

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert, Heinrich László

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert	December 13, 2019	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.1.0	2019-03-01	Első csokor kész.	heinrichlaszlo
0.2.0	2019-03-12	Második csokor kész.	heinrichlaszlo
0.3.0	2019-03-19	Harmadik csokor befejezve.	heinrichlaszlo
0.4.0	2019-03-19	Negyedik csokor kész.	heinrichlaszlo

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.5.0	2019-03-26	Ötödik csokor kész.	heinrichlaszlo
0.6.0	2019-04-02	Hatodik csokor kész.	heinrichlaszlo
0.7.0	2019-04-09	Hetedik csokor kész.	heinrichlaszlo
0.7.1	2019-04-13	Az eddigi fejezetek javítása.	heinrichlaszlo
0.8.0	2019-04-23	Nyolcadik csokor kész.	heinrichlaszlo
0.9.0	2019-04-29	Kilencedik csokor kész.	heinrichlaszlo
0.9.3	2019-05-04	Az eddigi fejezetek javítása.	heinrichlaszlo
0.9.4	2019-05-05	Az eddigi fejezetek javítása.	heinrichlaszlo
0.9.5	2019-05-07	Az eddigi fejezetek javítása.	heinrichlaszlo
0.9.6	2019-05-08	Az eddigi fejezetek javítása.	heinrichlaszlo

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](#)]

Contents

I Bevezetés	1
1 Vízió	2
1.1 Mi a programozás?	2
1.2 Milyen doksikat olvassak el?	2
1.3 Milyen filmeket nézzek meg?	2
II Tematikus feladatok	3
2 Helló, Turing!	5
2.1 Végtelen ciklus	5
2.2 Lefagyott, nem fagyott, akkor most mi van?	6
2.3 Változók értékének felcserélése	8
2.4 Labdapattogás	10
2.5 Szóhossz és a Linus Torvalds féle BogoMIPS	13
2.6 Helló, Google!	13
2.7 100 éves a Brun téTEL	15
2.8 A Monty Hall probléma	15
3 Helló, Chomsky!	18
3.1 Decimálisból unárisba átváltó Turing gép	18
3.2 Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	18
3.3 Hivatkozási nyelv	19
3.4 Saját lexikális elemző	20
3.5 l33t.l	21
3.6 A források olvasása	23
3.7 Logikus	24
3.8 Deklaráció	25

4 Helló, Caesar!	27
4.1 int *** háromszögmátrix	27
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
5 Helló, Mandelbrot!	37
5.1 A Mandelbrot halmaz	37
5.2 A Mandelbrot halmaz a std::complex osztállyal	39
5.3 Biomorfok	42
5.4 A Mandelbrot halmaz CUDA megvalósítása	44
5.5 Mandelbrot nagyító és utazó C++ nyelven	49
5.6 Mandelbrot nagyító és utazó Java nyelven	55
6 Helló, Welch!	59
6.1 Első osztályom	59
6.2 LZW	62
6.3 Fabejárás	65
6.4 Tag a gyökér	68
6.5 Mutató a gyökér	71
6.6 Mozgató szemantika	73
7 Helló, Conway!	76
7.1 Hangyszimulációk	76
7.2 Java életjáték	93
7.3 Qt C++ életjáték	101
7.4 BrainB Benchmark	107
8 Helló, Schwarzenegger!	109
8.1 Szoftmax Py MNIST	109
8.2 Mély MNIST	111
8.3 Minecraft-MALMÖ	112

9 Helló, Chaitin!	113
9.1 Iteratív és rekurzív faktoriális Lisp-ben	113
9.2 Gimp Scheme Script-fu: króm effekt	115
9.3 Gimp Scheme Script-fu: név mandala	120
10 Helló, Gutenberg!	125
10.1 Juhász István: Magas szintű programozási nyelvek - olvasónapló	125
10.2 KR: A C programozási nyelv - olvasónapló	129
10.3 Benedek Zoltán, Levendovszky Tihamér: Szoftverfejlesztés C++ nyelven - olvasónapló	132
III Második felvonás	137
11 Helló, Arroway!	139
11.1 OO szemlélet	139
11.2 "Gagyi"	140
11.3 Yoda	142
11.4 Kódolás from scratch	143
12 Helló, Berners-Lee!	146
12.1 Forstner Bertalan, Ekler Péter, Kelényi Imre : Bevezetés a mobil programozásba	146
12.2 C++ és Java nyelv összehasonlítása	146
13 Helló, Liskov!	148
13.1 Liskov helyettesítés sértése	148
13.2 Szülő-gyerek	150
13.3 Anti OO	152
14 Helló, Mandelbrot!	154
14.1 Reverse engineering UML osztálydiagram	154
14.2 Forward engineering UML osztálydiagram	155
14.3 BPMN	156
15 Helló, Chomsky!	158
15.1 Encoding	158
15.2 l334d1c4 5	159
15.3 Full screen	162
15.4 Perceptron osztály	166

16 Helló, Stroustrup!	168
16.1 Változó argumentumszámú ctor és Összefoglaló	168
16.2 JDK osztályok	171
16.3 Hibásan implementált RSA törése	173
17 Helló, Gödel!	178
17.1 Alternatív Tabella rendezése	178
17.2 STL map érték szerinti rendezése	181
17.3 GIMP Scheme hack	183
17.4 Gengszterek	188
18 Helló, !	189
18.1 OOCWC Boost ASIO hálózatkezelése	189
18.2 BrainB	190
18.3 SamuCam	194
18.4 FUTURE tevékenység editor	196
19 Helló, Lauda!	199
19.1 Port scan	199
19.2 Android Játék	201
19.3 Junit teszt	208
20 Helló, Calvin!	212
20.1 MNIST	212
20.2 CIFAR-10	215
20.3 Android telefonra a TF objektum detektálója	218
IV Irodalomjegyzék	223
20.4 Általános	224
20.5 C	224
20.6 C++	224
20.7 Lisp	224

Eloszó

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 alkálni 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 feladatesokrot. 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 készítettem 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 mászt is) példával.

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

Part I

Bevezetés

Chapter 1

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.

Part II

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.

Chapter 2

Helló, Turing!

2.1 Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/vegln1.c>
- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/vegln2.c>
- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/vegln3mag.c>

Végtelen ciklus , ami 100 százalékban dolgoztat meg egy magot:

```
#include <stdio.h>

int main()
{
    for( ; ; )
    {

    }

    return 0;
}
```

Ez a program nem túl bonyolult. Leegyszerűsítve annyi történik, hogy létrehozunk egy olyan ciklust, amelynek a ciklusfejrésze üres. Az így kapott ciklusban a futási feltétel rész is üres (első pontosvessző utáni rész), magyarul mindenkor igaz lesz, így a ciklus addig, amíg valaki kívülről meg nem szakítja (pl.: **ctrl+c**), folyamatosan futni fog egy processzormagot 100 százalékban dolgoztatva.

Végtelen ciklus , ami 0 százalékban dolgoztat meg egy magot:

```
#include <stdio.h>

int main()
{
    for(;;)
    {
        sleep(1);
    }

    return 0;
}
```

A második programot ha összehasonlítjuk az első programmal csupán két különbség üti fel a fejét. Az egyik különbséget az include-olás során fedezhetjük fel : az unistd.h header fájlt include-oljuk a stdio.h helyett. Ez a fájl tartalmazza a sleep() functiont. A sleep() function a zárójelében megadott időmennyiséggel (ha nem adunk meg másik akkor alapértelmezetten másodpercen át) késlelteti a szál feldatainak végrehajtását. Mivel a megírt ciklusunk a "végtelenséggel" megy, így a sleep() számtalansor végrehajtódik, 0 százalékban dolgoztatva meg az adott magot.

Végtelen ciklus , ami 100 százalékban dolgoztat meg minden magot:

```
#include <omp.h>

int main()
{
    #pragma omp parallel for
    {
        for(;;);
    }

    return 0;
}
```

A harmadik programban szintén egy új header-rel találkozunk: omp.h. Ez header az OpenMP-t include-ol, ami lehetővé teszi számunkra a párhuzamos kódolást.

A fenti programok eredménye a htop paranccsal lett letesztelve.

2.2 Lefagyott, nem fagyott, akkor most mi van?

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

Megoldás videó:

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

```
Program T100
{
```

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

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

A T1000-res programot elkészítjük a T100-as programot felhasználva.

```
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...

A T100-as program eldöntia neki átadott programról , hogy van-e benne végtelen ciklus, majd vissza tér a true-val vagy false-sal. A T100-as visszatérési értékét átadjuk a T1000-es programnak, ha a kapott érték igaz, a program futása megáll, ha hamis, akkor végtelen ciklust kapunk . A T1000-es program akkor áll meg, ha a kapott érték igaz, vagyis ha a programban van végtelen ciklus, ellenkező esetben a T1000-es végtelen ciklusba kerül.

2.3 Változók értékének felcserélése

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

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

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/csereexor.c>
- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/cserekulonbseg.c>
- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/cserekulonbseg.c>

```
{  
    int elseo= 5;  
    int masodik= 10;  
  
    printf("elseo erteke:%d, masodik erteke:%d\n",elseo, masodik);  
  
    elseo = elseo-masodik; //elseo = -5, masodik = 10  
    masodik = elseo+masodik; // masodik = 15, elseo = 5  
    elseo = masodik-elseo; // elseo = 10, masodik = 5  
  
    printf("Ertekeik a csere utan: ");  
  
    printf("elseo:%d, masodik:%d\n",elseo, masodik);  
}
```

```
int elseo = 5;  
int masodik = 10;  
  
printf("%d, %d \n",elseo,masodik);  
printf("%s\n", "A csere után: ");  
  
elseo = elseo^masodik; //0000 1111 = 15 - elseo  
masodik=elseo^masodik;//0000 0101 = 5 - masodik  
elseo=elseo^masodik;//0000 1010 = 10 - elseo  
  
printf("%d, %d \n",elseo,masodik);
```

```
int elso = 5;
int masodik = 10;

printf("%d, %d \n",elso,masodik);
printf("%s\n", "A csere után:");

elso = elso*masodik; //elso = 5*10=50;
masodik=elso/masodik;//masodik = 50/10=5;
elso=elso/masodik;//elso = 50/5=10;

printf("%d, %d \n",elso,masodik);
```

Minden programot egyszerű matematikai műveletekkel , bármiféle logikai utasítás vagy kifejezés nélkül hajtottuk végre. Először bekérünk a felhasználótól 2 tetszőleges számot , majd 3 lépésekben felcseréljük őket. Végül kiiratjuk egy egyszerű tetszőleges kiiratással.

A BogoMips

Mi az a BogoMips? A Bogomiaz a processzor sebességét hivatott mutatott mérőszám.

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

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

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);
```

A program elején includoljuk a stdio és time könyvtárakat. Majd létrehozzuk a delay függvényt , ami i=0-tól "elszámol" loops-ig.Deklaráljuk a loops_per_sec és a ticks változókat. A fflush függvény miatt a megelelőző sor print f kiiratásnak minden féle képpen végbe kell mennie. Következik egy while ciklus, ami addig tolja egyel balra a loops_per_sec értékét amíg az egész sor 0 nem lesz. A clock () függvény eredményét megkapja a ticks változó. Meghívjuk a delay függvényt , megadjuk

neki a loops_per_sec paraméterként. A ticks új értékét kap : clock függvény értékéből kivonjuk a ticks értékét. Majd kiiratunk printf segítségével.

```
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;
}
```

If feltétel vizsgálat : Ha a ticks értéke nagyobb vagy egyenlő mint CLOCKS_PER_SEC-é, a loops_per_sec értékét módosítjuk. Majd ezt a számot kiíratjuk. Ha a feltétel vizsgálat hamis akkor a programunk -1-gyel tér vissza.

