

# Univerzális programozás

---

## A kódolás dekódolása kezdőknek

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

**COLLABORATORS**

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Schachinger, Zsolt	2019. december 12.	

**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 <a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-02-23	Turing feladatok megoldásának megkezdése.	salesz9902
0.0.6	2019-02-27	A Turing nagyjából készen van. Ismerkedés a Chomsky-val.	salesz9902
0.0.7	2019-03-02	Turing befejezése. Chomsky feladatainak elkezdése.	salesz9902

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME
0.0.8	2019-03-03	Chomsky nagyjából készen van.	salesz9902
0.0.9	2019-03-04	Ismerkedés a Caesar feladatsorral.	salesz9902
0.1.0	2019-03-08	Caesarban lévő programok kipróbálása, elmélkedés. Enyhén elkezdve.	salesz9902
0.1.1	2019-03-08	Caesarban lévő programok kipróbálása, elmélkedés. Enyhén elkezdve.	salesz9902
0.1.2	2019-03-10	Chomsky teljesen befejezve. Caesar nagyjából készen van.	salesz9902
0.1.3	2019-03-11	Ismerkedés a Mandelbrot fejezzel.	salesz9902
0.1.4	2019-03-16	Belekezdés a Mandelbrotba.	salesz9902
0.1.5	2019-03-17	Mandelbrot nagyjából készen van. Bár még hiányos.	salesz9902
0.1.6	2019-03-18	Ismerkedés a Welch fejezzel.	salesz9902
0.1.7	2019-03-24	Teljes gőzerővel neki a Welchnek. Nagyjából készen van.	salesz9902
0.1.8	2019-03-25	Ismerkedés a Conway fejezzel.	salesz9902
0.1.9	2019-03-30	Conway fejezet megkezdése. Nagyjából kész, bár hiányos.	salesz9902

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME
0.2.0	2019-04-01	Ismerkedés a Schwarzenegger fejezettel.	salesz9902
0.2.1	2019-04-05	Mandelbrot teljesen készén van. Conway elmélkedés, utánaolvasás.	salesz9902
0.2.2	2019-04-08	Ismerkedés a Chaitin fejezettel.	salesz9902
0.2.3	2019-04-20	Csiszolgatás a feladatokon, olvasónaplók írása folyamatban.	salesz9902
0.2.4	2019-04-28	Utolsó finomítgatások, csiszolgatások a feladatokon visszamenőleg.	salesz9902
0.2.5	2019-04-29	Sikertelen védés. Binfa újratanulmányozása, feladatok átnézése.	salesz9902
0.2.6	2019-05-01	Git kezelésének megtanulása linuxon. Innentől kezdve ilyen módon megy a commitolás, pusholás.	salesz9902
0.2.7	2019-05-06	Sikeres védés a Hibavisszaterjesztéses perceptron feladatból.	salesz9902
0.2.8	2019-05-07	További csiszolások a feladatokon. Képek beillesztése pár feladathoz.	salesz9902
0.2.9	2019-05-09	A könyv végleges befejezése, leadható állapotba hozása.	salesz9902

# 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

<b>I. Bevezetés</b>	<b>1</b>
<b>1. Vízió</b>	<b>2</b>
1.1. Mi a programozás? . . . . .	2
1.2. Milyen doksikat olvassak el? . . . . .	2
1.3. Milyen filmeket nézzek meg? . . . . .	2
1.4. Pár információ a könyv írásával kapcsolatban . . . . .	2
<b>II. Tematikus feladatok</b>	<b>4</b>
<b>2. Helló, Turing!</b>	<b>6</b>
2.1. Végtelen ciklus . . . . .	6
2.2. Lefagyott, nem fagyott, akkor most mi van? . . . . .	7
2.3. Változók értékének felcserélése . . . . .	9
2.4. Labdapattogás . . . . .	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . . .	11
2.6. Helló, Google! . . . . .	13
2.7. 100 éves a Brun téTEL . . . . .	15
2.8. A Monty Hall probléma . . . . .	16
<b>3. Helló, Chomsky!</b>	<b>19</b>
3.1. Decimálisból unárisba átváltó Turing gép . . . . .	19
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen . . . . .	19
3.3. Hivatalos nyelv . . . . .	21
3.4. Saját lexikális elemző . . . . .	21
3.5. l33t.l . . . . .	22

---

3.6. A források olvasása . . . . .	24
3.7. Logikus . . . . .	26
3.8. Deklaráció . . . . .	26
<b>4. Helló, Caesar!</b>	<b>29</b>
4.1. double ** háromszögmátrix . . . . .	29
4.2. C EXOR titkosító . . . . .	29
4.3. Java EXOR titkosító . . . . .	31
4.4. C EXOR törő . . . . .	32
4.5. Neurális OR, AND és EXOR kapu . . . . .	35
4.6. Hiba-visszaterjesztéses perceptron . . . . .	35
<b>5. Helló, Mandelbrot!</b>	<b>37</b>
5.1. A Mandelbrot halmaz . . . . .	37
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal . . . . .	38
5.3. Biomorfok . . . . .	39
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . . .	40
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . . .	44
5.6. Mandelbrot nagyító és utazó Java nyelven . . . . .	45
<b>6. Helló, Welch!</b>	<b>46</b>
6.1. Első osztályom . . . . .	46
6.2. LZW . . . . .	47
6.3. Fabejárás . . . . .	47
6.4. Tag a gyökér . . . . .	47
6.5. Mutató a gyökér . . . . .	47
6.6. Mozgató szemantika . . . . .	48
<b>7. Helló, Conway!</b>	<b>49</b>
7.1. Hangyaszimulációk . . . . .	49
7.2. Java életjáték . . . . .	49
7.3. Qt C++ életjáték . . . . .	59
7.4. BrainB Benchmark . . . . .	59
<b>8. Helló, Schwarzenegger!</b>	<b>60</b>
8.1. Szoftmax Py MNIST . . . . .	60
8.2. Mély MNIST . . . . .	60
8.3. Minecraft-MALMÖ . . . . .	60

---

---

<b>9. Helló, Chaitin!</b>	<b>61</b>
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .	61
9.2. Gimp Scheme Script-fu: króm effekt . . . . .	61
9.3. Gimp Scheme Script-fu: név mandala . . . . .	61
<b>10. Helló, Gutenberg!</b>	<b>62</b>
10.1. Juhász István - Magas szintű programozási nyelvek 1 . . . . .	62
10.2. Kernighan Ritchie - A C programozási nyelv . . . . .	63
10.3. Programozás . . . . .	64
<b>III. Második felvonás</b>	<b>66</b>
<b>11. Helló, Berners-Lee!</b>	<b>68</b>
11.1. Olvasónapló: C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I--II.II. . . . .	68
11.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal) . . . . .	69
<b>12. Helló, Arroway!</b>	<b>70</b>
12.1. OO szemlélet . . . . .	70
12.2. Homokozó, Binfa program elkészítése Java-ban . . . . .	71
12.3. "Gagyi" . . . . .	73
12.4. Yoda . . . . .	74
12.5. Kódolás from scratch . . . . .	75
<b>13. Helló, Liskov!</b>	<b>78</b>
13.1. Liskov helyettesítés sértése . . . . .	78
13.2. Szülő-gyerek . . . . .	79
13.3. Anti OO . . . . .	80
13.4. Hello, Android! . . . . .	81
13.5. Ciklomatikus komplexitás . . . . .	82
<b>14. Helló, Mandelbrot!</b>	<b>84</b>
14.1. Reverse engineering UML osztálydiagram . . . . .	84
14.2. Forward engineering UML osztálydiagram . . . . .	85
14.3. Egy esettan . . . . .	86
14.4. BPMN . . . . .	87

---

<b>15. Helló, Chomsky!</b>	<b>89</b>
15.1. Encoding . . . . .	89
15.2. OOCWC lexer . . . . .	90
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció . . . . .	94
15.4. Perceptron osztály . . . . .	94
<b>16. Helló, Stroustrup!</b>	<b>97</b>
16.1. JDK osztályok . . . . .	97
16.2. Másoló-mozgató szemantika . . . . .	98
16.3. Hibásan implementált RSA törése . . . . .	99
16.4. Változó argumentumszámú ctor . . . . .	100
16.5. Egyszerű string osztály . . . . .	101
<b>17. Helló, Gödel!</b>	<b>103</b>
17.1. Gengszterek . . . . .	103
17.2. C++11 Custom Allocator . . . . .	104
17.3. STL map érték szerinti rendezése . . . . .	105
17.4. GIMP Scheme hack . . . . .	106
<b>18. Helló, unknown!</b>	<b>108</b>
18.1. OOCWC Boost ASIO hálózatkezelése . . . . .	108
18.2. SamuCam . . . . .	109
18.3. BrainB . . . . .	110
<b>19. Helló, Lauda!</b>	<b>113</b>
19.1. Port scan . . . . .	113
19.2. AOP . . . . .	114
19.3. Junit teszt . . . . .	115
<b>20. Helló, Calvin!</b>	<b>117</b>
20.1. MNIST . . . . .	117
20.2. Android telefonra a TF objektum detektálója . . . . .	121
20.3. CIFAR-10 . . . . .	122

---

<b>IV. Irodalomjegyzék</b>	<b>125</b>
20.4. Általános . . . . .	126
20.5. C . . . . .	126
20.6. C++ . . . . .	126
20.7. Lisp . . . . .	126

# 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 allokálni 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 más is) példával.

## Hogyan nyomjuk?

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

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

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

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



#### A DocBook XML 5.1 új neked?

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

## I. rész

### Bevezetés

# 1. fejezet

## Vízió

### 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.

### 1.4. Pár információ a könyv írásával kapcsolatban

A könyvet egy Dell Latitude E5570 notebookon írtam már az elejtől kezdve. Oracle VM VirtualBox program segítségével felsetupoltam egy virtuális gépet, mégpedig pontosabban Ubuntu Linux 18.04 disztró, verzió alatt vittem végig. Választásom azért az Ubuntu Linuxra esett, mert eddig ebben a linuxban volt elég tapasztalom ahhoz, hogy gördülékenyen menjenek a dolgok.

A programokat Visual Studio Code-ban írtam, terminálban fordítottam gcc-vel, illetve az Apache Netbeans 11.0-t használtam a könyv szerkesztésére. Az XML olyan szempontból új volt számomra, hogy nem írtam még benne kódöt, mindezek ellenére egész hamar bele lehet tanulni a dolgokba, könnyen tanulható, egyszerű nyelv.

Utólag sajnálom, hogy későn tanultam meg használni a git-et linuxon, hiszen sokkal átláthatóbban lehet vezetni a repositorykat, mintsem grafikus módon összekattingatjuk a böngészőben.

Azt gondolom, hogy eme könyv írása megtanít minket többek között arra, hogy hogyan dokumentáljuk kódjainkat, hogyan kezeljük azokat, illetve erős alapot ad a jövőre nézve ezekkel kapcsolatban.

## **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 forrása:

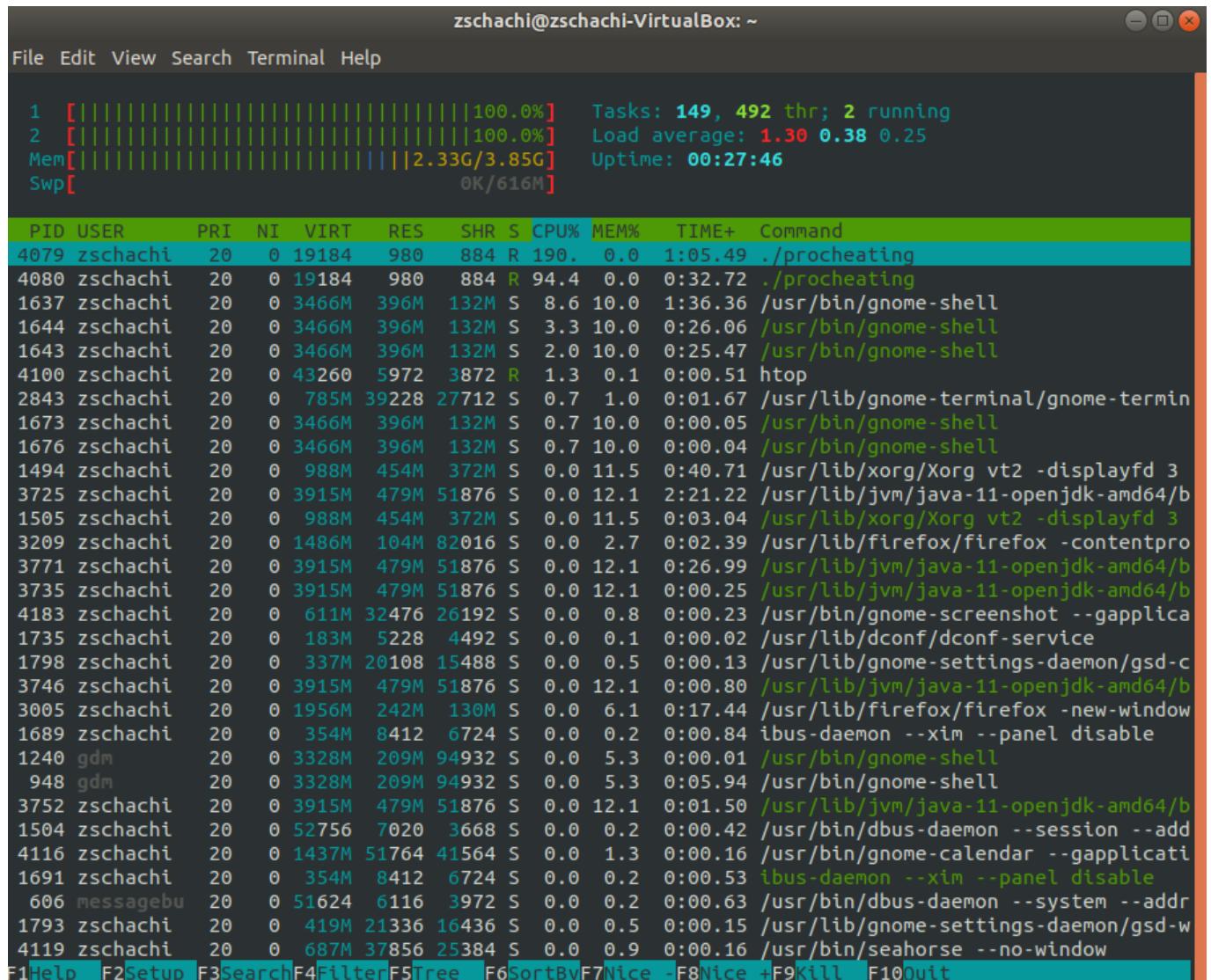
```
Program procheating
{
    #include <stdio.h>
    #include <omp.h>

    int main()
    {
        #pragma omp parallel for
        for (int i=0; i<10; i++)
        {
            i--;
        }

    }
}
```

Az OpenMP (Open Multi-Processing) egy api, ami támogatja a multi többprocesszoros programozást. Ilyen esetben, ezt sokkal egyszerűbb használni, mintsem elkezdenénk a több-szálkezelő (multithread) módszerrel dolgozni..

Azt, hogy a program a processzor összes magját kihasználja, OpenMP segítségével oldottam meg. Ez nagyobb programoknál alapszabály, hogy gondoskodjunk a processzor teljes kihasználtságáról.



Ubuntu linux screenshot

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

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

```
        return false;
    }

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Ezt a programot matematikailag lehetetlen megírni számunkra. Ugyanis ilyen program, mint a feladat közben is olvashatjuk nem hozható létre. Ha elkezdjük boncolgatni a problémát, újabb problémába ütközünk, hiszen hamar ellentmondást kapunk a dolgok vizsgálata kapcsán... Elvégre többször is bizonyítva volt már sok nagyobb ember által is, hogy a program nem megírható.

## 2.3. Változók értékének felcserélése

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

Megoldás videó: [https://bhaxor.blog.hu/2018/08/28/10\\_begin\\_goto\\_20\\_avagy\\_elindulunk](https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk)

Megoldás forrása:

```
Program procheating
{
    #include <stdio.h>

    int main()
    {
        //change the values with an extra variable
        int a, b, c;
        a=2;
        b=5;
        c=0;

        //before the trade
        printf("csere elott:\n a=%d, b=%d\n", a, b);
        c=a;
        a=b;
        b=c;
        //after the trade
        printf("csere utan:\n a=%d, b=%d\n", a, b);

        //no extra variable used..
        //change values with exor
        a=3;
        b=8;

        //values before the trade
        printf("\ncsere exorral:\n");
    }
}
```

```
printf("csere elott:\n a=%d, b=%d\n", a, b);

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

printf("csere után:\n a=%d, b=%d\n", a, b);

}
```

Mint fent látható, először segédváltozóval oldom meg a cserét, aztán segédváltozó használata nélkül, művelettel(összefoglalás).

Jelen esetben a műveletekkel való csere előtt, az a értéke 3, a b értéke 8. Hogyan is zajlik pontosan itt a csere?

Így:

```
a = 3
b = 8
a = a + b -> a = 11 (3+8)
b = a - b -> b = 3 (11-8)
a = a - b -> a = 8 (11-3)
```

## 2.4. Labdapattogás

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

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/bouncingball.c>

**A programkódok Bátfai Norbert tulajdonában állnak.**

Ahhoz, hogy a programot megfelelően tudjuk fordítani, használnunk kell a `-lncurses` kapcsolót a következő módon:

```
gcc "programneve" -o "futtathatoneve" -lncurses
```

Ahhoz, pedig, hogy tudjuk használni a `-lncurses` kapcsolót, telepítenünk kell a `libncurses5-dev`-et:

```
sudo apt-get install libncurses5-dev
```

Az if nélküli módszer

```
Program bouncingball
{
    #include <stdio.h>
    #include <stdlib.h>
    #include <curses.h>
```

```
#include <unistd.h>

int main(void)
{
    int xj = 0, xk = 0, yj = 0, yk = 0;
    int mx = 80 * 2, my = 24 * 2;

    WINDOW *ablak;
    ablak = initscr();
    noecho ();
    cbreak ();
    nodelay (ablak, true);

    for (;;)
    {
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;

        yj = (yj - 1) % my;
        yk = (yk + 1) % my;

        clear();

        mvprintw(0, 0,
                  " ←
-----
");
        mvprintw(24, 0,
                  " ←
-----
");
        mvprintw(abs ((yj + (my - yk)) / 2),
                  abs ((xj + (mx - xk)) / 2), "X");

        refresh();
        usleep(150000);
    }
    return 0;
}
```

## 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

A programkód Bátfai Norbert tulajdonában áll.

A feladat kidolgozásához Racs Tamás könyvét használtam.

Megoldás forrása:

Program BogoMIPS

```
{  
#include <time.h>  
#include <stdio.h>  
  
void  
delay (unsigned long long int loops)  
{  
    unsigned long long int i;  
    for (i = 0; i < loops; i++);  
}  
  
int  
main (void)  
{  
    unsigned long long int loops_per_sec = 1;  
    unsigned long long int ticks;  
  
    printf ("Calibrating delay loop..");  
    fflush (stdout);  
  
    while ((loops_per_sec <= 1))  
    {  
        ticks = clock ();  
        delay (loops_per_sec);  
        ticks = clock () - ticks;  
  
        printf ("%llu %llu\n", ticks, loops_per_sec);  
  
        if (ticks >= CLOCKS_PER_SEC)  
        {  
            loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;  
  
            printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / ←  
                    500000,  
                    (loops_per_sec / 5000) % 100);  
  
            return 0;  
        }  
    }  
  
    printf ("failed\n");  
    return -1;  
}
```

A BogoMIPS a processzor egy magjának a gyorsaságát méri meg 1 másodperc alatt. Sokan így akarják

összehasonlítani számítógépük erősségeit, ezt nem erre találták ki. Nyilvánvalóan ez nem lesz annyira pontos, bár elég jó megközelíti a valóságot. A programunk kimenetének második oszlopában a 2 hatványai szerepelnek. Az első oszlop pedig azt mutatja meg, hogy egyes lépésközzel mennyi ideig jutott el a 0-tól az adott hatvány számolásáig, azaz hányszor tickelt a processzorunk.

A program futtatása során az alábbi értékeket kapom:

```
zschachi@zschachi-VirtualBox:~/programming/prog1/turing$ ./bogomips
Calibrating delay loop..3 2
1 4
0 8
1 16
1 32
1 64
1 128
1 256
2 512
3 1024
7 2048
13 4096
24 8192
85 16384
97 32768
255 65536
478 131072
921 262144
1704 524288
2947 1048576
6016 2097152
22113 4194304
23350 8388608
49226 16777216
99516 33554432
201043 67108864
409525 134217728
862637 268435456
1531133 536870912
ok - 700.00 BogoMIPS
```

*Ubuntu linux screenshot*

## 2.6. Helló, Google!

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/pagerank.c>

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

```
#include <math.h>

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

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

double
tavolsag (double PR[], double PRv[], int n)
{
    double osszeg = 0.0;
    int i;

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

    return sqrt(osszeg);
}

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 } ;

    int i, j;

    for (;;)
    {

        for (i = 0; i < 4; ++i)
        {
            PR[i] = 0.0;
            for (j = 0; j < 4; ++j)
                PR[i] += (L[i][j] * PRv[j]);
        }
        if (tavolsag (PR, PRv, 4) < 0.00000001)
```

```
        break;

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

}

kiir (PR, 4);

return 0;
}
}
```

### A programkód Bátfai Norbert tulajdonában áll.

A PageRank egy olyan algoritmus, amely linkekhez számokat rendel, majd azokat sorrendbe teszi a hálózatban betöltött szerepük alapján. A Google keresőmotorjának ez az egyik legfontosabb eleme. A PageRank szó egyben a Google bejegyzett védjegye.

Ez alapján egyértelműen látjuk, hogy melyik weboldal mennyire fontos, és segítségével hasznos listát tudunk felállítani az oldalak fontosságáról. Nyílvánvalóan ez nem jelenti azt, hogy ha a Google keresőnk ötödjére jelenít meg valamit, hogy nem annyira validok rajta az információk, mintsem az első helyen szereplőnek. Közel sem... hiszen ez a kattintásokat veszi figyelembe, ami az oldal népszerűségében nyílvánul meg.

A PageRank-ra a következő a képlet:  $\text{PageRank}(i) = (1-d) + d * (\text{PageRank}(j)/L(j))$

## 2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/brun.r>

Bruntetel

