

Programozás könnyüszerrel

**veszélyes könyv, még megtanulhatsz
programozni, ha nem vigyázol!**

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Programozás könnyüszerrel		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Tóth, Balázs	2019. december 11.	

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	Aktualizálás, javítások.	nbatfai
0.1.0	2019-02-26	Feladatok elkezdése, chapter 1	Balázs Tóth
0.1.1	2019-03-03	Feladatok elkezdése, chapter 2	Balázs Tóth
0.1.2	2019-03-11	Feladatok elkezdése, chapter 3	Balázs Tóth
0.1.3	2019-03-18	Feladatok elkezdése, chapter 4	Balázs Tóth

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.1.4	2019-03-27	Feladatok elkezdése, chapter 5	Balázs Tóth
0.1.5	2019-04-05	Feladatok elkezdése, chapter 6	Balázs Tóth
0.1.6	2019-04-12	Feladatok elkezdése, chapter 7	Balázs Tóth
0.1.7	2019-04-19	Feladatok elkezdése, chapter 8 és olvasónapló	Balázs Tóth
0.1.8	2019-04-26	Feladatok elkezdése, chapter 9	Balázs Tóth
0.1.9	2019-04-30	Feladatok elkezdése, chapter 10	Balázs Tóth
0.2.0	2019-05-05	Feladatok finomítása, bejelezés	Balázs Tóth
1.0.1	2019-09-09	Második felvonás elkezdése	Balázs Tóth
1.0.2	2019-09-11	Hello Berners beillesztése, arroway inicializálása	Balázs Tóth
1.0.3	2019-09-14	Java és C++ könyvek összehasonlításának elkezdése/első oldal megírása	Balázs Tóth
1.0.4	2019-09-16	összehasonlítás folytatása, python napló elkezdése	Balázs Tóth
1.0.5	2019-09-24	Arroway chapter befejezése	Balázs Tóth
1.0.6	2019-09-25	Liskov chapter elkezdése	Balázs Tóth

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0.7	2019-10-2	Liskov chapter befejezése	Balázs Tóth
1.0.8	2019-10-5	Mandelbrot chapter elkezdése	Balázs Tóth
1.0.9	2019-10-8	Mandelbrot chapter befejezése	Balázs Tóth
1.1.0	2019-10-13	chomsky chapter elkezdése	Balázs Tóth
1.1.1	2019-10-15	chomsky chapter befejezése	Balázs Tóth
1.1.2	2019-10-16	Gödel és Stroustrup chapterek elkezdése	Balázs Tóth
1.1.3	2019-10-26	Gödel és Stroustrup chapterek befejezése	Balázs Tóth
1.1.4	2019-11-1	név nélküli chapter inicializálása	Balázs Tóth
1.1.5	2019-11-3	név nélküli chapter kész	Balázs Tóth
1.1.6	2019-11-6	Schwarz és Calvin chapterek inicializálása	Balázs Tóth
1.1.7	2019-11-1	schwarz chapter elkezdése	Balázs Tóth
1.1.8	2019-11-7	schwarz chapter vége	Balázs Tóth
1.1.9	2019-11-15	calvin chapter elkezdése	Balázs Tóth
1.2.0	2019-11-26	calvin chapter befejezése	Balázs Tóth

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.2.1	2019-11-27	chapterek átnézése	Balázs Tóth

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]

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	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun téTEL	14
2.8. A Monty Hall probléma	15
3. Helló, Chomsky!	17
3.1. Decimálisból unárisba átváltó Turing gép	17
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	17
3.3. Hivatalos nyelv	18
3.4. Saját lexikális elemző	19
3.5. Leetspeak	20
3.6. A források olvasása	22
3.7. Logikus	23
3.8. Deklaráció	24

4. Helló, Caesar!	28
4.1. double ** háromszögmátrix	28
4.2. C EXOR titkosító	31
4.3. Java EXOR titkosító	32
4.4. C EXOR törő	34
4.5. Neurális OR, AND és EXOR kapu	39
4.6. Hiba-visszaterjesztéses perceptron	42
5. Helló, Mandelbrot!	43
5.1. A Mandelbrot halmaz	43
5.2. A Mandelbrot halmaz a std::complex osztállyal	44
5.3. Biomorfok	48
5.4. A Mandelbrot halmaz CUDA megvalósítása	52
5.5. Mandelbrot nagyító és utazó C++ nyelven	52
5.6. Mandelbrot nagyító és utazó Java nyelven	53
6. Helló, Welch!	57
6.1. Első osztályom	57
6.2. LZW	59
6.3. Fabejárás	66
6.4. Tag a gyökér	67
6.5. Mutató a gyökér	68
6.6. Mozgató szemantika	69
7. Helló, Conway!	74
7.1. Hangyaszimulációk	74
7.2. Java életjáték	77
7.3. Qt C++ életjáték	83
7.4. BrainB Benchmark	86
8. Helló, Gutenberg!	88
8.1. Programzási alapfogalmak	88
8.2. Programozás bevezetés	89
8.3. Programozás	90

9. Helló, Schwarzenegger!	92
9.1. Szoftmax Py MNIST	92
9.2. Mély MNIST	95
9.3. Minecraft-MALMÖ	100
10. Helló, Chaitin!	101
10.1. Iteratív és rekurzív faktoriális Lisp-ben	101
10.2. Gimp Scheme Script-fu: króm effekt	101
10.3. Gimp Scheme Script-fu: név mandala	101
III. Második felvonás	102
11. Helló, Berners-Lee!	104
11.1. Olvasónaplók	104
12. Helló, Arroway!	107
12.1. OO szemlélet	107
12.2. "Gagyi"	109
12.3. Yoda	110
12.4. Kódolás from scratch	111
13. Helló, Liskov!	113
13.1. Liskov helyettesítés sértése	113
13.2. Szülő-gyerek	115
13.3. Anti OO	116
13.4. Ciklomatikus komplexitás	117
14. Helló, Mandelbrot!	122
14.1. Reverse engineering UML osztálydiagram	122
14.2. Forward engineering UML osztálydiagram	123
14.3. BPMN	124
14.4. Egy Esettan	125
15. Helló, Chomsky!	127
15.1. Encoding	127
15.2. Full screen(ződ)	128
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	131
15.4. I334d1c4 ⁵ (ződ)	133

16. Helló, Stroustrup!	136
16.1. JDK osztályok	136
16.2. Másoló-mozgató szemantika	138
16.3. Hibásan implementált RSA törése	139
17. Helló, Gödel!	143
17.1. C++11 Custom Allocator	143
17.2. STL map érték szerinti rendezése	145
17.3. GIMP Scheme hack	147
18. Helló, !	149
18.1. FUTURE tevékenység editor	149
18.2. SamuCam	153
18.3. BrainB	155
19. Helló, Schwarzenegger!	158
19.1. Port scan	158
19.2. AOP	160
19.3. Android Játék	161
20. Helló, Calvin!	166
20.1. MNIST	166
20.2. Android telefonra a TF objektum detektálója	168
20.3. Minecraft MALMO-s példa	172
21. Helló, védés!	177
21.1. BrainB védés	177
IV. Irodalomjegyzék	184
21.2. Általános	185
21.3. C	185
21.4. C++	185
21.5. Lisp	185

Ábrák jegyzéke

3.1. Az átváltó Turing gép	17
3.2. A környezetfüggő grammatikák	18
4.1. A double ** háromszögmátrix a memóriában	30
5.1. A Mandelbrot halmaz a komplex síkon	43
7.1. A hangyaszimuláció UML diagram	74
7.2. A hangyák akcióban	75
7.3. Az életjáték futás közben	84
12.1. A JDK kód	107
12.2. A gagyi futás	110
12.3. A yoda futás	111
12.4. A piBBP futás	112
13.1. A szülőgyerek futás	116
13.2. A 4 nyelv futás	117
13.3. A 4 nyelv futás	117
13.4. A ciklomatikus ábra	120
13.5. A lizard ciklomatikus számolás	121
14.1. Az LZW UML	122
14.2. Az új UML	123
14.3. Az új UML	123
14.4. A BPMN ábra	124
15.1. A mandel futás	127
15.2. A fullscreen futás	131
15.3. A fullscreen futás	132

15.4. A fullscreen futás	135
16.1. A jdk futás	136
16.2. Az rsa futás	141
16.3. Az rsa futás	141
16.4. Az rsa futás	142
17.1. Az allokátor futás	144
17.2. Az allokátor2 futás	145
17.3. Az stl futás	147
18.1. Az editor futás	149
18.2. Az editor javított futása	150
18.3. A javafx	151
18.4. A samu futás	155
18.5. A brainb futás	156
19.1. A socket futás	158
19.2. A socket Exception	159
19.3. A socket eredménye	159
19.4. A szövés eredménye	160
19.5. A szövés függvényei	160
19.6. A játék proto	161
19.7. A játék proto	163
19.8. A játék proto2	164
19.9. A játék reset	165
20.1. A pip telepítés	166
20.2. Az mnist futása	168
20.3. A kettes	168
20.4. A tensor futása	169
20.5. A tensor futása2	170
20.6. A tensor futása3	171
20.7. A tensor exception	172
20.8. A malmo kliens futás	173
20.9. A malmo future modul	173

20.10A malmo kliens futtatása	174
20.11A python forráskód futás	174
20.12egy malmo példa	175
20.13A második malmo példa	175
20.14A malmo saját kód alapja	176
 21.1. A Brainb ablak	177
21.2. A Brainb slotok	178
21.3. A Brainb bill.funkciók	183

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 gyerekeknek, 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 gyerekeknek, 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 másit is) példával.

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

- Olvasgasd 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
- [KERNIGHANRITCHIE]
- [BMECPP]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a Monty Hall probléma bemutatása.

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

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://github.com/ghjbku/DE/blob/master/video.flv>

Megoldás forrása:

Elsőként a 100%-os végtelen ciklust készítettem el, hiszen ezt volt a legegyszerűbb megírni. Amint lát-hatjuk elég egyszerűen meg lehet oldani, hogy a cpu 100%-ban dolgozzon a program futása alatt. Itt én a WHILE ciklus-t választottam, de FOR-ral is hasonlóképpen lehet megvalósítani a végtelenítést. Az egész program lényege egyetlen értéken alapszik, amit az `*asd*` változó hordoz. Mivel ez a változó semmiképp sem kap 1-et értékül, a program soha sem fog kilépni a ciklusból.

```
//100%-ban megdolgoztat egy magot
//lefordítás: gcc forrásnév -o késznév
#include <stdio.h>
int main()
{
int asd =0;
while (asd=1) {}
return 0;
}
```

A következő program a 0%-os végtelen ciklus volt. Ha ismerjük az API-t, vagy tapraesettek vagyunk a google-n való keresést illetően, akkor itt is egyszerű dolgunk volt. Amint azt észrevehetünk, a programkód nagyon hasonlít az előző kódra, csupán annyi változás történt, hogy a ciklus belsejében megjelent egy függvény, a `*sleep()*`. Ez a függvény annyi milisecond-ig állítja meg a programot, amely számot a két zárójel közé írtunk. Jelen esetben ez `*1*`, de mivel egy végtelen ciklusban vagyunk, ezért végtelen sokszor vár majd 1 milisec-ot a program, így tehát nem használ erőforrást.

```
//0%-os cpu használat
//lefordítás: gcc forrásnév -o késznév
#include <stdio.h>
```

```
int main()
{
int asd =0;
while (asd=1)
{
sleep(1);
}
return 0;
}
```

Utolsóként pedig jön a "legnehezebb", minden magot 100%-on futtatni. Az igazat megvallva, ez sem valami nagy ördöngősséggel, itt is csak egy pár dolog változott a legelső programhoz képest. A legfontosabb dolog ez a sor `*#include "omp.h"*`, ez a header fájl előfeltétele annak, hogy a `*#pragma omp parallel*` kódot értelmezni tudja a fordítóprogram. A `*#pragma...*` sor veszi rá a programunkat, hogy párhuzamos módon, az összes magon futtassa a programot a számítógép.

```
// minden mag 100%-on fut
// lefordítás: gcc -fopenmp forrásnév -o késznév
#include <stdio.h>
#include <unistd.h>
#include "omp.h"
int main () {
int asd=0;

#pragma omp parallel
    while(asd=1)
    {
    }
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Amint láthattuk, elég egyszerű dolgunk volt ezen programok megírását illetően, viszont ez nem azt jelenti, hogy félválról vehetjük a programozást, hiszen kevés olyan program létezik, aminek valamilyen hétköznapi haszna van, és mégis ilyen egyszerű lenne megírni. Ezen programkódok csak az egyszerűbb megértést segítik elő, gyakorlati hasznuk sajnos nincs.

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

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

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a Lefagy függvényt, amely tetszőleges programról el tudja döntenи, hogy van-e benne vlgó ciklus:

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

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

}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

Amint láthatjuk, lehetetlen olyan programot írni, amely egy másik programról eldönntené, hogy az le fog-e fagyni, vagy sem.

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:

Ezen feladat megoldása igencsak egyszerűnek bizonyult. Na persze nem annyira egyszerű, mint egy végtelen ciklus megírása, de közel azonos szinten mozognak. A lentebb lévő forráskód elég egyszerűen értelmezhető, ezért hát nem megyek bele részletesen, csak a nagyon fontos dolgokat mondjam el. A *C* nyelvben a változók értékét egy parancsal tudjuk hozzáfűzni egy printf függvényhez, attól függően, hogy milyen típusú adatot hordoz a változó. Esetünkben minden két változó *szám/Digit* típust hordoz, ezért a kód, amivel meghívjuk a behelyettesítő paramétert, ez lesz: * %d *, majd ha végeztünk a kiírni kívánt szöveggel, egy vesszővel jelezzük a fordítóprogramnak, hogy most a behelyettesítendő változók következnek. A kódban megjelenik egy másik kód is, ami ismeretlen lehet az olvasó számára, ez a * \n *, amely annyit tesz, hogy új sorba kezdi az *n* után beírt szöveget, és a szóközt is értelmezi!

```
#include <stdio.h>
int main()
{
    int a=5,b=3;
    printf("A value = %d\n",a);
    printf("B value = %d\n",b);
    b=b-a;
    a=a+b;
```

```
b=a-b;  
printf("A value = %d\n",a);  
printf("B value = %d\n",b);  
return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

Ne tévesszen meg bennünket a feladat komplexitása, ha nem gondolunk bele, hogy pontosan hogyan is kellene segédváltozó nélkül elérni céljainkat, elég sokáig el tudunk időzni ezen az egyszerű feladaton. Tehát próbálunk meg minden feladatot úgy kezdeni, hogy elgondolkozunk azon, hogyan tudnánk megvalósítani a feladatban megírtakat.

2.4. Labdapattogás

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

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

Megoldás forrása:

A feladat nehézségi szintjét tekintve már egy magassabb szinten van, vagyis inkább, gondolkodást igényel. Ebben a forráskódban már megjelenik egy pár új parancs, melyeket eddig még nem láttunk. Kezdve az új Header fájllal, a `<math.h>` fájllal, amely a matematikai függvényekért felel és minden értéket double típussal kezel(double típust kér, és azt ad vissza), ilyen függvény például az `*abs*`, amely az abszolút értéket jelöli, de ebben a header fájlban található a `*pow*` és az `*sqrt*` is, az előbbi a hatványozást, míg utóbbi a négyzetgyököt kezeli. Aztán ott van az a furcsa sor két sorral alább, az a bizonyos `*#define*`... ezeket a sorokat úgynevezett "Nevesített konstansok" definiálásánál használjuk. Ezek a konstansok értéket nem változtatnak a program futása során, és bármilyen értéket adhatunk nekik.

```
//Labdapattogás if nélkül (mentorálva Gila Attila által)  
#include<stdio.h>  
#include<math.h>  
  
#define szel 80  
#define mag 24  
  
int putX(x,y)  
{  
    int ix,iy;  
  
    for(ix=0;ix<x;ix++)  
        printf("\n");  
  
    for(iy=0;iy<y;iy++)  
        printf(" ");  
  
    printf("O\n");
```

```
return 0;
}

int main()
{
long int x=0,y=0;

while(1)
{
system("clear");
putX(abs(mag-(x++%(mag*2))),abs(szel-(y++%(szel*2)))); 
usleep(15000);
}

return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A labda "pattogása" egyszerű módon van megoldva, miszerint minden egyes "tick" után, amit a program a végtelen ciklusban tölt, a *system("clear")* parancs miatt a terminál jelenlegi tartalma törlődik, de mivel az túl gyorsan történik, mi csak úgy érezzük, hogy a labda szépen mozog az ablakban. A "tick" periódust az *usleep()* függvény zárójelében megadott szám határozza meg, a mértékegység microsecond. Viszont ha fontos a pontosság, akkor számolnunk kell a számítógép kalkulációs képességeivel, plusz az is időbe telik, hogy a program eljut az *usleep* függvényhez, ezután az egész program "alvó" állapotba kerül, kilép a processzor ütemezési sorából, és a delay attól is függhet, hogy a processzor maga mikor válassza újra a programot, miután a *usleep* függvény lefutott. Tehát ne lepődjünk meg, ha néhány ezer microsec-ot téved a program.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: <https://github.com/ghjbku/DE/blob/cpp/bitshift.cpp>

A feladat megoldása C++ nyelven történt, viszont C-ben is hasonló módon kell megoldani a problémát.

```
//a bitshift C nyelvben
#include <stdio.h>

int main(){

    unsigned int the_Bit = 1;
    int length = 0;
```

```
do
    length++;
while((the_Bit <= 1));

printf("A szóhossz mérete: %u\n", length);

return 0;
}
```

A C megoldás BogoMIPS-el: [itt található](#)

Tanulságok, tapasztalatok, magyarázat...

A bit méretét a **length** változó tárolja, amit úgy töltünk fel, hogy amíg a bit el nem éri a kezdési értéket, addig a ciklusban minig növeljük a változó értékét 1-el. Majd ezen értéket a végén kiírjuk. Az unsigned típus 2^n különböző értéket vehet fel 0 és n között.

2.6. Helló, Google!

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

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása C++-ban: <https://github.com/ghjbku/DE/blob/cpp/bearazas.cpp> és C-ben: <https://github.com/ghjbku/DE/blob/master/c%20cuccok/bearaz.c>

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

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

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

int main(void)
{
double L[4][4] = {
```

```
{0.0, 0.0, 1.0 / 3.0, 0.0},  
{1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},  
{0.0, 1.0 / 2.0, 0.0, 0.0},  
{0.0, 0.0, 1.0 / 3.0, 0.0}  
};  
  
double PR[4] = {0.0, 0.0, 0.0, 0.0};  
double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};  
  
long int i,j,h;  
i=0; j=0; h=5;  
  
for (;;) {  
    for (i=0;i<4;i++)  
        PR[i] = PRv[i];  
    for (i=0;i<4;i++) {  
        double temp=0;  
        for (j=0;j<4;j++)  
            temp+=L[i][j]*PR[j];  
        PRv[i]=temp;  
    }  
  
    if ( tavolsag(PR,PRv, 4) < 0.00001)  
        break;  
}  
kiir (PR,4);  
return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

A pagerank algoritmust a google fejlesztette ki azzal a céllal, hogy a weboldalak minőségét rangsorolja.
A feladat komplexitása miatt a soronkénti értelemezést választottam. Kezdjük is el.

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

Ez a függvény a minősítés végeredményét fogja kiírni. Egy egyszerű for ciklusból áll, amely a függvény-paraméterként megadott **db**-szor fog lefutni és kiírja az ugyancsak függvényparaméterből származó **tomb[]** tömb elemeit.

```
double tavolsag(double pagerank[],double pagerank_temp[],int db)  
{
```

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

Ez a következő függvény már bonyolultabb. A függvényünk két double típusú tömböt és egy számot kér paraméterül. A távolság kiszámítására itt is egy for-ciklus lesz segítségünkre, azon belül pedig egy abszolútérték függvény, amelyben a **pagerank** tömbből kivonjuk a **pagerank_temp** tömböt.

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

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

long int i, j, h;
i=0; j=0; h=5;
```

Amint láthatjuk, a main függvényen belül elkezdjük definiálni a változókat, melyeket az előbbi két függvényre majd ráeresztünk.

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

    if ( tavolsag(PR,PRv, 4) < 0.00001)
break;
}
kiir (PR,4);
return 0;
```

Továbbra is a main függvényben vagyunk. Már megtörtént a változó deklarálás, tehát elkezdődhet a rangsorolás. Belépünk egy for-ciklusba, majd azon belül rétegezve létrehozunk még 3 másik for-ciklust. Az első réteg a **PR** tömböt azonosítja a **PRv** tömbbel. A második réteg egy **temp** változóban összeszorozza

az **L[][]** kétdimenziós tömböt és az újonan kapott **PR** tömböt. Majd minden ciklus végén hozzáadja az új értékeket az előző értékhez. Ezután a **PRv** tömb értékéül adjuk a temp változót. ezután visszatérünk az első for-ciklusba, ahol pedig egy if elágazással megnézzük, hogy a **tavolsag()** függvény visszatérési értéke kisebb-e, mint 0.00001 . Ha igen, akkor kilép a ciklusból. A program a végén kiírja a **PR** tömb tartalmát a **kiir()** függvény segítségével.

2.7. 100 éves a Brun tétele

Írj R szimulációt a Brun tétele demonstrálására!

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R

```
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")
```

A Brun tétele az ikerprímekkel foglalkozik, és kimondja, hogy ezen prímek reciprokösszege egy véges értékhez konvergál, ún. Brun-konstans felé. jelölése: **B₂**

Mivel az R nyelvben még gyakorlatlanok vagyunk, így megint soronként fogok magyarázatot adni a forrás-kódra.

```
library(matlab)

stp <- function(x) {

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

A library(matlab) paranncsal meghívjuk a matlab külső fájlt, amelyben egyéb függvények mellett megtalálható a **primes()** függvény. Ez a függvény a paraméterként megadott számig kiszámolja a prímszámokat. R-ben a függvény létrehozása így történik:

```
függvénynév <- function(paraméter lista){függvény törzs return(visszatérési érték)}
```

az első sorban feltöltjük a primes változót a **primes(x)** függvény értékeivel. majd a második sorban a diff változóba belerakjuk nd> és a **primes[1:length(primes)-1]** vektorok különbségét.

a **primes[2:length(primes)]** vektor a primes 2. elemétől a változó hosszáig tartalmazza a számokat. Ezzel szemben a **primes[1:length(primes)-1]** rész a primes 1. elemétől az utolsó előtti elemig tartalmazza a számokat. Ezeket kivonva megkapjuk a prímszámok különbségét. Ha ez a különbég 2, akkor beszélünk ikerprímekről. Azt, hogy a különbég 2-e, az idx változó nézi meg, majd az indexüket eltárolja.

a t1primes változó tartalmazza azokat a prímeket, amelyeknek a helyét már meghatároztuk az idx változóban. Tehát az ikerprímek első fele. A t2primes viszont nem szimplán a primes[idx]-et adja vissza, hiszen az az ikrek első fele lenne, de tudjuk, hogy a különbég a kettő prím között 2, tehát a másik felét úgy kapjuk meg, ha hozzáadunk az [idx] helyen álló számhoz 2-öt.

Az rt1plust2 változóban összeadjuk a t1 és a t2 reciprokait. majd végezetül visszaadjuk a return()-ben a reciprokértékek összegét.

2.8. 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_nagykonyben_a_monty_hall-paradoxon_kapcsan

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R

```
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

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

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

  }else{

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


  }

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

```
}

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

for (i in 1:kiserletek_szama) {

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

}

valtoztatesnyer = which(kiserlet==valtoztat)

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

Tanulságok, tapasztalatok, magyarázat...

a kísérlet változóban található a nyeremény helye, a játékos változóban található a játékos által választott ajtó. Az első for ciklusban megnézzük, hogy a játékos eltalálta-e a helyes ajtó számát, és a műsorvezető ezen feltételtől függően választ ajtót magának. Ha eltalálta, akkor a műsorvezető véletlenszerűen választ a két üres ajtó közül. Viszont ha nem találta el a játékos, akkor a vezető csak 1 ajtót választhat, hiszen nem nyithatja ki a nyereményt, se a játékos által választott ajtót.

3. fejezet

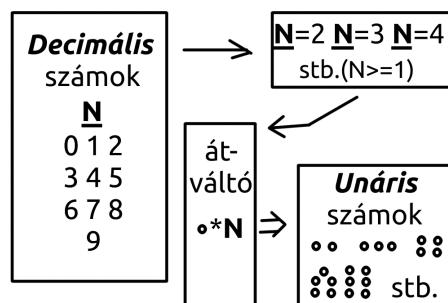
Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:



3.1. ábra. Az átváltó Turing gép

Tanulságok, tapasztalatok, magyarázat...

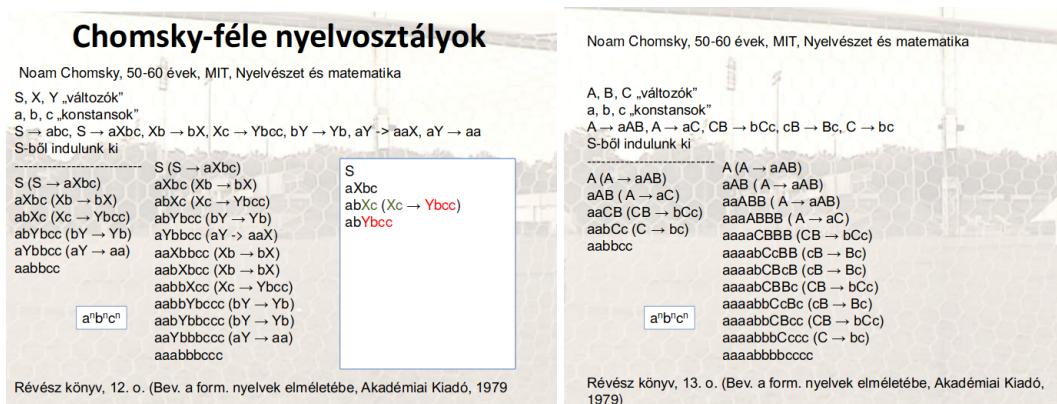
Az átváltó bekér egy Decimális számot, legyen ez egy tetszőleges szám, és nevezzük el N-nek. Ezután egy képlettel átváltja azt Unáris számrendszerbe. Ez a képlet a következő: **unar=egysegelem*N** majd "kirajzolja" az eredményt.

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: [A lentebb látható képek itt megtalálhatóak a 30. oldalon](#)



3.2. ábra. A környezetfüggő grammatikák

Tanulságok, tapasztalatok, magyarázat...

A konstansok és változók alatt helyezkednek el a helyettesítési szabályok. Ezeket a szabályokat alkalmazva addig változtatjuk a megadott szót, amíg a szó maga már csak konstansokból áll. Mivel nincs olyan lehetőség, hogy a szó csak változókból áll, ezért a nyelvezet nem lehet környezetfüggetlen.

Környezetfüggő(hossznemcsökkentő)

$P1XP2 \rightarrow P1QP2$, $P1, P2$ eleme $(VN \cup VT)^*$, X VN beli, Q $(VN \cup VT)^+$ beli, kivéve $S \rightarrow \text{üres}$, de akkor S nem lehet jobb oldali egyetlen szabályban sem, tehát Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

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:

```
#include <stdio.h>

int main()
{
    int inline asdfu
    {
        int b = a*a;
        a=b*a;
        return a;
    }
}

asdfunc(5);
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

<https://github.com/ghjbku/DE/blob/master/c%20cuccok/Bildschirmfoto%20von%202019-03-12%2011-35-07.png> Amint láthatjuk, a funkció minden gond nélkül lefordul C99-ben, míg C89-ben hibát észlel. A hiba az *inline* parancs miatt van. ez a cmd a C99-el jött be, amely általában *extern inline* -al párban jelenik meg egy programkódban. Segítségével egy program nagyobb sebességre képes, viszont ezért cserébe nagyobb a helyigénye. Az inline függvények a sorba illesztődnek be, ha arra szükség van, így jön létre a nagyobb sebesség.

3.4. Saját lexikális elemző

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

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.l

```
//lex fájl
//fordítás c-re : lex -o output.c lexfajl.l (szükséges hozzá a flex ←
)
//jelen esetben: lex -o realnumbers.c realnumbers.l
%{
#include <stdio.h>
int realnumbers = 0;
%
digit [0-9]
%%
{digit}*(\.{digit}+) ? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

a C-re fordított program ha készen van a fordítás, akkor mondhatjuk a gcc-nek, hogy csináljon nekünk futtatható programot a c forrásból. Ezt a következő sor beírásával tehetjük meg: * gcc realnumber.c -o realnumber -lfl *

Tanulságok, tapasztalatok, magyarázat...

Ebben a feladatban kicsit eltértünk a megszokott dolguktól, ugyanis nem mi írtuk meg a C forráskódot, hanem a lexer. Ez nagyban megkönnyíti a dolgunkat, hiszen ha megnézzük a C forrást, amit a lex elkészített helyettünk, láthatjuk, hogy nem éppen egy rövid kis kód sorozatról van szó A lexer segítségével nekünk már

csak annyi a dolgunk, hogy megmondjuk neki, milyen típust keressen az inputban "digit[0-9]", és hogyan ismerje azt fel " {digit}*(\.{digit}+)? "

3.5. Leetspeak

Lexelj össze egy l33t cipher-t!

Tutoráltam: Zsolt Schachinger-t Megoldás videó: https://youtu.be/06C_PqDpD_k

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/1337d1c7.1

```
%{
//a fordítás megegyezik az előző feladatéval: lex -o output.c lexfájl.l
//majd gcc output.c -o output
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>

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

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

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

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

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

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

%}
%%%
. {

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

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

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

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

            found = 1;
            break;
        }

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

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

Tanulságok, tapasztalatok, magyarázat...

Ez a program a terminálról beolvassa a karaktereket, és randomizálva 4 különböző karaktert rak az eredeti helyére. Ha számára ismeretlen karaktert írunk be, akkor visszaadja ugyan azt. A program magja egy 1337d1c7 tömb, amely tárolja a helyettesítési értékeket minden karakterhez. Ha nem talál egyetlen karaktert sem az inputban, akkor nem ír ki semmit.

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);
```

Ha a **SIGINT** jelzés nem volt ignorálva, akkor ignorálja.

ii.

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

egy for ciklus, amely 0-tól 5-ig tart, tehát 5x fut le, és minden lefutás után inkrementálja az i értékét 1-el, majd az inkrementálási értéket adja vissza.

iii.

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

ez is egy for ciklus, viszont ebben az esetben az i inkrementálása után az eredeti, növelés előtti értéket adja vissza.

iv.

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

Ebben a for ciklusban nem csak, hogy növeljük az i értékét minden kör után, de ezen értéket behelyezzük egy tömbbe is.

v.

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

a ciklus 0-tól indul, és addig megy, amíg i kisebb mint n, továbbá a **d** pointer növelt értéke megegyezik az **s** pointer növelt értékével.

vi.

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

kiír két függvényt, melyek számokat adnak vissza, viszont a visszatérési érték precedenciát sugall.

vii.

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

visszaad kettő számot, melyekből az egyik egy függvény visszatérési értéke

viii.

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

ugyanúgy két számot ad vissza, de az f függvényben az **a** változó memóriacíme helyezkedik el.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

az első formula természetes nyelvi értelmezése: **minden x számra létezik egy olyan y szám, amely nagyobb, mint x és y prím**

a második formula természetes nyelvi értelmezése: **minden x számra létezik egy olyan y szám, amely nagyobb, mint x és y prím. Továbbá y rákövetkezőjének a rákövetkezője is prím.**

a harmadik formula természetes nyelvi értelmezése: **minden y számra létezik egy olyan x szám, hogy x prím és x kisebb, mint y**

a negyedik formula természetes nyelvi értelmezése: **minden y számra létezik egy olyan x szám, hogy y kisebb, mint x, és x nem prím.**

3.8. Deklaráció

Tutorálva Nagy Krisztián által. Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- `int a;`
egy szám típusú **a** változót
- `int *b = &a;`
egy szám typusú **b pointer**, aminek az értéke **a memóriában foglalt helye**
- `int &r = a;`
integer típusú **r értéke a lesz**
- `int c[5];`
létrehoz egy 5 számnak helyet adó **c tömböt**
- `int (&tr)[5] = c;`
létrehoz egy **tr tömb referenciaját**, melynek az értéke **c**
- `int *d[5];`
egy egészre mutató **d tömb pointer**

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

### **h funkcióra mutató pointer**

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

egy pointer, ami az l függvényre mutató pointerre mutat

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

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

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

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

Megoldás video:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:

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

```
#include <stdio.h>

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

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

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

```
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;  
}
```

4. fejezet

Helló, Caesar!

4.1. double ** háromszög mátrix

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)

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

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

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

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

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

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

```
}

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

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

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

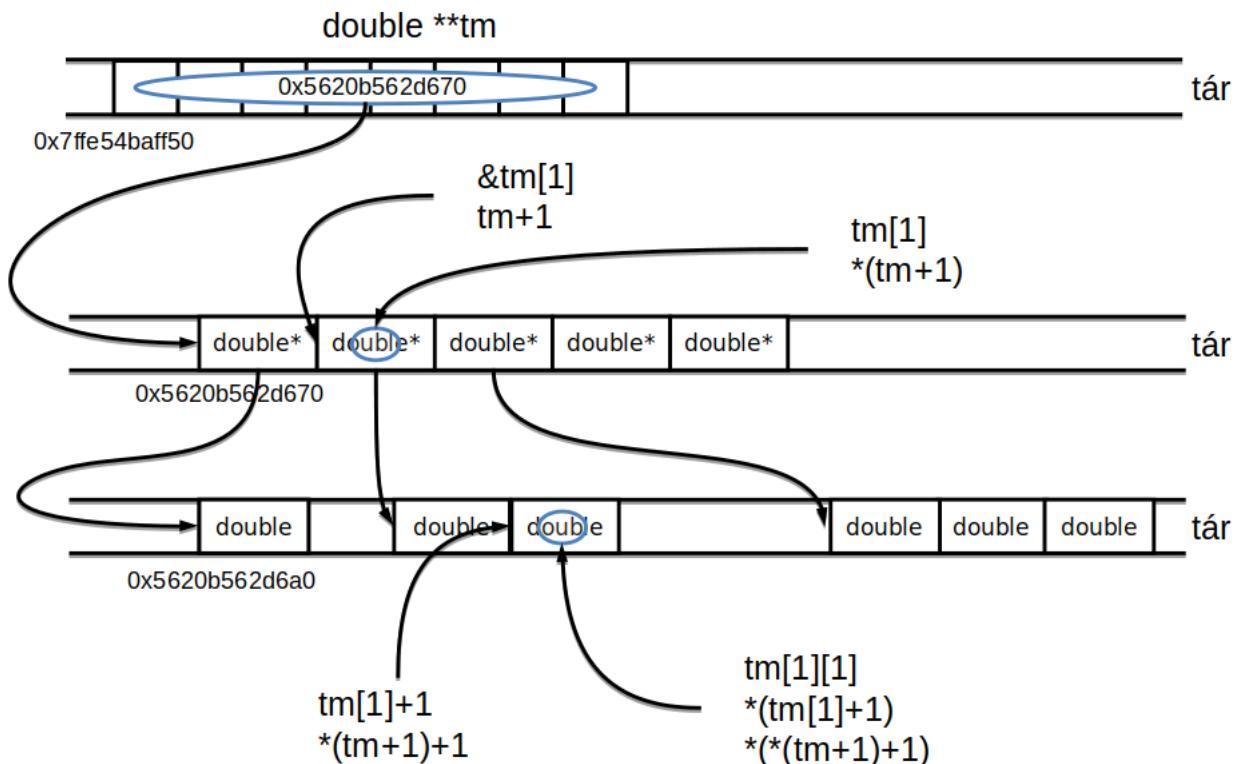
tm[3][0] = 42.0;
(* (tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
* (* (tm + 3) + 3) = 45.0;

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

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

free (tm);

return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

Az

```
int nr = 5;
```

sorral létrehozunk egy integer változót, amelynek az értéke 5, ez a változó lesz a háromszög sorainak száma.
A

```
double **tm;
```

sor pedig létrehoz egy double típusú változót, ez lesz később a programunk magja.