2.4 Labdapattogás

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

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

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/turing/pat.c>
- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/turing/patif.c>

if-es megoldás:

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

int
main ( void )
{

WINDOW *ablak;
ablak = initscr ();

int x = 0;
```

```
int y = 0;  
  
int xnov = 1;  
int ynov = 1;  
  
int mx;  
int my;
```

A programunk legelején includoljuk a `stdio`, `curses` és `unistd` header fájlokat, könyvtárakat. A `stdio` teszi lehetővé számunkra az adat outputra történő kiiratását. A többi header fájlról, könyvtárról is ejtsünk pár szót : `curses` segítségével tudjuk meghívni a `initscr` függvényt, míg a `unistd`-re a `usleep` függvény miatt van szükségünk. Deklaráljuk az `ablak` nevű mutatót, majd inicializáljuk az `initscr` segítségével. Deklaráljuk még a azokat a változókat amik az ablak méretét tárolják, illetve a `xnov` és `ynov` változókat, ezek a labda pozíció változtatására vonatkozó adatokat tárolják.

```
for ( ; ; )  
{  
    getmaxyx ( ablak, my , mx );  
  
    mvprintw ( y, x, "O" );  
  
    refresh ();  
    usleep ( 100000 );
```

Következik egy végtelen ciklus. A `getmaxyx` tárolja az ablak méreteit, és a labdánk kiindulási pontját. A `mvprintw` írja ki folyton folyvást az `x` és `y` koordinátára azt az értéket amit harmadik paraméterként megadtunk ("O"). A `refresh` frissíti a képernyőt, hogy minden az aktuális, friss képet láthassuk a képernyőnkön. A `usleep(x)` felel a labdánk lassításáért, a zárójelben megadott érték tetszőleges mikroszekundummal lassítja a program futását.

```
x = x + xnov;  
y = y + ynov;  
  
if ( x>=mx-1 ) {  
    xnov = xnov * -1;  
}  
if ( x<=0 ) {  
    xnov = xnov * -1;  
}  
if ( y<=0 ) {  
    ynov = ynov * -1;  
}  
if ( y>=my-1 ) {  
    ynov = ynov * -1;  
}
```

Ebben a kód részletben kerül sor a labdánk mozgatására, egyszerüen növeljük az `x` és `y` értékeit. Következik 4 feltétel vizsgálat amelyek lehetővé teszik, hogy a labdánk visszapattanjon a falról, olyan módon, hogy -1-el szorozza meg azokat a változókat amik a labda pozíójának a módosításáért felelnek.

if nélküli megoldás:

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

int main (void)
{
    int xj = 0, xk = 0, yj = 0, yk = 0;
    int mx = 160, my = 48;

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

A program elején includoljuk az előző feladatban már kifejtett header fájlokat és az `stdlib.h`-t , ami az `abs()` függvény használatához lesz szükségünk. A main-ben megadjuk a szükséges változókat , illetve az ablak méreteit amik a maximum koordináták. Deklaráljuk az `ablak` nevű pointert az `initscr` segítségével.

```
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), "0");

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

Következik egy végtelen ciklus. A ciklusunk első 4 sorában definiáljuk a labdánk helyzetének meghatározására szolgáló változókat. Következik a `clear()` függvény ami a képernyő letisztítására szolgál. A `mvprintw` segítségével kiiratjuk a pálya alját és tetejét , amik jelen esetben szaggatot vonalak. Majd kiiratjuk magát a labdát is. Majd az előző if-es feladatban megismert `refresh()` és `usleep` függvényel találkozunk, annyi különbséggel , hogy az `usleep` értékének 1 tizedmsp lett megadva.

2.5 Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/sz%C3%B3hossz.c>
- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/turing/bogomips.c>

```
#include "std_lib_facilities.h"

int main()
{
    int szam=1;
    int db=0;

    while(szam!=0) {
        szam = szam<<1;
        ++db;
    }
    cout <<db<<"\n";
}
```

Deklarálunk két Integer típusú változót `szam` névvel 1-es értéket adva neki , majd `db` néven 0-ás értéket adva neki. Elindítunk egy `while` ciklust , a ciklus akkor fejezi be a futást , ha a `szam` nem lesz 0. Ezután a left shifting-et fogjuk használni, ami annyit tesz , hogy a `i` változó értéke minden egyes ciklus lefutásának a végén egyel balra ugrik.

2.6 Hello, Google!

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/pagerank.c>

A PageRank a Google által, a kereséskor alkalmazott algoritmus. Ez alapján állít fel a google egy úgynevezett fontossági sorrendet a honlapok között. Minél nagyobb az adott honlap PageRank értéke annál előkelőbb helyen helyezkedik el ezen a bizonyos rangsorban.

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

Legelső sorban include-olunk kell az `stdio` könyvtárat az eredmények képernyőn való megjelenítéséhez és a `math` könyvtárat a matematikai műveletek elvégzéséhez

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

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

Deklaráljuk a `kiir` függvényt. Deklaráljuk az `int` típusú `i` változót. A `for` ciklus `i=0`-tól `db`-ig megy, és kiírja a sorszámot, illetve a tömb `i`-edik elemét. Mivel a függvényünk `void` típusú, ezért a visszatérési értéke `null` (nincs)

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

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

Deklaráljuk a `tavolsag` függvény, két `double` típusú tömbet adunk neki paraméterként, `PR` és `PRv` névvel, illetve deklarálunk egy `Integer` típusú, `n` nevű változót. Deklaráljuk az `i` `Integer` tipussal és az `osszeg` `double` típusú változót. A `for` ciklus `i=0`-tól `n`-ig megy, és a `for` ciklus `i=0`-tól `db`-ig megy, és az `osszeg` változó értéke a `PRv` tömb `i`-edik eleme és a `PR` tömb `i`-edik elemének a különbéségének a szorzata. Eredményül a függvény az `sqrt` functionnel az `osszeg` változó négyzetgyökét adja vissza.

```
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;
```

Létrehozzuk az `L` egy `double` típusú kétdimenziós, 4 soros, 4 oszlopos és a `PR` és `PRv` tömböket. Ezen kívül deklaráljuk `i` és `j`, `int` típusú változók is.

```
for(;;)
{
```

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

}
```

Létrehozunk egy végtelen ciklust. Ezt követi egy belső `for` ciklus, ami 0-tól 4-ig fut, a `PR` tömb i-edik elemét 0.0-ra állítjuk be. Majd indítunk még egy ciklust, ami szintén 0-tól 4-ig megy. A `PR` tömb i-edik elemének értékét növeljük az `L` tömb i-edik, j-edik elemének és a `PRv` tömb j-edik elemének szorzatával.

```
if(tavolsag(PR, PRv, 4)<0.00001)
    break;

for(i=0; i<4; ++i)
    PRV[i]=PR[i];
}
kiir(PR, 4);
```

Feltételvizsgálat : Ha a `PR`, `PRv` és 4 paraméterekkel ellátott `tavolsag` függvény értéke kisebb, mint 0.00001, meghívódik a `kiir` függvény két paraméterrel: `PR`-rel és a 4-el. Ha a feltételvizsgálat hamis, elindul egy `for` ciklus 0-tól 4-ig, ami futása során `PRv` i-edik elemhelyére `PR` i-edik eleme kerül. Végül kiíratjuk.

2.7 100 éves a Brun tétele

Írj R szimulációt a Brun tétele demonstrálására!

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

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

Ez a szimuláció a Brun tétele ábrázolja szemléletesen. Tétel szerint a 2 különbségű prímek reciprokainak összege egy konstanshoz közelít. A primes a prímekből álló vektor, a diff pedig ezeknek különbségeit számolja ki, majd az idx az ikerprímek sorszámait tárolja, amik értékei a t1- és t2primes-ban tárolódnak. Aztán az rt1plusrt2 ezek reciprokosszegeit veszi, majd a függvény visszatéríti a reciprokosszegeket összegeit.

Ez után maga a szimuláció ábrázolja ezeket az értékeket, bizonyos maximális értékekig vett prímekkel vett függvényeredményként, és valóban úgy tűnik, egy bizonyos számhoz konvergálnak.

forrás : <https://hu.wikipedia.org/wiki/Brun-t%C3%A9tel>

2.8 A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

A probléma : Az egyik mögött rejlik a főnyeremény, egy autó, a másik kettő mögött pedig nincs semmi. Ön a második ajtót választja. Ekkor a műsorvezető kinyitja a másik két ajtó közül az egyiket, mondjuk a harmadikat. ami üres. Majd felteszi a kérdést, hogy szeretne-e változtatni és esetleg másik ajtót választani?

A válasz a kérdésre: IGEN, érdemes változtatni.

```
kiserletek_szama=100
kiserlet = sample(1:3, kiserletek_szama, replace=TRUE)
jatekos = sample(1:3, kiserletek_szama, replace=TRUE)
musorvezeto=vector(length = kiserletek_szama)
```

Deklaráljuk kiserletek_szama, a kiserlet és jatekos változókat a sample function-nel, amely minden esetben kiserletek_szama darabszámú (vagyis 100), 1 és 3 közötti, ismétlődhető véletlen számot generál. A kiserlet változó azt fogja tárolni, hogy éppen melyik ajtó mögött van a nyeremény, míg a jatekos azt, hogy melyik ajtót választottuk. A musorvezeto egy vector lesz, aminek mérete a kiserletek_szama.

```
for (i in 1:kiserletek_szama) {

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

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

  }else{

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

  }

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

}
```

Egy ciklust hívunk meg, amelyben az 1-től indul egészen a kiserletek_szama által tárolt értéig. majd feltétel vizsgálat : ha a kiserlet i-edik eleme megegyezik a jatekos i-edik elemével, akkor a mibol vektor értéke egyenlő lesz az 1,2,3 számok és a kiserlet i-edik elemének "különbségével". Lényegében egy olyan sorszám lesz ez esetben a mibol értéke, amely mögött biztosan nincs már a nyeremény, ugyanis a játékos elsőre beletrafált, melyik mögött van a nyeremény, igaz, ő még nem tud róla. Ha ez nem teljesül, jön az else ág, amely lényegében ugyanez fordítva. A mibol értéke ugyanúgy az az érték lesz, ami az 1,2,3 számsorban megtalálható, azonban nem i-edik eleme sem a kiserletnek, sem a jatekos i-edik eleme, ugyanis a műsorvezető csak olyan ajtót nyithat ki, amelyet a játékos még nem választott, és nyeremény sincs mögötte. A musorvezeto i-edik eleme pedig a mibol azon indexű eleme lesz, melyet az 1: mibol hossza intervallumból képzett, egy darab olyan szám lesz, melyet az előző feltételeknek megfelel, elemei a mibol elemei lesznek.

```
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)
```

A `which` függvénytel megkapjuk azt az indexet, ahol a `kiserlet` és a `jatekos` értékek megegyeznek, ez lesz a `nemvaltoztatesnyer` értéke. Ez az az eset, amikor a játékos elsőre jól tippel és kitart a tippje mellett. A `valtoztat` egy `vector` lesz, aminek mérete a `kiserletek_szamaval` egyenlő. A következő cikluson belül történnek még érdekkességek: a `holvalt` értéke az lesz, ami 1,2,3 számlában megtalálható, és cserébe sem a `musorvezeto`, sem pedig a `jatekos` i-edik eleme, vagyis se nem a játékos előzőleg, se nem a műsorvezető nem választotta ki még. A `valtoztat` i-edik eleme a `holvalt` azon indexű eleme lesz, amely indexet az 1: 'holvalt hossza' (`length(holvalt)`) intervallumból kapunk, elemei az előző feltételeknek megfelel, vagyis a `holvalt` elemei. `valtoztatesnyer` értékét egyenlővé tesszük azzal az indexsel, amely indexen `kiserlet` és a `valtoztat` értékek megegyeznek.

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

Kiiratjuk az eredményt , amiből megállapíthatjuk , hogy tényleg nagyon esély van a győzelmre , ha új ajtót választunk.

Chapter 3

Helló, Chomsky!

3.1 Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/decitounar.c>

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

int main(){
    int decimal;
    string unary;
    cin >> decimal;
    while(decimal>0) {
        decimal--;
        unary+="1";
    }
    cout << unary;
}
```

Decimális számrendszerből vagy magyarul 10-es számrendszerből váltunk át egy számot egyes számrendszerbe.