```
{  
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  
    rt1plus2 = 1/t1primes+1/t2primes  
    return(sum(rt1plus2))  
}  
}
```

### A programkód Bátfai Norbert tulajdonában áll.

A Brun tételet az ikerprímszámok reciprokaiból képez sorösszegeket, Brun konstans néven ismert véges értékhez konvergál.

A példánkban egy olyan programot írtunk, amely próbálja megközelíteni a Brun konstans értékét. Tehát kiszámolja az ikerprímeket, összegzi a reciprokaikat és részeredményt mutat.

Tisztázzuk az ikerprím fogalmát:

Ikerprímnek két olyan prímszám együttesét nevezzük, amelyek 2-vel térnek el egymástól: például 5 és 7. Mivel a prímszámok (a 2-t kivéve) csak páratlan számok lehetnek, két prímszám között nem lehet kisebb a különbség 2-nél (a (2, 3) pár kivételével). Más megfogalmazás szerint: az ikerprímek két olyan prímszám együttese, amelyek között a prímhézag 2.

## 2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/montyhall.r>

Kép forrása: <https://probabilityandstats.wordpress.com/2017/05/11/monty-hall-problem/>

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

  for (i in 1:kiserletek_szama) {

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

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

    } else{

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

    }

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

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

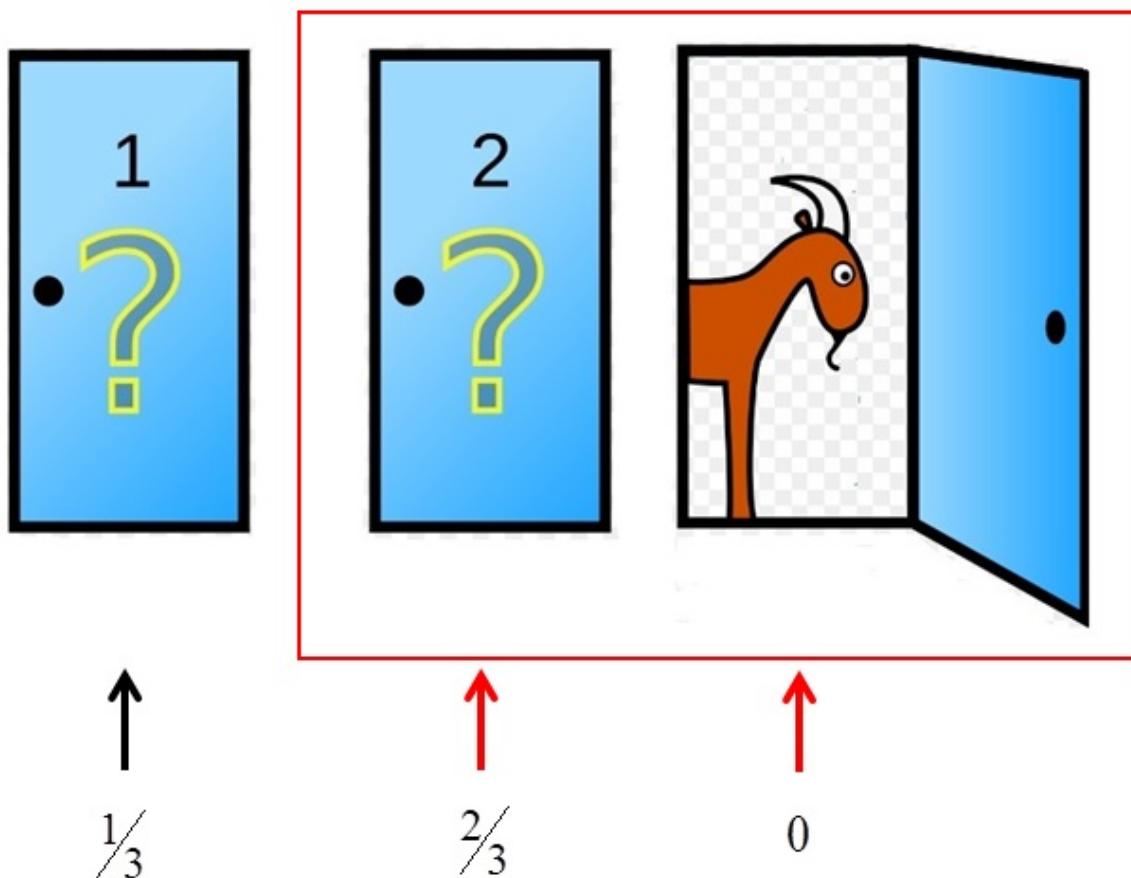
```
for (i in 1:kiserletek_szama) {  
  
    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
    valtoztat[i] = holvalt[sample(1:length(holvalt), 1)]  
  
}  
  
valtoztatesnyer = which(kiserlet==valtoztat)  
  
  
sprintf("Kiserletek szama: %i", kiserletek_szama)  
length(nemvaltoztatesnyer)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

### A programkód Bátfai Norbert tulajdonában áll.

A Monty Hall egy valószínűségi paradoxon. Az Egyesült Államokban, a Let's Make a Deal televíziós vetélkedő egyik feladatán alapul. Nevét a műsorvezetőről, Monty Hall-ról kapta.

A probléma alap kiindulása az, hogy van 3 csukott ajtónk, amelyek közül 2 mögött van valami számunkra értéktelen doleg, viszont az egyik mögött valami rendkívül értékes lapul. Azt kapjuk meg, amelyik az általunk választott ajtó mögött van. Tehát létezik egy egyszerű valószínűségszámítási eszköz, amely megmutatja, hogy melyik ajtót érdemes nekünk választani az adott esetben.

Először tegyük fel, hogy van 1-es 2-es és 3-as ajtónk. A játékosunk először a 3-as ajtót választja. Itt 1/3 eséllyel lesz értékes tárgy. Majd kinyílik a 2-es ajtó. Ez egy értéktelen tárgy lett, ezért ott 0 eséllyel lesz értékes, viszont a mellette lévőben 2/3 az esély.



Az egyik ajtó mögött egy autó, másik kettő mögött kecske található. Az autót keressük. Képen egyértelműen láthatjuk mennyi eséllyel találjuk meg az autót az ajtókban, így, hogy már 1 ajtót kinyitottunk, amiben kecske van.

## 3. fejezet

# Helló, Chomsky!

### 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/chomsky/binarunary.c>

A programkód Molnár Antal Albert tulajdonában van, picit módosítva lett általam.

A Turing-gép Alan Turing angol matematikushoz fűződik, hiszen ő dolgozta ki ennek fogalmát. Ez mindenféle folyamat precízebb megfogalmazására lett kitalálva. Például eljárások, algoritmusok pontosabb leírására.

Írnom kell az unáris számrendserről is. Ez egy nagyon egyszerű számrendszer, amiben vonalakkal ábrázoljuk a számokat. Vegyük példának az 5-öt, ezt öt vonallal ábrázoljuk, a következőképpen:

||||| =5

A programunk annyit csinál, hogy bekér a felhasználótól egy decimális számot, majd azt kiírja unárisban, 5-ösével elválasztva.

```
zschachi@zschachi-VirtualBox:~/programming/prog1/chomsky$ ./binarunary
Kérlek adj meg egy decimális számot: 16
||||| | | | |
zschachi@zschachi-VirtualBox:~/programming/prog1/chomsky$
```

Ubuntu linux screenshot

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

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

Legyenek  $S, X, Y$  változók. Legyen  $a, b, c$  konstansok.

$S \rightarrow abc, S \rightarrow aXbc, Xb \rightarrow bX, Xc \rightarrow Ybcc, bY \rightarrow Yb, Ay \rightarrow aax, Ay \rightarrow aa$

Noam Chomsky szintén egy nyelvész volt, akinek a fenti nyelv grammaticáját is köszönhetjük. Érdekes módon rengeteg kiemelkedő foglalkozása mellett informatikus is volt.

S, X, Y: „változók” (a nemterminálisok)  
a, b, c: „konstansok” (a terminálisok)  
 $S \rightarrow abc$ ,  $S \rightarrow aXbc$ ,  $Xb \rightarrow bX$ ,  $Xc \rightarrow Ybcc$ ,  $bY \rightarrow Yb$ ,  $aY \rightarrow aaX$ ,  $aY \leftrightarrow a$   
 $\rightarrow aa$  (a helyettesítési szabályok)  
S (a mondatszimbólum)

S ( $S \rightarrow aXbc$ )  
aXbc ( $Xb \rightarrow bX$ )  
abXc ( $Xc \rightarrow Ybcc$ )  
abYbcc ( $bY \rightarrow Yb$ )  
aabbcc

S ( $S \rightarrow aXbc$ )  
aXbc ( $Xb \rightarrow bX$ )  
abXc ( $Xc \rightarrow Ybcc$ )  
abYbcc ( $bY \rightarrow Yb$ )  
aYbbcc ( $aY \rightarrow aaX$ )  
aaXbbcc ( $Xb \rightarrow bX$ )  
aabXbcc ( $Xb \rightarrow bX$ )  
aabbXcc ( $Xc \rightarrow Ybcc$ )  
aabbYbccc ( $bY \rightarrow Yb$ )  
aabYbbccc ( $bY \rightarrow Yb$ )  
aaYbbbccc ( $aY \rightarrow aa$ )  
aaabbbccc

A, B, C: „változók” (a nemterminálisok)  
a, b, c: „konstansok” (a terminálisok)  
 $A \rightarrow aAB$ ,  $A \rightarrow aC$ ,  $CB \rightarrow bCc$ ,  $cB \rightarrow Bc$ ,  $C \rightarrow bc$  (a képzési ↔ szabályok)  
S (A kezdőszimbólum)

A ( $A \rightarrow aAB$ )  
aAB ( $A \rightarrow aC$ )  
aaCB ( $CB \rightarrow bCc$ )  
aabCc ( $C \rightarrow bc$ )  
aabbcc

A ( $A \rightarrow aAB$ )  
aAB ( $A \rightarrow aAB$ )  
aaABB ( $A \rightarrow aAB$ )  
aaaABBB ( $A \rightarrow aC$ )  
aaaaCBBB ( $CB \rightarrow bCc$ )  
aaaabCcBB ( $cB \rightarrow Bc$ )  
aaaabCBcB ( $cB \rightarrow Bc$ )  
aaaabCBBC ( $CB \rightarrow bCc$ )  
aaaabbCcBc ( $cB \rightarrow Bc$ )  
aaaabbCBcc ( $CB \rightarrow bCc$ )  
aaaabbbCccc ( $C \rightarrow bc$ )  
aaaabbbbcccc

### 3.3. Hivatkozási nyelv

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

Megoldás forrása:

```
Program hivatkozasinyelv
{
    #include <complex.h>
    #include <stdbool.h>

    int main()
    {
        long long int asd;
        complex stnum;
    }
}
```

A C nyelvnek is vannak régebbi, illetve újabb változatai. Ilyen például a C89, illetve a C99. Összehasonlítva a C89-hez képest rengeteg változás történt a C99-ben.

Például új header fájlok jöttek be a C99-nél, ilyen a `complex.h`, `stdbool.h` vagy a `tgmath.h`.

Jelentős újítás volt még például az új típusok: `long long int`, vagy a `complex` típus.

A fenti kód például C89-ben nem futna le, mivel még nem ismerné a header fájlokat, illetve a `long long int` típust...

### 3.4. Saját lexikális elemző

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

Megoldás forrása:

```
Program lexikalisi
%
#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 programkód Bátfai Norbert tulajdonában áll.**

A programnak megadjuk/definiáljuk a számokat, ezt a [0–9] sorban láthatjuk. Itt azt adjuk meg, hogy bármely szám nullától kilencig, hányszor fordulhat elő. Az ez utáni sorban a . | \n { } után következő utasításnál többöt figyelmen kívül hagyjuk.

### 3.5. l33t.l

Lexelj össze egy l33t cipher-t!

Megoldás forrása: <https://github.com/Salesz9902/prog1/blob/master/l33t.c>

**A programkód Bátfai Norbert tulajdonában áll.**

Ebben a feladatban Tóth Balázs volt a tutorom.

```
{  
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <ctype.h>  
  
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))  
  
struct cipher {  
    char c;  
    char *leet[4];  
} l337d1c7 [] = {  
  
{'a', {"4", "4", "@", "/-\\"}},  
{'b', {"b", "8", "|3", "|"}},  
{'c', {"c", "(", "<", "{"}},  
{'d', {"d", "|)", "[", "|"}},  
{'e', {"3", "3", "3", "3"}},  
{'f', {"f", "|=", "ph", "|#"}},  
{'g', {"g", "6", "[", "[+"}},  
{'h', {"h", "4", "|-", "[ - "}},  
{'i', {"1", "1", "|", "!"}},  
{'j', {"j", "7", "_|", "_/"}},  
{'k', {"k", "|<", "1<", "|{"}},  
{'l', {"l", "1", "|", "|_"}},  
{'m', {"m", "44", "(V)", "|\\|/|"}},  
{'n', {"n", "|\\|", "/\\/", "/V"}},
```

```
{'o', {"0", "0", "()","[]"}},  
'p', {"p", "/o", "|D", "|o"}},  
'q', {"q", "9", "O_","(,)"}},  
'r', {"r", "12", "12", "|2"}},  
's', {"s", "5", "$", "$"}},  
't', {"t", "7", "7", "'+'"}},  
'u', {"u", "|_|", "(_)","[_]"}},  
'v', {"v", "\\", "\\", "\\"}},  
'w', {"w", "VV", "\\\\"}, "(/\\\")"}},  
'x', {"x", "%", ")(","()"}},  
'y', {"y", "", "", ""}}},  
'z', {"z", "2", "7_",">_"}}},  
  
'0', {"D", "0", "D", "0"}},  
'1', {"I", "I", "L", "L"}},  
'2', {"Z", "Z", "Z", "e"}},  
'3', {"E", "E", "E", "E"}},  
'4', {"h", "h", "A", "A"}},  
'5', {"S", "S", "S", "S"}},  
'6', {"b", "b", "G", "G"}},  
'7', {"T", "T", "j", "j"}},  
'8', {"X", "X", "X", "X"}},  
'9', {"g", "g", "j", "j"}}  
  
// https://simple.wikipedia.org/wiki/Leet  
};  
  
%}  
%%  
. {  
  
    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;
}
```

A leet nyelv egy internetes nyelv. Bizonyos betű karaktereket számokkal helyettesítünk, amik erősen hasonlítanak a betűkhöz.

```
[

Például:
3 = E
4 = A
1 = l
7 = T
```

A fentiek ismeretében rájöhetünk, hogy a leet szó => 1337 leet nyelven írva. De akár írhatjuk így is: l33t Programunk annyit csinál, hogy beolvas a terminálról karaktereket, aztán ugyanazt visszaadja leet nyelven, feltéve, hogy definiálva van.

## 3.6. A források olvasása

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

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

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



## Bugok

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

i.

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

ii.

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

Itt egy egyszerű for ciklust láthatunk, amely 0-tól 5-ig megy. Megfigyelhető, hogy az i inkrementálása prefix formában van jelen.

iii.

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

Szintén, mint az előző, annyi különbséggel, hogy már itt postfix formában inkrementál az i.

iv.

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

Szintén egy for ciklus, ami egy tömb i-edik elemét teszi egyenlővé az i++-szal.

v.

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

Itt már találhatunk egy és operátort, ami kettő darab pointert növel eggyel-eggyel.

vi.

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

Itt egy printf függvényt láthatunk, amely kiír két változót, ami egy másik függvény visszatérési értéke lesz.

vii.

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

Itt szintén egy függvény visszatérési, illetve egy változó értékét írja ki.

viii.

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

Itt két számot iratunk ki, ugyanazt a változót, viszont először referenciaként hivatkozva rá.

Ebben a programban egy jelkezelővel "játszadozhatunk". Ha a program futása során megnyomjuk a Ctrl+C billentyűkombinációt, aminek meg kellene szakítani a folyamatot, először nem fogja. Aztán majd mégegyszeri lenyomás után már másképp veszi figyelembe az általunk kiküldött jelet a programunk.

## 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](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX)

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, [https://youtu.be/AJSXOQFF\\_wk](https://youtu.be/AJSXOQFF_wk)

A LaTeX egy texen alapuló szövegformázó rendszer, amely dokumentumok, szakdolgozatok, akár tudományos cikkek írására is használnak. Matematikusok gyakran használják.

A következőt másoltam be egy .tex kiterjesztésű fájlba:

```
$ \forall x \exists y Szeret(x, y) $  
$ \exists y \forall x Szeret(x, y) $  
$ \exists x \forall y Szeret(x, y) $  
$ \exists x \forall y \neg Szeret(x, y) $
```

A következőképpen tudjuk a .tex kiterjesztésű fájlunkat .pdf-é varázsolni:

```
pandoc -t latex logic.tex -o logic.pdf
```

A fenti tex-re a következő kimenetet kaptam a pdf-be:

$$\forall x \exists y Szeret(x, y) \quad \exists y \forall x Szeret(x, y) \quad \exists x \forall y Szeret(x, y) \quad \exists x \forall y \neg Szeret(x, y)$$

*Ubuntu linux screenshot*

## 3.8. Deklaráció

**Ebben a feladatban György Dóra tutora voltam.**

Vezessd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciajára

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás forrása: <https://github.com/Salesz9902/prog1/blob/master/deklaracio.c>

Már a legkisebb programokban is találhatunk változó vagy függvénydeklarációt. Ezek kulcsfontosságúak számunkra, hiszen így tudunk tárolni adatokat egyszerűen amiket programunk során felhasználunk. Illetve a függvényekkel saját függvényeket is írhatunk.

Fontos megemlíteni, hogy egy változó deklarálásakor, akár univerzális, akár konkrét típust, de meg kell adnunk, különben hibaüzenetet fogunk kapni fordításkor. Kezdőknél gyakori, hogy a deklarációt összekeverik az értékadással, avagy deklarációinak nevezik az értékadást.

A fenti pontokban több féle deklarációt láthatunk. Az előbb említett értékadás NEM (!) deklaráció. A következőképpen néznek ki:

```
int a; //deklaráció
a = 5; //értékkadás
int b = 10; //deklaráció és értékkadás egyben
```

Ha egy változót deklarálunk, és nem adunk neki értéket, akkor nagy eséllyel valamilyen "memóriaszemetet" kapunk, ugyanis ilyenkor a programunk az adott változóinknak véletlenszerűen foglal le helyet a memóriában, így ez keletkezik belőle. Ebből az következik, hogy olyan változóknak, amit még az értékének megváltoztatása előtt ki szeretnénk iratni, akkor ne feltétlenül nullára számítsunk, hiszen közel sem biztos, hogy az lesz a kezdőértéke. Ezt az alábbi példában láthatjuk:

```
#include <stdio.h>

int main()
{
    int a, b, c, d, e, f, g;
    printf("%d, %d, %d, %d, %d, %d, %d\n", a,b,c,d,e,f,g);
}
```

A program futtatása, kimenete:

```
zschachi@zschachi-VirtualBox:~/programming/practice$ ./memtrash
22014, 995882304, 22014, 565475760, 32767, 0, 0
zschachi@zschachi-VirtualBox:~/programming/practice$ █
```

*Ubuntu linux screenshot*

## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\* háromszögmátrix

Megoldás forrása: [https://github.com/salesz9902/textbook/blob/master/files/caesar/double\\_trimatrix.c](https://github.com/salesz9902/textbook/blob/master/files/caesar/double_trimatrix.c)

A programkód Bátfai Norbert tulajdonában áll.

Elsősorban megadjuk a mátrix sorainak számát. `int nr = 5` Ezután deklarálunk egy valós értékre mutató mutatót.

Érdemes megfigyelnünk a `malloc` függvényt. 5-ször 8 bájtot foglal le, és egy mutatót ad vissza. Ha 0 a méret, akkor a függvény NULL-t vagy egy egyéni mutatót ad vissza. Az egyik for ciklusban `nr` alkalommal (azaz 5) a `malloc` segítségével `tm[i]`-nek lefoglalja a helyet. Az elsőnél 8 bájtot, aztán 16-ot, aztán 32-t és így tovább az 5.-ig.

Futtatásnál a következő kimenetet láthatjuk:

```
./double_trimatrix
0x7ffce6bf4d40
0x55eba3f9a670
0x55eba3f9a6a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000
```

### 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: [https://github.com/salesz9902/textbook/blob/master/files/caesar/exor\\_task/xor.c](https://github.com/salesz9902/textbook/blob/master/files/caesar/exor_task/xor.c)

A programkód Bátfai Norbert tulajdonában áll.

Ebben a feladatban Tóth Balázs tutorált.

```
#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);

    }
}
```

Egy olyan program, amely általunk megadott kulcs, illetve karakterhossz által generál nekünk egy titkos szöveget, amit az előbb említett adatok felhasználásával tudunk feltörni. Sok esetben hasznunkra lehet.

A main függvényünknel érdemes megfigyelni a paraméterként adott argumentumokat, mégpedig `argc` és `**argv`. Az `argc` a futtatásnál terminálon keresztül bekért adatok számáért felel, ha lehet így fogalmazni. A `**argv` pedig eltárolja a beolvasott argumentumokat, egyesével.

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

**A programkód Molnár Antal Albert tuladonában áll.**

```
import java.util.*;

class XorEncode {
    public static void main(String[] args) {
        String kulcs = "";

        if(args.length > 0) {
            kulcs = args[0];
        } else {
            System.out.println("Kulcs nélkül nem ←
titkosítok!");
            System.out.println("Használat: java ←
XorEncode.java [kulcs]");
            System.exit(-1);
        }

        Scanner sc = new Scanner(System.in);
        String str = "";

        while(sc.hasNext()) {
            str = sc.next();
            System.out.println(xor(kulcs, str));
        }
    }

    public static String xor(String kulcs, String s) {
        StringBuilder sb = new StringBuilder();

        for(int i = 0; i < s.length(); i++) {
            sb.append((char)(s.charAt(i) ^ kulcs.←
charAt(i % kulcs.length())));
        }

        return sb.toString();
    }
}
```

Ebben a feladatban ugyanazt írjuk meg, mint az előzőben, csak nem C-ben, hanem egy erősen objektum orientált nyelvben, a Java-ban.

A program ugyan úgy működik, mint C elődje, szintén a bekért szöveget titkosítja.

## 4.4. C EXOR törő

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

Megoldás forrása: [https://github.com/salesz9902/textbook/blob/master/files/caesar/exor\\_task/exor.c](https://github.com/salesz9902/textbook/blob/master/files/caesar/exor_task/exor.c)

**A programkód Bátfai Norbert tulajdonában van.**

**Ebben a feladatban Butcovan György tutorált.**

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

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

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

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

        titkos[i] ^= kulcs[i%kulcs_meret];
    }
}

char *
szo_xor(char * szo,const char kulcs[],int index)
{
    int len=strlen(szo);

    for(int i=0;i<len;i++)
    {
        szo[i]^=kulcs[index%KULCS_MERET];
        index++;
    }
    return szo;
}

int
main (void)
{

    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
```

```
char *p = titkos;
int olvasott_bajtok;
char w[20];

// 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';

//printf("hossz:%d\n",strlen(titkos));

// 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;

for(int jj=0;jj<KULCS_MERET;jj++)
{
    strcpy(w,"és ");
    if(memmem(titkos,p-titkos,szo_xor(w,kulcs,jj),strlen(w))) //talalt =1, ↵
        vagy nem=0
    {
for(int kk=0;kk<KULCS_MERET;kk++)
{
    strcpy(w,"amelyik ");
    if(memmem(titkos,p-titkos,szo_xor(w,kulcs,kk),strlen(w))) //talalt =1, ↵
```

```
vagy nem=0
{
    for(int ll=0;ll<KULCS_MERET;ll++)
    {
        strcpy(w, " olyan ");

        if(memmem(titkos,p-titkos,szo_xor(w,kulcs,ll),strlen(w))) // ←
            talalt =1, vagy nem=0
        { //printf("%s\n",w);
            for(int mm=0;mm<KULCS_MERET;mm++)
            {
                strcpy(w,"ezért ");
                if(memmem(titkos,p-titkos,szo_xor(w,kulcs,mm), ←
                    strlen(w))) //talalt =1, vagy nem=0
                {
                    exor(kulcs, KULCS_MERET, titkos, p - titkos);
                    printf("Kulcs: [%c%c%c%c%c%c%c]\nTiszta szöveg: ←
                        [%s]\n",
                        ii, ji, ki, li, mi, ni, oi, pi, titkos);
                    // ujra EXOR-ozunk, így nem kell egy masodik ←
                    buffer
                    exor(kulcs, KULCS_MERET, titkos, p - titkos);
                }
            }
        }
    }
}
}

return 0;
}
```

Ha tudjuk a kulcsot, illetve a karakterhosszat, könnyen feltörhetjük az exor titkosított kódot az adatok megadásával. Ezzel a módszerrel akár üzenhetünk is társunknak, illetve olyan szövegeket törhetünk vele, amiről tudjuk mi alapján lett titkosítva.

Ebben a programkódban a Brute force-ot alkalmazzuk, annak segítségével törjük fel a szöveget és a kulcsot, ez addig fut, amíg nem ütközik helyes megoldásba.

