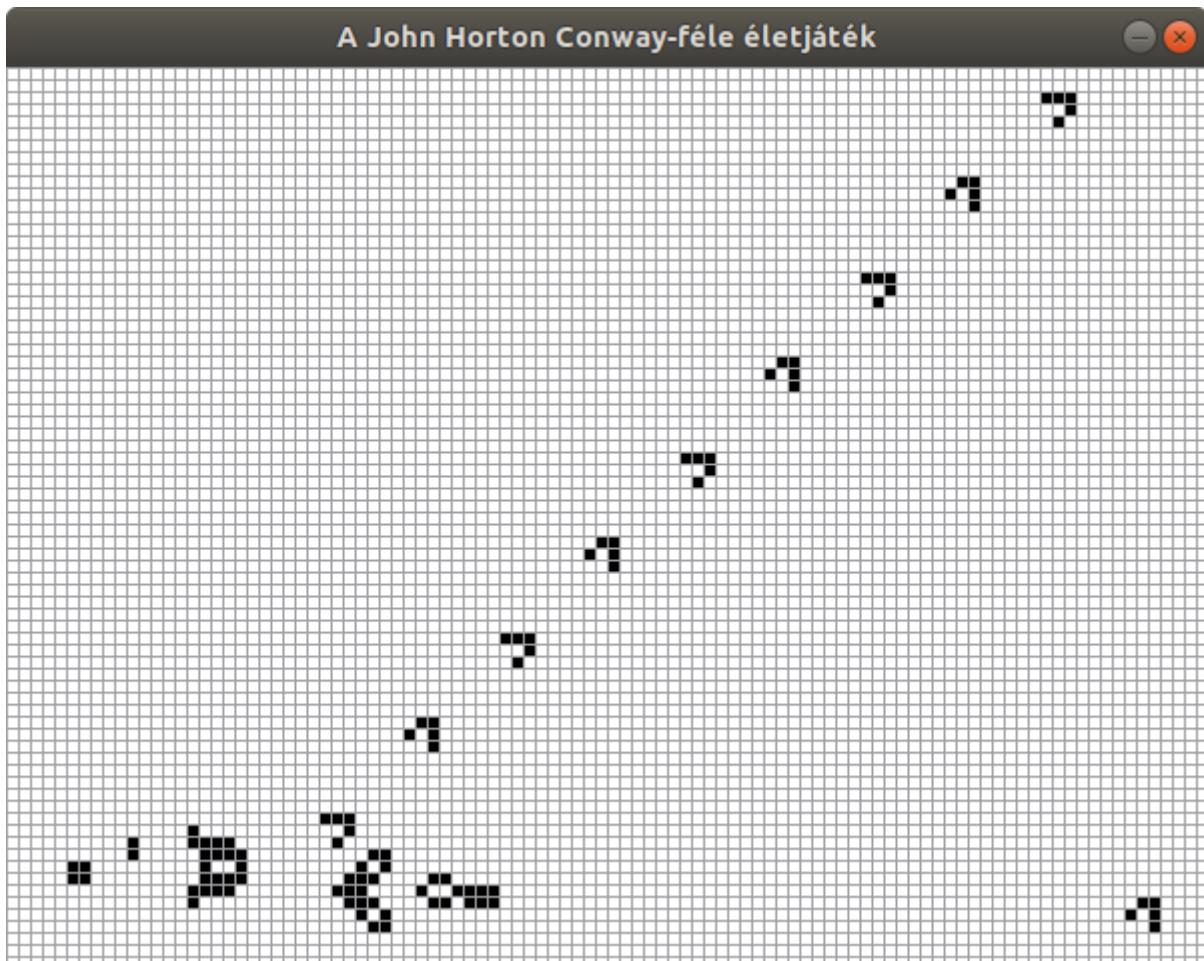
```
#include <QApplication>  
#include "sejtablak.h"  
#include <QDesktopWidget>
```

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    SejtAblak w(100, 75);
    w.show();

    return a.exec();
}
```



7.8. ábra. Terminál



7.9. ábra. ÉLetjáték

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A BrainB-s program fő célja a tehetségkutatás az e-sport terén. Azt vizsgálja, hogy az egyes sportolóknak milyen a reakcióképessége. A program futtatása után annyi a dolgunk, hogy a jobb egérgomb lenyomása alatt kövessük a Samu Entropy nevű karaktert. Ha kellő ideig tudjuk követni a négyzetben lévő kis teli kört, akkor új játékosok fognak megjelenni a képernyőn. Ha figylemesek vagyunk, akkor egyre több karakter lesz egy idő után, és természetesen egyre nehezebb is lesz követni Samut.

```
/**  
 * @brief Benchmarking Cognitive Abilities of the Brain with Computer Games  
 *  
 * @file BrainBThread.h  
 * @author Norbert Bátfai <nbatfai@gmail.com>  
 * @version 6.0.1
```

```
*  
* @section LICENSE  
*  
* Copyright (C) 2017, 2018 Norbert Bátfai, nbatfai@gmail.com  
*  
* This program is free software: you can redistribute it and/or modify  
* it under the terms of the GNU General Public License as published by  
* the Free Software Foundation, either version 3 of the License, or  
* (at your option) any later version.  
*  
* This program is distributed in the hope that it will be useful,  
* but WITHOUT ANY WARRANTY; without even the implied warranty of  
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
* GNU General Public License for more details.  
*  
* You should have received a copy of the GNU General Public License  
* along with this program. If not, see <http://www.gnu.org/licenses/>.  
*  
* @section DESCRIPTION  
*  
*/  
#ifndef BrainBThread_H  
#define BrainBThread_H  
  
#include <QThread>  
#include <QSize>  
#include <QImage>  
#include <QDebug>  
#include <sstream>  
#include < QPainter>  
#include <cstdlib>  
#include <ctime>  
#include <vector>  
#include <opencv2/opencv.hpp>  
#include <opencv2/core/core.hpp>  
#include <opencv2/imgproc/imgproc.hpp>
```

Az adott program egy header fájl lesz, az első két sorban definiáljuk is ezt. A defíciót menete a fenti C++-os életjátékban ki lett fejtve. Más header féjlokat hívunk segítségül hozzá, mint látjuk többnyire az összes az OpenCV programra épül, mivel annak a segítségével tudjuk majd megvalósítani a grafikus felületet.

```
class Hero;  
typedef std::vector<Hero> Heroes;  
  
class Hero  
{  
  
public:  
    int x;  
    int y;
```

```
int color;
int agility;
intconds {0};
std::string name;

Hero ( int x=0, int y=0, int color=0, int agility=1, std::string name ←
      ="Samu Entropy" ) :
    x ( x ), y ( y ), color ( color ), agility ( agility ), name ( name ←
      )
{ }
~Hero () {}
```

A class Hero a hősünk (akit követnünk kell) megalkotásában játszik szerepet. A publikus részben lévő deklarációkra a karakter megjelenítéséhez van szükség: x, y a méretek (a téglalap szélessége és magassága), color a színe, agility a mozgásának a sebessége, name pedig a karakter neve. Ezeket felhasználva jön a konstruktur és a destruktur megadása.

```
void move ( int maxx, int maxy, int env ) {

    int newx = x+ ( ( ( double ) agility*1.0 ) * ( double ) ( std::rand ←
        () / ( RAND_MAX+1.0 ) )-agility/2 ) ;
    if ( newx-env > 0 && newx+env < maxx ) {
        x = newx;
    }
    int newy = y+ ( ( ( double ) agility*1.0 ) * ( double ) ( std::rand ←
        () / ( RAND_MAX+1.0 ) )-agility/2 );
    if ( newy-env > 0 && newy+env < maxy ) {
        y = newy;
    }
}
```

A move függvény mint ahogy a neve is mutatja, a karakterek mozgásban van szerepe, hogy ne legyen annyira könnyű dolgunk a karakter követésével. A függvény egy mozgást ír le. A mozgás során az x és az y változók új értéket kapnak, persze az új (newx és newy) érétek a régi x-től és y-tól, a gyorsaságtól függnek.

A program következő részének elején lévő néhány sor a karakter megszerkeztésére szolgálnak, utánna pedig rengeteg függvénydeklaráció, illetve hogy azok a milyen visszatérítési értéket adnak vissza.

```
void draw ()
{
    cv::Mat src ( h+3*heroRectSize, w+3*heroRectSize, CV_8UC3, cBg );

    for ( Hero & hero : heroes ) {

        cv::Point x ( hero.x-heroRectSize+dispShift, hero.y- ←
                      heroRectSize+dispShift );
        cv::Point y ( hero.x+heroRectSize+dispShift, hero.y+ ←
                      heroRectSize+dispShift );
```

```
cv::rectangle ( src, x, y, cBorderAndText );

cv::putText ( src, hero.name, x, cv::FONT_HERSHEY_SIMPLEX, .35, ←
    cBorderAndText, 1 );

cv::Point xc ( hero.x+dispShift, hero.y+dispShift );

cv::circle ( src, xc, 11, cCenter, CV_FILLED, 8, 0 );

cv::Mat box = src ( cv::Rect ( x, y ) );

cv::Mat cbox ( 2*heroRectSize, 2*heroRectSize, CV_8UC3, cBoxes ←
    );
box = cbox*.3 + box*.7;
}

cv::Mat comp;

cv::Point focusx ( heroes[0].x- ( 3*heroRectSize ) /2+dispShift, ←
    heroes[0].y- ( 3*heroRectSize ) /2+dispShift );
cv::Point focusy ( heroes[0].x+ ( 3*heroRectSize ) /2+dispShift, ←
    heroes[0].y+ ( 3*heroRectSize ) /2+dispShift );
cv::Mat focus = src ( cv::Rect ( focusx, focusy ) );

cv::compare ( prev, focus, comp, cv::CMP_NE );

cv::Mat aRgb;
cv::extractChannel ( comp, aRgb, 0 );

bps = cv::countNonZero ( aRgb ) * 10;

prev = focus;

QImage dest ( src.data, src.cols, src.rows, src.step, QImage::Format_RGB888 );
dest=dest.rgbSwapped();
dest.bits();

emit heroesChanged ( dest, heroes[0].x, heroes[0].y );
}

long getT() const
{
    return time;
}

void finish ()
{
    time = endTime;
}
```

```
signals:
```

```
    void heroesChanged ( const QImage &image, const int &x, const int &y );
    void endAndStats ( const int &t );
};

#endif // BrainBThread_H
```

Ebben a részben mint ahogy a függvény neve is mondja draw a rajzolásról lesz szó. Mégpedig a hősök és és maga a megjeleníts kirajzolásában. Újra látjuk az ismerő "Q"-val kezdődő függvényhívásokat, az OpenCV jóvoltából. A számolások bevitelé után ami alapjaán lesz majd a megjelenítés, deklaráljuk a finish függvényt is, ami megmondja hogy az egész játék addig tart amíg a idő eléri az endtime-ot tehát a 10 percet.

```
void millis2minsec ( int millis, int &min, int &sec ) {

    sec = ( millis * 100 ) / 1000;
    min = sec / 60;
    sec = sec - min * 60;
}

bool save ( int t ) {

    bool ret = false;

    if ( !QDir ( statDir ).exists() )
        if ( !QDir().mkdir ( statDir ) ) {
            return false;
        }

    QString name = statDir + "/Test-" + QString::number ( t );
    QFile file ( name + "-screenimage.png" );
    if ( file.open ( QIODevice::WriteOnly ) ) {
        ret = pixmap.save ( &file, "PNG" );
    }

    QFile tfile ( name + "-stats.txt" );
    ret = tfile.open ( QIODevice::WriteOnly | QIODevice::Text );
    if ( ret ) {
        QTextStream textStremam ( &tfile );

        textStremam << appName + " " + appVersion << "\n";
        textStremam << "time      : " << brainBThread->getT() << "\n";
        textStremam << "bps       : " << brainBThread->get_bps() << "\n";
        textStremam << "noc      : " << brainBThread->nofHeroes() << "\n";
        textStremam << "nop      : " << brainBThread->get_nofPaused() << "\n";
    }
}
```

```
textStremam << "lost      : " << "\n";
std::vector<int> l = brainBThread->lostV();
for ( int n : l ) {
    textStremam << n << ' ';
}
textStremam << "\n";
int m = mean ( l );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( l, m ) << "\n";

textStremam << "found     : " ;
std::vector<int> f = brainBThread->foundV();
for ( int n : f ) {
    textStremam << n << ' ';
}
textStremam << "\n";
m = mean ( f );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( f, m ) << "\n";

textStremam << "lost2found: " ;
for ( int n : lost2found ) {
    textStremam << n << ' ';
}
textStremam << "\n";
int m1 = m = mean ( lost2found );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( lost2found, m ) << "\n" ←
;

textStremam << "found2lost: " ;
for ( int n : found2lost ) {
    textStremam << n << ' ';
}
textStremam << "\n";
int m2 = m = mean ( found2lost );
textStremam << "mean      : " << m << "\n";
textStremam << "var       : " << var ( found2lost, m ) << "\n" ←
;

if ( m1 < m2 ) {
    textStremam << "mean(lost2found) < mean(found2lost)" << "\n" ←
    ";
}

int min, sec;
millis2minsec ( t, min, sec );
textStremam << "time      : " << min << ":" << sec << "\n";
```

```
        double res = ( ( ( ( double ) m1+ ( double ) m2 ) /2.0 ) /8.0 ) ←
                     /1024.0;
    textStremam << "U R about " << res << " Kilobytes\n";
    tfile.close();
}
return ret;
}
#endif // BrainBWin
```

A következő program legfontosabb szerepe abban rejlik amikor deklaráljuk hogy az eredményeket hova mentésük. Ezt is az OpenCV segítségével csináljuk. Mint látjuk elég hosszú kód foglalja magába a mentés folyamatát. A fájl megnyitása után már tudunk is írni bele. Láthatjuk benne azt, hogy milyen nevű és hol található fájlba mentésük majd az eredményeket és hogy ezek az eredmények mik legyenek. A fájl egy szöveges fájl lesz, mely tartalmazni fogja a jéték nevét, a jétékidőt (mivel a 10 perc lejárta előtt is le lehet állítani a játékot az Esc billentyű segítségével), a "bábuk számát", hogy mikor veszette el felhasználó Samut stb. Az adatok fájlba való beírása után, be kell azt zárni a close segítségével.

```
#include "BrainBThread.h"

BrainBThread::BrainBThread ( int w, int h )
{
    dispShift = heroRectSize+heroRectSize/2;

    this->w = w - 3 * heroRectSize;
    this->h = h - 3 * heroRectSize;

    std::srand ( std::time ( 0 ) );

    Hero me ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100,
              this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 9 );

    Hero other1 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  5, "Norbi Entropy" );
    Hero other2 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  3, "Greta Entropy" );
    Hero other4 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
```

```
    5, "Nandi Entropy" );
Hero other5 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
) - 100,
            this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
7, "Matyi Entropy" );

heroes.push_back ( me );
heroes.push_back ( other1 );
heroes.push_back ( other2 );
heroes.push_back ( other4 );
heroes.push_back ( other5 );
}
```

Ebben a részebn azt deklaráljuk, hogy abban az esetben ha nagyon ügyesen tudjuk követni Samut, akkor a következőleg megjelenő játékosokat hozzá képest hova helyezzük. Az elhelyezés lényege az hogy minnen kövelebb Samu Entropy-hoz, hiszen így tudja befolyásolni a felhasználó reakció, figyelési és kognitív képességét.

```
void BrainBWin::mousePressEvent ( QMouseEvent *event )
{
    brainBThread->set_paused ( false );
}

void BrainBWin::mouseReleaseEvent ( QMouseEvent *event )
{
    //brainBThread->set_paused(true);
}

void BrainBWin::mouseMoveEvent ( QMouseEvent *event )
{
    start = true;

    mouse_x = event->pos().x() -xs - 60;
    mouse_y = event->pos().y() - ys - 60;
}

void BrainBWin::keyPressEvent ( QKeyEvent *event )
{
    if ( event->key() == Qt::Key_S ) {
        save ( brainBThread->getT() );
    } else if ( event->key() == Qt::Key_P ) {
        brainBThread->pause();
    } else if ( event->key() == Qt::Key_Q || event->key() == Qt::Key_Escape ) {
        close();
    }
}

BrainBWin::~BrainBWin()
```

{
}

Ebben a részben az egér kúrzon jelentőségét írjuk le a program számára. Fontos szerepe van hiszen ezzel kell tudjuk szem előtt tartani a mi Samunkat. Emellett deklrálunk néhány billentyű tevékenységet is: az S billentyű lenyomásával el tudjuk menteni az eddigi teljesítményünket, a P-vel szünetelteni tudjuk a játékot, a Q illetve az Esc gombok lenyomásával pedig ki tudunk lápni az egész játékból.

```
#include <QApplication>
#include <QTextStream>
#include <QtWidgets>
#include "BrainBWin.h"

int main ( int argc, char **argv )
{
    QApplication app ( argc, argv );

    QTextStream qout ( stdout );
    qout.setCodec ( "UTF-8" );

    qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( "  ↵
Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;

    qout << "This program is free software: you can redistribute it and ↵
        /or modify it under" << endl;
    qout << "the terms of the GNU General Public License as published ↵
        by the Free Software" << endl;
    qout << "Foundation, either version 3 of the License, or (at your ↵
        option) any later" << endl;
    qout << "version.\n" << endl;

    qout << "This program is distributed in the hope that it will be ↵
        useful, but WITHOUT" << endl;
    qout << "ANY WARRANTY; without even the implied warranty of ↵
        MERCHANTABILITY or FITNESS" << endl;
    qout << "FOR A PARTICULAR PURPOSE. See the GNU General Public ↵
        License for more details.\n" << endl;

    qout << QString::fromUtf8 ( "Ez a program szabad szoftver; ↵
        terjeszthető illetve módosítható a Free Software" ) << endl;
    qout << QString::fromUtf8 ( "Foundation által kiadott GNU General ↵
        Public License dokumentumában leírtak;" ) << endl;
    qout << QString::fromUtf8 ( "akár a licenc 3-as, akár (tetszőleges) ↵
        későbbi változata szerint.\n" ) << endl;

    qout << QString::fromUtf8 ( "Ez a program abban a reményben kerül ↵
        közreadásra, hogy hasznos lesz, de minden" ) << endl;
    qout << QString::fromUtf8 ( "egyéb GARANCIA NÉLKÜL, az ↵
        ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ" ) << endl;
    qout << QString::fromUtf8 ( "ALKALMAZHATÓSÁGRA való származtatott ↵
```

```
garanciát is beleértve. További" ) << endl;
qout << QString::fromUtf8 ( "részleteket a GNU General Public ←
License tartalmaz.\n" ) << endl;

qout << "http://gnu.hu/gplv3.html" << endl;

QRect rect = QApplication::desktop()->availableGeometry();
BrainBWin brainBWin ( rect.width(), rect.height() );
brainBWin.setWindowState ( brainBWin.windowState() ^ Qt::←
WindowFullScreen );
brainBWin.show();
return app.exec();
}
```

A mainben egy csomo kiíratást látunk, de nem a szokásos módon a cout segítségével, hanem a qout parancssal. A kiíratások a program licenszén alapulnak, illetve annak a helyes használatáról és továbbadásáról.

```
QT += widgets core
CONFIG += c++11 c++14 c++17
QMAKE_CXXFLAGS += -fopenmp
LIBS += -fopenmp
LIBS += `pkg-config --libs opencv`  
  
TEMPLATE = app
TARGET = BrainB
INCLUDEPATH += .  
  
HEADERS += BrainBThread.h BrainBWin.h
SOURCES += BrainBThread.cpp BrainBWin.cpp main.cpp
```

A project fájlnak ugyanaza a szerepe mint az előző programok esetében. Egy helyre begyűjt azoknak a programoknak a neveit amiket majd használni fog a BrainB összehozásához.

Fordítás: qmake BrainB.pro

make

Futtatás: ./BrainB

```
laci@laci-GL503VD: ~/programok/BrainB-Bench
Fájl Szerkesztés Nézet Keresés Terminál Súgó
http://gnu.hu/gplv3.html
laci@laci-GL503VD:~/programok/BrainB-Bench$ ./BrainB

NEMESPOR BrainB Test Copyright (C) 2017, 2018 Norbert Bátfai
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.

Ez a program szabad szoftver; terjeszthető illetve módosítható a Free Software
Foundation által kiadott GNU General Public License dokumentumában leírtak;
akár a licenc 3-as, akár (tetszőleges) későbbi változata szerint.

Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz, de minden
egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ
ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve. További
részleteket a GNU General Public License tartalmaz.

http://gnu.hu/gplv3.html
laci@laci-GL503VD:~/programok/BrainB-Bench$
```

7.10. ábra. Futtatás



7.11. ábra. Gameplay

Megnyitás ▾  Test-281-stats.txt ~/programok/BrainB... Mentés ⌂ ⌂ ⌂

```
NEMESPOR BrainB Test 6.0.3
time      : 282
bps       : 0
noc       : 1
nop       : 0
lost      :
16890 8840 6120 6080 2830 8390 60 4140 0 0 0 0
mean      : 4445
var       : 5189.13
found     : 5300 0 1370 7340
mean      : 3502
var       : 3404.53
lost2found: 5300 0
mean      : 2650
var       : 3747.67
found2lost: 6080 8390
mean      : 7235
var       : 1633.42
mean(lost2found) < mean(found2lost)
time      : 0:28
U R about 0.603333 Kilobytes
```

zöveg ▾ Tabulátorszélesség: 8 ▾ 1. sor, 1. oszlop ▾ BESZ

7.12. ábra. Eredmény txt

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...

Az MNIST egy olyan program ami a Tensorflow alapot használja. A tensorflow egy mesterséges intelektuálási rendszert használ, amelynek saját feladata a kézírás számok felismerése. A program a tensorflow környezetben futtatva a MNIST datasett használva tanítja a gépet, hogy hogyan kell a számokat felismerni. Minél több ideig engedjük futtatni a programot, annál pontosabb lesz a számok felismerésével.

Na de nem is húzom tovább az időt ugorunk fejest a forráskódba.

```
import tensorflow as tf
```

Itt beimportáljuk a tensorflow deep learning könyvtárát. A tensorok igazából csak több dimenziós array-ek.

```
mnist = tf.keras.datasets.mnist
```

Itt hívjuk le a az mnist dataset-et ami 28x28 pixel nagyságú kézzel írt számok amiknek az értékük a címük.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Kicsomagoljuk a képeket az x_train/x_test -be és a y_train/y_test -be

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Az x_train, x_test és x_train scaling-jét beállítjuk 1-255-re

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
```

```
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10)  
])
```

Modellünket ezen a részen állítjuk be első sorba beállítjuk a Sequential modelre ami egy basic feed-forward model. Flatten fogja a kapott 28x28 képeket és "kilapítja" 1x784-re. Következő sorba a neuronokat állítjuk be a mi esetünkbe 128 layer teljes összeköttetéssel. A harmadik sorba akadályozzuk meg overfitting-et. Utolsó sorba a dense kiértékeli a mátrxiokat bias értékükkel.

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

A losst beállítjuk egy alapértelmezett SparseCategoricalCrossentropy módra ami annyit tesz hogy a programunk a pontosságra fog törekedni.

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

Itt compile-oljuk a modellünket az optimezert az adam-ra állítjuk, lossnak megadjuk a loss_fn-t és a metric-el meg beállítjuk mit trackeljen.

```
model.fit(x_train, y_train, epochs=5)  
  
model.evaluate(x_test, y_test, verbose=2)  
  
probability_model = tf.keras.Sequential([  
    model,  
    tf.keras.layers.Softmax()  
])
```

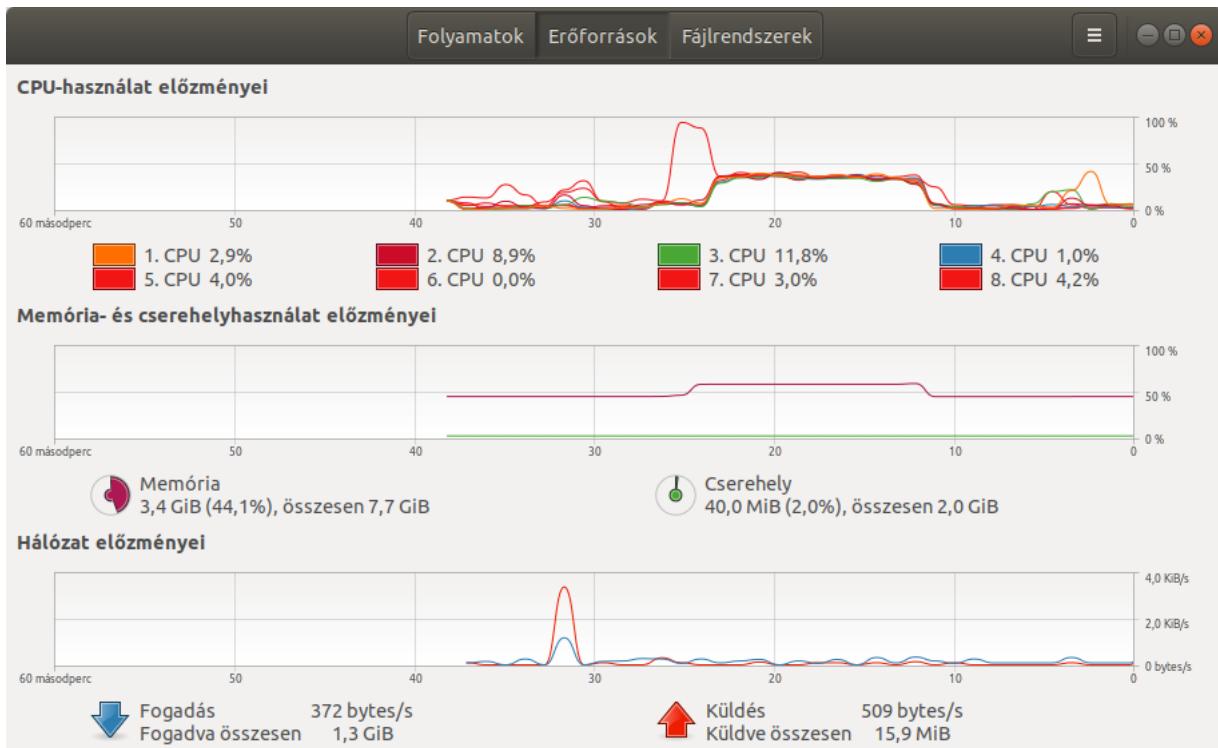
A model.fit-el megadjuk hogy hányszor füssünk át a modellekkel minnél többet adunk meg annál pontosabb eredményeket fogunk kapni de annál hosszabb lesz a program futási ideje.

```
laci@laci-GL503VD: ~/programok/tensorflow
Fájl Szerkesztés Nézet Keresés Terminál Súgó
laci@laci-GL503VD:~/programok/tensorflow$ python3 mnist.py
2020-04-29 10:54:50.773456: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libnvinfer.so.6'; dlerror: libnvinfer.so.6: cannot open shared object file: No such file or directory
2020-04-29 10:54:50.773568: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libnvinfer_plugin.so.6'; dlerror: libnvinfer_plugin.so.6: cannot open shared object file: No such file or directory
2020-04-29 10:54:50.773578: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:30] Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing libraries mentioned above are installed properly.
2020-04-29 10:54:51.528246: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1
2020-04-29 10:54:51.544893: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-04-29 10:54:51.545316: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1555] Found device 0 with properties:
pciBusID: 0000:01:00.0 name: GeForce GTX 1050 computeCapability: 6.1
coreClock: 1.493GHz coreCount: 5 deviceMemorySize: 3.95GiB deviceMemoryBandwidth: 104.43GiB/s
2020-04-29 10:54:51.545403: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libcudart.so.10.1'; dlerror: libcudart.so.10.1: cannot open shared object file: No such file or directory
2020-04-29 10:54:51.545481: W tensorflow/stream_executor/platform/default/dso
```