Az 1-es számrendszerben minden össze 1 számjegyből állnak a számok, annyi darab 1-es áll a helyiértékek helyén amennyi a számunk valós értéke.

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

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

Megoldás videó:

Megoldás forrása:

Vannak nyelven belüli szimbólumok, ezekhez a nyelvi szimbólumokhoz vannak hozzárendelési szabályok, melyek ezekhez az eredeti szimbólumtól eltérő szimbólumokat vagy szimbólumsorozatokat rendel.

A szimbólumoknak két fő típusa van: terminális és nem terminális.

A terminális szimbólum : amihez nem tartozik hozzárendelési szabály, tehát atomi, nem bontható tovább.

A nem terminális szimbólum : ezekhez a szimbólumokhoz tartozik hozzárendelés.

Ha van olyan grammatika ami őt generálja és szintén az akkor az adott nyelv környezet független.

1. $S \rightarrow aBC$

2. $S \rightarrow aSBC$

3. $CB \rightarrow CZ$

4. $CZ \rightarrow WZ$

5. $WZ \rightarrow WC$

6. $WC \rightarrow BC$

7. $aB \rightarrow ab$

8. $bB \rightarrow bb$

9. $b \rightarrow bc$

10. $cC \rightarrow cc$

3.3 Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/hivnyelv.c>

Tanulságok, tapasztalatok, magyarázat...

```
#include <stdio.h>

int main()
{
    for(int i=0; i<11; ++i)
    {
        printf("%d ", i);
    }

    return 0;
}
```

A program main részében egy for ciklus áll, ami 1-től 10-ig fut, minden futáskor a ciklusváltozó értékét növeljük egyel. Majd ezt az értéket kiiratjuk.

Nem sikerül a gcc-nek lefordítania a programot a C89-es szabvánnyal, ugyanis a változó ciklusfejben való deklarálása nem volt engedélyezett C89-ben. A fordító hibaüzenetben segítséget nyújt abbah, hogyan érdemes kijavítanunk a programunkat, hogy sikeres jussunk. Ez a for ciklus a legszemléletebb példa a különbségek bemutatására.

3.4 Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/valosszamok.l>

A Lex egy program, ami előre definiált szabályok alapján C kódot állít elő számunkra. A mi feladatunk ezeket a szabályokat definiálni.

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

Az egyes részeket százalékjelek választják el egymástól. Az első rész klasszik C kód, a definíciós rész, itt deklaráljuk változókat, lásd : számláló. Include-oljuk stdio függvénykönyvtárat

```
digit [0-9]  
%%  
{digit}*(\.{digit}+)? {++realnumbers;  
    printf("[realnum=%s %f]", yytext, atof(yytext));}  
%%
```

Difiniáljuk a szabályokat. Számjegyeinknek a digit elnevezést adtuk, későbbiekben ezen a néven fogunk rájuk hivatkozni.

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

A dupla százalékjelek közöt definiáljuk a szabájokat. Definiáljuk azokat a számokat, amiket a program keresni fog az inputon. Ha találtunk a szabályunknak megfelelő számot, a számlálónk értékét növeljük és egy egyszerű printf függvénytel kiiratjuk magát a számot, ami a yytext változóban tárolódik.

Ez azonban egy string, amit double típusúvá szeretnénk átalakítani, ehhez használjuk az `atof` függvényt használjuk.

Azokat a számokat amiket keresünk, formailag a következők: Pont előtti rész: nulla vagy több darab számjegy. A pont utáni rész: 1 vagy több darab számjegyet tartalmaz.

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

A `main`ben meghívásra kerül a `yylex` függvény.

3.5 I33t.I

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/l33t.l>

```
%{
#include <string.h>
%}
```

Az első százalékjelek közötti rész a definíciós rész. A feladat megoldásához include-oljuk `string` függvénykönyvtárat használjuk, mert a feladat során Stringek-re lesz szükség a megoldáshoz.

```
%%
[a-zA-Z]
%}
```

A dupla százalékjellel definiáljuk a szabályokat. Szögletes zárójelek között megadjuk, hogy mikkel kell majd dolgoznia, vagyis mik lesznek az átalakítandó karakterek. Esetünkben ezek az ábécé kis- és nagybetűi lesznek.

```
switch(yytext[0])
{
    case 'a' :
        printf("4");
        break;

    case 'b' :
        printf("8");
        break;
```

```
case 'e' :
    printf("3");
    break;

case 'g' :
    printf("9");
    break;

case 'l' :
    printf("1");
    break;

case 's' :
    printf("5");
    break;

case 'z' :
    printf("2");
    break;

case 'o' :
    printf("0");
    break;

default :
    printf("%s", yytext);
    break;
}
}

%%

%
```

Egy switch szerkezetben megvizsgáljuk az input string karaktereit. Az `yytext` egy egyelemű karaktertömböt jelöl, aminek első tagjára kíváncsiak. A következőkben minden case-ben egy-egy Leet-beli számot rendelünk a betűkhöz, majd `printf`-fel kiíratjuk ezt.

```
int
main()
{
yylex();
return 0;
}
```

Létrehozzuk a "main"-ünket és meghívjuk az "yylex" függvényt , ami a programunk második fő része.

3.6 A források olvasása

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

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

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



Bugok

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

i.

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

Ha mindenet jól csináltunk , akkor a programunk nem fog reagálni a billentyűzetről érkező jelekre.

ii.

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

A ciklusunk i=0-ról indul , i-nek az értékét addig növelem amíg az kisebb, mint 5 , magyarul 4-ig.

iii.

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

A ciklusunk i=0-ról indul , i-nek az értékét addig növelem amíg az kisebb, mint 5 , magyarul 4-ig.

iv.

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

a tömb I-edik eleme tárolja az i változó aktuális értékét , az i érték , ameddig a feltétel igaz , növekszik.

v.

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

vi.

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

A kiiratásban kétszer hívjuk meg az f függvény ami két paraméterrel rendelkezik. Paraméter átadáskor növeljük először a második paramétert , majd az első paramétert.

vii.

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

Kiiratjuk az f függvényt "a" paraméterrel és az "a" változóval.

viii.

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

Kiiratjuk az f függvényt "a" paraméterrel és annak a címével és az "a" változóval.

Megoldás forrása:

Megoldás forrása:

- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/forrasok_olvasasa.c
- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/forrasok_olvasasa2.c

stb. stb.

- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/forrasok_olvasasa9.c

Tanulságok, tapasztalatok, magyarázat...

3.7 Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

N interpretáció mellett a írt formulák:

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

vagyis

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

Prímek száma végtelen.

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

vagyis

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

Ikerprímek száma végtelen.

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

vagyis

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

Prímek száma véges.

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

vagyis

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

Prímek száma véges.

3.8 Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- `int a;`
egész
- `int *b = &a;`
egészre mutató mutató
- `int &r = a;`

egész referenciaját

- ```
int c[5];
```

egészek tömbje

- ```
int (&tr)[5] = c;
```

egészek tömbjének referenciaja

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

egészre mutató mutatók tömbje

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

egészre mutató mutatót visszaadó függvény

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

egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# Chapter 4

## Helló, Caesar!

### 4.1 int \*\*\* háromszögmátrix

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/ceasar%20/haromszogmatrix.c>

Tutor : Kincs Ákos <https://gitlab.com/kincsa>

Mi is az a háromszögmátrix? A háromszögmátrix egy kvadratikus (négyzetes) mátrix, sorainak és oszlopainak a száma megegyezik. Két fajta létezik belőle: alsó-, és felső háromszögmátrix. Az alsó háromszögmátrix főátlójá felett csupa nulla érték szerepel, míg a felsőnek a főátló alatti elemei mind nullák.

A mi programunk egy ilyen mátrixot fog készíteni: soook

A háromszögmátrix tartalmáról tudjuk, hogy bizonyos pozíciókon nulla áll, így felesleges minden elemét, vagyis  $n^*n$  darab értéket egyesével tárolni, elég csak  $n^*(n+1)/2$  darab elemet. A háromszögmátrixokat egy vektorba, vagyis egy dimenziós tömbbe lehet leképezni. A felső háromszögmátrixokat oszlopfolytonosan, míg az alsó háromszögmátrixokat sorfolytonosan szokás leképezni.

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

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

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

Includeoljuk a szükséges könyvtárakat, az stdio-val már többször találkoztunk, illetve az stdlib.h-ra is szükségünk lesz. Deklarálunk egy egészet nr néven, Ő a sorok illetve oszlopok számát tartalmazza. Deklaráljuk továbbá a \*\*tm mutatót is, ami egy pointerekre mutató pointer, majd kiíratjuk a címét a memóriában. Az eredményt hexadecimális számrendszerben kapjuk.

```

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

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

```

If feltétel vizsgálat : egy double\* méretének nr-szeresét allokáljuk, vagyis lefoglalunk a memóriában, double \*\*-nak, vagyis double\*-okra mutató mutatónak. A malloc void\*-ot ad vissza alapból, de ezt típuskényszerítéssel meg tudjuk változtatni. Amennyiben ez NULL érték, nem sikerült a helyfoglalás, így a program hibával tér vissza. Kiírjuk a lefoglalt tárterület címét.

```

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

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

```

Egy for ciklussal megyünk nr-szer, és a következő hajtjuk végre: lefoglalunk memóriát a double méretének i+1-szeresére. Erre double\* -ot kényszerítünk, és rendre a tm[] i-edik helyének adjuk értékül, vagyis ezekre a mutatókra fog mutatni tm[i]. A mutatót lényegében tömbként kezeljük. Ismét amennyiben ez NULL érték, nem sikerült a helyfoglalás, így a program hibával tér vissza. Kiírjuk az első mutató memóriacímét.

```

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

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

```

Az első for-ciklusban feltöljük a mátrixot tartalommal. I-vel és j-vel végigmegyünk a két dimenziót, majd az  $i * (i + 1) / 2 + j$  értéket tesszük az adott helyre, vagyis szimplán sorba lesznek benne a számok nullától.

A második for ciklussal kiírjuk a tömbünk elemeit soronként.

```

tm[3][0] = 42.0;
(*tm + 3)[1] = 43.0;
*(tm[3] + 2) = 44.0;
*(*tm + 3) + 3) = 45.0;

for (int i = 0; i < nr; ++i)

```

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

Négyszer megváltoztatjuk a mátrix tartalmát. Az első a legérhetőbb, a negyedik sor első eleme 42 lesz. Mindig a negyedik sor elemeit változtatjuk. A második alkalommal a második elem 43 lesz, majd a harmadik 44, illetve a negyedik 45.

For ciklussal kiírjuk a mátrix tartalmát.

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

free (tm);

return 0;
}
```

Felszabadítjuk a lefoglalt memóriát, először a for ciklusban a tm[]-ben levő double\*-okat, majd a for után magát a double\*\* tm-et is felszabadítjuk, majd a program hibamentesen tér vissza.

## 4.2 C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/ceasar%20/e.c>

Az EXOR titkosítás lényegében nem más , mint a következő:

Van egy szöveged, amit titkosítani szeretnél és van egy megoldó kulcsod a titkosításhoz. A titkosítani kívánt szövegen a kulccsal exor műveletet hajtasz végre, ennek hatására a szövegből olvashatatlan dolgot hozol létre. A szépsége abban rejlik, hogy ezen az olvashatatlan szövegen újra alkamlazva a műveletet, ugyanazzal a megoldó kulccsal, visszakapjuk az eredeti szöveget , immáron olvasható módon. Magyarán az EXOR titkosítás lényege röviden és egyszerüen összefoglalva : az olvashatatlan szöveget csak akkor olvashatod el ha tudod a kulcsot hozzá.

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

A kód elején deklaráljuk azokat a könyvtárakat amiket szeretnénk használni. Deklaráljuk a unistd.h a read() és write() miatt , majd deklaráljuk a string.h (strlen(),strncpy) String függvények miatt.Mindezek mellett deklarálásra kerül a stdio.h könyvtár is.

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

A kódunk legeslegelején megadunk két konstans értéket, a kulcs maximális méretét(MAX\_KULCS) néven emellett az ideiglenes tároló méretét(BUFFER\_MERET) néven.

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

Létrehozzuk a main() metódust. A main() paraméterei a következők : Az első az argc az argumentumok számát tárolja, a második pedig az argv az argumentumokat tároló vektor. Ezeknek az argumentumoknak a jelszó megadásánál lesz jelentős szerepe.

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

Létrehozunk 2 char típusú tömböt (azaz Stringet). Az egyiket MAX\_KULCS, a másikat BUFFER\_MERET mérettel.

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

Deklarálunk és inicializálunk 2 egész típusú változót amiken 0 értéket adunk. Az egyikben azt tároljuk, hogy a kulcs hanyadik indexénél járunk, a másikban pedig eltároljuk a bufferbe olvasott bajtok számát.

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

Ebben a kódrészletben létrehozunk egy egész típusú változót, amiben a kulcs tényleges méretét fogjuk eltárolni. Ezt úgy kapjuk meg, hogy az strlen() függvényt alkalmazzuk az argumentum vektor 2. elemére(argv[1]), magyarul a megadott jelszóra. A strncpy() metódust alkalmazva a már létrehozott kulcs char tömbbe másoljuk a program 2. argumentumát, magyarul a kulcsot. A függvény paraméterként a következőket várja: hova másoljon, mit másoljon és hogy hány bajtot másoljon. Utóbbiként a MAX\_KULCS-ot adjuk meg.