Van egy függvényünk, név szerint exor, ami maga az exor művelet végzi el számunkra. A kulcsokat egy

kulcs nevű tömbben tároljuk, mégpedig a main függvényen belül, illetve az olvasott bájtok mérete szintén itt van tárolva. A read segítségével beolvassuk a titkos szöveget, ezzel már visszakapjuk visszatérési értékként a beolvasott szöveget is.

## 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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R)

A neurális hálózat biológiai neuronok összekapcsolt csoporthja. Két koncepcióból jött létre, mégpedig a biológiai és a mesterséges neurális hálózatok ötvözetéből. Az agyunkban is neuronok találhatóak, amik hasonló elven működnek, mint itt, csak magától értetődően természetes módon...

Itt egy neurális hálót készítünk R nyelvben. A neurális háló mesterséges oldala úgy néz ki, hogy megadjuk a programunknak, milyen bemenetre milyen kimenetet adjon, aztán ezt a programunk ennek alapján elkezdi leutánozni egy bizonyos szinten.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/caesar/perceptron/ql.hpp> <https://github.com/salesz9902/textbook/blob/master/files/caesar/perceptron/main.cpp>

A programkódok Bátfai Norbert tulajdonában vannak.

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

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

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

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

    double* image = new double[size];

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

    double value = (*p)(image);
```

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

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

Ezt a neuronmodellt a 20. század közepén használták először hatékony képfelismerő algoritmusként.

A perceptron hátránya, hogy kettőnél több réteg esetén a tanítása nehezen kivitelezhető, ugyanis azok a gradiensereszkedések, melyek egy veszteségfüggvényt próbálnak iteratív módon minimalizálni, és ehhez a függvény gradiensével számolnak.

A program fordításához telepítenünk kell a libpng++-t.

```
sudo apt install libpng++
```

A programot a következőképpen kell fordítanunk:

```
g++ ql.hpp main.cpp -o perc -lpng -std=c++11
```

Aztán futtatni a következőképp:

```
./perc (something.png)
```

Használjuk például a mandelbrotnál legenerált png képünket.

A programunk kielemzi a képet, aztán a képünk alapján visszaad egy értéket.

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/mandelbrot/mandelbrot.cpp>

A Mandelbrot-halmaz a komplex számsíkon különböző pontok halmaza. Van rá egy rekurzív sorozat, amely abszolút értéken korlátos.

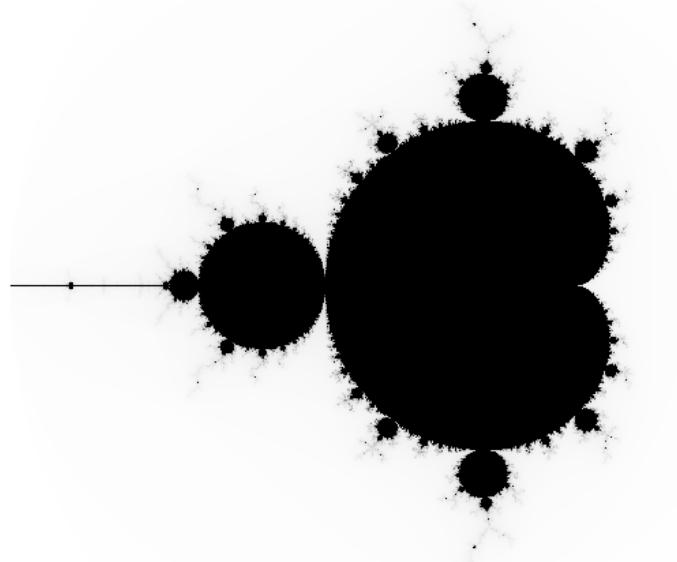
A rekurzív sorozat az alábbi:  $x_{n+1} := (x_n)^2 + c$

A fenti C++ programban a Mandelbrot-halmazt fogjuk ábrázolni, mégpedig egy .png kiterjesztésű képen.

Miután lefordítottuk a kódunkat a következőképpen:

```
g++ mandelbrot.cpp -lpng -o mandel
```

A forráskódban látszik, hogy futtatás után kapunk egy képet kimenet.png néven egész érdekes eredménnyel.

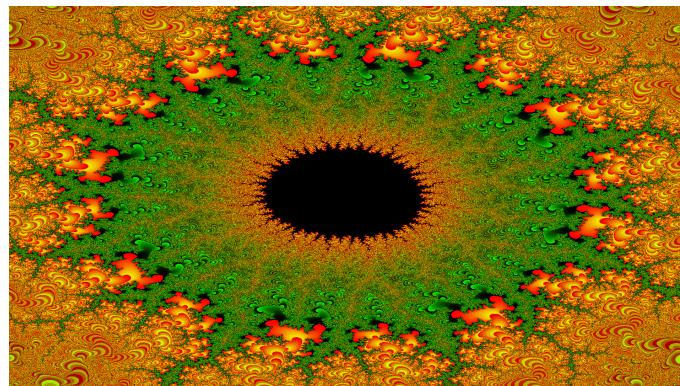


*Mandelbrot által generált default image*

Ha a másik módon futtatjuk le a programot, ahol már pici manuálisabban adhatunk meg több értéket is számára, mint például: szélesség, magasság stb.

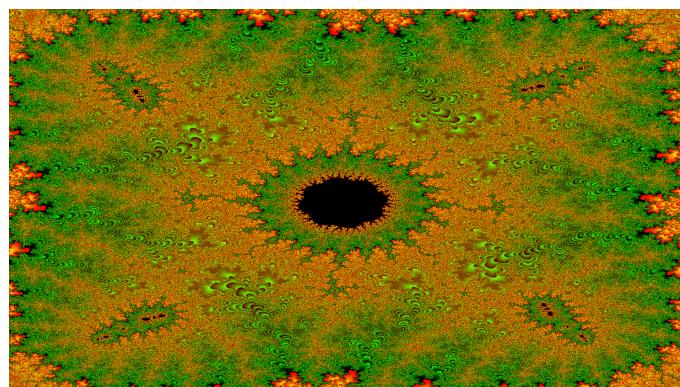
```
./3.1.2 mandel.png 1920 1080 2040 ←  
-0.01947381057309366392260585598705802112818 ←  
-0.0194738105725413418456426484226540196687 ←  
0.798505756933826860155341774655971676111 ←  
0.798505756934379196110285192844457924366
```

Például ezt kipróbálva a következő kimenetet kapjuk szintén kép formájában:



*Mandelbrot által generált image*

Egy pici másabb generált kép:



*Mandelbrot által generált image*

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/mandelbrot/mandelbrot2.cpp>

**Ebben a feladatban György Dóra tutoráltja voltam.**

Ezzel a programmal szintén a Mandelbrot-halmazt ábrázoljuk, viszont itt már `std::complex` osztállyal tesszük meg struktúra alkalmazása helyett. Miután fordítottuk, itt is ugyanazt a képet láthatjuk futtatásnál, mint az előbbinél, csupán a forráskód van másképp kivitelezve.

Itt elhagyjuk a struktúra használatát, helyette osztályt használunk. Talán a struktúra használata talán előnyösebb bizonyos szempontokból, bár egyáltalán nincs köztük olyan nagy különbség, hogy erős okunk legyen rá.

Deklaráljuk a `reC`, `imC`, `reZ` és `imZ` változókat, itt már sejtjük, hogy a komplex számoknak nagy szerepe lesz a programunkban. Ha programunk sikeresen elvégzett minden műveletet, akkor elmentjük a az álláspontot, képet, aztán ezt közöljük is a standard kimeneten.

Fordítás:

```
g++ mandelbrot2.cpp -lpng -O3 -o mandelbrot2
```

Futtatás:

```
./mandelbrot2.cpp mandelcomplex.png 1920 1080 1020  
0.4127655418209589255340574709407519549131  
0.4127655418245818053080142817634623497725  
0.2135387051768746491386963270997512154281  
0.2135387051804975289126531379224616102874
```

## 5.3. Biomorfok

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

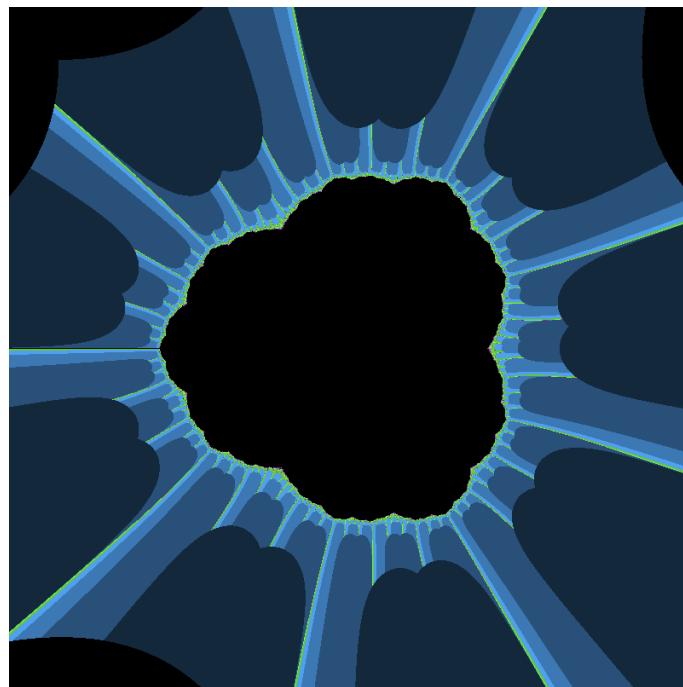
Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/mandelbrot/biomorf.cpp>

A biomorfok jelentősen közel állnak a Mandelbrot-halmazhoz, ugyanis itt szintén a komplex számsíkkal dolgozik a programunk.

Programunk fordítása után, a következőképpen futtassuk:

```
./bmorf bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
```

A fenti esetben ismét egy png kiterjesztésű fájlt fogunk kapni, mégpedig `bmorf.png` néven. Itt már sokkal színgazdagabb formát fogunk kapni, ami picit látványosabb is.



Biomorfok által generált image

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása:

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or
// modify
// it under the terms of the GNU General Public License as
// published by
// the Free Software Foundation, either version 3 of the License,
// or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public
// License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
```

```
//  
// Mandelbrot png  
// Programozó Páternoszter/PARP  
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←  
_01_parhuzamos_prog_linux  
//  
// https://youtu.be/gvaqijHlRUs  
//  
  
#include <png++/image.hpp>  
#include <png++/rgb_pixel.hpp>  
  
#include <sys/times.h>  
#include <iostream>  
  
#define MERET 600  
#define ITER_HAT 32000  
  
__device__ int  
mandel (int k, int j)  
{  
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:  
    // most eppen a j. sor k. oszlopaban vagyunk  
  
    // számítás adatai  
    float a = -2.0, b = .7, c = -1.35, d = 1.35;  
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ←  
        ITER_HAT;  
  
    // a számítás  
    float dx = (b - a) / szelesseg;  
    float dy = (d - c) / magassag;  
    float reC, imC, reZ, imZ, ujreZ, ujimZ;  
    // Hány iterációt csináltunk?  
    int iteracio = 0;  
  
    // c = (reC, imC) a rács csomópontjainak  
    // megfelelő komplex szám  
    reC = a + k * dx;  
    imC = d - j * dy;  
    // z_0 = 0 = (reZ, imZ)  
    reZ = 0.0;  
    imZ = 0.0;  
    iteracio = 0;  
    // z_{n+1} = z_n * z_n + c iterációk  
    // számítása, amíg |z_n| < 2 vagy még  
    // nem értük el a 255 iterációt, ha  
    // viszont elértek, akkor úgy vesszük,  
    // hogy a kiinduláci c komplex számra
```

```
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteracionsHatar)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;

}
return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{

    int j = blockIdx.x;
    int k = blockIdx.y;

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

}
*/

__global__ void
mandelkernel (int *kepadat)
{

    int tj = threadIdx.x;
    int tk = threadIdx.y;

    int j = blockIdx.x * 10 + tj;
    int k = blockIdx.y * 10 + tk;

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

}

void
cudamandel (int kepadat [MERET] [MERET])
{

    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (←
```

```
    int));

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

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

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

}

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

    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

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

    int kepadat [MERET] [MERET];

    cudamandel (kepadat);

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

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

```
255 -  
(255 * kepadat[j][k]) / ←  
ITER_HAT));  
}  
}  
kep.write(argv[1]);  
  
std::cout << argv[1] << " mentve" << std::endl;  
  
times(&tmsbuf2);  
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime  
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;  
  
delta = clock() - delta;  
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;  
}  
}
```

Szintén egy összetett feladattal állunk szemben. Ahhoz, hogy megfelelően tudjuk fordítani/futtatni a programot, telepítenünk kell az nvidia-cuda-toolkit nevű csomagot.

A programunk konkrétan optimalizálni próbálja a "munkánkat", egy gyorsabb számolást eredményez a háttérben, ami nagyon sok esetben nagy segítségünkre lehet, hiszen ki ne akarná, hogy gyorsabban dolgozzon a gépe.

Az optimalizálásról már korábban is volt szó a könyvben, mégpedig a legelső feladatunkban például, ahol OpenMP segítségével osztottuk fel processzormagokra a végrehajtandó "munkát".

## 5.5. Mandelbrot nagyító és utazó C++ nyelven

**A feladatban Molnár Antal Albert könyve volt segítségemre.**

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

Megoldás forrása: <https://github.com/salesz9902/prog1/tree/master/mandelzoom>

Ebben a feladatban egy GUI-t fogunk létrehozni a Qt Creator szoftverrel. (Ez egy multiplatformos keretrendszer, amit épp erre (is) találtak ki.)

Tehát itt arról van szó, hogy a Qt Creatorban létre tudunk könnyen hozni egy grafikus felületet az előző C++ kódunkhoz, a Mandelbrot-hoz.

YouTube-on rengeteg oktatóvideót találhatunk a Qt Creator szoftverről. Itt is van egy:

<https://www.youtube.com/watch?v=3SIj6zL6mmA>

Ebből a videóból már tényleg gond nélkül elindulhatunk egy úton a GUI szerkesztés felé.

## 5.6. Mandelbrot nagyító és utazó Java nyelven

Hasonló szituációban vagyunk, mint az előző feladatnál. Annyi változik, hogy már egy jóval felhasználóközelibb, magasabb szintű programozási nyelven valósítjuk meg, a Javában.

Itt is rengeteg keretrendszerünk van. A programban nagy a testreszabhatóság lehetősége. Mi magunk adhatjuk meg több paramétereit is a programunk elindulásakor felnyíló ablaknak stb.

A konstruktörben beállíthatjuk a Mandelbrot halmaz paramétereit. Például az élességet. Ami még érdekes lehet számunkra, az nem más, mint a BufferedImage típus, amit a Java biztosít számunkra. Ez tulajdonképpen egy osztály, ami lehetőséget ad, hogy külső könyvtár használata nélkül is képesek legyünk egyszerűen képfájlokat létrehozni.

## 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 forrása:

```
$ ./java/bin/java polargen.java
-0.7353431820414118
-0.33784190028284766
0.7750031835316805
0.5524713543467192
-0.5380423283211784
1.512849268596637
2.7148874695500966
-0.23688836801277952
-0.3238588036816322
-0.7963150809415576
$ ./java/bin/java polargen.java
-0.6566325405553158
0.40465899229436114
0.08634239512228409
-0.9470321445590416
0.1926238606249351
0.7705517022243931
0.9084531239664848
-1.4472688950554047
-1.6250659297425345
-0.7791586500972545
```

A program 10 darab véletlenszerűen generált normalizált számot köp ki, ahogyan azt várjuk is.

## 6.2. LZW

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Van egy osztályunk: LZWBInFa, ez építi fel a bináris fájlunkat az általunk beírt bemeneti fájlból. A következőképp kell futtatnunk a kódot:

```
./bin [bemeneti] -o [kimeneti]
```

A kimeneti fájlhoz értelemszerűen megadjuk, hova szeretnénk kiíratni az eredményt.

A programunk sokkal egyszerűbb módon van megírva C-ben. Ugyanis itt már elégé meg van kötve a kezünk a program írásában. Mondhatjuk, hogy ugyanaz a programkód, csak leegyszerűsítve, pár dolgot kivéve az eredeti c++ kódunkból.

## 6.3. Fabejárás

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Inorder: először a fa bal oldalát járjuk be, a gyökeret, aztán majd a jobb oldalát.

Preorder: gyökérrel indítunk, majd bezárjuk fa bal oldalát, aztán a fa jobb oldalát.

Postorder: gyökérrel indítunk, fa jobb oldalát, aztán a fa bal oldalát járjuk be.

Itt már viszont a programot picit másképpen futtatjuk:

```
./binfa [bemeneti] -o [kimeneti] [o /r]
```

Ha az utolsó argumentum o, postorder, ha pedig r, akkor inorder bejárást fog alkalmazni.

## 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 forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Itt az alapvetően C alapú LZWBInFa programkódunkat kellene átírni C++-ba. Ez nem feltétlenül bonyolult feladat. Mivel már el kell mozdulnunk picit az objektumorientált irányba, így egy külön osztályba kell megírnunk a fent említett dolgokat.

## 6.5. Mutató a gyökér

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

Megoldás forrása: [https://github.com/salesz9902/textbook/blob/master/files/welch/mutato\\_a\\_gyok.cpp](https://github.com/salesz9902/textbook/blob/master/files/welch/mutato_a_gyok.cpp)

Ebben az esetben a bináris fa gyökere egy mutató kell, hogy legyen. Ez semmit sem változtat a többi, vagy akár az eredeti LZWBinFa-hoz képest. Ugyanaz a végkimenetele, minden ugyanúgy működik, csak szimplán másiképp van megoldva.

## 6.6. Mozgató szemantika

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Itt használjuk a mozgató-konstruktorkat. A következő változásoknak kell végbemennük a kódunkban:

```
Csomopont * masol ( Csomopont * elem, Csomopont * regifa ) {
    Csomopont * ujelem = NULL;
    if ( elem != NULL ) {
        switch ( elem->getBetu() ) {
            case '/':
                ujelem = new Csomopont ( '/' );
                break;
            case '0':
                ujelem = new Csomopont ( '1' );
                break;
            case '1':
                ujelem = new Csomopont ( '0' );
                break;
            default:
                std::cerr<<"HIBA!"<<std::endl;
                break;
        }
        ujelem->ujEgyesGyermek (
            masol ( elem->egyesGyermek () , regifa )
        );
        ujelem->ujNullasGyermek (
            masol ( elem->>nullasGyermek () , regifa )
        );
        if ( regifa == elem )
            fa = ujelem;
    }
    return ujelem;
}
```

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyszimulációk

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

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

Megoldás forrása: <https://github.com/salesz9902/textbook/tree/master/files/conway/hangya>

Nem hiába kapta a feladatunk ezt a nevet. Hiszen programunk a hangyák viselkedését szimulálja le. Figyeljük meg, ahogyan véletlenszerűen változik az apró pontoknak a helyzete, aztán egyre több lesz belőlük. Egyszer itt gyűlnek össze, egyszer ott. Tisztára olyan viselkedést mutatnak mint a hangyák egy élő szituációban.

Az elején elkezdenek egy adott pontból elindulni. Olyan, mintha egy ételmaradékon összegyűlnének, aztán sorban egymás után elkezdenék haza cipelni azokat. A hangyáknak ez egy nagyon jellegzetes tulajdonságuk. A programot sokáig hagyhatjuk futni, hiszen a végtelenségig szimulálja a dolgokat számunkra.

### 7.2. Java életjáték

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

**A feladatot Racs Tamás könyve alapján oldottam meg.**

**Az alábbi programkód egésze Bátfai Norbert tulajdoná.**

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/conway/sejtautomata.java>

```
javacode
{
    /*
     * Sejtautomata.java
     *
     * DIGIT 2005, Javat tanítok
     * Batfai Norbert, nbatfai@inf.unideb.hu
     */
```

```
 */
/*
 * Sejtautomata osztaly.
 *
 * @author Batfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class Sejtautomata extends java.awt.Frame ←
    implements Runnable {
    /* Egy sejt lehet elo */
    public static final boolean ELO = true;
    /* vagy halott */
    public static final boolean HALOTT = false;
    /* Ket racsot hasznalunk majd, az egyik a sejtter ←
       allapotat
    * a t_n, a masik a t_n+1 idopillanatban jellemzi. ←
       */
    protected boolean[][][] racsok = new boolean ←
        [2][][];
    /* Valamelyik racsra mutat, technikai jellegű, hogy ←
       ne kelljen a
    * [2][][]-bol az elso dimenziot hasznalni, mert ←
       vagy az egyikre
    * allitjuk, vagy a masikra. */
    protected boolean[][] racs;
    /* Megmutatja melyik racs az aktualis: [racsIndex ←
       ][][] */
    protected int racsIndex = 0;
    /* Pixelben egy cella adatai. */
    protected int cellaSzelesseg = 20;
    protected int cellaMagassag = 20;
    /* A sejtter nagysaga, azaz hanyszor hany cella van? ←
       */
    protected int szelesseg = 20;
    protected int magassag = 10;
    /* A sejtter ket egymast koveto t_n es t_n+1 ←
       diszkret idopillanata
       kozotti valos ido. */
    protected int varakozas = 1000;
    // Pillanatfelvetel keszitesehez
    private java.awt.Robot robot;
    /* Keszitsunk pillanatfelvetelt? */
    private boolean pillanatfelvetel = false;
    /* A pillanatfelvetelek szamzasahoz. */
    private static int pillanatfelvetelSzamlalo = 0;
    /*
     * Letrehoz egy <code>Sejtautomata</code> objektumot ←
     *
     * @param      szelesseg      a sejtter szelessege.
```

```
* @param      magassag      a sejtter szelessege.
*/
public Sejtautomata(int szelesseg, int magassag) {
    this.szelesseg = szelesseg;
    this.magassag = magassag;
    // A ket racs elkeszitese
    racsok[0] = new boolean[magassag][szelesseg];
    racsok[1] = new boolean[magassag][szelesseg];
    racsIndex = 0;
    racs = racsok[racsIndex];
    // A kiindulo racs minden cellaja HALOTT
    for(int i=0; i<racs.length; ++i)
        for(int j=0; j<racs[0].length; ++j)
            racs[i][j] = HALOTT;
    // A kiindulo racsra "elolenyeket" helyezunk
    //siklo(racs, 2, 2);
    sikloKilovo(racs, 5, 60);
    // Az ablak bezarasakor kilepunk a programbol.
    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
            setVisible(false);
            System.exit(0);
        }
    });
    // A billentyűzetrol erkezo esemenyek feldolgozasa
    addKeyListener(new java.awt.event.KeyAdapter() {
        // Az 'k', 'n', 'l', 'g' es 's' gombok lenyomasat figyeljuk
        public void keyPressed(java.awt.event.KeyEvent e) {
            if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
                // Felezuk a cella mereteit:
                cellaSzelesseg /= 2;
                cellaMagassag /= 2;
                setSize(Sejtautomata.this.szelesseg*cellaSzelesseg,
                        Sejtautomata.this.magassag*cellaMagassag);
                validate();
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                // Duplazzuk a cella mereteit:
                cellaSzelesseg *= 2;
                cellaMagassag *= 2;
                setSize(Sejtautomata.this.szelesseg*cellaSzelesseg,
```

```
        Sejtautomata.this.magassag* ←
            cellaMagassag);
        validate());
    } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
        pillanatfelvetel = !pillanatfelvetel ←
            ;
    else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
        varakozas /= 2;
    else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
        varakozas *= 2;
    repaint();
}
});