Ezek a sorok:

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

pedig megnézik, hogy a programunk le tud-e foglalni **nr*8** bájtot, ha nem, akkor kilép a programból a **-1** visszatérési értékkel.

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

```
{  
    return -1;  
}
```

Itt azt láthatjuk, hogy a program a tm változót tömbként kezelve bejárja azt, és mindenig **(i+1)*8**bájtot allokkál/foglal le az aktuális tömb pozíciójához. Ha ez nem sikerül, akkor megint csak kilép **-1**es visszatérési értékkel.

```
for (int i = 0; i < nr; ++i)  
    for (int j = 0; j < i + 1; ++j)  
        tm[i][j] = i * (i + 1) / 2 + j; 0 1  
  
for (int i = 0; i < nr; ++i)  
{  
    for (int j = 0; j < i + 1; ++j)  
        printf ("%f, ", tm[i][j]);  
    printf ("\n");  
}
```

Ezek a ciklusok a kiíratásért felelnek. **az első egybeágazott for ciklus pár** a tm változót tölti fel számokkal 0-tól 15-ig, majd **a második cikluspár** kiíratja azokat

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

Ezek a sorok a program végén felszabadítják a lefoglalt memóriahelyeket, először a változó tömbelemeitől kezdve, majd végül a most már üres változót is letörli.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Tutoráltam: Schachinger Zsoltot Megoldás videó:

Megoldás forrása :

```
#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
  
#define MAX_KULCS 100  
#define BUFFER_MERET 256  
  
int  
main(int argc, char **argv)  
{
```

```
char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];
int kulcs_index=0;
int olvasott_bajtok=0;
int kulcs_meret=strlen (argv[1]);
strncpy (kulcs,argv[1], MAX_KULCS);

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

Tanulságok, tapasztalatok, magyarázat...

Amint láthatjuk, a main() függvényben megjelent két ismeretlen paraméter. ezek a program futtatásánál játszanak szerepet: **az argc** jelöli az argumentumok számát, beleértve a **./programnév** sort is. ezzel szemben a ****argv** egy vektor, amely az argumentumokat tárolja. Itt például, ha a terminálba ezt a sort írjuk: **./fájlnév 1234 -o output.txt**, akkor az argc értéke 4 lesz, míg a ****argv** vektor így néz ki: **<./fájlnév; 1234; -o; output.txt>**

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

Ez a két sor is ismeretlen lehet számunkra, de nem kell tőlük megijedni, elég egyszerű a kezelésük. az első sor az argumentum_vektor 1. elemét(ami az előző példában az 1234 volt) lekéri, és megszámolja annak hosszát(**ez az strlen() függvény dolga**), majd a kulcs_meret nevű változónak ezt a hosszt értékül adja. a második sor pedig egy string másoló függvény, ennek szintaktikája a következő: **strncpy(char cél_változó,char másolni_kívánt_érték,a_másolni_kívánt_érték_hossza)** (ha a másolt érték kisebb, mint az utolsó paraméterben megadott szám, akkor a maradékot **NULL** bájtokkal fogja kipótolni a program)

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

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcssSzöveg,
                          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 olvasottBájtak = 0;  
  
    while((olvasottBájtak =  
bejövőCsatorna.read(buffer)) != -1)  
{  
  
        for(int i=0; i<olvasottBájtak; ++i)  
{  
  
            buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);  
            kulcsIndex = (kulcsIndex+1) % kulcs.length;  
        }  
  
        kimenőCsatorna.write(buffer, 0, olvasottBájtak);  
    }  
  
}  
  
public static void main(String[] args) {  
  
    try {  
  
        new ExorTitkosító(args[0], System.in, System.out);  
    } catch(java.io.IOException e) {  
  
        e.printStackTrace();  
    }  
}  
}
```

Tanulságok, tapasztalatok, magyarázat...

A Java verzió is hasonlóképpen működik, mint a C verzió, csak a nyelvi sajátosságoknak köszönhetően találhatóak különbségek a két kód között. Mivel a Java is magasszintű programozási nyelv, ezért a forráskód értelmezése könnyebb, mint egy assembly nyelv.

A main függvényben található egy try-catch blokk, ez egyfajta hibakeresés. A try részbe kerülnek a kódok, amik nagy eséllyel hibát dobhatnak, és a catch részben ezeket a hibákat elkapja a program, és a programozó által megadott üzenetet dobja ki. Itt például Ha valami nem stimmel a megadott argumentumokkal, a program kiad egy exception-t. Ez a C programban nincs jelen, de ha szeretnénk, akár oda is beleírhatjuk.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

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

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

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

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

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

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

}

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

    return tiszta_lehet (titkos, titkos_meret);
}

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

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

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

    // osszes kulcs eloallitasa
    for (int ii = '0'; ii <= '9'; ++ii)
        for (int ji = '0'; ji <= '9'; ++ji)
            for (int ki = '0'; ki <= '9'; ++ki)
                for (int li = '0'; li <= '9'; ++li)
                    for (int mi = '0'; mi <= '9'; ++mi)
                        for (int ni = '0'; ni <= '9'; ++ni)
                            for (int oi = '0'; oi <= '9'; ++oi)
                                for (int pi = '0'; pi <= '9'; ++pi)
    {
```

```
        kulcs[0] = ii;
        kulcs[1] = ji;
        kulcs[2] = ki;
        kulcs[3] = li;
        kulcs[4] = mi;
        kulcs[5] = ni;
        kulcs[6] = oi;
        kulcs[7] = pi;

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

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

    return 0;
}
```

és a több magos változat:

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

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

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

    return (double) titkos_meret / sz;
}

int
tiszta_lehet (const char *titkos, int titkos_meret)
```

```
{  
    // a tiszta szöveg valszeg tartalmazza a gyakori magyar szavakat  
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a  
    // potenciális töréseket  
  
    double szohossz = atlagos_szohossz (titkos, titkos_meret);  
  
    return szohossz > 6.0 && szohossz < 9.0  
        && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")  
        && strcasestr (titkos, "az") && strcasestr (titkos, "ha");  
  
}  
  
void  
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret, ←  
      char *buffer)  
{  
  
    int kulcs_index = 0;  
  
    for (int i = 0; i < titkos_meret; ++i)  
    {  
  
        buffer[i] = titkos[i] ^ kulcs[kulcs_index];  
        kulcs_index = (kulcs_index + 1) % kulcs_meret;  
  
    }  
  
}  
  
void  
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],  
            int titkos_meret)  
{  
  
    char *buffer;  
  
    if ((buffer = (char *)malloc(sizeof(char)*titkos_meret)) == NULL)  
    {  
        printf("Memoria (buffer) faliora\n");  
        exit(-1);  
    }  
  
    exor (kulcs, kulcs_meret, titkos, titkos_meret, buffer);  
  
    if (tiszta_lehet (buffer, titkos_meret))  
    {  
        printf("Kulcs: [%c%c%c%c%c%c]\nTiszta szöveg: [%s]\n",  
              kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],kulcs ←  
              [6],kulcs[7], buffer);  
    }  
}
```

```
}

free(buffer);

}

int
main (void)
{

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

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

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

    // osszes kulcs eloallitasa
#pragma omp parallel for private(kulcs)
    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;

                                    exor_tores (kulcs, KULCS_MERET, titkos, ↵
                                                p - titkos);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
        }  
  
    return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

az atlagoz_szohossz függvény bekéri a titkos szöveget és méretét, majd egy 'sz' változóban megszámolja, hogy hány szóköz található a szövegben. Ezután a szöveg teljes méretét elosztja az 'sz' változó értékével, így megkapva az átlagos szóhosszt.

az strcasestr függvény azt keresi, hogy a titkos-ban megtalálható-e a **2. paraméterben megadott szöveg** eg.: "hogy" és "nem"

A többmagos változatban a különbség ott mutatkozik meg, hogy az **összes kulcs előállítása** résznél a #pragma sor megjelenik. Ez a processzor magok között szétosztja a tennivalót, és "párhuzamosan" számolja majd írja ki az összes lehetséges 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

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com  
#  
# This program is free software: you can redistribute it and/or modify  
# it under the terms of the GNU General Public License as published by  
# the Free Software Foundation, either version 3 of the License, or  
# (at your option) any later version.  
#  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU General Public License for more details.  
#  
# You should have received a copy of the GNU General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>  
#  
# https://youtu.be/Koyw6IH5ScQ  
  
library(neuralnet)  
  
a1      <- c(0,1,0,1)  
a2      <- c(0,0,1,1)  
OR      <- c(0,1,1,1)  
  
or.data <- data.frame(a1, a2, OR)
```

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

plot(nn.or)

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

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

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

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

plot(nn.operand)

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

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

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)
```

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

Tanulságok, tapasztalatok, magyarázat...

Ez a program a neurális hálóra alapszik. **A neurális hálózat biológiai neuronok összekapcsolt csoportja. Modern használatban a szó alatt a mesterséges neurális hálót értjük, amelyek mesterséges neuronokból állnak.** forrás: https://hu.wikipedia.org/wiki/Neur%C3%A1lis_h%C3%A1l%C3%A1t%C3%A1s

Itt a library(neuralnet) sor hasonlóképpen működik, mint a #include parancs a C nyelvben. A számításokért ez a könyvtár felel. Ezek a neurális hálók egyfajta ai-ként tekinthetőek, azaz megtanítjuk a számítógépnek, hogy az egyes kapukat felismerje. Ez a három kapu az OR,AND,ORAND és az EXOR.

Az OR kapu és annak "plotolása" itt található

```
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)
```

Az ORAND pedig itt:

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

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

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

plot(nn.orand)

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

Végül pedig az EXOR kapu:

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

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

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

plot(nn.exor)

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

4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

A forráskódöt a hossza miatt nem tenném bele a könyvbe, de kódsnippet-eket fogok használni.

```
//main.cpp fájl tartalma
#include <iostream>
#include "mlp.cpp"
#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;
}
```

Tanulságok, tapasztalatok, magyarázat...

A program lényegében annyit csinál, hogy végigfut a bemeneten, és megszámolja a piros pixeleket. Ezt a két egymásba épített forciklusban láthatjuk. az első ciklus **for(int i {0};i<png_image.get_width();++i)** 0-tól a kép szélességéig fut, míg a második ciklus **for(int j{0};j<png_image.get_height();++j)** a kép magasságáig fut, a ciklus belséjében található maga a számolási művelet. **image[i*png_image.get_Width() + j] = png_image[i][j].red;** A kép szélességét szorozza i-vel és hozzáad j-t, és ez lesz az **image** változónk indexe. ezen változót egyenlővé tesszük a **png_image[i][j]** kép piros(red) tagjával. ezután egy **value** változóban eltároljuk az **image** változó értékét, amely perceptronra mutat. Végül kiírjuk a **value** értékét. a lefordításhoz ezt kell beírni:**g++ mpl.hpp main.cpp -o perceptron -std=c++11**

5. fejezet

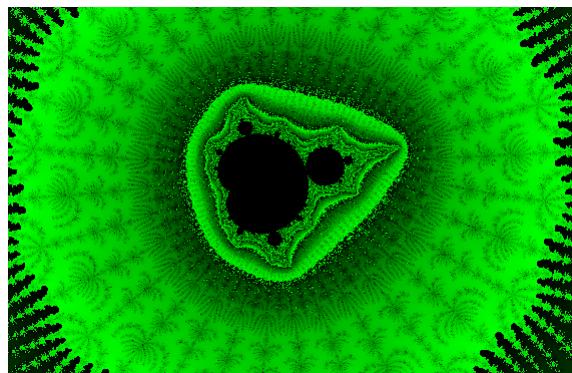
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: [/mandelpngt.c++](#) 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-et 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 végtelen sok z -t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácsponthoz. Ha eközben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácspont a halmaz része. (Színes meg úgy lesz a kép, hogy változatosan színezzük, például minél későbbi z -nél lép ki a körből, annál sötétebbre).

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

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

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhax/attention_raising/Mandelbrot/3.1.2.cpp](https://github.com/bhax/bhax/blob/master/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
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1 --left-footer="←
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
// color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
// 
// 
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
// 
```

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

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double a = -1.9;
    double b = 0.7;
    double c = -1.3;
    double d = 1.3;

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

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

    double dx = ( b - a ) / szelesseg;
```

```
double dy = ( d - c ) / magassag;
double reC, imC, rez, imZ;
int iteracio = 0;

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

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

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

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

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

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

        while ( std::abs ( z_n ) < 4 && iteracio < iteracioHatar )
        {
            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;
}
```

A program megírásához szükségünk lesz az STD::complex könyvtárra, és a png++/png.cpp könyvtárra. Ez utóbbi felel az adatok képként való megjelenítéséért. A kep.set_pixel függvény felel a kép elkészítéséért, míg a benne lévő rgb_pixel a színének módosításáért.

```
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;
    }
}
```

Ebben a kódsnippetben láthatjuk, ahogy a main-ben megadunk egy kezdő értéket a létrehozandó képnek. Az alatta lévő elágazásban megnézzük, hogy az argumentumok száma 9-e, ha igen, akkor a szelesseg változó értékét megváltoztatja a második argumentummal, a magassag változóét a harmadik argumentummal, az iteraciosHatar változóét a negyedik argumentummal, és ezeket minden az atoi funkcióval. Az atoi funkció stringet int-té alakít át. az a,b,c,d változók értékeit pedig az 5-dik, 6-dik, 7-dik és 8-dik argumentummal cseréli ki. Ezeket minden az atof függvény segítségével érik el, amely stringből double-t csinál. Ha a feltétel nem teljesül, akkor pedig kiírja a standard kimenetre a használatot, majd -1-el tér vissza.

```
png::image < png::rgb_pixel > kep ( szelesseg, magassag );
double dx = ( b - a ) / szelesseg; double dy = ( d - c ) / magassag; double ←
    reC, imC, reZ, imZ; int iteracio = 0;
std::cout << "Szamitas\n";
// j megy a sorokon for ( int j = 0; j < magassag; ++j )
{
// k megy az oszlopokon
for ( int k = 0; k < szelesseg; ++k ) {
// c = (reC, imC) a halo racspontjainak // megfelelo komplex szam
reC = a + k * dx; imC = d - j * dy;
std::complex<double> c ( reC, imC );
std::complex<double> z_n ( 0, 0 );
iteracio = 0;
```

```

while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar ) { z_n = z_n * ←
    z_n + c;
++iteracio;
}
kep.set_pixel ( k, j,
                png::rgb_pixel ( iteracio%255, (iteracio*iteracio ←
                    )%255, 0 ) );
}

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

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

```

Ebben a kódrészben történik a halmaz előállítása, a reC és imC változók értékmegadása,a c és z_n inicializálása. egy while ciklusban iteráljuk a z_n változót, megszorozzuk önmagával és hozzáadjuk a c-t. Amíg a z_n abszolutértéke kisebb mint 4, és az iteráció kisebb, mint az iterációs határ. Alatta láthatjuk a kép elkészítését a set_pixel funkcióval. Ezalatt található egy visszajelzés, majd végül a kep.write funkció segítségével az 1. argumentumba megadott fájlnévként kimenti a program a képet. Végül pedig kiírja annak nevét a konzolra.

5.3. Biomorfok

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

A forráskód Bátfai Norbert tulajdonában áll. 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 < szelessseg; ++k )
    {
        double reZ = a + k * dx;
        double imZ = d - j * dy;
        std::complex<double> z_n ( reZ, imZ );

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

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
```

```
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ↵
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
// color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↵
// Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

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

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
    }
}
```

```
iteraciosHatar = atoi ( argv[4] );
xmin = atof ( argv[5] );
xmax = atof ( argv[6] );
ymin = atof ( argv[7] );
ymax = atof ( argv[8] );
reC = atof ( argv[9] );
imC = atof ( argv[10] );
R = atof ( argv[11] );

}

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

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

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

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

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

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

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

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

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

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

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

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

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

}
```

A mandelbrothoz képest ez a program több argumentummal rendelkezik, 9 helyett 12 van. Az új argumentumok a **d,reC,imC és az R**. A reC és imC argumentumokból állítsuk elő a cc-t. Megtörténik a halmaz előállítása, a z_n változó harmadik hatványra emelése és cc hozzáadása. Ha az R(valós) kisebb, mint a z_n, vagy ha R(képzetes) kisebb, mint z_n, akkor az iteracio változó értékét i-vel tesszük egyenlővé. Alatta láthatjuk a kép elkészítését a set_pixel funkcióval. Ezalatt található egy visszajelzés, majd végül a kep.write funkció segítségével az 1. argumentumba megadott fájlnévként kimenti a program a képet. Végül pedig kiírja annak nevét a konzolra.

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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

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.

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: <https://github.com/ghjbku/DE/tree/master/sajat/biomoprhs>

Ezen feladat megoldása hihetetlenül sokáig tartott, és ez alatt nem a program megírását értem, hanem annak lefordítását. A legnagyobb fejfájást a makefile legenerálása okozta, viszont hosszas keresgélés után ráleltem a megoldásra, ami egyetlen sor:**sudo apt-get install qt5-default** ezután jöhet a **qmake -project** parancs, amivel létrehozzuk a *.pro fájlunkat, ebből lesz a make fájl. A makefile legenerálása ezen parancs beírásával történik: **qmake profájlneve.pro**, majd **make** és már futtatható a program. A képet a jobb egérgomb használatával lehet nagyítani.

5.6. Mandelbrot nagyító és utazó Java nyelven

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

Köszönét a forráskódért:[Lovász Botond](#)

```
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.awt.event.*;

public class Mandelbrot extends JFrame implements ActionListener {

    private JPanel ctrlPanel;
    private JPanel btnPanel;
    private int numIter = 50;
    private double zoom = 130;
    private double zoomIncrease = 100;
    private int colorIter = 20;
    private BufferedImage I;
    private double zx, zy, cx, cy, temp;
    private int xMove, yMove = 0;
    private JButton[] ctrlBtns = new JButton[9];
    private Color themeColor = new Color(150,180,200);

    public Mandelbrot() {
        super("Mandelbrot Set");
        setBounds(100, 100, 800, 600);
        setResizable(false);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        plotPoints();

        Container contentPane = getContentPane();
        contentPane.setLayout(null);

        ctrlPanel = new JPanel();
        ctrlPanel.setBounds(600,0,200,600);
        ctrlPanel.setBackground(themeColor);
        ctrlPanel.setLayout(null);

        btnPanel = new JPanel();
        btnPanel.setBounds(0,200,200,200);
        btnPanel.setLayout(new GridLayout(3,3));
        btnPanel.setBackground(themeColor);

        ctrlBtns[1] = new JButton("up");
        ctrlBtns[7] = new JButton("down");
    }
}
```

```
ctrlBtns[3] = new JButton ("left");
ctrlBtns[5] = new JButton("right");
ctrlBtns[2] = new JButton("+");
ctrlBtns[0] = new JButton("-");
ctrlBtns[8] = new JButton(">");
ctrlBtns[6] = new JButton("<");
ctrlBtns[4] = new JButton();

contentPane.add(ctrlPanel);
contentPane.add(new imgPanel());
ctrlPanel.add(btnPanel);

for (int x = 0; x<ctrlBtns.length;x++) {
    btnPanel.add(ctrlBtns[x]);
    ctrlBtns[x].addActionListener(this);
}

validate();
}

public class imgPanel extends JPanel{
    public imgPanel(){
        setBounds(0,0,600,600);

    }

    @Override
    public void paint (Graphics g){
        super.paint(g);
        g.drawImage(I, 0, 0, this);
    }
}

public void plotPoints(){
    I = new BufferedImage(getWidth(), getHeight(), BufferedImage. TYPE_INT_RGB);
    for (int y = 0; y < getHeight(); y++) {
        for (int x = 0; x < getWidth(); x++) {
            zx = zy = 0;
            cx = (x - 320+xMove) / zoom;
            cy = (y - 290+yMove) / zoom;
            int iter = numIter;
            while (zx * zx + zy * zy < 4 && iter > 0) {
                temp = zx * zx - zy * zy + cx;
                zy = 2 * zx * zy + cy;
                zx = temp;
                iter--;
            }
            I.setRGB(x, y, iter | (iter << colorIter));
        }
    }
}
```

```
        }
    }

public void actionPerformed(ActionEvent ae) {
    String event = ae.getActionCommand();

    switch (event) {
        case "up":
            yMove-=100;
            break;
        case "down":
            yMove+=100;
            break;
        case "left":
            xMove-=100;
            break;
        case "right":
            xMove+=100;
            break;
        case "+":
            zoom+=zoomIncrease;
            zoomIncrease+=100;
            break;
        case "-":
            zoom-=zoomIncrease;
            zoomIncrease-=100;
            break;
        case ">":
            colorIter++;
            break;
        case "<":
            colorIter--;
            break;
    }

    plotPoints();
    validate();
    repaint();
}

public static void main(String[] args) {
    new Mandelbrot().setVisible(true);
}
```

A program hasonlóképp funkcionál, mint a c++ verzió, viszont itt már több lehetőségünk van manipulálni a mutatott képet.

Először létrehozzuk a "ctrlBnts[*]" változókat, majd lehallgatjuk őket a **ctrlBnts[x].addActionListener(this)** parancssal. A Switch-ben inicializáljuk a gombok funkcióját. A felfelé nyíl esetén az y koordinátánkat csökkentsük 100-al, lefelé nyíl esetén növeljük azt. Bal nyíl megnyomására az x koordináta csökken 100-al, jobb nyíl-nál pedig növekszik. A plusz gomb nagyít a képen, a mínusz nyíl távolít. Végül a két kacsacsőr a szín megjelenítésén változtat.

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:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása Java-ban:

```
public class PolárGenerátor {  
  
    boolean nincsTárolt = true;  
    double tárolt;  
  
    public PolárGenerátor() {  
  
        nincsTárolt = true;  
    }  
  
    public double következő() {  
  
        if(nincsTárolt) {  
  
            double u1, u2, v1, v2, w;  
            do {  
                u1 = Math.random();  
                u2 = Math.random();  
  
                v1 = 2*u1 - 1;  
                v2 = 2*u2 - 1;  
  
                w = v1*v1 + v2*v2;  
            } while(w == 0);  
            nincsTárolt = false;  
            tárolt = w;  
        }  
        return tárolt;  
    }  
}
```

```
    } while(w > 1);

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

    tárolt = r*v2;
    nincsTárolt = !nincsTárolt;

    return r*v1;

} else {
    nincsTárolt = !nincsTárolt;
    return tárolt;
}
}

public static void main(String[] args) {

    PolárGenerátor g = new PolárGenerátor();

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

}
}
```

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása C++-ban:

```
#include "std_lib_facilities.h"

class PolarGenerator{

bool nincsTarolt=true;
double tarolt;

public :
double kovetkezo()
{

if(nincsTarolt)
{
double u1,u2,v1,v2,w;
u1= ((double) rand() / (double) (RAND_MAX));
u2= ((double) rand() / (double) (RAND_MAX));
v1=(2*u1)-1;
v2=(2*u2)-1;

w=(v1*v1)+(v2*v2);
while(w>1)
```

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

};

int main()
{
std::srand(std::time(0));
PolarGenerator g;
for(int i=0; i<10; ++i)
std::cout<<g.kovetkezo()<<std::endl;
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat... térj ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked! A java forrás szemantikailag megegyezik a JDK forrásokkal.

A Java kódhoz képes van egy kis változtatás a kódban. Már az elején található egy különbség, a **public** kulcsszó, amely Java-ban minden függvény elő bekerül, amit public-ként szeretnénk kezelní, amíg C++-ban egy egyszerű **public:** kulcsszó után bármennyi függvényt deklarálhatunk, az minden publikus lesz.

Egy másik különbség, hogy amíg Java-ban a matematikai műveleteket a **Math.művelet** előtaggal hívjuk meg, addig C++-ban az összes ilyen művelet függvényként van beépítve egy cmath header fájlba, így csak a függvény neveit kell meghívnunk. Viszont hátrányként tekinthető, hogy az **#include <cmath>** sor nélkül egy művelet se használható.(persze az alapműveletek kivételek: +-*/%)

6.2. LZW

Valósítsd meg C-ben az LZW algoritmus fa-építését! **Tutorált benne: Nagy Laszló Mihály**

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:

```
// z.c
//
// LZW fa építő
// Programozó Páternoszter
//
```

```
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, 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/>.
//
// 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.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
// Version history:
//
// 0.0.1, http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2, csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3, http://progpater.blog.hu/2011/03/05/ ←
//        labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
//

#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;

BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

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

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

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;

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

```
{  
    fa->jobb_egy = uj_elem ();  
    fa->jobb_egy->ertek = 1;  
    fa->jobb_egy->bal nulla = fa->jobb_egy->jobb_egy = NULL;  
    fa = gyoker;  
}  
else  
{  
    fa = fa->jobb_egy;  
}  
}  
}  
  
printf ("\n");  
kiir (gyoker);  
  
extern int max_melyseg, 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 ("atlag=%f\nszoras=%f\n", atlag, szoras);  
szabadit (gyoker);
```

```
}

// a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
// http://sourceforge.net/projects/javacska/
// az atlag() hivasakor is inicializalni kell oket, a
// a rekurziv bejaras hasznalja
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;

    }
}

}

// a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
// http://sourceforge.net/projects/javacska/
// az atlag() hivasakor is inicializalni kell oket, a
// a rekurziv bejaras hasznalja
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;  
    szorasoszeg += ((melyseg - atlag) * (melyseg - atlag));  
  
}  
  
}  
  
}  
  
//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 bejáráshoz képest  
        // 1-el nagyobb mélység, ezért -1  
        for (int i = 0; i < melyseg; ++i)  
            printf ("---");  
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↔  
               ,  
               melyseg-1);  
        kiir (elem->bal nulla);  
        --melyseg;  
    }  
}  
  
void  
szabadit (BINFA_PTR elem)  
{  
    if (elem != NULL)  
    {  
        szabadit (elem->jobb_egy);  
        szabadit (elem->bal nulla);  
        free (elem);  
    }  
}
```

```
BINFA_PTR  
uj_elem ()  
{  
    BINFA_PTR p;
```

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

Ez a függvény egy p nevű BINFA_PTR típusú változót hoz létre, ha a memória foglalás nem valósul meg, akkor a **perror()**-ban megadott szöveget adja vissza, majd kilép az EXIT_FAILURE kulcsszóval. visszatérési értéke ez a p változó lesz.

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

ez a négy függvény extern, ami annyit jelent, hogy globálisan használható. Egyébként visszatérési értéket nem várnak a függvények, argumentumként pedig egy BINFA_PTR típusú adatot várnak.

```
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;
```

A BINFA_PTR gyoker = uj_elem (); sor egy BINFA_PTR típusú változót hoz létre, és értékül az uj_elem függvény visszatérési értékét kapja, ami a p változó volt. Ez a gyoker változó rendelkezik ertek alváltozóval, aminek az értékére a '/' jelet állítsuk be. a következő sorban a gyoker változó bal nulla és jobb_egy értékeit lenullázzuk. Majd létrehozunk egy újabb változót, és értékül adjuk neki a gyoker változót.

```
    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;
        }
    }
```

```
}
```

Nézzük a nullás ágat először: ha a b változó 0-át tartalmaz ebben a tickben, akkor : Ha a fa változó bal_nulla tagja eddig nem tartlamazott értéket, akkor legyen az értéke az uj_elem függvény visszatérési értéke. a bal_nulla változó ertek tag értéke pedig legyen "0"; Ezután a bal_nulla változóra mutató bal_nulla objektum értékét tesszük egyenlővé a jobb_ely változóval, és azt lenullázzuk. Végül a fa objektumot egyenlővé tesszük a gyoker-rel. Ha a bal_nulla nem volt üres, akkor a fa változót tovább léptetjük a bal_nulla ágra.

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

Végül jöhet az eggyes ág: ha a jobb_ely objektum értéke eddig NULL volt, akkor: a jobb_ely legyen az uj_elem függvény p értéke. a jobb_ely ertek változó értéke legyen "1". A jobb_ely->bal_nulla objektum értékéül válaszzuk a jobb_ely-et, aminek pedig a NULL értéket adjuk. végül a fa objektumot egyenlővé tesszük a gyoker-el. Ha a jobb_ely nem volt NULL értékű, akkor a fa változót léptessük oda.

6.3. Fabejárás

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

Megoldás videó:

Preorderbejárás: azaz a gyökér elem majd a bal oldali részfa preorder bejárása, végül ajobboldali részfa preorder bejárása.

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

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

Inorderbejárás: azaz először a bal részfa inorder bejárása, majd a gyökérelem, végül a jobboldali részfa inorder bejárása.

Postorderbejárás: azaz először a bal részfa posztorder bejárása, majd a jobboldali részfaposztorder bejárása, végül a gyökérelem feldolgozása.

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

        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        kiir (elem->jobb_egy);
        kiir (elem->bal_nulla);

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

        --melyseg;
    }
}
```

[forrás](#)

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: [a nagy forráskód miatt csak linkként jelenítem meg](#)

```
protected:      // ha esetleg egyszer majd kiterjesztjük az osztályt, ←
               mert
// akarunk benne valami újdonságot csinálni, vagy meglévő tevékenységet ←
// máshogyan... stb.
```

```
// akkor ezek látszanak majd a gyerek osztályban is

/* A fában tagként benne van egy csomópont, ez erősen ki van tüntetve, ←
   Ő a gyökér: */

Csomopont gyoker;
int maxMelyseg;
double atlag, szoras;

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

};
```

Amint láthatjuk, a binfa osztály protected ágában megtalálható a Csmopont típusú gyoker változó. a fa objektum jelképezi a tree-t, míg a node-t jelképezi a Csmopont.

6.5. Mutató a gyökér

Tutorálva Nagy Krisztián által. Írd át az előző forrást, hogy a gyökér csomópont ne kompozícióban, csak aggregációban legyen a fával!

Megoldás videó:

A forráskód-részlet Bátfai Norbert tulajdonában áll. Megoldás forrása: [z3a7_gyoker.cpp](#)

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

Ehhez szükségünk lesz arra, hogy az előző feladatban megadott **Csmopont gyoker;** sort átírjuk **Csmopont gyoker*;**-ra

Viszont mivel mutató lett belőle, valahol inicializálnunk kell a változót. Ezt jelenti a(z) **LZWBinFa ()::fa (gyoker = new Csmopont('/'))** sor

Át kell írnunk továbbá a programban található **gyoker** változó hívásokat **gyoker*-ra**.

Viszont ez még mindig nem elég, hiszen a memóriafoglalást követően valahogyan fel is kell őket szabadítani. És itt jön be az ~LZWBinFa destruktur.

6.6. Mozgató szemantika

Tutorált Gila Attila Zoltán Írj az előző programhoz mozgató konstruktort és értékkadást, a mozgató konstruktor legyen a mozgató értékkadásra alapozva!

Megoldás videó:

A forráskód Tamás Racs tulajdonában áll. Megoldás forrása: std::move parancs API: <https://en.cppreference.com/w/cpp/utility/move>

```
#include <iostream>
#include <fstream>
class LZWBinaryTree
{
public:
LZWBinaryTree ()
{
currentNode = root;
}
~LZWBinaryTree ()
{
free (root);
}
LZWBinaryTree (LZWBinaryTree&& other)
{
root = nullptr;
*this = std::move(other);
}
LZWBinaryTree& operator=(LZWBinaryTree&& other)
{
std::swap(root, other.root);
return *this;
}
void operator<< (char b)
{
if (b == '0')
{
if (!currentNode->getLeftChild ())
{
Node *uj = new Node ('0');
currentNode->newLeftChild (uj);
currentNode = root;
}
else
{
currentNode = currentNode->getLeftChild ();
}
}
else
{
if (!currentNode->getRightChild ())
{
Node *uj = new Node ('0');
currentNode->newRightChild (uj);
currentNode = root;
}
else
{
currentNode = currentNode->getRightChild ();
}
}
}
```

```
{  
Node *uj = new Node ('1');  
currentNode->newRightChild (uj);  
currentNode = root;  
}  
  
else  
{  
currentNode = currentNode->getRightChild ();  
}  
}  
}  
  
void print (void)  
{  
depth = 0;  
print (root, std::cout);  
}  
  
  
int getDepth (void);  
friend std::ostream & operator<< (std::ostream & os, LZWBinaryTree & bf --  
)  
{  
bf.print (os);  
return os;  
}  
  
void print (std::ostream & os)  
{  
depth = 0;  
print (root, os);  
}  
  
private:  
class Node  
{  
public:  
Node (char b = '/'):value (b), leftChild (0), rightChild (0)  
{  
};  
~Node ()  
{  
};  
Node *getLeftChild () const  
{  
return leftChild;  
}  
Node *getRightChild () const  
{  
return rightChild;  
}  
void newLeftChild (Node * gy)  
{  
leftChild = gy;
```

```
}

void newRightChild (Node * gy)
{
rightChild = gy;
}
char getValue () const
{
return value;
}
private:
char value;
Node *leftChild;
Node *rightChild;
Node (const Node &);
Node & operator= (const Node &);
};

Node *currentNode;
int depth;
LZWBinaryTree (const LZWBinaryTree &);
LZWBinaryTree & operator= (const LZWBinaryTree &);
void print (Node * n, std::ostream & os)
{
if (n != NULL)
{
++depth;
print (n->getLeftChild (), os);
for (int i = 0; i < depth; ++i)
os << "----";
os << n->getValue () << "(" << depth << ")" << std::endl;
print (n->getRightChild (), os);
--depth;
}
}
void free (Node * n)
{
if (n != NULL)
{
free (n->getLeftChild ());
free (n->getRightChild ());
delete n;
}
}
protected:
Node* root = new Node ();
int maxDepth;
void getDepthRec (Node * n);
};
int LZWBinaryTree::getDepth (void)
{
depth = maxDepth = 0;
```

```
getDepthRec (root);
return maxDepth;
}
void LZWBinaryTree::getDepthRec (Node * n)
{
if (n != NULL)
{
++depth;
if (depth > maxDepth)
maxDepth = depth;
getDepthRec (n->getRightChild ());
getDepthRec (n->getLeftChild ());
--depth;
}
}
void usage (void)
{
std::cout << "Usage: lzwtree in_file" << std::endl;
}
int main (int argc, char *argv[])
{
if (argc != 2)
{
usage ();
return -1;
}
char *inFile = *++argv;
std::fstream beFile (inFile, std::ios_base::in);
if (!beFile)
{
std::cout << inFile << "Nem létezik a bemeneti fájl!" << std::endl;
usage ();
return -3;
}
char b;
LZWBinaryTree binFa;
while (beFile.read ((char *) &b, sizeof (char)))
{
if (b == '0')
{
binFa << b;
}
else if (b == '1')
{
binFa << b;
}
}
std::cout << "Eredeti fa:\n\n";
std::cout << binFa;
std::cout << "depth = " << binFa.getDepth () << std::endl;
```

```
LZWBinaryTree binFa2 = std::move(binFa);
std::cout << "\nEredeti fa mozgatás után:\n";
std::cout << binFa;
std::cout << "depth = " << binFa.getDepth () << std::endl;
std::cout << "\nMozgatással létrejött fa:\n\n";
std::cout << binFa2;
std::cout << "depth = " << binFa2.getDepth () << std::endl;
beFile.close ();
return 0;
}
```

A lent található két függvény teszi lehetővé az objektumok mozgatását.