```
while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
```

```
{
```

Következik egy while ciklus aminek a feltétele a következő : az olvasott\_bajtok változóban tároljuk a read() által visszaadott értéket, ami egészen addig pozitív lesz amíg a függvény sikeresen lefut, tehát amíg van mit beolvasni a programunk futni fog. Visszatérési értékként pedig nem csak egy egyszerű pozitív számot ad, hanem a beolvasni sikerült bajtok számát is.

```
for (int i = 0; i < olvasott_bajtok; ++i)
```

```
{
```

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

```
}
```

A while cikluson belül található egy for ciklus, ami végig megy a beolvasott szövegen. Miközben végig megy, az aktuális elem és a kulcs kulcs\_index-edik eleme között exor műveletet hajt végre. ezt követően a kulcs\_index-et növeljük egyel és elosztjuk az így kapott értéket kulcs\_meret-tel a maradékát véve, ezzel biztosítjuk azt , hogy az érték sose fogja túllépni a kulcs méretén.

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

}
```

A write() függvény segítségével kiírjuk a standard outputra(1) a buffer tartalmát, ami olvasott\_bajtok méretű. Ezzel amit beolvastunk, exoroztunk, most kiíratjuk. A ciklus így megy tovább, amíg az egész be-menet feldolgozásra nem kerül. A program teljes lefutása után a kimenet a titkosított vagy éppen már a visszakódolt szöveg lesz.

## 4.3 Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html> 1.12. példája - Titkosítás kizáró vaggal és <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/ceasar%20/ExorTitkos%C3%A1r.java>

Tanulságok, tapasztalatok, magyarázat...

```
public class ExorTitkosító {

 public ExorTitkosító(String kulcssSzöveg,
 java.io.InputStream bejövőCsatorna,
 java.io.OutputStream kimenőCsatorna)
```

C programozási nyelvtől eltérően a programunk elején nincs szükségünkünk könyvtárak includolásásra. Létrehozzuk az ExorTitkosító objektumunkat három paraméterrel : kulcsként használt szöveg , input , output.

```
throws java.io.IOException {

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

Definiálunk kettő byte típusú tömböt kulcs és buffer, kulcs-ba kerül majd a kulcsszöveg , buffer-be megadjuk a méretét. A kulcs indexelésének és a beolvasott bájtok számolására létrehozunk két integer típusú változót , úgy nevezett számlálót

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

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

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

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

Egy while ciklus fut addig , amíg van beolvasandó bájtunk. Azokat a bájtokat amiket nem tudunk beolvasni , úgy nevezett olvashatatlan szöveg megegyezik a buffer tömb i-edik elemével , ami a buffer i-edik eleme és az kulcs tömb kulcs\_index-edik elemének vett EXOR művelet eredménye. A művelet eredménye byte típusú. Majd növeljük a kulcs\_index értékét. Ezek után kiiratjuk a write függvényel a buffer tömb elemeit nullától az olvasottBájtok-ig.

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

 try {

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

 } catch(java.io.IOException e) {

 e.printStackTrace();
 }
}
```

A Try - Catch -en belül példányosítunk. Létrehozunk egy új ExorTitkosítót ami a parancsori argumentumként kapott szöveget fogja átalakítani. A catch-en belül meghívjuk a printStackTrace-t , ami visszajelzést ad arról , hogy mi a hiba és hol található.

## 4.4 C EXOR törő

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

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/ceasar%20/t.c>

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

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

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

 return (double) titkos_meret / sz;
}
```

Kiszámítjuk az átlagos szóhosszt. Először megszámoljuk a szövegen a szóközöket ezután visszaosztjuk vele az összes karakterek számát.

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

 double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}
```

A tiszta szöveg nagy valószínűséggel tartalmaz olyan gyakori magyar szavakat mint: nem, hogy, ha, az. Az átlagos szóhossz segítségével csökkenthetjük a végrehajtandó törések számát is.

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

 int kulcs_index = 0;

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

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

 }

}

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

```
 exor (kulcs, kulcs_meret, titkos, titkos_meret);

 return tiszta_lehet (titkos, titkos_meret);

}
```

Bájtonként végrehajtjuk az EXOR-t. A % segítségével a kulcs nem lépi át a keresett hoszt.

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

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

 for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
 titkos[p - titkos + i] = '\0';

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

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

 exor (kulcs, KULCS_MERET, titkos, p - titkos);
 }
}
```

```
 }

 return 0;
}
```

A while ciklus addig fog futni, amíg van bemenet. Egymásba ágyazott for ciklusokkal előállítjuk az összes lehetséges kulcsot majd ha a kulcsok előálltak ki is próbáljuk őket. A végén újra exor-ozunk.

## 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)

Ez az R szimuláció egy neurális háló használatával oldja meg a leggyakrabbi logikai műveleteket.

OR vagy VAGY: ha minden két bit 0, akkor az or is 0, egyébként pedig 1.

AND vagy ÉS: csak akkor 1, ha minden két bit 1, egyébként 0.

EXOR vagy KIZÁRÓ VAGY.

A tanuló algoritmust "megetetjük" ezen műveletek két argumentumával és ezek kimeneteleivel, aztán ezen halmazok alapján az idegháló megpróbálja rekreálni azokat.

Ez láthatóan az or és and műveletnél rendkívül pontos eredményre vezet, tehát konklúzióként levonható, hogy a program megtanulta a műveleteket.

## 4.6 Hiba-visszaterjesztéses perceptron

C++

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

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

Tanulságok, tapasztalatok, magyarázat...

A Neurális OR, AND és EXOR kapu feladatnál már találkozhattunk a neuron és a gépi tanulás fogalmával. A perceptron leegyszerűsítve nem más mint ezen neuron mesterséges intelligenciában használt változata. Tanulásra képes, a bemenő 0-k és 1-esek sorozatából mintákat tanul meg és súlyozott összegzést végez. A következő feladat során egy ilyen perceptron fogunk elkészíteni aminek alapvetően a feladata az, hogy a mandelbrot.cpp programunk által létrehozott Mandelbrot-halmazt ábrázoló PNG kép egyik színkódját vegye és az a színkód legyen a többrétegű neurális háló inputja. !!!!! ÁTFOGALMAZNI!!!!

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

Includoljuk aziostream, az mlp és a png++/png könyvtárakat. A mlp könyvtárra azért van szükségünk mert több rétegű percetront fogunk létrehozni. A png++/png könyvtár a PNG kiterjesztésű képállományokkal való munkát teszi számunkra lehetővé.

```
using namespace std;

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);
```

A main függvényünk első sorában megmondjuk , hogy a képállományunk beolvasása az 1es parancssori argumentum alapján történik. Eltároljuk egy segédváltozóban a kép méretét a get\_width és a get\_height szorzatát , majd létrehozzuk a percetonunkat a new operátor segítségével.

```
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;

delete p;
delete [] image;
```

Az egyik for ciklussal végig megyünk a kép szélességét alkotó pontokon , majd a másik for ciklussal végig megyünk a magasságát alkotó pontokon. Az image -ben fogjuk tárolni a vörös képpontokat. A value értéke adja meg a Percetron image-ra történő meghívását, ami egy double típusú változót tárol. Kiiratjuk és töröljük azokat az elemeket már nem használunk, így az addig lefoglalt memória egységeket újra használhatóvá tettük.

Fordítjuk és futtatjuk a képen látható módon:

# Chapter 5

## Helló, Mandelbrot!

### 5.1 A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: [https://progpater.blog.hu/2011/03/26/kepes\\_egypercesek](https://progpater.blog.hu/2011/03/26/kepes_egypercesek) <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/mandelpng.c++>

Tutor : Kincs Ákos <https://gitlab.com/kincsa>

Benoît Mandelbrot egy Lengyel származású matematikus, aki a 20. században élte életét és csinálta munkásságát, az Ő nevéhez fűződik többek között a Mandelbrot-halmaz fogalom is, amit a komplex számsíkon ábrázolunk és amelyet Mandelbrot fedezett fel az 1970-es évek végén. A Mandelbrot-halmazok ábrázolására a komplex számokat használjuk. A komplex számok egyik közös jellemzője, hogy abban az esetben tartoznak Mandelbrot halmaz elemei közé, ha őket  $x_{n+1} = x_n^2 + c$  sorozatba behelyettesítve egy konvergens sorozatot kapunk. Az előzőekben említett halmaz ábrázolása egy fraktál alakzatot eredményez, amely egy olyan alakzat, aminek körfonalai nem egyenes, hanem "kitülemkedések" figyelhetőek meg rajta illetve ezen alakzatok hasonlítanak egymásra. Forrás : [https://hu.wikipedia.org/wiki/Komplex\\_sz%C3%A1mok](https://hu.wikipedia.org/wiki/Komplex_sz%C3%A1mok)

Ebben a feladatban egy olyan programot fogunk vizsgálni, ami egy ilyen Mandelbrot-halmazt állít elő. A program nem más mint a következő:

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

Mindenek előtt A programunk legelején a png++/png és a iostream könyvtárakat inkludáljuk, a png++/png könyvtár lehetővé teszi számunkra a PNG kiterjesztésű képallokányokkal való munkát. Ez nagyon fontos számunkra , ugyanis egy PNG képet szeretnénk megrajzolatni programunkkal. Amennyiben a png++/png könyvtár még nincs telepítve a számítógépünkre , a feladatunk megoldásához feltétlenül szükséges ezt telepíteni.

```
int main (int argc, char *argv[])
{
 if (argc != 2) {
 std::cout << "Használat: ./mandelpng fajlnev";
 return -1;
 }
```

Ebben a kódrészletben parancssori argumentumként adjuk meg, hogy milyen fájlba legyen mentve a képünk, és ha ez a futtatás során elmarad, felhívjuk a felhasználó figyelmét a helyes futtatási módra.

```
// számítás adatai
double a = -2.0, b = .7, c = -1.35, d = 1.35;
int szelesseg = 600, magassag = 600, iteraciosHatar = 1000;

// png-t készítünk a png++ csomaggal
png::image<png::rgb_pixel> kep (szelesseg, magassag);

// a számítás
double dx = (b-a)/szelesseg;
double dy = (d-c)/magassag;
double reC, imC, rez, imZ, ujrez, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
std::cout << "Szamitas";
```

Létrehozzuk a számolásunkat szolgáló változókat. A double típusú változókkal a számsík határait adjuk meg az X és az Y tengelyen, majd újabb double típusú változók kerülnek létrehozásra , amik a kép szélességének és magasságának tárolására hivatott változók, illetve egy int típusú változó, ami az iterációk darabszámának felső korlátját jelentő érték.

Létrehozzuk a képünket, aminek mérete: szelesseg x magassag, Ez a kép jelenleg még egy sima, üres kép. A dx és dy változókat a számsíkon való lépésköz meghatározására létrehozzuk. Majd létrehozzuk az im és re előtagú változókat Z és C változókhöz, amelyek az adott komplex szám (C vagy Z) valós és képzetes részét jelölik.

Létrehozunk egy int típusú változót 0 értékkel , ami az iterációk számának nyilvántartására szolgál.