8.1. ábra. Futtatás

```
laci@laci-GL503VD: ~/programok/tensorflow
Fájl Szerkesztés Nézet Keresés Terminál Súgó
60000/60000 [=====] - 2s 37us/sample - loss: 0.0882 - accuracy: 0.9725
Epoch 5/5
 32/60000 [.....] - ETA: 2s - loss: 0.0922 - accuracy
 1376/60000 [.....] - ETA: 2s - loss: 0.0775 - accuracy
 2784/60000 [>.....] - ETA: 2s - loss: 0.0749 - accuracy
 4192/60000 [=>.....] - ETA: 2s - loss: 0.0704 - accuracy
 5568/60000 [=>.....] - ETA: 1s - loss: 0.0720 - accuracy
 6976/60000 ==>.....] - ETA: 1s - loss: 0.0701 - accuracy
 8384/60000 ==>.....] - ETA: 1s - loss: 0.0703 - accuracy
 9728/60000 ==>.....] - ETA: 1s - loss: 0.0705 - accuracy
11136/60000 ==>.....] - ETA: 1s - loss: 0.0701 - accuracy
12544/60000 ==>.....] - ETA: 1s - loss: 0.0680 - accuracy
13920/60000 ==>.....] - ETA: 1s - loss: 0.0669 - accuracy
15296/60000 ==>.....] - ETA: 1s - loss: 0.0677 - accuracy
16672/60000 ==>.....] - ETA: 1s - loss: 0.0699 - accuracy
18048/60000 ==>.....] - ETA: 1s - loss: 0.0702 - accuracy
19456/60000 ==>.....] - ETA: 1s - loss: 0.0701 - accuracy
20864/60000 ==>.....] - ETA: 1s - loss: 0.0698 - accuracy
22144/60000 ==>.....] - ETA: 1s - loss: 0.0709 - accuracy
23424/60000 ==>.....] - ETA: 1s - loss: 0.0713 - accuracy
24800/60000 ==>.....] - ETA: 1s - loss: 0.0714 - accuracy
26208/60000 ==>.....] - ETA: 1s - loss: 0.0725 - accuracy
27616/60000 ==>.....] - ETA: 1s - loss: 0.0742 - accuracy
29024/60000 ==>.....] - ETA: 1s - loss: 0.0739 - accuracy
30432/60000 ==>.....] - ETA: 1s - loss: 0.0729 - accuracy
31808/60000 ==>.....] - ETA: 1s - loss: 0.0741 - accuracy
33184/60000 ==>.....] - ETA: 0s - loss: 0.0745 - accuracy
34560/60000 ==>.....] - ETA: 0s - loss: 0.0756 - accuracy
35936/60000 ==>.....] - ETA: 0s - loss: 0.0748 - accuracy
37280/60000 ==>.....] - ETA: 0s - loss: 0.0746 - accuracy
38656/60000 ==>.....] - ETA: 0s - loss: 0.0747 - accuracy
40032/60000 ==>.....] - ETA: 0s - loss: 0.0752 - accuracy
41408/60000 ==>.....] - ETA: 0s - loss: 0.0750 - accuracy
42752/60000 ==>.....] - ETA: 0s - loss: 0.0747 - accuracy
44160/60000 ==>.....] - ETA: 0s - loss: 0.0746 - accuracy
45536/60000 ==>.....] - ETA: 0s - loss: 0.0742 - accuracy
46976/60000 ==>.....] - ETA: 0s - loss: 0.0737 - accuracy
48384/60000 ==>.....] - ETA: 0s - loss: 0.0736 - accuracy
49792/60000 ==>.....] - ETA: 0s - loss: 0.0735 - accuracy
51168/60000 ==>.....] - ETA: 0s - loss: 0.0738 - accuracy
52544/60000 ==>.....] - ETA: 0s - loss: 0.0737 - accuracy
53984/60000 ==>.....] - ETA: 0s - loss: 0.0742 - accuracy
55392/60000 ==>.....] - ETA: 0s - loss: 0.0743 - accuracy
56768/60000 ==>.....] - ETA: 0s - loss: 0.0748 - accuracy
58176/60000 ==>.....] - ETA: 0s - loss: 0.0750 - accuracy
59584/60000 ==>.....] - ETA: 0s - loss: 0.0747 - accuracy
60000/60000 [=====] - 2s 37us/sample - loss: 0.0745 - accuracy: 0.9769
10000/10000 - 0s - loss: 0.0728 - accuracy: 0.9762
laci@laci-GL503VD:~/programok/tensorflow$
```

8.2. ábra. Eredmény



8.3. ábra. Terhelés

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A mély MNIST ugyanolyan mint az MNIST csak training közbe jobban javul a pontossága és létrehoz egy gráfot a fejlődéséről amit nemek a C:\Users\asus\AppData\Local\Temp\ amit tensorflow meg lehet nyitni.

A feladat amúg nagyon megdolgoztatja a gépet mint a screenshotomról is látható hogy a 4 magos 8 szálas gépemen 100% -osan leterheli és a memóiát is nagyon szereti

```

Extracting /tmp/tensorflow/mnist/input_data\train-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data\train-labels-idx1-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data\t10k-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data\t10k-labels-idx1-ubyte.gz
Saving graph to: C:\Users\asus\AppData\Local\Temp\tmpoplx9xf
step 0, training accuracy 0.04
step 100, training accuracy 0.9
step 200, training accuracy 0.86
step 300, training accuracy 0.94
step 400, training accuracy 0.9

```

```
step 500, training accuracy 0.98
step 600, training accuracy 0.96
step 700, training accuracy 0.94
step 800, training accuracy 0.94
step 900, training accuracy 1
step 1000, training accuracy 1
step 1100, training accuracy 0.96
step 1200, training accuracy 0.98
step 1300, training accuracy 0.94
step 1400, training accuracy 0.96
step 1500, training accuracy 0.96
step 1600, training accuracy 0.94
step 1700, training accuracy 1
step 1800, training accuracy 1
step 1900, training accuracy 0.98
step 2000, training accuracy 0.98
step 2100, training accuracy 0.98
step 2200, training accuracy 0.98
step 2300, training accuracy 0.98
step 2400, training accuracy 0.98
step 2500, training accuracy 0.98
step 2600, training accuracy 0.98
step 2700, training accuracy 0.96
step 2800, training accuracy 1
step 2900, training accuracy 1
step 3000, training accuracy 1
step 3100, training accuracy 0.94
step 3200, training accuracy 1
step 3300, training accuracy 1
step 3400, training accuracy 0.94
step 3500, training accuracy 0.98
step 3600, training accuracy 1
step 3700, training accuracy 0.98
step 3800, training accuracy 0.94
step 3900, training accuracy 0.9
step 4000, training accuracy 0.98
step 4100, training accuracy 0.98
step 4200, training accuracy 1
step 4300, training accuracy 0.96
step 4400, training accuracy 1
```

```
# 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.  
# ↵  
=====
```

"""A deep MNIST classifier using convolutional layers.
See extensive documentation at
https://www.tensorflow.org/get_started/mnist/pros
"""

Disable linter warnings to maintain consistency with tutorial.
pylint: disable=invalid-name
pylint: disable=g-bad-import-order

```
from __future__ import absolute_import  
from __future__ import division  
from __future__ import print_function  
import argparse  
import sys  
import tempfile  
from tensorflow.examples.tutorials.mnist import input_data  
import tensorflow as tf  
FLAGS = None
```

def deepnn(x):
 """deepnn builds the graph for a deep net for classifying digits.
 Args:
 x: an input tensor with the dimensions (N_examples, 784), where 784 is ←
 the
 number of pixels in a standard MNIST image.
 Returns:
 A tuple (y, keep_prob). y is a tensor of shape (N_examples, 10), with ←
 values
 equal to the logits of classifying the digit into one of 10 classes (←
 the
 digits 0-9). keep_prob is a scalar placeholder for the probability of
 dropout.
 """

```
# Reshape to use within a convolutional neural net.  
# Last dimension is for "features" - there is only one here, since images ←  
# are  
# grayscale -- it would be 3 for an RGB image, 4 for RGBA, etc.  
with tf.name_scope('reshape'):  
    x_image = tf.reshape(x, [-1, 28, 28, 1])  
    # First convolutional layer - maps one grayscale image to 32 feature maps ←  
    .  
    with tf.name_scope('conv1'):  
        W_conv1 = weight_variable([5, 5, 1, 32])  
        b_conv1 = bias_variable([32])
```

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
# Pooling layer - downsamples by 2X.
with tf.name_scope('pool1'):
    h_pool1 = max_pool_2x2(h_conv1)
# Second convolutional layer -- maps 32 feature maps to 64.
with tf.name_scope('conv2'):
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
# Second pooling layer.
with tf.name_scope('pool2'):
    h_pool2 = max_pool_2x2(h_conv2)
# Fully connected layer 1 -- after 2 round of downsampling, our 28x28 ←
# image
# is down to 7x7x64 feature maps -- maps this to 1024 features.
with tf.name_scope('fc1'):
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])
    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
# Dropout - controls the complexity of the model, prevents co-adaptation ←
# of
# features.
with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
# Map the 1024 features to 10 classes, one for each digit
with tf.name_scope('fc2'):
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])
    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
    return y_conv, keep_prob
def conv2d(x, W):
    """conv2d returns a 2d convolution layer with full stride."""
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
    """max_pool_2x2 downsamples a feature map by 2X."""
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
def weight_variable(shape):
    """weight_variable generates a weight variable of a given shape."""
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
def bias_variable(shape):
    """bias_variable generates a bias variable of a given shape."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
def main(_):
    # Import data
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
```

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
# Build the graph for the deep net
y_conv, keep_prob = deepnn(x)
with tf.name_scope('loss'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                             logits=y_conv)
cross_entropy = tf.reduce_mean(cross_entropy)
with tf.name_scope('adam_optimizer'):
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)
graph_location = tempfile.mkdtemp()
print('Saving graph to: %s' % graph_location)
train_writer = tf.summary.FileWriter(graph_location)
train_writer.add_graph(tf.get_default_graph())
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0})
            print('step %d, training accuracy %g' % (i, train_accuracy))
            train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
            print('test accuracy %g' % accuracy.eval(feed_dict={
                x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
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, unparsed = parser.parse_known_args()
tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```



A screenshot of a Jupyter Notebook interface. The notebook title is "Untitled". The code in cell 18 is:

```
In [18]: import matplotlib.pyplot as plt  
plt.imshow(x_train[0], cmap = plt.cm.binary)  
plt.show()  
print(x_train[0])
```

The output shows a 28x28 pixel grayscale image of a handwritten digit, followed by its 28x28 pixel array representation:

```
[0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0.]
```

8.4. ábra.

A screenshot of a Jupyter Notebook interface. The notebook title is "Untitled". The code in cell 17 is:

```
In [17]: import tensorflow as tf # deep learning library. Tensors are just multi-dimensional arrays  
mnist = tf.keras.datasets.mnist # mnist is a dataset of 28x28 images of handwritten digits and their labels  
x_train, y_train, x_test, y_test = mnist.load_data() # unpacks images to x_train/x_test and labels to y_train/y_test  
train = tf.keras.utils.normalize(x_train, axis=1) # scales data between 0 and 1  
test = tf.keras.utils.normalize(x_test, axis=1) # scales data between 0 and 1  
  
del = tf.keras.models.Sequential() # a basic feed-forward model  
del.add(tf.keras.layers.Flatten()) # takes our 28x28 and makes it 1x784  
del.add(tf.keras.layers.Dense(128, activation=tf.nn.relu)) # a simple fully-connected layer, 128 units, relu activation  
del.add(tf.keras.layers.Dense(128, activation=tf.nn.relu)) # a simple fully-connected layer, 128 units, relu activation  
del.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax)) # our output layer. 10 units for 10 classes. Softmax for probability  
  
del.compile(optimizer='adam', # Good default optimizer to start with  
            loss='sparse_categorical_crossentropy', # how will we calculate our "error." Neural network aims to minimize loss.  
            metrics=['accuracy']) # what to track  
  
del.fit(x_train, y_train, epochs=3) # train the model  
  
l_loss, val_acc = model.evaluate(x_test, y_test) # evaluate the out of sample data with model  
int(val_loss) # model's loss (error)  
int(val_acc) # model's accuracy
```

The output shows the training progress and final accuracy:

```
Epoch 1/3  
60000/60000 [=====] - 4s 65us/sample - loss: 0.2676 - acc: 0.9205  
Epoch 2/3  
60000/60000 [=====] - 4s 61us/sample - loss: 0.1103 - acc: 0.9661  
Epoch 3/3  
60000/60000 [=====] - 4s 60us/sample - loss: 0.0745 - acc: 0.9762  
10000/10000 [=====] - 0s 30us/sample - loss: 0.0975 - acc: 0.9712  
0.9754772678739392
```

8.5. ábra.

Név	Állapot	100% Processzor	62% Memória	0% Lemez	0% Hálózat	0% GPU
▼ Python (4)		93,1%	1 058,8 MB	0 MB/s	0 Mb/s	0%
Python		93,1%	988,8 MB	0 MB/s	0 Mb/s	0%
Python		0%	58,4 MB	0 MB/s	0 Mb/s	0%
Python		0%	6,7 MB	0 MB/s	0 Mb/s	0%
Konzolablak-kezelő		0%	4,9 MB	0 MB/s	0 Mb/s	0%

8.6. ábra. Terhelés

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

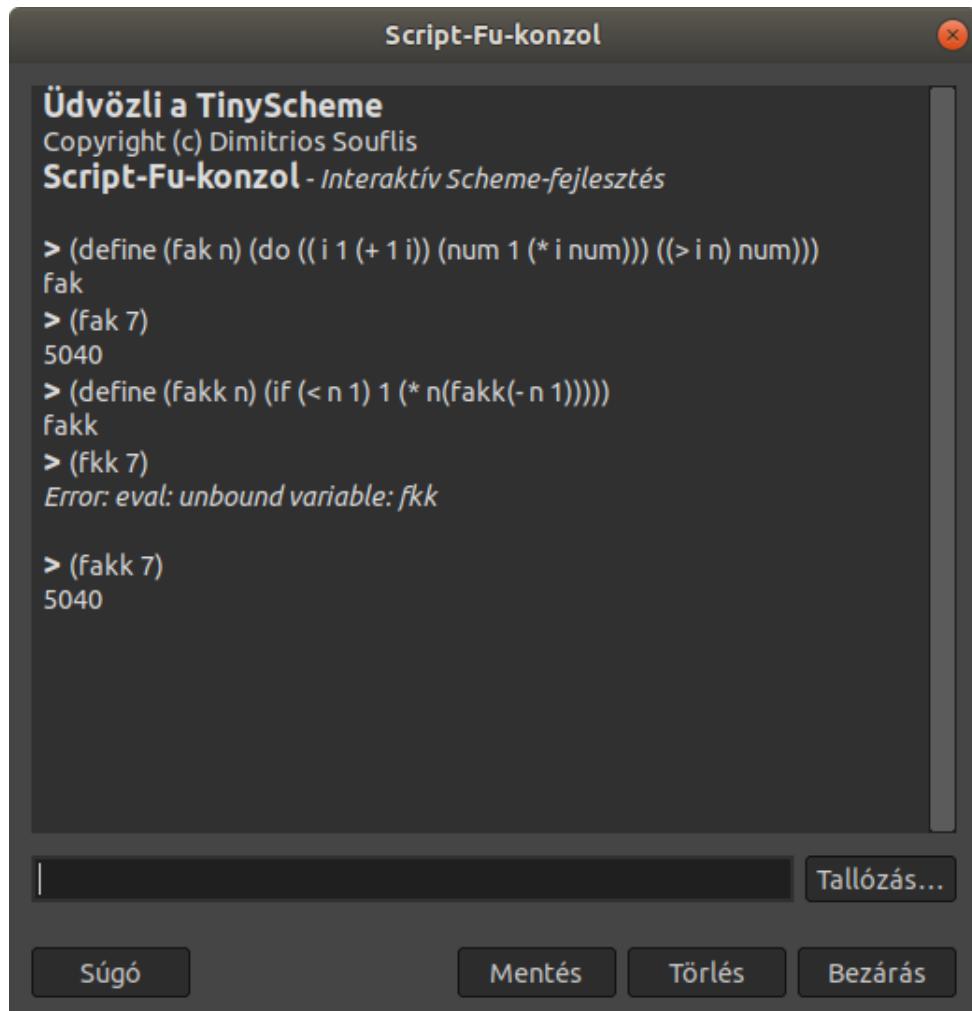
A lisp az alapból egy információ feldolgozó nyelvnek szánták de az évek alatt inkább mesterséges intelligenciai algoritmusok írására. Ma a két fegyverjebb lisp verziók azok a Common Lisp és a Scheme. A Common Lisp az egy multiparadigmás nyelv ami miatt inkrementális és evolúciós fejlesztésre használják. A Scheme pedig egy leegyszerűsített lisp amelyből minden "fölösleges" tulajdonságát proóbáltak kivenni és emiatt a nyelv tanításra tökéletes.

Elsőnek is az iteratív jelentése az hogy ismétlődő ezért is használtam a do paracsot

```
> (define (fak n) (do ((i 1 (+ 1 i)) (num 1 (* i num))) ((> i n) num)))
fak
> (fak 7)
5040
```

A rekurzív pedig olyan hogy a program addig hívja meg magát ameddig nem kapja meg a problmára a választ

```
> (define (fakk n) (if (< n 1) 1 (* n(fakk(- n 1)))))
fakk
> (fakk 7)
5040
```



The screenshot shows a terminal-like window titled "Script-Fu-konzol". The title bar also includes the text "Üdvözli a TinyScheme" and "Copyright (c) Dimitrios Souflis". Below the title, it says "Script-Fu-konzol - Interaktív Scheme-fejlesztés". The main area contains Scheme code and its output:

```
> (define (fak n) (do ((i 1 (+ 1 i)) (num 1 (* i num))) ((> i n) num)))
fak
> (fak 7)
5040
> (define (fakk n) (if (< n 1) 1 (* n(fakk(- n 1)))))
fakk
> (fkk 7)
Error: eval: unbound variable: fkk

> (fakk 7)
5040
```

At the bottom of the window, there is a text input field with a cursor, and several buttons: "Súgó", "Mentés", "Törles", and "Bezáras".

9.1. ábra. Script Fu futtatás

9.2. Gimp Scheme Script-fu: króm effekt

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

Megoldás videó: https://youtu.be/OKdAkl_c7Sc

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

Tanulságok, tapasztalatok, magyarázat...

A Gimp egy primitív képszerkesztő program amelyben lehet scripteket irni előző feladatban beszélt Lisp nyelven azon belül is a Scheme nyelven. Tanárúr által kreált skriptet csak be kell másolni a GIMP 2\share\gimp\2.0\scripts nevű mappába indítani a gimpet ráklikkelünk hogy fájl -> létrehozás -> bhax -> Chrome rá kiklikkelve beállíthatjuk: szöveget,betútpus,betűtipus méretét,kép magassága,kép szélessége,a színt, és hogy milyen effectel legyen chromosítva. Ezek után létrehozás és a script legenerál egy képet amin a megadott szövegnek chrome mintája van.

Chrome írás

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)

; (color-curve)

(define (elem x lista)

  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) ) )

)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )

  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))

  (list text-width text-height)
  )
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-chrome text font fontsize width height color ←
  gradient)
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
      LAYER-MODE-NORMAL-LEGACY)))
  
```

```
(textfs)
  (text-width (car (text-wh text font fontsize)))
  (text-height (elem 2 (text-wh text font fontsize)))
  (layer2)
)

;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND )
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
  ))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
  height 2) (/ text-height 2)))

(set! layer (car(gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←
  LAYER) ))

;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)

;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
  LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)

;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT- ←
  LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
      3)))
;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
  0 TRUE FALSE 2)
```

```
;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)

; (script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 '(255 0 0) " ←
  Crown molding")

(script-fu-register "script-fu-bhax-chrome"
  "Chrome3"
  "Creates a chrome effect on a given text."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 19, 2019"
  ""
  SF-STRING      "Text"      "Bátf41 Haxor"
  SF-FONT        "Font"       "Sans"
  SF-ADJUSTMENT   "Font size"  '(100 1 1000 1 10 0 1)
  SF-VALUE        "Width"     "1000"
  SF-VALUE        "Height"    "1000"
  SF-COLOR        "Color"     '(255 0 0)
  SF-GRADIENT     "Gradient"  "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```





9.2. ábra. Chrome írás

Chrome Border

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)
; (color-curve)

(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-wh text font fontsize)
(let*
```

```
(  
  (text-width 1)  
  (text-height 1)  
)  
  
(set! text-width (car (gimp-text-get-extents-fontname text fontsize ←  
  PIXELS font)))  
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ←  
  fontsize PIXELS font)))  
  
(list text-width text-height)  
)  
)  
  
;(text-width "alma" "Sans" 100)  
  
(define (script-fu-bhax-chrome-border text font fontsize width height new- ←  
  width color gradient border-size)  
(let*  
  (  
    (text-width (car (text-wh text font fontsize)))  
    (text-height (elem 2 (text-wh text font fontsize)))  
    (image (car (gimp-image-new width (+ height (/ text-height 2)) 0)))  
    (layer (car (gimp-layer-new image width (+ height (/ text-height 2) ←  
      ) RGB-IMAGE "bg" 100 LAYER-MODE-NORMAL-LEGACY)))  
    (textfs)  
    (layer2)  
  )  
  
(gimp-image-insert-layer image layer 0 0)  
  
(gimp-image-select-rectangle image CHANNEL-OP-ADD 0 (/ text-height 2) ←  
  width height)  
(gimp-context-set-foreground '(255 255 255))  
(gimp-drawable-edit-fill layer FILL-FOREGROUND )  
  
(gimp-image-select-rectangle image CHANNEL-OP-REPLACE border-size (+ (/ ←  
  text-height 2) border-size) (- width (* border-size 2)) (- height ←  
  (* border-size 2)))  
(gimp-context-set-foreground '(0 0 0))  
(gimp-drawable-edit-fill layer FILL-FOREGROUND )  
  
(gimp-image-select-rectangle image CHANNEL-OP-REPLACE (* border-size 3) ←  
  0 text-width text-height)  
(gimp-drawable-edit-fill layer FILL-FOREGROUND )  
  
(gimp-selection-none image)  
  
;step 1  
(gimp-context-set-foreground '(255 255 255))
```

```
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (* border-size 3) 0)

(set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←
    LAYER)))
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 25 TRUE TRUE)

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .18 .38 TRUE 1 0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)

;step 6
(set! layer2 (car (gimp-layer-new image width (+ height (/ text-height ←
    2)) RGB-IMAGE "2" 100 LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)

;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT- ←
    LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width 0 width (+ height (/ text-height 2)))

;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
    0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-image-scale image new-width (/ (* new-width (+ height (/ text- ←
    height 2))) width))

(gimp-display-new image)
(gimp-image-clean-all image)
)
;

;(script-fu-bhax-chrome-border "Bátf41 Haxor Stream" "Sans" 160 1920 1080 ←
  400 '(255 0 0) "Crown molding" 7)
```

```
; (script-fu-bhax-chrome-border "Programozás" "Sans" 110 768 576 300 '(255 0 ←
0) "Crown molding" 6)

(script-fu-register "script-fu-bhax-chrome-border"
"Chrome3-Border2"
"Creates a chrome effect on a given text."
"Norbert Bátfai"
"Copyright 2019, Norbert Bátfai"
"January 19, 2019"
"""

SF-STRING      "Text"      "Bátf41 Haxor"
SF-FONT        "Font"       "Sans"
SF-ADJUSTMENT  "Font size"  '(160 1 1000 1 10 0 1)
SF-VALUE        "Width"     "1920"
SF-VALUE        "Height"    "1080"
SF-VALUE        "New width" "400"
SF-COLOR       "Color"     '(255 0 0)
SF-GRADIENT    "Gradient"  "Crown molding"
SF-VALUE        "Border size" "7"
)
(script-fu-menu-register "script-fu-bhax-chrome-border"
"<Image>/File/Create/BHAX"
)
```



9.3. ábra. Chrome border

9.3. Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!
Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelete_a_scheme_programozasi_nyelv
Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala
Tanulságok, tapasztalatok, magyarázat...

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  text-width
)
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  ;;;
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  ;;; ved ki a lista 2. elemét
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
  ;;;
  (list text-width text-height)
)
)

;(text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
  gradient)
```

```
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
      LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-layer)
    (text-width (text-width text font fontsize))
    ;;
    (text2-width (car (text-wh text2 font fontsize)))
    (text2-height (elem 2 (text-wh text2 font fontsize)))
    ;;
    (textfs-width)
    (textfs-height)
    (gradient-layer)
  )
  (gimp-image-insert-layer image layer 0 0)

  (gimp-context-set-foreground '(0 255 0))
  (gimp-drawable-fill layer FILL-FOREGROUND)
  (gimp-image-undo-disable image)

  (gimp-context-set-foreground color)

  (set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
  (gimp-image-insert-layer image textfs 0 -1)
  (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
    height 2))
  (gimp-layer-resize-to-image-size textfs)

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
  (set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
  (set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
  (set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER))))
```

```
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
-LAYER)))

(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car (gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car (gimp-drawable-height textfs)) 100))

(gimp-layer-resize-to-image-size textfs)

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ←
"gradient" 100 LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ (/ ←
width 2) (/ textfs-width 2)) 8) (/ height 2))

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
)))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
```

```
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
    height 2) (/ text2-height 2)))

; (gimp-selection-none image)
; (gimp-image-flatten image)

(gimp-display-new image)
(gimp-image-clean-all image)
)

;

; (script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Ruge Boogie" 120 1920 ←
  1080 '(255 0 0) "Shadows 3")

(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"          "Bátf41 Haxor"
  SF-STRING      "Text2"         "BHAX"
  SF-FONT        "Font"          "Sans"
  SF-ADJUSTMENT   "Font size"     '(100 1 1000 1 10 0 1)
  SF-VALUE        "Width"         "1000"
  SF-VALUE        "Height"        "1000"
  SF-COLOR        "Color"         '(255 0 0)
  SF-GRADIENT    "Gradient"      "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```





9.4. ábra. Mandala

DPL

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

Ha a programozási alapfogalmakról beszélünk akkor fontos említeni arról hogy milyen szintekre tudjuk őket osztani. Az első szint az a gépi nyelv mely 0-kból és 1-ből álló kód. Ezt követi az assembly szintű nyelv mely közelebb van az általunk beszélt nyelvhez de még messze a könnyű használattól ezt nevezzük alacsony szintű nyelveknek. A harmadik típus a magas szintű programozási nyelvek ezek a nyelvek álnak a legközelebb az ember által értelmezhető nyelvkhöz tipp angol erősen ajánlott mert angolul vannak általában a programozási nyelvek.

A magas szintű programozási nyelven írt programot forrásprogramnak nevezzük. Ennek az létrehozáskor be kell tartani bizonyos az adott nyelvre jellemző formai, szintaktikai szabályokat és a tartalmi, szemantikai szabályokat.

A proceszorok saját gépi nyelvvel rendelkeznek, és csak az ezen írt programokat képesek végrehajtani. Tehát a forráskódokat át kell alakítani olyan kódra amit a számítógépünk is megért. Erre két módunk is van az egyik az analóg a másik megoldás pedig az interpreteres.

A fordítóprogram egyetlen egységeként kezeli a forrást, és lexikai, szintaktikai, szemantikai elemzést hajt végre, majd legenerálja a gépi kódot. Ez még nem futtatható, ebből a kapcsolatszerkesztő állít elő futtatható programot, melyet a betöltő behelyez a tárba, és a futtató rendszer felügyeli a futását. Bizonyos esetekben lehetőség van arra, hogy nem nyelvi elemeket használunk egy forrásprogramban, de ilyenkor szükség van egy előfordítónak.