Amint láthatjuk, az első sorban a jelenlegi root-ot lecseréli a program nullptr-re, majd a *this, tehát ezen tree értékét egyenlővé teszi az std::move(other) funkció visszatérési értékével. Maga a move függvény nem tesz semmit az objektumunkkal, egyszerűen csak a baloldali értékből jobboldali értéket készít.

```
LZWBinaryTree (LZWBinaryTree&& other)
{
root = nullptr;
*this = std::move(other);
}
```

Majd az std::swap függvény segítségével kicseréljük a két objektum memóriacímét, végül visszatérési értékként a *this pointert adjuk meg.

```
LZWBinaryTree& operator=(LZWBinaryTree&& other)
{
std::swap(root, other.root);
return *this;
}
```

7. fejezet

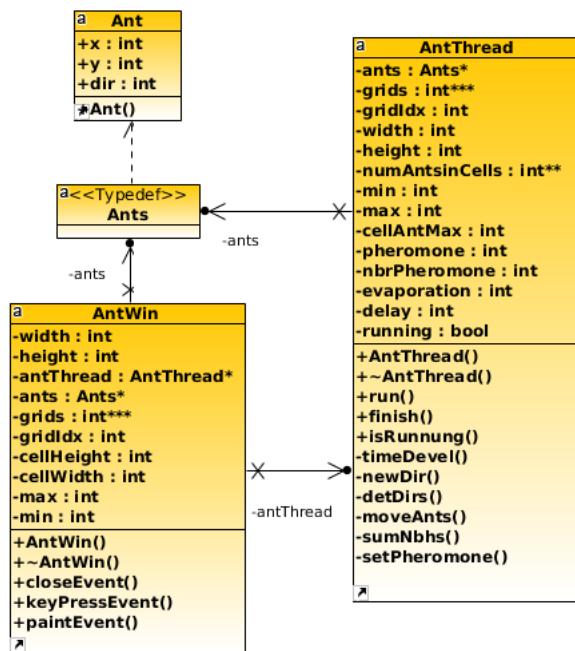
Helló, Conway!

7.1. Hangyszimulációk

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

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:



7.1. ábra. A hangyszimuláció UML diagram



7.2. ábra. A hangyák akcióiban

```
// BHAX Myrmecologist
//
// Copyright (C) 2019
// Norbert Bátfa, 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/>.
//
// https://bhaxor.blog.hu/2018/09/26/hangyaszimulacioik
// https://bhaxor.blog.hu/2018/10/10/myrmecologist
//

#include <QApplication>
#include <QDesktopWidget>
#include <QDebug>
#include <QDateTime>
#include <QCommandLineOption>
#include <QCommandLineParser>
```

```
#include "antwin.h"

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

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

    QApplication a ( argc, argv );

    QCommandLineOption szeles_opt ( { "w", "szelesseg" }, "Oszlopok (cellakban ←
        ) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( { "m", "magassag" }, "Sorok (cellakban) ←
        szama.", "magassag", "150" );
    QCommandLineOption hangyaszam_opt ( { "n", "hangyaszam" }, "Hangyak szama. ←
        ", "hangyaszam", "100" );
    QCommandLineOption sebesseg_opt ( { "t", "sebesseg" }, "2 lepes kozotti ←
        ido (millisec-ben).", "sebesseg", "100" );
    QCommandLineOption parolgas_opt ( { "p", "parolgas" }, "A parolgas erteke. ←
        ", "parolgas", "8" );
    QCommandLineOption feromon_opt ( { "f", "feromon" }, "A hagyott nyom ←
        erteke.", "feromon", "11" );
    QCommandLineOption szomszed_opt ( { "s", "szomszed" }, "A hagyott nyom ←
        erteke a szomszedokban.", "szomszed", "3" );
    QCommandLineOption alapertek_opt ( { "d", "alapertek" }, "Indulo ertek a ←
        cellakban.", "alapertek", "1" );
    QCommandLineOption maxcella_opt ( { "a", "maxcella" }, "Cella max erteke." ←
        , "maxcella", "50" );
    QCommandLineOption mincella_opt ( { "i", "mincella" }, "Cella min erteke." ←
        , "mincella", "2" );
    QCommandLineOption cellamerete_opt ( { "c", "cellameret" }, "Hany hangya ←
        fer egy cellaba.", "cellameret", "4" );
    QCommandLineParser parser;

    parser.addHelpOption();
    parser.addVersionOption();
    parser.addOption ( szeles_opt );
    parser.addOption ( magas_opt );
    parser.addOption ( hangyaszam_opt );
    parser.addOption ( sebesseg_opt );
    parser.addOption ( parolgas_opt );
    parser.addOption ( feromon_opt );
    parser.addOption ( szomszed_opt );
    parser.addOption ( alapertek_opt );
    parser.addOption ( maxcella_opt );
    parser.addOption ( mincella_opt );
```

```
parser.addOption ( cellamerete_opt );

parser.process ( a );

QString szeles = parser.value ( szeles_opt );
QString magas = parser.value ( magas_opt );
QString n = parser.value ( hangyaszam_opt );
QString t = parser.value ( sebesseg_opt );
QString parolgas = parser.value ( parolgas_opt );
QString feromon = parser.value ( feromon_opt );
QString szomszed = parser.value ( szomszed_opt );
QString alapertek = parser.value ( alapertek_opt );
QString maxcella = parser.value ( maxcella_opt );
QString mincella = parser.value ( mincella_opt );
QString cellameret = parser.value ( cellamerete_opt );

qsrand ( QDateTime::currentMSecsSinceEpoch() );

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

w.show();

return a.exec();
}
```

Tanulságok, tapasztalatok, magyarázat... A hangyszimuláció célja a hangyák viselkedésének rekonstruálása. A hangyák a szaglásuk segítségével tájékozódnak, mindíg a legerősebb szagot követik és ha szagot fognak, ők maguk is elkezdenek szagot kibocsátani, így a többi hangya is arra az útra jön. A szimulációhoz szükségünk van a QT5-ös verziójára.

Ez a forráskód a grafikus megjelenítési beállításokat és a hangyák tulajdonságait tartalmazza: A **QCommandLineOption szeles_opt** rész az ablak szélességét, A **QCommandLineOption magas_opt** pedig a magasságát. A **QCommandLineOption hangyaszam_opt**, **QCommandLineOption sebesseg_opt**, **QCommandLineOption parolgas_opt**, **QCommandLineOption feromon_opt** és **QCommandLineOption szomszed_opt** sorok a hangyák tulajdonságait szabályozzák. A **parser** parancs adja hozzá az ablakot a kerethez. A **w.show();** parancs jeleníti meg az ablakot, és a végén a program visszatér az **a.exec()** objektummal.

7.2. Java életjáték

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

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: <https://github.com/ghjbku/DE/blob/master/sajat/eletjatek/Sejtautomata.java>

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

    public Sejtautomata(int szélesség, int magasság) {
        this.szélesség = szélesség;
        this.magasság = magasság;
        rácsok[0] = new boolean[magasság][szélesség];
        rácsok[1] = new boolean[magasság][szélesség];
        rácsIndex = 0;
        rács = rácsok[rácsIndex];
        for(int i=0; i<rács.length; ++i)
            for(int j=0; j<rács[0].length; ++j)
                rács[i][j] = HALOTT;
        siklóKilövő(rács, 5, 60);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
            }
        });
        addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
                    cellaSzélesség /= 2;
                    cellaMagasság /= 2;
                    setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                            Sejtautomata.this.magasság*cellaMagasság);
                    validate();
                } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    cellaSzélesség *= 2;
                    cellaMagasság *= 2;
                    setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                            Sejtautomata.this.magasság*cellaMagasság);
                    validate();
                }
            }
        });
    }
}
```

```
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel = !pillanatfelvétel;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
            várakozás /= 2;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
            várakozás *= 2;
        repaint();
    }
});
```

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

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

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

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

```
public void paint(java.awt.Graphics g) {
    boolean [][] rács = rácsok[rácsIndex];
    for(int i=0; i<rács.length; ++i) {
```

```
        for(int j=0; j<rács[0].length; ++j) {
            if(rács[i][j] == ÉLŐ)
                g.setColor(java.awt.Color.BLACK);
            else
                g.setColor(java.awt.Color.WHITE);
            g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                       cellaSzélesség, cellaMagasság);
            g.setColor(java.awt.Color.LIGHT_GRAY);
            g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                       cellaSzélesség, cellaMagasság);
        }
    }

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

public int szomszédokSzáma(boolean [][] rács,
                           int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 0;
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            if(!((i==0) && (j==0))) {
                int o = oszlop + j;
                if(o < 0)
                    o = szélesség-1;
                else if(o >= szélesség)
                    o = 0;

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

                if(rács[s][o] == állapot)
                    ++állapotúSzomszéd;
            }
    }

    return állapotúSzomszéd;
}
public void időFejlődés() {

    boolean [][] rácsElőtte = rácsok[rácsIndex];
```

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

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

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

        if(rácsElőtte[i][j] == ÉLŐ) {

            if(élők==2 || élők==3)
                rácsUtána[i][j] = ÉLŐ;
            else
                rácsUtána[i][j] = HALOTT;
        } else {

            if(élők==3)
                rácsUtána[i][j] = ÉLŐ;
            else
                rácsUtána[i][j] = HALOTT;
        }
    }
}
rácsIndex = (rácsIndex+1)%2;
}

public void run() {

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

        időFejlődés();
        repaint();
    }
}

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;

}

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;
```

```
}

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

public void update(java.awt.Graphics g) {
    paint(g);
}

public static void main(String[] args) {
    new Sejtautomata(100, 75);
}
}
```

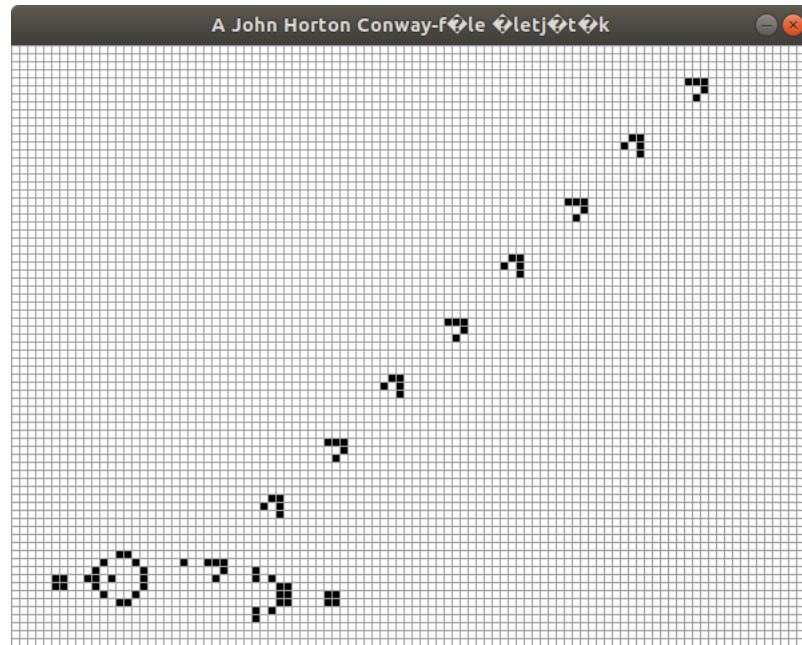
Tanulságok, tapasztalatok, magyarázat... A Java verzió ugyanazt tudja, mint a C++ verzió, persze a programnyelv miatt vannak különbségek a forrásokban. A Java kompatibilitása miatt viszont szinte bárhol lefuttatható a program, ahol jvm található. A programban megtalálható pár billentyű funkció is, melyekkel a program működését befolyásolhatjuk. Az **s** betű megállítja a programot, a **k** az ablakot kicsinyíti, míg az **n** nagyítja. van még a **g** és **az l**, ezek gyorsítják és lassítják a programot.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: [eletjatek/main.cpp](#)



7.3. ábra. Az életjáték futás közben

```
//main.cpp tartalma
#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();
}
```

A main fájl rövid, hiszen csak annyi a dolga, hogy meghívja magát a program-ablakot, melyet meg is tesz ebben a sorban: **SejtAblak w(100,75);**; itt a sejtablak osztályt felhasználva készítünk egy w objektumot, melyeknek a 100 szélesség és 75 magasság paramétereit adjuk meg. Majd a következő sorban a w.show(); függvényhívással megjelenítjük azt ablakot.

```
//sejtablak.h tartalma
#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 sejttár állapotút
    // 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;

    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
```

Ha betekintünk a SejtAblak.h fájlba, láthatjuk magát az osztályt, amit a main.cpp-ben felhasználtunk. Az osztály public ágában találhatunk egy konstruktort, egy destruktort, és két bool típusú változót, amely a sejtek állapotát szimbolizálja, halott, vagy él. A protected ágban megjelenik a ***racsok és a **racs pointer objektumok. Itt inicializáljuk a paintEvent funkciót és a két fő objektumunkat is, a siklót és a siklókilövőt. Ez utóbbi kettő a **racs pointert használja argumentumként, míg a paintEvent a QT-be integrált QPaintEvent osztályban található tagot használja. A fájl elején láthatunk még egy header fájlt, a sejtszal.h-t.

```
//sejtszal.h tartalma
```

```
#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 sejttár 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
```

a SejtSzal egy paraméterezett default konstruktur, mely bekéri a racsok pointer-t, a szélességet, a magasságot, a várakozást és egy sejtAblak típusú sejtablak pointert. Alatta megtalálható a default destruktur az osztályhoz. A protected ágban is láthatjuk a racsok pointert, a szelesseg és magassag változókat inicializálni. Itt deklaráljuk a várakozást, az idoFejlodes funkciót és a szomszedokSzama funkciót, legvégül pedig a sejtAblak* sejtAblak objektum is itt jelenik meg.

Tanulságok, tapasztalatok, magyarázat... A program forráskódja több fájlra esik szét, melyet **make** parancssal tudunk egy futtatható fájl-á konvertálni. Ez az életjáték a siklóKilövő szimulációt valósítja meg.

7.4. BrainB Benchmark

Megoldás video:

Megoldás forrása: [/esport-talent-search/](https://esport-talent-search/)

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Gutenberg!

8.1. Programozási alapfogalmak

[?]

Egy számítógép programozására három nyelvi szintet különböztetünk meg:

- Gépi kód
- assembly szint
- Magas szint

A magas szintű nyelveken megírt algoritmusokat forráskódoknak nevezzük. A forráskódok nyelvtani szabályi a szintaktikai szabályok, míg a jelentésbeli, tartalmi szabályzat a szemantika. Ezeket a kódokat interpreterrel, vagy fordítóprogrammal gépi kóddá kell konvertálni, hogy a processor értelmezni tudja. Egy fordítóprogram tetszőleges nyelvről tetszőleges nyelvre fordít. Amíg ez az egész kódból egy tárgyprogramot készít, addig az interpreter értelezi és rögtön lefuttatja a kódot, programfájl nélkül.

Kifejezések: Két részből állnak, értékből és típusból. Egy kifejezés ezekből az összetevőkből áll:

- operandus: ez egy literál, változó vagy konstans, ez a kifejezés "értéke".
- operátor: ezek a műveleti jelek, aritmetikai/logikai műveletek ↵ végrehajtására.
- kerek zárójelek: a műveleti sorrend befolyásolására.

Léteznek konstans kifejezések, ezek értéke fordításkor eldől, a program futása közben nem változik.

Utasítások:

értékadó utasítás: Ezen típusú utasítás a változók értékét módosítja a ↵ program futása során.

Üres utasítás: ilyenkor a processor egy üres gépi utasítást hajt végre.

Ugró utasítás: A program egyik soráról a másikra ugrik a vezérlés, ↵ általános alak a GOTO.

Elágazásos utasítás (kétfelé ágazás utasítás): A program egy pontján két ↵ lehetőség közül választ a program egy feltétel alapján.

Formája: if...then...else...

Többirányú elágazás: A program egy pontján meghatározott számú opciókból ↵ kiválaszt egyet, amelyet végrehajt.

A választást egy kifejezés értéke szerint határozzuk meg.

Formája: switch(kifejezés) {
case egész_kifejezés: [tevékenység]
case egész_kifejezés: [tevékenység]
default: tevékenység };

A case-ágak értékei különbözzenek.

Ha a case-ágak közül egyik sem egyezik a feltétellel, akkor a default tevékenység hajtódik végre.

Blokk: Egy programegység egy másik program belsejében. Eljárásorientált nyelvekben csak a **hatáskörben** van szerepe. Kétféle hatáskörkezelés létezik, statikus és dinamikus. Előbbi a fordítási időben valósul meg, a fordítóprogram által. A hatáskör csak befelé terjed, kinről nem látni, ha a változót látjuk kinről, akkor az globális változó. A dinamikus hatáskörkezelés futási idő közben zajlik le, azt a rendszer végzi. Ha szabad nevet talál, a hívási láncon keresztül lépked felfelé, amíg a nevet meg nem találja. Az Eljárásorientált nyelvek a statikus kezelést használják.

I/O: A programnyelvek azon eszközrendszere, amely a perifériákkal való kommunikációt hajtja végre, az operatív táróból küld, vagy vár adatokat. Az I/O középpontja az állomány. Ez lehet logikai és fizikai. Az előbbi egy olyan programozási eszköz, amely névvel rendelkezik, és állományjellemzői attribútumként vannak jelen. Fizikai állománynak pedig a hétköznapi, perifériákon megjelenő, adatokat tároló vagy tartalmazó állományt nevezzük.

8.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

A C nyelvben kevés adattípus találhatunk(**char, int, float, double**), de hozzájuk egyfajta minősítők is tartozhatnak:(short(16 bit), long(32 bit) int). ezek Különböző hosszúságú egészeket írnak le. Az ún. "bűvös számok" elkerülésére használhatjuk a #define parancsot, amellyel Szimbolikus állandókat hozhatunk létre. Használata: #define név behelyettesítendő-szöveg-vagy-szám

változóérényességi tartomány: A változók deklarálása általában lokális módon történik, ezek a függvényen kívül nem látszanak, a másik függvény/osztály nem fér hozzájuk. Ha azt szeretnénk, hogy a változónk látható legyen a függvényen kívül is, szükségünk lesz a változó előre beírni egy **extern** szót, ezzel globális változóvá téve az eredetileg lokális változót.

Változók deklarálása: A változók deklarálása 3 féle képpen történhet, lista szerint, szétválasztva, vagy értékadással.

Lista például: char c,t[5]; int i,g,k;

szétválasztva: char c; char t[5]; int i; int g; int k;

értékadással: char c = 'a'; int i=0;

Minden változó előre kitehetünk egy const minősítőt, ha azt szeretnénk, hogy a változó értékét ne lehessen megváltoztatni.

Típuskonverzió: atoi függvény, a char típusú változók értékét integerré konvertálja. ASCII karakterkészlet esetén használható a lower() függvény, ami a nagybetűket kisbetűkké alakítja.

Bitenkénti operátorok: A C nyelvben 6 olyan operátor található, amelyet bitenkénti műveletekre használhatunk. Ezek az operátorok csak char, short, int, long, és ezek előjeles/előjel nélküli formájára használhatóak.

&	bitenkénti ÉS-kapcsolat
	bitenkénti megengedő (inkluzív) VAGY-kapcsolat
^	bitenkénti kizáró (exkluzív) VAGY-kapcsolat
<<	balra léptetés
>>	jobbra léptetés
~	egyes komplementus képzés (unáris)

Regiszter változó: Ezen fajta deklaráció a fordítóprogrammal közli, hogy az adott változóra gyakran lesz szükségünk, ezért regiszterbe helyezzük, ezzel meggyorsítva a működést, és kisebb méretet kapunk. A fordítóprogram nem köteles ezt a kérést teljesíteni, figyelmen kívül is hagyhatja. deklarációs forma: register int c; register char g;

Rekurzivitás: A függvények rekurzívan hívhatják önmagukat, ilyenkor az automatikus változók értékei újramásolódnak az új híváshoz. Ezek az értékek egymástól függetlenül léteznek, nincs hatásuk egymásra.

Makrók: A #define paranccsal nem csak Szimbolikus állandók hozhatók létre. Segítségükkel makrókat is inicializálhatunk, tehát akár egy egész függvényt is egy szóval behelyettesíthetünk. Amikor ez a szó idézőjel nélkül, egymágában megjelenik a kódban, akkor a helyettesítési értéke illesztődik be a kódba.

Pointerek: Ezek olyan változók, amelyek más változók memóriacímét tárolják. C-ben gyakran használják a mutatókat, mivel hatékonyság és tömörség növelő hatása van. Ha van két változónk, char c és char* p, és p = &c; a &-jel (egyoperandusú operátor) hozzárendeli a p-hez a c címét. Ezt úgy szoktuk mondani, hogy a p c-re mutat. Ez az operátor csak változókra és tömbelemekre használható.

Parancssori argumentumok: Lehetőségünk van argumentumok átadására a programunknak a terminálról, ezek az argumentumok a main() függvény *argc* és *argv[]* tagjai. az argc egy számot tartalmaz, ez a szám az argumentumok száma. Az argv[] tömb pedig az összes argumentumot tartalmazza, argv[0] a program neve.

8.3. Programozás

[BMECPP] A C++ nyelv a C nyelv továbbfejlesztése, annak kényelmesebb használatára hivatott. Alapértelmezett függvényargumentumok és függvénynevek túlterhelése segíti a programozók dolgát.

C-ben egy üres paraméterlistával rendelkező függvények bármennyi paramétere lehet, ez C++-ban viszont egy paraméter nélküli függvény. Amíg C-ben megadhatunk típus nélküli függvényeket, (melyek int típusúak lesznek) addig C++-ban nincs Alapértelmezett típus, tehát ez hibát eredményez.

C++-ban a main() függvény Alapértelmezetten renelkezik két paraméterrel, az argc és argv[] paraméterekkel, előbbi az argumentumok számát, utóbbi az argumentumokat tárolja. C++-ban megjelenik a bool típus, amely logikai értékeket vehet fel. Előnye az olvashatóság, valamint operátor túlterhelhetőség.

C-ben a több-bájtos stringek eléréséhez meg kellett hívni a fejlécben az <stddef.h>, <stdlib.h> vagy <wchar.h> fájlokat. Ez C++-ban már beépített típus lett, így meghívását egyszerűen megtehetjük: wchar_t text=L"sss";

objektumok és osztályok

Az objektumorientáltság az 1960-as években kezdődött, és 1990-től terjedt el nagy mértékben. Egy egységbe záró adattstruktúra neve az osztály. Ezen osztálynak az egyedpéldányai az objektumok. Ha azt

szeretnénk, hogy a programban más férhessen hozzá egy bizonyos objektumhoz, akkor azt a valahogy meg kell oldanunk. Ez a védelmi mechanizmus lesz az "adatrejtés". Ezt megtehetjük azzal, hogy az objektum elől egy "private:" kulccsszót írunk. Így csak az osztályon belüli tagok férhetnek hozzá az adathoz. A dinamikus memóriakezelést C-ben a **malloc** és **free** függvénypárossal végeztük. A paraméterek átadása miatt C++-ban már nem is függvény felelős a memóriakazelésért, hanem operátor. Ez az operátor a **new**

```
int* p;
p=new int;
*p=10;
delete p; //a változó használata után felszabadítjuk a helyét a ←
           memóriában
```

Ha szeretnénk, hogy a private:-ban található objektumainkat az osztályon kívül is el tudjuk érni, szükségünk lesz egy **Friend** függvény vagy osztály megadására az osztályunkban.

Konstansok és inline függvények

A konstansokat a "közönséges" változókhöz hasonlóan használhatjuk, viszont ezek értékei nem változnak a program futása során. Bármiféle változtatás az értékkel programhibához vezet.

Léteznek konstans pointerek is, ezeket kétféle módon adhatjuk meg, ha a típus előre írjuk, akkor a mutatott érték válik megváltoztathatatlaná.

`elmelet/testprog.cpp`

```
char tomb[5];
const char* pointer=tomb;
*pointer="g"; //ez fordítási hibához vezet, hiszen ez a mutatott ←
               érték
pointer++; //hiba nélkül lefut
```

A másik változat, amikor a const-ot a pointer neve előre írjuk, ekkor a mutató lesz megváltoztathatatlan.

```
char tomb[5];
char* const pointer =tomb;
*pointer="g"; //lefut
pointer++; //fordítási hibát jelez
```

Ezek kombinálása is lehetséges.

Inline függvények

Ezen függvények esetében a fordító behelyettesíti a hívás helyére az inline-ban megadott kód részét, ezzel gyorsítva az általános függvényhívás menetét.

I/O alapok

operátor túlterhelés

A C++ nyelvben néhány új operátor lett bevezetve a C-hez képest. Ilyen például a hatókör operátor(:), a pointer-tag ooperátor(.* és ->*).

9. fejezet

Helló, Schwarzenegger!

9.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

```
# 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.
# ←
=====

"""Builds the MNIST network.

Implements the inference/loss/training pattern for model building.

1. inference() - Builds the model as far as is required for running the ←
   network
forward to make predictions.
2. loss() - Adds to the inference model the layers required to generate ←
   loss.
3. training() - Adds to the loss model the Ops required to generate and
apply gradients.
```

```
This file is used by the various "fully_connected_*.py" files and not meant ←
    to
be run.

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

import math

import tensorflow as tf

# The MNIST dataset has 10 classes, representing the digits 0 through 9.
NUM_CLASSES = 10

# The MNIST images are always 28x28 pixels.
IMAGE_SIZE = 28
IMAGE_PIXELS = IMAGE_SIZE * IMAGE_SIZE


def inference(images, hidden1_units, hidden2_units):
    """Build the MNIST model up to where it may be used for inference.

    Args:
        images: Images placeholder, from inputs().
        hidden1_units: Size of the first hidden layer.
        hidden2_units: Size of the second hidden layer.

    Returns:
        softmax_linear: Output tensor with the computed logits.
    """
    # Hidden 1
    with tf.name_scope('hidden1'):
        weights = tf.Variable(
            tf.truncated_normal([IMAGE_PIXELS, hidden1_units],
                               stddev=1.0 / math.sqrt(float(IMAGE_PIXELS))),
            name='weights')
        biases = tf.Variable(tf.zeros([hidden1_units]),
                            name='biases')
        hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)
    # Hidden 2
    with tf.name_scope('hidden2'):
        weights = tf.Variable(
            tf.truncated_normal([hidden1_units, hidden2_units],
                               stddev=1.0 / math.sqrt(float(hidden1_units))),
            name='weights')
        biases = tf.Variable(tf.zeros([hidden2_units]),
                            name='biases')
        hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
```

```
# Linear
with tf.name_scope('softmax_linear'):
    weights = tf.Variable(
        tf.truncated_normal([hidden2_units, NUM_CLASSES],
                            stddev=1.0 / math.sqrt(float(hidden2_units))),
        name='weights')
    biases = tf.Variable(tf.zeros([NUM_CLASSES]),
                         name='biases')
    logits = tf.matmul(hidden2, weights) + biases
    return logits

def loss(logits, labels):
    """Calculates the loss from the logits and the labels.

    Args:
        logits: Logits tensor, float - [batch_size, NUM_CLASSES].
        labels: Labels tensor, int32 - [batch_size].
    Returns:
        loss: Loss tensor of type float.
    """
    labels = tf.to_int64(labels)
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
        logits, labels, name='xentropy')
    loss = tf.reduce_mean(cross_entropy, name='xentropy_mean')
    return loss

def training(loss, learning_rate):
    """Sets up the training Ops.

    Creates a summarizer to track the loss over time in TensorBoard.

    Creates an optimizer and applies the gradients to all trainable variables ←
    .

    The Op returned by this function is what must be passed to the
    `sess.run()` call to cause the model to train.

    Args:
        loss: Loss tensor, from loss().
        learning_rate: The learning rate to use for gradient descent.
    Returns:
        train_op: The Op for training.
    """
    # Add a scalar summary for the snapshot loss.
    tf.scalar_summary(loss.op.name, loss)
    # Create the gradient descent optimizer with the given learning rate.
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
# Create a variable to track the global step.
global_step = tf.Variable(0, name='global_step', trainable=False)
# Use the optimizer to apply the gradients that minimize the loss
# (and also increment the global step counter) as a single training step.
train_op = optimizer.minimize(loss, global_step=global_step)
return train_op

def evaluation(logits, labels):
    """Evaluate the quality of the logits at predicting the label.

    Args:
        logits: Logits tensor, float - [batch_size, NUM_CLASSES].
        labels: Labels tensor, int32 - [batch_size], with values in the
            range [0, NUM_CLASSES).

    Returns:
        A scalar int32 tensor with the number of examples (out of batch_size)
        that were predicted correctly.
    """
    # For a classifier model, we can use the in_top_k Op.
    # It returns a bool tensor with shape [batch_size] that is true for
    # the examples where the label is in the top k (here k=1)
    # of all logits for that example.
    correct = tf.nn.in_top_k(logits, labels, 1)
    # Return the number of true entries.
    return tf.reduce_sum(tf.cast(correct, tf.int32))
```

Tanulságok, tapasztalatok, magyarázat... Ez a program egy számítógépet betanító program, futtatásához a TensorFlow szükséges. Alacsony felbontású képeket analízáltat a számítógéppel, majd ezen képek alapján megpróbál egy másik képről hasonlóságot találni, és kitaláni, hogy mi látható a képen. Itt konkrétan kézzel írt számokról dönti el, hogy a képen melyik szám látható. A betanítási folyamat hossza függ a program bonyolultságától, de ez a program egyszerűnek számít, tehát elég könnyen 90% fölé mehet a pontossága, és a betanítási idő is rövid. A **def** kulcsszóval definiálhatunk függvényeket. Az első ilyen függvény az "inference", amely három argumentumot vár: az **images**, **hidden1_units** és **hidden2_units** argumentumokat. Az első a képek helyét jelenti az **input()** függvényből, a második és a harmadik pedig a rejtegett rétegek méretét jelenti. A függvényünk visszatérési értéke a **softmax_linear**, ez a kiszámolt "logit"-eket tárolja.

A második függvény a **loss(logits,labels)**, ez a logitekből és címkekből kiszámítja a veszteséget. A harmadik függvény felel a betanításért, ez a **training(loss,learning_rate)**. Az utolsó függvény az **evaluation**, amely visszaadja, hogy milyen sikereséggel találta el a program a helyes számot.

9.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

```
# 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)
```

Tanulságok, tapasztalatok, magyarázat... 32 bites képeket használ a betanulási fázisban, ezek egyfajta "neurális hálón" haladnak keresztül, és a gép a csomópontokban megadott feltételek szerint dönt a kép azonosításáról. A végén százalékos formában kiírja a pontosságot. Ez kissébb erőforrásigényű programoknál 90+% fölé mehet, amely az embernél jobb hatékonyságot jelent.

A legelső függvényünk a deepnn(x), ez a függvény egy gráfot épít ki a mély hálónknak, amely a számokat osztályozza. Az X argumentum a dimenziók száma. Visszatérési értékként az y_conv ls a keep_prob térnek vissza. Az y változó a szám alakját adja vissza, 0 és 9 között változhat ez az érték.

Következő funkció a conv2d(x,W). Ez egy 2d konvolúciós réteget ad vissza.

Az ez alatt található funkció a max_pool_2x2, ez a függvény csökkenti a mintákat 2x. Ezután jön a weight_variable(shape) függvény, amely a nevéről is kideríthetően súlyozást ad a képhez, egy shape alapján. Végül a bias_variable, amely ugyancsak a shape alapján állít be bias-t.

A legvégén található egy main() függvény, amely először importálja az adatokat, majd elkészíti a modellt, a veszteséget kiszámítja és végül felépíti a gráfot.

9.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... Sajnos nem volt alkalmam beszerezni a játékot.

10. fejezet

Helló, Chaitin!

10.1. Iteratív és rekurzív faktoriális Lisp-ben

Az SMNIST-el kiváltva

Megoldás videó:

Megoldás forrása:

10.2. Gimp Scheme Script-fu: króm effekt

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

Megoldás videó: https://youtu.be/OKdAkl_c7Sc

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

Tanulságok, tapasztalatok, magyarázat...

10.3. 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_hackelese_a_scheme_programozasi_nyelv

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala

III. rész

Második felvonás

**Bátf41 Haxor Stream**

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

11. fejezet

Helló, Berners-Lee!

11.1. Olvasónaplók

C++:Benedek Zoltán,Levendovszky Tihámér Szoftverfejlesztés C++ nyelven Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 ebből a kettőből egy pár oldalas összehasonlító esszé. Python:Forstner Bertalan,Ekler Péter,Kelényi Imre:Bevezetés a mobilprogramozásba Gyors prototípus-fejlesztés Python és Java nyelven(35-51 oldal) -ebből 1-oldalas élmény-napló

C++ és Java összehasonlítás

java-ban minden metódusnak kell visszatérési érték típust adni. C++-ban viszont nem muszáj, hiszen alapértelmezetten int típust feltételez a program, ha nem adunk neki típust. A cpp-tól eltérően Java-ban String[] tömb tárolja az argumentumokat ezzel szemben a C ill. C++-ban argv és argc változó együttes tárolja azokat argv az argumentumokat tároló vektor, az argc pedig az argumentumok számát tároló változó. logikai típus C++-ban bool, Java-ban boolean.

basic io művelet java-ban System.out.Println(), C++-ban std::cout parancs használatával történik. ezek listázása is eltérő, java-ban a pascal-hoz hasonlóan + jellel fűzünk össze változókat ill. szöveget, Cpp-ban pedig >> jel váltja fel a + szerepét Továbbá Java-ban nincs operátor túlterhelés!

Java-ban a karakterkészlet defaultban utf-8, c++-ban include-olni kell egy library-t, hogy utf-8-as karaktereket használhassunk, anélkül nehézkes a használatuk. a konstansok megadása Java-ban a "final" kulcsszó használatával lehetséges, c++-ban ezt a const-al tehetjük meg Példák:

```
final static double pi=3.14;
```

```
const double pi=3.14;
```

Az objektumok elemeire hivatkozhatunk, az "**objektum neve**".**"elem neve"** módon. Ha az elem-nek egy elemére szertnénk hivatkozni, azt hasonló modon kell megtenni. A java-ban nincs explicit mód a memóriahez felszabadítására, a pointerek NULL-ra állítását tehetjük meg, amit a "garbage collector" később eltűntet. nincs dekonstruktur, ezzel szemben C-ben és C++-ban van dekonstruktur, amit az objektum elején elhelyezett "~" jellel "állítunk elő".

```
~LZWBinFa ()  
{  
}  
}
```

Továbbá C++-ban lehetőségünk van(és kell is) a pointerek által lefoglalt memóriahelyek felszabadítására 3 módszerrel.

```
delete ptr;  
ptr=NULL;  
//vagy pedig  
free(ptr);
```

C-ben nincs lehetőség a "**delete ptr;**" módszerre. A pointer NULL-ra állítása viszont nem szabadítja fel a helyet, csak ún. "árvát" hoz létre, mivel a C++ nem "garbage collector" nyelv. A lefoglalt terület még fennáll, de nem lehet elérni, és egy memory leak képződik. Ezek a memória leak-ek csak akkor okoznak nagyobb gondot, ha elfogy a memória, ekkor a program preemptív kilép, "crash-el".

Ha egy objektumot a **new()** metódussal hoztunk létre, akkor a **delete** paranccsal szabadítsuk fel a helyet, ha a **malloc()-ot** használtuk memória lefoglaláshoz, akkor a **free()**-vel szabadítjuk fel a memóriában foglalt helyet. Fontos tudni, hogy a **free()** nem hívja a destructor-t, azt csak a **delete** teszi meg!

Mindkét programnyelvben megtalálható az automatikus/implicit típus konverzió. Ha a fordító program a vártnál eltérő típusú adatot kap, azt automatikusan megpróbálja átkonvertálni a várt típusra. Ez nem minden lehetséges! Például az egyénileg létrehozott típusokat nem tudja átkonvertálni.