```
// Végigzongorázzuk a szélesség x magasság rácson:
for (int j=0; j<magassag; ++j) {
 //sor = j;
 for (int k=0; k<szelesseg; ++k) {
 // c = (reC, imC) a racs csomópontjainak
 // megfelelo komplex szam
 reC = a+k*dx;
 imC = d-j*dy;
 // z_0 = 0 = (rez, imZ)
 rez = 0;
 imZ = 0;
 iteracio = 0;
 }
}
```

Két for ciklussal végigmegyünk a szelesseg x magassag "rácson", majd inicializáljuk C valós és képzetes részeit tartalmazó változókat, ezek után ugyanezt tesszük Z esetében is. Létrehozásra kerül az iterációk számát tartalmazó változó .

```
while (rez*rez + imZ*imZ < 4 && iteracio < iteraciosHatar) {
 // z_{n+1} = z_n * z_n + c
 ujrez = rez*rez - imZ*imZ + reC;
```

```
ujimZ = 2*reZ*imZ + imC;
reZ = ujreZ;
imZ = ujimZ;

++iteracio;
```

A következőkben megvizsgáljuk egy while ciklusban , hogy  $z_n$  abszolútértéke kisebb-e mint 2 és, hogy elértek-e az iterációs határt. Ha a ciklusfejben található feltétel teljesül, akkor módosítjuk a  $Z$  változó értékeit, hogy  $Z$  változó értéke :  $z_n^2+c$ -re módosuljon. A  $Z$  változó új értékeit  $Z$  eredeti változóiba másoljuk. Az iteracio számlálót növeljük ,mivel ez egy iteráció volt. Ha az iterációk száma eléri a határt, az a következőt jelenti : az iterációnak van határértéke, vagyis a C szám eleme a Mandelbrot-halmaznak. Mivel a C szám eleme a Mandelbrot-halmaznak a hozzá tartozó képpontot színezzük.

```
kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,
 255-iteracio%256, 255-
 iteracio%256));
kep.write(argv[1]);
std::cout << argv[1] << " mentve" << std::endl;
```

A set\_pixel függvény hozzáférést ad nekünk a k,j koordinátákon található képpontokhoz , majd ezen képpontok színét az rgb\_pixel függvénnyel módosítjuk. A kep fájlba kiírjuk kirajzolt Mandelbrot-halmazunkat. A png fájl létrejöttéről a program egy üzenetet küld a felhasználónak --> "mentve".

Fordítjuk és futtatjuk a programot a következő parancsokkal:

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

Megoldás videó:

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/Mandelbrot/3.1.2.cpp](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Mandelbrot/3.1.2.cpp) <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/3.1.2.cpp>

A feladat lényegében ugyanaz, mint az előbb. Ugyanúgy egy olyan program elkészítése a cél, ami egy Mandelbrot-halmazt generál, azonban most az std::complex osztály használatával/segítségével. A megoldásnak a lényege annyiban más mint az előző feladatban, hogy nem szükséges a feladat megoldásához szükséges számok valós- és képzetes részét külön-külön változóba tárolni, ugyanis megoldható egyetlen egy darabban is.

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

Az első két könyvtárat már az előző feladatban kiveséztük. Továbbá ahogy a feladat is kérte, include-oljuk a complex függvénykönyvtárat is , ami komplex számokkal való könnyebb számolást teszi nekünk lehetővé.

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

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

A main függvényben megadjuk a készülő képünk méreteit majd megadjuk a maximálisan megengedett iterációk számát is , ez az iterációs határ. Ezek mellett is inicializálásra kerülnek a komplex számsík határértékei , magyarul , hogy az alaphalmaz mely tartományai között dolgozunk.

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

 return -1;
}
```

Kezdődik az if feltétel vizsgálat. A feltétel vizsgálatunk a következő : ha a futtatáskor megadott parancssori argumentumok száma 9, akkor a változók értékének beállítjuk a parancssori argumentumokat. Mindez az atoi és az atof függvényeknek köszönhetően jön létre. Röviden , hogy mit csinál a két függvény ? Az atoi egy olyan függvény amely stringet alakít át integerré. Az atof egy olyan függvény ami pedig stringet alakít át double -lé.

A függvény vizsgálatunk else része : Hogyan a felhasználó által megadott parancssori argumentumok száma nem 9, akkor egyszerűen kiiratjuk, mi az argumentumok megfelelő sorrendje . Majd -1-gyel tér vissza a program.

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

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

Az `image` függvényel létrehozzuk szelesség\*magasság felbontású képünket. Még ez a kép egy üres png kiterejszésű állomány. A `dx`, `dy` változókat definiáljuk, a `reC`, `imC`, `reZ`, `imZ` változókat pedig deklaráljuk. Definiálunk `double` típusú változókat és az `int` típusú `iteracio` változó értékét 0-ra állítjuk. A "re" előtag a komplex szám valós részét és az "im" előtag a komplex szám képzetét jelöli.

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

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

 for (int k = 0; k < szelesség; ++k)

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

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

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

Elindítunk két for ciklust , az egyiket magasságig, a másikat pedig szelességeig. Magyarul egy úgyn-evezett rácsot járunk be a komplex számok számsíkján.

Az `reC` és a `imC` változó értékét a képen látható módon adjuk meg. A program elején inkludált `complex` osztály segítségével létrehozzuk a `c` változót, ami megkapja a komplex szám valós részét, majd a képzetét is. Hasonló módon megkapjuk `z_n` értékét is. Ugyanúgy, a zárójelben lévő 1. szám a valós rész, míg a 2. szám a képzetés része lesz a számnak. A `c` változó lesz a vizsgált rácspont programunk során.

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

 ++iteracio;
}

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

Egy while ciklus segítségével történik meg halmaz színezése. A ciklus futási feltétele a következő : meghívjuk az `abs` függvényt `z_n`-re , ha `z_n` abszolútértéke kisebb , mint 4 és az `iteracio` számláló elérte a program elején definiált iterációhatárt, akkor van határértéke az iterációnak. A `z_n` értékét  $z_n^2 + c$ -re változtatjuk. Az `iteracio` számlálót egyel növelem, mert a ciklus akkor fejeződik be, ha el

értük az iterációs határt. Ez azt jelenti, hogy az adott pont Mandelbrot-halmaz elem, ezért azt be kell jelölünk a számsíkon. A kepre meghívjuk a set\_pixel függvényt, ami kiszínezi a k, j-edik képpontot az rgb\_pixel függvénynek átadott színnel.

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

Létrehozzuk az integer típusú szazalek változót, ami futtatás után azt jelöli majd, hány %-ban van kész a kép kirajzoltatása. Meghívjuk a flush függvényt. A flush függvény biztosítja azt, hogy a bufferben lévő összes bájt kiiratásra kerül.

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

A write függvénytel kiíratjuk az elkészült képünket az 1-es indexű parancssori argumentumkint megadott fájba. A legutolsó sorban kiiratunk egy üzenetet, hogy a képünk el lett mentve.

Fordítjuk és futtatjuk a programot:

### 5.3 Biomorfok

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

Megoldás forrása: [https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/3.1.3.cpp](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorfhttps://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/3.1.3.cpp)

Tutorált : Kincs Ákos <https://gitlab.com/kincsa>

Tanulságok, tapasztalatok, magyarázat...

A biomorfok leginkább olyan formák vagy alakzatok amik egy úgynevezett biológiai organizmusra hasonlítanak. Pickover amerikai kutató egy Julia-halmazhoz kapcsolódó programban található hiba következtében fedezte fel ezeket az alakzatokat. Aktuális feladatunk során egy ilyen biomorfot fogunk megrajzolatni felhasználva az előző, Mandelbrot-halmazos feladatot. A kirajzolt alakzat az előzőhöz hasonlóan szintén fraktál lesz. Forrás és a Biomorfokról bővebben : [https://www.emis.de/journals/TJNSA/includes/files-articles/Vol9\\_Iss5\\_2305--2315\\_Biomorphs\\_via\\_modified\\_iterations.pdf/](https://www.emis.de/journals/TJNSA/includes/files-articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf/)

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

Az inkludált könyvtárakat már kiveséztük , nincs bennük változás az előző feladatban inkludált könyvtárakhoz képest.

```
int main (int argc, char *argv[])
{
 int szelesseg = 1920;
 int magassag = 1080;
 int iteraciosHatar = 255;
```

```
double xmin = -1.9;
double xmax = 0.7;
double ymin = -1.3;
double ymax = 1.3;
double reC = .285, imC = 0;
double R = 10.0;
```

A main függvényben definiált változók nagy része kis mértékben tér el az előző feladatban definiált változóktól , így ez most nem igényel különösebb magyarázást. Definiáljuk a C vizsgált rácspontot definiáló `reC` duoble típusú változót ,ami C komplex szám valós része és az `imC` double típusú változót , ami a C komplex szám képzetes része.

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

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

Feltétel vizsgálat :ellenőrizzük a program helyesen történő futtatását. Ha nem, kiiratjuk neki a helyes módot. Ha igen, a megfelelő indexű parancssori argumentumokat az `atof` és `atoi` függvényekkel átalakítjuk megfelelő típusú értékekké , ezeknél az így kialakított értékeket hozzárendeljük a változóinkhoz.

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

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

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

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

Ebben a kód részletben hozzuk létre a képünket, melynek mérete: `szelesseg*magassag`. Definiáljuk a `cc` komplex számot is, `cc` paramétere azok a valós és képzetes részek lesznek, amit az előbbiekben parancssori argumentumként kapott.

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

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

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

int iteracio = 0;
for (int i=0; i < iteraciosHatar; ++i)
{
 z_n = std::pow(z_n, 3) + cc;
 //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
 if (std::real (z_n) > R || std::imag (z_n) > R)
 {
 iteracio = i;
 break;
 }
}
kep.set_pixel (x, y,
 png::rgb_pixel ((iteracio*20)%255, (iteracio*40)%255, (iteracio*60)%255));
}
```

Két egyszerű forciklussal végig megyünk a rácson . A complex osztály és a valós és képzetes részek felhasználásával létrehozzuk a z\_n változónkat. Egy harmadik for cikluson kerül meghívásra, ami iterációs határ eléréséig fut. A z\_n értékét a képlet szerint  $z_n = z_n^3 + c$ -re változtatjuk meg és egyszerűen az érintett koordinátákon megtalálható képpontokat színezzük.

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

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

Létrehozásra kerül a százalék nyilvántartására szolgáló változó, amit már az előző feladatban kiveséztünk.

## 5.4 A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/CUDA/mandelpngc\\_60x60\\_100](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/CUDA/mandelpngc_60x60_100)  
[https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/mandelpngc\\_60x60\\_100.cu](https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/mandelpngc_60x60_100.cu)

Tapasztalat, tanulságok, magyarázat...

Mi is az a CUDA? A CUDA az Nvidia által kifejlesztett, jellemzően C, C++ nyelvekkel remek összhangban működő technológia ami lehetővé teszi a programot író számára a videokártya erőforrásait felhasználva a párhuzamos számítást. A programot sajnos nem áll módómban futtatni, ugyanis nem áll rendelkezésemre CUDA kártya. A programkód .cu kiterjesztésű de leginkább C++ szintaktika található benne.

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

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

#define MERET 600
#define ITER_HAT 32000
```

A program legelején inkludáljuk a megfelelő könyvtárakat. A sys/times.h header fájl teszi lehetővé a processzoridővel való munkát. Ahogy azt már az előző Mandelbrotos programok során kiveséztük, most is nevesített konstansokat használunk a méret és az iterációs határ méretére vonatkozóan.

```
__device__ int mandel (int k, int j)
{
 // Végigzongorázza a CUDA a szélesség x magasság rácsot:
 // most eppen a j. sor k. oszlopban 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;
}
```