Az interpreteres megoldás nem készít tárgykódot, viszont a fentebb említett 3 elemzést végrehajtja. Utasításonként sorra veszi a forrásprogramot, értelmezi, és végrehajtja. tehát rögtön kapjuk meg eredményt. Bizonyos nyelvek esetén mind az interpreteres, mind a fordítóprogramos megoldást alkalmazzák.

Minden programozási nyelvhez tartozik egy hivatkozási nyelv, mely a szemantikai és szintaktikai szabályokat határozza meg. Emellett léteznek még implementációk. Az egyes rendszereken több fordítóprogram és interpreter létezik, és ennek következtében az implementációk nem kompatibilisek egymással, ez pedig meggátolja a programok tökéletes hordozását a platformok között.

A programozó dolgának megkönnyítése érdekében létrejöttek az integrált grafikus felületek(IDE), melyek egy csomagban tartalmaznak minden szükséges eszközt a programok megírásához, és futtatásához.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

A C nyelv utasításai

Utasítások és blokkok: A nyelv bármelyik kifejezése utasítássá válik ha pontosvesszőt teszünk a végére. Akárhány utasítást összefoglalhatunk egyetlen összetett utasításba a kapcsoszárójelek segítségével(pl.: while vagy if utáni több utasítás).

Az if-else utasítás: Döntést, választást írhatunk le a segítségével. Az else rész opcionális, azaz elhagyható. ha az if után zárójelben lévő kifejezés teljesül, akkor az Ő utána következő utasítások, vagy blokkba foglalt összetett utasítás fog lezajlani, egyébként pedig az else kulcsszót követő utasítások lépnek érvénybe. Előfordulhat hogy 'egyébként' semmint nem kell tennünk, ilyenkor hagyható el az else. Ha több if utasítás van egymásba ágyazva nem minden egyértelmű hova tartoznak az else-k: minden a hozzájuk legközelebbi if-hez.

Az else-if kifejezés: Több feltételvizsgálatot köthetünk össze, melyeket sorban vizsgálunk és értékelünk ki. Egy if-else if-else láncból minden csak egy utasítás vagy összetett utasítás teljesül. Az else itt is elhagyható

A switch utasítás: A többirányú programelágaztatás egyik eszköze. Megvizsgálja hogy egy változó értéke megegyezik e több állandó értékkel és ennek megfelelően hajta végre a megfelelő utasításokat. A switch után megadott változó értékét az utána felsorolt (case:)-eket követő értékekkel hasonlítja össze és hajta végre a hozzá kapcsolódó utasításokat ha megegyeznek. Megadható egy default: opció, ami akkor lép érvénybe ha nem volt egyezés.

A while és a for utasítások: ciklusok, a while az Őt követő zárójelben lévő kifejezést vizsgálja, ha értéke nem 0 akkor végrehajtja az utasítást, majd újra tesztel, és így tovább.

a for utasítást követő zárójel általában 3 utasítást tartalmaz, ezekből az első és a harmadik értékkedés, a második pedig függvényhívás vagy relációs kifejezés. Ilyen ciklussal adott számú alkalommal futtathatjuk le a ciklusunkat.

A do-while utasítás: szintén ciklus, azonban ez egyszer mindenkor lefut, ha a végén lévő while feltétel utáni kifejezés 0 értékű akkor is. Első lefutása után azonban csak a feltétel vizsgálatának függvényében fut le többször.

A break utasítás: Ezen utasítás hatására a program kilép a legbelől ciklusból(for, while, do-while) vagy switchből.

A continue utasítás: Olyan utasítás ami csak ciklusokra értelmezhető, switchre nem. A ciklus következő iterációjára ugrik.

A goto utasítás: Segítségével a program egy adott címkéjű részére ugorhatunk a végrehajtásban. Ritkán használatos, általában egyszerűen helyettesíthető.

10.3. Programozás

[BMECPP]

A C++ nem objektum orientált újdonságai

Ilyen újdonságok például:

Függvény paraméterek és visszatérési érték: C++-ban meg kell adni paraméterként a void kulcsszót, ha nem szeretnénk paramétert(C-ben elég üresen hagyni), valamint ha C-ben nem írunk visszatérési értéket a program int-et vissza alapértelmezetten, C++-ban viszont ilyenkor a program hibás lesz.

A main függvény: paraméterlistája vagy üres, vagy tartalmazza az argc változót és az argv vektort, a return 0; elhagyható, ugyanis ha nem írjuk ki sikeres futás esetén alapértelmezett 0-val tér vissza a program.

A bool típus: Megadásra kerül a bool típus. Automatikus konverzió bool és int között. C++ kulcsszavak: bool, true, false.

C stílusú több bájtos stringek: használatukhoz nem kell #include-olni semmit, beépített típus lett.

Változódeklaráció mint utasítás: Változódeklaráció akárhol állhat ahol utasítás. Változót minden csak akkor hozzunk létre és adjunk neki kezdőértéket, amikor használni fogjuk.

Függvénynevek túlterhelése: C++-ban túlterhelhetünk függvényeket, azaz ugyan azon nevű függvény többször is létezhet feltéve ha más a paraméterlistája(visszatérési érték különbözősége nem számít). Ezeket a függvényeket úgy különbözteti meg a fordító, hogy a neveket kiegészíti az argumentumokból generált elő vagy utótaggal. Ezt a technológiát "névelferdítésnek" nevezzük. Ha egy C függvény szerenénk C++ programunkban használni akkor ezt a fordítónak az extern "C" kifejezéssel jelezhezjük.

Alapértelmezett függvényargumentumok: C++-ban függvényeink paramétereinek megadhatunk alapértelmezett értékeket, amiket a metódus akkor használ ha nem adtunk meg neki adott paraméterként semmit. Itt figyelembe kell vennünk néhány szabályt: alapértelmezett értéket az argumentumlistában hátulról előre haladva folytatónak adhatunk, hívás során az argumentumok ugyan ilyen sorrendben hagyhatóak el, valamint alapértelmezett érték nem adható meg a egyidejűleg a függvénydefiníció és a függvénydeklaráció helyén.

Paraméterátadás referenciátipussal: A C-vel ellentétben, ahol érték szerinti paraméterátadás történik, a C++-ban hivatkozás szerinti megoldás van megvalósítva a függvényeknél. Ezt akkor alkalmazzuk amikor módosítani is szeretnénk az adott paramétert, nem csak felhasználni. Ilyenkor a paraméterlistában a kívánt paraméter megtoldjuk egy 'és' jelrel, ezzel jelölve azt, hogy a paraméter referenciáját szeretnénk felhasználni.

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Berners-Lee!

11.1. C++ és Java összehasonlítása



Feladat megoldásához szükséges források

C++: Benedek Zoltán, Levendovszky Tíhamér Szoftverfejlesztés C++ nyelven

Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.

Ebből a két könyvből pár oldalas esszé jellegű kidolgozást kérek, Java és C++ összehasonlítás mentén, pl. kb.: kifejezés fogalom ua., Javában minden objektum referencia, minden dinamikus a kötés, minden függvény virtuális, klónozás stb.

Tanulságok, tapasztalatok, magyarázat...

A Java nyelv a C++ továbbgondolásának tekinthető. Az egyik legfontosabb különbség, hogy a Java interpereteres nyelv, tehát nem klasszikus fordítással nyerünk ki a forrásfájlból gépi kódot, hanem a Java interpereter segítségével. Viszont ez nem egy hagyományos értelemben vett gépi kódot hoz létre, hanem a Java Virtual Machine számára futtatható állományt. A JVM az egyik legnagyobb előnye a Javanak, mert ennek segítségével a bármely platformon elérhető, és hordozható kódot tudunk írni.

A Java és a C++ alapvető szintaxisa nagyon hasonlít. Érdekesség ellenben, hogy a Java-ban nincs meg a klasszikus main-fájl, ami tartalmazza a program futásához szükséges main függvényt. Ugyanis ez a függvény egy osztályba van beágyazva. Ez annak a következménye, hogy Java-ban minden metódus, változó valamilyen osztálynak a tagja. A main függvényt public-nak és static-nak kell deklarálni. Nem lehet visszatérési értéke, ennek következtében void, szemben a C++-szal. Egy másik eltérés, hogy a parancssori argumentumok.

Java-ban a típusokat két fő csoportra oszthatjuk, ezek a primitív és a nem primitív típusok. A primitív típusok magát az értéket tárolják. Ezzel szemben a nem primitív típusok objektumhivatkozások. A Java nyelv lehetővé teszi a primitív típusok "objektumosítását". Tehát az int típust helyettesíthetjük Integer típussal. Jelen esetben az Integer egy csomagoló osztály. A Java 5-ös verziójától lehetővé vált az ezen típusok közötti automatikus be- és kicsomagolás. Nem létezik a char* Java-ban, ehelyett egy beépített osztályt használhatunk, a String-et. A C++-al ellentétben a tömbök nem a pointerek másik megnyilvánulási formái, hanem ezek is valódi típusok. Ennek köszönhetően elérhető a méretüket lekérdező függvény is, így nem szükséges azt tárolni egy másik változóban.

Osztályt a `class` kulcsszóval tudunk deklarálni, ahogy azt már C++-ban megszoktuk. minden osztály a az `Object` osztály leszármazottja Java-ban. Fontos megjegyezni, hogy nem engedélyezett ellenben a többszörös öröklődés. A szülőre a `super` referenciával hivatkozhatunk. Az osztály tagjainak láthatóságát egyenként kell beállítani. A C++-ban megszokott destruktora Java-ban nincs szükség, mivel azt az automatikus szemétyűjtögető helyettesíti. A másoló konstruktor helyett Javaban lehetőségünk van a `clone` metódus használatára. Ez alapesetben sekély másolást hajt végre, ezért szükséges bizonyos esetekben az adattagok külön lemásolása azok elkülönítése. Csak azok az osztályok támogatják a másolást, amelyek implementálják a `Cloneable` interfész. Mivel Java-ban, amikor példányosítunk egy objektumot, akkor egy objektumhivatkozást hozunk létre, ami egy adott tárterületen lévő objektumra mutat. Fontos megjegyezni, hogy Java-ban nincsenek már pointerek, csak referenciák használata engedélyezett. Sablon osztályok helyett a Java bevezeti a generikusok fogalmát. A legfontosabb tulajdonsága a típus törlés, melynek lényege, hogy a paraméteres típus egy nem paraméteres típusra. Erre azért volt szükség, hogy a korábbi Java verziókkal való kompatibilitás ne sérüljön.

Az osztályok örökölhetnek a szülőosztálytól tagfüggvényeket. Bizonyos esetekben pedig előfordul, hogy ezeket szükséges felüldefiniálni. Ehhez C++-ban a `virtual` kulcsszóra volt szükség, viszont erre a Java-ban már nincs szükség. Ez annak dinamikus kötésnek köszönhetően. Tehát az interpreter a dinamikus taghoz tartozó implementációját használja a metódusnak. A dinamikus tag fogalom azzal van összefüggésben, hogy a poliformizmusnak miatt, olyan helyen ahol állhat a gyermek, ott állhat a szülő osztályú objektum is. Ennek köszönhetően előfordulhat, hogy például egy szülő osztályú objektumreferencia hivatkozhat gyermek osztályú objektumra, és viszont.

C++-ban ahoz, hogy extra saját vagy beépített osztályokat metódusokat lehessen elérni, header fájlokat kellett a forrásunkba include-álni. Java-ban nincsenek header állományok, helyettük vannak csomagok, melyeket az `import` kulcsszóval tudjuk hozzáadni a forráshoz. Csomagokat mi is hozhatunk létre, ehhez a fájl legelején a `package <név>` sort kell beírni. Többféleképpen importálhatunk. Lehetőségünk van a teljes osztályt, de még a publikus metódusait is hozzáadni. Fontos megjegyezni, hogy sokszor importálásra nem feltétlenül van szükség, csak tisztábbá teszi a kódot. Lehetőség van teljes fájlelérési úttal elérni egy csomag tartalmát. Annyi könnyedséget biztosít az interpreter, hogy elég az egyes mappákat, almappákat pontokkal elválasztani. Ezeknek a csomagoknak köszönhetően Java-ban könnyen lehet hálózatkezelést, felhasználói felületet készíteni.

11.2. Gyors prototípus-fejlesztés Python és Java nyelven



Feladat megoldásához szükséges források

Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba(35-51. oldal)

Itt a kijelölt oldalakból egy 1 oldalas élmény-olvasónaplóra gondoltam.

Tanulságok, tapasztalatok, magyarázat...

A Python programozási nyelv Guido van Rossum nevéhez fűződik, aki 1990-ben alkotta meg.

A már korábban tanult C, C++, nyelvekhez képest nagy előnye, hogy nincs szükség fordítóra. A forrásból az értelmező azonnal képes futtatni a programot. Ez a tulajdonsága abból következik, hogy a Python az egy

szkriptnyelv. A komolyabb feladatok megoldása érdekében rengeteg kiegészítőket tartalmaz. A fejlesztők életét jelentősen megkönnyíti a standar Python kódkönyvtár, mely modulokat tartalmaz fájlkezeléshez, hálózatkezeléshez, rendszerhívásokhoz és felhasználói felület kialakításához is.

A Python nyelv szintaxisa jelentősen egyszerűsített, mivel behúzás alapú. Tehát nincs szükség kapcsoszárojelekre a blokk elejének és végének jelöléséhez. A blokk addig tart ameddig az értelmező nem talál egy kisebb behúzású sort. Érdekesség még, hogy az utasításokat nem kell ";"-vel elválasztani, azok végét a sor vége jelzi. Persze előfordulhat, hogy egy utasítás nem fér ki egy sorba, ezt az értelmezőnek egy a sor végére írt "\"-jellel jelölhetjük. Ha nem zárunk be egy zárójelet, akkor is a következő sort az utasítás folytatásának veszi.

Python-ban az adatokat objektumok reprezentálják. A C, C++, Java nyelvekhez hasonlóan az adatokon végrehajtható műveletek az objektum típusától függnek. Eltéres ezzel szemben, hogy nem kell explicit módon megadni az objektum típusát, azt az értelmező "kitalálja". Az adattípusok többsége a fentebb említett nyelvekhez hasonló. Ennél érdekesebb az ennesek, listák és szótárak használata. Mindháromra igaz, hogy az objektumoknak nem kell azonos típusúnak lennie, szemben a C/C++-ból ismert tömbökkel, vektorokkal. Az ennesek lényegében objektumok gyűjteményei, melyeket veszzővel választunk el. Ezeket sima zárójelek közé írjuk. A listák egy rendezett sorozatai az objektumoknak, melyeket szintén vesszővel válsztunk el, és szögletes zárójellel határolunk. Az elemeket a listában az indexükkel azonosítjuk. A szótárak pedig az objektumok mellett kulcsokat is tartalmaznak, ezekkel tudjuk az egyes elemeket azonosítani. Ebben a nyelvben a változók objektumokra mutató referenciai. Mivel a változóknak nincs típusuk, ezért bármilyen objektumra mutathatnak. Az értékadás a már ismert "="-lel történik. A változóhoz árendeléseket a `def` kulcsosval lehetjük meg. A hátramaradt objektumokat a `garbage collector` törli. Pythonban is léteznek globális és lokális változók. Egy lokális változót a `global` kulcsszóval lehetünk globálissá. A beépített típusok között létezik típuskonverzió. Szekvenciákkal, ide tartoznak a listák, sztringek, ennesek, műveleteket hajthatunk végre, beépített függvényeket alkalmazhatunk.

A Python biztosítja a hagyományos programnyelvi eszközöket. Tudunk kírni szövegeket, változók értékét a konzolra. Létrehozhatunk `if` elágazásokat. Ebben eltérés, hogy az `else if` ágat `elif`-el jelöljük. A nyelv támogatja a ciklusokat, tehát a `while`, `for` ciklusok gond nélkül létrehozhatók. Egyedül a szintaktikájukban térnek el a C/C++-os tól. A `for` ciklusok esetén érdemes megemlíteni az `xrange` függvényt, amellyel megadhatjuk, hogy mennyi meddig szeretnénk futtani a ciklusváltozót, és azt is, hogy milyen lépésközzel. Készíthetünk címkeket is, melyet a `label` kulcsszóval jelölünk, és a `goto` utasítással tudjuk a vezérlést átugratni rá. Létezik egy másik kulcsszó is, a `comefrom`. Ennek a segítségével a címkeből vissza lehet ugrani a hozzá tartozó `comefrom` részhez.

A függvényeket a `def` kulcsszóval tudjuk definiálni. Ezeknek lehetnek paramétereik, melyek csak érték szerint adhatók át, kivéve a listákat, szótárakat, melyeken való módosítás a híváshelyen lévő objektumra is végrehajtódnak. A visszatérési értéküket nem kell expliciten megadni, ahogy már a változóknál megszoktuk. Csak egy értékkal tudnak visszatérni, de lehet akár ennesekkel is.

Mivel a Python támogatja az objektumorientált eljárásokat, ezért létrehozhatunk osztályokat is, melyet a szokásos `class` kulcsszóval definiálunk. Jelentős különbség nincs a C/C++-ban megszokott osztályok alapvető működéséhez képest. Viszont Python-ban nem használhatunk `private`, `public`, `protected` címkeket. Ezeket a tagváltozó nevével tudjuk helyettesíteni. Ha nem rakunk aláhúzást elő, akkor publikus lesz. Abban az esetben, ha egy aláhúzás van a változó név elején, akkor `protected`, kettő esetén pedig `private` tagváltozóról beszélünk. Az osztály konstruktora az `__init__` függvény adja. Ennek első paramétere a `self`, vagyis maga a létrehozandó objektum.

A Python a fejlesztés megkönnyítése érdekében modulokat tartalmaz. Ezeken belül vannak olyanok, melyek kifejezetten a mobilos fejlesztést segítik. Biztosítják a felhasználói felület kezelését(`appui fw`), háló-

zatkezelést(messaging, de a kamera(camera) elérését is megoldják. A sysinfo modul segítségével lekérdezhetjük a telefon adatait, az audio modul segítségével pedig hangfelvételeket hozhatunk létre, és azok lejátszását is biztosítja. Létrehozhatunk saját modulokat is, melyeket a programunkba importálhatunk.

A kivételkezelés is egy ismert fogalom a Python nyelvben. A vizsgált blokkot, ahonnan a kivétel érkezhet, a try címkével jelöljük. Hiba esetén a vezérlés az except címkével jelölt blokkra ugrik. Lehetséges még adni else ágat is, plusz az else és except helyett használhatjuk a finally címkét, melyhez tartozó kód rész főleg "takarításra" szolgál.

DRAFT

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.!

Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPORG repó: source/labor/polargen)

Tanulságok, tapasztalatok, magyarázat...

Az egyik korábbi fejezetben már néztük a C++, és Java forrást. Most azt fogjuk megnézni, hogy a hivatalos Java random szám generátor mennyire hasonlít a mi megoldásunkra.

Akkor jöjjön is a saját megoldásunk.

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

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
        if(nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do{
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2* u1 -1;
                v2 = 2* u2 -1;
                w = v1*v1 + v2*v2;
            }
            while(w >= 1);
            nincsTarolt = false;
            tarolt = Math.sqrt(w);
            return tarolt;
        }
        else
            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;
}
}
```

A program a futtatás során generál 2 random számot. Az egyiket eltárolja, a másikat pedig visszaadja. Majd ezek után megvizsgáljuk, hogy van-e tárolt elem. Ezért van szükségünk a nincsTarolt logikai változóra, mely alapértelmezettben igaz. A double függvény, annyit csinál, hogy megnézi az előző változót és ha igaz, akkor kiszámolja a számokat. Az egyik számot eltároljuk majd a nincsTarolt-at hamisra állítja. Így elkerülhetjük azt, hogy a programnak feleslegesen kelljen újra random számot generálnia.

Most nézzük meg az OpenJDK Random.java forrásában, hogyan oldották meg ezt.

```
private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

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

Ha a két kódot megnézzük, akkor arra jutunk, hogy megegyeznek.

A továbbiakban a C++-os megoldást fogjuk megnézni.

```
class PolarGen {
public:
    PolarGen(); //konstruktor
    ~PolarGen() {} //destruktur
```

```
    double kovetkezo(); //random lekérés
private:
    bool nincsTarolt;
    double tarolt; //random értéke
};

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

double PolarGen::kovetkezo() { //random lekérő függvény kifejtése
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do{
            u1 = std::rand () / (RAND_MAX + 1.0); //innentől jön az algoritmus
            u2 = std::rand () / (RAND_MAX + 1.0);
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);
        double r = std::sqrt ((-2 * std::log (w)) / w);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
        return r * v1; //idáig tart az algoritmus
    }

    else
    {
        nincsTarolt = !nincsTarolt; //ha van korábbi random érték, akkor azt ←
            adja vissza
        return tarolt;
    }
};
```

12.2. Homokózó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden másról működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen).

Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ahogy a feladat leírásban olvasható azt fogjuk megnézni, hogy néz ki az LZW BinFa Java nyelven. Majd ezek után megismerkedünk azzal is, hogyan lehet ezt Java Servlet-ként böngészőben futtatni. A kód a következő képpen néz ki:

```
class LZWBinFa
{
    public LZWBinFa ()
    {
        fa = gyoker;
    }
    public void hozzarendel(char b)
    {
        if (b == '0')
        {
            if (fa.nullasGyermek () == null)
            {
                Csomopont uj = new Csomopont ('0');
                fa.ujNullasGyermek (uj);
                fa = gyoker;
            }
            else
            {
                fa = fa.nullasGyermek ();
            }
        }
        else
        {
            if (fa.egyesGyermek () == null)
            {
                Csomopont uj = new Csomopont ('1');
                fa.ujEgyesGyermek (uj);
                fa = gyoker;
            }
            else
            {
                fa = fa.egyesGyermek ();
            }
        }
    }
    public void kiir ()
    {
        melyseg = 0;
        kiir (gyoker, new java.io.BufferedWriter(new java.io.←
            OutputStreamWriter(System.out)));
    }
    public int getMelyseg ()
    {
        melyseg = maxMelyseg = 0;
        rmelyseg (gyoker);
        return maxMelyseg - 1;
    }
}
```

```
    }
    public double getAtlag ()
    {
        melyseg = atlagosszeg = atlagdb = 0;
        ratlag (gyoker);
        atlag = ((double) atlagosszeg) / atlagdb;
        return atlag;
    }
    public double getSzoras ()
    {
        atlag = getAtlag ();
        szorasosszeg = 0.0;
        melyseg = atlagdb = 0;
        rszoras (gyoker);
        if (atlagdb - 1 > 0)
            szoras = java.lang.Math.sqrt (szorasosszeg / (atlagdb - 1));
        else
            szoras = java.lang.Math.sqrt (szorasosszeg);
        return szoras;
    }
    public void kiir (java.io.BufferedWriter os)
    {
        melyseg = 0;
        kiir (gyoker, os);
    }
    private class Csomopont
    {
        public Csomopont (char b)
        {
            betu = b;
            balNulla = null;
            jobbEgy = null;
        }
        public final Csomopont nullasGyermek ()
        {
            return balNulla;
        }
        public final Csomopont egyesGyermek ()
        {
            return jobbEgy;
        }
        public void ujNullasGyermek (Csomopont gy)
        {
            balNulla = gy;
        }
        public void ujEgyesGyermek (Csomopont gy)
        {
            jobbEgy = gy;
        }
        public final char getBetu ()
```

```
{  
    return betu;  
}  
private char betu;  
private Csomopont balNulla;  
private Csomopont jobbEgy;  
}  
private Csomopont fa;  
private int melyseg, atlagosszeg, atlagdb;  
private double szorasosszeg;  
public void kiir (Csmopont elem, java.io.BufferedWriter os)  
{  
    if (elem != null)  
    {  
        try{  
            ++melyseg;  
            kiir (elem.egyesGyermek (), os);  
            for (int i = 0; i < melyseg; ++i)  
                os.write("---");  
            os.write(elem.getBetu () + "(" + (melyseg - 1) + ")\\n");  
            kiir (elem.nullasGyermek (), os);  
            --melyseg;  
        }  
        catch(java.io.IOException e){  
            System.out.println("Csmopont írása nem sikerült.");  
        }  
    }  
    protected Csmopont gyoker = new Csmopont ('/');  
    protected int maxMelyseg;  
    protected double atlag, szoras;  
    protected void rmelyseg (Csmopont elem)  
    {  
        if (elem != null)  
        {  
            ++melyseg;  
            if (melyseg > maxMelyseg)  
                maxMelyseg = melyseg;  
            rmelyseg (elem.egyesGyermek ());  
            rmelyseg (elem.nullasGyermek ());  
            --melyseg;  
        }  
    }  
    protected void ratlag(Csmopont elem) {  
        if (elem != null) {  
            ++melyseg;  
            ratlag(elem.egyesGyermek());  
            ratlag(elem.nullasGyermek());  
            --melyseg;  
            if (elem.egyesGyermek() == null && elem.nullasGyermek() == null ←
```

```
        )
    }
}
protected void rszoras(Csomopont elem) {
    if (elem != null) {
        ++melyseg;
        rszoras(elem.egyesGyermek());
        rszoras(elem.nullasGyermek());
        --melyseg;
        if (elem.egyesGyermek() == null && elem.nullasGyermek() == null ←
            ) {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}
```

Ahogy látható, semmilyen címképző operátorra nincs szükség a Java kódban. Az megfigyelhető, hogy az osztály konstruktora az nagyon hasonlít az eredetire. Viszont a destruktorra és a szabadít-ra nincs szükségünk. Ez azért van, mert a Java nyelv automatikusan törli azokat az objektumokat a memóriából, amire már nincs érvényes referencia. Lehet úgy példányosítani egy osztályt, mint ahogy C++-ban a sima változókat deklaráltuk

```
>
    Csomopont fa;
```

Ebben az esetben egy `null` referenciát kapunk, vagyis ennek még át kell adni egy bizonyos memóriacímen található objektumot. Jelen esetben ez nem okoz problémát a programban, mert a `a fa` objektumreferenciának átadjuk a `gyoker` által mutatott objektumot. az átírás során én nem inicializáltam a `gyoker` refenciát, így hibát dobott a program, mivel utána a `fa` referencia is `null` referencia maradt, mely által mutatott objektumra hivatkozunk a későbbi függvényekbe. C++-hoz hasonlóan a `new` operátorral tudunk tárhelyet foglalni, mely Java-ban referenciát ad át.

A Java nyelv nem támogatja a operátor túlterhelést, új metódus jelent meg a programban, a `huzzarendel`. Mostmár ezzel lehet a fába beletteni a nullákat és az egyeseket.