A try-catch hibakezelő metódus minden nyelvben elérhető. Ezekről egy-egy példakód itt:

Java

C++

A java nyelvben a tömb típus egy igazi típus, amíg c++-ban csak egy mutató típus. minden nyelvben 0-val kezdődik az indexelés a tömbben, Továbbá az enum típus is jelen van. Java-ban a pont minden esetben a tagok elérésére szolgál, és a C++-tól eltérően itt nincs megkülönböztető jelölés osztálytagok elérésénél(C++-ban :: operátor)

A C és C++-tól eltérően a Java-ban nincs GOTO utasítás, azzal a címszóval lett elhagyva, hogy ezáltal biztonságosabb és megbízhatóbb programokat kapunk. a korábbi goto-val megoldott problémákra új megoldások vannak: ciklus elhagyása a **break** utasítással történik, a ciklus folytatása a **continue** utasítással. Java-ban a legkissemőbb önálló egységek az osztályok.

Python könyv élmény-napló

A Python **magasszintű, általános célú nyelv**, szkriptnyelvként szokták emlegetni. A kódokat egy futtatókörnyezeten keresztül futtatják általában, viszont vannak már kísérleti stádiumban natív kódot generáló fordítóprogramok is. Mind a procedurális, és az objektumorientált programozást támogatja Könyvátárnak mérete a nyelv egyik erősségeinek mondható, melyben még http támogatás is megtalálható. C, cpp nyelven készült modulok is egyszerűen adhatók hozzá a környezethez. Népszerűségét az egyszerűségeknek köszönheti, **szinte bármilyen feladatot meg lehet oldani a nyelvben**, viszont elsősorban kliensszoftverek készítésére alkalmazzák. A mobileszközökön is futtatható, írható python kód, viszont ez nem újdonság, hiszen java ill. C++ kódokat is futtathatunk manapság.

A **Symbian OS** mobiltelefon operációs rendszer egyike a napjainkban legtöbbet alkalmazott operációs rendszereknek. Mind C++-ban, Python-ban és Java-ban is írhatónk rá kódokat, viszont a rendszer felett még ott van egy GUI(Grafikus felhasználói felület), ezekből manapság 2 típus ismert: **S60(korábban Series60)** és az **UIQ**. Hasonló funkcionális lehető fel minden rendszerben, viszont ennek ellenére a kettő nem kompatibilis egymással, az egyikre írt programok nem fognak a másikon el futni. Az operációs rendszer alapfunkcióiban azonban megegyezik, tehát csak az UI-hez kötődő részeket kell külön megírnunk kétféléképpen.

A **Windows Mobile** pedig a Microsoft mobiltelefonokhoz fejlesztett operációsrendszer, amely a Windows CE rendszeren alapul. A Symbian-hoz képest több lehetőségünk van a programnyelvek használatát illetően. A C++, Python és Java-n kívül natív alkalmazásokat készíthetünk C nyelven is, és persze az ugyancsak Microsoft által fejlesztett .NET compact framework technológiával egyaránt. Ez utóbbi a Microsoft .NET telefonokra kifejlesztett változata.

A Harmadik operációsrendszer mobiltelefonokra a nyílt forráskódú, Linux-ra épülő **Maemo**, ez a rendszer a Nokia fejlesztése alatt áll. A rendszer nagyrészen az internetes elérésekre, megoldásokra fókuszál, viszont az alapvető funkciókat ezen a rendszeren is el tudjuk végezni. Jelenleg a Bluetooth és WLAN funkciókat támogatja, de már a Wimax támogatását is híresztelik.

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! [Prog1_5.pdf](#)

Megoldás forrása:[github link](#)

```
454     * and one call to {@code StrictMath.sqrt}.
455     *
456     * @return the next pseudorandom, Gaussian ("normally") distributed
457     *         {@code double} value with mean {@code 0.0} and
458     *         standard deviation {@code 1.0} from this random number
459     *         generator's sequence
460     */
461     synchronized public double nextGaussian() {
462         // See Knuth, ACP, Section 3.4.1 Algorithm C.
463         if (haveNextNextGaussian) {
464             haveNextNextGaussian = false;
465             return nextNextGaussian;
466         } else {
467             double v1, v2, s;
468             do {
469                 v1 = 2 * nextDouble() - 1; // between -1 and 1
470                 v2 = 2 * nextDouble() - 1; // between -1 and 1
471                 s = v1 * v1 + v2 * v2;
472             } while (s >= 1 || s == 0);
473             double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
474             nextNextGaussian = v2 * multiplier;
475             haveNextNextGaussian = true;
476             return v1 * multiplier;
477         }
478     }
479     /**
480      * Serializable fields for Random.
481      */
482 }
```

12.1. ábra. A JDK kód

Amint láthatjuk, a változók elnevezésén kívül és **StrictMath,nextDouble** metódusokon kívül az általunk írt kód megegyezik. Mi ugyanis StrictMath helyett a Math metódust használtuk a gyökvonás és logaritmus képzésnél.

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

A fő eltérés a két metódus között(azon kívül, hogy a StrictMath-ban hiperbolikus és egyéb függvények is elérhetők) az, hogy a **StrictMath**-nál ha meghívunk egy függvényt, annak ugyan azt az értéket kell visszaadnia például x86-os lebegőpontos változónál, mint SPARC lebegőpontos-nál. Ezzel szemben a Math megengedi, hogy a pontosságért cserébe gyorsabban leforduljon a programunk.(na persze a pontossági eltérés nem számottevő a jelen programunk megírásánál)

```
public class polargenerator
{
```

Amint láthatjuk, az egész programkód egy osztályba van elhelyezve, ez a Java-nak egy sajátossága.

```
boolean nincstar=true;
double tarolt;
```

A forráskód elején inicializálunk két változót, a **nincstar** változó egy logikai változó, amely megmondja, hogy van-e eltárolva adat. Ezzel szemben a **tarolt** változó hordozza majd a tárolni kívánt értéket.

```
public double kovetkezo()
{
    if(nincstar)
    {
        double a1,a2,b1,b2,c;
```

Itt pedig egy kovetkezo nevű metódust hozunk létre, ha a nincstar értéke igaz, akkor végrehajtódik az if-ben leírt változók deklarálása.

```
do{
    a1=Math.random();
    a2=Math.random();
    b1=2*a1-1;
    b2=2*a2-1;
    c=b1 * b1 + b2 * b2;
    }while(c>1);
```

Ez a következő pár sor egy hárultesztelős ciklus. Először az a1,a2 változók értékét randomizáljuk, majd a b1,b2 változókban az előbbi két értéket megduplázzuk és csökkentjük az értéket 1-el. a c változó értékét a b1 és b2 változók négyzetével tesszük egyenlővé. Ez addig fut, amíg a c értéke nagyobb lesz, mint 1.

```
double r=Math.sqrt((-2*Math.log(c))/c);
tarolt=r*b2;
nincstar=!nincstar;
return r*b1;
```

ezekben a sorokban létrehozunk egy r változót, és annak az értékéül a Math.sqrt((-2*Math.log(c))/c) függvény eredményét adjuk. a tárolandó érték az r változó és a b2 változó szorzata lesz, az értékkadás után pedig a nincstar értékét negáljuk. végül visszatérési értékként a r és a b1 szorzatát adjuk meg.

```
else
{
```

```
nincstar!=nincstar;  
    return tarolt;  
}
```

Az else ágban csak annyi a dolgunk, hogy a nincstar értékét negáljuk, és a tárolt értéket adjuk meg vissza-térési értékként. Végezetül pedig kiíratjuk a main-ben.

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

12.2. "Gagyi"

Az ismert formális **while(x <=t && x>=t && t!=x);** tesztkérdéstípusra adj a szokásosnál "mélyebb" választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x és t értékekkel pedig nem! A példát építsd a JDK Integer.java forrására, hogy a 128-nál inkluzív objektum példányokat poolozza! [facebook post](#)

Megoldás forrása:[github link](#)

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

Amint láthatjuk, az első két tagja a ciklus feltételeknek a két változó értékét hasonlítja össze . A harmadik viszont (**t!=x**) a változók referenciáját, ami az integer-cache miatt -128 és 127 között ugyanaz lesz. Tehát ha a két objektum értéke nem haladja meg ezt a határt, akkor a referenciájuk ugyan oda mutat majd. A gagyi.java fájl ezért végtelen ciklust képez, hiszen a -129 már nincs benne a cache-ben, ezért a t!=x mindíg igaz lesz. A gagyi2.java viszont a -128 értékkel dolgozik, ami benne van a cache-ben, tehát a ciklus nem jön létre.

Érdekesség még, hogy a Java6 óta vezették a felső érték bindelését, amely a

java.lang.Integer.IntegerCache.high segítségével a programozó által is állítható lett. Fontos tudni viszont, hogy a maximális érték legalább 127 kell, hogy legyen. Ha az érték kevesebb, akkor egy AssertionError hibakódöt kapunk.

```
windsake@windsake-pc:~/Desktop/sajat/prog2/gagyi
[bhax-textbook-fdl.pdf to commit]
bhax-textbook-fdl.xml
bhax-textbook-feladatok2-arroway.xml
bhax-textbook-feladatok2-berners.xml
bhax-textbook-feladatok2-liskov.xml
bhax-textbook-feladatok2.xml
bhax-textbook-feladatok-caesar.xml
bhax-textbook-feladatok-chaitin.xml
bhax-textbook-feladatok-chomsky.xml
bhax-textbook-feladatok-conway.xml
bhax-textbook-feladatok-gutenberg.xml
bhax-textbook-feladatok-mandelbrot.xml
bhax-textbook-feladatok-schwarzenegger.xml
bhax-textbook-feladatok-turing.xml
bhax-textbook-feladatok-welch.xml
bhax-textbook-feladatok.xml
bhax-textbook-intro.xml
bhax-textbook-motto.xml
bhax-textbook-pre.xml
bhax-textbook-revhistory.xml
bhax-textbook-subtitle.xml
bhax-textbook-titleabbrev.xml
[windsake@windsake-pc prog2]$ cd gagyi/
[windsake@windsake-pc gagyi]$ ls
gagyi2.java gagyi.java
[windsake@windsake-pc gagyi]$ javac gagyi.java
[windsake@windsake-pc gagyi]$ java gagyit rev-parse master
-129
-129
^C[windsake@windsake-pc gagyi]$ javac gagyi2.java
[windsake@windsake-pc gagyi]$ java gagyi2 remote --verbose
-128
-128
[windsake@windsake-pc gagyi]$ 
```

12.2. ábra. A gagyi futás

12.3. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-el leáll, ha nem követjük a Yoda conditions-t! [yoda wiki](#)

Megoldás forrása:[github link](#)

```
public class yoda
{
    String i;
    public yoda()
    {
        i=null;

    }
    public static void main(String[] args)
    {
        yoda f=new yoda();

        if("5".equals(f.i)) //if(f.i.equals("5"))
        {
```

```
System.out.println("t");
}
else
{
System.out.println("f");
}
}
```

A fenti programkód a yoda kondíciós megoldást mutatja, ami annyit tesz, hogy a konstanst rakjuk a ciklus-magban az első helyre, amelyel a NullPointerException elkerülhetjük. A program maga annyit tesz, hogy összehasonlítja a konstans számot a yoda típusú f objektumunk i tagváltozójával. a **yoda f=new yoda();** sor példányosítja a yoda osztályt.

```
Terminal - windsake@windsake-pc:~/Desktop/sajat/prog2/yoda
File Edit View Terminal Tabs Help
bhax-textbook-feladatok-welch.xml      output.xml
bhax-textbook-feladatok.xml            polarthing.cpp
bhax-textbook-intro.xml                README.md
bhax-textbook-motto.xml               std_lib_facilities.h
bhax-textbook-pre.xml                 szohossz.c
bhax-textbook-revhistory.xml          'Ü' szöveges dokumentum.txt'
bhax-textbook-subtitle.xml           yoda
bhax-textbook-titleabbrev.xml         z3a7_gyoker.cpp
[windsake@windsake-pc prog2]$ cd yoda/
[windsake@windsake-pc yoda]$ ls
yoda.class  yoda.java
[windsake@windsake-pc yoda]$ java yoda
f
[windsake@windsake-pc yoda]$ gedit yoda.java
(
(org.gnome.gedit:4925): GLib-WARNING **: 00:14:31.372: Calling org.xfce.Session.M
anager.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: No
such method "Inhibit"
^C
[windsake@windsake-pc yoda]$ javac yoda.java
[windsake@windsake-pc yoda]$ java yoda
Exception in thread "main" java.lang.NullPointerException
        at yoda.main(yoda.java:13)
[windsake@windsake-pc yoda]$ rm scratch
```

12.3. ábra. A yoda futás

Ha a két érték megegyezik, akkor kiír az alapértelmezett output-ra egy t betűt, egyébként pedig egy f betűt. A yoda kondíció nélkül egyik sem történik meg, hiszen az i értéke **null**, tehát azt várnánk, hogy NullPointerException hibakódot látunk viszont. A kikommentelt részről beillesztve pedig a program lefutásánál meg is kapjuk a nullpointer exception-t. A kondíciót alkalmazva ugyebár egy f betű kerül ki a standard outputra.

12.4. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből:<http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben:https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei

Megoldás forrása:[github link](#)

```
[windsake@windsake-pc prog2]$ cd pibbp/
[windsake@windsake-pc pibbp]$ ls
PiBBP.class PiBBP.java piBBP.xml
[windsake@windsake-pc pibbp]$ java PiBBP
6C65E5308[windsake@windsake-pc pibbp]$
```

windsake@windsake-pc:~/Desktop/sajat/prog2/pibbp

12.4. ábra. A piBBP futás

a **public PiBBP(int d)** rész a $d+1$ hexadecimális jegytől számítja ki a jegyeket , azon belül a 16^d pi képlet kiszámítása.

```
public PiBBP (int d) {

    double d16Pi = 0.0d;

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

A függvény bekér majd egy számot a d változóban, melyből készíteni fog double típusú számokat(d16S1t..d16S6t)

```
public static void main(String args[]) {
    System.out.print(new PiBBP(1000000));
}
```

A main függvényben láthatjuk is, hogy meghívjuk a PiBBP-t és a d változónak az "1millió" értéket adjuk.

A **public double d16Sj(int d, int j)** eljárás a $\{16^d \text{ Sj}\}$ részlet kiszámítását végzi. Ehhez meghívja a n16modk függvényt, amely a **n16modk(d-k, 8*k + j)** paramétereket kapja. A paraméterek:

k: for(int k=0; k<=d; ++k).

d: A PiBBP argumentuma, egyébként a d16Sj argumentum listájának az első eleme, de értékként a PiBBP d-jét kapja meg a függvény.

j: A második argumentuma a d16Sj függvénynek.

public long n16modk(int n, int k) végzi a hatványozás modulóját, itt n a kitevő, k a modulus.

```
public String toString() {

    return d16PiHexaJegyek;
}
```

ez az eljárás a kiszámolt hexadecimális értékeket konvertálja át stringbe, majd visszaadja az értéket.

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megséríti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Megoldás forrása:[github link](#)

Először nézzük a liskov elvet követő programot:

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

itt nincs repul metódus a Madar osztály tagjaként, tehát nem örökli majd a pingvinünk a repul objektumot. Ez a RepuloMadar osztálytípusosított sas osztály-nak az egyedisége.

```
class Program {  
public:  
    void fgv ( Madar &madar ) {  
        // madar.repul(); a madár már nem tud repülni  
        // s hiába lesz a leszármazott típusoknak  
        // repül metódusa, azt a Madar& madar-ra úgysem lehet hívni  
    }  
};  
  
class RepuloMadar : public Madar {  
public:  
    virtual void repul() {};  
};  
  
class Sas : public RepuloMadar  
{};
```

```
class Pingvin : public Madar
{ };
```

itt látható a madar.repul funkció kikommentelve, tehát nem öröklődik a pingvinünk-nél, ezért az fgv függvény nem fogja repültetni azt.

A liskov elvet sértő program pedig így néz ki:

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

// ez a két osztály alkotja a "P programot" az LPS-ben
class Program {
public:
    void fgv ( Madar &madar ) {
        madar.repul();
    }
};
```

Itt nincs kikommentelve a madar.repül metódus, ezzel a későbbiekben megsérítjük majd az elvet.

```
class Sas : public Madar
{ };

class Pingvin : public Madar // ezt úgy is lehet/kell olvasni, hogy a ←
    pingvin tud repülni
{ };

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

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin ); // sérül az LSP, mert a P::fgv röptetné a ←
        Pingvint, ami ugye lehetetlen.

}
```

Itt pedig látható, hogy a sas-nak nincs külön alosztály generálva, a RepuloMadar osztály, ehelyett minden állat a madar osztály tagjaként van hitelesítve.

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)

A java forrás: **A forrás Kovács Ferencz tulajdona.**

```
class szulo
{
protected int m_age;
protected String m_name;
    public void setAge(int age) {
        m_age = age;
    }
    public void setName(String name) {
        m_name = name;
    }
    public int getAge() {
        return m_age;
    }
}
class gyerek extends szulo
{
    public String getName() {
        return m_name;
    }
}
class szuloGyerek
{
    public static void main (String args[])
    {
szulo s = new gyerek();
s.setName("Apuci");
s.setAge(60);
gyerek gy = new gyerek();
s.setName("Laci");
s.setAge(15);

System.out.println(gy.getName() + " " + s.getName());
    }
}
```

Látható, hogy a program három osztályra bomlik szét, az első, a szülő osztály, itt létrehozunk két protected változót, az m_name és m_age változókat. Továbbá itt inicializálunk két metódust, a setAge és setName-et. A setAge egy int típusú adatot kér be, majd a protected m_age változó értékét egyenlővé teszi a bekért értékkel. Ugyan ez történik a setName-ben, azzal a különbséggel, hogy ez a függvény string adatot vár. Van még továbbá egy getAge függvényünk, amely az m_age változó értékét adja vissza, ezzel biztosítva, hogy az értéket használhassuk az osztályon kívül is.

Következő osztály a gyerek. ez egy rövid kis osztály, egyetlen tagja a getName függvény, amely megpróbálja elérni a szülő osztály m_name változóját, de persze ez nem sikerül majd, mivel az protected.

A harmadik osztályban teszteljük az elméletünket, először feltültjük a változók értékeit a set függvények segítségével, majd megpróbáljuk lekérni a nevet a `System.out.println(gy.getName() + " " + s.getName());` sorral. Amint azt látni fogjuk, ez nem sikerül.

The screenshot shows a Java IDE interface. The top bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. The title bar says "szulosert.java - prog2 - Code - OSS". The left sidebar has sections for EXPLORER, OPEN EDITORS, and PROG2. In the OPEN EDITORS section, there are files: bhax-textbook-feladatok2-liskov.xml (M), szulosert.cpp (liskov) (U), szulosert.java (liskov) (U), liskovsert.cpp (liskov) (U), and liskovrafigyel.cpp (liskov) (U). In the PROG2 section, there are files: liskovrangyel.cpp, liskovsert.cpp, negy.png, nohup.out, sajat.cpp, szulo_Gyerek_98oldal.png, szulo.class (U), szulosert.cpp (U), and szulosert.java (U). The main area shows the code for szulosert.java:

```
14 }
15 class gyerek extends szulo
16 {
17     public String getName() {
18         return m_name;
19     }
20 }
21 class szuloGyerek
22 {
23     public static void main (String args[])
24 }
```

The terminal below shows the output of the run command:

```
tempCodeRunnerFile
tempCodeRunnerFile.java:32: error: cannot find symbol
    System.out.println(gy.getName() + " " + s.getName());
                                         ^
      symbol:   method getName()
      location: variable s of type szulo
1 error
[Done] exited with code=1 in 3.119 seconds
```

13.1. ábra. A szülőgyerek futás

13.3. Anti OO

A BBP algoritmussal 4 a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10 6, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/hu/tartanitok-javat/apas03.html#id561066>

Megoldás forrása:[github link](#)

	A	B	C	D
1	nyelv/jegy	6	7	8
2	C	2.883552	33.840	399.085
3	C++	2.88116	33.524	388.415
4	C#	2.799081	32.420	372.392
5	JAVA	2.609	30.251	349.842

13.2. ábra. A 4 nyelv futás

```
[windsake@windsake-pc liskov]$ mcs anti_c_sharp.cs
[windsake@windsake-pc liskov]$ mono anti_c_sharp.exe
6
2.841677
[bhax-textbook-feladatok-turing.xml]
[bhax-textbook-feladatok-welch.xml]
[windsake@windsake-pc liskov]$ ./anti_c
6
2.924904
[bhax-textbook-feladatok2-arroway.xml]
[windsake@windsake-pc liskov]$ ./anti_cpp
6
3.386714
[bhax-textbook-feladatok2-mandelbrot.xml]
[windsake@windsake-pc liskov]$ java anti_java
6
2.617
[bhax-textbook-intro.xml]
[bhax-textbook-motto.xml]
[bhax-textbook-pre.xml]
```

13.3. ábra. A 4 nyelv futás

Mint láthatjuk, a JAVA mind 3 esetben a legjobban teljesített a számítások során. Különös viszont, hogy minél kisebb számjegyet keressük a pi-nek, annál gyorsabban végez a java a többiekhez képest. Második helyen a C# volt, nem sokkal lassabb, mint a Java, és a legutolsó a C nyelv. minden 10-ed után kb. 10-szeresére nőtt a futási idő. A C fordításánál ügyelni kell arra, hogy a -lm kapcsolót is hozzáadjuk az argumentumok listájához, mivel anélkül nem fog lefordulni a program. A hibakód a következő:

```
/usr/bin/ld: /tmp/ccKOKIZA.o: in function `d16Sj':
anti_c.c:(.text+0xfe): undefined reference to `floor'
/usr/bin/ld: /tmp/ccKOKIZA.o: in function `main':
anti_c.c:(.text+0x1ff): undefined reference to `floor'
/usr/bin/ld: anti_c.c:(.text+0x227): undefined reference to `floor'
collect2: error: ld returned 1 exit status
```

A gcc dokumentációban csak annyi található az -lm kapcsolóról, hogy az -l kapcsoló paramétere az m, ez egy úgynevezett megosztott library, amely osztályokra van bonta, és objektumokat tartalmaz. A mi esetünkben az m könyvtár tartalmát kívánjuk használni, amelyben a floor függvény található. Ez a math lib. Ha includeoljuk a math.h headert, akkor erre a kapcsolóra is szükségünk lesz. Ez C++-ban már nem szükséges, elég csak az include.

13.4. 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)!

A ciklomatikus komplexitás számításakor a beágyazott ciklusok azonos súllyal vannak számolva, mint a külső ciklusok. Viszont valójában a belső ciklusok jelenlétével a programkód megértése nehezebb lesz. Attól függetlenül, hogy a ciklusok száma szerint a program bonyolultsága alacsony kéne, hogy legyen, még lehet bonyolult a forráskód.

a kód részlet saját tulajdonban áll.

```
void fight(int intel, int d)
{
    int i = 0;
    int j = 0;
    int k = 0;
    if (intel == 1)
    {
        cout << "you remember the paper you found had a bear drawing on it with ←
            three numbers: 5 2 4\n";
    }
    else if (intel == 0)
    {
    }
    cout << "the bear attacks first with a rush attack\nyou: 1.defend\n 2. ←
        side step to the right and try to hit its knee\n 3. back step\n 4. ←
        slash it with your sword \n5.side step to the left\n";
    cin >> i;
    if (i == 1)
    {
        cout << "you can't defend without a shield, you died.";
    }
    else if (i == 2)
    {
        cout << "you try to dodge by sidestepping to right, but the bear ←
            somehow knows you would do that\nit slashes you and deals critical ←
            damage\nyou died.";
    }
    else if (i == 3)
    {
        cout << "you try to backstep, but why? \nthe bear doesn't even have to ←
            turn anywhere, it easily kills you.\n";
    }
    else if (i == 4)
    {
        cout << "you slash the bear with your dagger as it reaches you\nyou ←
            dealt almost no damage.\nthe bear bites you with unbearable strength ←
            \nyou died";
    }
    else if (i == 5)
    {
        d++;
        cout << "you sidestep to the left,the bear, surprised lounges to the ←
            right.\nit hits nothing but air.\nthe bear,enraged even further try ←
            to attack you from the right.\n[decision making " << d << "]\\nyou: " ←
```

```
    ;
cout << "1.defend\n 2.side step to the right and try to hit its knee\n  ↵
      3. back step\n 4.slash it with your sword \n5.side step to the left\n  ↵
      n";
cin >> j;
if (j == 1)
{
    cout << "you can't defend without a shield, you died.";
}

if (j == 2)
{
    d++;
    cout << " you sidestep to the right,the bear surprised again, you use ↵
          the momentum to slash its knees.\nthe bear falls to the ground.\n ↵
          [decision making " << d << "]\\n ";
}

cout << "\nyou: 1.defend\n 2.side step to the right and try to hit ↵
      its knee\n 3. back step\n 4.slash it with your sword \n5.side step ↵
      to the left\n";
cin >> k;
if (k == 1)
{
    cout << "why would you..?,you died...";
}

else if (k == 2)
{
    cout << "why would you..?,you died...";
}

else if (k == 4)
{
    d = d + 2;
    cout << "you slash the head of the bear while its down.\nyou dealt ↵
          monsterous amount of damage.\nyou 1hit killed the bear.[decision ↵
          making " << d << "]\\n\\n\\nCongratulations, \nyou Won The Game!";
}

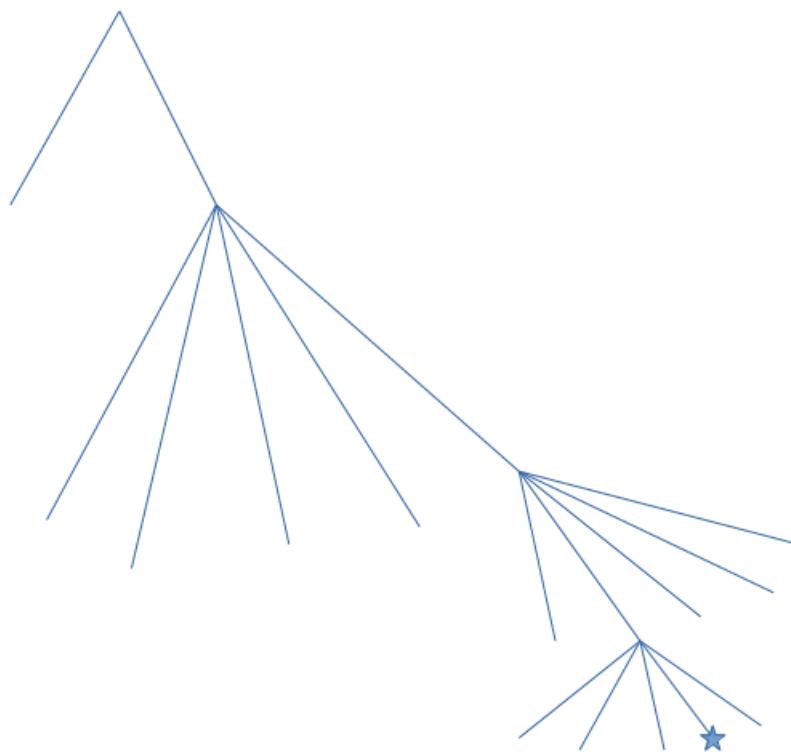
else if (k == 3)
{
    cout << "why would you..?,you died...";
}

else if (k == 5)
{
    cout << "why would you..?,you died...";
}
```

```
if (j == 3)
{
    cout << "you try to back step, and you succeed, but the bear somehow ←
              kicks you in the head and you die.";
}

if (j == 4)
{
    cout << "you try to slash it with your sword, but do you think you ←
              hit harder than a bear?\nyou died from a pawn hit in the head.";
}

if (j == 5)
{
    cout << "you try to sidestep to the left, but the bear remembers you ←
              doing that before, it turns toward you and slashes...\nyou died.";
}
}
```



13.4. ábra. A ciklomatikus ábra

A komplexitás $17-2+3=18$

a komplexitási érték 4 típusra bontja fel a forráskódokat. Ha a komplexitás végeredményeként **1-10 közötti**

számot kapunk, akkor a forráskód egyszerűnek mondható.

11 és 20 között mérsékelten nehézkes a kód megértése.

21 és 50 között a programkód bonyolult

az utolsó osztály pedig a **50 feletti** számok, mely a nagyon bonyolult forráskódoknál fordul elő, és magas a kockázat a hibák ejtésénél.

Számolásunkat a [lizard.ws](#) oldal is approválja.

The screenshot shows the lizard.ws web application interface. On the left, there is a code editor window with a .cpp file containing C++ code. The code includes an if-statement and a cout statement with a string. Some parts of the string are underlined with red wavy lines, indicating potential errors or warnings. On the right, there is a summary panel with three tabs: 'File Type .cpp', 'Token Count 311', and 'NLOC 82'. Below this, there is a table with columns 'Function Name', 'NLOC', and 'Complexity'. A single row is shown for the function 'fight' with NLOC 82 and Complexity 18.

Function Name	NLOC	Complexity
fight	82	18

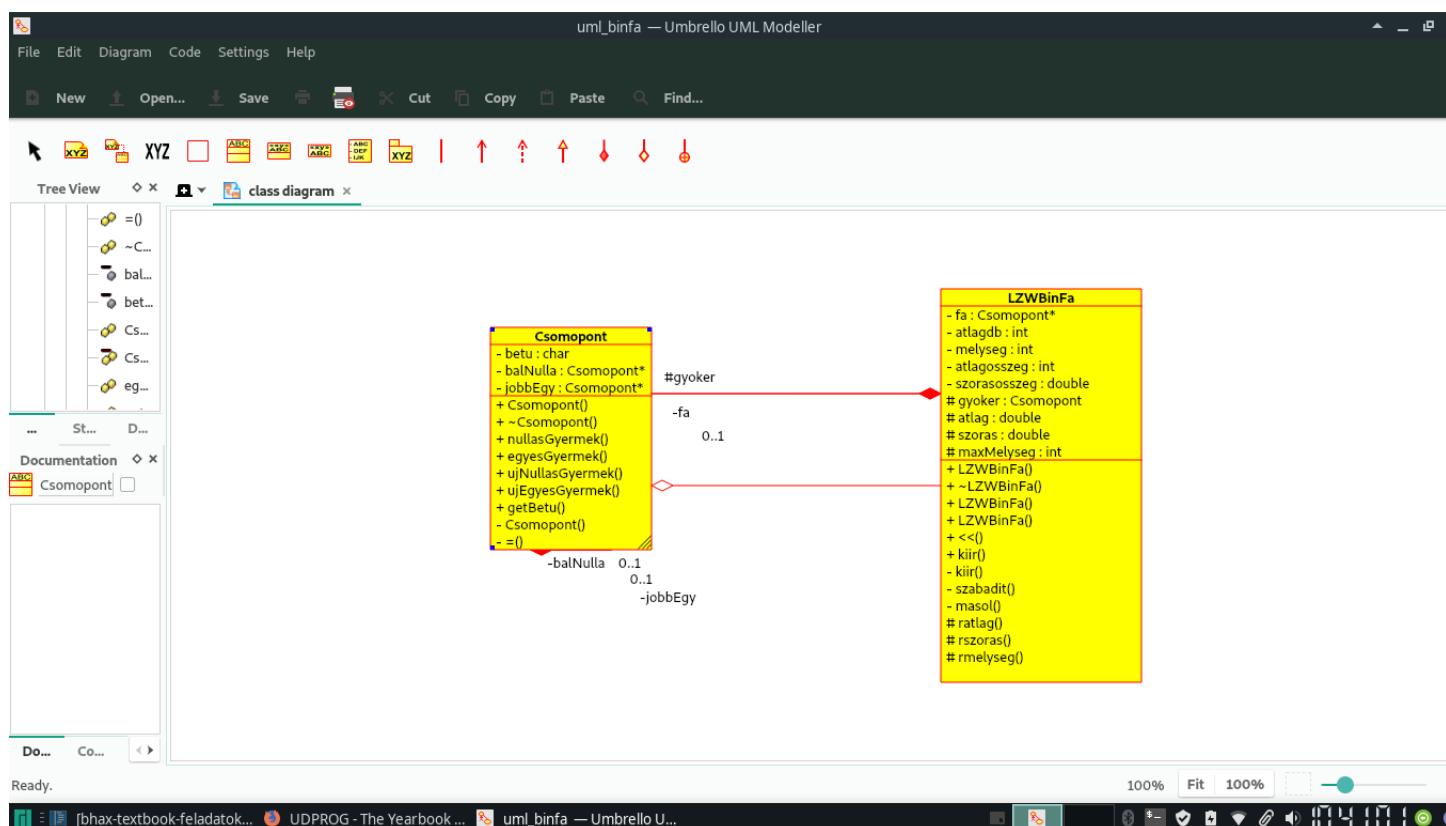
13.5. ábra. A lizard ciklomatikus számolás

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/UD> (28-32 fólia)



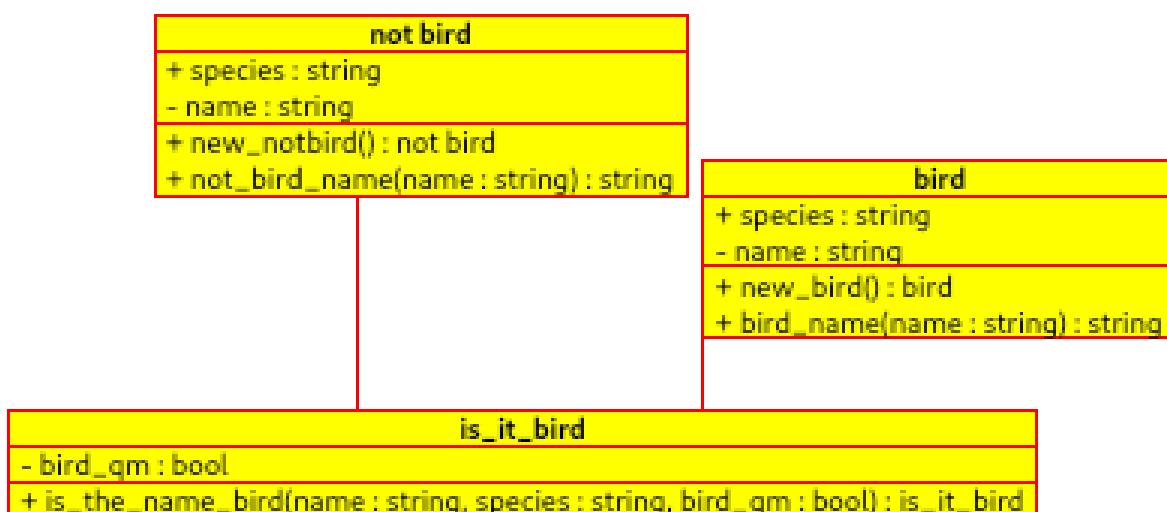
14.1. ábra. Az LZW UML

Az aggregáció egy rész-egész kapcsolatot jelent. A fő-objektum fizikailag is tartalmazza, vagy birtokolja a

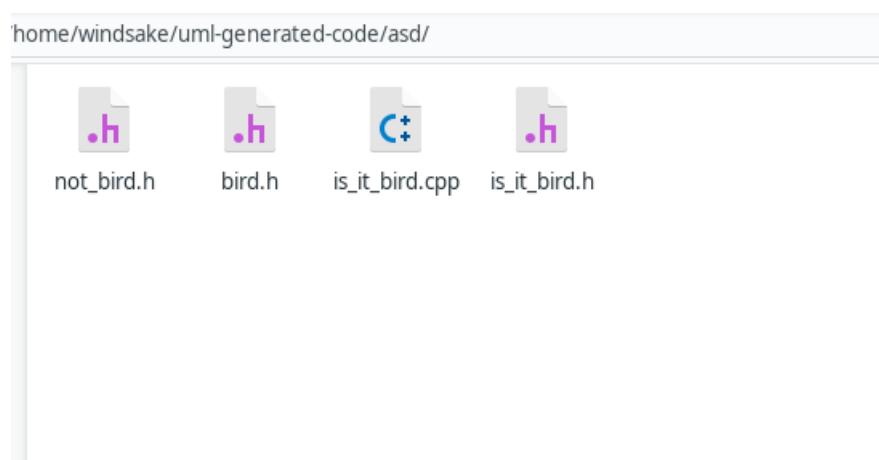
rész-objektumot, általában ezen objektum létezése függésben van a fő-objektumtól. Ennek két fajtája lehet, gyenge és erős aggregáció. A **gyenge aggregáció**-ban a rész-objektum létezhet a fő-obj. nélkül is, míg az **erős aggregáció**-nál ez nem lehetséges. Általában az erős aggregáció-t szokták **kompozíció**-nak nevezni. A **gyenge aggr.** jele a két osztály között húzott telített egyenes, melynek végén egy üres rombusz található. A rombusz azon az oldalon van, amely osztály a fő-objektum. Az **erős aggr./kompozíció** jele is egy vonal, viszont a végén egy telített rombusz található. Az UML ábrán megtalálható minden két típusú asszociáció.