A `__device__` a függvény neve előtt azt adja meg, hogy a függvényt a kártya hajtsa végre. Létrehozásra kerülnek változók amik a számításhoz szükséges adatokat tartalmazzák , mint a szélességet, magasságot, iterációs határt, lépésközöt és iterációk számát tartalmazó változók. A függvényen belül ugyanaz történik , mint az előző feladatok esetében. Nem áll rendelkezésre az `std::complex` osztályunk, ezért a `z` és `c` komplex számok valós és képzetes részeit külön változóban kell tárolnunk. Majd végigmegyünk a szélesség `x` magasság rácson. Következik egy `while` ciklus , amiben a következő vizsgáljuk : `z_n` abszolútértéke kisebb-e mint 2 és hogy elértek-e az iterációs határt. Ha ez igaz, akkor módosítjuk a `Z` változó értékeit :  $z_n^2+c$ -re. A `Z` változó új értékeit `Z` eredeti változóiba másoljuk. Az `iteracio` számlálót növeljük. Majd a végén a függvény az iterációknak a számát adja vissza.

```
/*
__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);

}
```

A `__global__` a függvényünk előtt azt jelzi, hogy ezt a functiont a videokártya fogja végrehajtani. A CPU hívja meg a függvényt, de a GPU hajta végre. A feladatban 60x60 darab blokkal dolgozunk, blokkonként 100 darab szállal, így a `threadIdx.x` és `threadIdx.y` azt jelölik, hogy az értékek kiszámítása melyik szalon fut, ezeket az adatokat `j` és `k` változókban tároljuk. Majd ezt az értékeket újra felhasználjuk a `j` és `k` változókat számoljuk ki. A `blockIdx.x` és `blockIdx.y` változókat is amik a blokkokat azonosítják. A `j` és `k` változó értéke: `blokk*10+szál`. Az ílyen módon megkapott értékeket paraméterként adjuk át a `mandel` függvénynek.

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

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

 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);

}
```

A függvénynek átadunk egy tömböt. Létrehozunk egy int típusú pointert vagy mutatót majd a `cudaMalloc` függvény segítségével memóriát foglalunk számára. Létrehozunk két 3 dimenziós vektort amikben számértékeket tárolunk. A `cudaMemcpy` függvénytel a `device_kepadat` értékét a `kepadat`ba másoljuk. A függvény harmadik paramétere az átmásolandó bájtokat, míg az utolsó a másolás típusát adja meg ami Device to Host. Majd a legvégen `cudaFree` függvény segítségével felszabadítjuk a lefoglalt memóriát.

```
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;
```

Egy változóban tároljuk a `clock` függvény értékét, ami a program által elhasznált processzoridőt adja vissza. Akár csak az előzőekben, ha helytelenül futtatná az illető a programot, figyelmeztetjük őt. Meghívásra kerül a `cudamandel` függvény és létrejön a képünk, ami természetesen még egy üres állomány. Már ismert módon végigmegyünk a sorokon és oszlopokon és ha az adott képpontról kiderül, hogy a Mandelbrot-halmaz eleme, akkor kiszínezzük azt. A legvégén tájékoztatjuk a felhasználót a kép mentésének sikereségéről: mentve és arról, hogy mennyi időnkbe került a kép elkészülése.

## 5.5 Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás videó:

Megoldás forrása: [https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel\\_nagyito/](https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel_nagyito/)  
[https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/mandelbrot/mandel\\_nagyito](https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/mandelbrot/mandel_nagyito)

Tanulságok, tapasztalatok, magyarázat...

A feladat megoldásához a Qt-t fogjuk használni, ha a Qt eszköztár nincs letöltve, le kell töltenünk. A Qt egy olyan eszköztár, amivel grafikus felhasználói felületű programokat tudunk létrehozni.

A következő 5 fájlra van szükségünk, hogy minden a tervez szerint hiba mentesen történjen:

- main.cpp
- frakszal.cpp
- frakszal.h
- frakablak.cpp
- frakablak.h

### main.cpp

```
#include <QApplication>
#include "frakablak.h"

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

 FrakAblak w1;
 w1.show();
 return a.exec();
}
```

Inkludolásra kerülnek a Qt-hez szükséges könyvtárak, a `QApplication` illetve egy másik .h kiterjesztésű , amelyet mi hoztunk létre. A `QApplication` segítségével példányosítunk, létre hozásra kerül egy objektum , majd a konstruktur kerül meghívásra.

### frakablak.h

```
#ifndef FRAKABLAK_H
#define FRAKABLAK_H

#include < QMainWindow>
#include < QImage>
#include < QPainter>
```

```
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"

class FrakSzal;

class FrakAblak : public QMainWindow
{
 Q_OBJECT

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

protected:
 void paintEvent(QPaintEvent*) ;
 void mousePressEvent(QMouseEvent*) ;
 void mouseMoveEvent(QMouseEvent*) ;
 void mouseReleaseEvent(QMouseEvent*) ;
 void keyPressEvent(QKeyEvent*) ;

private:
 QImage* fraktal;
 FrakSzal* mandelbrot;
 bool szamitasFut;
 // A nagyítandó kijelölt területet bal felső sarka.
 int x, y;
 // A nagyítandó kijelölt terület szélessége és magassága.
 int mx, my;
};

#endif // FRAKABLAK_H
```

Ez a header fájl, amely egyaránt tartalmaz public protected és private hozzáférésű részeket. A protected functionok dolgozzák fel a billentyűzetlenyomásokat és egérmozgatásokat- és kattintásokat. Ezek a függvények itt még csak deklarálva vannak, a definiálásra később kerül majdsor.

**frakablak.cpp (részlet)**

```
void FrakAblak::paintEvent(QPaintEvent*) {
 QPainter qpainter(this);
 qpainter.drawImage(0, 0, *fraktal);
 if(!szamitasFut) {
 qpainter.setPen(QPen(Qt::white, 1));
 qpainter.drawRect(x, y, mx, my);

 }
 qpainter.end();
}

void FrakAblak::mousePressEvent(QMouseEvent* event) {

 // A nagyítandó kijelölt területet bal felső sarka:
 x = event->x();
 y = event->y();
 mx = 0;
 my = 0;

 update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event) {

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

 update();
}

void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {

 if(szamitasFut)
 return;

 szamitasFut = true;

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

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

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

```
this->d = d;

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

update();
}

void FrakAblak::keyPressEvent (QKeyEvent *event)
{

 if (szamitasFut)
 return;

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

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

}
```

Lényegében az előző header fájlban deklarált funkcionok definiálásár kerül sor ebben a fájlban.

### frakszal.h

```
#ifndef FRAKSZAL_H
#define FRAKSZAL_H

#include <QThread>
#include <math.h>
#include "frakablak.h"

class FrakAblak;

class FrakSzal : public QThread
{
 Q_OBJECT

public:
 FrakSzal(double a, double b, double c, double d,
 int szelesseg, int magassag, int iteraciosHatar, FrakAblak ←
 *frakAblak);
 ~FrakSzal();
 void run();
```

```
protected:
 // A komplex sík vizsgált tartománya [a,b]x[c,d].
 double a, b, c, d;
 // A komplex sík vizsgált tartományára feszített
 // háló szélessége és magassága.
 int szelesseg, magassag;
 // Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt ←
 //
 // (tk. most a nagyítási pontosság)
 int iteraciosHatar;
 // Kinek számolok?
 FrakAblak* frakAblak;
 // Soronként küldöm is neki vissza a kiszámoltakat.
 int* egySor;

};

#endif // FRAKSZAL_H
```

Ebben az osztályban kerül deklarálásra azok a változókat, amiket a számolás és a rajzolás során fogunk alkalmazni.

### frakszal.cpp

```
#include "frakszal.h"

FrakSzal::FrakSzal(double a, double b, double c, double d,
 int szelesseg, int magassag, int ←
 iteraciosHatar, FrakAblak *frakAblak)
{
 this->a = a;
 this->b = b;
 this->c = c;
 this->d = d;
 this->szelesseg = szelesseg;
 this->iteraciosHatar = iteraciosHatar;
 this->frakAblak = frakAblak;
 this->magassag = magassag;

 egySor = new int[szelesseg];
}

FrakSzal::~FrakSzal()
{
 delete[] egySor;
}

void FrakSzal::run()
{
 // A [a,b]x[c,d] tartományon milyen sűrű a
```

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

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

 rez = ujrez;
 imZ = ujimZ;

 ++iteracio;

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

 iteracio %= 256;
```

```
 // a színezést viszont már majd a FrakAblak osztályban ←
 // lesz
 egySor[k] = iteracio;
 }
 // Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.
 frakAblak->vissza(j, egySor, szelesség);
}
frakAblak->vissza();

}
```

A Mandelbrot halmaz kirajzolását segítő osztály, már megszokottan végigmegyünk a szélesség X magasság rácson minden pontra megnézve, eleme-e a Mandelbrot halmaznak.

Hogyan bírjuk működésre a Qt-t?

## 5.6 Mandelbrot nagyító és utazó Java nyelven

Megoldás videó:

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html> - Mandelbrot halmaz nagyító programja [https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/mandelbrot/mandel\\_nagyito.java](https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/mandelbrot/mandel_nagyito.java)

A már előzőleg C++ nyelven megírt Mandelbrot-halmaz kirajzoló programunk továbbfejlesztett változatát, a Mandelbrot nagyító és utazót fogjuk megírni Java nyelven.

```
public class MandelbrothalmazNagyító extends Mandelbrothalmaz {
 /** A nagyítandó kijelölt területet bal felső sarka. */

 private int x, y;
 /** A nagyítandó kijelölt terület szélessége és magassága. */

 private int mx, my;
```

Létrehozzuk a MandelbrothalmazNagyító osztályunkat aminek a neve után megtalálható az extends szó. Ez teszi lehetővé számunkra, hogy a program a Mandelbrothalmaz.java program változóit és függvényeit is tudja használni. Majd létre hozzuk a nagyítandó területet tároló, privát hozzáférésű változókat is.

```
public MandelbrothalmazNagyító(double a, double b, double c, double ←
 d,
 int szélesség, int iterációsHatár) {

 super(a, b, c, d, szélesség, iterációsHatár);
 setTitle("A Mandelbrot halmaz nagyításai");

 // Egér kattintó események feldolgozása:
 addMouseListener(new java.awt.event.MouseAdapter() {
 // Egér kattintással jelöljük ki a nagyítandó területet
```

```
// bal felső sarkát:
public void mousePressed(java.awt.event.MouseEvent m) {
 // A nagyítandó kijelölt területet bal felső sarka:
 x = m.getX();
 y = m.getY();
 mx = 0;
 my = 0;
 repaint();
}

public void mouseReleased(java.awt.event.MouseEvent m) {
 double dx = (MandelbrotHalmazNagyító.this.b
 - MandelbrotHalmazNagyító.this.a)
 /MandelbrotHalmazNagyító.this.szélesség;
 double dy = (MandelbrotHalmazNagyító.this.d
 - MandelbrotHalmazNagyító.this.c)
 /MandelbrotHalmazNagyító.this.magasság;

 // Az új Mandelbrot nagyító objektum elkészítése:
 new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+←
 x*dx,
 MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
 MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
 MandelbrotHalmazNagyító.this.d-y*dy,
 600,
 MandelbrotHalmazNagyító.this.iterációsHatár);
}

// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
 // Vonszolással jelöljük ki a négyzetet:
 public void mouseDragged(java.awt.event.MouseEvent m) {
 // A nagyítandó kijelölt terület szélessége és magassága:
 mx = m.getX() - x;
 my = m.getY() - y;
 repaint();
 }
});
```

Ebben a függvényben hívjuk meg az eredeti osztályban található konstruktort. A különböző egérműveleteket a megfelelő functionök kezelik, ilyen például a kattintás és az egérgomb felengedése. Az egérgomb felengedése esetén egy új MandelbrotHalmazNagyító objektum is létrehozásra kerül ugyanis ilyenkor lehet megtudni, makkora területet akarunk nagyítani. Ezeken kívül nyomon követjük az egér mozgását. Az itt található repaint teszi lehetővé ,hogy a halmazunk újra kirajzoltatható legyen a nagyítások elvégzése után után.