És most következik a `main` függvény.

```
public static void main (String[] args)
{
    if (args.length != 3)
    {
        usage ();
        System.exit (-1);
    }

    String inFile = args[0];
```

```
if (args[1].charAt(1) != 'o')
{
    System.out.println("Missing -o argument");
    usage();
    System.exit(-2);
}

try {
    java.io.FileInputStream beFile = new java.io.FileInputStream(inFile ←
        );
    java.io.DataInputStream beFile_datastream = new java.io. ←
        DataInputStream(beFile);
    java.io.BufferedReader beFile_bufferedreader = new java.io. ←
        BufferedReader(new java.io.InputStreamReader(beFile_datastream)) ←
        ;

    java.io.FileWriter kiFile = new java.io.FileWriter(args[2]);
    java.io.BufferedWriter kiFile_bufferedwriter = new java.io. ←
        BufferedWriter(kiFile);
    LZWBInFa binFa = new LZWBInFa(); // s nyomjuk majd be az LZW fa ←
        objektumunkba

    int c;
    while ((c = beFile_bufferedreader.read()) != -1) {
        if (c == 0x0a) {
            break;
        }
        boolean kommentben = false;

        while ((c = beFile_bufferedreader.read()) != -1)
        {

            if (c == 0x3e) {          // > karakter
                kommentben = true;
                continue;
            }

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

            if (kommentben) {
                continue;
            }

            if (c == 0x4e) // N betű
            {
                continue;
            }
        }
    }
}
```

```
}

for (int i = 0; i < 8; ++i) {
    if ((c & 0x80) == 128) // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW fa objektumunkba
    {
        binFa.hozzarendel('1');
    } else // különben meg a '0' betűt:
    {
        binFa.hozzarendel('0');
    }
    c <<= 1;
}

binFa.kiir(kiFile bufferedwriter);
kiFile bufferedwriter.write("depth = " + binFa.getMelyseg() + '\n');
kiFile bufferedwriter.write("mean = " + binFa.getAtlag() + '\n');
kiFile bufferedwriter.write("var = " + binFa.getSzoras() + '\n');

kiFile bufferedwriter.close();
kiFile.close ();
beFile_datastream.close ();
beFile_bufferedreader.close ();
beFile.close ();
} catch (Exception e) {
    for (StackTraceElement elem : e.getStackTrace()) {
        System.err.println(elem);
    }
}
}

}

A függvény ugyanazt csinálja, mint korábban. Azonban már nem 4 paraméternek kell lennie, hanem csak 3-nak. Ez abból következik, hogy Java-ban az első paraméter nem a program neve, hanem az általunk megadott első paraméter a program neve után. Tehát leellenőrzük, hogy meg van-e minden szükséges paraméter. Ezután jön a try-catch blokk. Java-ban a kivételkezelés sok esetben elvárásnak számít. Jelen esetben a fájlkezelés miatt volt rá szükség, mivel enélkül le se fordulna a kód. De ezzel az esetleges null referenciaikra való hivatkozást is ki tudjuk szűrni. Először megnyitjuk a bemenethez és a kimenethez szükséges fájlokat. Ehhez a java.io osztályait használjuk. Ezután pedig elkezdjük beolvasni a forrást, és az abban lévő betűk függvényében 0-kal és 1-kel feltölteni a fát. A for ciklust kellett egy kicsit átalakítani, mivel Java-ban a bájtsorrend big-endian, ezért nem 1-el kell egyenlőnek lennie a (c & 0x80) kifejezésnek, hanem 128-al, ahhoz, hogy 1-est nyomjunk a fába. Majd a fát és annak mélységét, átlagát, szórását kiírjuk a kimenetként kapott fájba. Azt érdemes tudni, hogy a BufferedWriter osztály nem azonnal írja ki a fájlba a tartalmat, hanem csak akkor, amikor bezárjuk az általa megnyitott fájlt. Tehát ha ezt elfelejtjük megtenni, akkor ne lepődjünk meg a program eredménye láttán.
```

A továbbiakban pedig átalakítjuk a programot Java Servletté. Ebben segítségünkre lesz az Apache Tomcat

webszerver. Első lépésként letöltsük a legújabb verzióját. (<http://tomcat.apache.org/>) Ki kell csomagolni, majd /bin/startup.sh-val tudjuk elindítani a szervert, leállítani pedig a /bin/shutdown.sh-val lehet. A módosított java forrásból készített class fájlt a /webapps/ROOT/WEB-INF/classes mappába kell helyezni, majd a /webapps/ROOT/WEB-INF/web.xml fájlba el kell helyezni a következő sorokat:

```
<servlet>
    <servlet-name>LZWBinFa</servlet-name>
    <servlet-class>LZWBinFa</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>LZWBinFa</servlet-name>
    <url-pattern>/LZWBinFa</url-pattern>
</servlet-mapping>
```

Fontos, hogy ezeket a webapp tagak közé kell helyezni. Ha ezekkel megvagyunk, akkor már lehet futtatni a böngészőben a programot. Nézzük meg, hogy milyen módosításokat kellett végezni az eredeti forráson.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LZWBinFa extends HttpServlet
{
    ...
}
```

Az első lényeges dolog, hogy importálni kell a Java Servlethez szükséges csomagokat. Ezeket a javac nem találja meg azonnal, mivel a servlet-api.jar-t nem tartalmazza a Java JDK. Ezt a Tomcat webserverrel együtt kapjuk meg, ezért a classpath-t meg kell adni a javac számára. Ehhez a -cp kapcsolót használhatjuk.

```
javac -cp /path/to/apache-tomcat-9.0.xx/lib/servlet-api.jar ↪
      LZWBinFa.java
```

A másik, amit egyből észre lehet venni, hogy az osztályunk a HttpServlet osztály leszármazottja. Enélkül nem tudnánk Servlet-ként használni a programot. Az osztályunk konstruktora már nincs szükség, mivel nincs main függvény sem. A megmaradt tagfüggvények nagy része nem változott. Kivétel a kiir függvény, mely most a az LZWBinFa tagváltozójába, a kimenet-be építi fel a fát.

```
public void kiir (Csomopont elem)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ↪
    // leállítása
    if (elem != null)
    {

        ++melyseg;
        kiir (elem.egyesGyermekek ());
        // ez a postorder bejárás hoz képest
```

```
// 1-el nagyobb mélység, ezért -1
for (int i = 0; i < melyseg; ++i)
    kimenet.append("---");
kimenet.append(elem.getBetu () + "(" + (melyseg - 1) + ")<br>\n" +
");
kiir (elem.nullasGyermekek ());
--melyseg;

}
}
```

A main függvényt a CreateStringBuffer váltotta fel. Lényegében ugyan azt a szerepet látja el, annyi különbséggel, hogy ez már egy tagfüggvénye az osztályunknak.

```
public void CreateStringBuffer(String bemenet)
{
    for(int i = 0; i < bemenet.length(); ++i) {
        int c = bemenet.charAt(i);
        if(c == 0x0a) {
            break;
        }
    }
    boolean kommentben = false;

    for(int i = 0; i < bemenet.length(); ++i)
    {
        int c = bemenet.charAt(i);

        if (c == 0x3e) {          // > karakter
            kommentben = true;
            continue;
        }

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

        if (kommentben) {
            continue;
        }

        if (c == 0x4e) // N betű
        {
            continue;
        }

        for (int j = 0; j < 8; ++j) {
```

```
        if ((c & 0x80) == 128)
        {
            hozzarendel('1');
        } else
        {
            hozzarendel('0');
        }
        c <<= 1;
    }

}

kiir();
kimenet.append("depth = " + getMelyseg() + "<br>\n");
kimenet.append("mean = " + getAtlag() + "<br>\n");
kimenet.append("var = " + getSzoras() + "<br>\n");

}
```

Már nincs szükség a fájlkezeléshez szükséges objektumokra, mivel a kiírás a bemenet bufferbe történik. Az utolsó, és egyben legfontosabb függvény pedig a doGet függvény, mely a HttpServlet osztály egy tagfüggvénye. Ez szolgál arra, hogy GET kérést intézzünk a szerver felé.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    if(request.getParameter("bemenet") == null){
        return;
    }
    else
        CreateStringBuffer(request.getParameter("bemenet"));

    PrintWriter out = response.getWriter();
    String title = "LZWBinFa";
    String docType =
        "<!doctype html public \"-//w3c//dtd html 4.0 \" + \"transitional// ←\n        en\">\n";

    out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body>\n" +
        "    <h1 align = \"center\">" + title + "</h1>\n" +
        kimenet +
        "</body>" +
        "</html>"
```

```
) ;  
}
```

A `setContentType` függvény segítségével be tudjuk állítani, hogy milyen formátumban küldje a szerver a kért reprezentációt. Jelen esetben mi ezt egy HTML formátumban kérjük. A hibás működés elkerülése érdekében kezelni kell azt az esetet, ha a felhasználó elfelejt megadni a bemenetet. Ha van bemenetünk, akkor a `CreateStringBuffer` segítségével felépítjük, és kiírjuk a fát a bufferbe. A `PrintWriter` osztály segítségével megírjuk, hogy hogyan nézzen ki a válasz, mit tartalmazzon. Lényegében egy klasszikus html fájlt hozunk létre. Ha a fenti parancssal fordítjuk a forrást, akkor ideális esetben kapunk két `.class` fájlt. Mind a kettőt be kell másolni a fentebb megadott helyre, és módosítani kell az xml-fájlt. Mielőtt bemásoljuk a class-okat, ellenőrizni kell, hogy le van-e állítva a Tomcat server. Ha nincs, le kell állítani, majd a módosítások után be kell írni a következőt:

```
http://localhost:8080/LZWBinFa?bemenet=szöveg
```

12.3. Gagyi

Az ismert formális „`while (x <= t && x >= t && t != x);`” tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referencia) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK `Integer.java` forrására3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Java `Integer` osztálya a 128-nál kisebb egészeket pool-ozza. Ez abban nyilvánul meg, hogy ha példányosítunk egy `Integer` objektumot ami 128-nál kisebb értéket kap, akkor az objektum referencia ugyanarra a memóriaterületre hivatkozik. Mert úgynevezett pooling megy a háttérben, hogy a 128-nál kisebb értékű `Integer` objektumokat csak egyetlen példányban hozza létre, viszont a 128-nál nagyobbak esetén minden új példányt hozunk létre.

Érdekesség képpen nézzük meg az alábbi kódot:

```
Integer i1 = new Integer(127);  
Integer i2 = new Integer(127);  
System.out.println(i1 == i2); // true  
  
Integer i1 = new Integer(128);  
Integer i2 = new Integer(128);  
System.out.println(i1 == i2); // false
```

Az történik, hogy a `==` összehasonlítjuk, hogy a változók ugyanazt az objektumot mutatják-e. Az eredmény eltérhet attól függően, hogy milyen JVM-et használunk. A megoldás ennek a kiküszöbölésére az, hogy az `Integer` objektumok összehasonlításakor minden az `Integer.equals()` metódust használjuk.

```
System.out.println(i1.equals(i2)); // true
```

Valójában, amikor ezt a funkciót először vezették be a Java 5-be, a tartományt -127 és +127 között rögzítették. Később a Java 6-ban a tartomány csúcspontját a `java.lang.Integer.IntegerCache.high` jelölték. IntegerCache széleskörűen dokumentálva van a JDK-ban.

```
    /**
     * Cache to support the object identity semantics of autoboxing for ←
     * values between
     * -128 and 127 (inclusive) as required by JLS.
     *
     * The cache is initialized on first usage. The size of the cache
     * may be controlled by the {@code -XX:AutoBoxCacheMax=} option.
     * During VM initialization, java.lang.Integer.IntegerCache.high ←
     * property
     * may be set and saved in the private system properties in the
     * sun.misc.VM class.
     */

```

```
private static class IntegerCache {
    static final int low = -128;
    static final int high;
    static final Integer cache[];

    static {
        // high value may be configured by property
        int h = 127;
        String integerCacheHighPropValue =
            sun.misc.VM.getSavedProperty("java.lang.Integer. ←
                IntegerCache.high");
        if (integerCacheHighPropValue != null) {
            try {
                int i = parseInt(integerCacheHighPropValue);
                i = Math.max(i, 127);
                // Maximum array size is Integer.MAX_VALUE
                h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
            } catch( NumberFormatException nfe) {
                // If the property cannot be parsed into an int, ignore ←
                    it.
            }
        }
        high = h;

        cache = new Integer[(high - low) + 1];
        int j = low;
        for(int k = 0; k < cache.length; k++)
            cache[k] = new Integer(j++);

        // range [-128, 127] must be interned (JLS7 5.1.7)
        assert IntegerCache.high >= 127;
    }
}
```

```
private IntegerCache() { }
```

Ha az érték String, akkor a parseInt segítségével konvertáljuk egyszerű int értékké. Ha a bemenet a tartományban van -128 és 127, akkor az Integer objektumot minden visszaadjuk egy belső IntegerCache-nek. Az Integer osztály belső statikus IntegerCache osztályt tart fent, amely gyorsítótárként szolgál és az egész Objektumok -128 és 127 között tartja, ezért tapasztalható az, hogy ha megpróbálunk egész szám objektumot kapni a 127-re akkor minden ugyanazt az objektumot kapjuk. Jogosan adódik a kérdés, hogy annak idején miért választották a -127 és 127 közötti számtartományt. A válasz roppant egyszerű: Ez az egész számok legszélesebb körű tartománya.

Ezek után nézzük a forrást:

```
class Gagyi {
    public static void main (String[] args) {
        if (args.length != 2) {
            System.out.println("Használat: java Gagyi szám");
            return;
        }

        Integer x = Integer.parseInt(args[0]);
        Integer t = Integer.parseInt(args[0]);

        while (x >= t && x <= t && x != t) {
            System.out.println("Gagyiii");
        }
    }
}
```

A program 1 egész számot kér paraméterként, majd ezek értékét átadjuk x és t objektumreferenciáknak. Ezután pedig végrehajtjuk a feladatban megadott ciklust. Első ránézésre azt hisszük, hogy azt vizsgáljuk, hogy teljesül-e az, hogy x és t egyenlő, emellett pedig x és t nem egyenlő. Viszont Java-ban létezik az automatikus csomagolás fogalma. Ennek annyi a lényege, hogy a program automatikusan kicsomagolja az Integer osztály értékét, abban az esetben, ha ez szükséges. Például a <=, >= operátorok esetén megtörténik a kicsomagolás. Ellenben a != operátor objektum referenciákat hasonlít össze. Ha a két Integer objektum értéke ugyanaz, és 128-nál kisebb az értéke, akkor nem lépünk be a végtelen ciklusba. Ellenben ha 127-nél nagyobb egész adunk meg, akkor viszont belépünk, amiatt, mert ebben az esetben az objektum referenciák különbözőek lesznek.

12.4. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-leáll, ha nem követjük a Yoda conditions-t!

https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Azért találták ki ezt, hogy megkönnyítsék a programozók számára az elírások kiszűrését, abban az esetben, amikor egyenlőséget vizsgálnak. Főleg akkor érdemes használni, amikor a változókat nem változóval hasonlítjuk össze, hanem valamilyen konstanssal.

```
Integer a = 5;  
//Yoda használata nélkül  
if (a == 6)  
    //valamit csinál  
  
//Yoda használatával  
if (6 == a)  
    //valamit csinál
```

Ahogy látható a Yoda conditions lényege, hogy felcseréljük a feltételben a változót a konstanssal. Ez a technika megkönnyíti a hibakeresését például ebben az esetben:

```
if (6 = a)           if (a = 6)
```

Ha így írjuk a feltételt, akkor is működni fog a program, de az `if`-be minden be fog lépni a vezérlés, mivel annak fejlécében csak értéket adunk át, és nem feltételt vizsgálunk. Ezzel az a probléma, hogy a program nem megfelelően működik, viszont erről nem kapunk visszajelzést. Ha jobban szemügyre vesszük, akkor bizonyítást nyer az, hogy az első esetben sokkal egyszerűbb észrevenni, hogy valami probléma van mint a másodiknál.

Most, hogy tudjuk, mi is az a Yoda conditions, lássuk a feladat megoldását.

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

Jelen állapotában a program kivételt dob, pontosan a `java.lang.NullPointerException`-et. Ez annak köszönhető, hogy az `a` objektum refencia `null` értékű, és ennek a tagfüggvényét hívjuk meg. Mivel nem létező objektumra mutat hivatkozik, ezért kapunk hibát. Viszont a kikommentezett feltételt használva már nem kapunk hibát. Ez annak köszönhető, hogy a beírt szöveg egyből `String`-é konvertálódik, és el tudjuk érni annak tagfüggvényeit.

Egy szó is száz a lényeg az, hogy igen kis hasznos program ez és kijelenthetjük, hogy bizonyos esetben igazán hasznos lehet.

12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbpalg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását!

Ha megakadsz, de csak végső esetben: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitokjavat/apbs02.html> (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A BBP, teljes nevén Bailey-Borwein-Plouffe algoritmus a Pi hexadecimális jegyeinek egy $d > 0$ számjegytől történő kiszámolására szolgál. Az alapját a BBP formula jelenti, amit 1995-ben találtak ki.

Maga az algoritmus a következő:

$$\{16^d\pi\} = \{4\{16^dS1\} - 2\{16^dS4\} - \{16^dS5\} - \{16^dS6\}\}$$

A feladat ennek az algoritmusnak az implementálása.

```
class PiBBP {
    // {16^dπ} = {4{16^dS1}} - {2{16^dS4}} - {16^dS5} - {16^dS6}
    public double result;
    public String HexaJegyek;
    public PiBBP(int d) {
        double d16S1 = dj16s(d, 1);
        double d16S2 = dj16s(d, 4);
        double d16S3 = dj16s(d, 5);
        double d16S4 = dj16s(d, 6);

        double temp = 4.0d*d16S1 - 2.0d*d16S2 - d16S3 - d16S4;

        result = temp - java.lang.StrictMath.floor(temp);

        StringBuffer sb = new StringBuffer();

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

        while(result != 0.0d) {

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

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

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

        HexaJegyek = sb.toString();
    }
}
```

```
public double dj16s(int d, int j) {
    double sum = 0;
    for(int k = 0; k <= d; ++k) {
        sum += binmodk(8*k+j, d-k) / (double)(8*k + j);
    }
    sum -= java.lang.StrictMath.floor(sum);

//    long iPart = (long) sum;
//    Double fpart = sum -iPart;

    System.out.println(sum);
    return sum;
}

public double binmodk(int k, int d) {
    int t = 1;
    double r = 1.0;
    while(t <= d) {
        t *= 2;
    }

    while(t < 1) {
        if(d >= t) {
            r = (16*r) % k;
            d = d-t;
        }
        t = t/2;
        if (t >= 1) {
            r = (r*r) % k;
        }
    }
//    System.out.println(r);
    return r;
}

public static void main(String[] args) {
    PiBBP m_pibbp = new PiBBP(100000000);
    System.out.println(m_pibbp.HexaJegyek);
}
```

12.6. EPAM: Java Object metódusok

12.7. EPAM: Eljárásorientál vs Objektumorientált

12.8. EPAM: Objektum példányosítás programozási mintákkal

DRAFT

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megséríti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés.

[https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf\(93-99 fólia\)](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf(93-99 fólia)) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/BatfaiBarki/madarak/)

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Liskov elv azt mondja ki, hogy minden osztálynak helyettesíthetőnek kell lennie a leszármazottaival anélkül, hogy a program helyes működése megváltozná.

A következő példa egy matematikai problémát tartalmaz, ami jól szemlélteti az előzőekben megbeszélt szabály hiányát.

```
#include <iostream>

using namespace std;

class Negyszog{
public:
    Negyszog(int a_oldal, int b_oldal, int c_oldal, int d_oldal) {
        a = a_oldal;
        b = b_oldal;
        c = c_oldal;
        d = d_oldal;
    }
    int a;
    int b;
    int c;
    int d;
    int kerulet;
    int terulet;
};
```

```
class Negyzet:public Negyszog{
public:
    Negyzet(int a_oldal, int b_oldal, int c_oldal, int d_oldal):Negyszog( ←
        a_oldal, b_oldal, c_oldal, d_oldal){
    }
};

class Teglalap:public Negyzet{
public:
    Teglalap(int a_oldal, int b_oldal, int c_oldal, int d_oldal):Negyzet( ←
        a_oldal, b_oldal, c_oldal, d_oldal){
    }
};

void setKeruletTerulet(Negyzet &negyzet) {
    negyzet.kerulet = negyzet.a * 4;
    negyzet.terulet = negyzet.a * negyzet.a;
}

int main()
{
    Negyzet& negyzet = *new Negyzet(5,5,5,5);
    Teglalap& teglalap = *new Teglalap(4,5,4,5);

    setKeruletTerulet(negyzet);
    setKeruletTerulet(teglalap);

    cout << "Téglalap kerülete: " << teglalap.kerulet << ", Négyzet   ←
        kerülete: " << negyzet.kerulet << endl;
}
```

Először van egy Negyszog osztályunk. Ebből származtatjuk a Negyzet osztályt. Viszont a hiba az, hogy minden négyzet téglalap, de nem minden téglalap négyzet. A Negyzet osztályból akarjuk a Teglalap osztályt származtatni. Ezek után létrehozunk egy függvényt, mely egy négyzet kerületét és területét számolja ki. Viszont az objektum orientált nyelvben az ōs helyén használható a gyermek is. Vagyis a setKeruletTerulet függvényt meg tudjuk hívni egy Teglalap objektummal is. Ami nyilvánvalóan hibás eredményhez vezet, ha olyan táglalapot adunk át, ami nem négyzet.

Most következik a Java-s verzió.

```
class LiskovSert{
    public static class Négyszög{
        public int a;
        public int b;
        public int c;
        public int d;

        public Négyszög(int a_oldal, int b_oldal, int c_oldal, int d_oldal) ←
    }
```

```
a = a_oldal;
b = b_oldal;
c = c_oldal;
d = d_oldal;
}

public int kerület;
public int terület;
}

public static class Négyzet extends Négyszög{
    public Négyzet(int a_oldal, int b_oldal, int c_oldal, int d_oldal){
        super(a_oldal, b_oldal, c_oldal, d_oldal);
    }
}

public static class Téglalap extends Négyzet{
    public Téglalap(int a_oldal, int b_oldal, int c_oldal, int d_oldal) ←
    {
        super(a_oldal, b_oldal, c_oldal, d_oldal);
    }
}

public static class Program{
    public void setKerületTerület(Négyzet negyzet){
        negyzet.kerület = negyzet.a * 4;
        negyzet.terület = negyzet.a * negyzet.a;
    }
}

public static void main(String[] args){
    Program program = new Program();
    Négyzet negyzet = new Négyzet(5,5,5,5);;
    Téglalap teglalap = new Téglalap(4,5,4,5);

    program.setKerületTerület(negyzet);
    program.setKerületTerület(teglalap);

    System.out.println("Téglalap kerülete: " + teglalap.kerület + ", ←
        Négyzet kerülete: "+ negyzet.kerület);

}
}
```

A különbség az, hogy a `setKerületTerület` függvény a `Program` osztálynak egy tagfüggvénye.

13.2. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek!

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf(98. fólia)

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat lényege az lenne, hogy bemutassuk, az ős referenciáin vagy pointerein keresztül csak az ős függvényeit tudjuk meghívni. A C++-os megoldás a következő.

```
#include <iostream>

using namespace std;

class Vehicles{
public:
//    void setNumberOfWheels() {
//        wheels = 0;
//    }
    int wheels = 0;
};

class Cars: public Vehicles{
public:
    void setNumberOfWheels(){
        wheels = 4;
    }
};

int main()
{
    Vehicles &vehicle = *new Cars();

    vehicle.setNumberOfWheels(); //Hiba, az ős nem tudja elérni a gyermek ←
                                //metódusait.

    cout << "The number of wheels: " << vehicle.wheels << endl;
}
```

Van egy Vehicles osztályunk, aminek a wheels változójában a kerekek számát tároljuk. A kerekek száma alapértelmezetten nulla. A Cars ennek az osztálynak a leszármazota. Tartalmaz egy függvényt, amely beállítja a kerekek számát 4-re. A main függvényben deklarálunk egy Vehicle osztály referenciát, és értékül egy Cars osztályreferenciát adunk. Ha a vehicle referencia keresztül meghívjuk a setNumberOfWheels függvényt, akkor hibát dob a program, nem találja a függvény definícióját.

Java nyelven a következőképpen néz ki:

```
public class Vehicles{
```

```
// public void setNumberOfWheels(){
//     wheels = 0;
// }
public int wheels;

public static class Cars extends Vehicles{
    public void setNumberOfWheels() {
        wheels = 4;
    }
}

public static void main(String[] args)
{
    Vehicles vehicle = new Cars();

    vehicle.setNumberOfWheels(); //Hiba, mert az ős a leszármazott ←
                                metódusait nem éri el.

    System.out.println("The number of wheels: "+vehicle.wheels);
}
}
```

13.3. Anti OO

A BBP algoritmussal www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10^6 , 10^7 , 10^8 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket!

<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apas03.html#id561066>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A alábbi táblázatban láthatóak az eredmények:

A mérés alapján a Java került ki győztesnek, nem sokkal lemaradva a C#. A C/C++ meglepő módon jelentősen lassabban teljesített.

13.4. *deprecated - Hello, Android!*

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.5. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/-SMNISTforHumansExp3/app/src/main> Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.6. Hello, SMNIST for Humans!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.7. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf (77-79 fóliát)!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A ciklomatikus komplexitás, másnéven McCabe-komplexitás a program döntési komplexitását adja meg. A képlet a következő: $M = E - N + 2P$, ahol E a gráf éleinek a száma, N a csúcsok száma és P az összefüggő komponensek száma. Tehát minél több elágazás van a program vezérlésében, annál nagyobb értéket kapunk.

Az a program, aminek ki fogjuk számolni a ciklomatikus komplexitását, a BBP program lesz. Ehhez egy Maven projektet hozunk létre, és abba beleágyazzuk a programunkat. Ez abból áll, hogy a Netbeans-ben létrehozunk egy új Maven projectet, és ahhoz hozzáadjuk a PiBBP.java osztályt. Majd az mvn parancssal lehet felépíteni a .class fájlt. Fontos, hogy az mvn parancsot ott futassuk, ahol a pom.xml található.



Maven használata

Telepíteni a sudo apt install maven parancssal lehet.

Emellett még szükségünk lesz a SonarQuebe Community Edition programra, amely elemezni fogja a forrásunkat. Ha letöltöttük, akkor futtani kell a `sonar.sh start` parancsot a terminálba, ez egy szervert indít el, mely a <http://localhost:9000> linken érhető el. Jelentkezzünk be admin/admin-nal, majd hozzunk létre egy új projektet. A végén megkapjuk a parancsot, amivel a SonarQube leellenőrzi a kódot, és az eredmények megnézhetők a weboldalon.

13.8. EPAM: Interfész evolúció Java-ban

Mutasd be milyen változások történtek Java 7 és Java 8 között az interfészekben. Miért volt erre szükség, milyen problémát vezetett ez be?

Java 8 újdonságok:

-default és static interfész metódusok behozása

A default metódusok lehetővé teszik új metódusok hozzáadását létező interfészekhez anélkül, hogy megtörné azon interfészek régebbi verzióihoz készült kóddal való kompatibilitást. A default és absztrakt metódusok közötti különbség az, hogy az absztrakt metódusokat kötelező implementálni, de a default metódusokat nem.

```
package asd;

public interface DefaultExample {
    default String notRequired() {
        return "Default implementation";
    }
}

public class DefaultExampleImpl implements DefaultExample {

}

public class DefaultExampleImpl2 implements DefaultExample {
    @Override
    public String notRequired() {
        return "Overridden implementation";
    }
}
```

A DefaultExample interfész deklarál egy `notRequired()` default metódust a metódus definíciójában szereplő `default` kulcsszóval. Az egyik leszármazott osztály, a DefaultExampleImpl implementálja ezt az interfészt, de meghagyja a default metódus alapértelmezett implementációját. Egy másik, a DefaultExampleImpl2 felülírja a default implementációját a sajátjával.

Lambda bevezetése

A lambda kifejezések bevezetése a Java 8 egyik legjobban várt és legnagyobb újdonsága. Magát a fogalmat többféleképpen magyarázzák el különböző bemutatók és tananyagok. Egyesek szerint ez végre egy módszer arra, hogy funkcionális programozási stílust vezessünk be az objektum-orientált Java-ba. Más megközelítés szerint lehetővé teszi, hogy függvényeket használunk metódusparaméterként vagy kódot kezeljünk adatként. Egy harmadik elképzelés szerint - és ezt az utat járja az Oracle oktatási leírása is - a megismerés

legegyszerűbb első lépése, ha azt mondjuk, hogy anonim belső osztályokat lehet kiváltani lambda kifejezésekkel. (Emellett nem elhanyagolható módon hatékonyá és egyszerűbbé teszi a modern többmagos processzorok kihasználását is.)

Syntaxa

Paraméterlista	Nyíl token	Törzs
(int x, int y)	->	x+y

13.9. EPAM: Liskov féle helyettesíthetőség elve, öröklődés

13.10. EPAM: Interfész, Osztály, Absztrak Osztály

Mi a különbség Java-ban a Class, Abstract Class és az Interface között? Egy tetszőleges példával / példa kódon keresztül mutasd be őket és hogy mikor melyik koncepciót célszerű használni.