14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!



14.2. ábra. Az új UML



14.3. ábra. Az új UML

Az UML segítségével létrehozott diagramon az Osztályok között kétirányú összeköttetés, ún. asszociáció áll fenn. Ezen kapcsolat iránya megadható, de nem szükséges. A kapcsolatban résztvevő osztályok léte

egymástól általában független, de minimum az egyikük ismeri/használja a másikat. Gyakorlatilag az osztályokból létrejövő objektumok között van összefüggés. Jele: a két osztály között telített vonal, a végén nyíllal, ha írányt is jelölünk.

Ebben az UML-ben 3 osztályt ábrázoltam, amelyek segítségével eldönti a program, hogy a bekért állat neve és faja alapján madár-e, avagy sem. Az első két osztály a **bird/madár** és a **not_bird/nem_madár** osztályok. Ezekben az osztályokban tároljuk majd a különböző állatokat aszerint, hogy madár vagy sem a megadott adat. A harmadik osztály az **is_it_bird/madár?**, melynek feladata e fontos kérdés eldöntése. Miután megkapja a **species** és **name** adatokat, egy if-statement sorozat segítségével eldönti, hogy melyik osztályhoz küldi tovább az adatokat.

14.3. BPMN

Rajzolunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog> (34-47 fólia)



14.4. ábra. A BPMN ábra

A képen egy regisztrációs folyamat BPMN ábrája látható.(Business Process Model and Notation/ üzleti folyatmodellezés és jelölés) Indulunk a **regisztráció** gomb megnyomásától. Ekkor az oldal egy kérést küld a szervernek, amely után egy új lap jelenik meg, ezen a lapon helyezkedik majd el a regisztrációhoz szükséges szövegdobozok és utasítások sorozata, na meg persze a **regisztráció** gomb. A szerver válaszában már az egész oldalt küldi el, amelyben az oldal tartalma egészében megtalálható, a felhasználónak csak annyi a dolga, hogy megadja a szükséges információkat. Ezután két dolog történhet, ha az oldal képes az on-the-spot információ-ellenőrzésre, akkor már a **regisztráció** gomb megnyomása előtt visszajelzést ad, hogy a megadott adatok helyesek-e, esetleg már valaki által foglaltak-e vagy sem. A másik lehetőség ugyebár,hogy csak a gomb megnyomása után tudjuk meg, hogy valid adatokat adtunk-e meg. A model az első opción szerint lett elkészítve.

Ha minden validálva lett, akkor jöhét a regisztráció. A felhasználó újra megnyomja a gombot. Ezután a kliens kérést küld a szervernek, amelyben a titkosított adatok is jelen vannak. Ez egy keretbe van foglalva, és egy ún. SET/POST kérés-típus kel útra a szerver felé, amely arra kéri a szervert, hogy az adatokat vigye be az adatbázisba, ezzel végleglegítve a regisztrációt. Végezetül a szerver visszaigazoló üzenetet küld, amelyben megtaláljuk, hogy a regisztráció sikeres volt-e, vagy egy esetleges szerver-oldali hiba miatt sikertelen. Az utóbbi esetben később újrapróbálkozhat a felhasználó, esetleg ha van elérhetőség kitéve az oldalra, ott tájékoztathatja az oldal üzemeltetőjét.

14.4. Egy Esettan

A BME-s C++ tankönyv 14. fejezetét (427-444 elmélet, 445-469 az esettan) dolgozzuk fel!

forrás:[github link](#)

A product.h headerfájl tartalmazza a termékkel kapcsolatos adatokat, mint például a termék kora, beszerzési dátum és ár, a termék neve és a jelenlegi ára. Mivel viszont az **initialPrice**, **dateOfAcquisition** és **a name** tagváltozók protected/véddettek, ezért azok eléréséhez szükségünk van a publikus:

```
time_t Product::GetDateOfAcquisition() const
{
    return dateOfAcquisition;
}

int Product::GetInitialPrice() const {return initialPrice;}

string Product::GetName() const {return name;}
```

függvényekre.

A termék korának kiszámítását a **GetAge** függvény végzi, amely az aktuális dátumból és a beszerzési dátumból számolja ki azt. A függvény forráskódja a következő:

```
int Product::GetAge() const
{
    time_t currentTime;
    time(&currentTime);
    double timeDiffInSec = difftime(currentTime, dateOfAcquisition) ←
    ;
    return (int)(timeDiffInSec / (3600 * 24));
}
```

A függvény return sorában láthatjuk az (int) szócskát a timediff előtt, ez egy ún. típus kényszerítés, jelen esetben a visszatérési érték típusát integerré konvertálja.

Az osztályban megtalálható a GetCurrentPrice függvény is, amely segítségével a termék jelenlegi ára számítható ki, viszont mivel az ár termékfüggő, ezért azt virtuálisan definiáljuk, hogy később a leszármazott osztályok felüldefiniálhassák azt. Ez a függvény a beszerzési árat adja vissza a product.h fájlból.

A hardDisk osztályban megtalálható ezen függvény felüldefiniálása:

```
class HardDisk : public Product
{
    int speedRPM;
public:
    int GetCurrentPrice() const;
    ...
};

int HardDisk::GetCurrentPrice() const
{
    int ageInDays = GetAge();
    if(ageInDays < 30)
    {
```

```
        return initialPrice;
    }
    else if (ageInDays >= 30 && ageInDays <90)
    {return (int)(initialPrice * 0.9);}
    else
    {return (int)(initialPrice * 0.8);}
}
```

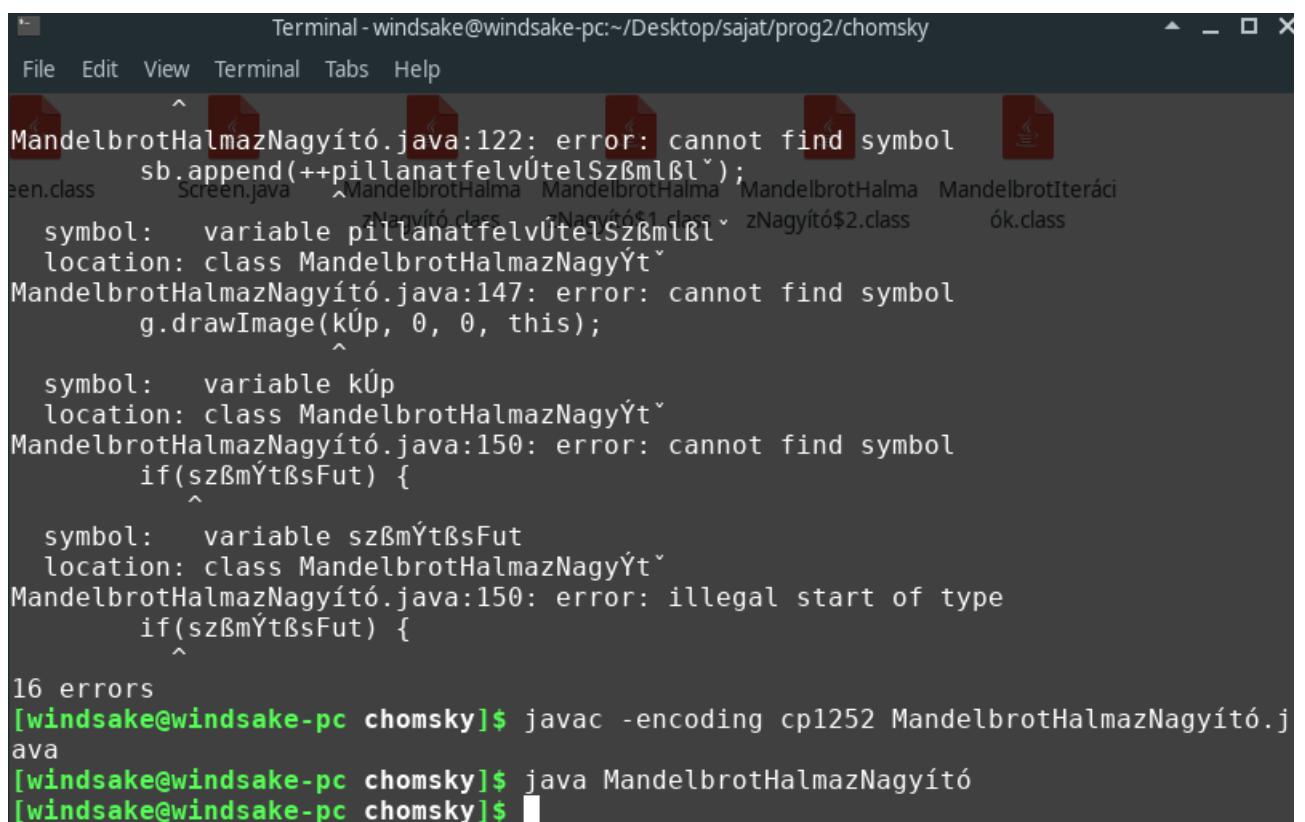
Itt is megtalálható a típuskényszerítés, amivel elérjük, hogy ne tört számot kapunk vissza, hanem egész számot.

15. fejezet

Helló, Chomsky!

15.1. Encoding

Fordítsuk le és futassuk 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>



The screenshot shows a terminal window titled "Terminal - windsake@windsake-pc:~/Desktop/sajat/prog2/chomsky". The window contains the following text:

```
Terminal - windsake@windsake-pc:~/Desktop/sajat/prog2/chomsky
File Edit View Terminal Tabs Help
MandelbrotHalmazNagyító.java:122: error: cannot find symbol
    sb.append(++pillanatfelvÜtelSzßmlßl");
               ^
symbol:   variable pillanatfelvÜtelSzßmlßl
location: class MandelbrotHalmazNagyító
MandelbrotHalmazNagyító.java:147: error: cannot find symbol
    g.drawImage(kÚp, 0, 0, this);
               ^
symbol:   variable kÚp
location: class MandelbrotHalmazNagyító
MandelbrotHalmazNagyító.java:150: error: cannot find symbol
    if(szßmÝtßsFut) {
               ^
symbol:   variable szßmÝtßsFut
location: class MandelbrotHalmazNagyító
MandelbrotHalmazNagyító.java:150: error: illegal start of type
    if(szßmÝtßsFut) {
               ^
16 errors
[windsake@windsake-pc chomsky]$ javac -encoding cp1252 MandelbrotHalmazNagyító.java
[windsake@windsake-pc chomsky]$ java MandelbrotHalmazNagyító
[windsake@windsake-pc chomsky]$
```

15.1. ábra. A mandel futás

A képen látható, amint megpróbáljuk az encoding nélkül lefordítani a programunkat, kevés sikkerrel. A

-encoding kapcsoló használatával és a megfelelő karakterkészlettel viszont tökéletesen/hiba nélkül lefordul és lefut a program. A lehetséges karakterkészletek megtalálhatóak az oracle oldalán:

<https://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html>

Az általunk használt készlet a 1252-es, amit Latin-1-nek is hívnak. A javac -encoding nélkül általában a rendszer által használt codeset-et használja alapértelmezetten, de mivel a JVM nem igazán veszi figyelembe a helyi rendszert(hiszen virtual machine), ezért ez nem univerzális "igazság". Viszont az igaz, hogy a javac-ban konfigurálható az alapértelemezett enkódolási charset, és ez általában megyegyezik a helyi gépen használtat. A program futása folyamán ez az alapértelemezett karakterkészlet megállapítható a **Charset.defaultCharset()** parancs használatával.

15.2. Full screen(ződ)

Készítünk egy teljes képernyős Java programot! Tipp: https://www.tankonyvtar.hu/en/tartalom/tkt/javatitok-javat/ch03.html#labirintus_jatek

```
import java.awt.*;
import javax.swing.JFrame;

public class bucky extends JFrame
{
    public static void main (String[] args)
    {
        DisplayMode dm = new DisplayMode(800, 600, 16, DisplayMode.REFRESH_RATE_UNKNOWN);
        bucky b = new bucky();
        b.run(dm);
    }

    public void run(DisplayMode dm)

    {
        setBackground(Color.BLACK);
        setForeground(Color.WHITE);
       setFont(new Font("Arial",Font.PLAIN, 24));

        Screen s = new Screen();
        try{
            s.SetFullScreen(dm, this);
            try{
                Thread.sleep(2700);

            }catch(Exception ex) {}
            finally{
                s.RestoreScreen();
            }
        }
    }
}
```

```
public void paint(Graphics g)
{
    g.drawString("This...A programozásra gondolva olykor valami misztikus ←
    érzés járja ",200,200);
    g.drawString("át az embert, ami további hajtóerőt és lelkesedést kölcsönöz. ←
    A sikeres oktatás ",200,300);
    g.drawString("elsőleg es célja ennek átadása, ebből fejlődhet majd ki ←
    minden más, ami csak kell:",200,400);
    g.drawString(" A könyvet elsősorban\n tanári kézikönyvként ajánljuk ←
    középiskolai informatikatanároknak, de kiegészítő\n irodalomként hasznos ←
    olvasmánynak tartjuk informatikus tanárjelöltek, diákok és általában a ←
    programozni tanulni vágyók számára \nagyaránt. ",200,450);
    g.drawString("az ipar számára a professzionális programozó, a tudománynak ←
    és az emberiségnek a komplex problémákkal megküzdeni tudó ←
    algoritmuselmélész, az\n egyénnek és a társadalomnak a boldog és tetter ←
    ős polgár. Reményeink szerint ennek az érzésnek az átadásához ezzel a ←
    digitális szakkönyvvel \nmi is hozzá tudunk járulni, miközben általában ←
    foglalkozunk \na programozással, annak elméletével és filozófiájával, ←
    majd részletesen és gyakorlatiasan az objektumorientált paradigmával, és ←
    ezen belül\n a Java programozással. Ebben a kézikönyvben\n a Java ←
    nyelvvel egy labirintus játék világának felépítése során ismerkedünk meg ←
    , a\nmi során ezt az egyszerű, példa labirintus játékot elkészítjük a ←
    weboldalon elhelyezhető appletként, mobiltelefonos és teljes képernyős\n ←
    PC-s, illetve hálózati: TCP/IP, Java Servlet\n, Java RMI és CORBA ←
    változatban is. De ezeken a labirintus témájú esettanulmányokon túl ←
    biológiai, matematikai és informatikai programozási példákkal is ←
    megismерkedhet majd \na kedves Olvasó.",200, 500);
}
}
```

Itt látható a bucky.java fájl tartalma. Mint láthatjuk, a program a dm objektumban inicializálja a megjelenítései beállításokat,

```
DisplayMode dm = new DisplayMode(800,600,16,DisplayMode. ←
    REFRESH_RATE_UNKNOWN);
```

Az első két integer típus a felbontást, a harmadik a színkészletet, az utolsó paraméter pedig a frissítési rátát állítják be, viszont az utóbbit nem tudjuk, ezért az REFRESH_RATE_UNKNOWN parancsot használjuk.

```
public void run(DisplayMode dm)
{
    .
    .
    .
    Screen s = new Screen();
    try{
        s.SetFullScreen(dm, this);
```

A run metódus a program megjelenítési képét inicializálja, itt található a SetFullScreen parancs is, amely a teljes-képernyős mód beállításáért felel.

```
setBackground(Color.BLACK);
    setForeground(Color.WHITE);
   setFont(new Font("Arial",Font.PLAIN, 24));
```

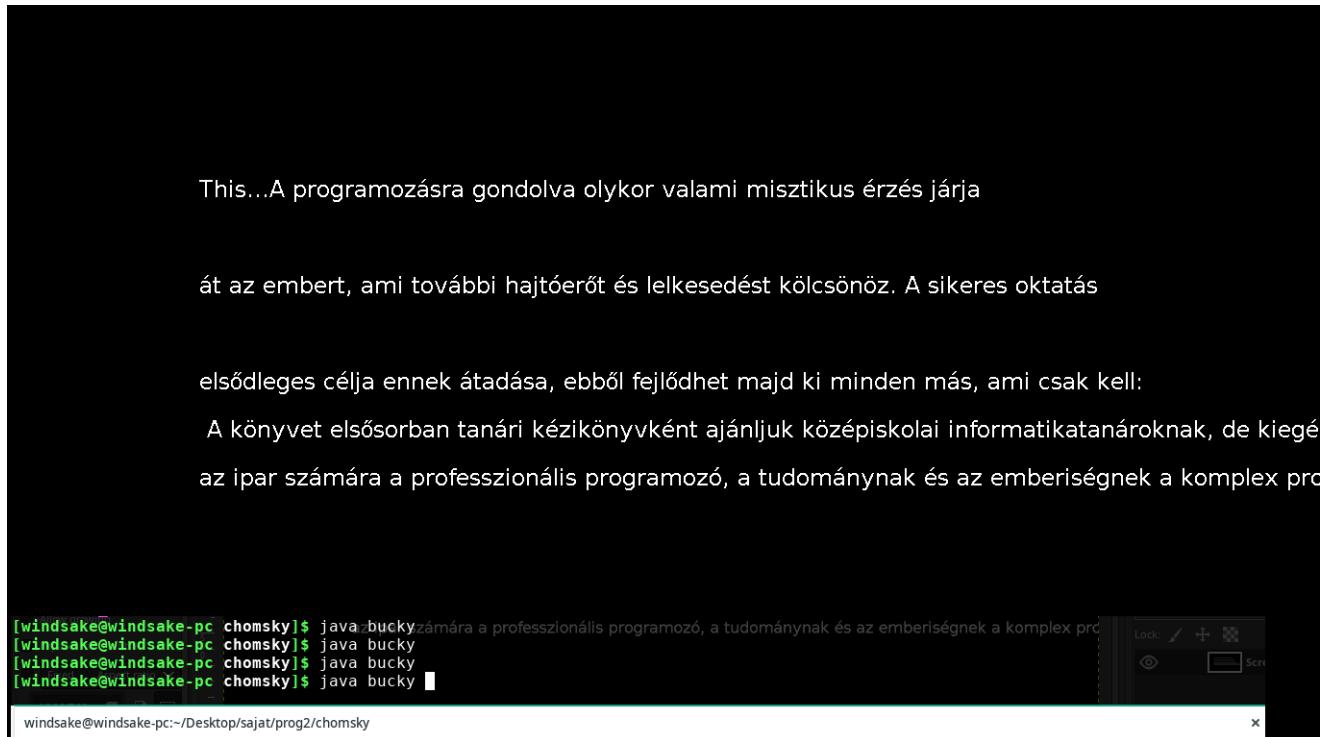
Itt állítjuk be a betűk típusát, méretét, illetve a betűk és a háttér színét is, melyek ebben a sorrendben fehér ill. fekete szinűek.

A **Screen s = new Screen();** parancs példányosítja a megjeleníteni kívánt ablakunkat.

```
public void paint(Graphics g)
{
    g.drawString("This...A programozásra gondolva olykor valami misztikus ←
        érzés járja ",200,200);
    g.drawString("át az embert, ami további hajtóerőt és lelkesedést kölcsönöz. ←
        A sikeres oktatás ",200,300);
    g.drawString("elsődleg es célja ennek átadása, ebből fejlődhet majd ki ←
        minden más, ami csak kell:",200,400);
    g.drawString(" A könyvet elsősorban\n tanári kézikönyvként ajánljuk ←
        középiskolai... ",200,450);
    g.drawString("az ipar számára a professzionális programozó, a tudománynak ←
        ...",200, 500);
}
```

Végül pedig látható a paint metódus. Ez felel a képernyőre "festésért", ami esetben csak pár sor szöveg, de lehet ez akár ábrák, képek sokasága is, a megjeleníteni kívánt objektum szerint viszont a kód is változik, és nehézségi foka is egyre nől.

Megoldás forrásai:[github link](#)



15.2. ábra. A fullscreen futás

15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

A program módosítás szempontjából az alábbiakon esett át: színvilág változtatás, mostantól a kockák teste kék színű. Az irányítás kézreállóságának biztosítása: wasd billentyűk használhatóak a nyilak funkciójának ellátásához, továbbá a Z irányú mozgásra (ami eddig a page up és page down bill.-el történt), most a q és e billentyűk szolgálhatnak. A Forgatási sebesség pozitív irányban meg lett növelve, korrigálhatóság miatt a negatív irány azonos értéken maradt. A régi irányítás is működik, esetleges használati kényelmetlenségek elkerülése végett.

```
void keyboard ( unsigned char key, int x, int y )
{
    .
    .
    .
    else if( key == 'w' ) {
        cubeLetters[index].rotx += 25.0;
    } else if ( key == 's' ) {
        cubeLetters[index].rotx -= 5.0;
    } else if ( key == 'd' ) {
        cubeLetters[index].roty -= 5.0;
    } else if ( key == 'a' ) {
```

```
        cubeLetters[index].rotY += 25.0;
    } else if ( key == 'q' ) {
        cubeLetters[index].rotZ += 25.0;
    } else if ( key == 'e' ) {
        cubeLetters[index].rotZ -= 5.0;
    }

    glutPostRedisplay();
}
```

itt látható az irányításra szolgáló billentyűk kiosztásának a kódrészlete, emellett pedig a forgatások sebességét is láthatjuk a kódrészleten.

a program lefordításához a következő kapcsolók kellenek: g++ para6.cpp -o para **-lboost_system -lGL -IGLU -lglut**



15.3. ábra. A fullscreen futás

Az lboost_system library a boost-nak a használata miatt szükséges, az lGL, az lGLU ls az lglut kapcso-

lók pedig az OpenGL könyvtárai, hiszen az OpenGL-t használva jelenítettük meg a kockákat, ezért ezekre is szükségünk volt. Érdekes viszont megfigyelnünk, hogy az glut(openGL utility toolkit) támogatása már több, mint 15 éve lejárt, ehelyett a freeglut-ot használják manapság az OpenGL programok írásánál, viszont ez a jelenlegi programunkat nem befolyásolja, a megfelelő library-k segítségével tökéletesen működőképes. A freeglut megtalálható itt: <http://freeglut.sourceforge.net/>

15.4. I334d1c4⁵ (zód)

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet> (Ha ez első részben nem tettek meg, akkor írasd ki és magyarázd meg a használt struktúratömb memória foglalását!)

A cipher java-ban.

```
import java.util.Scanner;
import java.util.Vector;
import java.util.HashMap;
import java.util.Random;
import java.util.Map.Entry;
```

A program "backbone"-ja a leet hashmap, amely a "betűket társítja a számokkal és egyéb karakterekkel. A fontossági sorrendben a következő maga a translate funkció, amely a szó összes karakterén végighalad a for-ciklussal, míg a szó végéhez nem ér, és kicseréli a karaktert egy random választott számmal a map-ból.

```
private HashMap<String, String[]> leet = new HashMap<String, String[]>() {{
    put("a", new String[] { "4", "4", "@", "-" });
    put("b", new String[] { "b", "8", "3", "}" });
    put("c", new String[] { "c", "(", "", "{" });
    put("d", new String[] { "d", ")", "]", "}" });
    put("e", new String[] { "3", "3", "3", "3" });
    put("f", new String[] { "f", "=", "ph", "#" });
    put("g", new String[] { "g", "6", "[", "+"]);
    put("h", new String[] { "h", "4", "-", "[-" });
    put("i", new String[] { "1", "1", "", "!" });
    put("j", new String[] { "j", "7", "_", "__" });
    put("k", new String[] { "k", "", "1", "{" });
    put("l", new String[] { "l", "1", "", "_" });
    put("m", new String[] { "m", "44", "(V)", "" });
    put("n", new String[] { "n", "", "", "V" });
    put("o", new String[] { "0", "0", "()", "[" });
    put("p", new String[] { "p", "o", "D", "o" });
    put("q", new String[] { "q", "9", "O_", "(,)" });
    put("r", new String[] { "r", "12", "12", "2" });
    put("s", new String[] { "s", "5", "$", "$" });
    put("t", new String[] { "t", "7", "7", "/" });
    put("u", new String[] { "u", "_", "(_)", "[_" });
    put("v", new String[] { "v", "", "", "" });
    put("w", new String[] { "w", "VV", "", "()" });
    put("x", new String[] { "x", "%", ")" ("") ("") });
}}
```

```

        put("y", new String[] { "y", "", "", "" });
        put("z", new String[] { "z", "2", "7_", "_" });

        put("0", new String[] { "D", "0", "D", "0" });
        put("1", new String[] { "I", "I", "L", "L" });
        put("2", new String[] { "Z", "Z", "Z", "e" });
        put("3", new String[] { "E", "E", "E", "E" });
        put("4", new String[] { "h", "h", "A", "A" });
        put("5", new String[] { "S", "S", "S", "S" });
        put("6", new String[] { "b", "b", "G", "G" });
        put("7", new String[] { "T", "T", "j", "j" });
        put("8", new String[] { "X", "X", "X", "X" });
        put("9", new String[] { "g", "g", "j", "j" });

    }};

    public Leet translate() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < word.length(); ++i) {
            char c = word.charAt(i);
            String loc = Character.toString(Character.toLowerCase(c));
            if (leet.containsKey(loc))
                sb.append(leet.get(loc)[rand.nextInt(leet.get(loc).length)]);
            else
                sb.append(c);
        }
        this.sb = sb.toString();

        return this;
    }
}

```

```

public static void main(String[] args) {
    String wordvect;
    Scanner scan = new Scanner(System.in);
    String word;
    String question;
    boolean questionbool = true;
}

```

A main függvényben példányosítjuk a wordvect objektumot, majd egy szöveg-scannert inicializálunk, melyel majd a szavakat kérjük be a felhasználótól.

```

while(questionbool=true)
{
    questionbool=false;
    System.out.println("irj be egy szót!");
    word=scan.nextLine();
    System.out.println("szó: "+word);

    wordvect=(new Leet(word).translate().getLeet());
    System.out.println(wordvect);
}

```

a `word=scan.nextLine();` sor kéri be a word változóba a szót,a `translate();` függvény pedig magáért a lefordításért felelős, majd a getleet függvénnnyel visszaadjuk a lefordított szót.

15.4. ábra. A fullscreen futás

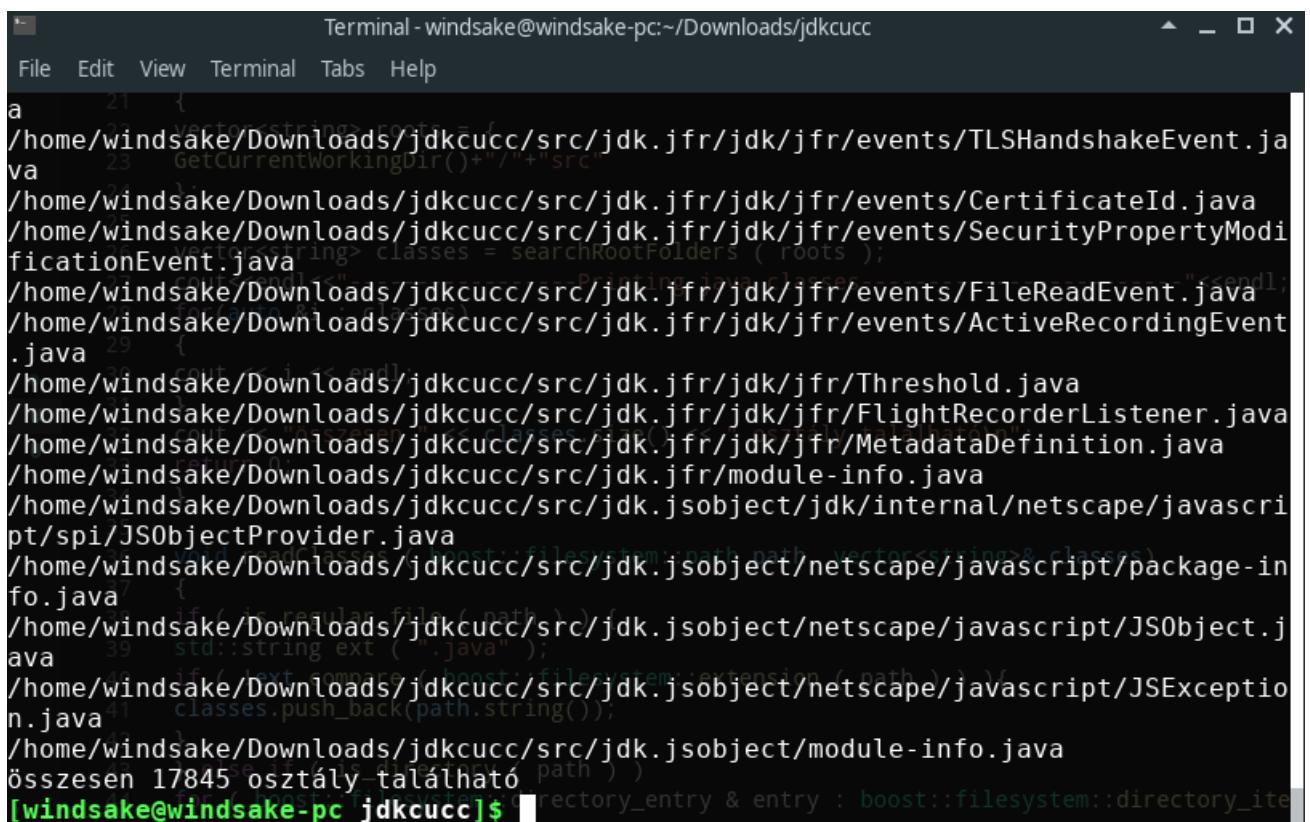
16. fejezet

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)!

A programkód forrása (**Nem saját source**): [github link](#)



```
Terminal - windsake@windsake-pc:~/Downloads/jdkcucc
File Edit View Terminal Tabs Help
a 21  {
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/events/TLSHandshakeEvent.java
va 23 GetCurrentWorkingDir() + "src"
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/events/CertificateId.java
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/events/SecurityPropertyModificationEvent.java
classes = searchRootFolders(roots);
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/events/FileReadEvent.java
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/events/ActiveRecordingEvent.java
29 cout << i << endl;
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/Threshold.java
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/FlightRecorderListener.java
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/jdk/jfr/MetadataDefinition.java
/home/windsake/Downloads/jdkcucc/src/jdk.jfr/module-info.java
/home/windsake/Downloads/jdkcucc/src/jdk.jsobject/jdk/internal/netscape/javascript/spi/JSObjectProvider.java
/home/windsake/Downloads/jdkcucc/src/jdk.jsobject/netscape/javascript/package-info.java
if( isRegularFile( path ) )
/home/windsake/Downloads/jdkcucc/src/jdk.jsobject/netscape/javascript/JSObject.java
39 std::string ext( ".java" );
if( ext == boost::filesystem::extension( path ) )
/home/windsake/Downloads/jdkcucc/src/jdk.jsobject/netscape/javascript/JSException.java
41 classes.push_back( path.string() );
/home/windsake/Downloads/jdkcucc/src/jdk.jsobject/module-info.java
összesen 17845 osztály található path )
[windsake@windsake-pc jdkcucc]$
```

16.1. ábra. A jdk futás

A program három függvényből áll, a GetCurrentWorkingDir,readClasses és searchRootFolders függvé-

nyekből. Nézzük először a **GetCurrentWorkingDir** funkciót:

```
string GetCurrentWorkingDir( void )
{
char buff[FILENAME_MAX];
GetCurrentDir( buff, FILENAME_MAX );
string current_working_dir(buff);
return current_working_dir;
}
```

A függvényünk nem kér be paraméternek semmit, a GetCurrentDir beépített függvényt használja(getcwd), hogy az aktuális könyvtár nevét bekérje, majd létrehoz egy string típusú **current_working_dir** változót, melyben eltároljuk a buffer-ben található fájlnevet, melyet végül visszaad a függvény.

Következő funkció egy visszatérési érték nélküli funkció/metódus lesz, a **readClasses**:

```
void readClasses ( boost::filesystem::path path, vector<string>& classes)
{
if ( is_regular_file ( path ) ) {
std::string ext ( ".java" );
if ( !ext.compare ( boost::filesystem::extension ( path ) ) ) {
classes.push_back(path.string());
}
} else if ( is_directory ( path ) )
for ( boost::filesystem::directory_entry & entry : boost::filesystem::directory_iterator ( path ) )

readClasses ( entry.path(), classes );
}
```

A függvény paraméterként egy path típusú path változót és egy string-vector objektumot vár. Ha az **is_regular_file** függvény igaz értéket ad vissza, akkor megadjuk a keresni kívánt fájl-kiterjesztést az ext objektumban, ami most a ".java"

A Következő if elágazásban megnézzük, hogy az ext objektumunk értéke megegyezik-e a path-ban található kiterjesztéssel. Ha nem,akkor push_back, tehát a classes vektor végehez adja a path-et.

Else if ágba lépünk, ha az **is_regular_file** nem volt igaz, ekkor az **is_directory** függvényt hívjuk meg ellenőrzésre. Ha a path egy könyvtár, akkor meghívunk egy for ciklust, melyben az entry.path() elérési utakat behelyezzük a classes vektorunkba. A ciklus addig megy, amíg van entry.

Utolsó függvény pedig a **searchRootFolders**:

```
vector<string> searchRootFolders (vector<string> folders)
{
vector <string> classes;
for ( const auto & path : folders)
{
boost::filesystem::path root ( path );
readClasses ( root, classes );
}
return classes;
}
```

Láthatjuk, hogy ez a függvény már használja a readClasses függvényt, amit az előbb néztünk át, itt a path a root objektum lesz, a class pedig marad a régi visszatérési érték a classes objektum lesz.

Végül jöhet a main függvény:

```
int main(int argc, char const *argv[])
{
vector<string> roots = {
GetCurrentWorkingDir() + "/" + "src"
};

vector<string> classes = searchRootFolders(roots);
cout << endl << "-----Printing java classes-----" <
" << endl;
for(auto &i : classes)
{
cout << i << endl;
}
cout << "Összesen " << classes.size() << " osztály található\n";
return 0;
}
```

A classes vektorunk értékéül a searchRootFolders függvény visszatérési értékét adjuk, ami ugyancsak a classes objektum volt. Ezután egy for ciklusban kiíratjuk a classes-ben található elérési utakat, végül a classes.size()-al megnézzük, hogy mennyi osztályt talált a program.

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!

```
LZWBInFa (LZWBInFa&& regi){
    std::cout << "LZWBInFa movetor" << std::endl;

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

Itt látható az std::move funkció, amely egy jobbértek referenciát ad vissza, ezzel "mozgatva" az objektumot.