// Eger kattinto események feldolgozása:
addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent m) {
        // Az egermutató pozíciója
        int x = m.getX()/cellaSzelesseg;
        int y = m.getY()/cellaMagassag;
        racsok[racsIndex][y][x] = !racsok[←
            racsIndex][y][x];
        repaint();
    }
});
// Eger mozgas események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // Vonzolással jelöljük ki a negyzetet:
        int x = m.getX()/cellaSzelesseg;
        int y = m.getY()/cellaMagassag;
        racsok[racsIndex][y][x] = ELO;
        repaint();
    }
});
// Cellameretek kezdetben
cellaSzelesseg = 10;
cellaMagassag = 10;
// Pillanatfelvetel készítéséhez:
```

```
try {
    robot = new java.awt.Robot(
        java.awt.GraphicsEnvironment.
        getLocalGraphicsEnvironment().
        getDefaultScreenDevice());
} catch(java.awt.AWTException e) {
    e.printStackTrace();
}
// A program ablakanak adatai:
setTitle("Sejtautomata");
setResizable(false);
setSize(szelesseg*cellaSzelesseg,
    magassag*cellaMagassag);
setVisible(true);
// A sejtter eletrekeltese:
new Thread(this).start();
}
/* A sejtter kirajzolása. */
public void paint(java.awt.Graphics g) {
    // Az aktualis
    boolean [][] racs = racsok[racsIndex];
    // racsot rajzoljuk ki:
    for(int i=0; i<racs.length; ++i) { // vegig ←
        lepek a sorokon
        for(int j=0; j<racs[0].length; ++j) { // s ←
            az oszlopok
            // Sejt cella kirajzolása
            if(racs[i][j] == ELO)
                g.setColor(java.awt.Color.BLACK);
            else
                g.setColor(java.awt.Color.WHITE);
            g.fillRect(j*cellaSzelesseg, i* ←
                cellaMagassag,
                cellaSzelesseg, cellaMagassag);
            // Racs kirajzolása
            g.setColor(java.awt.Color.LIGHT_GRAY);
            g.drawRect(j*cellaSzelesseg, i* ←
                cellaMagassag,
                cellaSzelesseg, cellaMagassag);
        }
    }
    // Készítünk pillanatfelvetelt?
    if(pillanatfelvetel) {
        // a biztonság kedveert egy kép készítése ←
        utan
        // kikapcsoljuk a pillanatfelvetelt, hogy a
        // programmal ismerkedő olvasó ne irja tele ←
        a
        // fájlrendszeret a pillanatfelvetelekkel
        pillanatfelvetel = false;
    }
}
```

```
        pillanatfelvetel(robot.createScreenCapture
            (new java.awt.Rectangle
                (getLocation().x, getLocation().y,
                szelesseg*cellaSzelesseg,
                magassag*cellaMagassag)));
    }
}
/*
 * Az kerdezett allapotban levo nyolcszomszedok ←
 * szama.
 *
 * @param racs a sejtter racs
 * @param sor a racs vizsgalt sora
 * @param oszlop a racs vizsgalt oszlopa
 * @param allapot a nyolcszomszedok vizsgalt ←
 *     allapota
 * @return int a kerdezett allapotbeli ←
 *     nyolcszomszedok szama.
 */
public int szomszedokSzama(boolean [][] racs,
    int sor, int oszlop, boolean allapot) {
    int allapotuSzomszed = 0;
    // A nyolcszomszedok vegigzongorazása:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgalt sejtet magat kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejtterbol szelenek szomszedai
                // a szembe oldalakon ("periodikus ←
                // hatarfeltetel")
                int o = oszlop + j;
                if(o < 0)
                    o = szelesseg-1;
                else if(o >= szelesseg)
                    o = 0;

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

                if(racs[s][o] == allapot)
                    ++allapotuSzomszed;
            }
    return allapotuSzomszed;
}
/*
 * A sejtter idobeli fejlodese a John H. Conway fele

```

```
* eletjatek sejtautomata szabalyai alapjan tortenik ←
.
*
* A szabalyok reszletes ismerteteset lasd peldaul a
* [MATEK JATEK] hivatkozasban (Csakany Bela: ←
* Diszkret
* matematikai jatekok. Polygon, Szeged 1998. 171. ←
* oldal.)
*/
public void idoFejlodes() {

    boolean [][] racsElotte = racsok[racsIndex];
    boolean [][] racsUtana = racsok[(racsIndex+1) ←
        %2];

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

                    int elok = szomszedokSzama(racsElotte, i ←
                        , j, ELO);

                    if(racsElotte[i][j] == ELO) {
                        /* Elo elo marad, ha ketto vagy harom ←
                        elo
                        szomszedja van, kulonben halott lesz. ←
                        */
                        if(elok==2 || elok==3)
                            racsUtana[i][j] = ELO;
                        else
                            racsUtana[i][j] = HALOTT;
                    } else {
                        /* Halott halott marad, ha harom elo
                        szomszedja van, kulonben elo lesz. */
                        if(elok==3)
                            racsUtana[i][j] = ELO;
                        else
                            racsUtana[i][j] = HALOTT;
                    }
                }
            }
        racsIndex = (racsIndex+1)%2;
    }
/* A sejtter idobel fejlodese. */
public void run() {

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

```
        idoFejlodes();
        repaint();
    }
}

/*
 * A sejtterbe "elolenyeket" helyezunk, ez a "siklo ←
 * ".
 * Adott irányban halad, masolja magat a sejtterben.
 * Az eloleny ismerteteset lasd peldaül a
 * [MATEK JATEK] hivatkozasban (Csakany Bela: ←
 * Diszkret
 * matematikai jatekok. Polygon, Szeged 1998. 172. ←
 * oldal.)
 *
 * @param racs      a sejtter ahova ezt az allatkat ←
 *                   helyezzük
 * @param x         a befoglalo tegla bal felső ←
 *                   sarkanak oszlopa
 * @param y         a befoglalo tegla bal felső ←
 *                   sarkanak sora
 */
public void siklo(boolean [][] racs, int x, int y) {

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

}
/*
 * A sejtterbe "elolenyeket" helyezunk, ez a "siklo ←
 * agyu".
 * Adott irányban siklokat lo ki.
 * Az eloleny ismerteteset lasd peldaül a
 * [MATEK JATEK] hivatkozasban /Csakany Bela: ←
 * Diszkret
 * matematikai jatekok. Polygon, Szeged 1998. 173. ←
 * oldal./,
 * de itt az abra hibas, egy oszloppal told meg ←
 * balra a
 * bal oldali 4 sejtes negyzetet. A helyes agyu ←
 * rajzat
 * lasd pl. az [ELET CIKK] hivatkozasban /Robert T.
 * Wainwright: Life is Universal./ (Megemlithetjuk, ←
 * hogy
 * mindenketto tartalmaz ket felesleges sejtet is.)
 *
 * @param racs      a sejtter ahova ezt az allatkat ←
 *
```

```
    helyezzuk
    * @param x      a befoglalo tegla bal felső ←
        sarkanak oszlopa
    * @param y      a befoglalo tegla bal felső ←
        sarkanak sora
    */
    public void sikloKilovo(boolean [][] racs, int x, ←
        int y) {
        racs[y+ 6][x+ 0] = ELO;
        racs[y+ 6][x+ 1] = ELO;
        racs[y+ 7][x+ 0] = ELO;
        racs[y+ 7][x+ 1] = ELO;

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

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

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

        racs[y+ 6][x+ 11] = ELO;
        racs[y+ 6][x+ 15] = ELO;
        racs[y+ 6][x+ 16] = ELO;
        racs[y+ 6][x+ 22] = ELO;
        racs[y+ 6][x+ 23] = ELO;
        racs[y+ 6][x+ 24] = ELO;
        racs[y+ 6][x+ 25] = ELO;

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

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

        racs[y+ 9][x+ 13] = ELO;
        racs[y+ 9][x+ 21] = ELO;
        racs[y+ 9][x+ 22] = ELO;
        racs[y+ 9][x+ 23] = ELO;
```

```
        racs[y+ 9][x+ 24] = ELO;
        racs[y+ 9][x+ 34] = ELO;
        racs[y+ 9][x+ 35] = ELO;

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

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

    }

/* Pillanatfelvetelek készítése. */
public void pillanatfelvetel(java.awt.image.←
    BufferedImage felvetel) {
    // A pillanatfelvetel kép fajlneve
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvetelSzamlalo);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelne a vaszon felületet).
public void update(java.awt.Graphics g) {
    paint(g);
}
/*
 * Peldányosít egy Conway-féle életjátek szabályos
 * sejtter obektumot.
*/
public static void main(String[] args) {
    // 100 oszlop, 75 sor merettel:
    new Sejtautomata(100, 75);
}
```

Itt John Conway Neumann János munkáját felhasználva állt elő az úgynevezett életjátékkal. Magát a kifejezést: sejtautomata, Neumann János találmánya.

A következő szabályok módosítják az egyes cellákat tartalmuktól függően: Egy sejt két, vagy 3 szomszéd esetén maradhat életben; Ha túlnépesedés van, akkor a sejt elpusztul, ergő, ha 3-nál több szomszéddal rendelkezik, szintén meghal; ha pontosan 3 élő szomszédja van, egy új sejt kel életre.

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: <https://github.com/salesz9902/textbook/tree/master/files/conway/sejtauto>

**Ebben a feladatban Nagy Krisztián tutorált.**

Ismét egy elégé árulkodó nevet kaptunk feladatunknak. Itt különböző pici "sejtautók" mozgásának szemtanú lehetünk! Az apró mozgalmas sejtek egy-egy élő sejt viselkedését próbálják reprezentálni. Látható, ahogy úgymond legyártódnak a sejtek, aztán elindulnak egymás után egy irányba.

Aztán ahogyan a képernyőnk széléhez érnek, egy újabb részen megjelennek, aztán addig-addig osztódnak, amíg rengeteg nem lesz belőlük.

Alapvetően a sejtek a valóságban is hasonlóan működnek. Kell, hogy legyen egy-egy társuk, különben elpusztulnak.

## 7.4. BrainB Benchmark

Megoldás forrása: <https://github.com/salesz9902/textbook/tree/master/files/conway/brainB>

Itt a Qt keretrendszer segítségével tudjuk megoldani a feladatot. Fontos, hogy amíg eljutunk addig, hogy a programunk megfelelő környezetben megfelelően lefordul és lefut, rengeteg helyet kell annak igénybe vennie. Tehát készüljünk fel rá, hogy legyen 20-30 GB szabad helyünk, hogy kényelmes legyen a program kezelése minden szempontból.

A Qt keretrendszer nagyon nagy eszközökkel rendelkezik, és rengeteg lehetőségünk van kihasználni ezeket. A nagy eszközökkel a szükséges letöltött dolgok méretében is erősen megnyilvánul.

E program célja, hogy benchmarkot készítsen egy E-sportoló játékosról. Felmére annak tudását, amely segítségével már könnyen besorolható lesz egy adott szintre. A program futtatás után, és akár pár perc játék után részletes leírásokat kapunk kimenetként. Ennek segítségével akár már irányt kaphatunk E-sport karrierünk felépítésében, kérdésében.

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

aa Python

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.2. Mély MNIST

Python

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9. fejezet

# Helló, Chaitin!

### 9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

### 9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

### 9.3. Gimp Scheme Script-fu: név mandala

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

Megoldás videó: [https://bhaxor.blog.hu/2019/01/10/a\\_gimp\\_lisp\\_hackelete\\_a\\_scheme\\_programozasi\\_nyelv](https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelete_a_scheme_programozasi_nyelv)

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

Tanulságok, tapasztalatok, magyarázat...

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Juhász István - Magas szintű programozási nyelvek 1

[?]

Alapfogalmak:

Gépi nyelv: A gépi nyelv, egy olyan nyelv, amely a számítógép számára közvetlenül értelmezhető. Kettes vagy tizenhatos számrendszeren alapul (számokkal ábrázolandó).

Magas szintű nyelv: A magas szintű programozási nyelvek már felhasználó közelibbek. Ezek nem értelmezhetőek közvetlenül a számítógép által. Itt már szükségünk van egy fordítóra, ami lefordítja gépi nyelvre, ahhoz, hogy futtatható legyen.

Gépi nyelvezetű avagy alacsony szintű programnyelv például az Assembly, amelyben sokkal nehezebben igazodunk el, hisz egyértelműen látszik, hogy a géphez áll közelebb. Viszont magas szintű programnyelv például a C, amelyben érzékelhetjük is, hogy sokkal jobban érthetőek a C-ben írt kódok, mint akár Assemblyben. Hisz C-ben mondhatjuk, hogy angol kulcsszavakkal adunk ki "parancsokat" a számítógép számára, ami persze ugyanúgy lefordul majd gépi kódra, aztán futtathatóvá is válik.

Egy programot tudunk szintaktikailag, illetve szemantikailag elemezni. Szintaktikai elemzésnél konkrétan a programkódunk "helyesírására" figyelünk. Tehát, hogy nem-e írtunk el egy adott parancsot például, stb.. Szemantikai elemzésnél már arra figyelünk, hogy miután szintaktikailag helyes a programunk, helyesen fut-e le. Tehát itt azt nézzük, hogy tényleg azt csinálja-e a programunk, ami a célunk volt vele. Helyesen fut-e le.

A programnyelveket két fő csoportba soroljuk: vannak imperatív és dekleratív nyelvek. Az imperatív nyelvek általában az értékadó utasítások megfelelő sorrendben való kiadására koncentrálnak. Ez az a típus, amelyben feltehetően többen programoznak, bár nem feltétlenül, de ha valaki komolyabb programozásra vágyik, ezzel a típussal kezdi el a gyakorlást, majd folytatja a bonyolultabb programokkal. Imperatívak például az eljárásorientált nyelvek, vagy az objektumorientált nyelvek.

A dekleratív csoportba sorolható programkódoknál általában a programíró arra koncentrál, hogy mit szeretne kapni az adott program futása során. Ilyenek például a funkcionális nyelvek, illetve a logikai nyelvek is.

Fontos megjegyezni, hogy elméleti szinten nem fogunk megtanulni programozni. Ezalatt azt értem, hogy ahhoz, hogy valaki jó programozvá váljon, rengeteg programkódot kell átvészelnie mind elméletben, de legfőképpen gyakorlatban.

Utasítások:

Sokféle utasítás létezik. Ilyenek például az értékkadó, üres, ugró, elágaztató, ciklusszervező, hívó, I/O illetve számos egyéb utasítások.

Itt azért pár utasítás eléggé magától értendő. Mint például az értékkadó utasításokkal egy vagy több változó értékkomponensét állítjuk be, vagy éppen módosítjuk.

Az elágaztató utasítások például az if feltétellel kapcsolatos megoldásokat fedi le, azaz valamelyen feltételek, kétirányú logikai utasítás. Ezekből vannak egyirányú, illetve többirányúak is. Ide tartozik a switch utasítás is.

Ez a könyv összegezve egy erősen elméletorientált könyv. Az ilyesfél könyvek is nagyon sokat hozzá tudnak tenni a fejlődésünkhez. Fontos, hogy tudunk beszálni is arról amit csinálunk, ne csak a gyakorlati pályán arassunk. Egy jó programozó nyelvbotlás nélkül közli társai felé a felmerült problémákat, ráadásul maga a programozás nyelvén...

## 10.2. Kernighan Ritchie - A C programozási nyelv

[KERNIGHANRITCHIE]

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

Ez a könyv egy tematikusan felépülő könyv, amely a C nyelv elsajátításához segíti hozzá olvasóját. Próbálja megismertetni a C nyelvet elég erős szinten, próbál bizonyos keretek között picit mélyebben belemenni a dolgokba.

Nagyon sok világhírű, avagy ismert programozó ismerte el ezt a könyvet. Több programozó generáció nőtt fel ezen a könyvön, és még manapság is nagyon hasznosnak minősül, hiszen a könyv erősen pártolja a gondolkodásmód elsajátítását, miközben megtanít clean code-ot iratni az olvasójával. Azt gondolom hihetetlenül jól megírt, logikusan felépített könyv. Pont annyira szól kezdőknek, amennyire kell, nagyon el lett találva.

A könyv alapismeretekkel indít. Hamar lényegre is tör, hiszen már egy "Hello World!" stílusú program megírásával szemléltet az olvasó felé. Erősen ragaszkodik a UNIX-on való fordításra, illetve futtatásra.

Bemutatja a szokásos alap programozási eszközöket. Mint például ciklusok (for, while, do-while), a változó deklarálásától megkezdve a tömbökön át a függvényekig. Kitér külön a változótípusokra, amik működését el is magyarázza, több példán keresztül bemutatja.

Már viszonylag hamar elkezdődnek folyamatos rövid példákkal való szemléltetések, illetve a programkódok kipróbálásra való készítések. A fent leírtak mindegyike érthető módon be van mutatva, le van egyszerű programokba bonyolítva, amelyek értelemszerűen a lehető legjobb megértetésre törekednek. De hiszen nincs ezzel semmi baj, ez így van jól, így a helyes.

Sok dolog magyarázatát a könyv eltolja 3-4-5 fejezettel későbbre, hiszen megértésüket számos más dolog megértése igényli.

Az 1. fejezet tisztázza az alapokat, bár annak ellenére azért nem a legalapvetőbb programkódokkal szemléltet, ami kihívásérzetet kelt az emberben, és talán, mivel az elején még egész jól megérthetők szerintem akár egy laikus számára is, ezért hiába nehéz a feladat, nagyobb lesz a sikerérzete az olvasónak, ezáltal kap egy hatalmas löketet, hogy haladjon tovább, és ez majdnemhogy végig motiválja az olvasót.

A 2. fejezetben kitér a típusokra jobban, adattípusok, méretek, állandók, deklarációk, típuskonverziók, értékadó operátorok, kifejezések. Aztán a 3. fejezetben már megjelennek a vezérlési szerkezetek, a feltételes kifejezések után már több if-else egymásban, switch, while, for, hältutesztelő ciklus. Itt már az olvasó megismerkedhet a break, continue, goto utasításokkal.

A 4. fejezet már jobban szóba kerülnek a függvények, szóba jönnek már fontosabb dolgok, amiket mindenkiépp meg kell értenünk a fejlődés hatékonysága érdekében. Már szó van a rekurzióról, statikus változók, blokkstruktúra inicializálás.

Azt gondolom nem hiába volt már abban az időben is elterjed eme könyv olvasása. Nagyon lényegi módon vezeti rá az olvasót a megfelelő gondolkodás elsajátítására. Szokták mondani, hogy programozni nem fogsz egy könyvből megtanulni, más nem tudja neked megtanítani. Nahát erre a könyvre ez most pont nem klappol...

## 10.3. Programozás

[?]

A könyv a C és C++ nyelvet mutatja be. Már a második részben a lényegre tér. Átveszi az alapvető tudnivalókat, közben összehasonlító módon párhuzamosan íródi mindenkit nyelvről. Ez azért is lehet érdekes számunkra, mert a C++ alapvetően egy objektumorientált nyelv, így C-vel összehasonlítva elég sok eltérés lehet. Még akkor is, ha csak egygel tér el "verziószámuk".

A C inkább alacsony szintű programnyelvnek mondható, mintsem magasnak. Hiszen mondhatjuk, hogy valamilyen szinten közel áll a géphez, talán még közelebb is, mint az emberhez. C-ben rengeteg megoldás furcsa lehet azok számára, akik egyből erősen objektumorientált, magas szintű nyelvekkel ismerkedtek meg.

Viszonylag hamar szó esik a függvények túlterheléséről. Az is egy érdekesség, hogy C-ben a nevük alapján azonosítunk egyértelműen egy függvényt, míg C++-ban a nevük és argumentumlistájuk együttesen azonosítja. Tehát ez azt jelenti számunkra, hogy akár előfordulhat C++-ban azonos nevű függvény is.

C++ nyelvben lehetőségünk van, hogy függvények argumentumainak alapértelmezett értéket adjunk meg. Amennyiben ezen argumentumoknak a függvény hívásakor nem adunk értéket, az az alapértelmezett értékével kerül meghívásra.

Paraméter referenciatípussal: C-ben kizárolag érték szerinti paraméterátadás történik, így az `f` függvény hívásakor az `a` változó értéke a veremre másolódik, és erre kell hivatkoznunk az `i` szimbólummal az `f` függvényen belül.

A 3. részben már rá is tér a könyvünk az objektumokra és osztályokra. Ebben a fejezetben egész jól el van magyarázva az objektumorientáltság alapja. Már egyből egy érdekes felvetéssel kezdődik a fejezet, mégpedig, hogy a számítástecnika fejlődése során egyre jobban feltűnhet számunkra, hogy a szoftveren húzódik le a súly. Hiszen manapság már ha megtudják tenni, inkább a hardvereken spórolnak hatalmasakat, és egyre jobban inkább optimalizálják a dolgokat, mintsem inkább erősebb hardver legyen alatta. A kérdés az, hogy miért? Miért csinálják ezt a legtöbbeni? Csak azért, hogy minél nagyobb profitjuk lehessen?

Minden esetre ez egy egész jól sikeres chapter lett, szerintem teljesen jó irányból lett közelítve az objektumorientálás, még akkor is, ha személy szerint annyira még nem mentem bele a dolgokba.

Mindezek ellenére azért nem mondhatjuk el, hogy lassú tempóban halad a könyv, alapelvárásnak tűnik, hogy önszorgalomból fejlődjünk a könyvvel, mintsem szájbarágósan magyarázzon el minden egyes kis

apró részletet. De jól is van ez így, hiszen ez a lényege a programozásnak, hogy önállóan sajátítsuk el, hiszen máshogyan esélyünk sincs.

Tagváltozók, tagfüggvények: ezen megnevezések a struktúra részei, adattagjai. Ez így összhangban van a C nyelv implementációival is, ami szintén érdekes, hogy . operátorokkal hivatkozhatunk tagváltozókra.

Összességében ez a könyv eléggy gyors tempóban viszi végig a dolgokat. Érdemes egy erős alap után kézbe venni, hiszen anélkül túlságosan sok ismeretlen dologgal találkoznánk, ami nem feltétlen lesz hatékony módszer a fejlődésre. Érdemes először belőni a szintünket, utána pedig cselekedni.

## **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. Olvasónapló: C++: Benedek Zoltán, Levendovszky Tihamér Szoftver C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 Útikalauz programozóknak 5.0 I-II.II.

Bár mind a C++ és Java programozási nyelv magas szintű programozási nyelv, illetve támogatja az objektumorientált módosításokat, mégis találunk nagyobb eltéréseket a nyelvek összehasonlítása során.

A Java erősen osztály alapú és objektumorientált nyelv. Az egyik legjobban eltérő dolg a két nyelv között a hordozhatóság. Ez alatt azt értem, hogy ha egy C++ kódot lefordítok egy linux rendszeren, nem feltétlenül fog lefutni egy másik linux disztribúción. Ez azért van, mert a C++ gépi kódot fog előállítani fordításnál. Ezzel szemben a Java esetében egy saját virtuális géppel (JVM) fordítja a kódokat, és itt mutatkozik meg a hordozhatóság fogalma, hiszen, bármilyen rendszeren, ahol fent van a JVM, le fog futni a fordított kódunk.

Másik különbség a két nyelv között az, hogy a C++-ban főleg pointerekkel (mutatókkal) dolgozunk, addig Java-ban ez referenciaként lesz jelen. Tehát Java-ban nincsenek mutatók, de C++-ban van referencia és mutató is.

A két nyelv szintaktikában is elég erősen eltér. Nézzünk meg egy egyszerű kiíratást C++-ban:

```
std::cout << b.x << std::endl;
```

Java-ban:

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

## 11.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)

Személy szerint nagyon tetszik a Python nyelv, bár eddig is tudtam róla pár dolgot, hisz láttam már python kódot, de most az alapokba jobban belelátva nőtt a nyelv iránti szimpatiám...