Interfész: -Minden metódus benne absztrakt. -Minden eleme public. Ha használni szeretnénk az interfészet akkor kreálnunk kell egyet majd a benne létrehozott absztract metódusokat definiálni kell a programban.

Absztrak Osztály: -Tartalmazhat konkrét metódusokat. -Lehetnek benne a tagok public, private, protected és default. Örököltetünk kell a használatához.

Osztály: -Jellemzni a cselekvéseit és az attribútumait egy objectumnak. -Absztract és konkrét metódusokat tárolhat. -Lehet public, private, protected és default.

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

        Car car = new Car();
        car.setFuel();
        car.run();
        Truck truck = new Truck();
        truck.setFuel();
        truck.run();
    }
}

interface Vehicle {
    public void setFuel();
    public void run();
}

class Car implements Vehicle {
    public void setFuel() {
        System.out.println("Car: Tank is full.");
    }
    public void run() {
        System.out.println("Car: Running.");
    }
}

abstract class MotorVehicle {
```

```
public void setFuel() {  
    System.out.println("MotorVehicle: Tank is full.");  
}  
abstract public void run();  
}  
class Truck extends MotorVehicle {  
    public void run() {  
        System.out.println("Truck: Running.");  
    }  
}
```

DRAFT

14. fejezet

Helló, Mandelbrot!

14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nlERIEOs

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_6.pdf (28-32 fólia)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az UML (Unified Modelling Language) egy szabványos általános célú modellező nyelv. Az UML nyelv egy kicsi részével az objektumorientált programozási nyelvekkel készített programok ábrázolásával fogunk megismерkedni.

Ahogy a feladat szövegében is látható sok fajta fejlesztői környezet található, viszont mi most az Umbrello-t fogjuk használni. Az Umbrello elérhető Windows-ra és Unix alapú rendszerekre is. Nekünk a legfontosabb, hogy lehetővé teszi azt, hogy forrásokból generálunk UML-t és ugyan ezt fordítva.

A feladat szövege alapján a z3a7.cpp forrásból kell UML-t generálni. Ehhez az Umbrello Kódimportálási varázslójára lesz szükségünk. Itt ki kell választani, hogy milyen kódot szeretnénk importálni. A C++-nál alapértelmezetten csak header állományokat keres, ezért ki kell egészíteni a keresést .cpp kiterjesztéssel. Majd ezek után beimportáljuk a fájlt. Majd oldalt a fastruktúra nézetből ki kell választani a class neveket és behúzni az osztálydiagramba. Itt még nem ér véget a megpróbáltatásunk mivel még pár dolgot módosítanunk kell.

Fontos dolog megemlíteni a kompozíciót és aggregációt. Mind a kettő az asszociációknak egy speciális fajtája. Az aggregáció tartalmazást jelöl. Jele az ábrán az üres rombusz. A kompozíció teli rombuszzal van jelölve, lényegében erős aggregációknak is nevezhető. Fontos megkötések a kompozíció esetén, hogy a tartalmazott objektum nem lehet megosztva több objektum között. Emellett a tartalmazó és a tartalmazott objektum együtt jön létre, ennek következtében a tartalmazott objektumért a tartalmazó felel. A tartalmazó objektum destrukturábnak kell tartalmaznia a tartalmazott objektum felszabadításához szükséges kódot. Kompozícióra példa a gyoker a mi feladatunkban.

14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ebben az esetben azt fogjuk csinálni, hogy az előző feladatban létrehozott diagramból fogunk kódot generálni. Ezt úgy tehetjük meg, hogy a Kódgenerálási varázslót használjuk. Valójában az történik, hogy a kódgenerátor legenerálja az osztálydeklarációt, a megadott és az asszociációkat megvalósító tagváltozókat, valamint a függvények fejlécét, de persze a törzset üresen hagyja, hiszen ezekre nem történik utalás a diagramban.

```
int getMelyseg ()
{
}

/**
 * @return double
 */
double getAtlag ()
{}
```

14.3. Egy esettan

A BME-s C++ tankönyv 14. fejezetét (427-444 elmélet, 445-469 az esettan) dolgozzuk fel!

Tanulságok, tapasztalatok, magyarázat...

UML-ben az osztályokat téglalapok ábrázolják, melyek 3 részből állnak. Az első részben szerepel az osztály neve, ezt követi az osztály attribútumai és végül tagváltozói. Mind a változók, mind a függvények előtt szerepel egy láthatóságot jelölő operátor. A "+" jelöli a publikus tagokat, a "-" a privát tagokat és a "#" a védetteket. Ezket követi az attribútum neve, típusa, multiplicitása és alapértelmezett értéke.

```
láthatóság név : típus multiplicitás = alapértelmezett ←
érték {tulajdonságok}
```

UML osztálydiagramok esetén a műveleteket a következő szintaxiszal adhatjuk meg:

```
láthatóság név paraméterlista: visszatérési_érték { ←
tulajdonságok}
```

```
például:
+lepesValidator(x:int, y:int):bool
```

A paraméterlistában meg lehet adni aparaméter irányát is, mely alapértelmezetten nulla. 3 féle lehet a paraméter iránya: in, out és inout. Az in az érték szerinti paraméterátadást reprezentálja, a out esetén a

művelet megváltoztatja a pointerként vagy referenciaként átadott objektumot. Végezetül az inout paramétert a függvény felhasználja és módosítja is. A parameterlista szintaxisa a következő:

írány név : típus = alapértelmezett _ érték

A műveleteket a query tulajdonsággal tudjuk konstanssá tenni, melyet az attribútumokhoz hasonlóan kapcsos zárójelek közé kell írni. Mind a változóra, mind a függvényekre igaz, hogy a statikusságot UML-ben aláhúzással jelöljük.

Most következik a csomópontokat összekötő irányított szakaszok, kapcsolatok.

Az asszociációt, mint két osztály kapcsolatát a diagramon egy vonallal jelöljük, melyenek közepén helyezkedik el az asszociáció neve, de ennek irány nem minden egyértelmű, ezért használunk nyilakat. Az asszociácóhoz tartozhatnak szerepnevek is, melyek azt jelölik, hogy miként vesznek részt az osztályok a kapcsolatban. Az asszociációknak lehetnek attribútumai, ezeket az asszociációs osztály reprezentálja. Ezek az osztályok szaggatott vonallal vannak összekötve az asszociációval. Az asszociációs osztályokat normál osztályként implementáljuk, mely tartalmaz egy bal és egy jobb osztályra mutató mutatót. És segíti a másik oldal elrését is az osztályoknak.

Az esettanulmány

Egy számítógép-alkatrészkel és számítógép konfigurációval foglalkozó kereskedésnek kell elkészíteni egy szoftvert. A szoftver feladata, hogy nyílvántartsa az alkatrészeket és a konfigurációkat. Támogatnia kell a termékek állományból való betöltését, képernyőn történő megjelenítését, az árak rugalmas kialakítást.

A koncepció 2 részből áll. Elsőnek ki kell alakítani egy keretrendszer osztálykönyvtár formájában, mely alapszinten támogatja a termékek kezelését. Erre kell ráépíteni a számítógép-alkatrészeket kezelő rendszert. Cél, hogy a keretrendszer módosítása nélkül lehessen a jövőben új termékcsaládot támmogató szoftvereket készíteni.

Először a keretrendszer kialakításával foglalkozunk. Szükség van egy Product osztályra, mely a termékek általános kezelését implementálja. Ebből fognak leszármazni azok az osztályok amik minden termék jellemzői, mint pl a kezdő ár. ezek fognak belekerülni a Product osztályba.

Mivel a kereskedés számítógép-konfigurációkat is árul, ezért szükséges az összetett termékek kezelése is. Ezt a CompositeProduct osztály valósítja meg. Az egyes alkatrészeket vektorban tároljuk, melyhez az AddPart metódussal tudunk elemeket adni. Az adatfolyamból az összetett termékeket egy extra paraméterrel adjuk meg. Ez a paraméter tartalmazza, hogy hány elemből áll, így tudjuk, hogy az utána következő x darab hozzá tartozik.

Az egyes termékeket nyílván kell tartani. Ezért van szükség a ProductInventory osztályra. Feladata a betöltött termékek tárolása memóriában, azok adatfolyamba írása, és formázott megjelenítése. Ebben az osztályban is egy vektorban tároljuk a termékeket.

A probléma abból adódik, hogy a keretrendszer egy osztálykönyvtár, emiatt az általunk bevezetett termékosztályokat nem ismeri. Emiatt szükség van még egy osztályra ami nem más mint a ProductFactory osztály. Ennek a ReadAndCreateProduct függvényét a keretrendszerben kell definiálni, de ez által meghívott CreateProduct függvény már virtuális, tehát implementációját róbízzuk a leszármazott osztályokra.

Amikor elkszíjtük a keretrendszerre épülő alkalmazásunkat, a ProductFactory osztályt kell leszármaztatnunk, ez lesz a ComputerProductFactory. Erre azért van szükség, hogy az előbb említett virtuális függvényhez definíciót tudunk adni.

14.4. BPMN

Rajzolunk le egy tevékenységet BPMN-ben!

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_7.pdf (34-47 fólia)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A BPMN(Business Process Model and Notation) üzleti folyamatok ábrázolásához használt nyelv. Jelenlegi legújabb verziója a BPMN 2.0.

Az UML diagramon kívül van még más lehetőségünk is arra, hogy modellezzenk egy folyamatot. Bár ez nem konkrétan osztályokat vázol fel, hanem a folyamatát adja meg.

Maga a folyamat a könyv kölcsönzési folyamatot írja le, ami úgy történik, hogy kérést nyújtunk be erről a szándékunkról, és erre kapunk majd egy választ, hogy a könyv elérhető-e vagy sem. Ha elérhető, "kifizetjük", majd erről értesítést kap a pénztár is, végül a végállapotba érünk. A másik esetben, ha a könyv foglalt és erről értesítést kapunk, döntethetünk három opció közül: félrerakatjuk a könyvet magunknak, mikor elérhetővé válik, elvettük a könyvet, vagy 1 hét múlva ismét jelentkezünk érte. Az utóbbi kettő esetében a könyv kölcsönzési kérelmet visszadobtuk és így kerültünk a végállapotba. A könyv félrerakásakor választ küldünk ezzel kapcsolatban, majd két hét múlva értesítést kapunk a könyv aktuális állapotáról.

14.5. BPEL Helló, Világ! - egy visszhang folyamat

Egy visszhang folyamat megvalósítása az alábbi teljes „videó tutoriál” alapján: https://youtu.be/0OnIYWX2v_I

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.6. TeX UML

Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.7. EPAM: Neptun tantárgyfelvétel modellezése UML-ben

Tanulságok, tapasztalatok, magyarázat...

14.8. EPAM: Neptun tantárgyfelvétel UML diagram implementálása

Tanulságok, tapasztalatok, magyarázat...

14.9. EPAM: OO modellezés

Tanulságok, tapasztalatok, magyarázat...

DRAFT

15. fejezet

Helló, Chomsky!

15.1. Encoding

Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezes betűket!

<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Most azzal fogunk megismerkedni, hogy a Java nyelv milyen módon teszi lehetővé azt, hogy ékezes betűket használjunk a forrásban.

Java nyelv abban különleges, hogy a forráskódban támogatja az Unicode karakterkódolást, ellentétben a C/C++-al, ahol csak ASCII karaktereket használhatunk.

A megoldás a problémára a Javac -encoding kapcsolója. Ezután megadhatjuk a kívánt karakterkódolást, és lefordul a program.

```
javac -encoding utf8 MandelbrothalmazNagyító
```

15.2. OOCWC lexer

Izzítsük be az OOCWC-t és vázoljuk a <https://github.com/nbatfai/robocaremulator/blob/master/justine-rcemu/src/carlexer.ll> lexert és kapcsolását a programunk OO struktúrájába!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A lexer feladata az, hogy tokenizálja a bemenetet, és ezek alapján elemezzük azt. A Lex egy program, amely lexikális elemzőt generál. Beolvassa a bemenetet, és elkészíti a lexikális elemzőt, majd pedig legenerálja a C++ forrást.

Az alap struktúra a következő:

```
{definíciók}  
%%  
{szabályok}  
%%  
{felhasználói utasítások}
```

Ez alapján nézzük végig a carlexer.ll

```
%option c++  
%option noyywrap
```

A forrás elsőrésze azt adja meg, hogy c++ forrást szeretnénk generálni, és nincs szükségünk az yywrap() függvényre. Az yywrap() feladata az lenne, hogy amikor az yylex() végére ér a bemenetnek, akkor meghívja, és ha 1-et ad vissza, akkor nincs már több bemenet, 0 esetén pedig tovább olvassa a bemenetet amire az yyin mutat. Mivel a yywrap() függvény definícióját nem adja meg a Lex program a kimeneti C++ fájlban, ezt a programozónak kell elkészítenie.

```
% {  
#define YY_DECL int justine::robocar::CarLexer::yylex()  
#include "carlexer.hpp"  
#include <cstdio>  
#include <limits>  
}%
```

Az opciók megadása után include-áljuk a szükséges header állományokat, deklarálhatjuk a szükséges változókat, vagy macro-kat hozhatunk létre.

```
INIT "<init"  
INITG "<init guided"  
WS [ \t]*  
WORD [^-:\n \t()]{2,}  
INT [0123456789]+  
FLOAT [-.0123456789]+  
ROUTE "<route"  
CAR "<car"  
POS "<pos"  
GANGSTERS "<gangsters"  
STAT "<stat"  
DISP "<disp">"
```

A következő lépés az egyes lexer-hez szükséges változók megadása. Lényegében megadunk egy nevet és ehhez hozzákapcsolunk egy stringet, így utána a nevet tudjuk használni a szabályok meghatározása során. Tehát, ahogy látható, az INIT esetén a "<init" sztringet olvasunk be. A WS a whitespace-t jelenti, és azért van ott a csillag, mert bármennyi lehet belőle, akár 0 is. A WORD egy kicsit érdekesebb reguláris kifejezés. Ez olyan bemeneteket jelent, amik nem -,:,\n, \t,(,) karakterekből állnak, és legalább 2 elemből.

```
%%  
{DISP} {  
m_cmd = 0;
```

```
        }
{POS} {WS} {INT} {WS} {INT} {WS} {INT}   {
    std::sscanf(yytext, "<pos %d %u %u", &m_id, ←
                &from, &to);
    m_cmd = 10001;
}
{CAR} {WS} {INT}   {
    std::sscanf(yytext, "<car %d", &m_id);
    m_cmd = 1001;
}
{STAT} {WS} {INT}   {
    std::sscanf(yytext, "<stat %d", &m_id);
    m_cmd = 1003;
}
{GANGSTERS} {WS} {INT}      {
    std::sscanf(yytext, "<gangsters %d", &m_id) ←
    ;
    m_cmd = 1002;
}
{ROUTE} {WS} {INT} {WS} {INT} ({WS} {INT}) * {
    int size{0};
    int ss{0};
    int sn{0};

    std::sscanf(yytext, "<route %d %d%n", &size, & ←
                m_id, &sn);
    ss += sn;
    for(int i{0}; i<size; ++i)
    {
        unsigned int u{0u};
        std::sscanf(yytext+ss, "%u%n", &u, &sn);
        route.push_back(u);
        ss += sn;
    }
    m_cmd = 101;
}
{INIT} {WS} {WORD} {WS} ("c"|"g")  {
    std::sscanf(yytext, "<init %s %c>", name, &role ←
                );
    num = 1;
    m_cmd = 0;
}
{INIT} {WS} {WORD} {WS} {INT} {WS} ("c"|"g")  {
    std::sscanf(yytext, "<init %s %d %c>", name, & ←
                num, &role);
    if(num >200)
    {
        m_errnumber = 1;
        num = 200;
    }
}
```

```
        m_cmd = 1;
    }
{INITG} {WS} {WORD} {WS} ("c"|"g") {
    std::sscanf(yytext, "<init guided %s %c>", name ←
                  , &role);
    num = 1;
    m_guided = true;
    m_cmd = 3;
}
{INITG} {WS} {WORD} {WS} {INT} {WS} ("c"|"g") {
    std::sscanf(yytext, "<init guided %s %d %c>", ←
                  name, &num, &role);
    if(num >200)
    {
        m_errnumber = 1;
        num = 200;
    }
    m_guided = true;
    m_cmd = 2;
}
.
{;}
%%
```

A szabályok részben pedig megadjuk, hogy az egyes bemenetek szerint mi legyen az egyes változók értéke. Ehhez a `sscanf`-et használjuk, aminek az első paramétere a bemenet, majd annak a formátuma, és végül az egyes változók, amikbe "beleírjuk" az értékeket.

```
int yyFlexLexer::yylex() {return -1;}
```

Végül definiáljuk az `yylex()` függvényt. Ebből a `carlexer.ll` fájlból pedig a `lex` parancs generálja a forrást.

C++ forrás generálása

```
lex carlexer.ll
```

15.3. I334d1c45

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet> (Ha ez első részben nem tette meg, akkor írasd ki és magyarázd meg a használt struktúratömb memória foglalását!)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Mielőtt nekiugrunk a feladat megoldásának azelőtt érdemes megnézni a `1337d1c7.1` forrásban lévő `cipher` struktúrát.

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

Ha a `sizeof` függvénytel megnézzük a méretet, akkor azt kapjuk, hogy 40 bájt. A `leet` mérete viszont 32 bájt, mivel a `char*` 8 bájt, és a tömb 4-szer ennyi helyet foglal le. A `char` mérete pedig 1 bájt. Akkor mi okozza a problémát? A megoldás a **padding**. A padding lényege, hogy a fordító a címzési hibák elkerülése érdekében extra területet foglal. Az alábbi szabály alapján történik a plusz terület foglalás: ha egy tag előtt vagy mögött vagy a struktúra legvégén deklarálva van egy nagyobb tag, akkor mérete a nagyobb tagével lesz egyenlő. A padding megszüntetése is megoldható a `#pragma pack(1)` segítségével. Ezek után a várt eredményt fogjuk kapni.

Továbbiakban ugyan azt fogja csinálni a program, csak kap egy grafikus felületet a Qt segítségével, viszont nem fogjuk használni a Lex programot. Először a `Cipher` osztályt fogjuk megnézni.

```
#include <QVector>
#include <QRandomGenerator>

class Cipher
{
public:
    Cipher();
    QString get_result();
    void encryption(QString);

private:
    struct m_cipher{
        char c;
        char const *leet[4];
    };

    QVector<struct m_cipher> m_vector;
    QString m_result;
};
```

Korábban C forrásban tömböt használtunk az egyes karakterekhez tartozó konverziókhöz, itt most vektorral csináljuk `QVector`. A csere oka egyszerűen annyi, hogy egy fejlettebb adatstruktúrát használjunk, aminek van olyan függvénye, amely megadja az elemei számát. A random számok generálását pedig a `QRandomGenerator` osztályra bízzuk. A `get_result()` függvény feladata, hogy az `m_result` privát tag értékéhez hozzáférést biztosítson.

```
QString Cipher::get_result() {
    return m_result;
}
```

A szöveg kódolását a `encryption()` függvény végzi, melynek definíciója a következő:

```
void Cipher::encryption(QString input) {
```

```
m_result.clear();
for (int x = 0; x < input.length(); ++x) {
    int found = 0;
    for(int i=0; i<m_vector.length(); ++i)
    {

        if(m_vector[i].c == input[x].toLowerCase())
        {

            int r = static_cast<int>(QRandomGenerator::global()->generate()%4);

            m_result.append(m_vector[i].leet[r]);

            found = 1;
            break;
        }
    }

    if(!found)
        m_result.append(input[x]);
}
}
```

Hasonló mint a korábbi megoldás, csak kisebb dolgok lettek módosítva. Az `m_result`-ot minden híváskor ürítjük, hogy ne legyenek benne a korábbi fordításból származó eredmények. Majd végig megyünk a bemenetben karakterenként, és ha van olyan karakter, amit tudunk helyettesíteni, akkor random kiválasztunk egyet. Ha valamelyen oknál fogva nem oldható ez meg, akkor pedig az eredetit rakjuk be az `m_result` stringbe. A random számot a `QRandomGenerator::global()->generate()` függvénytel generáljuk. De mivel nekünk csak 4 elem van a helyettesíthető elemek között, ezért a random számot maradékosan osztjuk 4-gyel.

15.4. Full screen

Készítsünk egy teljes képernyős Java programot! Tipp: https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanitok-javat/ch03.html#labyrinth_jatek

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kékreállóbb irányítás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.6. Paszigráfia Rapszódia LuaTeX vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.7. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! Lásd <https://youtu.be/XpBnR31BRJY>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Perceptron annyit csinál, hogy kétfelé választja a bemenetet. Egy perceptron a következő részeiből áll: bemenetek, a hozzájuk tartozó súlyok, az összegző csomópont, az aktivációs függvény és a kimenet. Ezt implementálja a Perceptron osztályunk, ami az ml.hpp-ben található.

```
class Perceptron
{
public:
    Perceptron ( int nof, ... )
    {
        n_layers = nof;

        units = new double*[n_layers];
        n_units = new int[n_layers];

        va_list vap;

        va_start ( vap, nof );

        for ( int i {0}; i < n_layers; ++i )
        {
            n_units[i] = va_arg ( vap, int );

            if ( i )
                units[i] = new double [n_units[i]];
        }
    }
}
```

```
va_end ( vap );

weights = new double**[n_layers-1];

#ifndef RND_DEBUG
std::random_device init;
std::default_random_engine gen {init() };
#else
std::default_random_engine gen;
#endif

std::uniform_real_distribution<double> dist ( -1.0, 1.0 );

for ( int i {1}; i < n_layers; ++i )
{
    weights[i-1] = new double *[n_units[i]];

    for ( int j {0}; j < n_units[i]; ++j )
    {
        weights[i-1][j] = new double [n_units[i-1]];

        for ( int k {0}; k < n_units[i-1]; ++k )
        {
            weights[i-1][j][k] = dist ( gen );
        }
    }
}

Perceptron ( std::fstream & file )
{
    file >> n_layers;

    units = new double*[n_layers];
    n_units = new int[n_layers];

    for ( int i {0}; i < n_layers; ++i )
    {
        file >> n_units[i];

        if ( i )
            units[i] = new double [n_units[i]];
    }

    weights = new double**[n_layers-1];

    for ( int i {1}; i < n_layers; ++i )
    {
        weights[i-1] = new double *[n_units[i]];
    }
}
```

```
for ( int j {0}; j < n_units[i]; ++j )
{
    weights[i-1][j] = new double [n_units[i-1]];

    for ( int k {0}; k < n_units[i-1]; ++k )
    {
        file >> weights[i-1][j][k];
    }
}

double sigmoid ( double x )
{
    return 1.0/ ( 1.0 + exp ( -x ) );
}

double operator() ( double image [] )
{
    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
    {

#define CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );
#define else
#define pragma omp parallel for
        for ( int j = 0; j < n_units[i]; ++j )
        {
            units[i][j] = 0.0;

            for ( int k = 0; k < n_units[i-1]; ++k )
            {
                units[i][j] += weights[i-1][j][k] * units[i-1][k];
            }

            units[i][j] = sigmoid ( units[i][j] );
        }
#define endif
    }
}
```

```
}

return sigmoid ( units[n_layers - 1][0] );

}

void learning ( double image [], double q, double prev_q )
{
    double y[1] {q};

    learning ( image, y );
}

void learning ( double image [], double y[] )
{
    //(*this) ( image );

units[0] = image;

double ** backs = new double*[n_layers-1];

for ( int i {0}; i < n_layers-1; ++i )
{
    backs[i] = new double [n_units[i+1]];
}

int i {n_layers-1};

for ( int j {0}; j < n_units[i]; ++j )
{
    backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ( ←
        units[i][j] ) ) * ( y[j] - units[i][j] );

    for ( int k {0}; k < n_units[i-1]; ++k )
    {
        weights[i-1][j][k] += ( 0.2* backs[i-1][j] *units[i-1][←
            k] );
    }
}

for ( int i {n_layers-2}; i >0 ; --i )

#pragma omp parallel for
for ( int j =0; j < n_units[i]; ++j )
{

    double sum = 0.0;
```

```
        for ( int l = 0; l < n_units[i+1]; ++l )
        {
            sum += 0.19*weights[i][l][j]*backs[i][l];
        }

        backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ←
            ( units[i][j] ) ) * sum;

        for ( int k = 0; k < n_units[i-1]; ++k )
        {
            weights[i-1][j][k] += ( 0.19* backs[i-1][j] *units[ ←
                i-1][k] );
        }
    }

    for ( int i {0}; i < n_layers-1; ++i )
    {
        delete [] backs[i];
    }

    delete [] backs;
}

~Perceptron()
{
    for ( int i {1}; i < n_layers; ++i )
    {
        for ( int j {0}; j < n_units[i]; ++j )
        {
            delete [] weights[i-1][j];
        }

        delete [] weights[i-1];
    }

    delete [] weights;

    for ( int i {0}; i < n_layers; ++i )
    {
        if ( i )
            delete [] units[i];
    }

    delete [] units;
    delete [] n_units;
}
```

```
void save ( std::fstream & out )
{
    out << " "
    << n_layers;

    for ( int i {0}; i < n_layers; ++i )
        out << " " << n_units[i];

    for ( int i {1}; i < n_layers; ++i )
    {
        for ( int j {0}; j < n_units[i]; ++j )
        {
            for ( int k {0}; k < n_units[i-1]; ++k )
            {
                out << " "
                << weights[i-1][j][k];
            }
        }
    }
}

private:
    Perceptron ( const Perceptron & );
    Perceptron & operator= ( const Perceptron & );

    int n_layers;
    int* n_units;
    double **units;
    double ***weights;
};
```

A régi program módosítása után már nem csak egy számot fogunk kapni hanem egy képi kimenetünk is lesz. A módosítás a következő:

```
srand(time(nullptr));

for(int i = 0; i < png_image.get_width(); ++i)
    for(int j = 0; j<png_image.get_height(); ++j){
        png_image[i][j].green = rand()%256; //newPNG[j* ←
                                            png_image.get_height() + j];
    }
png_image.write("output.png");
```

Annyit csinálunk, hogy vesszük a beolvasott képet, és annak a zöld elemét változtatjuk random.

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

for(int i = 0; i < png_image.get_width(); ++i)
```

```
for(int j = 0; j<png_image.get_height(); ++j){  
    png_image[i][j].green = newPNG[j*png_image.  
        get_height() + j];  
}  
png_image.write("output.png");
```

Valójában van egy tömbünk csupa nullával, és ebből kiválasztjuk az adott elemet, amivel kinullázzuk az RGB valamelyikét. Valójában elég lenne az alábbi kódcsipet is a tömb helyett:

```
png_image[i][j].green = 0;
```

DRAFT

16. fejezet

Helló, Stroustrup!

16.1. JDK osztályok

Írunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Program használata



```
sudo apt-get install libboost-all-dev
sudo apt-get install qtdeclarative5-dev
qmake JavaClassCounter.pro
make
./JavaClassCounter /path/to/src
```

Vagy telepítsd a Qt Creator-t, és abba betöltve a *.pro fájlt, működni fog.

A Boost egy kifejezetten C++hoz használható eszközkönyvtár. Használata egyszerű mivel csak include-álni kell a megfelelő header file-okat és a fordító magának megtalája a szükséges függvényeket, osztályokat. A Boost.Filesystem és a Boost.Regex könyvtárakat fogjuk használni a feladat során. Filesystem azért lesz fontos számunkra, hogy a megadott mappában lévő fájlokat el tudjuk érni, Regex meg a kiterjesztés szűrésénél fog szerepet játszani.