```
LZWBInFa (const LZWBInFa& regi)
{
std::cout << "LZWBInFa copytor" << std::endl;

gyoker.ujEgyesGyerek (masol (regi.gyoker.egyesGyerek (), regi.fa));
gyoker.ujNullasGyerek (masol (regi.gyoker.nullasGyerek (), regi.fa));

if (regi.fa == & (regi.gyoker)) {fa = &gyoker;}
}
```

Ez a másoló konstruktor, mely lemásolja a regi.fa-ban található nullás és egyes gyermeket-ét, és az új fa-ba másolja azokat.

```
LZWBinFa (const LZWBinFa& regi)
{
    std::cout << "LZWBinFa masolo ertekekadas" << std::endl;

    szabadit (gyoker.egyesGyerek());
    szabadit (gyoker.nullasGyerek());

    gyoker.ujEgyesGyerek (masol (regi.gyoker.egyesGyerek (), regi.fa));
    gyoker.ujNullasGyerek (masol (regi.gyoker.nullasGyerek (), regi.fa));

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

}
```

EZ pedig a másoló értékkopírozás, amely nagyban hasonlít a konstruktőrök megoldásra, csak annyi a különbség, hogy a másolás előtt felszabadítja az egyes- és nullásgyermekeket, Null-ra állítja értéküket.

```
LZWBinFa (LZWBinFa&& regi)
{
    std::cout << "LZWBinFa movetor" << std::endl;

    gyoker.ujEgyesGyerek (regi.gyoker.egyesGyerek ());
    gyoker.ujNullasGyerek (regi.gyoker.nullasGyerek ());

    regi.gyoker.ujEgyesGyerek(nullptr);
    regi.gyoker.ujNullasGyerek(nullptr);
}
```

Különbség az eredeti "movetor"-hoz képest, hogy itt nem egy jobbértek referenciát ad vissza egy move függvény, helyette az új fa gyokerének az új Egyes és Nullás gyermeket-ét a régi gyökér nullás és egyes gyermekére állítjuk be, majd a regi fa gyökerének gyermeket beállítjuk, hogy nullpointerre mutassanak.

16.3. Hibásan implementált RSA törése

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: <https://arato.inf.unideb.hu/batfai.rsa> (71-73 fólia) által készített titkos szövegen.

A forráskód itt található: [github link](#)

Az encrypt függvény:

```
public BigInteger[] encrypt(String msg) {
    byte[] buffer = msg.getBytes();
    java.math.BigInteger[] priv = new java.math.BigInteger[buffer.length];
    for (int i = 0 ; i < priv.length ; i++) {
        priv[i] = new java.math.BigInteger(new byte[] {buffer[i]});
```

```
priv[i] = priv[i].modPow(this.e, this.n);
}
return priv;
```

A BigInteger[] egy BigInteger tömb, ez a típus a java.math.BigInteger importálásával érhető el.

```
byte[] buffer = msg.getBytes();
```

Ez a sor létrehoz egy byte tömböt, amely a buffer névre hallgat. Ez a buffer értékül kapja a **msg** string byte-sorozattá alakított változatát. A sorozattá alakítás a platform alapértelmezett charset-ével történik.

A következő sorban egy priv változót hozunk létre, amely szintén BigInteger tömb típusú lesz. Ezután egy for ciklust indítunk, ami a priv változó összes elemén végigmegy, először az i-edik elemét egyenlővé teszi egy új BigInteger-el, majd egy modulus exponenciális függvényt végez el azokon. A **this.e** jelenti az exponenciális számot, míg a **this.n** a modulus-t. A pow-al szemben a modpow képes elvégezni a számítást akkor is, ha az exp negatív szám.

```
public String atlagDecrypt(String clean, BigInteger[] msg, Map<Character, Integer> m) {
    byte[] buffer = clean.getBytes();
    for (int i = 0; i < msg.length; i++) {
        msg[i] = msg[i].modPow(this.d, this.n);
        buffer[i] = msg[i].byteValue();
    }
    .
    .
    .
    .
    .
}
```

Ez pedig az átlag szerinti szöveg-törés függvénye. Mint az encrypt-ben, itt is lesz egy buffer változónk, amely a clean változó byte-sorozattá alakított változatát tárolja. A for ciklusban a msg tömb elemein megyünk végig, először megint a modpow függvény segítségét hívjuk, majd a buffer-en is végigmegyünk, és a buffer i-edik elemének értékéül a msg i-edik elemének a byte-szerinti értékét adjuk.(mivel ugye a msg egy szám típusú tömb, annak vesszük a modulusát, hatványozását, attól még szám lesz, és ezt a számot konvertáljuk át byte formára.)

```
for (int i = 0; i < wrong_dm.length(); i++) {
    Character c;
    if ((c = contains(currentMap.get(wrong_dm.charAt(i)), m)) != null) {
        wrong_dm = wrong_dm.replace(wrong_dm.charAt(i), c);
        currentMap.remove(wrong_dm.charAt(i));
    }
}
```

Itt látható a karakterek kicserélése a map-ban látott gyakoriságot követve, és ezzel létrejön a "tiszta, lefordított szöveg".

16.2. ábra. Az RSA futás

Látható, hogy a program egyfajta bruteforce metódusként üzemel azzal, hogy az RSA kódolás betűk helyett szavanként történt meg, ezért a betű előfordulási arány segítségével készíthető olyan program, amely az eredeti szöveg egy részét képes "lefordítani". Sajnos ez nem teljesen lehetséges, hiszen ha a betűk aránya a szövegben nem egyezik meg az online található statisztikával, akkor az betű-elcsúszásokhoz vezet, emiatt szinte lehetetlen a teljes decriptálás.

```
[windsake@windsake-pc strostrup]$ java RSA
character map of the private message: {a=2,b=1,v=1,i=1; l=1,m=1}tring[] args) {
Mncripted private msg l♦L♦
l 1      RSA.class
L 1      RSA.java
♦ 2      > yoda
1       asd.cpp
M 1
the result of the decryption: valami
[windsake@windsake-pc strostrup]$
```

16.3. ábra. Az rsa futás

Látható viszont a felső képnél, hogy ha minden betűt sikeresen eltalál, onnan már egész egyszerű kitalálnunk az eredeti szót. Ha nem sikerült volna megfejteni a szót, amire gondoltam, a szó: **valami**

*ervlel *v* *t* *ts CobBt v*äl*?*D*n is* de e*eken * l*irin*us. T
or?*r*?*ási b*fl.ákk*l is *e*is*erke*he* *j* * ke*ves *lv*s*f*

beriségnek a komplex problémákkal megküzdeni tudó algoritmuselmélés
veink szerint ennek az érzésnek az átadásához ezzel a digitális sza
k a programozással, annak elméletével és filozófiájával, majd részl
pelül a Java programozással. Ebben a kézikönyvben a Java nyelvvel e
során ezt az egyszerű, példa labirintus játékot elkészítjük a webol
tozatban is. De ezeken a labirintus témájú esettanulmányokon túl bi
herkedhet majd a kedves Olvasó.

16.4. ábra. Az rsa futás

Ezen a képen látható, hogy az utolsó két sor szinte pofon-egyszerűen kitalálható a törést követően, na persze nem 100%-os a törés, de egy pár futás után össze lehet illeszteni a szöveget.

17. fejezet

Helló, Gödel!

17.1. 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!

forráskód:[github link](#)

```
template<class T>
class TrackingAllocator
{
public:
    using value_type = T;

    using pointer = T *;
    using const_pointer = const T *;

    using size_type = size_t;

    TrackingAllocator() = default;

    template<class U>
    TrackingAllocator(const TrackingAllocator<U> &other) { }

    ~TrackingAllocator() = default;
```

A template segítségével bármilyen típusra alkalmazható az allokátor, a használni kívánt paramétereket a **using** kulcsszóval ellátott parancsok-nál láthatjuk. Maga a **T** lesz az értéktípus, a pointer ugyebár egy **T** típusú pointer lesz, ugyanez a constans pointer is, csak a const kulcsszóval ellátva. Továbbá deklarálunk egy méret-típust is a **size_t**-re.

Láthatjuk továbbá a konstruktort és dekonstruktort, melyekhez a default értéket párosítjuk. Ez az érték a compiler felé utasítást ad, miszerint ha nem adunk más értéket, akkor a compiler maga fog egy alapértelmezett értéket adni. Látható még egy fúrcsa szócska a **template<class U>** sor után, az a bizonyos **&other**. Ez egy ún. referencia, ez a referált érték memóriahelyét kapja, nincs külön memóriahelye. Itt a TrackingAllocator osztályon egy ún. overload-ot hajtunk végre.

```
class TrackingAllocator
{
    .
    .
    .

pointer allocate(size_type numObjects)
{
    mAllocations += numObjects;
    return static_cast<pointer>(operator new(sizeof(T) * numObjects));
}

void deallocate(pointer p, size_type numObjects)
{
    operator delete(p);
}
```

A programkód legfontosabb részletei az allokációért felelős **allocate** és a deallokációért felelős **deallocate** függvények. Ezek a TrackingAllocator osztályban foglalnak helyet.

az **allocate** függvényben láthatjuk, hogy a visszatérési értékben allokáljuk az új memóriahelyeket a new funkcióval, melyben megszámoljuk a T méretét, majd azt megszorozzuk az objektumok számával, amit ugye paraméterként kap az allokációs függvény. Ez egy ún. operátor túlterhelés/overload. Ezt az egészet egy static_cast-al átkonvertáljuk pointer típusára, és ezt a pointert fogja majd a függvényünk visszaadni. Ez a metódus a fordítás során castolja át a megadott paraméterként kapott értéket. Na persze ebben a függvényben adjuk az mAllocations változónak az értéket is, melyet majd a végén használunk fel, hogy megkapjuk, mennyi memóriahelyet allokáltunk.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

d: 33

Sorted numbers:
a: 7
d: 33
c: 41
b: 95
[windsake@windsake-pc gode1]$ ./alloc
bash: ./alloc: Permission denied
[windsake@windsake-pc gode1]$ c++ alloc.cpp -o alloc
[windsake@windsake-pc gode1]$ ./alloc
allokált memóriahelyek száma:3
[windsake@windsake-pc gode1]$
```

17.1. ábra. Az allokátor futás

```
std::cout << "allokált memóriahelyek címe:" << std::endl;
for (auto i=0;i<v.size();++i){

    std::cout << &v[i] << std::endl;
```

```
}
```

A memóriahelyek címe és a program futása.

```
[windsake@windsake-pc prog2]$ cd gode1/
[windsake@windsake-pc gode1]$ ls
alloc alloc2.png alloc.cpp alloc.png
[windsake@windsake-pc gode1]$ g++ alloc.c
[windsake@windsake-pc gode1]$ ./alloc
allokált memóriahelyek száma:3
allokált memóriahelyek címe:
0x55c769e1eeb0
0x55c769e1eed0
0x55c769e1eef0
[windsake@windsake-pc gode1]$ █
```

17.2. ábra. Az allokátor2 futás

17.2. STL map érték szerinti rendezése

Például: <https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180>

forráskód:[github link](#)

```
#include <iostream>
#include <map>

int main(void) {
    srand(time(NULL));
    std::map<char, int> map;
    std::map<int, char> sortedMap;

    for(char c = 'a'; c <= 'd'; c++) {
        map.insert({c, (rand()% 100 + 0)});
    }
```

```
std::cout << "unsorted numbers: " << std::endl;
for(auto const numb: map){
    std::cout << numb.first << ":" << numb.second << std::endl;
    sortedMap[numb.second] = numb.first;
}

std::cout << std::endl << "Sorted numbers: " << std::endl;
for(auto const numb: sortedMap)
    std::cout << numb.second << ":" << numb.first << std::endl;

return 0;
}
```

a map-ek használatához szükségünk van a map library-ra, ezek olyan tárolók, melyeknek van egy azonosítója (egy kulcs) és van egy értéke. Két map rekordnak az azonosítója soha nem egyezik/egyezhet meg.

```
int main(void) {
    srand(time(NULL));
    std::map<char, int> map;
    std::map<int, char> sortedMap;
```

Először létrehozunk két map-et, az egyik tárolja majd a rendezetlen, a másik a rendezett számokat. Az **std::map** "paraméterei" a következők: az első, kacsacsőr utáni paraméter a kulcs érték típusa, a második param. pedig a map érték. Így tehát az unsorted-map kulcsértéke char típusú lesz, és számokat tárol, a sortedMap kulcsértéke viszont már szám típusú, és karaktereket tárol.

```
for(char c = 'a'; c <= 'd'; c++) {
    map.insert({c, (rand() % 100 + 0)});
}
```

A deklarálás után feltöljük az unsorted mapunkat. Ehhez egy for-ciklus lesz segítségünkre.

A ciklus a-tól d-ig fog menni, és minden tick-ben insertel egy adatpárt a map-ba. A map.insert szintaktikája a következő:

map_név.insert({kulcs, map_érték});

A mapunk értékét randomizált 100-ig terjedő számokkal töltjük meg, a kulcs pedig a ciklus-változó értéke lesz.

```
std::cout << "unsorted numbers: " << std::endl;
for(auto const numb: map){
    std::cout << numb.first << ":" << numb.second << std::endl;
    sortedMap[numb.second] = numb.first;
}
```

Harmadik lépésként kiíratjuk az unsorted-map értékpárosait, ezt is for-ciklus segítségével. Észrevehetjük, hogy a for-ciklus szintaxisa nem éppen az előzőhez hasonló. Ezt a for ciklust range-based for-nak hívják. Ennek a szintaktikája a következő:

for(típus1 változó: vektor/map/array/etc.)

Fontos tudni viszont, hogy a változónak azonos típusúnak kell lennie a map-ban levő érték típusával. Mivel a mi esetünkben auto típust adtunk meg, ezért a típusmegválasztást a compiler-re hagyjuk, az pedig a map értékeit figyelembe véve választ majd típust.

```
std::cout << std::endl << "Sorted numbers: " << std::endl;
for(auto const numb: sortedMap)
    std::cout << numb.second << ":" << numb.first << std::endl;
```

Végül kiíratásra kerül a sortedMap tartalma, amely az előbbi ciklushoz hasonlóan zajlik.

```
26 | return 0;
27 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

[windsake@windsake-pc gode1]$ ./stl
unsorted numbers:
a: 82
b: 80
c: 61
d: 34

Sorted numbers:
d: 34
c: 61
b: 80
a: 82
[windsake@windsake-pc gode1]$ 
```

17.3. ábra. Az stl futás

17.3. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témát (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala

Tanulságok, tapasztalatok, magyarázat... A megszokott-tól eltérően a nyelvben nem infix alakban adjuk meg a műveleteket, hanem prefix alakban.

```
(define (elem x lista)

  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )

)
```

Mint láthatjuk, itt a define kulcsszóval lehet függvényeket inicializálni. Ez a függvény az "elem" névre hallgat, és 2 paramétert kér be, az "x"-et és a "lista"-át. Ha az x=1, akkor a (car lista) parancs hajtódi

végre, ha nem, akkor pedig az (elem (x-1) (cdr lista)) parancs. Ez miatt a sor miatt a függvény rekurzív lesz, hiszen addig hajta végre az "else" ágat, amíg az $x \neq 1$. A car és cdr függvények a lista elején ill. végén lévő elemet adják vissza.

```
(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
  )
)
```

Ez a függvény a szöveg szélességét kezeli. paraméterként bekéri a szöveget, a betűtípusit és a betűméretet. A függvény törzsében a megadott paraméterekkel a gimp beállítja a szöveg kinézetét.

```
(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
  gradient)
(let*
  (
```

Ezen kezdetű függvény tekinthető a main fügvénynek, hiszen itt alkalmazzuk az előbb megadott függvényeket, itt írjuk a program érdemi részét.

```
(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)
```

Az első sor beilleszt egy layert, a második sor beállítja annak háttérszínét zöldre, a harmadik sor kitölti a háitteret, az utolsó sor pedig újra aktiválja a képet.

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>



18.1. ábra. Az editor futás

látható, hogy a tevékenység-fa-ba kattintás hatására a "tevékenységekhez hozzárendelt tulajdonságok" szöveg eltűnik, ez a szöveg az **actPropsLabel** objektumban található. Ezen bug kiküszöbölhető, ha az else ágból kiveszük a `actPropsLabel.setText("");` sort, így csak akkor tűnik el a szöveg, ha egy tevékenységre rákattintunk. Viszont a szerekesztés után még ekkor sem jelenik meg újra az alapértelmezett szöveg, helyette üres marad a label. A szöveget visszahozhatjuk, ha az `onClicked` event-ben az `item!=null` elágazás után beillesszük ezt a pár sort:

```
class FileTree extends javafx.scene.control.TreeView<java.io.File> {
    static final String actlab="A tevékenységekhez hozzárendelt ←
        tulajdonságok";
    .
    .
    .

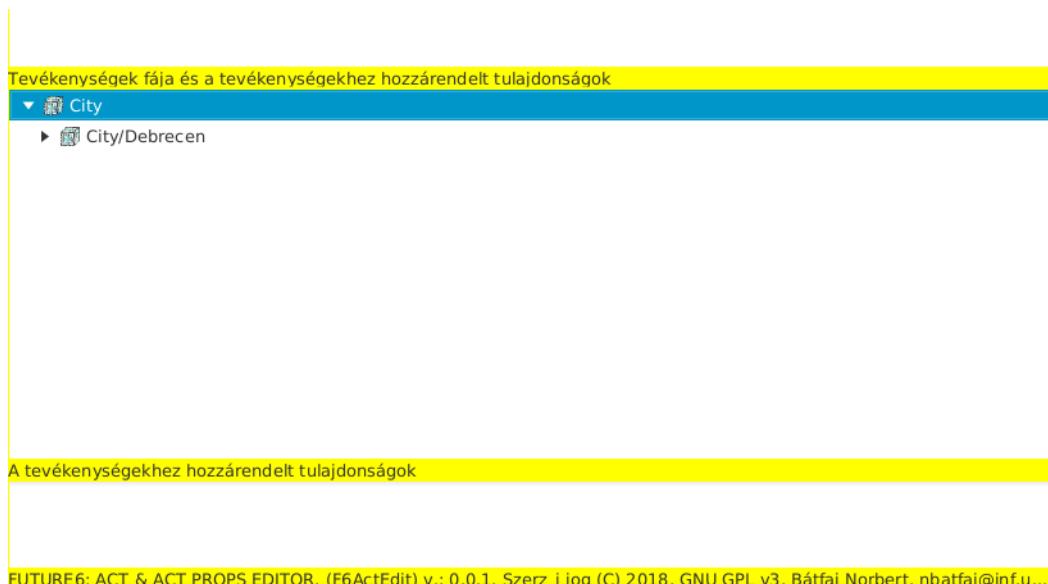
/*ez az if(item!=null) utolsó ága*/} else {

    save(propsEdit, actPropsLabel);
    propsEdit.setText("");

}

if(item==root){
    save(propsEdit, actPropsLabel);
    actPropsLabel.setText(actlab);
}
```

Ez azt fogja tenni, hogy ha rákattintunk a City-re, ami a gyökérkönyvtár, akkor elmenti a `propsEdit`-ben található szöveget, majd az `actPropsLabel` szövegét újra beállítja az alapértelmezettre, ezt a szöveget az `actlab` változóban tároljuk el.



18.2. ábra. Az editor javított futása

Sajnos a java 8.1-től nem található meg a java-ban a JavaFX, ezért ezt külön telepítenünk kell, A program

lefordításához szükségünk van az openjfx 8.u202-3 verziójára(minimum):



18.3. ábra. A javafx

kód snippetek a változtatásról:

```
import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import javafx.scene.control.TextField;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundImage;

.
.
.

//410-dik sor
public class ActivityEditor extends javafx.application.Application {
```

a paint.Color osztályt azért importáltuk, mivel használni fogjuk a Color.GREENYELLOW parancsot, na persze bármely színt használhatjuk, miután a class importálva lett. A paint.Paint osztály felel a háttér "befestésséért", de más ábrák, szövegdobozok és egyéb színezéséért is felelős. Kódunkban a fill objektum (alul) egy Paint típusú változó. A TextField-et használjuk a szövegmezők megjelenítéséhez, a "tevékenységekhez hozzárendelt tulajdonságok" label alatti rész egy ilyen textfield. A Background és BackgroundFill class-ok a háttér-ért felelősek. A BackgroundFill teszi lehetővé a háttér színezését, egyedül a Paint nem elegendő. A 410-edik sor-tól kezdődik az editor megjelenítésének "leírása".

```
private static Paint fill;
.

.

public void start(javafx.stage.Stage stage) {
```

```
        .
        .
        .

javafx.scene.layout.VBox box = new javafx.scene.layout.VBox();
    box.setBackground(new Background(new BackgroundFill(null,null,null) ←
        ));

String style = "-fx-background-color: rgba(255, 255, 0, 5);";

final javafx.scene.Scene scene = new javafx.scene.Scene(box ←
    ,800,400,Color.GREENYELLOW);

        .
        .

javafx.scene.control.TreeView<String> stringTree = new StringTree( ←
    properties, true, propsEdit);
    stringTree.setEditable(false);
    stringTree.setBackground(new Background(new BackgroundFill(fill, ←
        null,null)));
    .

        .
        .

box.getChildren().add(new javafx.scene.control.Label("Tulajdonságok ←
    fája"));

box.getChildren().add(stringTree);
box.getChildren().add(new javafx.scene.control.Label("Tevékenységek ←
    fája és a tevékenységekhez hozzárendelt tulajdonságok"));
box.getChildren().add(fileTree);
box.setStyle(style);

javafx.scene.layout.VBox box = new javafx.scene.layout.VBox();
    box.setBackground(new Background(new BackgroundFill(null,null,null ←
        ));
```

Ez a sor létrehoz egy vertikális "teret", és a gyermeket mind egy sorba rendezi ezen a téren belül. A setBackground metódusban a BackgroundFill paraméterként null-t kapott, tehát a háttér transzparens/átlátszó. A BackgroundFill második paramétere a CornerRadii, ami a sarok-lekerekítések rádiuszát/sugarát adja meg, a harmadik pedig az inseteket adja meg, ezek a keretek. A használatukhoz be kell importálnunk a javafx.geometry.Insets osztályt.

```
String style = "-fx-background-color: rgba(255, 255, 0, 5);";
```

Ez egy narancssárgás színárnyalatot kölcsönöz a GREENYELLOW szín mellé., az objektumot később, a setStyle metódussal használjuk fel.

```
final javafx.scene.Scene scene = new javafx.scene.Scene(box,800,400,Color. ←
    GREENYELLOW);
```

A scene lesz a "vászon", ezen foglal helyet minden, ha a szövegdobozok háttérszíne nem lenne fehér, akkor láthatnánk, hogy az egész ablak zöldes-sárga.

```
javafx.scene.control.TreeView<String> stringTree = new StringTree( ←
    properties, true, propsEdit);
    stringTree.setEditable(false);
    stringTree.setBackground(new Background(new BackgroundFill(fill, ←
        null, null)));
```

Ez a parancs hoz létre egy fa ágazat menüt, paraméterekként egy listát(properties), egy boolean típusú paramétert (az expanded opció ki és bekapcsolására) és egy TextArea szövegmezőt tartalmaz. A setEditable funkció segítségével állíthatjuk be, hogy az ágazatunk szerkeszthető legyen-e. Végül itt is beállítjuk a hátteret.

```
box.getChildren().add()
```

Ez a függvény adja hozzá az üres tárolónkhöz a sorokat.

18.2. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>

```
#include "opencv2/objdetect.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
```

<https://github.com/opencv/opencv/blob/master/modules/videoio/include/opencv2/videoio.hpp>

```
class SamuCam : public QThread
{
    Q_OBJECT

public:
    SamuCam ( std::string videoStream, int width, int height );
    ~SamuCam();

    void openVideoStream();
```

Az opencv segítségével képes a program a kamera használatára, a SamuCam osztály egy QThread objektumot képvisel, ezen belül található az openVideoStream, melyet majd a felvétel kezeléséhez használunk a cv::VideoCapture **videoCapture**; segítségével.

```
void SamuCam::openVideoStream()
{
    videoCapture.open ( videoStream );

    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
    videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
    videoCapture.set ( CV_CAP_PROP_FPS, 10 );
}
```

Itt láthatjuk az openVideoStream felhasználását. A videoCapture.open indítja el a felvételt, paraméterként pedig egy fájlnevet kap meg. A metódus először hívja a **VideoCapture::release**-t, ha a felvétel már folyamatban van.

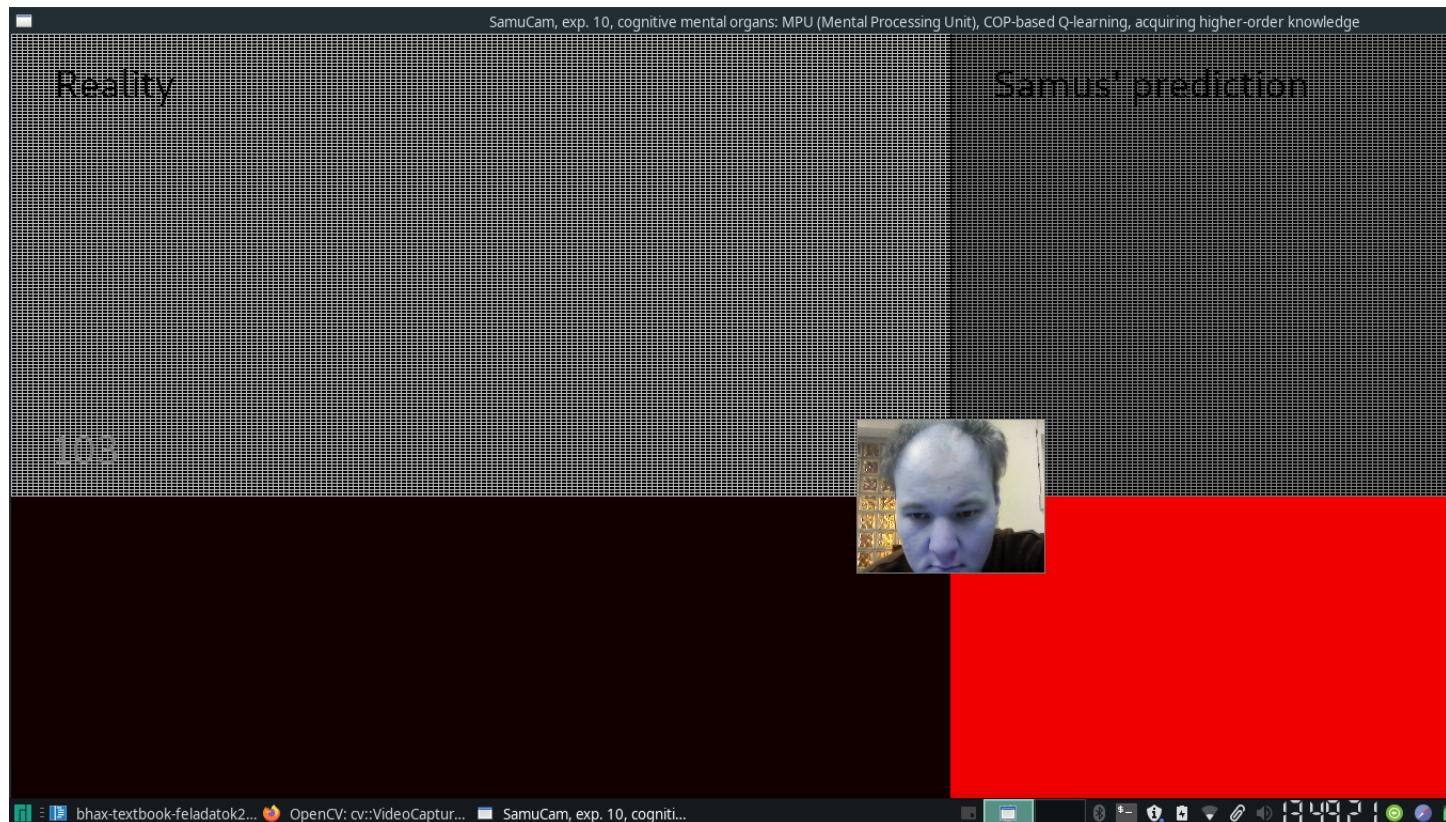
Alatta beállítjuk a felvétel szélességét és magasságát, valamint az fps rátát.

```
if ( ! videoCapture.isOpened() )  
{  
    openVideoStream();  
}
```

Ezzel a sorral hívjuk meg a felvételt.

```
void SamuCam::run()  
{  
  
    cv::CascadeClassifier faceClassifier;  
  
    std::string faceXML = "lbpcascade_frontalface.xml"; // https://github.com ←  
    // Itseez/opencv/tree/master/data/lbpcascades  
  
    if ( !faceClassifier.load ( faceXML ) )  
    {  
        qDebug() << "error: cannot found" << faceXML.c_str();  
        return;  
    }
```

Található továbbá a samu::run-ban egy **cv::CascadeClassifier** is, ez az objektumok értelmezésére hivatott, használatához az objdetect library behívása szükséges.



18.4. ábra. A samu futás

a faceXML fájl egy osztályozást tárol emberi arcokról, ha a program nem tudja megnyitni a fájlt (mint lát-hatjuk, a program feltételezi, hogy ez a bizonyos xml fájl azonos mappában található a cpp fájl-al.), akkor egy qDebug funkció-n keresztül dob egy hibaüzenetet. Ez a Qdebug library-ban található, használatával egyszerűbbé válik a hibák értelmezése. Például a **QDate::currentDate()** metódussal lehetőségünk van a dátum megjelenítésére a hibaüzenetben.

18.3. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esport-talent-search>

```
BrainBWin::BrainBWin ( int w, int h, QWidget *parent ) : QMainWindow ( ←
    parent )
{
    //    setWindowTitle(appName + " " + appVersion);
    //    setFixedSize(QSize(w, h));

    statDir = appName + " " + appVersion + " - " + QDate::currentDate() ←
        .toString() + QString::number ( QDateTime::←
        currentMSecsSinceEpoch() );
```

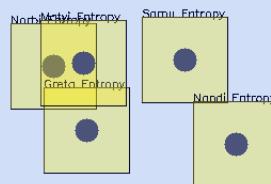
```
brainBThread = new BrainBThread ( w, h - yshift );
brainBThread->start ();

connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ) ,
          this, SLOT ( updateHeroes ( QImage, int, int ) ) );

connect ( brainBThread, SIGNAL ( endAndStats ( int ) ),
          this, SLOT ( endAndStats ( int ) ) );

}
```

Press and hold the mouse button on the center of Samu Entropy
0:0/10:0 0 bps PAUSED (0)



18.5. ábra. A brainb futás

ez a slot-signal mechanizmus az objektumok közötti kommunikációt teszi lehetővé. Ez a meta-object rendszer segítségével lehetséges.

Ez a fajta szintaktika már elavultnak számít, helyette itt látható az új szintakszis. Más keretrendszerök callback funkciójához hasonlóan működik, viszont a callback-el ellentétben ez a mechanizmus nem küszködik típus-tartási hibákkal, hiszen **csak akkor hajtódik végre a slot-ban helyetfoglaló függvény, ha a jel típusa megegyezik a slot-ban várt típussal.** A slotban található funkció virtuálisként is definiálható, és teljes mértékben egy normál funkcióval megegyező függvény található a slotban. Tehát ha a heroesChanged függvény végrehajtódi, akkor egy jel küldődik az updateHeroes slotba, és ezzel az updateHeroes függvény is végrehajtásra kerül.

```
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 );

}
```

A brainBThread header fájlban találunk rá erre a signál-ra,a draw függvényben.

az endAndStats szignál akkor kerül elküldésre, ha lejár az idő, ezt a slot "elkapja" és végrehajtódik az ugyanezzel a névvel ellátott függvény, amely a következőt jeleníti meg:

```
void BrainBWin::endAndStats ( const int &t )
{

    qDebug() << "\n\n\n";
    qDebug() << "Thank you for using " + appName;
    qDebug() << "The result can be found in the directory " + statDir;
    qDebug() << "\n\n\n";

    save ( t );
    close();
}
```

megköszöni az app használatát, kiírja a statisztikák "lelőhelyét", elmenti az időt és végül bezárja az ablakot.

19. fejezet

Helló, Schwarzenegger!

19.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! <https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/ch01.html#id527287>

The screenshot shows a Java code editor with the file 'KapuSzkenner.java' open. The code is a simple port scanner that tries to connect to ports 0 to 1024 on the first argument passed to it. If a connection is successful, it prints 'figyeli'; if not, it prints 'nem figyeli'. The code is annotated with line numbers from 1 to 17. Below the code editor is a terminal window showing the execution of the program. The output consists of 1024 lines, each containing either 'figyeli' or 'nem figyeli', corresponding to the results of the port scans. The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with the TERMINAL tab selected. The status bar at the bottom indicates '1: bash'.

```
schwarz > KapuSzkenner.java > KapuSzkenner > main(String[])
1  public class KapuSzkenner {
2
3      public static void main(String[] args) {
4
5          for(int i=0; i<1024; ++i)
6
7              try {
8                  //String s=null;
9                  java.net.Socket socket = new java.net.Socket(args[0], i);
10
11                  System.out.println(i + " figyeli");
12
13                  socket.close();
14
15              } catch (Exception e) {
16
17                  System.out.println(i + " nem figyeli");
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1010 nem figyeli
1011 nem figyeli
1012 nem figyeli
1013 nem figyeli
1014 nem figyeli
1015 nem figyeli
1016 nem figyeli
1017 nem figyeli
1018 nem figyeli
1019 nem figyeli
1020 nem figyeli
1021 nem figyeli
1022 nem figyeli
1023 nem figyeli
[windsake@windsake-pc schwarz]$
```

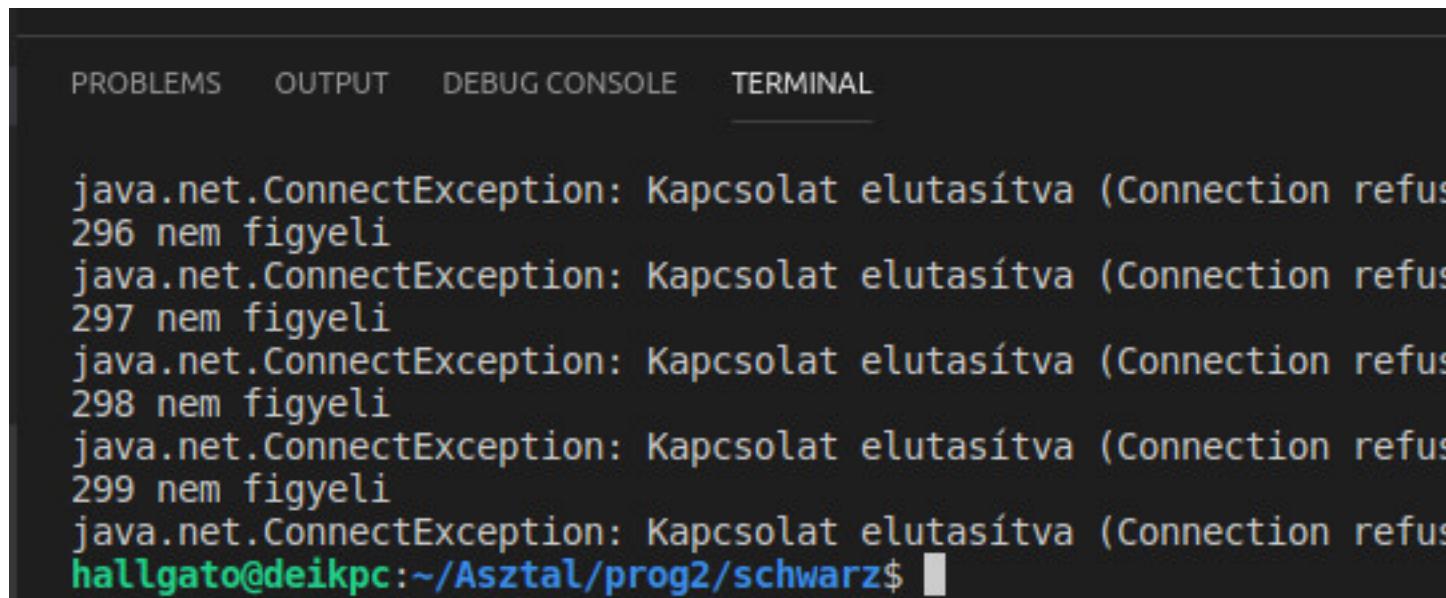
19.1. ábra. A socket futás

```
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");  
  
    }
```

Hogy lássuk, mi az az exception, amit dob, amikor a "nem figyeli" szöveg jelenik meg, ki kell íratnunk az e objektum tartalmát. Tehát az előző programkód így bővül:

```
System.out.println(i + " nem figyeli\n" + e);
```



The screenshot shows a terminal window with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
  
java.net.ConnectException: Kapcsolat elutasítva (Connection refused)  
296 nem figyeli  
java.net.ConnectException: Kapcsolat elutasítva (Connection refused)  
297 nem figyeli  
java.net.ConnectException: Kapcsolat elutasítva (Connection refused)  
298 nem figyeli  
java.net.ConnectException: Kapcsolat elutasítva (Connection refused)  
299 nem figyeli  
java.net.ConnectException: Kapcsolat elutasítva (Connection refused)  
hallgato@deikpc:~/Asztal/prog2/schwarz$ █
```

19.2. ábra. A socket Exception

Ez az exception akkor dobódik, ha valami hiba lépett fel a távoli ip-cím-hez csatlakozáskor. A mi esetünkben ez a hiba az, hogy nincs figyelő folyamat a cél ip/port együttesen. [A leírása megtalálható angolul itt.](#)

```
631 igen figyeli  
632 nem figyeli  
java.net.ConnectException: Connection refused (Connection refused)  
633 nem figyeli
```

19.3. ábra. A socket eredménye

Látható, hogy a 631-es porton van figyelő folyamat, tehát a programunk megfelelően működik, hiszen talált egy figyelt portot is. Na persze az nem jelentené, hogy hibás a program, ha nem találtunk volna 1 figyelt portot se.

19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

források:[Lzw.java](#) [LZWBinFa.java](#) [atszovo_lzw.aj](#)

```
[windsake@windsake-pc schwarz]$ java Lzw lzwcucc.txt -o lzw_kesz
getAtlag meghívása előtt ez a szöveg megjelenik
getAtlag meghívása előtt ez a szöveg megjelenik
[windsake@windsake-pc schwarz]$
```

19.4. ábra. A szövés eredménye

Alul láthatjók, hogy a getAtlag függvény meghívódik a getSzoras függvényen belül, tehát mindenleg 2x kell majd viszont látnunk az átszölt szöveget.