Nagyon egyszerű, dinamikus és erősen magas szintű programozási nyelv. Nagyon tetszik a behúzásalapú szyntaxa a nyelvnek. Már-már annyira egyszerű, hogy ennél egyszerűbbet nem is nagyon tudok elkövetni, hogy az ne menjen a kód olvasásának és érthetőségének rovására. Szintén jó, hogy ezáltal sokkal tömörebben lekódolhatunk dolgokat, mint pl. Java-ban vagy C++-ban, és még olvasható is annak ellenére, hogy rövidebb a programkódunk.

Ami szintén erősen elnyerte a tetszésemet, hogy egy scriptnyelvben konkrétan ennyi minden el tudunk érni, ráadásul ugyanúgy futtatjuk kódjainkat, akár egy bash scriptet, nincs szükség linkelni/fordítani azokat.

A változóknak nincs konkrét típusa, ez szintén egy teljesen jó pont szerintem. Hiszen a nyelv "maga dönti el", hogy milyen típusú változót használ az adott értékre. Szerintem ez rövidtávon megszokható és komfortos élményt ad vissza.

Tök jó, hogy Python-ban többfélét is választhatunk szekvenciák, avagy gyűjtők között. Szekvenciák például a sztring, ennes és a lista. Aztán ott van még a szótár is, ami kulcsokkal azonosított elemek rendezetlen halmaza, ahol a kulcs bármilyen típusú lehet.

Pár példa az ennesekre (tuples):

Ennes (tuple) szyntaxa:

```
('a', 'b', 'c')      # ez egy három elemű ennes
tuple('abc')          # tuple generálás kulcsszóval
()                   # ez egy üres ennes
```

Lista szyntaxa:

```
['a', 'b', 'c']      # lista, hasonló a tuple-höz, csak itt []-t ←
                     # használunk
list('abc')          # itt szintén listát generálunk kulcsszóval
```

Szótár szyntaxa:

```
{'a':1, 'b':5, 'e':1982}    # így néz ki egy szótár
{}                          # ez pedig egy üres szótár
```

## 12. fejezet

# Helló, Arroway!

### 12.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.!

Megoldás forrása:

```
class PolarGen {  
  
    boolean stored = false;  
    double val;  
  
    public PolarGen() {  
        stored = false;  
    }  
  
    public double next() {  
        if ( ! stored) {  
            double u1, u2, v1, v2, w;  
            do {  
                u1 = Math.random();  
                u2 = Math.random();  
                v1 = 2 * u1 - 1;  
                v2 = 2 * u2 - 1;  
                w = v1 * v1 + v2 * v2;  
            } while (w > 1);  
            double r = Math.sqrt((-2 * Math.log(w)) / w);  
            val = r * v2;  
            stored = ! stored;  
            return r * v1;  
        } else {  
            stored = ! stored;  
        }  
    }  
}
```

```
        return val;
    }
}

public static void main(String[] args) {
    PolarGen g = new PolarGen();

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

A polártranszformációs generátor pszeudo-random számok generálására lett kitalálva. Itt kettő pszeudo-random szám jön létre a lefutás során, így elegendő az előzőleg generáltakból a másodikat visszaadni.

Ennek a feladatnak a megoldásához csak egy osztályt kell írnunk, amely képes kezelni a megadott képleteket. Nekünk szinte egyáltalán nincs szükségünk matematikai háttérre. Feladatunk: van egy matematikai feladat (amivel igázából nem sokat kell foglalkoznunk, mert csak bemásoljuk a képleteket), írunk hozzá egy osztályt, amely képes kezelni azt.

## 12.2. Homokozó, Binfa program elkészítése Java-ban

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden máris működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen).

Megoldás forrása:

```
class LZWBinFa {

    public LZWBinFa() {
        fa = gyoker;
    }

    public void egyBitFeldolg(char b) {
        if (b == '0') {

            if (fa.nullasGyermek() == null) {
                Csomopont uj = new Csomopont('0');
                fa.ujNullasGyermek(uj);
                fa = gyoker;
            } else {
                fa = fa.nullasGyermek();
            }
        } else {
    }}
```

```
        if (fa.egyesGyermek () == null) {
            Csomopont uj = new Csomopont ('1');
            fa.ujEgyesGyermek (uj);
            fa = gyoker;
        } else {
            fa = fa.egyesGyermek ();
        }

    }
}
```

A java nyelvnek van egy olyan tulajdonsága, hogy minden referencia és így nem kell vacillálnunk, hogy pointert vagy referenciát használunk. Ennek köszönhetően C++-os programunkból eltávolíthatjuk a pointereket jelző csillagot(\*), programunk ugyanúgy fog működni.

A fordítónk millió hibák jelzésének elkerülése érdekében még fordítás előtt érdemes a pontosvesszők eltávolítása az osztályok és metódusaik végéről. A fájlkezeléshez szükséges metódus és típuscserék után kész is a Java-s verzió.

Gyakorlatvezetőm ajánlásával Tomcat-et használtam saját szerver létrehozására a Servletbe való beépítéshez. [Ezen a linken](#) leírást kapsz arról, hogyan telepítsd a Tomcatet (Ubuntu esetében).

```
fa = asd
-----
-----1(2)
-----1(1)
-----0(2)
-----0(3)
---/(0)
-----1(3)
-----0(4)
-----1(2)
-----0(3)
-----0(1)
-----0(2)
depth = 4
mean = 2.8
var = 0.8366600265340756
```

Ubuntu linux screenshot

## 12.3. "Gagyi"

Az ismert formális `while (x <=t && x >=t && t !=x);` tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciaja) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás forrása:

```
import java.lang.Number;

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

        Integer st = 140, nd = 140;

        while (st <= nd && st >= nd && st != nd)
            System.out.println("running...");
    }
}
```

Az ismert formális tesztkérdéstípusban a következő láthatjuk:

Ha x kisebb vagy egyenlő mint t

és x nagyobb vagy egyenlő mint t

és t nem egyenlő x

Látható a változók deklarálásánál, hogy az adott érték 127-től nagyobb. Így a while ciklusunknál minden feltételünk igaz lesz. Azért fog mind a 3 feltételünk igazat visszakapni, mert más memóriacímük, annak ellenére, hogy egyeznek az értékeik. Tehát így értelemszerűen egy végtelen ciklusba fogunk torkollani, ami az alábbi képen látható is.

```
zschachi@zschachi-VirtualBox:~/java$ javac gagyi.java
zschachi@zschachi-VirtualBox:~/java$ java Gagyi
running...
```

Ubuntu linux screenshot

## 12.4. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-t leáll, ha nem követjük a Yoda conditions-t!

[https://en.wikipedia.org/wiki/Yoda\\_conditions](https://en.wikipedia.org/wiki/Yoda_conditions)

Megoldás forrása:

```
class Yoda {
    public static void main(String[] args) {
        int lightsaber = 2923; // 2FF923 is the green ←
                               lightsaber's color code
                               // so if we skip the ←
                               characters, we get ←
                               2923...
        if(2923 = lightsaber) {
            // compile error
        }
    }
}
```

A Yoda condition egy programozási stílus, elnevezését az ismert Star Wars szereplője, Yoda nem-szabványos beszéde adta. Ez programunkban úgy észlelhető, hogy például egy if-ben a feltételünk bal oldalára írjuk az értéket. (ld. megoldás forrásánál)

A fenti kódcsipetemben egy pici "easter egg"-et is elhelyeztem, ez pedig a zöld fénykardnak a színkódja, hiszen Yodának is zöld színű a fénykardja, csak betűk nélkül, tehát 2FF923 => 2923.

Ahogy fent is írtam, Yoda furcsa, nem szabványos beszéde miatt kapta erről a nevét. Hiszen ez is ilyen nem szabványos módszer, amire persze fordításkor hibát is kapunk.

## 12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbpalg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását!

Megoldás forrása:

```
public class PiBBPBench {
    public static double d16Sj(int d, int j) {

        double d16Sj = 0.0d;

        for(int k=0; k<=d; ++k)
            d16Sj += (double)n16modk(d-k, 8*k + j) / ( ←
                double)(8*k + j);

        /* (bekapcsolva a sorozat elején az első utáni ←
           jegyekben növeli pl.
           a pontosságot.)
        for(int k=d+1; k<=2*d; ++k)
            d16Sj += Math.pow(16.0d, d-k) / (double)(8*k + ←
                j);
    }

    return d16Sj - Math.floor(d16Sj);
}

public static long n16modk(int n, int k) {

    int t = 1;
    while(t <= n)
        t *= 2;

    long r = 1;

    while(true) {

        if(n >= t) {
            r = (16*r) % k;
            n = n - t;
        }

        t = t/2;

        if(t < 1)
            break;
    }
}
```

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

}

return r;
}
public static void main(String args[]) {

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
    double d16S6t = 0.0d;

    int jegy = 0;

    long delta = System.currentTimeMillis();

    for(int d=100000000; d<100000001; ++d) {

        d16Pi = 0.0d;

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

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

        d16Pi = d16Pi - Math.floor(d16Pi);

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

    }

    System.out.println(jegy);
    delta = System.currentTimeMillis() - delta;
    System.out.println(delta/1000.0);
}
}
```

Ebben a példában Bátfai Tanár Úr kódját használtam, időhiány miatt.

A BBP rövidítés a "Bailey-Borwein-Plouffe"-ból ered, ami egy olyan algoritmus amely igen megközelítőleg kiszámítja nekünk a pi értékét. 1995-ben lett felfedezve, 1996-ban publikálva.

A BBP algoritmus a pi értékének, valamint n. jegyeinek gyors kiszámítására született algoritmus. Az első

programmagát a Pí szám értékét fogja kiszámítani, míg a második képes lesz az n. helyiértéken álló szám hexadecimális értékének megmondására.

Tetszőleges kiindulási helyzetből indulva bináris és hexadecimális számjegyeit számolja ki Pi-nek.

Eme tudományos közlemény segít megérteni a BBP algoritmus működését, és ennek segítségével sokkal gördülékenyebben megírhatjuk programunkat.

```
public static double d16Sj(int d, int j) {  
  
    double d16Sj = 0.0d;  
  
    for(int k=0; k<=d; ++k)  
        d16Sj += (double)n16modk(d-k, 8*k + j) / (double) ←  
                  (8*k + j);  
  
    /* (bekapcsolva a sorozat elején az első utáni ←  
     jegyekben növeli pl.  
     a pontosságot.)  
    for(int k=d+1; k<=2*d; ++k)  
        d16Sj += Math.pow(16.0d, d-k) / (double) (8*k + j);  
    */  
  
    return d16Sj - Math.floor(d16Sj);  
}
```

Programunkban a végső számítást a fent látható d16Sj nevezetű függvény fogja végezni, amely 2 bemenetet vár. Ez a függvényünk gyakorlatilag az alábbi képlet alapján számol, amit a fenti linken láthatunk részletesen kielemezve:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right).$$

*Fedora linux screenshot*

## 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.

**A feladat megoldásában Szilágyi Csaba tutorált!**

Megoldás forrása:

```
class liskov_example{

    public static void main (String[] args){

        Program program = new Program();
        Hal hal = new Hal();
        program.vizsgal(hal);

        FeherCapa fehercapa = new FeherCapa();
        program.vizsgal(fehercapa);

        Balna balna = new Balna();
        program.vizsgal(balna);
    }

    class Hal{
        public
            void uszik(){
                System.out.println("hal");
            }
    }

    class Program{
        public
            void vizsgal(Hal hal){
```

```
        hal.uszik();
    }
}

class FeherCapa extends Hal{}

class Balna extends Hal{}
```

A liskov-helyettesítési elvet Robert C. Martin alkotta meg. Része a S.O.L.I.D alapelveknek.

A liskov elv azt mondja ki, hogy ha P altípusa K-nak, és mindenhol, ahol K-t felhasználjuk, ugyanúgy, P-t is fel tudjuk használni anélkül, hogy a program másképpen viselkedne.

A fenti feladat jól demonstrálja a helyzetet. Az a kontextus, hogy attól még, hogy valami tud úszni nem biztos, hogy hal. A lenti képen pedig láthatjuk, hogy fordul és fut a programunk.

A Liskov-elv ott lesz megsértve, hogy ugyebár a cápa és a bálna is úsznak, de nem mindkettő hal lesz, ugyanis amíg a cápa a halak osztályába, a bálna az emlősökébe tartozik.

```
[salesz@prdtr java_practice]$ javac liskov_example.java
[salesz@prdtr java_practice]$ java liskov_example
hal
hal
hal
[salesz@prdtr java practice]$ █
```

*Ubuntu linux screenshot*

## 13.2. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek!

Megoldás forrása:

```
public class ParentChild {
    public static void main(String[] args) {
        Parent parent = new Child();
        parent.output();
    }
}

class Parent {}

class Child extends Parent{
    public void output() {
        System.out.println("Child");
    }
}
```

```
}
```

A fenti kód demonstrálja, hogyan is nézne ki egy szülő-gyerek kompozíció java-ban.

Mint láthatjuk, van egy üres szülő osztályunk, aztán lentebb a belőle származtatott gyerek osztály. Ott egy metódusban kiíratunk valamit. Aztán a main függvényünkben szülő típusú új gyerek objektumot hozunk létre. Aztán a szülő (osztály) típusú "változónkon" keresztül íratjuk ki a metódusunkat.

Itt tehát a polimorfizmust használjuk a szülő osztállyal való új gyerek osztály típusú objektum létrehozására. Tehát a szülő nem képes meghívni gyermeké metódusát, amit ő nem definiált...

### 13.3. Anti OO

A BBP algoritmussal<sup>4</sup> a Pi hexadecimális kifejtésének a 0. pozíciótól számított  $10^6$ ,  $10^7$ ,  $10^8$  darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu-hu/tartalom/tkt/javat-tanitok-javat/apas03.html#id561066>

Megoldás forrása:

```
for(int d=10000000; d<10000001; ++d) {  
  
    d16Pi = 0.0d;  
  
    d16S1t = d16Sj(d, 1);  
    d16S4t = d16Sj(d, 4);  
    d16S5t = d16Sj(d, 5);  
    d16S6t = d16Sj(d, 6);  
  
    d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - ←  
            d16S6t;  
  
    d16Pi = d16Pi - Math.floor(d16Pi);  
  
    jegy = (int)Math.floor(16.0d*d16Pi);  
  
}
```

Ebben a feladatban teszteltük az előző fejezetünkön a PiBBP kódunk futási idejét a 10 hatodik, hetedik és nyolcadik hatványaival.

A fenti forrásba a módosítandó kódrészletet tettem be. Tehát ebben a for ciklusban éppen 10 nyolcadik hatványával futtatjuk.

Nyolcadik hatvánnyal az én esetemben 205.641 mp alatt futott le a program. Alább beillesztettem egy képet a 3 eset futásáról java esetében.

```
[salesz@prdtr java_practice]$ javac PiBBPBench.java
[salesz@prdtr java_practice]$ java PiBBPBench
6
1.527
[salesz@prdtr java_practice]$ javac PiBBPBench.java
[salesz@prdtr java_practice]$ java PiBBPBench
7
18.065
[salesz@prdtr java_practice]$ javac PiBBPBench.java
[salesz@prdtr java_practice]$ java PiBBPBench
12
205.641
[salesz@prdtr java_practice]$ █
```

*Ubuntu linux screenshot*

### 13.4. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans-SMNISTforHumansExp3/app/src/main> Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás forrása:

```
int [] bgColor =
{
    android.graphics.Color.rgb(0, 85, 128),
    android.graphics.Color.rgb(255, 255, 102)
};
```

Ezt a feladatot android studio-ban oldottam meg. Nem saját telefont, hanem szimuláltam a program kipróbálására.

A fenti kódrészletben láthatjuk azt a részt, ami a háttér színéért felel. Azért szerepel kétszer, mivel programunkban váltakoznak a színek. Így kétféle színt beállítunk, aztán az fog váltakozni.

Az alábbi programkód-részben a gombok, betűk színét, annak háttereit tudjuk megváltoztatni.

A setColor értelemszerűen a színét változtatja, ez esetben konkrét szín nevet adunk meg. (pl. BLUE, GRAY, RED, YELLOW stb.)

A setStyle a stílusát adja meg, azaz, hogy pl FILL\_AND\_STROKE - azaz legyen kitöltése és körvonala.

A setTextAlign maga a szöveg igazításáért felel (pl. CENTER, LEFT, RIGHT stb.)

A textSize pedig a szöveg méretéért, amit konkrét számmal adunk meg.

```
textPaint.setColor(android.graphics.Color.GRAY);
textPaint.setStyle(android.graphics.Paint.Style. ←
    FILL_AND_STROKE);
textPaint.setAntiAlias(true);
textPaint.setTextAlign(android.graphics.Paint.Align.CENTER) ←
;
textPaint.setTextSize(55);

msgPaint.setColor(android.graphics.Color.GRAY);
msgPaint.setStyle(android.graphics.Paint.Style. ←
    FILL_AND_STROKE);
msgPaint.setAntiAlias(true);
msgPaint.setTextAlign(android.graphics.Paint.Align.LEFT);
msgPaint.setTextSize(40);

dotPaint.setColor(android.graphics.Color.BLUE);
dotPaint.setStyle(android.graphics.Paint.Style. ←
    FILL_AND_STROKE);
dotPaint.setAntiAlias(true);
dotPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
dotPaint.setTextSize(50);

borderPaint.setStrokeWidth(2);
borderPaint.setColor(android.graphics.Color.RED);
fillPaint.setStyle(android.graphics.Paint.Style.FILL);
fillPaint.setColor(android.graphics.Color.BLUE);
```

## 13.5. 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](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf) (77-79 fóliát)!

Megoldás forrása:

```
class cyclomatic {
    public static void main(String args[]) {
        int A = 2, B = 5, C = 7;
        if (B > C) A = B;
        else A = C;

        System.out.println(A);
        System.out.println(B);
        System.out.println(C);
    }
}
```

A ciklomatikus komplexitás a forráskódot jellemző tulajdonság, azt mondja meg, hogy egy adott programban hány lineárisan független út található egy programban. Lefordítva, hogy hány féle képpen lehet a vezérlés a program kezdetétől a végéig.

A ciklomatikus komplexitást Thomas J. McCabe alkotta. Emiatt nevéről, McCabe-komplexitásnak is nevezik. Az alkotás maga egy szoftvermetrika, amely egy adott forráskód alapján meghatározza a forráskód komplexitását egy konkrét számértékben. Gráfelmélet alapján számít.

Az érték kiszámítása a következő módon történik:

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

Ahol E: a gráf éleinek száma, N: csúcsok száma, P: az összefüggő komponensek.

A képlet alapján, így, hogy tudjuk mi micsoda, mostmár mi is egyszerűen ki tudjuk számítani egy adott programkód komplexitását.

A következő példát [innen](#) vettem.

Ahogy a linken is láthatjuk, esetünkben 8 csúcsunk, 7 élünk és 2\*1 a komponens. Tehát:  $8 - 7 + 2 = 3$

## 14. fejezet

# Helló, Mandelbrot!

### 14.1. Reverse engineering UML osztálydiagram

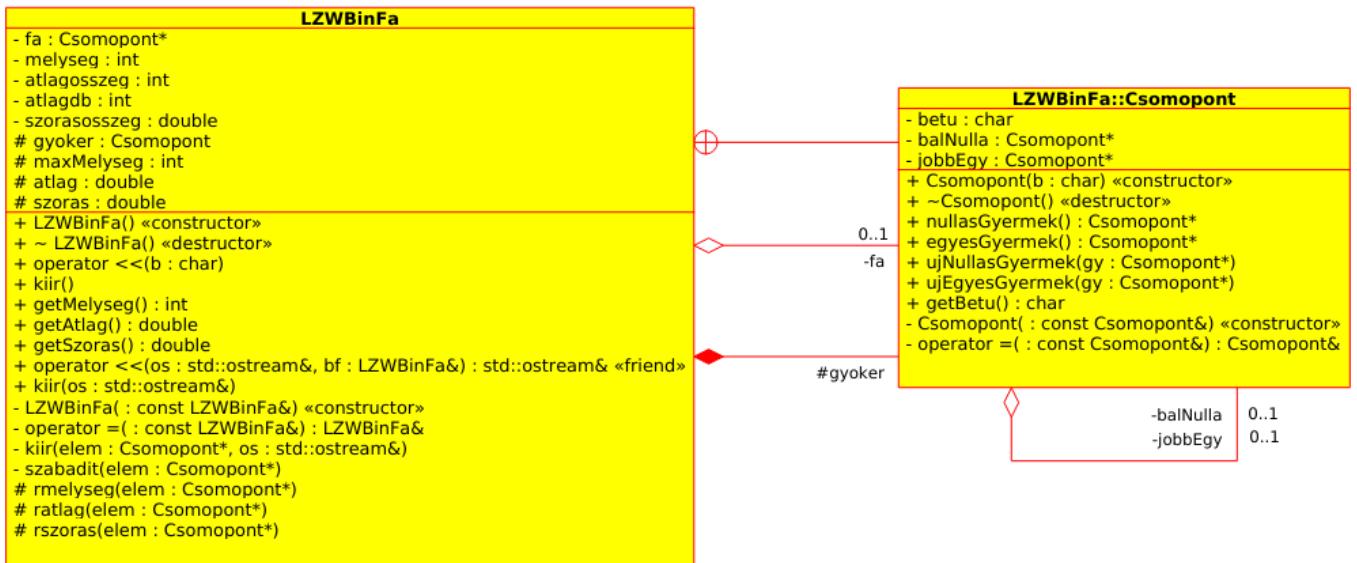
UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: [https://youtu.be/Td\\_nlERIEOs](https://youtu.be/Td_nlERIEOs).

Megoldás forrása:

Az UML a Unified Modeling Language rövidítése. Ahogy a nevében is láthatjuk, ez egy szabványos modellező nyelv. Szöveges és grafikus modelleket tudunk vele készíteni. Sokféle folyamatokról készíthetünk UML modellt, ugyanakkor szoftverekről és programokról, sőt még adatbázisokról is készíthetünk. Bizonyos cégeknél ez egy kifejezetten hatékony kommunikációs eszközként van jelen. Segíti az együttműköést, bizonyos dolgok felvázolását, könyebbé téve annak megérthetőségét.

Az UML specifikációja viszonylag nagy terjedelmű. Az osztálydiagrammok jelölésének rendszere a következőképp néz ki: van egy három részre bontott téglalapunk, a legfelső részben az osztály neve van. A következő részben a mezők, majd alattuk a metódusok találhatók. A publikus adattagok jelölése a +, a priváté a -, a protected láthatóságú adattagokat pedig a # azonosítja.

Ebben a feladatban a jó öreg binfa programunkat fogjuk lemodellezni UML-ben. Ehhez az Eclipse UML-t fogom használni, amely képes osztálydiagramot generálni programkódunkból.



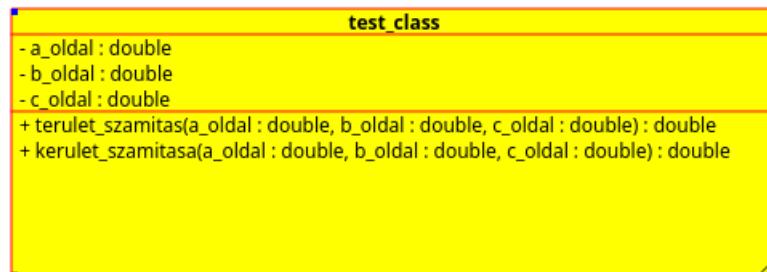
Fedora linux screenshot

## 14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Ebben a feladatban csakúgy, mint az előzőben is az Umbrello nevezetű programot használtam.

Ugyanaz a felállás, csakmost egy kisebb, saját elképzelt programkódból hozzuk létre modellünket. Esetemben ahogyan a modellből is kivehető, egy háromszög kerület/terület programról van szó. Létrehoztam 3 változót, értelemszerűen a háromszög 3 oldala lesz ez (a, b, c), aztán két külön függvényt, egyik kerületet, másik területet számol.