A forrás az alábbiakat tartalmazza.

```
int iterator(boost::filesystem::path p) {
    int osztalyok_szama = 0;
    boost::regex expr{".*\\".java"};

    for (const boost::filesystem::directory_entry& x : ←
        boost::filesystem::recursive_directory_iterator(p)) {
```

```
        if(boost::regex_match(x.path().string(), expr) && ←
            boost::filesystem::is_regular_file(x.path())){
                std::cout << ++osztalyok_szama << std::endl;
                std::cout << "    " << x.path() << '\n';
            }
        }
        return osztalyok_szama;
    }
}
```

Az iterator függvény feladata, hogy kiszámolja a Java osztályok számát. Ehhez paraméterként kap egy boost::filesystem::path típusú objektumot, mely tartalmazza a elérési útját a src-nek. A osztalyok_szama változóban tároljuk el a megtalált osztályok számát. Az expr objektum egy reguláris kifejezést tartalmaz, azokat a fájlokat vizsgálja, amik .java-ra végződnek. Majd a for ciklus segítségével bejárjuk az src mappát. Ehhez a recursive_directory_iterator -t használjuk, melynek segítségével az almappákat is be tudjuk járni. Csak akkor növeljük a megtalált osztályok számát, ha illeszkedik a fájl neve a expr kifejezésünkre, és maga a fájl egy hagyományos fájl, nem pedig mappa. A ciklus végeztével visszaadjuk a talált osztályok számát.

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

    if (argc < 2)
    {
        std::cout << "Usage: tut3 path\n";
        return 1;
    }

    boost::filesystem::path p (argv[1]);

    try
    {
        int osztalyok_szama = iterator(p);
        std::cout << "Java JDK osztályok száma " << ←
                    osztalyok_szama << std::endl;
    }

    catch (const boost::filesystem::filesystem_error& ex)
    {
        std::cout << ex.what() << '\n';
    }

    return a.exec();
}
```

A main függvényben pedig p objektumban eltároljuk az átadott útvonalt. Majd meghívjuk a korábban tárgyalt iterator függvényt, és kiíratjuk a Java JDK osztályok számát terminálba.

16.2. Másoló-mozgató szemantika

Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékkadásra!

Megoldás video:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Érdemes említést tenni arról, hogy mikor is hívódik meg a konstruktor, és az értékkadás.

```
LZWBInFa binFa = binFa_regi; //konstruktor (másoló)
LZWBInFa binFa2;
binFa2 = binFa_regi; //értékkadás (másoló)
```

Szóval a lényeg az, hogy amikor az objektum létrehozásakor adunk értéket, akkor a mozgató/másoló konstruktor hívódik meg, viszont ha inicializált objektumnak adunk meg új értéket, akkor pedig a másoló/mozgató értékkadás.

A továbbiakban a konstruktort fogjuk vizsgálni. Mivel konstruktorról beszélünk, ezért a neve megegyezik az osztály nevével. A vezérlés akkor adódik át erre a konstruktorra, ha magával megegyező típusú referenciait adunk át.

```
LZWBInFa::LZWBInFa (const LZWBInFa & forras) {
    std::cout << "Copy ctor" << std::endl;
    gyoker = new Csomopont ('/');
    gyoker->ujEgyesGyermek(masol(forras.gyoker-> ←
        egyesGyermek(), forras.fa));
    gyoker->ujNullasGyermek(masol(forras.gyoker-> ←
        nullasGyermek(), forras.fa));
    if (forras.fa == forras.gyoker) {
        fa = gyoker;
    }
}
```

Látható, hogy ez a konstruktor egy LZWBInFa konstans referenciát vár paraméterként. Mivel ilyenkor nem hívódik meg az alap konstruktor, emiatt a gyoker pointerünknek értéket kell adni, különben memória-címzési hibát kaphatunk. Ha ezzel kész vagyunk, akkor létre kell hozni a gyökértől kiindulva az egyes gyermekeket, ezért van szükség a ujEgyesGyermek és ujNullasGyermek függvényekre. Mivel egy már elkészített fából indulunk ki, ezért annak a mintájára készítjük el a gyermekeket. Ebben a masol függvény van segítségünkre. Ennek két paramétere van, az egyik a forrás gyökerének a gyermekje, a másik pedig a forrás fa mutatója, mivel azt is szeretnénk átmásolni, hogy a fa mutató éppen hol áll az eredeti fában.

```
Csomopont* LZWBInFa::masol (Csmopont* elem, Csmopont* ←
    regi_fa) {
    Csmopont* ujelem = nullptr;
    if (elem != nullptr) {
```

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

A `masol` egy `Csomopont` mutatót ad vissza. Első lépésben létrehozzuk a később átadni kívánt csomópon-tot. Majd ellenőrizzük, hogy az eredeti fa csomópontjának van-e értéke, vagy null pointer. Utóbbi esetben szimplán visszaadunk egy null pointert. Ellenkező esetben a `ujelem` pointernek átadunk egy az eredeti csomópont alapján inicializált csomóponthoz tartotó memóriacímet. Ezután újra meghívjuk a `masol` függ-vényt annak érdekében, hogy elkészítsük az `ujelem` gyermekeit. Ezt addig folytatjuk, ameddig az eredeti fa végére nem érünk. Abban az esetben, ha a régi fához tartozó fa mutató a `masol` függvénynek átadott elemre mutat, akkor az új fánk fa mutatóját ezen elem alapján létrehozott csomópontra állítjuk. Végezet-tül pedig visszatérünk a csomóponttal. Ahhoz, hogy a fa mutatót a `gyoker`-re is állíthassuk, a másoló konstruktörben is ellenőrizzük, hogy hova mutat az eredeti fa fa mutatója.

Most, hogy megismerkedtünk a másoló konstruktörral, nézzük át röviden a másoló értékkadást. Ezt is a `masol` függvényre alapozzuk.

```
LZWBinFa & LZWBinFa::operator= (const LZWBinFa & forras) {  
    std::cout << "Copy assaignement" << std::endl;  
  
    gyoker->ujEgyesGyermek(masol(forras.gyoker-> ←  
        , forras.fa));  
    gyoker->ujNullasGyermek(masol(forras.gyoker-> ←  
        , forras.fa));  
  
    if (forras.fa == forras.gyoker) {  
        fa = gyoker;  
    }  
    return *this;  
}
```

Lényegében egy operátor túlterhelésről van szó. Szemantikailag különbség a konstruktőrhoz képest, hogy a másoló értékkadásnak van visszatérési értéke, jelen esetben `LZWBinFa` referencia. Ennek megfelelően visszaadunk egy mutatót arról az objektumról, amely az egyenlőség bal oldalán volt. Ezen kívül a már ismert eljárást követjük.

A C++11 másoló szemantikájának megismerése után folytassuk a mozgató szemantikával, azon belül is a mozgató konstruktörral. A mozgató konstruktur paraméteréül egy jobbértékreferenciát vár. Ezt jelöli a dupla `&`. A feladat szerint a mozgató értékkadásra kell alapoznunk.

```
LZWBinFa::LZWBinFa (LZWBinFa&& forras)
{
    std::cout<<"Move ctor\n";
    gyoker = nullptr;
    *this = std::move(forras); //ezzel kényszerítjük ki, ←
        hogy a mozgató értékkadást használja

}
```

Az alapkoncepciója az a mozgató értékkadásra alapozásnak, hogy lényegében felcseréljük a két fa gyoker mutatójának értékét. Ennek érdekében az újonan létrehozni kívánt fa gyökerét null mutatóvá tesszük, majd meghívjuk a mozgató értékkadást. Ehhez a `std::move` függvényre van szükségünk, amely bemenetéül kapott paramétert jobbértekreferenciává alakítja. Mivel a `this` egy már inicializált objektumot jelöl, ezért nem a mozgató konstruktur, hanem a mozgató értékkadás hívódik meg.

```
LZWBinFa& LZWBinFa::operator= (LZWBinFa&& forras)
{
    std::cout<<"Move assignment ctor\n";
    std::swap(gyoker, forras.gyoker);
    return *this;
}
```

A mozgató értékkadás feladata a már korábban említett érték csere, melyet a `std::swap` valósít meg. Mivel a megcserélődik a `gyoker` és a `forras.gyoker` által mutatott tárterület, ezért a mozgatás teljes egészében megvalósul. Hiszen a `gyoker` egy null mutató, emiatt a csere után a `forras.gyoker` is null mutató lesz, vagyis az eredeti fa "töröldött". Természetesen ez nem szószerint igaz, hiszen csak egy másik pointeren keresztül hivatkozunk a már korábban a memóriában lefoglalt, és tárolt fára.

16.3. Hibásan implementált RSA törése

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_3.pdf (71-73 fólia) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az RSA kódolást Ron Rivest, Adi Shamir és Len Adleman fejlesztette ki 1976-ben. Ez egy nyílt kulcsú titkosító algoritmus, mely a moduláris számelméleten és a prímszámelméleten alapul. A titkosításhoz szükség van egy publikus és egy privát kulcsra, a nyílt kulccsal lehet kódolni a szövegünket, de csak a privát kóddal lehet dekódolni.

Maga az algoritmus a következő:

- i. Választani kell tetszőleges nagy prímet, **p**-t és **q**-t.

- ii. Ki kell számolni az $N = p^*q$ -t, melynél N jelöli a modulusát a nyílt és a titkos kulcsnak.
- iii. Euler-féle ϕ függvényérték kiszámítása, melynek képlete a következő: $\phi(N) = (p-1)(q-1)$
- iv. Kell találni egy olyan e számot, amely 1-nél nagyobb és $\phi(N)$ -nél kisebb. Emellett fontos, hogy e nek és $\phi(N)$ -nek a legnagyobb közös osztólya 1 legyen. Az e lesz a nyilvános kucs kitevője, ezért nyilvánossá tesszük.
- v. A privát kulcshoz is kell találni egy d kitevőt, mely esetén teljesül, hogy $d^*e \phi(N)$ osztva 1 maradékot ad. Mivel ez a titkos kulcsnak a kitevője, ezért ezt is titkosítjuk.

A linkelt diáisorban ezt az algoritmust megvalósító program forrását találjuk. Nézzük át, hogy mit is tartalmaznak ezek:

```
class KulcsPar {  
  
    java.math.BigInteger d, e, m;  
  
    public KulcsPar() {  
        int meretBitekben = 700 * (int) (java.lang.Math.log((double) ←  
            10)  
            / java.lang.Math.log((double) 2));  
  
        java.math.BigInteger p =  
            new java.math.BigInteger(meretBitekben, 100, new java. ←  
                util.Random());  
        java.math.BigInteger q =  
            new java.math.BigInteger(meretBitekben, 100, new java. ←  
                util.Random());  
  
        m = p.multiply(q);  
  
        java.math.BigInteger z = p.subtract(java.math.BigInteger.ONE).  
            multiply(q.subtract(java.math.BigInteger.ONE));  
  
        do {  
  
            do {  
                d = new java.math.BigInteger(meretBitekben, new java. ←  
                    util.Random());  
            } while (d.equals(java.math.BigInteger.ONE));  
  
        } while (!z.gcd(d).equals(java.math.BigInteger.ONE));  
  
        e = d.modInverse(z);  
  
    }  
}
```

A Kulcspar osztály feladata a fentebb leírt algoritmus megvalósítása. Ahhoz, hogy kellően nagy prím-számokat találhassunk, a Java BigInteger osztályát használjuk. Ez lehetővé teszi, hogy sokkal nagyobb értékekkel számolhassunk, mint bármelyik primitív adattípussal. Az egyszerűség kedvéért az algoritmus betűit használja a program is, kivéve az **N** betűt, melyet a forrásban **m** jelöl.

```
public class RSAPelda {

    public static void main(String[] args) {

        KulcsPar jSzereplo = new KulcsPar();

        String tisztaSzoveg = "Hello, Vilag!";

        //kódol
        byte[] buffer = tisztaSzoveg.getBytes();
        java.math.BigInteger[] titkos = new java.math.BigInteger[buffer.length];
        for (int i = 0; i < titkos.length; ++i) {
            titkos[i] = new java.math.BigInteger(new byte[] { buffer[i] });
            titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo.m);
        }

        //dekódol
        for (int i = 0; i < titkos.length; ++i) {
            titkos[i] = titkos[i].modPow(jSzereplo.d, jSzereplo.m);
            buffer[i] = titkos[i].byteValue();
        }

        System.out.println(new String (buffer));
    }
}
```

Az RSAPelda osztály végzi el a bemeneti szövegen a titkosítást és a dekódolást a KulcsPar osztály által generált publikus és privát kulcsok alapján. A szöveg titkosítása úgy zajlik, hogy a titkos tömbben eltároljuk a bemenet adott eleméhez tartozó bytokat, majd a modPow függvény segítségével vesszük a titkos[i] érékenek a e-edik hatványát modulo **m**. A dekódolás során ugyanezt a folyamatot hajtjuk végre, csak d-edik hatványát vesszük az adott titkos[i]-nek.

Azután, hogy megismertük az eredeti programot, lássuk annak átalakított (elronrott) verzióját. Az első probléma, hogy az eredeti nem karakterenként végzi a kódolást. Tehát az módosítani kell a kódon az alábbiak szerint.

```
PrintWriter writer = new PrintWriter("output.txt");
for (int idx = 0; idx < tisztaSzoveg.length(); ++idx) {
    Character tisztaSzoveg = tisztaSzoveg.charAt(idx);
    tisztaSzoveg = Character.toLowerCase(tisztaSzoveg);
```

```
byte[] buffer = tisztaSzoveg.toString().getBytes();
java.math.BigInteger[] titkos = new java.math.←
    BigInteger[buffer.length];
byte[] output = new byte[buffer.length];

for (int i = 0; i < titkos.length; ++i) {
    titkos[i] = new java.math.BigInteger(new byte[] { ←
        buffer[i]}));
    titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo ←
        .m);
    output[i] = titkos[i].byteValue();
    writer.print(titkos[i]);
}
writer.println();
writer.close();
```

A PrintWriter osztályt fogjuk használni a fájlbaíráshoz. Előnye a FileWriter osztályhoz képest, hogy képesek vagyunk átalakításnél a fájlba írni BigInteger típusú objektumokat. A karakterenkénti kódolás, ahogy már megszokhattuk, abból áll, hogy a bemeneten egy for ciklus segítségével végigmegyünk a tisztaSzoveg-en. A charAt függvény segítségével minden i. indexnél lévő karaktert tudjuk visszaadni, melyet eltráolunk lokális változóban, majd kisbetűsítünk a toLowerCase függvénnel. Erre azért van szükség, hogy könnyebben tudjuk dekódolni a szöveget. Majd a karaktert bájtsorozattá alakítjuk, végül a már korábban látott módon az RSA algoritmussal kódoljuk, az eredményt pedig kiírjuk egy fájlba.

```
BufferedReader inputStream = new BufferedReader(new ←
    FileReader("output.txt"));
int lines = 0;
String line[] = new String[100000];
while ((line[lines] = inputStream.readLine()) != null) {
    lines++;
}
inputStream.close();
```

Következő lépésként a korábban kiírt titkosított sorokat visszaolvassuk egy String tömbbe. Hogy az egyes karakterek titkosított alakját, és gyakoriságát könnyen tudjuk kezelni, ezért bevezetünk egy új osztályt, melynek neve Karakterek lesz.

```
class Karakterek {

    private String encrypted;
    private char karakter = ' ';
    private int gyakorisag = 0;

    public Karakterek(String str, char k) {
        encrypted = str;
        karakter = k;
    }

    public Karakterek(String str) {
```

```
        encrypted = str;
    }

    public void setEncrypted(String str) {
        encrypted = str;
    }

    public String getEncrypted() {
        return encrypted;
    }

    public void setKarakter(char k) {
        karakter = k;
    }

    public char getKarakter() {
        return karakter;
    }

    public void increment() {
        gyakorisag += 1;
    }

    public int getGyak() {
        return gyakorisag;
    }
}
```

Ezután a beolvasott sorokat el kell tárolni Karakterek típusú objektumokba. Ennek érdekében létrehozunk egy tömböt, melybe csak akkor rakunk új karaktert, ha még nem létezik, ellenkező esetben az increment függvény segítségével növeljük az adott karakter gyakoriságát. Ezt írja le a következő kód részlet.

```
Karakterek kar[] = new Karakterek[1000];
boolean found;
kar[0] = new Karakterek(line[0]);
int count = 1;
for (int i = 1; i < lines; i++) {
    found = false;
    for (int j = 0; j < count; j++) {
        if (kar[j].getEncrypted().equals(line[i])) {
            kar[j].increment();
            found = true;
            break;
        }
    }
    if (!found) {
        kar[count] = new Karakterek(line[i]);
        count++;
    }
}
```

}

Ha elkészítettük a tömböt, akkor ezt az elemek gyakorisága szerint sorba kell rendezni. Erre azért lesz szükség, mivel a statisztikai karaktergyakoriság alapján fogjuk ezeket a titkosított elemeket heylettesíteni normál karakterekkel. A sorba rendezés a következőképpen néz ki:

```
for (int i = 0; i < count; i++) {
    for (int j = i + 1; j < count; j++) {
        if (kar[i].getGyak() < kar[j].getGyak()) {
            Karakterek x = kar[i];
            kar[i] = kar[j];
            kar[j] = x;
        }
    }
}
```

Ahhoz, hogy a megtudjuk a statisztikai gyakoriságát az egyes karaktereknek éredmes ellátogatni a következő weboldalra: https://mdickens.me/typing/letter_frequency.html. Ezalapján elkészítjük a gyakorisag.txt fájlt, melynek minden egyes sorában egy karakter van. Ha ezt elkészítettük, akkor már csak be kell olvasni a tartalmát.

```
FileReader f = new FileReader("gyakorisag.txt");
char[] karakter = new char[70];
int karCount = 0;
int k;
while ((k = f.read()) != -1) {
    if ((char) k != '\n') {
        karakter[karCount] = (char) k;
        karCount++;
    }
}
f.close();
```

Az egyes karaktereket szintén egy tömbbe gyűjtjük össze, mely eleve a karakterekhez tartozó gyakoriság alapján lesz (csökkenően) rendezve. Ezután az a feladatunk, hogy a korábban létrehozott kar tömb elemeit kicseréljük a beolvasott karakterekre.

```
for (int i = 0; i < count; i++) {
    kar[i].setKarakter(karakter[i]);
}
```

Végezetül pedig elvégezzük a dekódolást, ami valószínűleg nem fog tökéletes eredményt adni. Ezt orvolandóan, el lehet készíteni a bemeneti szövegünkhez tartozó karaktergyakoriságot, de mivel a kódolt szöveg előzetes ismerete törésnél nem valószínű, ezért ezzel most nem élünk.

16.4. Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ebben a fejezetben azzal ismerkedünk meg, hogyan lehet változó argumentumszámú konstruktort készíteni. Szintaktikailag C++-ban a ... karakter sorozat tudatja a fordítóval, hogy bármely paramétere lehet a konstruktornak, függvénynek. Ahhoz, hogy elérjük az összes paramétert a cstdarg header-re lesz szükségünk. Lássuk, hogyan néz ki a konstruktor:

```
Perceptron ( int nof, ... )
{
    n_layers = nof;

    units = new double*[n_layers];
    n_units = new int[n_layers];

    va_list vap;

    va_start ( vap, nof );

    for ( int i {0}; i < n_layers; ++i )
    {
        n_units[i] = va_arg ( vap, int );

        if ( i )
            units[i] = new double [n_units[i]];
    }

    va_end ( vap );

    ...
}
```

A Perceptron osztály konstruktőrjében egy megnevezett paraméter van, a nof. A maradék paraméterek eléréshöz a deklaráljuk a vap va_list típusú változót, melyben a változó paramétereket tároljuk. Értékét a va_start inicializálja. A nof azt adja meg, hogy hány paramétert adtunk meg. Ez két paramétert kér, egy va_list-et melybe betölti az argumentumokat, és egy argumentumot, ahonnan kezdve betöltsse az attribútumokat. A va_arg makro az aktuális int típusú paramétert adja vissza. Előnye, hogy minden meghívása módosítja a vap tartalmát oly módon, hogy a korábban visszaadott értéket követő értéket adja vissza. Tehát segítségével be tudjuk járni a paraméterlistát. A va_end pedig végrehajtja a megfelelő műveleteket, hogy többet ne tudjuk használni az argumentum listát. Összegezve a va_start segítségével beolvassuk egy va_list-be a paramétereket, a va_arg segítségével kiolvassuk, és a va_end pedig törli a paraméterlistát.

Az előző fejezetben már használtuk a Perceptron osztályt. Ebben a fejezetben kisebb módosítást hajtunk végre a működésében. A feladatban leírtak szerint nem egy értéket fog visszaadni, hanem egy vektort, aminek a mérete megegyezik a bemeneti kép méretével. A vektorban lévő elemekkel pedig manipuláljuk az eredeti képet, és elmentjük output.png néven.

Lássuk, hogy milyen módosításokkal érhető el, hogy a egy képet kapunk vissza. Az első lépés, hogy amikor példányosítunk egy Perceptron típusú objektumot, akkor az utolsó paraméternek nem 1-et, hanem

size értéket kell adnunk.

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

Ezután a Perceptron osztályban meg kell oldanunk, hogy ezt megfelelően kezeljük, mivel alapból arra van kialakítva, hogy az utolsó paraméter 1.

```
std::vector< double > operator() ( double image [] )
{
    std::cout << "std" << std::endl;
    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
    {

        #ifdef CUDA_PRCPS

            cuda_layer ( i, n_units, units, weights );

        #else

            #pragma omp parallel for
            for ( int j = 0; j < n_units[i]; ++j )
            {
                units[i][j] = 0.0;

                for ( int k = 0; k < n_units[i-1]; ++k )
                {
                    units[i][j] += weights[i-1][j][k] * units[i-1][k];
                }

                units[i][j] = sigmoid ( units[i][j] );
            }

        #endif
    }

    std::vector< double > result_vector;

    for (int i = 0; i < units[n_layers - 1]; ++i){
        result_vector.push_back(sigmoid( units[n_layers - 1][i]));
    }
    return result_vector;
}
```

A operator() túlterhelés során egy vektort készítünk, és azt adjuk vissza. A túlterhelés vége fontos,

ahol a `result_vector`-ba betöljtjük a `sigmoid` függvény által kiszámított értékeket. Végezetül pedig a `main` függvényben a vektor értékei alapján módosítjuk az eredeti kép pixeleihez tartozó zöld értékeket.

```
    ...
    std::vector< double > value = (*p) (image);

    for (int i = 0; i<png_image.get_width(); ++i)
        for (int j = 0; j<png_image.get_height(); ++j){
            png_image[i][j].green = value[i*png_image.get_width() -->
                () + j];

        }

    png_image.write("output.png");
    ...
}
```

A `png_image.write` tagfüggvénye segítségével pedig kiírjuk a módosított képet a háttérára.

16.5. Összefoglaló

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutató „esszé” szöveget!

A mozgató és másoló konstruktur

A C++ objektum orientált nyelv, vagyis lehetőséget biztosít a polimorfizmusra, az egységbázisról, és az adatrejtést. Definiálhatunk benne osztályokat, melyek olyan tagfüggvényekkel rendelkező struktúráknak lehet nevezni, természetesen kibővítve az objektum orientáltság alapelveivel. Amikor egy osztályt példányosítunk, akkor az osztály konstruktora hívódik meg. Ha nem definiálunk konstruktort, akkor a fordító az alapértelmezett konstruktort használja. A konstruktorkok olyan függvények, melyeknek feladata az objektum inicializálása. Vagyis ebben szoktunk a az egyes tagváltozóknak értéket adni, bizonyos tagfüggvényeket meghívni, melyek szükségesek az objektum alapjának elkészítéséhez. Egy másik szembetűnő tulajdonsága a konstruktorknak, hogy a nevük megegyezik az osztály nevével. A konstruktur ellentéte a destruktur, mely az objektum törlésével együtt hívódik meg, és felszabadítja a már nem használt memoriaterületeket. Egy jól megírt C++ programban viszont nem elég ennek a kettőnek a megléte, szükség van még másoló és mozgató konstruktorkra.

A másoló és mozgató konstruktorkok segítségével szintén objektumok inicializálását hajthatjuk végre, viszont egy másik azonos osztályú objektum alapján. Ha nem szeretnénk megírni ezeket, akkor a legegyszerűbb, ha letiltjuk használatukat, vagyis privát taggá tesszük őket. Erre azért van szükség, mert ha nem tiltjuk le őket, akkor a fordító az alapértelmezett másoló/mozgató konstruktort hívja meg, ami váratlan eredményekhez vezethet, kiszámíthatatlanná teszi a program működését.

A másoló konstruktur, ahogy a nevében is benne van egy objektumot másol le, és az alapján készít egy másikat. Onnan lehet felismerni, hogy paraméterként minden az őt magába foglaló osztálytalál azonos osztályú objektumrefenciát vár. Két fajta másolásról beszélhetünk: sekély és mély másolás. A sekély másolás lényege, hogy csak létrehozunk egy másik mutatót, ami a paraméterként megadott objektumra mutat. Ennek

implementálása a legegszerűbb, viszont használata problémákat okozhat. A legfőbb hátrány eme megoldási módnak, hogy közös a memóriaterület, tehát ha az egyiket módosítjuk, akkor az a másikról is kifejezi hatását. Tehát ez nem egy igazi másolat, nevezhetnénk alias-nak. A mély másolás ezzel szemben egy különálló objektumot hoz létre, mely megegyezik a paraméterről kapott elemmel, viszont külön memóriacímen. Sekély társához képest implementálása összetettebb, viszont magabiztos használata jobb C++ programozóvá tesz.

A 2. feladatban megvalósított másoló konstruktor a mély másolást implementálja. Tehát a forrásként kapott LZW fát bejárjuk, és minden elemnek a másolatát elkészítjük a konstruálandó fában is. Ennek következtében lesz két külön fánk a méoriában, külön gyökér és fa mutatóval. Binárisfa bejárások közül létezik preorder, postorder és inorder. Ezek közül bármelyiket használhatjuk, mindegyik a megfelelő eredményhez vezet. A mi programunk a postorder bejárást alkalmazza, vagyis elsőnek dolgozzuk fel a gyemeket, majd gyökeret.

A mozgató konstruktor a többi konstruktorhoz megfelelően egy inicializálást hajt végre, a másoláshoz hasonlóan ez is egy másik objektumra alapul. A különbség annyi, hogy a még a másolásnál a másolásban részt vevő objektumok megmaradnak, addig a mozgatásnál a forrás objektum megszűnik, pontosabban nem determináns állapotba kerül. A mozgató konstruktor arról ismerszik meg, hogy a paramétere egy jobbérték referencia. A jobbérték referenciát a `&&` jelöli. Annak érdekében, hogy ez a konstruktor hívódjon meg, a `std::move` függvényt kell meghívni. Gyakori tévedés, köszönhetően ennek a függvénynek a félreérthető nevének, hogy mozgatást hajt végre. Valójában a paraméterről kapott objektum jobbérték referenciájával tér vissza.

Az LZWBInFa programunkban maga a mozgató konstruktor a mozgató értékkadásra van alapozva. Az alap koncepció az, hogy az inicializálni kívánt fa gyökerét null mutatóvá tesszük, majd megcseréljük a forrás és a cél fa gyökér mutatójának értékét, ezzel megvalósítva a mozgatást. Ezt úgy érjük el, hogy a mozgató konstruktoron keresztül meghívjuk a mozgató értékkadást. Fontos látni, hogy mikor hívódik meg a konstruktor és mikor az értékkadás. Ha egy még nem inicializált objektumot szeretnénk másolással, mozgatással inicializálni, akkor a konstruktor hívódik. Ezzel szemben egy már inicializált objektumba szeretnénk másolni, mozgatni, akkor már az értékkadás hajtódik végre. Ennek tudatában szerepel a mozgató konstruktorban a következő sor:

```
*this = std::move(forras);
```

Szóval meghívódik a mozgató értékkadás, ahol a pointer értékek cseréjét hajtjuk végre. Ehhez a `std::swap` függvényt használjuk. Természetesen lehetne saját cserét is írni, ebben a könyvben több megoldást adtunk erre, de összességében elmondható, hogy érdemes a már előre implementált függvényeket, metódusokat használni.

Összegezve a leírtakat, a másoló és mozgató szemantika a C++ nyelvben kiemelt szerepet játszik. Magabiztos használatuk elsajátítása nélkülözhetetlen a megbízható programok készítéséhez.

16.6. EPAM: It's gone. Or is it?

Adott a következő osztály:

```
public class BugousStuffProducer {  
    private final Writer writer;  
    public BugousStuffProducer(String outputFileName) throws  
        IOException {
```

```
writer = new FileWriter(outputFileName);
}
public void writeStuff() throws IOException {
writer.write("Stuff");
}
@Override
public void finalize() throws IOException {
writer.close();
}
}
```

Mutass példát arra az esetre, amikor előfordulhat, hogy bár a program futása során meghívtuk a writeStuff() metódust, a fájl, amibe írtunk még is üres.

16.7. EPAM: Kind of equal

Adott az alábbi kódrészlet.