```
public double getAtlag() {
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag(gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}

public double getSzoras() {
    atlag = getAtlag();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;
    rszoras(gyoker);
    if (atlagdb - 1 > 0) {
        szoras = Math.sqrt(szorasosszeg / (atlagdb - 1));
    } else {
```

19.5. ábra. A szövés függvényei

Itt látható, hogy a getAtlag és a getSzoras egymás után hívódnak meg.

```
kiFile.println("mean = " + binFa.getAtlag());
kiFile.println("var = " + binFa.getSzoras());
```

Itt pedig láthatjuk az aspectj forráskódot:

```

public aspect atszovo_lzw {

    public pointcut fgvHívás(): call(public double LZWBinFa.getAtlag());

    before(): fgvHívás() {
        System.out.println("getAtlag meghívása előtt ez a szöveg megjelenik ←
                           " );
    }
}

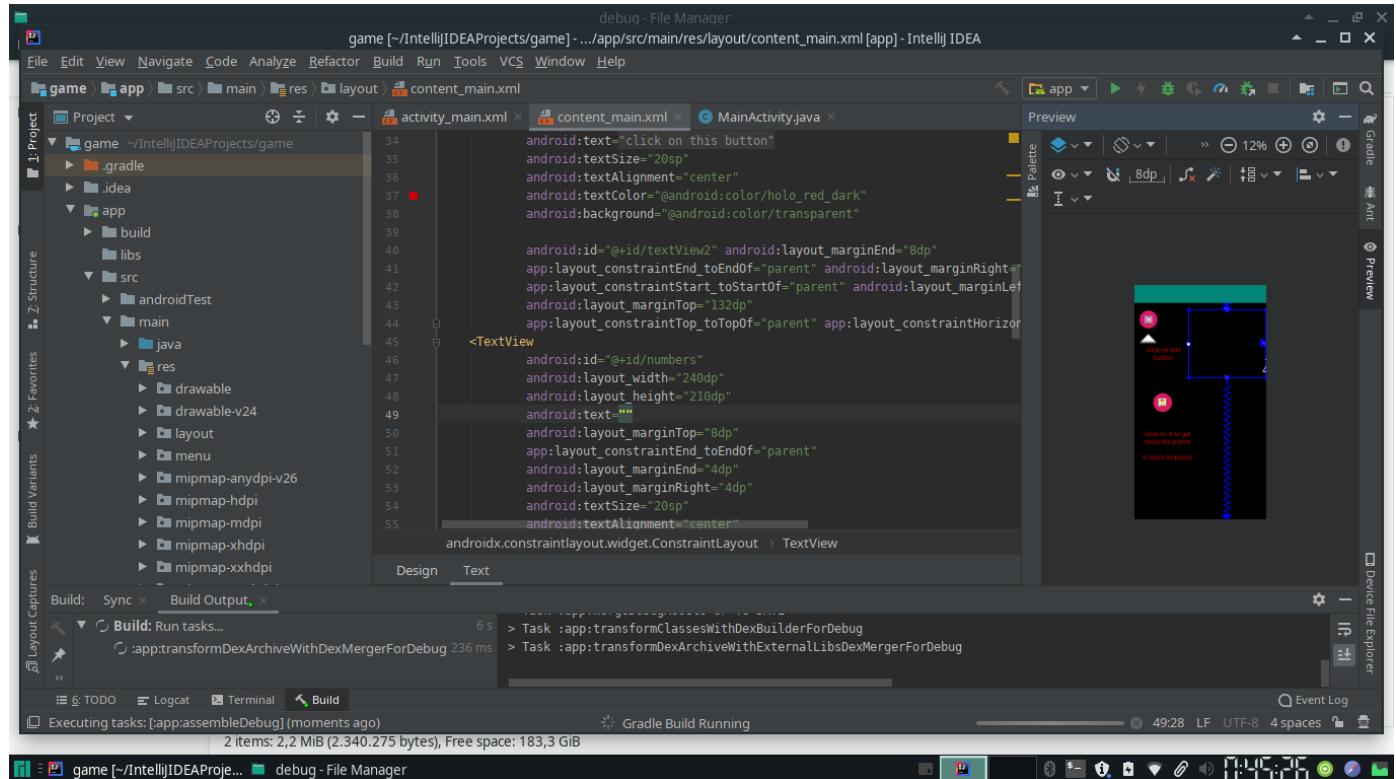
```

A program a pointcut kifejezést használja, a join point egy olyan pont a programban, amelyet használhatunk egy aspektus készítéséhez. Ilyen pont például egy függvény meghívása vagy egy kivétel lekezelése. A pointcut egy olyan predikátum, amely a join point-ra hivatkozik. Az új verziójú aspectJ-ben ezt a kifejezést annotációval hívjuk meg: @Pointcut

A pointcut-ról bővebben itt: [pointcut](#)

19.3. Android Játék

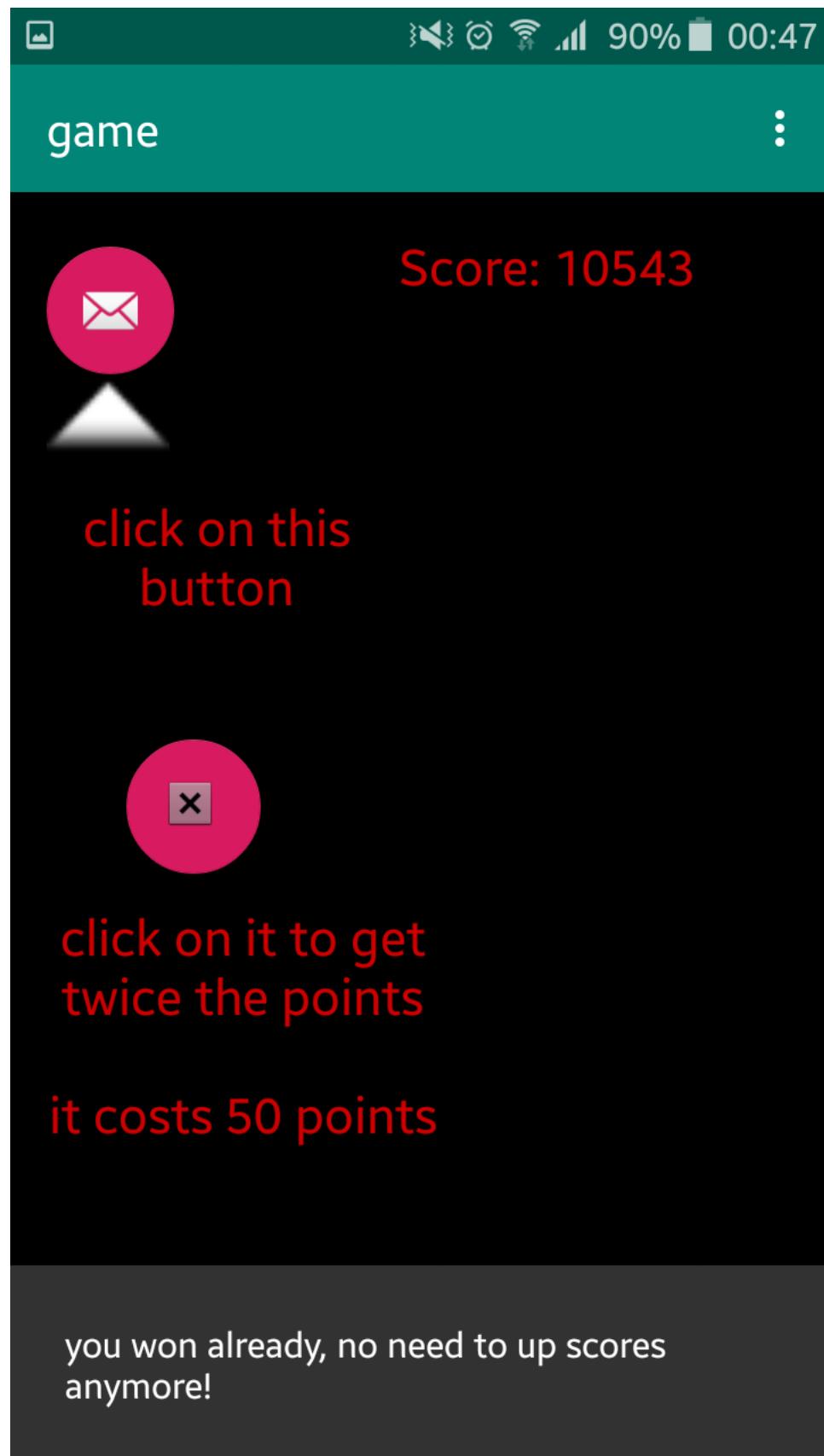
Írunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Hello, Android!” feladatára!



19.6. ábra. A játék proto

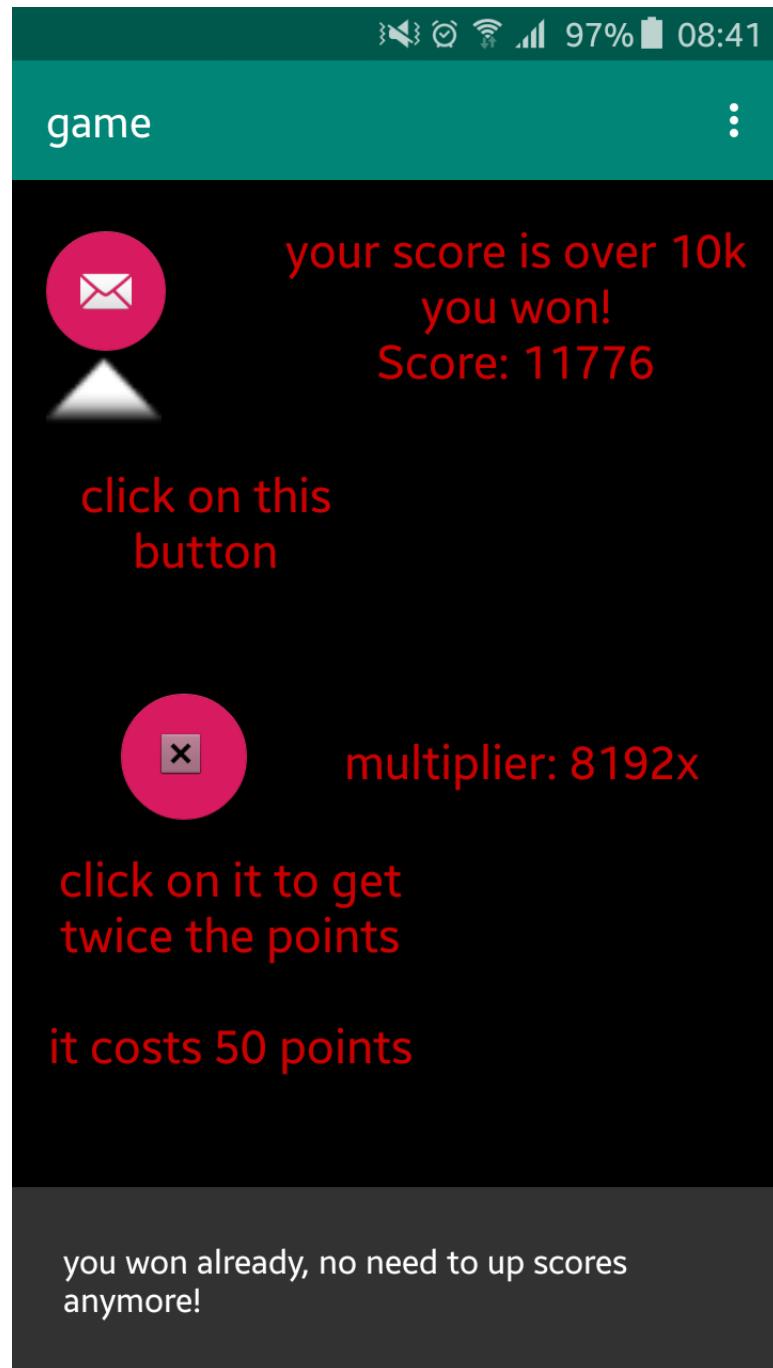
Az applikáció készítése, IntelliJ IDEA szoftver segítségével. Maga a program dolgozhatóvá tétele vette el

a legtöbb időt, sajnos a laptop kevés memóriája miatt kompromisszumokat kellett kötnöm, nem tudtam használni a virtuális android készüléket. Az applikáció készítése során egyre egyszerűbbé vált a fejlesztés, míg eleinte hosszadalmas volt 1-1 bug javítása is.



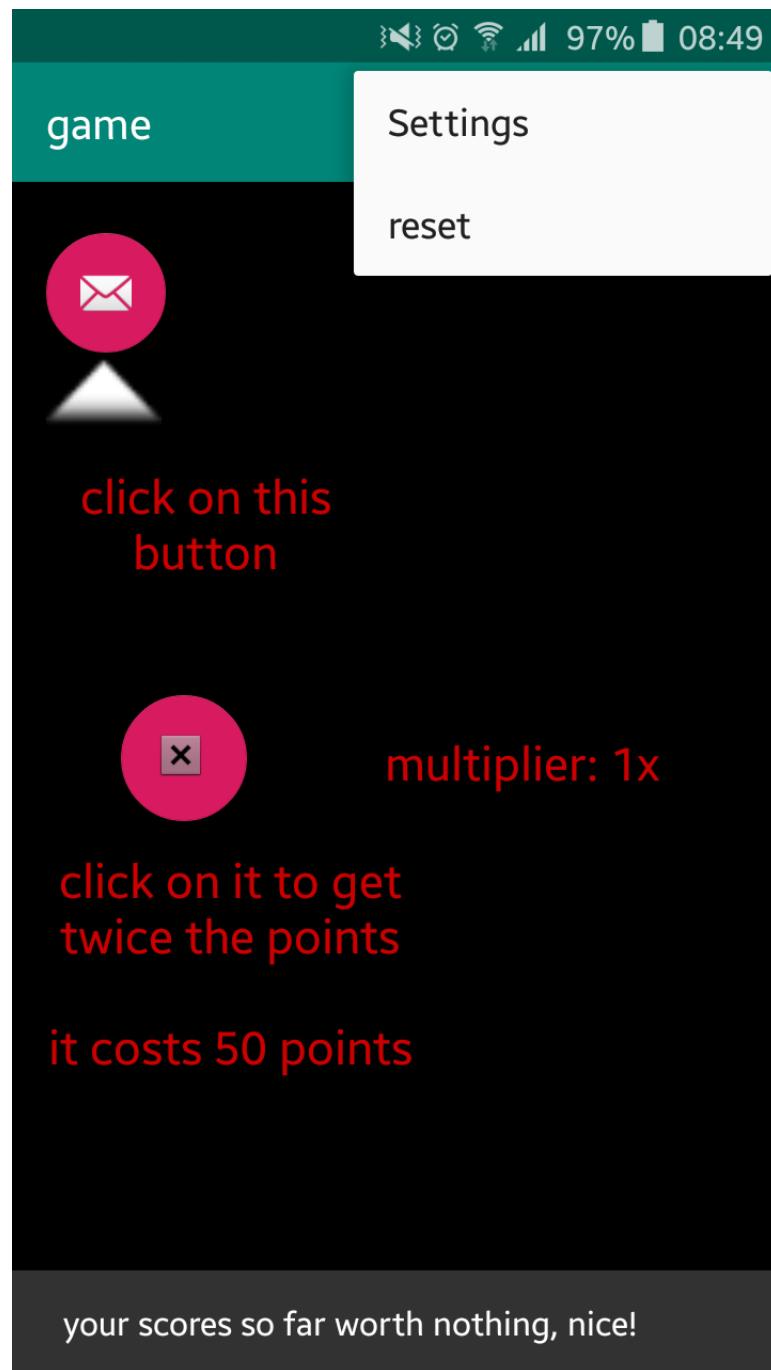
19.7. ábra. A játék proto

A játék 10k pontnál véget ér, na persze ez még csak egy alpha verzió.



19.8. ábra. A játék proto2

Újabb verzióként belekerült egy multiplier számláló is, és a menüben resetelhető a progress.



19.9. ábra. A játék reset

Az apk megtalálható [itt](#)

Az xml fájlok és a main fájl pedig [itt](#)

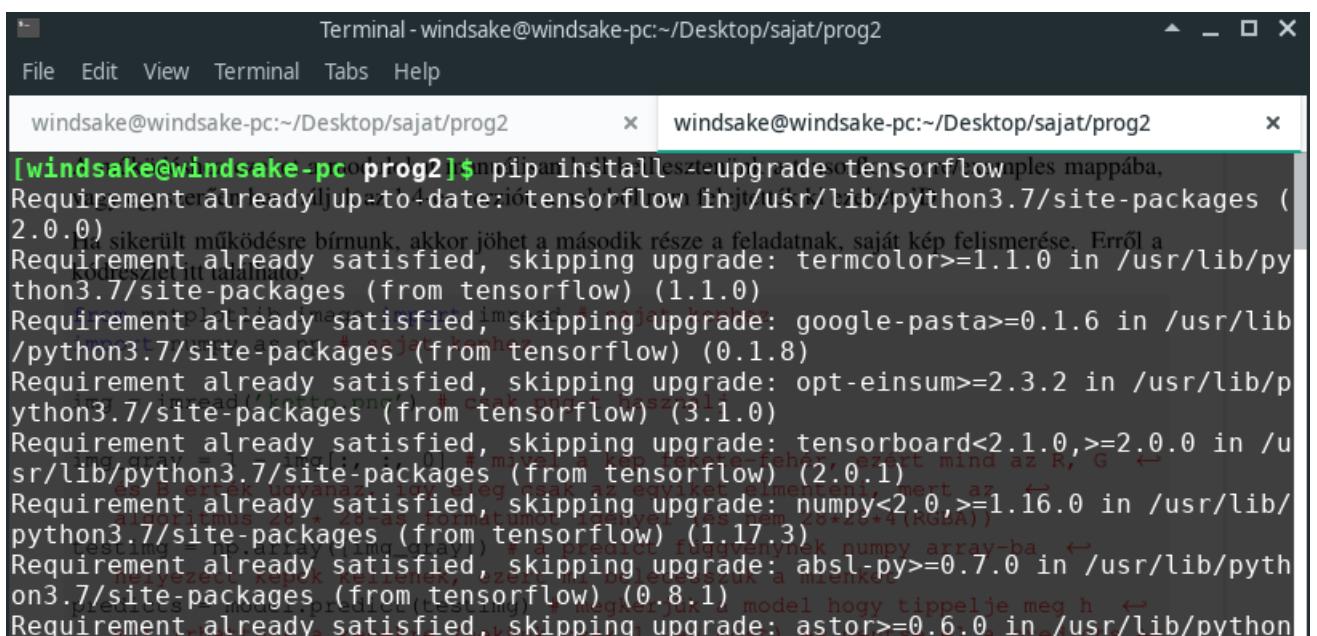
20. fejezet

Helló, Calvin!

20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13/hello_sajat.html bol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

A futtatáshoz szükségünk lesz a tensorflow library meglétére a gépen. A telepítésre 2 módszert használtam, a packet-manager-en keresztül való telepítést és a python pip-es telepítést. A második módszerhez a python is szükséges, viszont mindenkor le kell töltenünk azt, hiszen a program is python nyelven íródott. A python-t a packet-manager használatával szinte pillanatok alatt feltelepíthetjük és már használatra kész. Ennek a 3.7-es verziójára lesz szükségünk, amíg a tensorflow 2.0 vagy 1.4 között választhatunk.



The screenshot shows a terminal window with two tabs. The active tab displays the command `windsake@windsake-pc:~/Desktop/sajat/prog2$ pip install tensorflow` and its output. The output shows several requirements already satisfied, such as `termcolor >= 1.1.0`, `google-pasta >= 0.1.6`, `opt-einsum >= 2.3.2`, `tensorboard >= 2.1.0, < 2.0.0`, `numpy >= 1.16.0`, `absl-py >= 0.7.0`, and `astor >= 0.6.0`. The terminal window has a dark theme and is running on a Linux system.

20.1. ábra. A pip telepítés

Láthatjuk a képen, hogy nálam már up-to-date a tensorflow, de ha először írjak be a parancsot, akkor 1-2 percen át telepítgeti a szükséges fájlokat. **Fontos még megemlíteni, hogy néhány distrón permission**

denied lesz a vége, ha sudo nélkül, vagy su nélkül próbálkozunk az installációval, másrészről viszont van, ahol anélkül is tökéletesen feltelepítődik. A biztonság kedvéért használjuk a sudo parancsot a pip előtt.

Itt viszont fontos megjegyezni, hogy az újabb verzió nem mindig jobb, mivel a 2.0-ás verzióban hiányoznak olyan modulok, amelyek létfontosságúak lennének a jelenlegi programunkhoz(**matplotlib.pyplot, tensorflow.examples.tutorials.mnist**)

A működéshez ezeket a modulokat manuálisan kell beillesztenünk a tensorflow-core/examples mappába, vagy egyszerűen használjuk az 1.4-es verziót, amelyből nem felejtették ki ezeket :’D

Ha sikerült működésre bírnunk, akkor jöhet a második része a feladatnak, saját kép felismerése. Erről a kódrészlet itt található:

```
from matplotlib.image import imread # sajat kephez
import numpy as np # sajat kephez

img = imread('ketto.png') # csak png-t hasznalj

img_gray = 1 - img[:, :, 0] # mivel a kép fekete-fehér, ezért minden az R, G ←
    # és B érték ugyanaz, így elég csak az egyiket elmenteni, mert az ←
    # algoritmus 28 * 28-as formátumot igényel (és nem 28*28*4(RGBA))
testimg = np.array([img_gray]) # a predict függvénynek numpy array-ba ←
    # helyezett képek kellenek, ezért mi beletesszük a miénket
predicts = model.predict(testimg) # megkérjük a model hogy tippelje meg h ←
    # mit irhattunk a kepekre (nekünk csak 1 van most) és mentse el a predicts ←
    # valtozozba tömbként (pl. [szerintem8, talán7, biztos3])
print("sajat kep: ")
print( np.argmax(predicts[0]) )
"""
megnézzük hogy az else (nulladik) képre mit tippelt. Ehhez szükség van ←
az argmax függvényre, mert a model 10 értéket ad vissza egy kép ←
tippelésekor (pl. szerinte ez 10%-ra 1-es, 20%-ra 2-es...), de nekünk ←
csak a legnagyobb kell, amiben a legbiztosabb """

```

Az egész forráskód megtalálható ezen a [linken](#)

a regresszióról és a modellről itt találhatunk leírást.

A modellünk:

```
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, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test, verbose=2)
```

A model maga az az objektum, amit tanítunk a program futása során. A modellen belül taláhatóak a layerek, amelyek 1-1 tanítási blokk-nak felelnek meg. A layerek-et elérjük a model.layers parancssal.

```
55680/60000 [=====] - ETA: 0s - loss: 0.074
56224/60000 [=====] - ETA: 0s - loss: 0.074
56800/60000 [=====] - ETA: 0s - loss: 0.074
57344/60000 [=====] - ETA: 0s - loss: 0.074
57888/60000 [=====] - ETA: 0s - loss: 0.074
58432/60000 [=====] - ETA: 0s - loss: 0.074
59008/60000 [=====] - ETA: 0s - loss: 0.074
59552/60000 [=====] - ETA: 0s - loss: 0.074
60000/60000 [=====] - 6s 101us/sample - loss: 0.0375 - accuracy: 0.9770
[2019-11-25 23:42:45.675910: W tensorflow/core/framework/cpu_allocator.cc:110] Allocation of 62720000 exceeds 10% of system memory.
10000/1 - 1s - loss: 0.0375 - accuracy: 0.9770
sajatakep:tbook-fdl.pdf
2ax-textbook-fdl.pdf' successfully built
[windsake@windsake-pc calvin]$
```

20.2. ábra. Az mnist futása



20.3. ábra. A kettes

20.2. Android telefonra a TF objektum detektálója

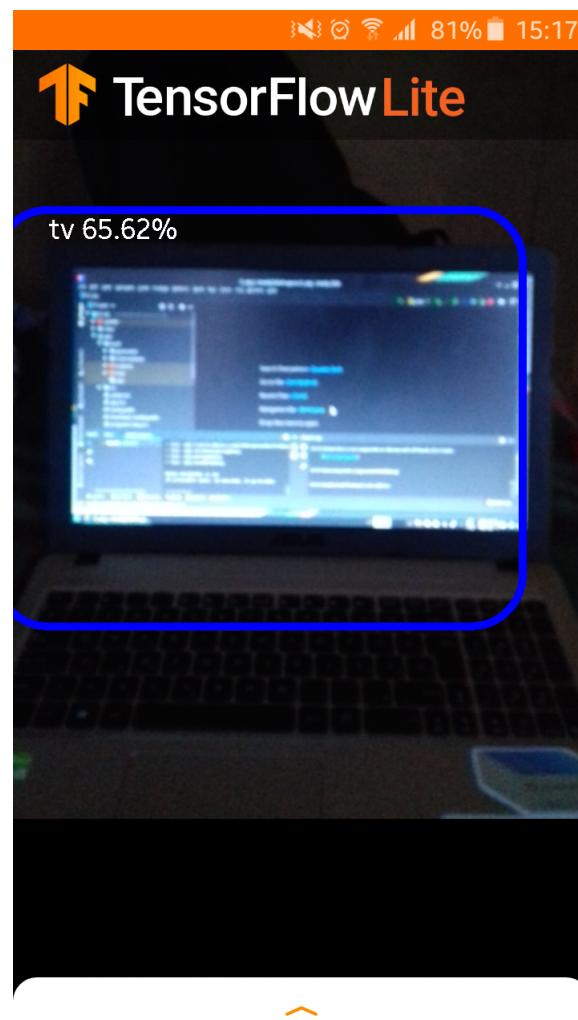
Telepítsük fel, próbáljuk ki!

A feltelepítéséhez szükségünk van egy Android studió-ra, vagy olyan IDE-re, ami képes az Android Studio-t plugin-ként kezelni. Én az IntelliJ IDEA programot használom erre.

A program működéséhez a gradle modult frissitenünk kell 4.8-as verzióra, melyet az IDE magától meg fog tenni, erre azért van szükség, mert a 4.6-os gradle nem képes a 10-es JAVA-verziószámú-tól nagyobb verziót kezelni. Ha a gradle update megvan, akkor szinkronizálnunk kell a beimportált projektet, majd le-buildelni. Buildelésnél le fog tölteni elég sok dependency-t, viszont ezen csak 1x kell átesnünk, azután pár másodperc alatt felépül a program.

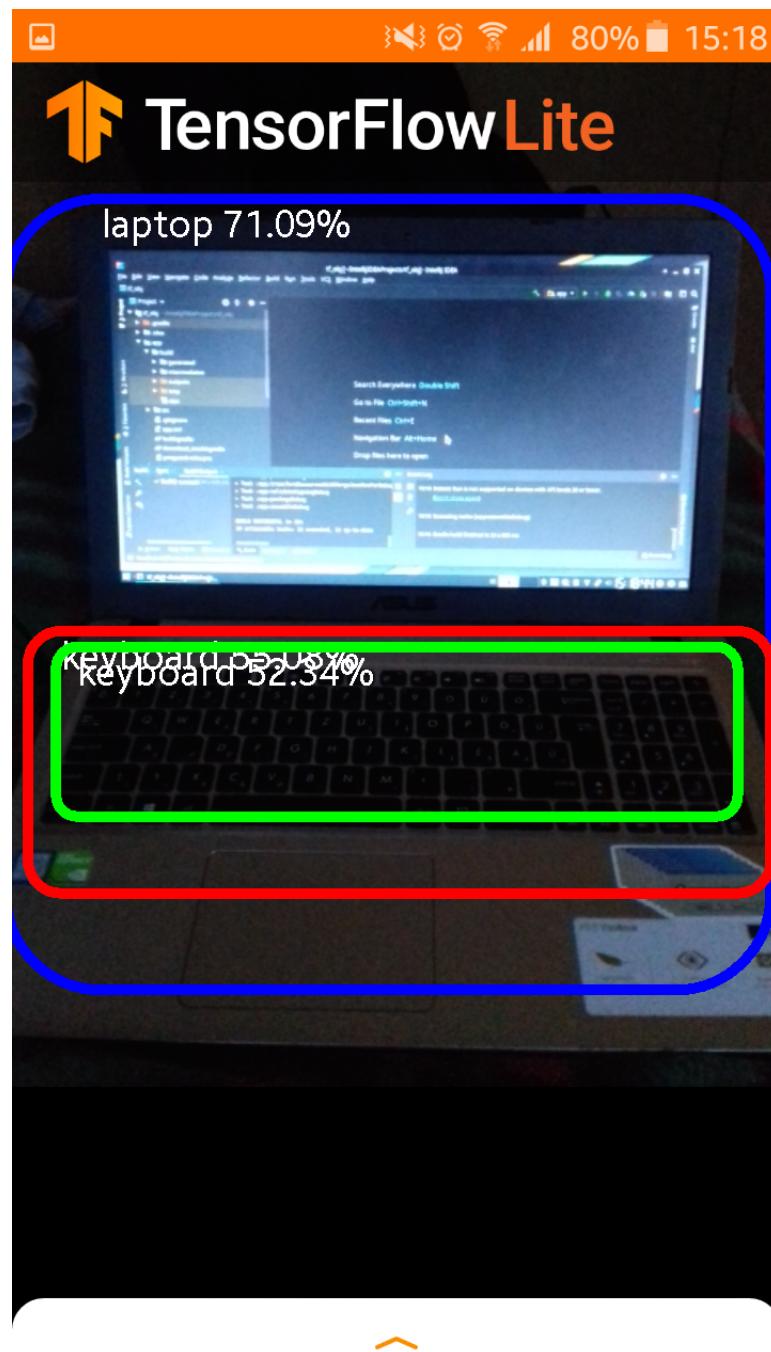
Ha kész a building, akkor futtathatjuk a programot kétféle módon, saját android eszköz használatával, vagy egy virtuális android eszköz(emulátor) segítségével. Én saját telefon-t használtam, mivel a virtuális

emuláció során nem lenne módom az eszköz kameráját használni(feltéve, ha a laptop kamerát használom helyette), másrészről pedig a virtuális eszköz használata igencsak megerőlteti szegény kis laptopomat.



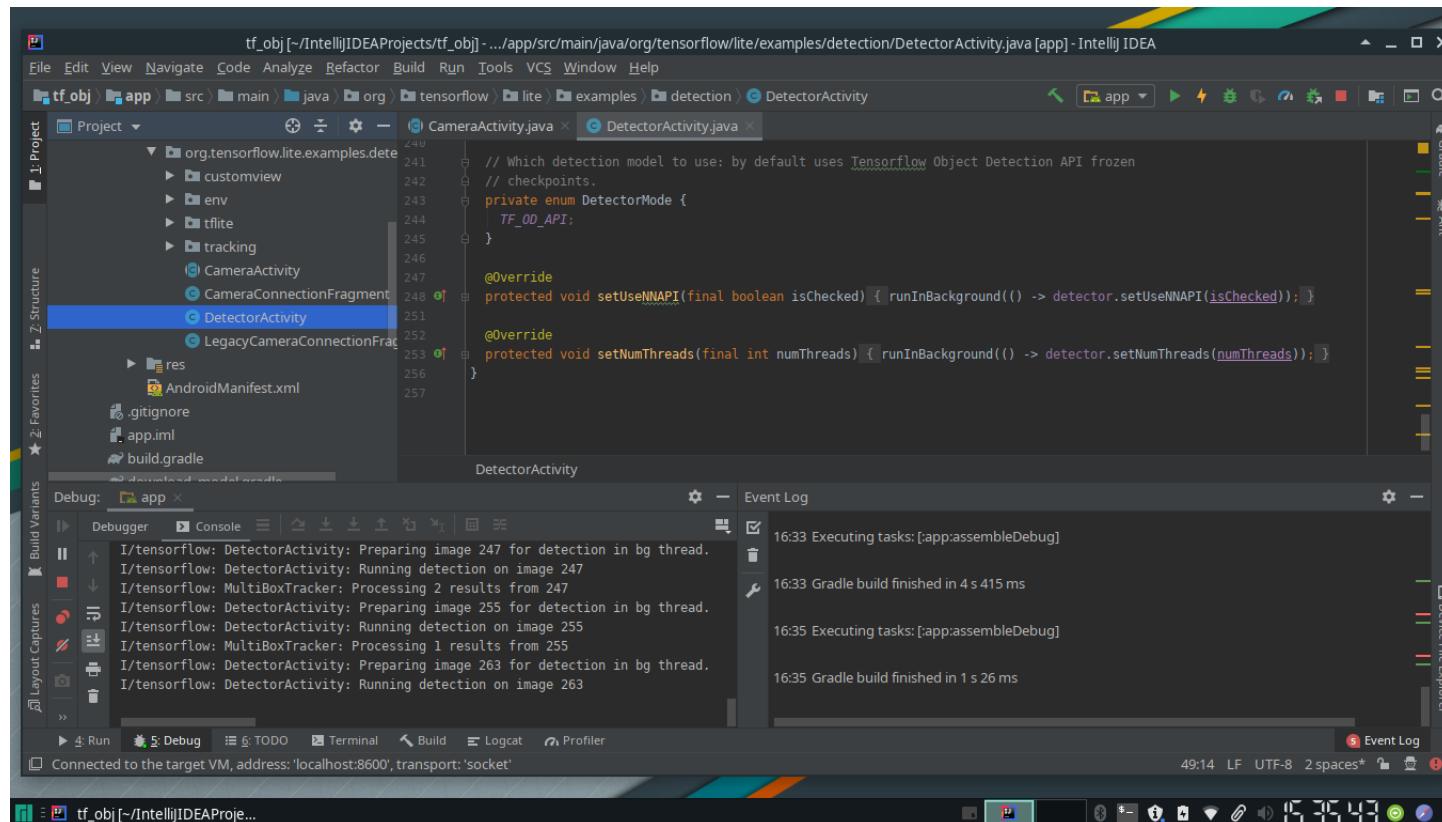
20.4. ábra. A tensor futása

A program a kamerán keresztül érzékelt objektumokról próbálja eldönteni, hogy mi látható a képen. Ez sokszor helyes, de lehetséges macskát felismerni az ásványvizes dobozra, szóval van még mit javítani rajta.

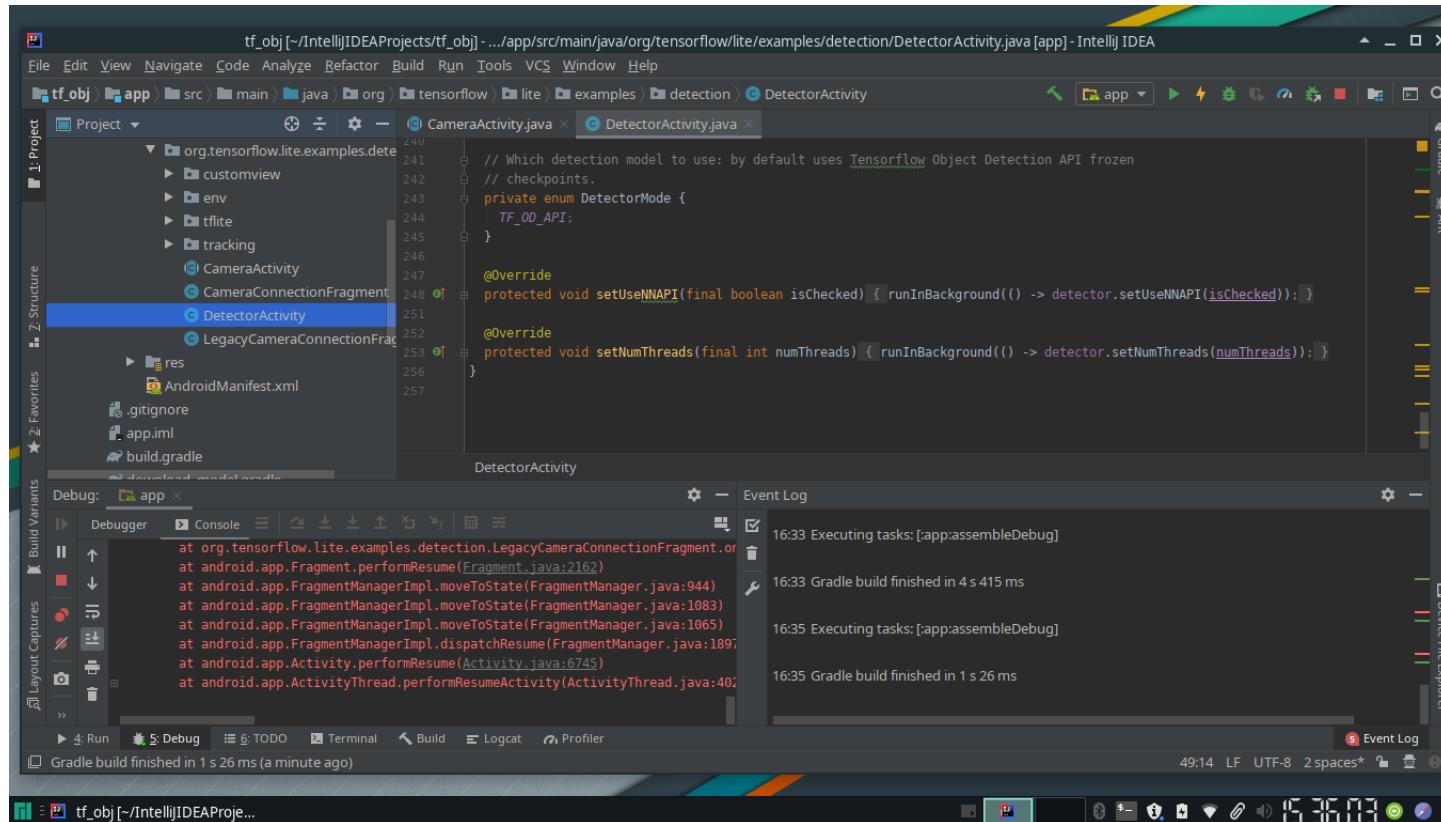


20.5. ábra. A tensor futása2

A program tesztelése során a laptop-ot és az ásványvizes palackot ismertettem fel a szoftverrel, majd később egy debuggert is kapcsoltam hozzá, melyről a kép kicsit lentebb található meg.



20.6. ábra. A tensor futása3



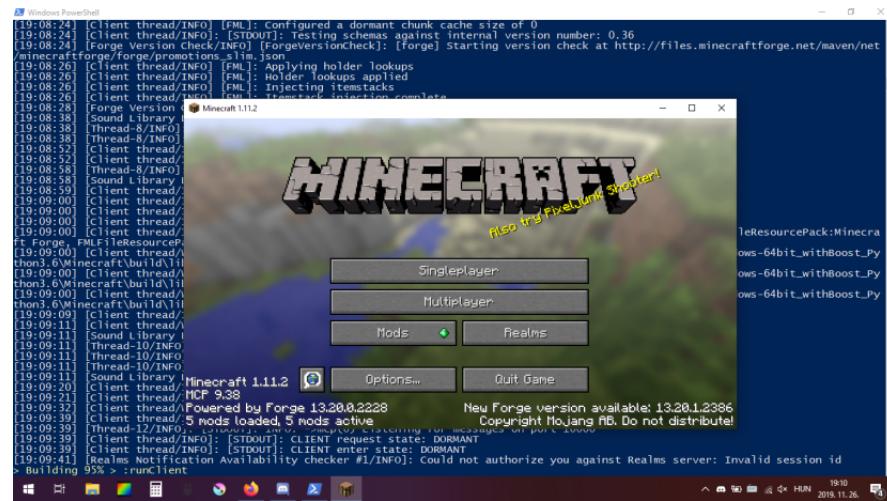
20.7. ábra. A tensor exception

Látható, hogy a debugger hosszas télenség után egy exception-t dob ki (én voltam hosszú ideig tétlen, majd véletlenül kihúztam az usb kábelt :D) A kábel nélkül pedig a debugger nem találta többé a portot, amin kapcsolódhad az android eszközre, tehát jöttek a vérvörös sorok.

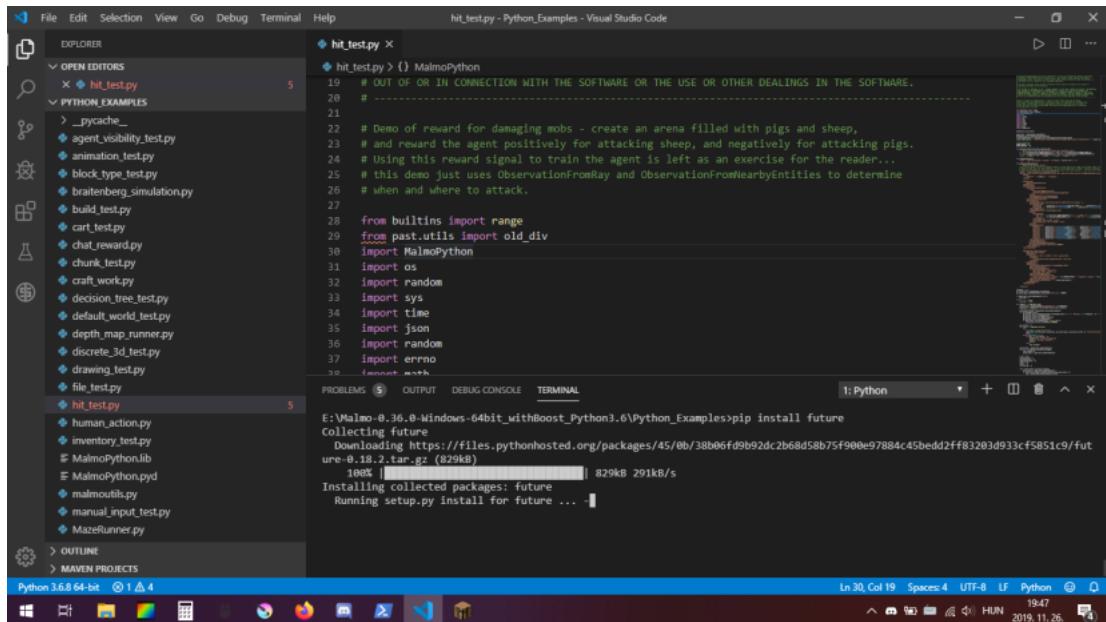
Összességeben szerintem elég érdekes gondolat, hogy felismertessünk dolgokat egy AI-el, élvezetes körbenézni a kamera lencséje mögött, és rádöbbenni, hogy mennyi minden meg lehet tanítani egy kis programnak pár száz sor segítségével.

20.3. Minecraft MALMO-s példa

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lásd pl.: [youtube](#), [bhax_blog_eddig_csal](#), [bhax_szemüveg](#)



20.8. ábra. A malmo kliens futás



20.9. ábra. A malmo future modul

Ezen forráskódot hívtam segítségül a line-of-sight forrásrész megírásakor. Itt a képen látható, hogy van pár példa, mely a future modul nélkül nem fut le, tehát a pip-*el* installálnunk kellett a modult, mielőtt a scripteket futtathattuk volna.

pip install future