Fedora linux screenshot

Bár nem egy nagy durranás, de azért itt hagyom a generált forráskódöt is:

```

[
  class test_class {

    public static double terulet_szamitas(double a, double b, ←
      double c) {
      return a*b*c;
    }
  }
]
  
```

```
public static double kerulet_szamitas(double a, double b, ←
    double c) {
    return a+b+c;
}
public static void main(String[] args) {
    double a_oldal = 3;
    double b_oldal = 4;
    double c_oldal = 5;

    terulet_szamitas(a_oldal, b_oldal, c_oldal);
    kerulet_szamitas(a_oldal, b_oldal, c_oldal);
}
}
```

### 14.3. Egy esettan

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

Megoldás forrása:

```
//File: product.h
#ifndef PRODUCT_H
#define PRODUCT_H

#include <iostream>
#include <ctime>

class Product
{
protected:
    int initialPrice; // Beszerzési ár
    time_t dateOfAcquisition; // Beszerzés dátuma
    std::string name; // Név
    virtual void printParams(std::ostream& os) const;
    virtual void loadParamsFromStream(std::istream& is) ←
    ;
    virtual void writeParamsToStream(std::ostream& os) ←
        const;
public:
    Product();
    Product(std::string name, int initialPrice, time_t ←
            dateOfAcquisition);
    virtual ~Product(); {}
    int GetInitialPrice const;
    std::string GetName() const;
    time_t GetDateOfAcquisition() const;
    int GetAge() const;
```

```
virtual int GetCurrentPrice() const;
void Print(std::ostream& os) const;
virtual std::string GetType() const = 0;
virtual char GetCharCode() const = 0;
friend std::istream& operator>>(std::istream& is, ←
    Product& product);
friend std::ostream& operator>>(std::ostream& os, ←
    const Product& product);
};

#endif /* PRODUCT_H */
```

A könyv 14. fejezetében egy esettanulmány során kaphattunk betekintést a C++ különböző nyelvi elemeibe, melynek fő témaja az öröklés és a virtuális függvények megfelelő alkalmazása.

Fejezetünk első részében egy számítógép-alkatrész és konfiguráció értékesítő kereskedésnek készítünk egy olyan alkalmazást, amely lehetőséget biztosít eme eszközök nyilvántartására, persze ezt úgy oldjuk meg, hogy később könnyen módosítható legyen új termékcsaládok bevezetésére is. Tehát konkrétan egy keretrendszer hozunk létre erre a célra.

A termékek reprezentálására egy Product nevű osztályt tartunk fent. Ebbe van az összes termékre vonatkozó közös kódunk. Tehát a keretrendszer a Product osztályon alapul.

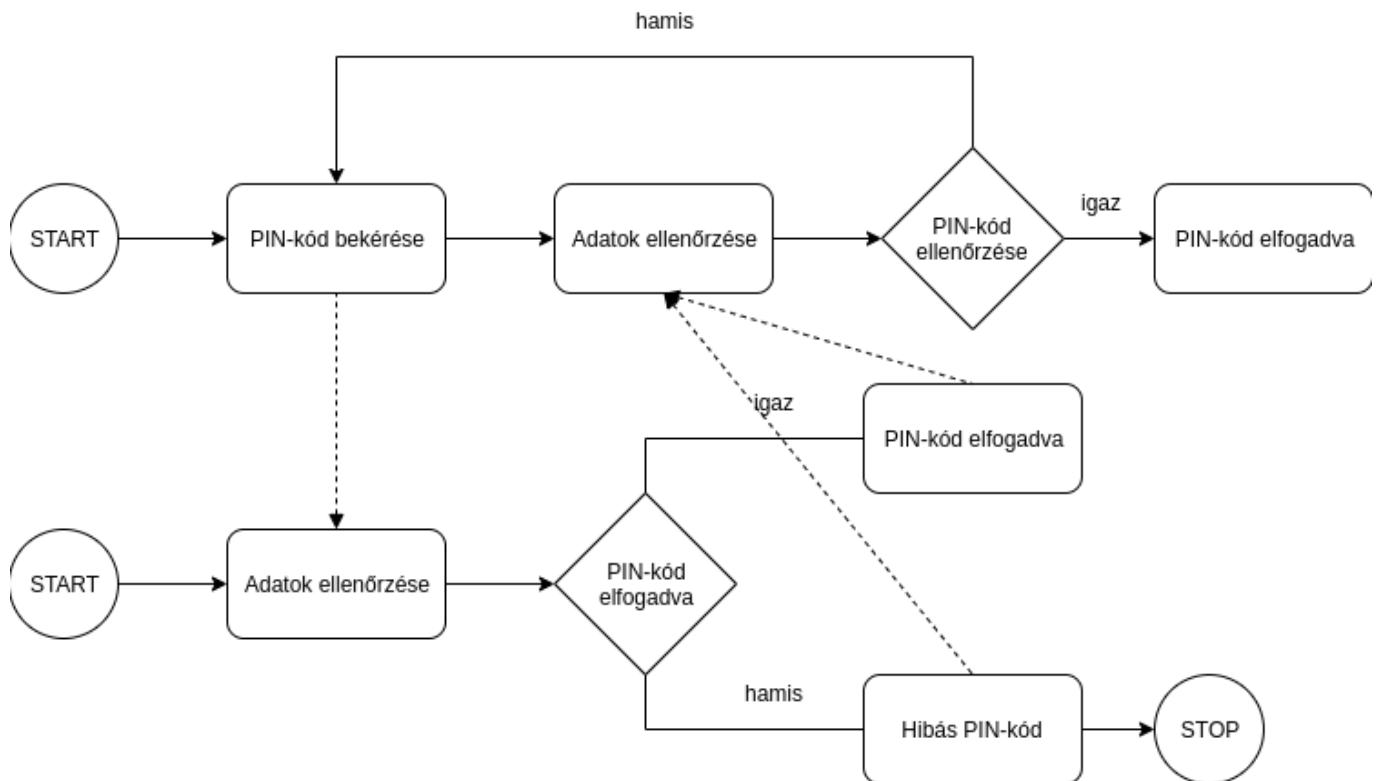
## 14.4. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated-/Prog2\\_7.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated-/Prog2_7.pdf) (34-47 fólia)

**A feladat megoldásában Győri Márk Patrik tutorált!**

Ebben a feladatban egy valós eseményt kellett lealgoritmizálnunk, ha úgy tetszik folyamatábrával ábrázolnunk.

Személy szerint már középiskolában is volt szerencsém folyamatábrázni. Nagyon szerettem. Jelen esetben egy bankkártyás elfogadórendszer egy kis részét modelleztem le. Ehhez a Google draw.io felületét használtam. Azért erre esett a választás, mert ennek kezelése tünt számomra a legkézenállóbbnak. A végeredményt lásd a képen:



*Fedora linux screenshot*

# 15. fejezet

## Helló, Chomsky!

### 15.1. Encoding

Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Alapértelmezetten a fordítónk utf-8 encodinggal próbál fordítani, ez esetben nem jár sikerrel. Ugyanis ebben nem találhatóak meg bizonyos magyar ékezes betűk, mint például az í,ő,ö,ü betűk. Tehát ha szimplán fordítjuk (javac Mandelbrot.java) ezt a kimenetet kapjuk:

```
MandelbrotHalmaz.java:44: error: unmappable character for encoding utf-8
    int szólesség, int iterációsHatór) {
                           ^
MandelbrotHalmaz.java:44: error: unmappable character for encoding utf-8
    int szólesség, int iterációsHatór) {
                           ^
MandelbrotHalmaz.java:49: error: unmappable character for encoding utf-8
    this.szólesség = szólesség;
                           ^
MandelbrotHalmaz.java:49: error: unmappable character for encoding utf-8
    this.szólesség = szólesség;
                           ^
100 errors
```

*Fedora linux screenshot*

Ha azt szeretnénk, hogy a fent említett ékezes karakterek is el legyenek fogadva egy másik kódolással kell próbálkoznunk. Például a cp1250 tartalmazza a fent említett karaktereket. Ezesetben: javac -encoding cp1250 Mandelbrot.java és márás hiba nélkül fordul a kódunk.

```
[salesz@localhost java_practice]$ javac -encoding Cp1250 MandelbrotHalmaz.java
[salesz@localhost java_practice]$
```

*Fedora linux screenshot*

## 15.2. OOCWC lexer

Izzítsuk be az OOCWC-t és vázoljuk a <https://github.com/nbatfai/robocareemulator/blob/master/justine/rceemu/src/> lexert és kapcsolását a programunk OO struktúrájába!

Megoldás forrása:

```
%option c++
%option noyywrap

%{
#define YY_DECL int justine::robocar::CarLexer::yylex()
#include "carlexer.hpp"
#include <cstdio>
#include <limits>
%}

INIT "<init"
INITG "<init guided"
WS [ \t]*
WORD [^-:\n \t ()]{2,}
INT [0123456789]+
FLOAT [-.0123456789]+
ROUTE "<route"
CAR "<car"
POS "<pos"
GANGSTERS "<gangsters"
STAT "<stat"
DISP "<disp>"
%%
```

Lexerekkel találkozhattunk már a könyv 1. fejezetében is, viszont itt egy kis rövid emlékeztetés a lexerekről:

A lexerek feladata, ahogy a nevéből (lexikális/nyelvi) kiolvasható, szövegek elemzése, pontosabban olyan programok előállítása, amely adott szabályok mellett, képes feldolgozni szöveges állományokat.

A lexerek, ahogy a nevéből is láthatjuk, szövegek elemzésére, adott szabályok mellett, képes feldolgozni szöveges állományokat.

A fenti kód első része a definíciókból áll. Láthatjuk, hogy kódunk egy Carlexer osztályt fog létrehozni. Ez lesz maga a szövegelemző.

Ez utáni sorban már egy c++ kódcsipetet láthatunk, amely beincludolja nekünk a carlexer.hpp fájlt.

Ezután konstansokat hozunk létre, hogy ne legyen túl komplikálnak tűnő a kód, leolvasható, hogy valamilyen xml fájl lesz majd belőle, lesz feldolgozva.

```
{DISP}          {
                           m_cmd = 0;
}
```

```
{POS} {WS} {INT} {WS} {INT} {WS} {INT}    {
    std::sscanf(yytext, "<pos ←
                  %d %u %u", &m_id, &←
                  from, &to);
    m_cmd = 10001;
}

{CAR} {WS} {INT}    {
    std::sscanf(yytext, "<car ←
                  %d", &m_id);
    m_cmd = 1001;
}

{STAT} {WS} {INT}    {
    std::sscanf(yytext, "< ←
                  stat %d", &m_id);
    m_cmd = 1003;
}

{GANGSTERS} {WS} {INT}    {
    std::sscanf(yytext, "< ←
                  gangsters %d", &m_id);
    m_cmd = 1002;
}

{ROUTE} {WS} {INT} {WS} {INT} ({WS} {INT}) * {
    int size{0};
    int ss{0};
    int sn{0};

    std::sscanf(yytext, "<route %d %d ←
                  %n", &size, &m_id, &sn);
    ss += sn;
    for(int i{0}; i<size; ++i)
    {
        unsigned int u{0u};
        std::sscanf(yytext+ss, "%u%n", ←
                    &u, &sn);
        route.push_back(u);
        ss += sn;
    }
    m_cmd = 101;
}

{INIT} {WS} {WORD} {WS} ("c"|"g")  {
    std::sscanf(yytext, "<init %s %c> ←
                  ", name, &role);
    num = 1;
    m_cmd = 0;
}

{INIT} {WS} {WORD} {WS} {INT} {WS} ("c"|"g")  {
    std::sscanf(yytext, "<init %s %d ←
                  %c>", name, &num, &role);
    if(num >200)
    {
```

```
        m_errnumber = 1;
        num = 200;
    }
    m_cmd = 1;
}
{ INITG} {WS} {WORD} {WS} ("c" | "g") {
    std::sscanf(yytext, "<init guided <
                  %s %c>", name, &role);
    num = 1;
    m_guided = true;
    m_cmd = 3;
}
{ INITG} {WS} {WORD} {WS} {INT} {WS} ("c" | "g") {
    std::sscanf(yytext, "<init guided <
                  %s %d %c>", name, &num, &role <
                  );
    if(num >200)
    {
        m_errnumber = 1;
        num = 200;
    }
    m_guided = true;
    m_cmd = 2;
}
.
{ ; }
%%
```

A modulok előtt (blokk: lásd pl DISP) láthatjuk a korábban említett reguláris kifejezéseket, ha megegyeznek a minták a vizsgált bemenettel, akkor az adott modulban található utasítás fog végrehajtódni.

Tulajdonképpen a fent látható forráskódban szabályokat láthatunk. Ezek azért vannak, hogy a kapott információk után felhasználják, és miután kielezte, különböző típusokban hozzon létre valamilyen altípust.

Végül maga a docker futtatásához Molnár Antal dockerét használtam fel. Így könnyedén, sok időt spórolva beizzíthatjuk a projektet. A környezet a következőképp néz ki:



Fedora linux screenshot

Aztán, mint a következő képen látható, futtatásra tudjuk bírni a projektet:

```
root@4a35b26b5600: ~/Desktop/justine/rceemu
File Edit Tabs Help
checking for osmium/osm/relation.hpp... yes
checking for osmium/io/any_input.hpp... yes
checking for library containing shm_open... -lrt
checking for shm_open... yes
checking for flex... flex
checking lex output file root... lex.yy
checking lex library... none needed
checking whether yytext is a pointer... no
checking for flex... flex
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: executing depfiles commands
config.status: executing libtool commands
root@4a35b26b5600:~/Desktop/justine/rceemu# src/smartycity --osm=../debrecent.osm -
-city=Debrecen --shm=JustineSharedMemory
Robocar City Emulator and Robocar World Championship, City Server
Copyright (C) 2014, 2015 Norbert Bőtfai
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Debrecen is... ready.
```

Fedora linux screenshot

## 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.

Megoldás forrása:

```
glColor3f(0.818f,.900f,0.824f);
glColor3f(0.818f,.900f,0.824f);
glColor3f(0.818f,.900f,0.824f);
glColor3f(0.818f,.900f,0.824f);
glColor3f(.188f,0.209f,0.190f);
glColor3f(.82f,.15f,.15f);
glColor3f(.188f,0.209f,0.190f);
glColor3f(0.15f,.29f,.82f);
```

A Paszigráfia Rapszódia egy olyan mesterséges nyelv kialakítására törekvő kezdeményezés, mely lehetővé teszi a homunkulusz és a mesterséges homunkulusz közötti kommunikációt, ergo ez az esport kultúra nyelve.

Ahhoz, hogy programunk színvilágát módosítsuk, meg kell keresnünk a kódban a glColor3f(); sorokat. Amelynek paraméterként RGB formátumban vannak megadva a színek, tehát első számunk a piros, második a zöld, végül pedig a kék szín arányát láthatjuk. Lásd fent, megoldás forrásánál.

```
glColor3f(0.818f,.900f,0.824f);
```

## 15.4. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! Lásd <https://youtu.be/XpBnR31BRJY>

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

using namespace std;
using namespace png;

int main ( int argc, char *argv[] )
{
    image <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);
cout << " " << value << endl;

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

### A feladat megoldásában Győri Márk Patrik tutorált!

A gépi tanulásban a perceptron egy algoritmus, a bináris osztályozás tanítására. A bináris osztályozás azt jelenti, hogy az osztályozandó elemeket két részre osztjuk pl.: vagy benne van valami egy adott osztályban vagy nincs. A következő programban ilyen algoritmust fogunk használni.

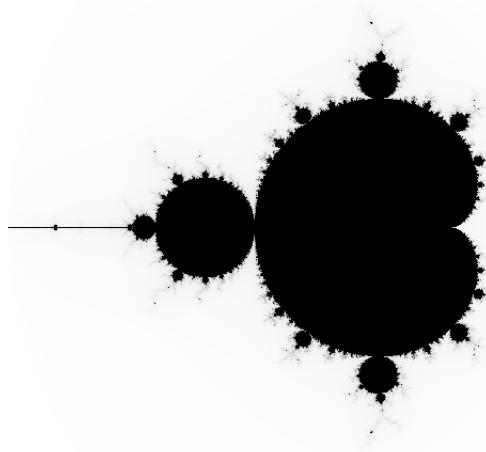
A feladat elkészítéséhez szükségünk lesz értelemszerűen a perceptron osztályra, ami az nlp.hpp fájlban az első osztály. Ezt át kell tennünk egy másik header fájlba, ugyanis most csak azt fogjuk használni.

Feladat megoldásához szükségünk lesz a libpng12 és ennek dev változatára, illetve a png++ könyvtárra. A png++-t forráskódóból fel tudjuk telepíteni. Például innen tölthetjük le: <http://download.savannah.nongnu.org/releases/pngpp/>

```
[salesz@localhost java_practice]$ g++ ql.hpp perceptron.cpp -o perc -lpng -std=c++11
[salesz@localhost java_practice]$ ./perc kimenet.png
[salesz@localhost java_practice]$
```

*Fedora linux screenshot*

A program futtatásakor az előző Perceptron programunk által generált kimenet.png-t használom fel. Ami az alábbi kép:



*Fedora linux screenshot*

Mint láthatjuk a programunk így is megfelelően működik. A kimenet.png fájlt a korábbi mandelbrot, szintén c++ programunkkal generáltam le.

# 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)!

Ebben a feladatban szükségünk lesz értelemszerűen a JDK-ra. Én a jdk-11-et telepítettem fel hozzá. Miután feltelepítettük a jdk mappájában lesz egy src.zip nevű állományunk, ezt kell kicsomagolnunk valahová. Konkrétan itt egy könyvtárbejárásról beszélünk. Tehát megszámoljuk, hogy hány darab .java fájlunk van a kibontott mappánkban lévő könyvtárakban.

```
int main()
{
    string pa = "/home/salesz/Downloads/src";
    path apk_path(pa);
    long long ennyimVan = 0;
    recursive_directory_iterator end;
    for (recursive_directory_iterator i(apk_path); i != end; ++i)
    {
        const path cp = (*i);
        string name = cp.string();
        vector<string> strs;
        split(strs, name, is_any_of("\\\\"));
        string fName = strs[strs.size() - 1];

        if (fName.compare("module-info.java") == -1 && fName.compare("package-info.java") == -1) {
            if (ends_with(fName, ".java")) {
                cout << " " << repeat(strs.size() - 7, "+") << " " << fName.substr(0, fName.size() - 5) << endl;
                ++ennyimVan;
            }
        }
    }
}
```

```

        cout << " | " << fName << " >" << endl;
    }
}

cout << "JDK -> Ennyi osztalyom van: " << ennyimVan << endl <-
;
}

```

### A feladathoz Győri Márk Patrik forráskódját használtam fel.

A forráskódban először is egy string változóban eltároljuk az elérési útunkat. Esetünkben ez a /home/salesz/Downl... A long long típusú számláló minden növekedik egygyel. A mappa bejárását a recursive\_directory\_iterator boost osztály végzi. Ez az adott mappánk összes fájlján végig megy.

## 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 értékadásra!

```
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 gyermekek-et, és az új fa-ba másolja azokat.

```
LZWBInFa (const LZWBInFa& regi)
{
    std::cout << "LZWBInFa masolo ertekadas" << 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 konstruktoros megoldásra, csak annyi a különbség, hogy a másolás előtt felszabadítja az egyes- és nullásgyermek-et, 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 gyermekei-t a régi gyökér nullás és egyes gyermekeire állítjuk be, majd a regi fa gyokerének gyermekeit 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.

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.

```

        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".

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.

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

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.

## 16.4. Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

A perceptron eddig csak egy értéket adott vissza, tehát ahhoz hogy egy úgymond "képet" kapunk vissza, meg kell babrálunk egy kicsit azt az osztályt. Ehhez 3 dolgot kell megváltoztani. Először gyorsan átírjuk az inicializálásnál a konstruktorban az utolsó argumentumot 1-ről size-ra, ami a kép területe.

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

Ezután jön a lényeg, amit a zárójelek túlterhelésénél találunk, ugyanis így adjuk át a kép értékeit a perceptronnak (lásd: `double* value = (*p) (image);`). A visszatérési értéket `double*-ra` állítjuk, tehát mostmár az eredményünk ugyanolyan típusú mint ahogy kaptuk. De ez még nem elég, ugyanis eddig csupán az utolsó réteg első elemét adtuk vissza egy kicsit átszámolva, ami nekünk kevés egy kép megalkotásához. Ezért csak simán kitöröljük a függvényhívást és a második "[" zárójeleket és visszakapjuk az utolsó réteget teljes egészében.

```

double* operator() ( double image [] ) // ----- valtozas itt
{
    units[0] = image;
    for ( int i {1}; i < n_layers; ++i )
    {
        #ifdef CUDA_PRCPS
        cuda_layer ( i, n_units, units, weights );

```

```
#else
#pragma omp parallel for
for ( int j = 0; j < n_units[i]; ++j )
{
    units[i][j] = 0.0;
    for ( int k = 0; k < n_units[i-1]; ++k )
    {
        units[i][j] += weights[i-1][j][k] * units[i-1][k];
    }
    units[i][j] = sigmoid ( units[i][j] );
}
#endif
}
return units[n_layers - 1];//sigmoid ( units[n_layers - 1][0] ) ←
; ----- es itt
}
```

Összegezve: a mandelbrot halma ról készített kép vörös (red) értékeit átszámoltuk a perceptron osztállyal és most visszaírjuk azokat egy újonnan létrehozott kép (png\_image2) red értékeibe. Ha az új kép kék és zöld értékeit az eredeti kép értékeivel pótoljuk, akkor annak egy elszínezett változatát kapjuk.

Ha viszont az előző eljárást alkalmazzuk mind a 3 színre, akkor először egy szinte teljesen fekete képet kapunk. Ez annak a következménye, hogy a perceptron által visszaadott értékek 0-nál kisebbek. De ha beszorozzuk őket 100-al, akkor egy értékes kép rajzolódik ki.

## 16.5. Egyszerű string osztály

Egy egyszerű string osztály elkészítését (hogy ne a verembe tegyük a kockászárójel megvalósítását, mert az nyilván odacsap az osztály értelmezésének. A feladat leírása: [itt](#) található.

A feladatban Borviz Róbert tutorált.

A feladathoz a fenti posztban lévő **Borviz Róbert megoldását használtam**.

Egyszerű string class, egy heapen foglalt memóriaterületre mutató pointert tartalmaz.