```
public void pillanatfelvétel() {
```

```
// Az elmentendő kép elkészítése:

java.awt.image.BufferedImage mentKép =
 new java.awt.image.BufferedImage(szélesség, magasság,
 java.awt.image.BufferedImage.TYPE_INT_RGB);
java.awt.Graphics g = mentKép.getGraphics();
g.drawImage(kép, 0, 0, this);
g.setColor(java.awt.Color.BLUE);
g.drawString("a=" + a, 10, 15);
g.drawString("b=" + b, 10, 30);
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iterációsHatár, 10, 75);
if(számításFut) {
 g.setColor(java.awt.Color.RED);
 g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
// A pillanatfelvétel képfájl nevének képzése:
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyitas_");
sb.append(++pillanatfelvételszámLálo);
sb.append("_");
// A fájl nevébe belevesszük, hogy melyik tartományban
// találtuk a halmazt:

sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");
// png formátumú képet mentünk

try {
 javax.imageio.ImageIO.write(mentKép, "png",
 new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
 e.printStackTrace();
}
}
```

Itt jön létre az üres képunk és a megfelelő képpontok színezése.

```
public void paint(java.awt.Graphics g) {
 // A Mandelbrot halmaz kirajzolása
```

```
g.drawImage(kép, 0, 0, this);
// Ha éppen fut a számítás, akkor egy vörös
// vonallal jelöljük, hogy melyik sorban tart:
if(számításFut) {
 g.setColor(java.awt.Color.RED);
 g.drawLine(0, sor, getWidth(), sor);
}
// A jelző négyzet kirajzolása:
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
}
```

Megtörténik a keret kirajzoltatása amit a nagyításkor használunk.

```
public static void main(String[] args) {
 // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
 // tartományában keressük egy 600x600-as hálóval és az
 // aktuális nagyítási pontossággal:
 new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
```

A main-ben sor kerül példányosításra.

Fordítjuk és futtatjuk:

# Chapter 6

## 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:<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html/>

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

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

using namespace std;

class Random
{

public:
 Random();
 ~Random() {}

 double get();
private:
 bool exist;
 double value;
};

}
```

Deklaráljuk a Random osztályt, a publikus részben az osztály konstruktora és destruktora és az a függvény ami random számokat fog lekérni kap helyet, míg a privát részben azt fogjuk tárolni, hogy létezik-e korábban kiszámolt másik random és ennek a kiszámolt randomnak az értéke.

```
Random::Random()
{
 exist = false;
 srand (time(NULL));
}
```

A konstruktorban inicializáljuk a randomszám generáló srand függvényt.

```
double Random::get()
{
 if (!exist)
 {
 double u1, u2, v1, v2, w;

 do
 {
 u1 = rand () / (RAND_MAX + 1.0);
 u2 = rand () / (RAND_MAX + 1.0);
 v1 = 2 * u1 - 1;
 v2 = 2 * u2 - 1;
 w = v1 * v1 + v2 * v2;
 }
 while (w > 1);

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

 value = r * v2;
 exist = !exist;

 return r * v1;
 }

 else
 {
 exist = !exist;
 return value;
 }
}
```

Lekérő függvény. Abban az esetben , ha nincs eltárolva random számunk, akkor létrehozunk két számot. Az elsőt ki fogjuk íratni, a másodikat eltároljuk. Ellenkező esetben az eltárolt számot kiíratjuk.

```
int main()
{
 Random rnd;
```

```
for (int i = 0; i < 2; ++i) cout << rnd.get() << endl;
}
```

A main függvényben meghívjuk a random osztály lehívó függvényét.

Polártranszformációs algoritmus JAVA:

```
public class PolárTranszF {

 boolean létezik_tárolt = false;
 double tárolt;

 public PolárTranszF() {

 létezik_tárolt = false;

 }
}
```

A publikus PolárTranszF osztályunkban deklaráljuk a logikai tagot : van-e korábban eltárolt random, és ha van mi annak az értéke.

```
public double matek_rész() {

 if(!létezik_tárolt) {

 double u1, u2, v1, v2, w;
 do {
 u1 = Math.random();
 u2 = Math.random();

 v1 = 2*u1 - 1;
 v2 = 2*u2 - 1;

 w = v1*v1 + v2*v2;

 } while(w > 1);

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

 tárolt = r*v2;
 létezik_tárolt = !létezik_tárolt;

 return r*v1;

 } else {
 létezik_tárolt = !létezik_tárolt;
 return tárolt;
 }
}
```

Ha nincs korábban eltárolt random értékünk, akkor meghívjuk a misztikus random-generáló matematikai eljárást. Ellenkező esetben az értékét adjuk vissza és a logikai tagot átállítjuk.

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

 PolárTranszF g = new PolárTranszF();

 for(int i=0; i<2; ++i)
 System.out.println(g.matek_rész());

}
}
```

Két hívás következik : Ki iratjuk a kiszámolt számok egyikét, második hívásnál pedig a kiszámolt, majd elmentett értéket adjuk vissza.

## 6.2 LZW

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

Tutoriált : Kincs Ákos <https://gitlab.com/kincsakos>

Megoldás forrása : [https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa\\_z.c](https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa_z.c)

Az Lempel–Ziv–Welch (LZW) algoritmust az 1980-as évek közepén, Abraham Lempel és Jacob Ziv már létező algoritmusát továbbfejlesztve, Terry Welch publikálta. Az algoritmus a UNIX alapú rendszerek fájltömörítő segédprogramja által terjed el leginkább, továbbá GIF kiterjesztésű képek és PDF fájlok veszteségmentes tömörítéséhez is használják. Az USA-ban 2003-ban, a világ többi részén 2004-ben az algoritmus szabadalma lejárt, ezért azóta alkalmazása a háttérbe szorult. Forrás : [Wikipédia](#)

A bináris fát úgy építjük fel, hogy az input nullákat és egyeseket beolvassuk. Ha olyat olvasunk be, amivel már találkoztunk , akkor olvassuk tovább. Az így kapott új egységet fogjuk a fa gyökerétől kezdve kiiratni. Ha az adott egység ábrázolva van, visszaugrunk a gyökérbe és olvassuk tovább az inputot.

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

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

Definiáljuk a binfa struktúrát.

```
BINFA_PTR
uj_elem ()
```

```
{
 BINFA_PTR p;

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

A új elem létrehozásakor memóriát szabadítunk majd fel. Ha hiba lép fel akkor kiléünk a programból.

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

Deklaráljuk, de még nem definiáljuk a fa kiírásához és a lefoglalt memória felszabadításához használt függvényeket. (Egyelőre nem foglalunk le nekik helyet a memóriában.)

```
int
main (int argc, char **argv)
{
 char b;

 BINFA_PTR gyoker = uj_elem ();
 gyoker->ertek = '/';
 BINFA_PTR fa = gyoker;

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

```
 fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
 fa = gyoker;
}
else
{
 fa = fa->jobb_egy;
}
}
}

printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d\n", max_melyseg);
szabadit (gyoker);
}
```

Először létrehozzuk a gyökeret és ráállítjuk a fa pointert. Olvassuk az inputon érkező 0-kat és 1-eseket.

- Ha 0-t olvasunk és az aktuális nodenak nincs nullás gyermekje, akkor az uj\_elem függvényel létrehozunk neki egyet, majd megkapja értékül a 0-t. Ezután ennek az újonnan létrehozott gyermeknek beállítjuk a bal és jobb gyermekjeit NULL pointerekre és visszaugrunk a pointerrel a gyökérre. Ha a beolvasáskor már volt nullás gyermekje az aktuális nodenak, akkor a mutatót ráállítjuk.
- Ha 1-t olvasunk és az aktuális nodenak nincs egyes gyermekje, akkor az uj\_elem függvényel létrehozunk neki egyet, majd megkapja értékül az 1-t. Ezután ennek az újonnan létrehozott gyermeknek beállítjuk a bal és jobb gyermekjeit NULL pointerekre és visszaugrunk a pointerrel a gyökérre. Ha a beolvasáskor már volt egyes gyermekje az aktuális nodenak, akkor a mutatót ráállítjuk.

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

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

A `kiir` függvényel jelenítjük meg magát a fát a parancssorban. Mivel 90 fokkal el van forgatva az eredmény balra és fentről lefelé írjuk ki az ágakat ezért inorder bejárás esetén először a jobb oldali ágat, majd a gyökeret, majd végül a bal oldali ágat rajzoltatjuk ki.

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

Rekurzívan hívjuk a függvényt. Ezzel a lépéssel felsazabadítjuk azokat az elemeket amiket korábban lefoglaltunk.

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ gcc binfa_z.c -o binfa -lm
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$./binfa <bemenet.txt> out.txt
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat out.txt

-----1(2)
----1(1)
-----1(4)
----1(3)
-----0(2)
-----0(3)
---/(0)
----1(2)
----0(1)
----0(2)
melyseg=4
altag=2.600000
szoras=0.894427
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$
```

## 6.3 Fabejárás

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

Megoldás forrása:

- [https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa\\_z\\_pre.c](https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa_z_pre.c)
- [https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa\\_z\\_post.c](https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa_z_post.c)

Példa Inorder bejárásra:

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

void
kiir (BINFA_PTR elem)
```

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

A preorder bejárás : a gyökeret dolgozzuk fel, majd a bal oldali részfát és végül a jobb oldali részfát .

Példa Preorder bejárásra:

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

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

 --melyseg;
 }
}
```

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ gcc binfa_z_pre.c -o pre -lm
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$./pre <bemenet.txt> preki.txt
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat preki.txt

---/(1)
----0(2)
-----0(3)
-----1(3)
----1(2)
-----0(3)
-----0(4)
-----1(4)
-----1(5)
----1(3)
melyseg=4
altag=2.600000
szoras=0.894427
```

A postorder bejárás : bal oldali részfát majd a jobb oldali részfát majd a gyökeret dolgozzuk fel.

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

 for (int i = 0; i < melyseg; ++i)
 printf ("---");
 printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↔
 , melyseg);
 kiir (elem->jobb_egy);
 kiir (elem->bal nulla);
 --melyseg;
 }
}
```

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ gcc binfa_z_post.c -o post -lm
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$./post <bemenet.txt> postki.txt
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat postki.txt

-----0(3)
-----1(3)
-----0(2)
-----0(4)
-----1(5)
-----1(4)
-----0(3)
-----1(3)
-----1(2)
---/(1)
melyseg=4
altag=2.600000
szoras=0.894427
```

## 6.4 Tag a gyökér

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

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/z3a7.cpp>

```
.

#include <iostream>

class LZWTREE
{
public:
 LZWTREE () : fa(&gyoker) {}

 ~LZWTREE ()
 {
 szabadit (gyoker.egyesGyermek ());
 szabadit (gyoker.nullasGyermek ());
 }

 void operator<<(char b)
 {
 if (b == '0')
 {

 if (!fa->nullasGyermek ())
 {
 Node *uj = new Node ('0');
 fa->ujNullasGyermek (uj);
 fa = &gyoker;
 }
 else
 {
 fa = fa->>nullasGyermek ();
 }
 }
 else
 {
 if (!fa->egyesGyermek ())
 {
 Node *uj = new Node ('1');
 fa->ujEgyesGyermek (uj);
 fa = &gyoker;
 }
 else
 {
 }
 }
 }
};
```

```
 fa = fa->egyesGyermek ();
 }
}
void kiir (void)
{
 melyseg = 0;
 kiir (&gyoker);
}
void szabadit (void)
{
 szabadit (gyoker.jobbEgy);
 szabadit (gyoker.balNulla);
}
```

A fa konstruktorá és destruktora után az LZWTree osztályon belül definiáljuk a balra eltoló bitshift operátort . Ez fogja az inputról érkező karaktereket eltolni egészen a LZWTree objektumba. Így fog felépülni a fa.

**private:**

```
class Node
{
public:
 Node (char b = '/') :betu (b), balNulla (0), jobbEgy (0) {};
 ~Node () {};
 Node *nullasGyermek ()
 {
 return balNulla;
 }
 Node *egyesGyermek ()
 {
 return jobbEgy;
 }
 void ujNullasGyermek (Node * gy)
 {
 balNulla = gy;
 }
 void ujEgyesGyermek (Node * gy)
 {
 jobbEgy = gy;
 }

private:
 friend class LZWTree;
 char betu;
 Node *balNulla;
 Node *jobbEgy;
 Node (const Node &);
 Node & operator=(const Node &);
};
```

Ha a Node konstruktor paraméter nélküli, akkor az alapértelmezett '/'-jellel fogja azt létrehozni. Egyébként a meghívó karakter kerül a betű helyére. A bal és jobb gyermekre mutató mutatókat nulára állítjuk majd az aktuális Node minden meg tudja mondani, hogy mi a bal illetve jobb gyermeke. Deklaráljuk az LZWTree osztályt a csomópontok kezelésére.