```

Windows PowerShell
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6> java -version
openjdk version "1.8.0_232"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_232-b09, mixed mode)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.232-b09, mixed mode)
PS E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6> python -V
python 3.6.8
PS E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6> cd .\Minecraft
PS E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6\Minecraft> ls

Directory: E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6\Minecraft

Mode                LastWriteTime         Length Name
-d----

```

20.10. ábra. A malmo kliens futtatása

A kliens első futtatása nagyon hosszadalmas, 5-10 percet is igénybe vehet, hiszen telepíti a dependency-ket, és frissíti az esetleges outdated szoftvereket.

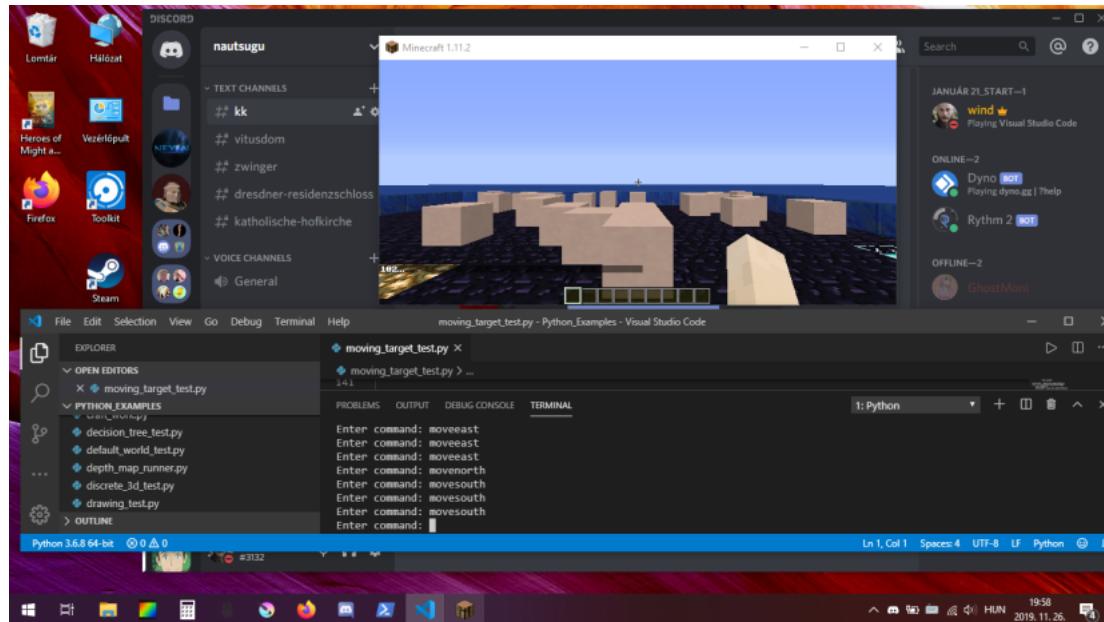
```

Windows PowerShell
PS E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6\Python_Examples> python .\tutorial_5.py
Traceback (most recent call last):
  File ".\tutorial_5.py", line 25, in <module>
    from past.utils import old_div
ModuleNotFoundError: No module named 'past'
PS E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6\Python_Examples> python .\tutorial_6.py
Traceback (most recent call last):
  File ".\tutorial_6.py", line 29, in <module>
    from future import standard_library
ModuleNotFoundError: No module named 'future'
PS E:\Malmo-0.36.0-windows-64bit_withBoost_Python3.6\Python_Examples> python .\tutorial_8.py
Waiting for the mission to start ...

```

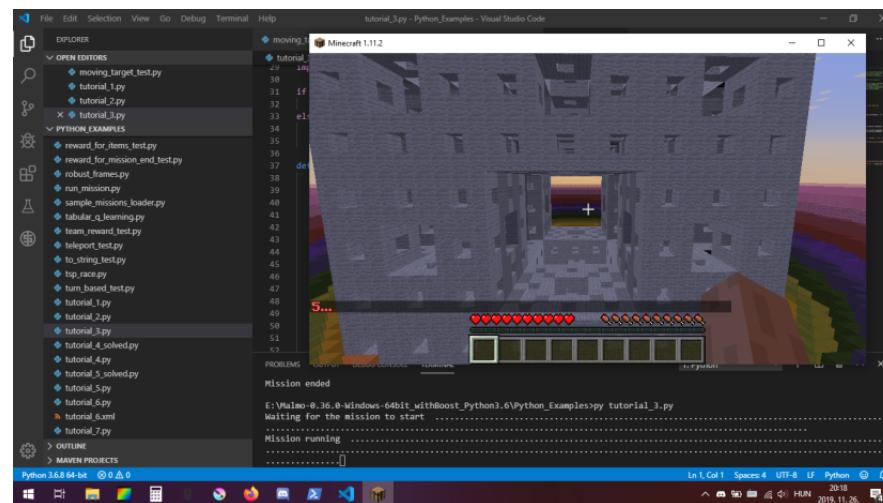
20.11. ábra. A python forráskód futás

Ha elindult a kliens, akkor futtathatjuk a python kódokat, a kliens indulása előtt hiába próbáljuk futtatni őket, hiszen nincs mi értelmezze a kódot.



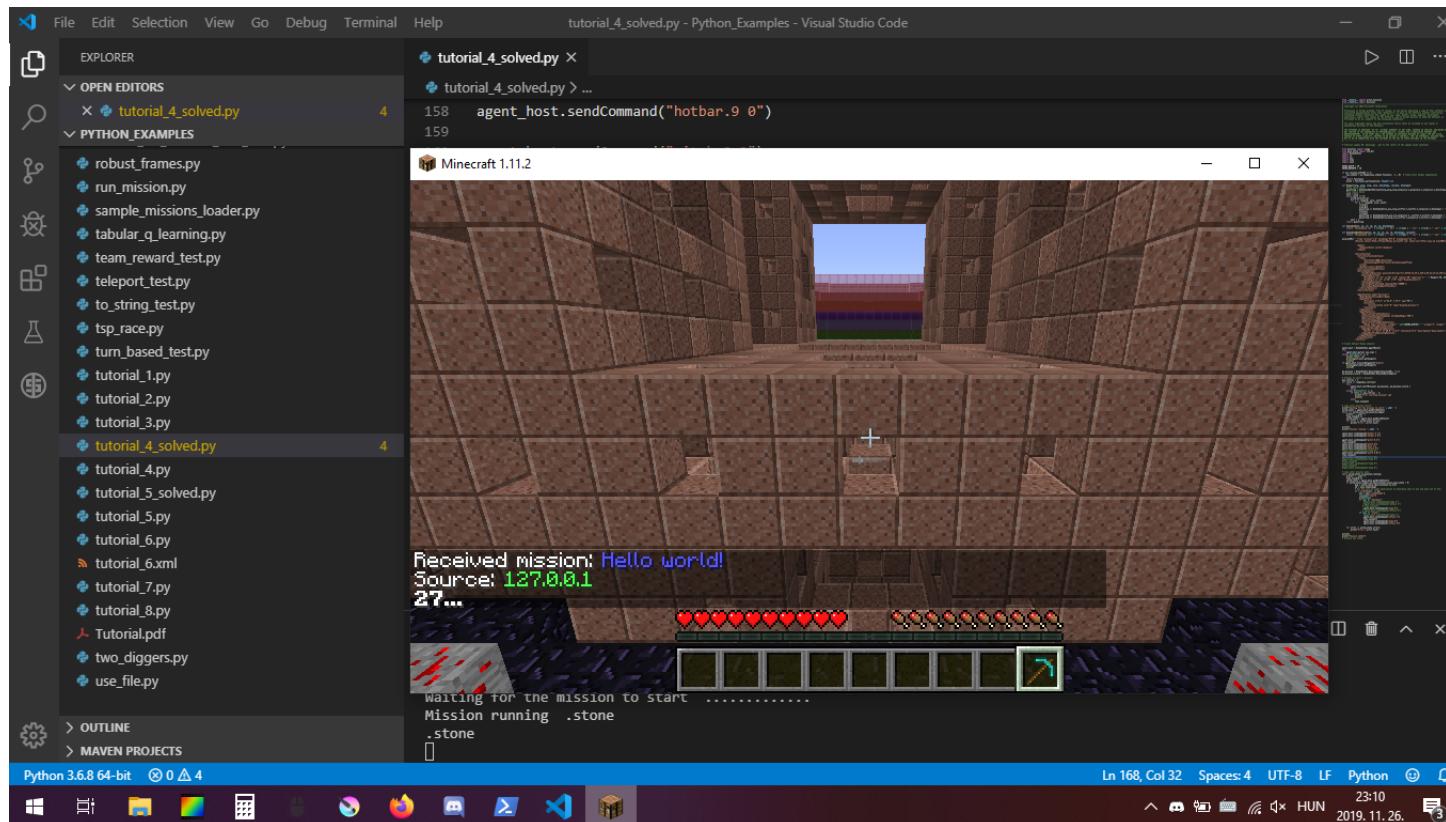
20.12. ábra. egy malmo példa

Ebben a példában a konzolból irányíthattuk a karaktert szöveges módon. Érdekes megoldás, de akkor már használjuk a billentyűzetet.



20.13. ábra. A második malmo példa

Ez a példa pedig egy a képen látható objektumot hozott létre a semmiből, szimpla xml kódokat használva.
[youtube videó a szkript futtatásáról](#)



20.14. ábra. A malmo saját kód alapja

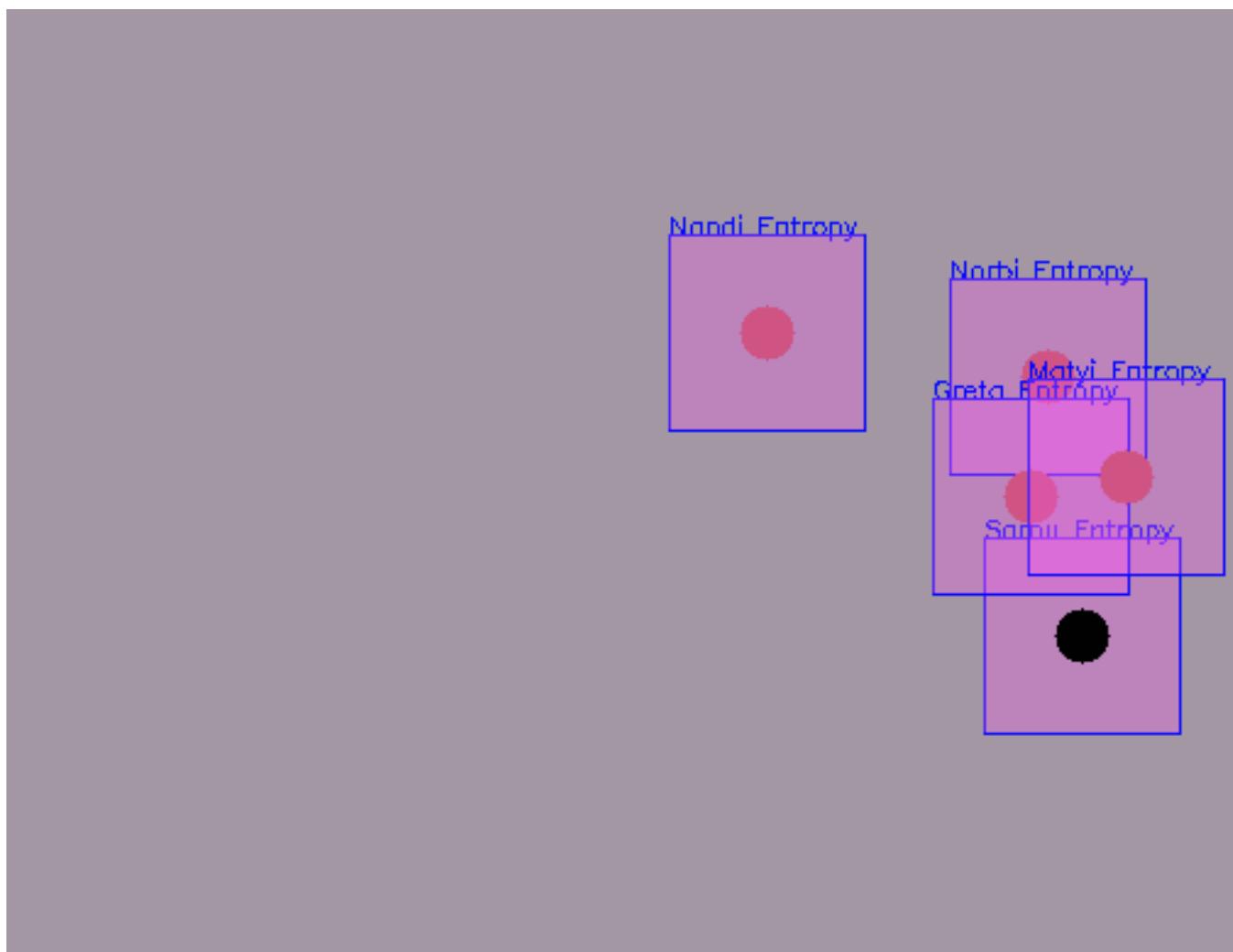
A kód segítségével egy ún. ágenst hívunk meg a minecraft egy helyi szerverére, melynek nevet, inventory-t, video-recording engedélyt és egyéb finomságokat adhatunk egy belső xml használatával. Ez az ágens a játékban levő karakterünket irányítja olyan módon, ahogyan azt mi megírjuk a python szkript-ben.

A forráskód itt található:<https://github.com/ghjbku/prog2/tree/master/calvin/malmo>

21. fejezet

Helló, védés!

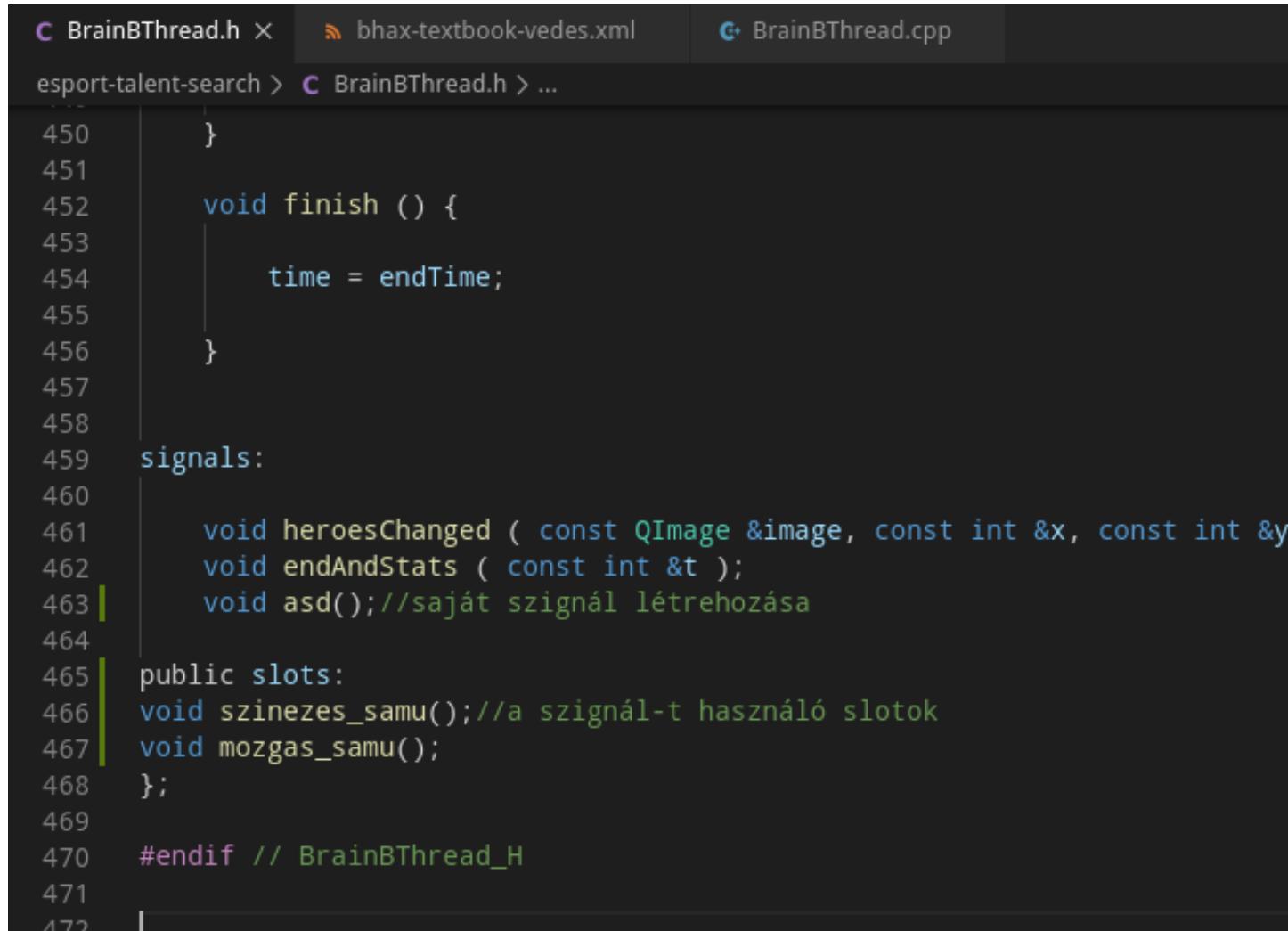
21.1. BrainB védés



21.1. ábra. A Brainb ablak

```
QTimer *timer = new QTimer(this);
connect(timer, &QTimer::timeout, [=] () { this->asd(); }); //az asd(saját) ←
    szignál használata
connect(this, SIGNAL(asd()), this, SLOT(szinezes_samu())); //samu 5s-enként ←
    színt vált
connect(this, SIGNAL(asd()), this, SLOT(mozgas_samu())); //samu 5s-enként ←
    ugrik
    timer->setInterval(5000);
    timer->start();
```

Az 5 másodpercenkénti szín- és pozíció váltáshoz szükségünk volt egy QTimer-re, amely a setInterval tagban beállított időközönként hajtja majd végre a slot-ban megadott függvényt. Ez az intervallum millisec-ben értendő, tehát 5 helyett 5000-et kellett beírni. A BrainThread.h fájlból létrehoztunk egy signált asd() azonosítóval, amelyet connectelünk a QTimer::timeout event-hez. Tehát ez a signal akkor lesz "eltüzelve", amikor ez az event megtörténik. A szignált aztán hozzákötjük a két létrehozott slotunk-hoz, a szinezes_samu és a mozgas_samu-hoz. Ezek definiálása itt található:



The screenshot shows a code editor window with the following details:

- File tabs: BrainBThread.h (active), bhax-textbook-vedes.xml, BrainBThread.cpp.
- Project navigation: esport-talent-search > BrainBThread.h > ...
- Code content:

```
450     }
451
452     void finish () {
453         time = endTime;
454     }
455
456
457
458
459 signals:
460
461     void heroesChanged ( const QImage &image, const int &x, const int &y );
462     void endAndStats ( const int &t );
463     void asd(); //saját szignál létrehozása
464
465 public slots:
466     void szinezes_samu(); //a szignál-t használó slotok
467     void mozgas_samu();
468 };
469
470 #endif // BrainBThread_H
471
472
```

21.2. ábra. A Brainb slotok

Valamint a függvények:

```
void BrainBThread::szinezes_samu()
{
    for ( Hero & hero : heroes ) {
        if(hero.samu==true) {
            double r=std::rand()%255;
            double g=std::rand()%255;
            double b=std::rand()%255;
            cv::Scalar ujc { r, g, b };
            cCentersam=ujc;
        }
    }
}
```

Ez a függvény az összes Hero típusú hero objektumon keresztül-pásztáz, és ha egy olyan elemet talál, amelyben a samu változó értéke igaz, akkor a cCentersam skalár-hoz egy új érték hármast rendel. Ez a változó az azonosítás megkönnyítése érdekében lett hozzáadva a Hero típus definíálásánál a változók közé. Ehelyett a hero objektum nevével is azonosíthattuk volna az egyedet. Snippet a Hero típus-ról:

```
class Hero
{
public:
    int x;
    int y;
    int color;
    int agility;
    int cond {0};
    int uj=0;
    bool samu=false;
    std::string name;

    .
    .
    .
```

A samu változón kívül egy másik változó is bekerült a már meglévők mellé, az "uj", amely az új entitások azonosítására szolgál ilyen módon:

```
Hero other ( heroes[0].x + rx*std::rand() / ( RAND_MAX+1.0 )-rx/2,
              heroes[0].y + ry*std::rand() / ( RAND_MAX+1.0 )-ry/2,
              255.0*std::rand() / ( RAND_MAX+1.0 ), 11, "New Entropy ←
              " );
other.uj=1;
```

Az új entitások uj változója az 1-es értéket hordozza, a többiek viszont a 0-ás értéket. Az azonosítás pedig itt látható:

```
if(hero.uj>0) {
    cv::circle ( src, xc, 11, cCenteruj, cv::FILLED, ←
                 8, 0 );
}
```

Itt beállítjuk, hogy ha ez az entitás egy "new Entropy" lesz, akkor a színét ne a cCenter skalár hordozza, hanem a cCenteruj.

A samu azonosítására pedig itt található a második módszer:

```
if(hero.name=="Samu Entropy")
{
    cv::circle ( src, xc, 11, cCentersam, cv::FILLED, 8, 0 );
}
```

Samu mozgatásáért ezen függvények felelősek:

```
void BrainBThread::mozgas_samu()
{
    samu_move_random();
}

.

.

.

void samu_move_random() {

    heroes[0].move_random ( w, h, 300);

}

.

.

.

void move_random ( int maxx, int maxy, int env )
{
    std::srand ( std::time ( 0 ) );
    int v1 = rand() % 4;

    //up
    if(v1==0) {
        env=env/2;
        int newy = y - env;
        if ( newy-env > 0 && newy+env < maxy ) {
            y = newy;
        }
    }

    //down
    if(v1==1) {
        env=env/2;
        int newy = y + env;
        if ( newy-env > 0 && newy+env < maxy ) {
```

```
        y = newy;
    }
}

//left
if(v1==2) {

int newx = x - env;
    if( newx-env > 0 && newx+env < maxx ) {
        x = newx;
    }
}
//right
if(v1==3) {

int newx = x + env;
    if( newx-env > 0 && newx+env < maxx ) {
        x = newx;
    }
}
}
```

Ugyebár samu a legelső hero, tehát a heroes[0]-val hivatkozunk rá.

A move_random függvény 4 féle mozgást képes használni, a fel,le,balra és jobbra irányokat, amelyek között a v1 változó értéke segítségével változatunk. Ugyebár az y tengely a fel és le irányokért felelős, míg az x tengely a bal és jobb irányokért.

következő függvény a szinezés(), amely a háttér és az entitások dobozának színét változtatja meg.

```
void BrainBThread::szinezes()
{

    double r=rand()%255;
    double g=rand()%255;
    double b=rand()%255;
    cv::Scalar ujc { r, g, b };
    std::srand (1);
    double r2=rand()%255;
    double g2=rand()%255;
    double b2=rand()%255;
    cv::Scalar ujc2 { r2, g2, b2 };
    cBoxes=ujc;
    cBg=ujc2;//random háttér

}
```

Az srand seed-et azért kellett megváltoztatnunk, mert anélkül minden a háttérszín minden doboz színe ugyanazon random értéket kapná, hiába a másik változó. A háttér színét a cBg skalár tartalmazza.

A szinezés2 függvény felelős a randomizált kezdő-entitások színezéséért és az új entitások színéért.

```
void BrainBThread::szinezes2()
{
    double r=std::rand()%255;
    double g=std::rand()%255;
    double b=std::rand()%255;
    cv::Scalar ujc { r, g, b };
    std::srand (2);
    double r2=std::rand()%255;
    double g2=std::rand()%255;
    double b2=std::rand()%255;
    cv::Scalar ujc2 { r2, g2, b2 };
    cCenter=ujc;
    cCenteruj=ujc2; //random új entitás színe
}
```

az előzőhöz hasonlóan itt is azonos lett volna a színezés, ha ugyanazon seed-et használtuk volna a skalárok értékeinek beállításához.

A következő funkció az entitások mozgatása, samu kivételével. Ennek a kódja itt található:

```
void minden_move_nemsamu_random() {
    for (size_t i = 1; i < heroes.size(); i++)
    {
        heroes[i].move_random ( w, h, 300 );
    }
}
```

Mivel samu a 0-dik entitás, ezért a ciklusunkat az első entitásnál kezdjük, tehát Samun kívül mindenki más ugrani fog.

A billetyűlenyomásra vonatkozó függvények a BrainBWin.cpp állományban találhatóak:

```
esport-talent-search > C++ BrainBWin.cpp > BrainBWin::keyPressEvent(QKeyEvent *)  
187  
188  
189     if ( event->key() == Qt::Key_S ) {  
190         save ( brainBThread->getT() );  
191     } //saját  
192  
193     else if ( event->key() == Qt::Key_P ) {  
194         brainBThread->pause();  
195     } else if ( event->key() == Qt::Key_Q || event->  
196                 close();  
197     }  
198     else if ( event->key() == Qt::Key_E ) {  
199         brainBThread->minden_move_nemsamu_random();  
200     }  
201     else if ( event->key() == Qt::Key_Space ) {  
202         brainBThread->samu_move_random();  
203         brainBThread->szinezes_samu2();  
204         brainBThread->asd();  
205     }  
206 }
```

21.3. ábra. A Brainb bill.funkciók

A forráskódok megtalálhatóak itt: <https://github.com/ghjbku/prog2/tree/master/esport-talent-search/>

IV. rész

Irodalomjegyzék

21.2. Általános

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

21.3. C

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

21.4. C++

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

21.5. Lisp

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

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

Ezen túl kiemelt köszönet illeti az említett UDPORG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségen született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.