```
char* data;
```

Mint láthatjuk, ez egy karakter pointer típusú privát adattag, ez a memóriaterület, ami a string számára van, első karakterére fog mutatni.

Itt pedig egy egyszerű operátort láthatunk a privát adattag elérésére:

```
ostream & operator<< (ostream & os, const mystring & x)
{
    return os << x.Data ();
}
```

Itt látható, hogy az output operátorunkat túlterheljük:

```
char operator[](int i) {
    size_t size = strlen(data)-1;
    if( i > size ) {
        cout << "Out of range" << endl;
    }
    char& ref = data[i];
    return ref;
}
```

Az [] operátor túlterhelése, először hibakezelés (azaz out of range figyelése), ha nagyobb indexű értékhez szeretnénk hozzáérni mint amelyet ténylegesen tartalmaz a data pointer akkor, hibát jelzünk, ellenkező esetben visszaadunk egy referenciát ami a megfelelő indexű karakterre mutat. Ezen a referenciaján keresztül tudjuk állítani az i-edik elem értékét.

Ez itt látható:

```
mystring b ("almafa");
b[2] = 'v';
```

Az operátor túlterhelésénél, azért a const kulcsszót használjuk, mert konstans objektumokra szeretnénk meghívni. Itt is először hibakezelünk. Maga az operátor pedig az i-edik elem értékéről ad másolatot, amit nem módosíthatunk, mivel konstans.

# 17. fejezet

## Helló, Gödel!

### 17.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban <https://youtu.be/DL6iQwPx1Yw> (8:05-től)

A Robotautó Világbajnokság projekt a városi forgalomirányító algoritmusok tesztelésére, illetve az okos városok és a robotautók között fennálló kapcsolat vizsgálatára jött létre.

A mi feladatunk a lambda kifejezések megértése, mely hasznunkra fog válni az elkövetkezendő feladatok során is.

A lambda kifejezések lehetőséget nyújtanak számunkra, hogy létrehozzunk egy névtelen függvényt, vagyis egy olyan konstrukciót, amely úgy viselkedik, mint egy függvény.

Ha általánosan kellene megfogalmazni: bárhol, ahol függvénymutatót használunk, használhatunk lambda kifejezéseket. Azaz akkor használunk lambda kifejezéseket, ahol másképpen függvénymutatókat használnánk.

Például az `std::sort` STL függvény deklarációja a következőképpen néz ki:

```
void sort()
    template <class RandomAccessIterator>
        void sort (RandomAccessIterator first, RandomAccessIterator last);
    template <class RandomAccessIterator, class Compare>
        void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);
```

A következő kódcsipet a Robotautó Világbajnokság projektből származik:

```
std::vector<Gangster> gangsters;
[...]
std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( Gangster x, Gangster y )
{
    return dst ( cop, x.to ) < dst ( cop, y.to );
} );
```

Azt látjuk, hogy egy Gangster objektumokból álló vektort rendezünk. Ekkor megtehetnénk azt, hogy az osztályban felüldefináljuk az operátort, vagy csinálhatjuk így is, lambda kifejezéssel. Itt a rendezés alapját a cop-tól, azaz a rendőrtől mért távolság adja, azaz azok az elemek kerülnek a vektor elejére, amelyek a rendőrhöz a legközelebb vannak.

## 17.2. C++11 Custom Allocator

<https://prezi.com/jvvbytkwgsxj/high-level-programming-languages-2-c11-allocators/> a CustomAlloc-os példa, lásd C forrást az UDPORG repóban!

C++-ban az allokátorok szerepe, mint ahogyan a nevük is mutatja, hogy memóriát allokáljanak az adatszerkezeteink számára. Ugyan az alapértelmezett allokátor. Megtehetjük, hogy sajátot írunk, például egy vektorban tárolt értékek tárolására, a header ezt tartalmazza:

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```

Látható, hogy az Allocatornak van alapértelmezett értéke, de ezt felülírhatjuk.

A cél, hogy egy osztály minél kevesebb dologért feleljön. A memória foglalás minden veszélyes, jobb ha ezt egy külön részre bontjuk.

Ezen kívül még van a template allokátor. Ilyen template-eket akkor használunk, amikor azt szeretnénk elérni, hogy ugyan az a kód több típusra is működjön. Mivel példán keresztül könnyebb megmutatni, megpróbálom így is. Jelen esetünkben a template használata nélkül a következőképpen kellene, hogy kinézzünk a kódunkat:

<https://www.facebook.com/groups/udprog/permalink/1231713563683197/>

```
struct CustomAllocInt {
    using size_type = size_t;
    using value_type = int;
    using const_pointer = const int*;
    using reference = int&;...
```

```
struct CustomAllocChar {
    using size_type = size_t;
    using value_type = char;
    using const_pointer = const char*;
    using reference = char&;...
```

Tehát a cél az, hogy minél kevesebbet kelljen írni. Ha nem használnánk ezt a templatet, akkor meg kéne írjuk ezt az allokátort a létező összes típusra.

A program kimenete a következőképp néz ki:

```
$ g++ --std=c++17 CustomAlloc.cc && ./a.out
    Allocating 1 object(s) of 4 bytes. i=int
    Allocating 1 object(s) of 8 bytes. l=long
```

```
Allocating 1 object(s) of 32 bytes. ←
NST7__cxx11basic_stringIcSt11char_traitsIcESaIcEEE=std::←
__cxx11::basic_string<char, std::char_traits<char>, std::←
allocator<char> >
```

Láthatjuk, hogy hány bájt megy el a foglalásokra, illetve hogy milyen típusból foglalunk. A string foglalásánál megjelenő típusnév rövidítését és típusnevét az magyarázza, hogy az std::string egy osztály, amely eldönti, hogy milyen konkrét típussal dolgozzon.

### 17.3. STL map érték szerinti rendezése

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

```
#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-re, 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 előzőhöz hasonló. Ezt a for ciklust range-based for-nak hívják. Ennek a szintaktikája a következő:

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.

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

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 hajtódiék 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!=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 betűtípust és a betűméretet. A függvény törzsében a megadott paraméterekekkel 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 háttérét, az utolsó sor pedig újra aktiválja a képet.

# 18. fejezet

## Helló, unknown!

### 18.1. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocaremulator/blob/master/justine/rc.cpp>

Az std::sscanf adatok beolvasására alkalmas, egy null végződésű karakter string bufferból.

Az std::sscanf függvény:

```
int sscanf( const char* buffer, const char* format, ... );
```

buffer:

Mutató egy null végződésű karakter string-re, ahonnan olvasunk.

format:

Mutató egy null végződésű karakter string-re, ami azt határozza meg, hogy hogyan olvassuk az inputot.

Visszatérési érték:

Siker esetén egy int ami azt adja meg hogy hány argumentumot sikerült beolvasnunk. Input hiba esetén, mielőtt bármit is beolvastunk volna, EOF- el tér vissza.

A pos-al kezdődik három int kell legyen benne és az értékek között nulla vagy több \t kell szerepeljen, az ilyen szövegrészre fog illeszkedni a kifejezés, ha ilyen szövegrészt olvastunk akkor az utánna szereplő kifejezés fog kiértékelődni:

```
[  
 {  
     std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to) ←  
     ;  
     m_cmd = 10001;  
 }
```

Itt pedig az std::sscanf függvényel yytext-ből, yytext tartalmazza azt a szövegrészt amire a lexer egyezést talált, beolvassuk az értékeket m\_id, from, és to-ba. Ezek a változók a carlexer.hpp állományban vannak definiálva. Majd az m\_cmd 10001-re állítjuk, ez a változó is a carlexer.hpp-ban található.

## 18.2. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>  
<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();
```

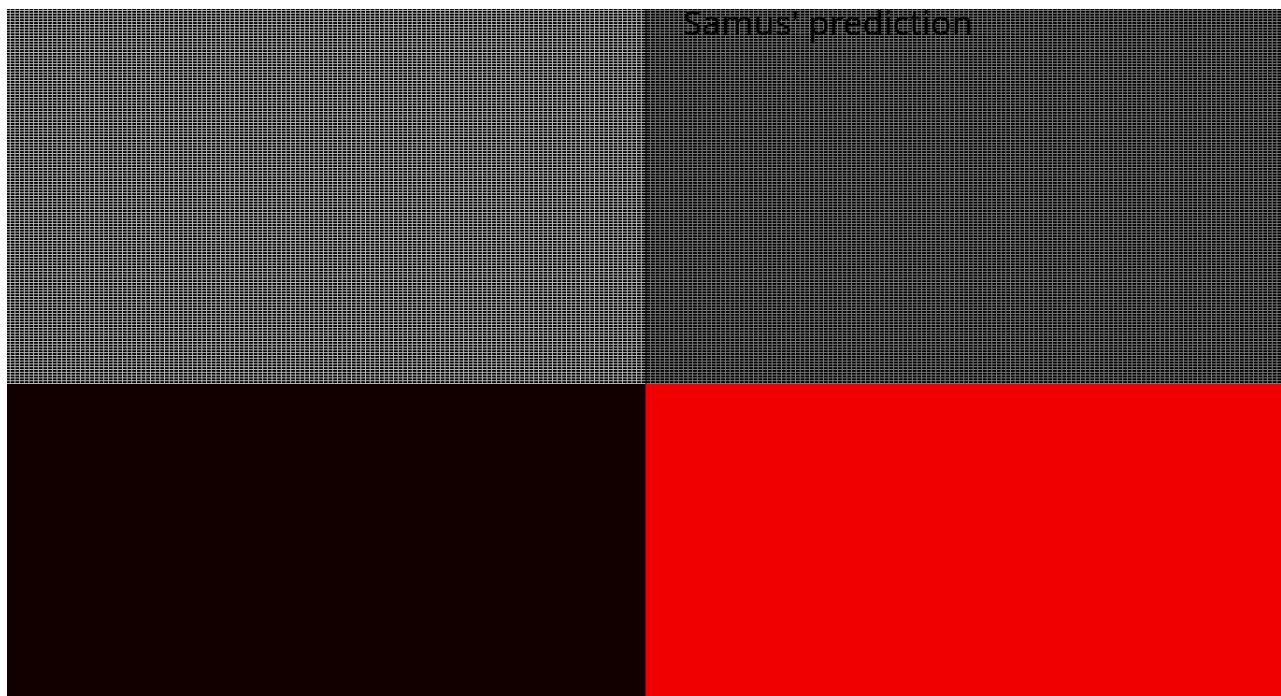
Az opencv segítségével képes a program a kamera használatára, a SamuCam osztály egy QThread objektumot képvisel, ezen belül található az openVideoStream, melyet majd a felvétel kezeléséhez használunk a **cv::VideoCapture::videoCapture**; segítségével.

```
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.

Igen ám, el is jutottam a program felélesztéséhez, de sajnos valahogy mégsem sikerült elérnem, hogy működő kamerán keresztül kapjam az adatokat vissza. Ha alapértelmezett kamerát szeretnénk használni (pl. a laptop kameráját), akkor a fent látható videoCapture.open()-nek videoStream helyett 0 értéket kell megadnunk. Ha pedig mondjuk a telefonunkra letöltött bármilyen IP Cam alkalmazással szeretnénk elérni a kamera működését, maradhat a videoStream paraméter. Ez esetben futtatásnál egy --ip opcióval megadjuk az ip kameránk ip címét, és ha minden jól megy, működnie kell.



Screenshot a SamuCam futtatásáról, kamera nélkül...

### 18.3. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esport-talent-search>

A program 10 percig fut, ezalatt az idő alatt az a feladatunk hogy a bal egérgombot lenyomva Samu Entropy-n tartsuk az egeret. A program futás közben statisztikát készít az eredményeinkről amit a program végezettel megtekinthetünk. A program Qt segítségével van elkészítve, ami egy keresztpalermos alkalmazás-keretrendszer, amit GUI-s alkalmazások, illetve nem GUI-s programok fejlesztésére használnak.

Ismertebb alkalmazások és felhasználók közé tartozik, amiknek fejlesztéséhez Qt-t használtak pl. az Autodesk, Google Earth, KDE, Skype, VLC media player, VirtualBox, Mathematica stb. Hivatalosan Linux, Mac OS, Windows és Symbian platformokat támogat.

A Qt részletes leírása ezen a linken megtalálható: <https://hu.wikipedia.org/wiki/Qt>

A Qt innen töltethető le: <https://www.qt.io/>

A program használatához a következők telepítése a szükséges:

```
sudo dnf install libqt4-devel  
sudo dnf install opencv  
sudo dnf install opencv-devel
```

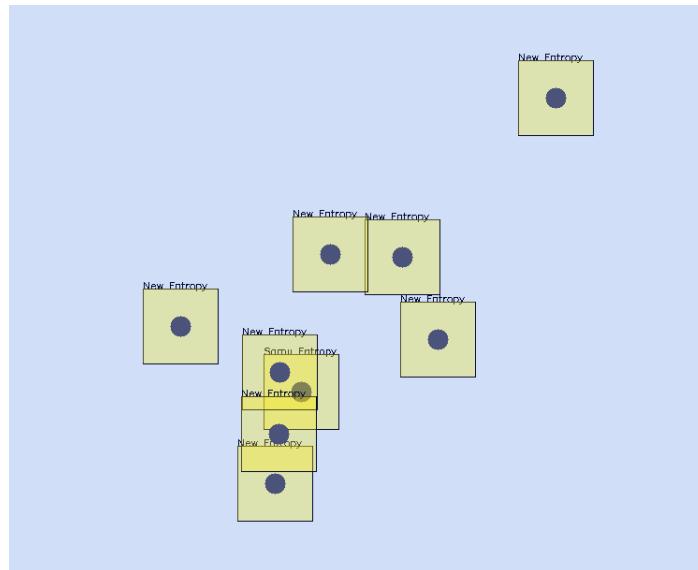
```
BrainBWin::BrainBWin ( int w, int h, QWidget *parent ) : QMainWindow ( ←  
    parent )  
{  
    //      setWindowTitle(appName + " " + appVersion);  
    //      setFixedSize(QSize(w, h));
```

```
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 ) ) );
}
```

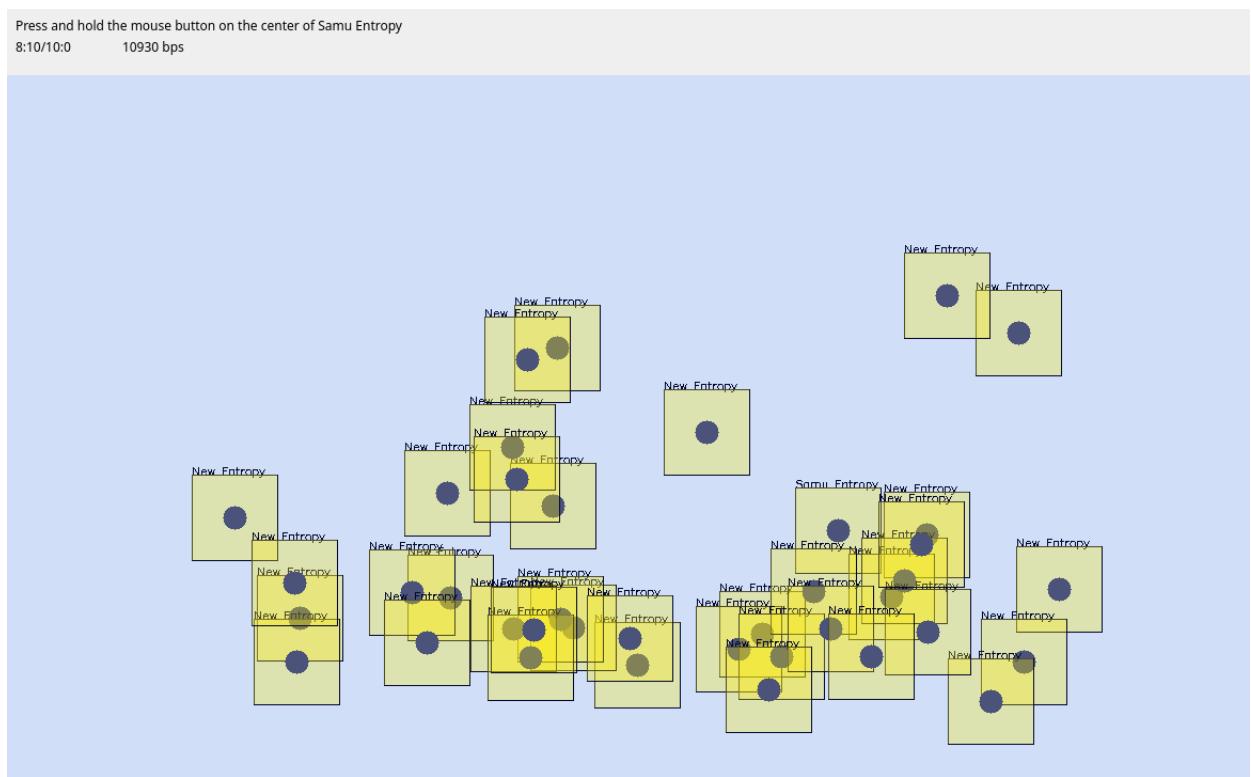
Ez a slot-signal mechanizmus az objektumok közötti kommunikációt teszi lehetővé. Ez a meta-object rendszer segítségével lehetséges.

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 végrehajtódik, akkor egy jel küldődik az updateHeroes slotba, és ezzel az updateHeroes függvény is végrehajtásra kerül.

Program futását a Qt letöltése után tudjuk elérni, mégpedig az én esetemben a saját home mappámba mentettem, tehát megadjuk az elérési útat, aztán így "qmake"-eljük a BrainB.pro fájlt, ezután make, aztán futtatjuk a keletkezett futtatható BrainB fájlt. Két kép a program futásáról:



Fedora linux screenshot



Fedora linux screenshot

## 19. fejezet

# Helló, Lauda!

### 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>

```
public class KapuSzkenner {  
  
    public static void main(String[] args) {  
  
        for(int i=0; i<1024; ++i)  
  
            try {  
  
                java.net.Socket socket = new java.net.Socket(args ↪  
                    [0], i);  
  
                System.out.println(i + " figyeli");  
  
                socket.close();  
  
            } catch (Exception e) {  
  
                System.out.println(i + " nem figyeli");  
  
            }  
    }  
}
```

A portszkennelés lényege, hogy kiderítsük, mennyi portunk van, ami használható.

A fent látható forráskód annyit tesz, hogy végigmegy 1024 kapun, megpróbál kapcsolatot létesíteni a `java.net.Socket` osztály segítségével egy socketet nyitni, ha ez sikerül, az azt jelenti, hogy a célporton

van valamilyen szerver folyamat működésben. Azokat a kimeneteket kell figyelembe vennünk, ami portszám + figyeli formában van jelen.

A java.net.Socket osztály dokumentációjában a következőt olvashatjuk:

SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.

Azaz ha nem tud rácsatlakozni, beleakad a fenti kivételbe, ezáltal nem engedélyezni az adott műveletet.

Az output a következőképp néz ki. Természetesen 1024-ig:

```
[salesz@localhost java_practice]$ java KapuSzkenner
0 nem figyeli
1 nem figyeli37): WARNING **: 18:57:15.742: Error s
2 nem figyelictory
3 nem figyeli
4 nem figyeli37): WARNING **: 18:57:15.742: Error s
5 nem figyelictory
6 nem figyeli
7 nem figyeli37): WARNING **: 18:57:15.754: Error s
8 nem figyelictory
9 nem figyelik-fdl.pdf' successfully built
10 nem figyelinst textbook]$
11 nem figyeli Gtk-CRITICAL **: 18:57:21.577: gtk_t
12 nem figyeli priv->model != NULL' failed
13 nem figyeli
```

Fedora linux screenshot

## 19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Az aspektus orientált programozást (AOP) egy olyan szemlélet, amely lehetőséget ad számunkra egy bizonyos kód módosítására, anélkül, hogy a kódot módosítanánk. Lehetőségünk van például megadni kódcsipeteket egy külön fájlban, amiknek megmondhatjuk, hogy fussanak le az után, vagy előtt, amikor a programban például lefut egy metódus.

Az AspectJ az AOP szemlélet Java-s kivetülése. Ezt a legtöbb GNU/Linux disztribúción telepíthetjük a csomagkezelőnkkel. Ubuntu-n:

```
$ sudo apt install aspectj
```

Ha szeretnénk megszámlálni az egyesek és nullások számát a fában, de nem vagyunk elég ügyesek ahhoz, hogy az LZWBInFa osztályt módosítsuk, segíteni fog az AspectJ, ahol úgynevezett pointcut-ok segítségével megmondhatjuk, hogy mi történjen egy bizonyos metódus lefutása után.

```
public aspect Aspect {
    private int ones = 0, zeros = 0;

    // Az addbit() pointcut -- hogy tudjuk számolni a bemenő ←
        adatokat
```

```
pointcut addbit(): execution(public void addBit(char));  
  
before(char b): addbit() && args(b) {  
    if ('1' == b)  
        ++ones;  
    else if ('0' == b)  
        ++zeros;  
}  
// A main() függvény pointcut -- hogy ki tudjuk írni a lefutás ←  
// után az eredményt  
pointcut main(): execution(public static void main(String[]));  
  
after(): main() {  
    System.out.println("ones = " + ones);  
    System.out.println("zeros = " + zeros);  
}  
}
```

A program futása:

```
in = 01111001001001000111  
melyseg = 4  
atlag = 2.75  
szoras = 0.9574271077563381  
ones = 10  
zeros = 10
```

### 19.3. Junit teszt

A [https://progpater.blog.hu/2011/03/05/labormeres\\_otthon\\_avagy\\_hogyan\\_dolgozok\\_fel\\_egy\\_pedat poszt kézzel számított mélysegét és szórását dolgozd be egy Junit tesztbe](https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat_poszt_kézzel_számított_mélysegét_és_szórását_dolgozd_be_egy_Junit_tesztbe) (sztenderd védési feladat volt korábban).

Az úgynevezett unit tesztek lényege, hogy a kódok automatikusan tesztelhetőek legyenek. Ahelyett, hogy rengeteg időt tölténénk minden egyes verzió kézi tesztelésével, úgynevezett unit teszteket hozunk létre, amik például minden egyes commit után (ha használunk verziókövetőt, ami még az unit teszteknél is fontosabb) letesztelek az aktuális kódot előre megadott módon.

Élő példát ilyen tesztekre a Firefox Treeherder oldalán láthatunk:

<https://treeherder.mozilla.org/#/jobs?repo=autoland>

Természetesen ezek a tesztek nem helyettesítik teljesen a szoftver kézi tesztelését, csupán lerövidítik, kiegészítik azt. Megnézhetjük például a Firefox stratégiáját a következő oldalon:

[https://developer.mozilla.org/en-US/docs/Mozilla/QA/Automated\\_testing](https://developer.mozilla.org/en-US/docs/Mozilla/QA/Automated_testing)

A JUnit a Java programnyelvhez létrehozott unit tesztelési megoldás. A használata: létrehozunk egy új osztályt, ahol beimportáljuk a tesztelni kívánt kód osztályait, majd egy előre meghatározott szintaxisával megadjuk, hogy tesztelésnél milyen eredményeket várunk. Tekintsük a következő példát a JUnit leírásából: <https://junit.org/junit5/docs/current/user-guide/#writing-tests>.

Íme a példa:

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import example.util.Calculator;
import org.junit.jupiter.api.Test;
class MyFirstJUnitJupiterTests {
    private final Calculator calculator = new Calculator();
    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }
}
```

Látjuk, hogy importálásra kerül a Calculator osztály, és az addition() metódusban történik maga az összehasonlítás a calculator objektum (ami a Calculator osztály egy példánya) add() metódusát hívjuk 1-re és 1-re.

## 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\\_s/](https://progpater.blog.hu/2016/11/13/hello_s/) bol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Tulajdonképpen az MNIST rövidítése nem más, mint: Modified National Institute of Standards and Technology, ami egy hatalmas adatbázis, ami rengeteg kézzel írott számjegyeket tartalmaz. Ez egy gyakran használt tanítható képfeldolgozó rendszer. Most pedig életre fogjuk kelteni, mégpedig úgy, hogy egy általunk rajzolt számjegyet is felismerjen a programunk.

Először is nézzük meg forráskódunkban a szükséges csomagokat, amiket telepítenünk kell ahhoz, hogy működésre tudjuk bírni a kódunkat. Ezeket könnyű felismerni, ezek az import kulcsszóval kezdődő sorok lesznek, pontosabban:

```
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D ←
    , MaxPooling2D
from keras.models import Sequential
from keras.utils import to_categorical,np_utils
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
```

Tehát első körben szükségünk lesz a pip-re, amit könnyedén tudunk telepíteni. Fedora rendszeren a következőképp

```
yum install python-pip
```

Ha ezzel megvagyunk feltesszük a tensorflow-t, amit a következőképpen tehetünk meg a pip használatával:

```
pip install tensorflow
```

Aztán majd a fent látható csomagokat külön feltelepítgetjük és meg is vagyunk:

```
pip install keras  
pip install matplotlib
```

Ha ezzel megvagyunk már is fut a programunk...