```
// Given
String first = "...";
String second = "...";
String third = "...";
// When
var firstMatchesSecondWithEquals = first.equals(second);
var firstMatchesSecondWithEqualToOperator = first == second;
var firstMatchesThirdWithEquals = first.equals(third);
var firstMatchesThirdWithEqualToOperator = first == third;
}
```

Változtasd meg a String third = "..."; sort úgy, hogy a firstMatchesSecondWithEquals, firstMatchesSecondWithEqualToOperator, firstMatchesThirdWithEquals értéke true, a firstMatchesThirdWithEqualToOperator értéke pedig false legyen. Magyarázd meg, mi történik a háttérben.

16.8. EPAM: Java GC

Mutasd be nagy vonalakban hogyan működik Java-ban a GC (Garbage Collector). Lehetséges az OutOfMemoryError kezelése, ha igen minden esetekben? Források: <https://medium.com/@hasithalgamge/seven-types-of-java-garbage-collectors-6297a1418e82>, <https://stackoverflow.com/questions/2679330/catching-javlang-outofmemoryerror>

17. fejezet

Helló, Gödel!

17.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban

[https://youtu.be/DL6iQwPx1Yw\(8:05-től\)](https://youtu.be/DL6iQwPx1Yw(8:05-től))

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Azt fogjuk megnézni, hogy hogyan lehet `std::sort()` függvényt használni. A következőkben a kódot fogjuk elemezni.

```
std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( ←
              Gangster x, Gangster y )
{
    return dst ( cop, x.to ) < dst ( cop, y.to );
} );
```

Az `std::sort()` függvénynek létezik kettő illetve három paraméteres verziója. Az első két paraméter minden az, hogy mettől meddig szeretnénk rendezni a vektort, vagy tömböt. A harmadik pedig egy opcionális paraméter, mely a rendezés alapjául szolgáló függvényt adja meg. Ha ezt kihagyjuk, akkor az alapértelmezett függvény dönti el, hogy mi alapján fogja rendezni az elemeket.

Ha megnézzük a kódunkat akkor az látható hogy az első paraméter a `gangsters` vektor első elemének az indexét a második paraméter viszont a vektor utolsó elemének az indexét tartalmazza. Ezeket a `begin()` és az `end()` függvény segítségével tudjuk megvalósítani. Mivel látható egy harmadik paraméter is így arról is érdemes pár szót beszélni. Ez egy név nélküli függvény vagy ha úgy tetszik lambda függvény.

A lambda függvény C++11 óta támogatott. A szintaxisa a következő:

```
[] (paraméterek) -> visszatérési típusa {utasítások}
```

A [] jelek közé lehet olyan változókat megadni, amelyeket a függvényen kívül szeretnénk használni. A feladatban az aktuális objektum mutatójára és a `cop` objektumra van szükség. Paraméterül két `Gangster` osztályú objektumot kap, melyek a `sort` által összehasonlított elemek. A lambdában lévő utasítás pedig azt

vizsgálja, hogy melyik gangster van közelebb a rendőrhöz. A vizsgálat eredményeként egy bool értékkel tér vissza.

Vagyis az történik, hogy az `std::sort()` függvény, és egy lambda függvény segítségével a `gangsters` vektor elemeit a rendőröktől való távolságuk szerint rendezzük növekvő sorrendbe.

17.2. C++11 Custom Allocator

<https://prezi.com/jvvbytkwgsxj/high-level-programming-languages-2-c11-allocators/> a CustomAlloc-os példa, lásd C forrást az UDPORG repóban

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Amikor példányosítunk egy objektumot, akkor a memóriában automatikusan lefoglalódik az ehhez szükséges terület. C++-ban a `new` operátorral tudjuk meghívni az alapértelmezett allokátort. Viszont ha szeretnénk beleszólni a programunk memóriakezelésébe, akkor saját allokátor osztályt kell írnunk. A továbbiakban `CustomAlloc` osztályunkat fogjuk megnézni amit azért hoztunk létre, hogy nyomonkövessük a lefoglalt memóriaterületeket.

```
template<typename T>
class CustomAlloc
{
public:
    CustomAlloc() {}
    CustomAlloc(const CustomAlloc&) {}
    ~CustomAlloc() {}

    using size_type = size_t;
    using value_type = T;
    using pointer = T*;
    using const_pointer = const T*;
    using reference = T&;
    using const_reference = const T&;
    using difference_type = ptrdiff_t;

    pointer allocate( size_type n) {
        int s;
        char* p = abi::__cxa_demangle( typeid (T).name ←
            (), 0, 0, &s);
        std::cout << "Allocating"
            << n << " objects of "
            << n*sizeof (T)
            << " bytes. "
            << typeid (T).name() << "=" << p
            << std::endl;
        delete p;
    }
}
```

```
        return reinterpret_cast<T*>(new char[n*sizeof(T) ←
            )]);
    }

    void deallocate(pointer p, size_type n) {
        delete[] reinterpret_cast<char*>(p);
        std::cout << "Deallocating"
            << n << " objects of "
            << n*sizeof(T)
            << " bytes."
            << typeid(T).name() << "=" << p
            << std::endl;
    }
};
```

Ahogy látható ez egy template-elt osztály, tehát bármilyen típussal használhatjuk. Újdonságként számít a `using` kulcsszó. Ez a C++11-től kezdve a `typedef` helyett használható. Tehát arra szolgál, hogy az egyenlőségjel jobb oldalán található típusokra más néven is tudunk hivatkozni. Az `allocate` osztály végezi a szükséges terület lefoglalását, és erre a területre mutató pointerrel tér vissza. Ebben a függvényben az `std` névtér mellett megismerkedünk egy újjal, mely az `abi`. Ez az *Application Binary Interface* rövidítése. Ennek segítségével meg tudjuk határozni, hogyan legyenek az argumentumok tárolva a hívási stack-ben, a registerekben, hogyan legyenek rendezve és padding-elve az egyes típusok a struktúrákban. Mielőtt tovább haladnánk a kód elemzésében, elsőnek ejtsünk párt a **name mangling**-ról.

A lényege annyi, hogy a forrásfájlban szereplő neveket átalakítja belsőleg használtakra. Például a `void pelda(int i, char b)` függvényt a fordító átalakítja `pelda_FviCE` formára. A **name mangling** lényege, hogy megkönnyítse a linker feladatát abban, hogy megkülönböztesse az egyes neveket. Ahogy látható a fenti példában, az átalakított formának is tartalmaznia kell a függvény nevét, és a kért paraméterek típusát. A visszatérési érték mindenkorán az F utáni első betű, ami jelenleg `void`-ot jelöli.

Ezek után már körvonalazódik, hogy mire fogjuk használni a `__cxa_demangle` függvényt. Ennek segítségével tudjuk visszafejteni az átalakított alakból az eredetit. A `typeid().name()` segítségével megkapjuk, hogy egy objektum típusa milyen néven lett átalakítva. Ebben az esetben a fenti példában tárgyalt `void pelda(int i, char b)` esetén a `typeid(pelda).name()` függvény a `FviCE` kifejezést adja vissza. Tehát a `CustomAlloc` osztályunkban a `T` típusát fogjuk eltárolni karaktertömbként a `p` változóban. Majd `cout` segítségével kiíratjuk a lefoglalt bájtok méretét, és, hogy milyen típusnak foglaltunk területet. Végezetül pedig visszatérünk egy `T` típusú pointerrel, mely egy újonnan lefoglalt memóriaterületre mutat.

A `deallocate` függvény a memória felszabadítását végzi. Ezek után kiiírjuk a terminálba, hogy mennyi a felszabadított bájtok száma és azt, hogy melyik memóriaterületet szabadítottuk fel.

A `main` függvény a következőképpen néz ki:

```
int main()
{
    std::vector<int, CustomAlloc<int>> v;

    v.push_back(1);
    v.push_back(2);
```

```
v.push_back(3);
v.push_back(4);
v.push_back(5);
v.push_back(6);
v.push_back(7);
v.push_back(8);
v.push_back(9);
v.push_back(10);
v.push_back(11);
v.push_back(12);
v.push_back(13);
v.push_back(14);
v.push_back(15);

for(int x : v){
    std::cout << x << std::endl;
}

std::cout << typeid(pelda).name() << std::endl;
return 0;
}
```

Deklarálunk egy vektort, majd ezek után megadjuk, hogy a mi allokátorunkat használja. Majd elkezdünk bekerakni elemeket, és végül kiíratjuk a vektorban tárolt elemeket. A vektorok dinamikus adatszerkezetek, tehát "bármennyi" eleme lehet. Viszont ha betelik a lefoglalt terület, és új elemet adunk hozzá, akkor egy új területet foglal le, melynek mérete az előző duplája lesz. A már benne lévő elemeket átmásolja az új területre, majd törli a régit. Viszont ha használjuk a vektor `reserve` tagfüggvényét, akkor megadhatjuk, hogy hány bájtot foglaljon le. Ennek segítségével elérhetjük, hogy soha ne kelljen újabb területet foglalnia a vektornak.

17.3. STL map érték szerinti rendezése

Például: <https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az STL szó a Standard Template Library rövidítése, melynek fontos részét képezik a tárolók. Alapvető cél az volt, hogy olyan adatszerkezeteket használjanak, melyek hatékonyan, biztonságosan, kivételbiztosan és típushelyesen tárolják az adatokat. STL tároló például a `vector`, a feladatban szereplő `map`, de a `list`, a `set` és a `stack` is ide sorolható.

A `map` tárolók asszociatív tömbök, melyek kulcs-adat párokból áll, és kulcsértékek alapján növekvően van rendezve. Lássuk, hogy hogyan használjuk ezt a `fenykard.cpp` forrásban.

```
std::vector<std::pair<std::string, int>> sort_map ( std::map<std::string, int> &rank )
{
```

```
    std::vector<std::pair<std::string, int>> ordered;

    for ( auto & i : rank ) {
        if ( i.second ) {
            std::pair<std::string, int> p {i.first, i.second};
            ordered.push_back ( p );
        }
    }

    std::sort (
        std::begin ( ordered ), std::end ( ordered ),
        [ = ] ( auto && p1, auto && p2 ) {
            return p1.second > p2.second;
        }
    );

    return ordered;
}
```

Látható, hogy a `sort_map` függvény visszatérési értékének típusa egy vektor. Az érdekesség, hogy párokból áll. Ehhez az `std::pair` adatszerkezetet használjuk, mely lehetővé teszi heterogén adatok párbán való tárolását. Vagyis jelen esetben sztringeket és egészek páját tudjuk tárolni. Az `std::map` Kulcs és érték párokból áll, és minden kulcs csak egyszer fordulhat benne elő. Maga a függvény által végrehajtott utasítások már ismertek. Elsőnek bejárjuk a `rank` map-et, és az érték párokat beletesszük az `ordered` vektorba. Említést érdemel az `auto` típus. Ez a C++11 óta létező funkció, melynek lényege, hogy a fordító automatikusan találja meg a megfelelő típust az adott változóhoz. Végül a vektor rendezése az előző feladatban megismert lambda kifejezéssel történik. Vagyis értékek alapján rendezzük át a vektort. A fenykard programot régen arra használták, hogy jegyeket adjanak a diákoknak. Ekkor már logikusnak tűnik, hogy elsőnek a `map`-ben kell tárolni az egyes hallgatókat, hogy ne szerepelhessen többször ugyanaz a tanuló. Viszont az értékelés szempontjából nem kívánatos, hogy név alapján legyen rendezve a tömb. Ahhoz, hogy a teljesítményt lehessen osztályozni, át kell alakítani az adatszerkezetet érték szerint rendezetté.

17.4. Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a `java.lang Interface Comparable<T>` szerepét!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az Alternatív Tabella a magyar labdarúgó bajnokságon résztvevő csapatok rangsorát tartalmazza. A plusz dolog annyi az egészben, hogy nem csak azt vizsgálja, hogy az adott csapat nyert, vesztett vagy döntetlenet játszott, hanem azt is, hogy ki ellen érte el az adott eredményt.

A továbbiakban az összehasonlítás menetével fogunk megismerkedni. A csapatok kezeléséhez létrehozunk egy `Csapat` osztályt, mely implementálja a `Comparable` interfést. Az osztály a következő:

```
class Csapat implements Comparable<Csapat> {

    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }

    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Tehát egy Csapat típusú objektum rendelkezik egy névvel és egy értékkal. Mivel implementáltuk a Comparable interfészét, ezért definiálnunk kell a compareTo függvényt. Ennek a segítségével tudjuk megmondani, hogyan legyen összehasonlítva a két objektum.

Az eredeti példában két programunk volt. Az egyik Wiki2Matrixx. Ez a csapatok egymás elleni eredményeit tartalmazó kereszttáblázatból kiszámolja a linkmátrixot, amely alapjául szolgál a AlternativTabella programunkhoz. Viszont ekkor a kiírt linkmátrixot be kellett másolni az AlternativTabella forrásába, mely nem túl felhasználóbarát. Ezért a linkmátrixot a Wiki2Matrix osztály tagváltozójában tároljuk el. Ezt a nyom3 függvény átalakításával érjük el.

```
void nyom3(int[][][] k, int[] oszlopOsszeg) {
    for (int i = 0; i < k.length; ++i) {
        matrix.add(new ArrayList<>());
        System.out.println();
        System.out.print("{");
        for (int j = 0; j < k[i].length; ++j) {

            if (oszlopOsszeg[j] != 0.0) {
                System.out.print(k[i][j] * (1.0 / ←
                    oszlopOsszeg[j]) + ", ");
                matrix[i][j] = k[i][j] * (1.0 / ←
                    oszlopOsszeg[j]);
                matrix.get(i).add(k[i][j] * (1.0 / ←
                    oszlopOsszeg[j]));
            } else {
                System.out.print(k[i][j] + ", ");
                matrix.get(i).add(k[i][j] * 1.0);
            }
        }
    }
}
```

```
        }
        System.out.print("} , ");
    }
}
```

Magát a linkmátrixot egy kétdimenziós ArrayList-ben tároljuk, mely a Vector osztályt helyettesíti. A másik módosítás a AlternativTabella osztályban történt.

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

    Wiki2Matrix Matrix = new Wiki2Matrix();

    ArrayList<ArrayList<Double>> L = Matrix.getMatrix();

    ...

}
```

Tehát példányosítjuk a Wiki2Matrix osztályt. Végül pedig az osztály getMatrix() tagfüggvényével érjük el.

17.5. Prolog családfa

Ágyazd be a Prolog családfa programot C++ vagy Java programba! Lásd para_prog_guide.pdf!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Prolog egy programozási nyelv, melynek segítségével matematikai logikai formulákat tudunk vizsgálni.

SWI-Prolog használata



```
sudo apt install swi-prolog-jav
```

Az előző parancs sikeres lefutása után a NetBeans-ben a projektet megnyitva a Libraries-ra jobb egér kattintás majd hozzáadjuk a könyvtárat. Először létrehozunk egy új osztályt, majd a /usr/lib/swi-prolog/lib/jpl.jar fájlt kell megadni a ClassPath mezőben.

A program úgy működik, hogy elsőnek betöljük a Prolog fájlt, majd arról készítünk lekérdezéseket.

A family.pl fájl tartalmazza a pdf-ben megadott Prolog programot.

```
férfi(nándi).
férfi(matyi).
férfi(norbi).
férfi(dodi).
```

```
férfi(joska).  
nő(gréta).  
nő(erika).  
nő(kitti).  
nő(marica).  
gyereke(nándi, norbi).  
gyereke(matyi, norbi).  
gyereke(gréta, norbi).  
gyereke(nándi, erika).  
gyereke(matyi, erika).  
gyereke(gréta, erika).  
gyereke(norbi, dodi).  
gyereke(norbi, kitti).  
gyereke(erika, joska).  
gyereke(erika, marica).
```

... .

Ezt a programunkba a következő módon fogjuk beolvasni.

```
String s = "consult('family.pl');  
Query q = new Query(s);  
System.out.println(q.hasSolution());
```

Először megadunk egy stringet, ami a kiértékelni kívánt formulát tartalmazza. Majd létrehozunk egy Query osztályú objektumot. Ez az osztály teszi lehetővé, hogy kiértékeljük a formula igazságértékét, vagy az egyes változók lehetséges értékeit. Ha csak az igazságértékre vagyunk kíváncsiak, akkor a hasSolution() függvényt kell használni. Ha itt hamis eredményt ad, akkor a fájlt nem találja.

```
String t2 = "apa(gréta)";  
System.out.println(t2 + " is " + (Query.hasSolution(t2) ? " ↵  
provable" : "not provable"));
```

Az első példa tehát azt mutatja be, hogy hogyan lehet megtudni a formula igazságértékét. Jelen esetben azt vizsgáljuk, hogy Gréta apának tekinthető-e. Mivel nem, ezért a "not provable" kifejezés jelenik meg.

```
String t3 = "nagyapja(X, matyi)";  
System.out.println("each solution of " + t3);  
Query q3 = new Query(t3);  
while (q3.hasMoreSolutions()) {  
    Map<String, Term> s3 = q3.nextSolution();  
    System.out.println("X = " + s3.get("X"));  
}
```

A változók értékének lehetséges értékeit kétféleképpen lehet felsorolni. A fenti példában azokat az X-eket keressük, akik Matyi nagyapjai lehetnek. Ezt úgy tudjuk megfogalmazni matematikai logikával, hogy azokat az X-eket keressük, akik Matyi apjának vagy anyjának az apja. Tehát végig iterálunk a lehetséges X-eken. Ehhez a hasMoreSolution() használjuk, mely azt adja vissza, hogy van-e még több megoldás. A nextSolution() eredményét eltároljuk kulcs-érték formátumban. Ezt a Map interfész segítségével

oldjuk meg. Ebből az adatszerkezetből a get tagfüggvényel tudjuk kinyerni az "X"-hez tartozó értéket. Itt látható egy másik megoldás a változóértékek kilistázására.

```
String t6 = "testvér(matyi, X)";  
Map<String, Term>[] ss6 = Query.allSolutions(t6);  
System.out.println("all solutions of " + t6);  
for (int i = 0; i < ss6.length; i++) {  
    System.out.println("X = " + ss6[i].get("X"));  
}
```

Most azt fogjuk használni, hogy a kulcs-érték párokból tömböt készítünk, és egy for ciklus segítségével végig megyünk rajta és kiírjuk. Ebben a részben azt vizsgáljuk, hogy kik Matyi testvérei. Matyi testvérei azok akiknek ugyanaz az apja és anyja.

```
String t4 = "nagyapa(X)";  
System.out.println("first solution of " + t4 + ": X = " + ←  
    Query.oneSolution(t4).get("X"));
```

Egy másik példában azokat az X-eket keressük, akik nagyapák, vagyis olyan személyeket, akinek a gyermeké szülő.

```
String t7 = "gyereke(X, norbi)";  
System.out.println("each solution of " + t7);  
Query q7 = new Query(t7);  
while (q7.hasMoreSolutions()) {  
    Map<String, Term> s7 = q7.nextSolution();  
    System.out.println("X = " + s7.get("X"));  
}
```

Azt nézzük meg, hogy kik Norbi gyermekei. Ahhoz, hogy Norbi gyermekeit megtaláljuk, ahhoz nem kell semmit tennünk, hiszen, ezt a family.pl fájlban leírtuk.

17.6. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témat (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ezt a feladatot már feldolgoztam az előző félévben 9.2 és 9.3

17.7. EPAM: Mátrix szorzás Stream API-val

Implementáld le a mátrix szorzást Java-ban for és while ciklusok használata nélkül.

17.8. EPAM: LinkedList vs ArrayList

Mutass rá konkrét esetekre amikor a Java-beli LinkedList és ArrayList rosszabb performanciát eredményezhet a másikhoz képest. (Lásd még LinkedList és ArrayList forráskódja). Végezz méréseket is. (mit csinál az ArrayList amikor megtelik)

LinkedList > ArrayList

```
public static void main(String[] args) {  
  
    ArrayList<Integer> arrli  
        = new ArrayList<Integer>();  
  
    for (int i = 1; i <= 10000000; i++) {  
        arrli.add(i);  
    }  
    long startTime = System.currentTimeMillis();  
    arrli.remove(0);  
    arrli.add(0, 0);  
    long stopTime = System.currentTimeMillis();  
    long elapsedTime = stopTime - startTime;  
    System.out.println(elapsedTime + " milli");  
  
    LinkedList<String> object  
        = new LinkedList<String>();  
  
    for (int i = 1; i <= 10000000; i++) {  
        object.add("A");  
    }  
    long startTime2 = System.currentTimeMillis();  
    object.remove(0);  
    object.set(0, "A");  
    long stopTime2 = System.currentTimeMillis();  
    long elapsedTime2 = stopTime2 - startTime2;  
    System.out.println(elapsedTime2);  
  
}  
}
```

ArrayList > LinkedList

```
long startTime3 = System.currentTimeMillis();  
arrli.get(4999999);  
long stopTime3 = System.currentTimeMillis();  
long elapsedTime3 = stopTime3 - startTime3;  
System.out.println(elapsedTime3);  
  
long startTime4 = System.currentTimeMillis();  
object.get(4999999);  
long stopTime4 = System.currentTimeMillis();  
long elapsedTime4 = stopTime4 - startTime4;
```

```
System.out.println(elapsedTime4);  
}
```

ArrayList vs LinkedList

- 1.Keresés: ArrayList kereső operációja O(1)-es sebessége van miközben a LinkedListnek csak O(n)
 - 2.Törlés: LinkedList törlő operációja O(1) sebességű az ArrayListnek O(n) (legrosszabb esetben ha az első elemet akarjuk kitörölni) és O(1) ha az utolsó elemet akarjuk kitörölni. Azért gyorsabb a törlés linkedlistnél mert minden elemnek van 2 pointerre ami a szomszédjaira mutat emiatt ha kitörlünk egy elemet csak a pointer lokációját kell megváltoztatni de arraylistnél meg shiftelnünk kell az összes elemet.
 - 3.Beszúrás: Linkedlist O(1) ArrayList O(n)
 - 4.Memória használat: ArrayList csak az elemek indexét és maga az elemet tárolja a LinkedList meg az elemet és az elemek a 2 szomszéd pointerjét így magas lesz a memória használata.
- Konklúzió: Használunk LinkedListet ha olyan tárolóna van szükségünk amihez gyakran addunk és eltávolítunk elemeket. Használunk a ArrayListet ha olyan tárolót akarunk ahol inkább az elemek értékére van szükségünk.

17.9. EPAM: Refactoring

Adott egy “legacy” kód mely tartalmaz anonymous interface implementációkat, ciklusokat és feltételes kifejezések. Ebben a feladatban ezt a “legacy” kódot szeretnénk átírni lambda kifejezések segítségével (metódus referencia használata előnyt jelent!)

18. fejezet

Helló,...!

18.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>

Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocaremulator/blob/master/justine/rcemu/src/carlexer.ll>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Volt már arról szó, hogy az OOCWC projekt lexerjében mire használjuk a sscanf-et. Ez a függvény egy formázott sztringet vár bemenetül. Az első paramétere az a sztring, amiből olvassa a bemenetet, a többi paraméterrel pedig azt adjuk meg, hogyan tárolja azt. Itt egy példa az OOCWC projekt myshmclient .cpp fájljából.

```
while ( std::sscanf ( data+nn, "<OK %d %u %u %u>%n", &idd, &f, &t, &s, &n ) == 4 )
{
    nn += n;
    gangsters.push_back ( Gangster {idd, f, t, s} );
}
```

Ebben az esetben olyan sztringet olvasunk be, amely <OK-val kezdődik. Utána várunk egy egészet, és 3 előjel nélküli egészet. Az utolsó paraméter az érdekesebb, hiszen az eddigiekkel már találkozhattunk a printf használatakor is. Az utolsó paraméternek egy olyan egészet adunk meg, melyben a sscanf a beolvasott bájtok számát adja meg. Ezt tároljuk el az n változóban. Értékét arra használjuk fel, hogy a beolvasni kívánt sztringen tovább lépjünk, ne olvassuk be újra ugyanazt az elemet. Azért van arra szükség, hogy ezt a beolvasást egy cikluson belül hajtsuk végre, mert nem tudhatjuk biztosan, hogy hány gengszter van a térképen. A ciklus addig folytatódik, ameddig a az argumentumlista 4 elemének értéket tud adni. Ugyanis a sscanf sikeres beolvasás esetén a helyesen "kitöltött" elemek számát adja vissza.

18.3. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.4. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esporttalent-search>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Szükséges csomag

```
sudo apt install libopencv-dev
```

Ez a program a karakter-elvészést vizsgálja. Rajta kell tartani az egeret Samu Entropy-n, eközben egyre több a négyzetek száma a képernyőn. A program bezárása után egy értékelést láthatunk.

A Qt egy alkalmazás-keretrendszer, mely lehetővé teszi GUI-s alkalmazások készítését, de hagyományos parancssoros alkalmazásokat is fejleszhetünk vele.

Grafikus felületek fejlesztésekor elvárás, hogy egyes elemeken végrehajtott interakciók, más elemeken fejtések ki hatásukat. A Qt egy új mechanizmust vezetett be, mely a signal-slot. Például, ha megnyomunk egy gombot, az egy signalt vált ki, melynek hatására a vele összekapcsolt objektum egyik slotját hajtja végre. Ha megnézzük a BrainB program forrását, akkor láthatjuk, hogyan is néz ez ki.

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ) ,  
          this, SLOT ( updateHeroes ( QImage, int, int ) ) );
```

Látható, hogy melyik a signal és melyik a slot.

A signal-ok olyan függvényeknek tekinthetők, amik nem rendelkeznek definícióval, csak deklarációval. Nem lehet visszatérési értéke, viszont lehetnek paramétereik. Fontos tulajdonságuk a signal-oknak, hogy nem lehet őket hagyományos módon meghívni, ehhez `emit` makrót kell használni.

```
emit mysignal(value);
```

A slot-ok ezzel szemben teljes mértékben a függvényekre hasonlítanak, meg lehet őket hívni, mint a függvényeket, van definíciójuk, lehetnek paramétereik, viszont ezeknek sem lehet visszatérési értékük. Fontos, hogy csak olyan signal-slot párokat tudunk összekötni, melyek kompatibilisek egymással. Ez alatt azt kell érteni, hogyha a signal paramétere 2 db egész, akkor a slot-nak vagy nem szabad lennie paraméterének, vagy pedig azonos paramétereket kell kérnie. Ez azért fontos, mert a signal paramétereinek értékei át tudnak adódni a slot-nak.

A Qt által biztosított osztályok mind tartalmaznak beépített signal-okat és slot-okat. Ha egyedi megoldásokat is használni akarunk, akkor készíthetünk leszármazott osztályokat. Ha olyan osztályt akarunk készíteni, ami képes kezelni a signal-slot mechanizmust, akkor mindenkorban tartalmaznia kell az osztálynak a `Q_OBJECT` makrót. Egy pontosítás még szükséges, ugyanis csak akkor lehet ezt a makrót használni, ha az osztályunk őse a `QObject` osztály, vagy a ős osztályunk leszármazottja a `QObject` osztálynak. A Qt összes saját osztálya leszármazottja ennek az osztálynak, tehát csak az általunk készített osztályokkal kell figyelni erre a kitételre.

Ha programunk fejlesztése során szükségünk lenne sablon osztályokra, akkor érdemes megismekedni a Woboq GmbH által fejlesztett Verdigris projektet (<https://github.com/woboq/verdigris>). A templatelt osztályok támogatása mellett gyorsabb fordítási időt érhetsünk el, és programunk működése is egy kicsit gördülékenyebb lesz.

Végül megnézzük, hogy mire is használjuk a signal-slot mechanizmust a BrainB-ben.

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ) ,
           this, SLOT ( updateHeroes ( QImage, int, int ) ) );

connect ( brainBThread, SIGNAL ( endAndStats ( int ) ) ,
           this, SLOT ( endAndStats ( int ) ) );
```

A BrainB-ben két fő osztályunk van, az egyik az ablak, és az abban lévő objektumok mejelenítéséért felel, meg a számításokért. Az első `connect` függvénynél azt láthatjuk, hogy ha megváltozik a hősünk száma/ pozíciója, akkor a `brainBThread` objektum `heroesChanged` signal-ja kiváltódik, ehhez hozzákötjük az ablakunkat kezelő osztály `updateHeroes` slot-ját. Ezzel érjük el azt, hogy minden a számítási osztály által végzett módosulás láthatóvá váljon a képernyőn. A másik `connect` pedig azt teszi lehetővé, hogyha leállítjuk a programot, akkor záródjon be az ablak.

18.5. OSM térképre rajzolása

Debrecen térképre dobunk rá cuccokat, ennek mintájára, ahol én az országba helyeztem el a DEAC hekkereket: <https://www.twitch.tv/videos/182262537> (de az OOCWC Java Swinges megjelenítőjéből: <https://github.com/nbatfai/robocar-emulator/tree/master/justine/rcwin> is kiindulhatsz, mondjuk az komplexebb, mert ott időfejlődés is van...)

Alternatívaként készíthetsz egy GoogleMaps alapú Androidos „GPS trackert”, 2007 óta csinálok ilyen példát: <https://youtu.be/QStgBZ6JfAU> az aktuális a Bátfai Haxor Stream keretében: https://bhaxor.blog.hu/2018/09/19/nandigps_ismerkedes_a_gps-el

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.6. EPAM: XML feldolgozás

Adott egy koordinátákat és államokat tartalmazó XML (kb 210ezer sor), ezt az XML-t feldolgozva szeretnék létrehozni egy SVG fájlt, melyben minden város megjelenik egy pont formájában az adott koordináták alapján (tetszőleges színnel) Plusz feladat: A városokat csoportosíthatjuk államok szerint, és minden állam külön színnel jelenjen meg a térképen, így látszódni fognak a határok is.

18.7. EPAM: ASCII Art

ASCII Art in Java! Implementálj egy Java parancssori programot, ami beolvas egy képet és kirajzolja azt a parancssorba és / vagy egy szöveges fájlba is ASCII karakterekkel.

18.8. EPAM: Titkos üzenet, száll a gépben!

Implementájl egy olyan parancssori alkalmazást, amely a billentyűzetről olvas soronként ASCII karakterből álló sorokat, és a beolvasott szöveget Caesar kódolással egy txt fájlba írja soronként.

19. fejezet

Helló, Lauda!

19.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére!

<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#id527287>

Tanulságok, tapasztalatok, magyarázat...

A Java nyelvben kiemelt szerepe van a kivételkezelésnek, ugyanis szerves része a nyelvnek. Sok esetben a Java VM nem is engedi fordítani a kódot, ha nem kezeljük a lehetséges kivételeket. A továbbiakban arról fogunk beszálni, hogy miért érdemes használni a hibakezelést és mit is csinál ez pontosan.

A kivétel a program végrehatása során keletkezik, mely megszakítja az utasítások végrehajtását. A nem objektumorientált nyelvekben, mint a C, minden gyanús helyen külön kellett kezelnünk az egyes kivételeket, mely egy hibaüzenet kiírásából és a program leállításából állt. Ezzel szemben C++/Java-ban akár egy helyen is kezelhetjük őket. Az alap szintaxisa a objektum orientált nyelvekben a kivételkezelésnek a következő:

```
try {
    ...
}
catch(kivétel deklaráció) {
    ...
}
finally{
    ...
}
```

Annyit csinálunk, hogy a vizsgálni kívánt kódrészegyszerűen behelyezzük egy `try` blokkba, majd pedig tetszőleges számú `catch` blokk segítségével kezeljük a kivételeket. A `finally` blokk mindenkor meg hívódik, függetlenül attól, hogy történt-e kivétdobás vagy sem. Ebben a blokban lehetjük meg, hogy bezárjuk a megnyitott fájlokat amire már nem lesz szükségünk. Ahhoz, hogy kivételkezeléssel tudunk foglalkozni, ahol kellenek kivételek, melyeket a metódusok a `throw` kulcsszóval teszik lehetővé. Ezután vesszővel elválasztva tudjuk felsorolni a dobható kivételek típusát. A Java nyelv több osztályt biztosít a kivételek kezeléséhez, melyek minden a `Throwable` osztály leszármazottai. De mi is hozhatunk létre saját osztályt ennek érdekében.

A kódunk a következőképpen néz ki:

```
public class KapuSzkenner {  
  
    public static void main(String[] args) {  
  
        for(int i=0; i<1024; ++i)  
  
            try {  
  
                java.net.Socket socket = new java.net.Socket(args[0], i);  
  
                System.out.println(i + " figyeli");  
  
                socket.close();  
  
            } catch (Exception e) {  
  
                System.out.println(i + " nem figyeli");  
  
            }  
    }  
}
```

A program futtatása során annyi történik, hogy a prancssori argumentumként kapott számítógép 1024-nél kisebb portjaihoz próbálunk kapcsolódni. Java-ban a kapcsolódáshoz egy `Socket` típusú objektumra van szükségünk. Ennek a konstruktora meg kell adni az IP-címet és a portot, ahova csatlakozni szeretnénk. Viszont adódik az a kérdés, hogy mi van akkor ha nem tudunk csatlakozni a megadott porthoz? Abban az esetben a `socket` objektumunk konstruktora `IOException` típusú kivételt dob. Ha tudunk csatlakozni, akkor a cél gép adott portján egy szerver folyamat üzemel, ellenkező esetben pedig nem. Mivel dobhat kivételt a programunk, ezért azt le kell kezelnünk, vagyis berakjuk egy `try-catch` blokkba. Ha nem dob hibát a csatlakozás során a program, akkor kiírjuk a terminálba, hogy az adott port figyelt, hiba esetén pedig azt, hogy nem figyelt.

19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat megoldása során azzal fogunk megismerkedni, hogy az AspectJ programozási nyelv használatával, hogy tudjuk azt elérni, hogy egy régi programunkban ha szeretnénk bármilyen változást eszközölni, akkor, hogy tehetjük azt meg úgy, hogy nem nyúlunk bele a forráskódba. Helyette írunk egy szöveges (aspect-et) mely megoldja a módosítást.

Mielőtt a kódot megnéznénk pár dolgot érdemes említeni az aspektusok felépítéséről. Ezek nem használhatók önnálóan ezért szükséges minden hozzá egy program. Ha ezzel megvagyunk, akkor létrehozunk egy aspektust ami nagyon hasonlít egy Java osztályra. Viszont a létrehozás előtt érdemes három fogalommal megismерkedni. Az egyik a **kapcsolódási pont**, mely az eredeti programban van, vagyis az eredeti program egyik függvénye. A másik fontos fogalom a **vágási pont**, mely az aspektus része, és a csatlakozási pontokat jelöljük vele. Végül még szükség van **tanácsra**, mely lényegében azt tartalmazza, hogy hogyan szeretnénk módosítani az eredeti program működését.

Azt fogjuk megoldani, hogy a programunk a fát preorder módon járja be. Az Aspect.j fájl tartalma a következő.

```
privileged aspect Aspect{
    void around(LZWBInFa fa, LZWBInFa.Csomopont elem, java.io.←
    BufferedWriter os):
        call(public void LZWBInFa.kiir(LZWBInFa.Csomopont, java.io.←
            BufferedWriter))
            && target(fa) && args(elem, os) {
    if (elem != null)
    {
        try{
            ++fa.melyseg;
            for (int i = 0; i < fa.melyseg; ++i)
                os.write("---");
            os.write(elem.getBetu () + "(" + (fa.melyseg - 1) + ")");
            fa.kiir(elem.egyesGyermek (), os);
            fa.kiir(elem.nullasGyermek (), os);
            --fa.melyseg;
        }
        catch(java.io.IOException e){
            System.out.println("Csomópont írása nem sikerült.");
        }
    }
}
}
```

Az első fontos észrevétel, hogy szerepel a privileged mely azt szolgálja, hogy az aspektusunk hozzá tudjon férni az osztályok privát tagjaihoz. Az aspect szó meg azt jelöli, hogy most aspektust írunk, nem pedig hagyományos osztályt. Az aspektusunk egyetlen függvényből áll, ami az around. Ahhoz, hogy a bejárás módját módosítani tudjuk elsőnek a kiir(Csomopont, BufferedWriter) függvényre van szükségünk. Mivel ez a függvény olyan tagváltozókat és tagfüggvényeket tartalmaz, melyek nem statikusak, ezért az around függvény paraméterének meg kell adnunk egy LZWBInFa objektumot, melyen keresztül ezeket a tagokat el tudjuk érni. Majd megadjuk, hogy az around melyik függvény helyett hívódjon meg, ehhez meg kell adnunk a teljes paraméterlistáját, mivel enélkül a fordító nem tudná egyértelműen azonosítani, melyikre gondolunk. Ezután && elválasztva megadjuk, hogy az around paraméterei közül melyiket szeretnénk paraméterként átadni a kiir függvénynek, és azt is, hogy melyik LZWBInFa objektumra szeretnénk végrehajtani a függvényt. Az előbbihez a args, utóbbihoz a target kulcsszót használjuk. Ezután következik az around függvény törzse, mely a már ismert kód részletet tartalmazza.

Program futtatása



```
sudo apt install aspectj
ajc LZWBinFa.java Aspect.aj
java -cp /usr/share/java/aspectjrt.jar:. LZWBinFa input.txt -o ↵
output.txt
```

A alap programunk a következő kimenetet adja a `input.txt`-ben található szövegre:

```
-----1 (2)
-----0 (3)
-----1 (5)
-----0 (4)
----1 (1)
----0 (2)
----0 (3)
-----0 (4)
---/(0)
-----1 (2)
----0 (1)
----0 (2)
-----0 (3)
depth = 5
mean = 3.5
var = 1.2909944487358056
```

Aspektus használatával pedig a következőt:

```
---/(0)
----1 (1)
-----1 (2)
-----0 (3)
-----0 (4)
-----1 (5)
-----0 (2)
-----0 (3)
-----0 (4)
----0 (1)
-----1 (2)
-----0 (2)
-----0 (3)
depth = 5
mean = 3.5
var = 1.2909944487358056
```

Az aspektusok legnagyobb előnye az, hogy nem szükséges komolyabban belenyúlnunk a régi kódunkba ahhoz, hogy módosítsunk a működésén.

19.3. Android Játék

Írunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Helló, Android!” feladatára!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.4. JUnit teszt

A https://progater.blog.hu/2011/03/05/labormeres_othton_avagy_hogyan_dolgozok_fel_egy_pedat poszt kézzel számított mélységet és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A JUnit a Java nyelvhez kifejlesztett egységesztelő keretrendszer. Segítségével automatizált módon tudjuk tesztelni, hogy a programunk a várt módon működik-e. Azt fogjuk megnézni, hogy a szórás és az átlag értékek megfelelnek-e annak, amit korábban láttunk. A NetBeans-ben létre kell hoznunk egy projektet, amibe bekerakjuk a LZWBInFa osztály forrását, majd ha erre rákattintunk a Tools menüpontban létrehozunk egy tesztet. Ha elkészítettük a tesztet, akkor a "Test Libraries"-hez hozzá kell adni a Junit 4.x könyvtárát. Majd megnyítjuk újra a projektet. Ezután pedig a forrásunkra kattintva jobb egér gombbal lesz egy olyan opción, hogy "Test file". Ha minden okés, akkor lefut a teszt hiba nélkül.

Teszthez tartozó forrás a következő:

```
import static junit.framework.TestCase.assertEquals;
import org.junit.Test;

public class LZWBInFaTest {
    LZWBInFa binfa = new LZWBInFa();

    @Test
    public void testHozzarendel() {
        char[] minta = "01111001001000111".toCharArray();

        for(char betu: minta ){
            binfa.hozzarendel(betu);
        }

        assertEquals(4, binfa.getMelyseg());
        assertEquals(2.75, binfa.getAtlag(), 0.01);
        assertEquals(0.957427, binfa.getSzoras(), 0.000001);
    }
}
```

Szintaktikailag egy normál Java osztályt látunk. A különlegességét az adja, hogy tartalmazza a @Test az egy annotációt, melyel jelöljük, hogy ezután egy tesztelő függvényt definiálunk. Az LZWBInFaTest osztályban példányosítunk egy LZWBInFa típusú objektumot. Majd a testHozzarendel függvény segítségével teszteljük, hogy a hozzarendel függvény megfelelően épít-e fel a fát. A minta tömbben eltároljuk a honlapon található mintát, majd maszkolás nélkül belerakjuk az 1-eseket és a 0-ákat a fába. Ha ezzel végeztünk, akkor a assertEquals függvény segítségével ellenőrizzük, hogy azokat a szórás, mélység és átlag értékeket kaptuk, amire számítottunk. A forrásban ennek a függvénynek két verzióját használtuk. Az első esetben sima összehasonlítást végzünk, a többi esetben pedig megadunk egy delta számot, amivel elérhet a várt érték a kapott értéktől. Erre azért van szükség, mert nem egész számokat hasonlítunk össze. Valójában minden programban, ahol double/float típusú változókat hasonlítunk össze, ott szükséges lenne minden megadni egy tetszőlegesen kicsi delta számot, aminél ha kisebb az eltérés a két szám között, akkor egyenlőnek tartjuk őket.

19.5. OSCI

Készíts egyszerű C++/OpenGL-es megjelenítőt, amiben egy kocsit irányítasz az úton.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.6. OSCI2

Készíts egyszerű C++/OpenGL-es megjelenítőt, amiben egy kocsit irányítasz az úton. A kocsi állapotát minden pillanatban mentsd le. Ezeket add át egy Prolog programnak, ami egyszerű reflex ágensként adjon vezérlést a kocsinak, hasonlítsd össze a kézi és a Prolog-os vezérlést. Módosítsd úgy a programodat, hogy ne csak kézzel lehessen vezérelni a kocsit, hanem a Prolog reflex ágens vezérelje!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.7. OSCI3

Készíts egy OSM utakat megjelenítő C++/OpenGL-es progit!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.8. EPAM: DI

Implementálj egy alap DI (Dependency Injection) keretrendszer Java-ban annotációk és reflexió használatával megvalósítva az IoC-t (Inversion Of Control).

19.9. EPAM: JSON szerelzació

Implementálj egy JSON szerelzaciós könyvtárat, mely képes kezelni sztringeket, számokat, listákat és beágyazott objektumokat. A megoldás meg kell feleljen az összes adott unit tesztnek.

19.10. EPAM: Kivételek kezelés

Adott az alábbi kódrészlet. Mi történik, ha az input változó 1F, "string" vagy pedig null? Meghívódik e minden esetben a finally ág? Válaszd indokolt!

Floatnál nem hívódik mert elsőnek alkapjuk a floatot visszaadunk egy child exceptiont amit elkap a catch childeception és visszaadja a parentexceptionjet amit elkap a kovetkező catch ami kiírja hogy elkapta és exit-el

Stringnél sem hívódik meg alkapjuk visszaadunk egy parentexceptiont amit instant elkap a parentexceptionos catch és kilép

Nullnál meghívódik mert elsőnek kapunk egy runtimeexceptiont belőle amit csak a exception fog elkapni és utánna lefutódik a finally

20. fejezet

Helló, Calvin!

20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13-hello_samu_a_tensorflow-bol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Megoldás video:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az MNIST egy olyan program mely egy előre elkészült kézzel rajzolt számok képadatbázisból megtanítja magát hogy felismerje azokat. A program python nyelven íródott és a tensorflowra könyvtárhoz is szükségeünk van Mi is az a tensorflow? Google Brain team által létrehozott library melyben machine learning és deep learning modelleket és algoritmusokat találunk.

Elsőnek importáljuk a tanításhoz szükséges mintákat adatbázist.

```
from tensorflow.examples.tutorials.mnist import input_data
```

Elsőnek deklarálunk egy függvény mely majd beolvassa a custom képünket. A file abban adhatjuk meg a custom képünk nevét. Majd a beadott képet dekódoljuk uint8 vagy uint16 ra.

```
def readimg():
    file = tf.io.read_file("egytizes.png")
    img = tf.image.decode_png(file, channels=1)
    return img
```

Most olvassuk be a képeket az x nem egy konkrét érték lesz hanem egy placeholder jelen esetben a mi modellünk 2D-s lesz és 784 dimenziós vektorú. A w az egy variable típusú ami csak annyit jelenet hogy módosítható tensor és ez tárolja a súlyokat. b is variable típusú csak ebben az eltolási értéket tároljuk. y-ba összeszorozzuk a x w matrixokat (784x10) és hozzáadjuk a b-t így kapunk egy 10 dimenziójú vektort.

```
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
```

Az y-ba fogjuk tárolni az általunk kiszámított eloszlásokat, míg az y_-ban a valódi eloszlást.

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

Ezután szükségünk lesz egy indicátorra hogy tesztelni tudjuk a modellt hogy mennyire rossz a célunk az hogy ezt minimalizáljuk. A tf.nn.softmax_cross_entropy_with_logits kiszámolja a cross entropyt y y_- között. A tf.reduce_mean meg kiszámolja az átlagot. tf.train.GradientDescentOptimizer használjuk hogy csökkentük az átlagos veszteséget 0.5 benne annyit adjatunk meg mekkora egységekkel közelítsük meg jelen esetben 0.5.

```
cross_entropy = tf.reduce_mean(tf.nn. ←
    softmax_cross_entropy_with_logits(logits=y, labels=y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize( ←
    cross_entropy)
```

Itt inicializálja változóinkat és elinítünk egy session-t.

```
sess = tf.InteractiveSession()
tf.initialize_all_variables().run(session=sess)
```

Itt folyik maga a tanítás 1000x fut el és minden alkalommal kiválaszt random 100 elemet az adatbázisból. Itt újra y y kiszámítjuk értékeket majd ezzel kiszámoljuk a átlagos veszegéget és minimalizáljuk.

```
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, "%")
```

Most hogy a tanítással megvagyunk teszteljük. tf.argmax megadja egy adott tengelyen a legnagyobb indexet és ha ez 2 megegyezik akkor helyesen tippeltünk. A tf.cast átalakítja az igazságértékünket float 32-re. Ezután kiszámljuk mennyire pontos a model.

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

20.2. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a <https://arato.inf.unideb.hu/batfai.norbert/NEMESPOR/DE/denbatfai2.pdf> 8. fóliáját!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Sima mnistel-el 90-92% pontosságokat kaptunk ami papíron elég jól néz ki de a valóságban ez az eredmény gyenge. Feladatunk ugyanaz mint az előzőbe csak a deep mnistbe kell megvalósítani ami annyit jelent hogy pontosabb értékek létrehozásáért neurális hálókat is behozunk a programunkba.

Létrehozzuk deepnn-t ahol a képek átalakítjuk.

```
def deepnn(x):
    with tf.name_scope('reshape'):
        x_image = tf.reshape(x, [-1, 28, 28, 1])
```

W_conv1 fogja tartalmazni a súlyokat melynek az alakja [5, 5, 1, 32] ebben az első kettő a kép alakját írja le a harmadik a bemeneti streamet 4. pedig ki streamet. Így azt kaptuk hogy minden 5x5 képhez 32 db összeget fog kiszámítani. b_conv1 tárolunk 32 bias-t. Majd összeszorozzuk az értékeinket, A tf.nn.relu az pedig ha negatív értéke van 0 ad vissza ha pedig pozitív akkor magát a számot.

```
with tf.name_scope('conv1'):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

    with tf.name_scope('pool1'):
        h_pool1 = max_pool_2x2(h_conv1)
```

Mivel neurális hálót készítünk több mint egy rétegre lesz szükségünk. A második réteget az elsőből számoljuk ki. Itt már a súlyunk 32 bemenetet kap és 64-eset ad vissza. Szóval itt már minden 5x5 képhez 64 összeget számol ki.

```
with tf.name_scope('conv2'):
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
    with tf.name_scope('pool2'):
        h_pool2 = max_pool_2x2(h_conv2)
```

Ezután egy 1024 neuronból álló fc réteget adunk hozzá a modelhez itt fel is dolgozzuk az egész képet. Kiszámoljuk a súlyokat és a biasokat majd a tf.reshape függvényel egy mátrixá alakítjuk a h.pool2-től. Majd ezt a mátrixot megszorozzuk a w_fc1 mátrixal és hozzáadjuk a b_fc1 mátrixot majd odaadjuk a tf.nn.relu-nak

```
with tf.name_scope('fc1'):
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

Itt a fölösleges neuronokat eldobjuk és létrehozunk egy placeholder-t.

```
with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Ez lesz szoftmax réteg

```
with tf.name_scope('fc2'):
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])

    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
    return y_conv, keep_prob
```

A conv2d 4D-s inputból és szűrőbből egy 2D-s konvolúciót számol ki és visszaadja.

```
def conv2d(x, W):
    """conv2d returns a 2d convolution layer with full stride ←
    ."""
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='←
    SAME')

def max_pool_2x2(x):
    """max_pool_2x2 downsamples a feature map by 2X."""
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1], padding='SAME')
```

Itt súly számolunk az alakból.A bias generálunk a shapeből.És az öreg readimg-be meg beolvassuk a saját képünket.

```
def weight_variable(shape):
    """weight_variable generates a weight variable of a given shape."""
    initial = tf.compat.v1.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    """bias_variable generates a bias variable of a given shape."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def readimg():
    file = tf.compat.v1.read_file("asd4.png")
    img = tf.image.decode_png(file, 1)
    return img
```

20.3. CIFAR-10

Az alap feladat megoldása, +saját fotót is ismerjen fel, https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tf_tutorial_peldabol

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.4. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Linkek: [Developer Optinok bekapsolása](#), [Android Studio](#), [tensorflow repó](#).

Előfeltételek:

- Kell az Android Studio
- Kell egy androidos készülék és fejlesztői környezet melyek min api 21-esek
- Android Studionak 3.2 vagy újabb verziójúnak kell lenni

Építése:

- 1. Nyissuk meg az android studiot és válasszuk ki a Open an existing Android Studio project
- 2. válasszuk ki azt a mappát ahova leclonoltuk a TensorFlow Lite sample GitHub repót
- 3. Ha megkérdezni akarunk e Gradle Sync nyumjunk ok-ra
- 4. Ha hibákat ad ki hiányoznak key könyvtárak csak telepitsük azokat
- 5. Csatlakoztassuk a telefonunkat a számítógéphez(készüléken be kell kapcsolni a developer optiont)
- 6. Futassuk :)

20.5. SMNIST for Machines

Készíts saját modellt, vagy használj meglévőt, lásd: <https://arxiv.org/abs/1906.12213>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.6. Minecraft MALMO-s példa

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lásd pl.:

<https://youtu.be/bAPSu3Rndi8>,

https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innentol_mi,

https://bhaxor.blog.hu/2018/10/28/minecraft_steve_szemuvege

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Minecraft MALMO egy minecraft amibe lehet python c, c++, és java kóddal irányítani a karaktert es alakítani a világot főleg AI fejlesztésre használják. Ezekben kívül készíthetünk mission-öket az elobb felsorolt programozási nyelveken.

Windows-on való telepítése:

Először 3 dolog kell: JDK 8.0 telepítése ,Python 3.6 telepít, zip letöltése és kicsomagolása

2 módszer van a letöltésre

Githubról letölthet az egész mappát

Vagy powershellbe bizonyos parancsokkal

2-es módszernél telepítéséhez ezeket kell beírni

```
Set-ExecutionPolicy -Scope CurrentUser Unrestricted | ↵
    Engedélyezzük a powershellnek külsős progok telepítését
    cd ahova kicsomagoltad\Malmo-0.35.6-Windows-64bit_Python3.6\ ↵
        scripts
        .\malmo_install.ps1
```

Ezzel kész is minecraft elindításához ezt kell beírni

```
cd ahola ki lett csomagolva\Malmo-0.37.0-Windows-64 ↵
    bit_withBoost_Python3.6\Minecraft
    .\launchclient
```

Ezzel el is indul a minecraft nem kell várni hogy a számláló 100% üssön 95% már indul és sose éri el a 100-at

És mondjuk egy missiót akarunk indítani python nyelvben

```
cd ahola ki lett csomagolva\Malmo-0.37.0-Windows-64 ↵
    bit_withBoost_Python3.6\Python_Examples
    python tutorial_1.py
```

Ezzel letesz minket a játék egy szerver alapú világba és 10 másodpercig egyhelyben állunk ennél vannak izgalmasabb küldetések is de csak eddig jutottam

Pár alap parancs

```
!! az 1 az truera állítja így folyamatosan csinálja a 0 falsera ←  
azzal leállítjuk.  
agent_host.sendCommand("turn -0.5") | fordulás  
agent_host.sendCommand("move 1") | előre mozgás  
agent_host.sendCommand("jump 1") | ugrás  
agent_host.sendCommand("pitch 1") | sötétség  
time.sleep(1) | alvás  
agent_host.sendCommand("attack 1") | támadás
```

20.7. EPAM: Reaktív programozás

Számoljuk ki az első 10 nem negatív egész szám összegét és átlagát.

20.8. EPAM: Back To The Future

Adott az alábbi kódrészlet:

```
public class FutureChainingExercise {  
    private static ExecutorService executorService = Executors. ←  
        newFixedThreadPool(2);  
    public static void main(String[] args) {  
        CompletableFuture<String> longTask  
        = CompletableFuture.supplyAsync(() -> {  
            sleep(1000);  
            return "Hello";  
        }, executorService);  
        CompletableFuture<String> shortTask  
        = CompletableFuture.supplyAsync(() -> {  
            sleep(500);  
            return "Hi";  
        }, executorService);  
        CompletableFuture<String> mediumTask  
        = CompletableFuture.supplyAsync(() -> {  
            sleep(750);  
            return "Hey";  
        }, executorService);  
        CompletableFuture<String> result  
        = longTask.applyToEitherAsync(shortTask, String::toUpperCase, ←  
            executorService);  
        result = result.thenApply(s -> s + " World");
```

```
CompletableFuture<Void> extraLongTask
= CompletableFuture.supplyAsync(() -> {
sleep(1500);
return null;
}, executorService);
result = result.thenCombineAsync(mediumTask, (s1, s2) -> s2 + ", " +
s1, executorService);
System.out.println(result.getNow("Bye"));
sleep(1500);
System.out.println(result.getNow("Bye"));
result.runAfterBothAsync(extraLongTask, () ->
System.out.println("After both!"), executorService);
result.whenCompleteAsync((s, throwable) -> System.out.println("Complete: " +
s), executorService);
executorService.shutdown();
}
/**
*
* @param sleeptime sleep time in milliseconds
*/
private static void sleep(int sleeptime) {...}
```

1

Mi lesz kiíratva a standard kimenetre és miért?

20.9. EPAM: AOP

Készíts két példa projektet, melyben egyes metódusok futási idejét méred majd kiíratod úgy, hogy a metódus futási idejének méréséhez AOP-t használsz. Az első projektben csak az AspectJ könyvtárat, a második esetében pedig Spring AOP-t használj.

IV. rész

Irodalomjegyzék

DRAFT

20.10. Általános

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

20.11. C

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

20.12. C++

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

20.13. 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.