```
salesz@localhost:~/Prog2/java_practice  
File Edit View Search Terminal Help  
57088/60000 [=====>...] - ETA: 1s - loss: 0.1496 - accuracy: 0.954  
57216/60000 [=====>...] - ETA: 1s - loss: 0.1494 - accuracy: 0.954  
57344/60000 [=====>...] - ETA: 1s - loss: 0.1493 - accuracy: 0.954  
57472/60000 [=====>...] - ETA: 1s - loss: 0.1491 - accuracy: 0.954  
57600/60000 [=====>...] - ETA: 1s - loss: 0.1491 - accuracy: 0.954  
57728/60000 [=====>...] - ETA: 1s - loss: 0.1490 - accuracy: 0.954  
57856/60000 [=====>...] - ETA: 1s - loss: 0.1487 - accuracy: 0.954  
57984/60000 [=====>...] - ETA: 1s - loss: 0.1485 - accuracy: 0.955  
58112/60000 [=====>...] - ETA: 1s - loss: 0.1484 - accuracy: 0.955  
58240/60000 [=====>...] - ETA: 1s - loss: 0.1483 - accuracy: 0.955  
58368/60000 [=====>...] - ETA: 1s - loss: 0.1482 - accuracy: 0.955  
58496/60000 [=====>...] - ETA: 0s - loss: 0.1479 - accuracy: 0.955  
58624/60000 [=====>...] - ETA: 0s - loss: 0.1479 - accuracy: 0.955  
58752/60000 [=====>...] - ETA: 0s - loss: 0.1481 - accuracy: 0.955  
58880/60000 [=====>...] - ETA: 0s - loss: 0.1481 - accuracy: 0.955  
59008/60000 [=====>...] - ETA: 0s - loss: 0.1479 - accuracy: 0.955  
59136/60000 [=====>...] - ETA: 0s - loss: 0.1477 - accuracy: 0.955  
59264/60000 [=====>...] - ETA: 0s - loss: 0.1476 - accuracy: 0.955  
59392/60000 [=====>...] - ETA: 0s - loss: 0.1475 - accuracy: 0.955  
59520/60000 [=====>...] - ETA: 0s - loss: 0.1473 - accuracy: 0.955  
59648/60000 [=====>...] - ETA: 0s - loss: 0.1471 - accuracy: 0.955  
59776/60000 [=====>...] - ETA: 0s - loss: 0.1469 - accuracy: 0.955  
59904/60000 [=====>...] - ETA: 0s - loss: 0.1468 - accuracy: 0.955  
60000/60000 [=====] - 37s 621us/step - loss: 0.1466 - accuracy: 0.9556
```

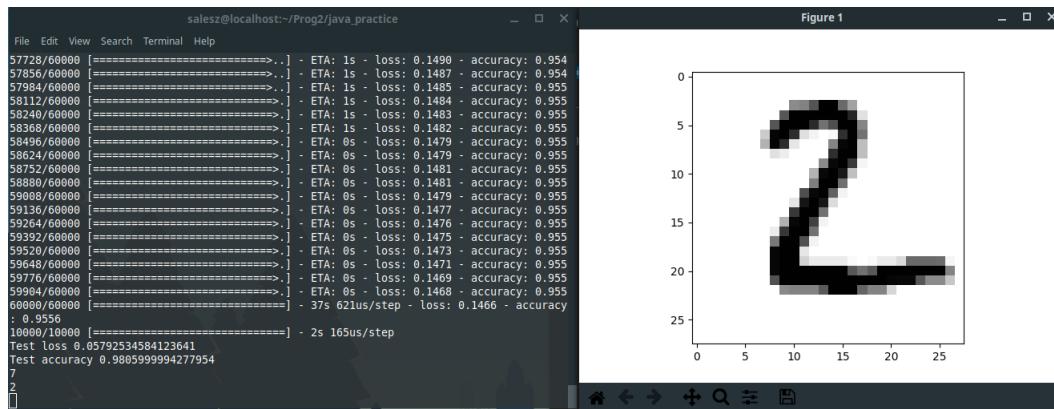
Fedora linux screenshot

Aztán fel is ugrik az első képünk, ami egy 7-es, és mint látjuk a terminálban, ezt jól is ismerte fel programunk:

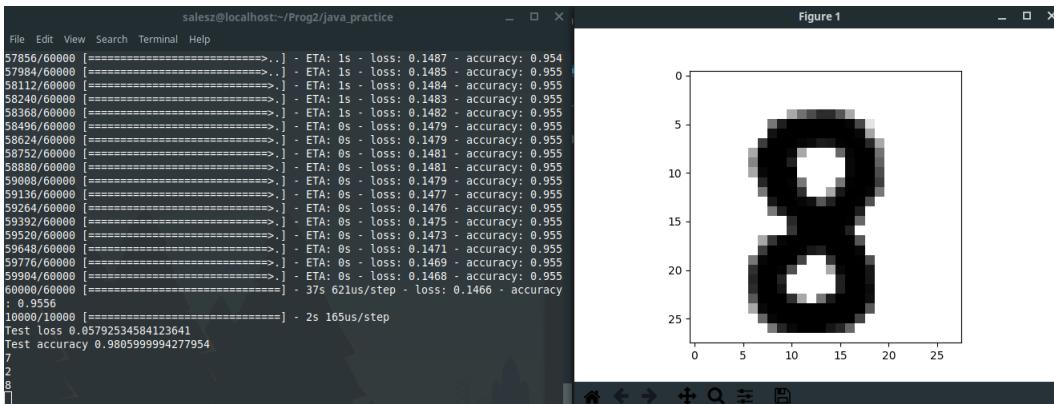
```
salesz@localhost:~/Prog2/java_practice  
File Edit View Search Terminal Help  
57600/60000 [=====>...] - ETA: 1s - loss: 0.1491 - accuracy: 0.954  
57728/60000 [=====>...] - ETA: 1s - loss: 0.1490 - accuracy: 0.954  
57856/60000 [=====>...] - ETA: 1s - loss: 0.1487 - accuracy: 0.954  
57984/60000 [=====>...] - ETA: 1s - loss: 0.1485 - accuracy: 0.955  
58112/60000 [=====>...] - ETA: 1s - loss: 0.1484 - accuracy: 0.955  
58240/60000 [=====>...] - ETA: 1s - loss: 0.1483 - accuracy: 0.955  
58368/60000 [=====>...] - ETA: 1s - loss: 0.1482 - accuracy: 0.955  
58496/60000 [=====>...] - ETA: 0s - loss: 0.1479 - accuracy: 0.955  
58624/60000 [=====>...] - ETA: 0s - loss: 0.1479 - accuracy: 0.955  
58752/60000 [=====>...] - ETA: 0s - loss: 0.1481 - accuracy: 0.955  
58880/60000 [=====>...] - ETA: 0s - loss: 0.1481 - accuracy: 0.955  
58908/60000 [=====>...] - ETA: 0s - loss: 0.1479 - accuracy: 0.955  
59036/60000 [=====>...] - ETA: 0s - loss: 0.1477 - accuracy: 0.955  
59164/60000 [=====>...] - ETA: 0s - loss: 0.1476 - accuracy: 0.955  
59292/60000 [=====>...] - ETA: 0s - loss: 0.1475 - accuracy: 0.955  
59520/60000 [=====>...] - ETA: 0s - loss: 0.1473 - accuracy: 0.955  
59648/60000 [=====>...] - ETA: 0s - loss: 0.1471 - accuracy: 0.955  
59776/60000 [=====>...] - ETA: 0s - loss: 0.1469 - accuracy: 0.955  
59904/60000 [=====>...] - ETA: 0s - loss: 0.1468 - accuracy: 0.955  
60000/60000 [=====] - 37s 621us/step - loss: 0.1466 - accuracy: 0.9556  
Figure 1
```

Fedora linux screenshot

Aztán ugyanezt láthatjuk egy 2-essel, aztán egy általam rajzolt 8-as karakterrel, amit szintén sikeresen felismert:



Fedora linux screenshot



Fedora linux screenshot

A teljes python forráskódöt pedig itt láthatjuk:

```

import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D ←
    , MaxPooling2D
from keras.models import Sequential
from keras.utils import to_categorical,np_utils
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
(train_X,train_Y), (test_X,test_Y) = tf.keras.datasets. ←
mnist.load_data()

train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255

```

```
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, ←
    optimizer=keras.optimizers.Adam(), metrics=['accuracy'])

model.fit(train_X, train_Y_one_hot, batch_size=64, epochs ←
    =1)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot ←
    )
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)

print(np.argmax(np.round(predictions[0])))
plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
print(np.argmax(np.round(predictions[1])))
plt.imshow(test_X[1].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
img = Image.open('sajat8a.png').convert("L")
img = np.resize(img, (28,28,1))
im2arr = np.array(img)
im2arr = im2arr.reshape(1,28,28,1)
print(np.argmax(np.round(model.predict(im2arr))))
plt.imshow(im2arr[0].reshape(28,28),cmap = plt.cm.binary)
plt.show()
```

## 20.2. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

A TensorFlow objektum dektelektálóját könnyedén beszerezhetjük a következő linkről:

<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android>

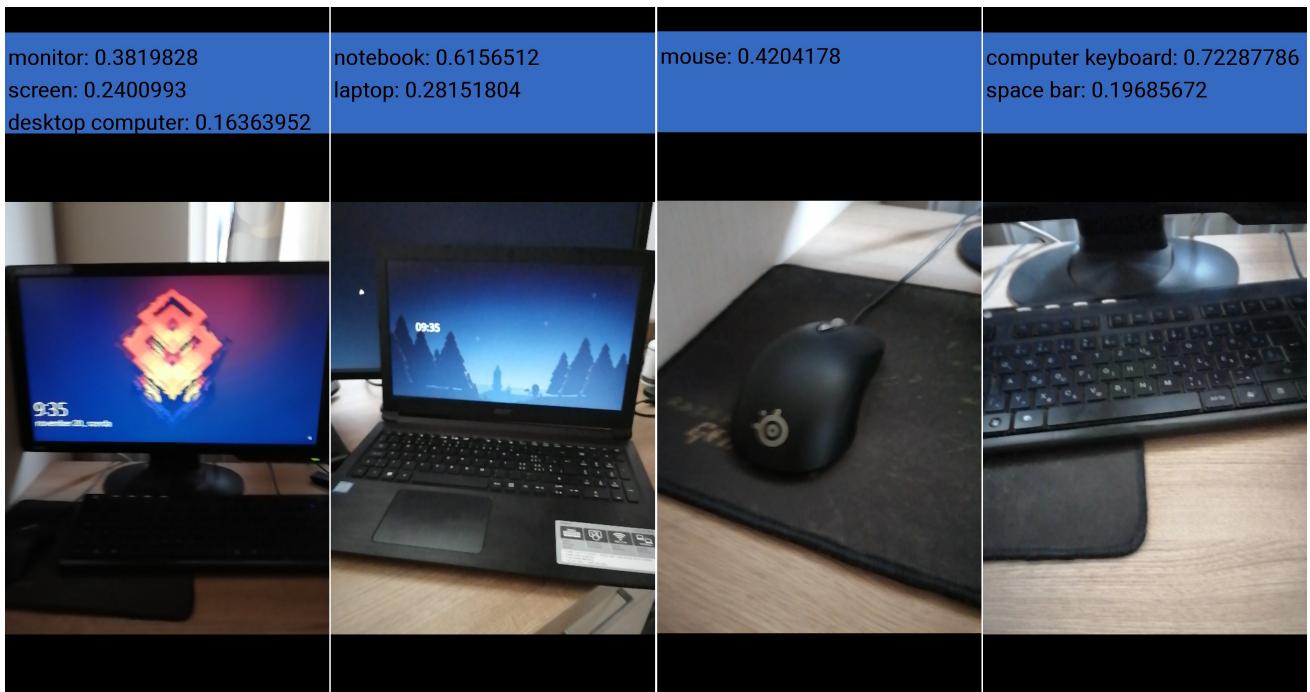
Mint a neve is sugallja, ez a szoftver különböző objektumokat próbál detektálni/felismerni. Most pedig leteszteljük, mennyire végez jó munkát.

Miután leklónoztuk a fent linkelt repót, el is kezdhetjük telepíteni az apk fájlt. Ezen hamar, zökkenőmentesen töllehetünk, a telepítés a következőképp alakul, semmi bonyolult nincs benne, semmiféle plusz beállításra sincs szükség.

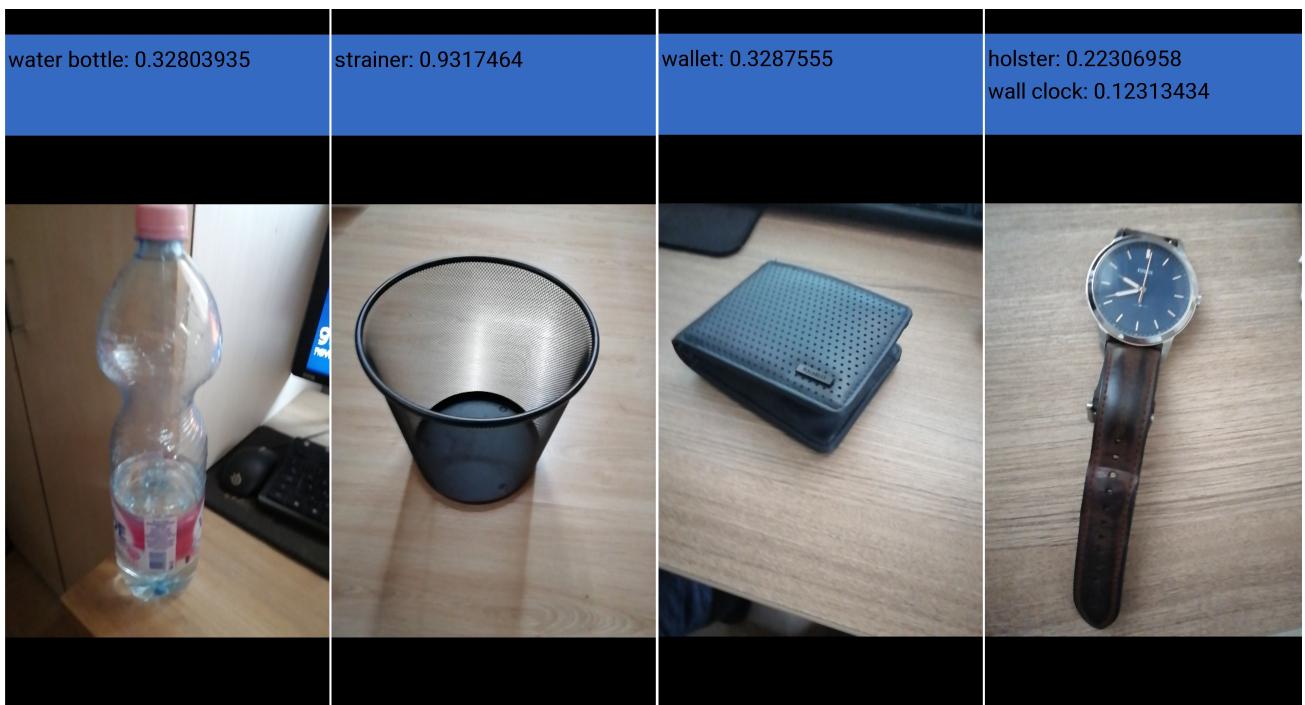


*Android Screenshot - TensorFlow telepítése*

Miután feltelepítettük, megnyitjuk az alkalmazást, aztán engedélyezzük, hogy hozzáférjen az eszköz kamerájához, illetve fájljaihoz. Ezután pedig kezdődhet a felfedezés/tesztelés. Irányítsuk a kamerát különböző objektumokra (lehetőleg egyszerre csak egy látszódjon), és már lábjuk a felső kék sávban, hogy programunk minek ismeri fel az adott objektumot. Az én esetben egész jól teljesített a program, 8 képből 6-ot helyesen felismert, így elmondhatom, hogy esetben a program 75%-os pontossággal dolgozott. Ez egész jó szám. Képek a tesztelésről:



Android Screenshot - Első tesztek



Android Screenshot - Tesztelés folytatása

## 20.3. CIFAR-10

Az alap feladat megoldása, +saját fotót is ismerjen fel, [https://progpater.blog.hu/2016/12/10/hello\\_samu\\_a\\_cifar-10\\_tf\\_tutorial\\_peldabol](https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tf_tutorial_peldabol)

Azt, hogy mik szükségesek a feladathoz, hogy futtatásra bírjuk, nem fogom mégegyszer leírni, előzőből láthatod részletesen kielemezve. Ezt szintén virtuális környezettel fogom megoldani.

Első körben, mikor futtatjuk a programot, rengeteg adatot tölt le nekünk, most már egy jóval szívósabb adatbázisunk van, hiszen mostmár nem számokat, hanem képeket, ráadásul színes képeket fogunk felismertetni programunkkal. Szerintem érezhető a különbség. Itt láthatjuk, ahogy a programunk futtatás során elkezdi letölteni az adatokat, ami hosszabb időt is igénybe vehet:

```
[salesz@localhost java_practice]$ source ./venv/bin/activate
(venv) [salesz@localhost java_practice]$ python3 cifar10.py
Using TensorFlow backend.
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
  10772480/170498071 [>.....] - ETA: 8:01
```

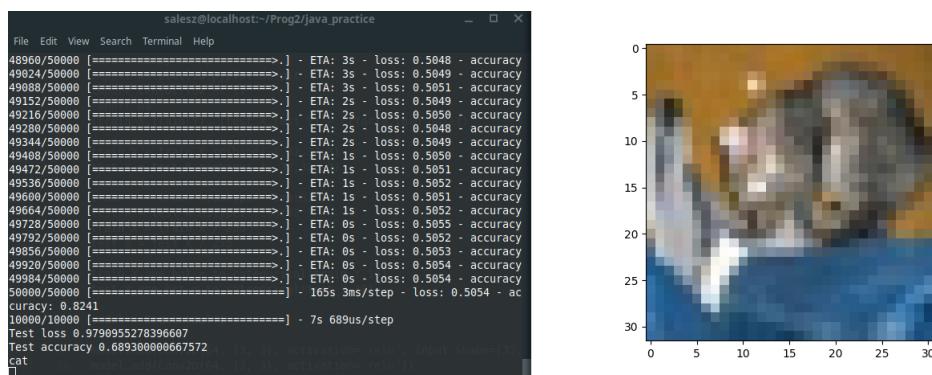
#### Fedora Screenshot - cifar futtatása

A cifar első futásánál elkezdi magát tanítani, ami még több idő, ugyanis 25 fázison keresztül megy végig 50.000-szer. Ez akár órákba is telhet...

Pontosabban végigmenne 25 fázison, ha nem órán csinálnám a feladatot és nyúlnának hozzá a laptopomhoz a hallgató kollegák, na de sebaj... fussunk neki mégegyszer immár csak 5 próbálkozással. Egyébként a tanítások számát, hogy hány fázisban történjen a következő sorban tudjuk megadni, pontosabban az epochs változónk értéke a meghatározó:

```
[model.fit(train_X, train_Y_one_hot, batch_size=64, epochs ←
=5)
```

Hát igen, ne is várunk jó eredményeket 5 fázis után... Nos, 3 képből sajnos egyet sem ismert fel helyesen a programunk. Íme az eredmények:

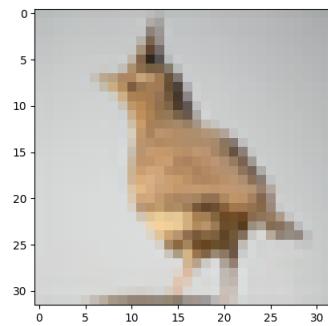


Fedora linux screenshot

```
salesz@localhost:~/Prog2/java_practice
```

```
File Edit View Search Terminal Help
```

```
49024/50000 [=====>..] - ETA: 3s - loss: 0.5049 - accuracy
49088/50000 [=====>..] - ETA: 3s - loss: 0.5051 - accuracy
49152/50000 [=====>..] - ETA: 2s - loss: 0.5049 - accuracy
49216/50000 [=====>..] - ETA: 2s - loss: 0.5056 - accuracy
49280/50000 [=====>..] - ETA: 2s - loss: 0.5048 - accuracy
49344/50000 [=====>..] - ETA: 2s - loss: 0.5049 - accuracy
49408/50000 [=====>..] - ETA: 1s - loss: 0.5051 - accuracy
49472/50000 [=====>..] - ETA: 1s - loss: 0.5051 - accuracy
49536/50000 [=====>..] - ETA: 1s - loss: 0.5052 - accuracy
49600/50000 [=====>..] - ETA: 1s - loss: 0.5051 - accuracy
49664/50000 [=====>..] - ETA: 1s - loss: 0.5052 - accuracy
49728/50000 [=====>..] - ETA: 0s - loss: 0.5055 - accuracy
49792/50000 [=====>..] - ETA: 0s - loss: 0.5052 - accuracy
49856/50000 [=====>..] - ETA: 0s - loss: 0.5053 - accuracy
49920/50000 [=====>..] - ETA: 0s - loss: 0.5054 - accuracy
49984/50000 [=====>..] - ETA: 0s - loss: 0.5054 - accuracy
50000/50000 [=====>..] - 165s 3ms/step - loss: 0.5054 - accuracy: 0.8241
10000/10000 [=====>..] - 7s 689us/step
Test loss 0.9790955278396607
Test accuracy 0.689300000667572
cat
airplane
automobile 1
```

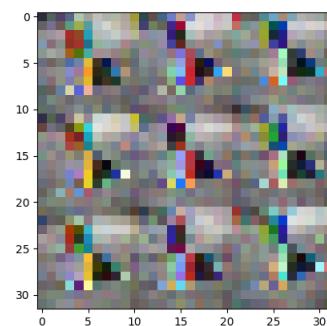


Fedora linux screenshot

```
salesz@localhost:~/Prog2/java_practice
```

```
File Edit View Search Terminal Help
```

```
49088/50000 [=====>..] - ETA: 3s - loss: 0.5051 - accuracy
49152/50000 [=====>..] - ETA: 2s - loss: 0.5049 - accuracy
49216/50000 [=====>..] - ETA: 2s - loss: 0.5056 - accuracy
49280/50000 [=====>..] - ETA: 2s - loss: 0.5048 - accuracy
49344/50000 [=====>..] - ETA: 2s - loss: 0.5049 - accuracy
49408/50000 [=====>..] - ETA: 1s - loss: 0.5050 - accuracy
49472/50000 [=====>..] - ETA: 1s - loss: 0.5051 - accuracy
49536/50000 [=====>..] - ETA: 1s - loss: 0.5052 - accuracy
49600/50000 [=====>..] - ETA: 1s - loss: 0.5051 - accuracy
49664/50000 [=====>..] - ETA: 1s - loss: 0.5052 - accuracy
49728/50000 [=====>..] - ETA: 0s - loss: 0.5055 - accuracy
49792/50000 [=====>..] - ETA: 0s - loss: 0.5052 - accuracy
49856/50000 [=====>..] - ETA: 0s - loss: 0.5053 - accuracy
49920/50000 [=====>..] - ETA: 0s - loss: 0.5054 - accuracy
49984/50000 [=====>..] - ETA: 0s - loss: 0.5054 - accuracy
50000/50000 [=====>..] - 165s 3ms/step - loss: 0.5054 - accuracy: 0.8241
10000/10000 [=====>..] - 7s 689us/step
Test loss 0.9790955278396607
Test accuracy 0.689300000667572
cat
airplane
automobile 1
```



Fedora linux screenshot

## **IV. rész**

### **Irodalomjegyzék**

## 20.4. Általános

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

## 20.5. C

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

## 20.6. C++

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

## 20.7. Lisp

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

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

Ezen túl kiemelt köszönet illeti az említett UDPORG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségen született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.