```
Node gyoker;
Node *fa;
int melyseg;

LZWTree (const LZWTree &);
LZWTree & operator=(const LZWTree &);

void kiir (Node* elem)
{
 if (elem != NULL)
 {
 ++melyseg;
 kiir (elem->jobbEgy);

 for (int i = 0; i < melyseg; ++i)
 std::cout << "---";
 std::cout << elem->betu << "(" << melyseg - 1 << ")" << std::endl;
 kiir (elem->balNulla);
 --melyseg;
 }
}
void szabadit (Node * elem)
{
 if (elem != NULL)
 {
 szabadit (elem->jobbEgy);
 szabadit (elem->balNulla);
 delete elem;
 }
}

};
```

Mindig az aktuális csomópontra mutatunk. A íír függvényteljesen jelenítjük meg magát a fát.

```
int
main ()
{
 char b;
 LZWTree binFa;

 while (std::cin >> b)
 {
 binFa << b;
 }
```

```
binFa.kiir ();
binFa.szabadit ();

 return 0;
}
```

Bitenként olvassuk a bemenetet.

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7.cpp -o z3a7
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7.cpp -o fa
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$./fa bemenet.txt -o faki.txt
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat faki.txt
-----1(2)
-----0(3)
-----0(4)
-----1(1)
-----0(2)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
--/(0)
-----1(3)
-----0(4)
-----0(5)
-----1(2)
-----0(3)
-----1(6)
-----1(5)
-----0(4)
-----0(1)
-----1(4)
-----0(5)
-----0(6)
-----0(7)
-----1(3)
-----0(4)
-----1(6)
-----0(5)
-----0(2)
-----1(5)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
-----1(7)
-----0(6)
depth = 7
mean = 5.36364
var = 1.02691
```

## 6.5 Mutató a gyökér

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

Megoldás forrása: [https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/z3a7\\_new.cpp](https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/z3a7_new.cpp)

Eddig a LZWTree-ben a fa gyökere egy objektum volt. Mostantól a gyökér is és a fa is egy-egy mutató lesz. Tehát a fa mutatóinknak a gyökér mutatót adjuk.

```
class LZWTree
{
public:
 LZWTree ()
 {
 gyoker = new Node();
 fa = gyoker;
 }

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

A csomópontra mutató mutatóként létrehozzuk a gyökeret, a destrukturban fel szabadítjuk . Az eddig a gyökér előtt álló referenciajeleket mindenhol kitöröljük.

```
Node *gyoker;
Node *fa;
int melyseg;
```

Ahol eddig objektumként szerepelt ott átirjuk pointerré.

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7_new.cpp -o fanew
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$./fanew bemenet.txt -o fanewki.txt
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat fanewki.txt
-----1(2)
-----0(3)
-----0(4)
--1(1)
--0(2)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
-/0)
-----1(3)
-----0(4)
-----0(5)
-1(2)
--0(3)
-----1(6)
-----1(5)
-----0(4)
0(1)
-----1(4)
-----0(5)
-----0(6)
-----0(7)
-1(3)
-----0(4)
-----1(6)
-----0(5)
-0(2)
-----1(5)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
-----1(7)
-----0(6)
depth = 7
mean = 5.36364
var = 1.02691
```

## 6.6 Mozgató szemantika

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

Megoldás forrása: [https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/z3a7\\_mozgato.cpp](https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/z3a7_mozgato.cpp)

```
LZWTree& operator= (LZWTree& copy)
{
 szabadit(gyoker->egyesGyermek ());
 szabadit(gyoker->>nullasGyermek ());

 bejar(gyoker,copy.gyoker);

 fa = gyoker;
 copy.fa = copy.gyoker;

}
```

Törlöm a régi értékeket majd bejárom a fát és és átmásolom az értékeket , ezek után minden két fában visszaugrok a gyökérhez.

```
void bejar (Node * masolat, Node * eredeti)
{
 if (eredeti != NULL)
 {
 if (!eredeti->nullasGyermek())
 {
 masolat->ujNullasGyermek(NULL);
 }
 else
 {

 Node* uj = new Node ('0');
 masolat->ujNullasGyermek (uj);
 bejar(masolat->nullasGyermek(), eredeti->nullasGyermek());
 }

 if (!eredeti->egyesGyermek())
 {
 masolat->ujEgyesGyermek (NULL);
 }
 else
 {

 Node *uj = new Node ('1');
 masolat->ujEgyesGyermek (uj);
 bejar(masolat->egyesGyermek(), eredeti->egyesGyermek());
 }
 }
 else
 {
 masolat = NULL;
 }
}
```

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7_mozgato.cpp -o famozg
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$./famozg bemenet.txt -o famozgki.txt
LZWBInFa mozgato ctor
LZWBInFa mozgato ctor
LZWBInFa mozgato ctor
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat famozgki.txt
-----1(2)
-----0(3)
-----0(4)
----1(1)
----0(2)
----1(4)
----0(3)
----1(5)
----0(4)
----0(5)
--/(0)
----1(3)
----0(4)
----0(5)
---1(2)
---0(3)
----1(6)
----1(5)
----0(4)
---0(1)
----1(4)
----0(5)
----0(6)
----0(7)
---1(3)
---0(4)
----1(6)
----0(5)
---0(2)
----1(5)
----1(4)
---0(3)
----1(5)
----0(4)
----0(5)
----1(7)
----0(6)
depth = 7
mean = 5.36364
var = 1.02691
```

Utóirat : a std::move semmit nem mozgat.

# Chapter 7

## 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://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/myrmecologist>

Kezdjük elsősorban azzal h miért is érdekesek a hangyák? A hangyák feromon nevezetű kémiai anyagot bocsátanak ki. A feromon hatására a kiszemelt hangya az anyagot kibocsátó ellenétes nemű hangya iránt vonzalmat kezd érezni. A feromon leginkább a párválasztásban játszik fontos szerepet. Merül fel bennünk a kérdés tehát, hogy a hangyák feromon kibocsátása hogyan befolyásolja a többi hangya mozgását, illetve a többi hangya elhelyezkedését és viselkedését. Logikusan belegondolva, a hangyák azokhoz a szomszédjaikhoz fognak közel kerülni akiknek a feromonszintje a legmagasabb. Nos a mi programunk is ezt hivatott bemutatni.

#### A main.cpp

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

#include "antwin.h"

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

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

```
{

 QApplication a (argc, argv);

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

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

 parser.process (a);

 QString szeles = parser.value (szeles_opt);
 QString magas = parser.value (magas_opt);
 QString n = parser.value (hangyaszam_opt);
 QString t = parser.value (sebesseg_opt);
 QString parolgas = parser.value (parolgas_opt);
 QString feromon = parser.value (feromon_opt);
```

```
QString szomszed = parser.value (szomszed_opt);
QString alapertek = parser.value (alapertek_opt);
QString maxcella = parser.value (maxcella_opt);
QString mincella = parser.value (mincella_opt);
QString cellameret = parser.value (cellamerete_opt);

qsrand (QDateTime::currentMSecsSinceEpoch());

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

w.show();

return a.exec();
}
```

Szokásos módon includoljuk a szükséges könyvtárakat , jelenleg ez a Qt-s és az antwin.h könyvtárakat. Majd a mainben példányosítjuk a QApplication objektumot , illetve megadjuk és feldolgozzuk a futtatáskor szükséges kapcsolókat.

### A ant.h

```
#ifndef ANT_H
#define ANT_H

class Ant
{

public:
 int x;
 int y;
 int dir;

 Ant (int x, int y) : x(x), y(y) {

 dir = qrand() % 8;

 }

};

typedef std::vector<Ant> Ants;

#endif
```

Először is vegyük a grand függvényt , ennek a függvénynek a segítségével állítjuk be a hangya irányát , ami véletlenszerű. A hangya mozgásának leírására hivatott változok minden publikusak. Abból a célból , hogy

megkönnenítsük a magunk dolgát `TypeDef`-et használunk , ami annyit tesz , hogy azokra a vektorokra amik `Ant`-okat tartalmaznak a későbbiekben `Ants` néven fogunk hivatkozni.

### Az `antthread.h`

```
#ifndef ANTTHREAD_H
#define ANTTHREAD_H

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

class AntThread : public QThread
{
 Q_OBJECT

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

 ~AntThread();

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

 void pause()
 {
 paused = !paused;
 }

 bool isRunnung()
 {
 return running;
 }

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

```
int height;
int gridIdx;
int delay;

void timeDevel();

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

signals:
void step (const int &);

};

#endif
```

Deklaráljuk azokat a függvényeket amiknek feladata az ablak képét megállítani , illetve a szimuláció befejezése. Ebben a kódrészletben deklaráljuk a konstruktort és a destruktur-t is. Továbbá létrehozzuk azokat a privát változókat amik az úgynevezett "rács" vagy "háló"-ra vonatkoznak. Ezek például a : min, max , width , height , running , pause.

### Az antthread.h

```
#include "antthread.h"
#include <QDebug>
#include <cmath>
#include <QDateTime>

AntThread::AntThread (Ants* ants, int*** grids,
 int width, int height,
 int delay, int numAnts,
 int pheromone, int nbrPheromone,
 int evaporation,
 int min, int max, int cellAntMax)
{
 this->ants = ants;
 this->grids = grids;
 this->width = width;
 this->height = height;
 this->delay = delay;
 this->pheromone = pheromone;
 this->evaporation = evaporation;
 this->min = min;
 this->max = max;
 this->cellAntMax = cellAntMax;
 this->nbrPheromone = nbrPheromone;
```

```
numAntsinCells = new int*[height];
for (int i=0; i<height; ++i) {
 numAntsinCells[i] = new int [width];
}

for (int i=0; i<height; ++i)
 for (int j=0; j<width; ++j) {
 numAntsinCells[i][j] = 0;
 }

qsrand (QDateTime::currentMSecsSinceEpoch());

Ant h {0, 0};
for (int i {0}; i<numAnts; ++i) {

 h.y = height/2 + qrand() % 40-20;
 h.x = width/2 + qrand() % 40-20;

 ++numAntsinCells[h.y][h.x];

 ants->push_back (h);
}

gridIdx = 0;
}

double AntThread::sumNbhs (int **grid, int row, int col, int ←
 dir)
{
 double sum = 0.0;

 int ifrom, ito;
 int jfrom, jto;

 detDirs (dir, ifrom, ito, jfrom, jto);

 for (int i=ifrom; i<ito; ++i)
 for (int j=jfrom; j<jto; ++j)

 if (! ((i==0) && (j==0))) {
 int o = col + j;
 if (o < 0) {
 o = width-1;
 } else if (o >= width) {
 o = 0;
 }

 int s = row + i;
```

```
 if (s < 0) {
 s = height-1;
 } else if (s >= height) {
 s = 0;
 }

 sum += (grid[s][o]+1)*(grid[s][o]+1)*(grid[s][o ↔
]+1);

 }

 return sum;
}

int AntThread::newDir (int sor, int oszlop, int vsor, int ←
 voszlop)
{
 if (vsor == 0 && sor == height -1) {
 if (voszlop < oszlop) {
 return 5;
 } else if (voszlop > oszlop) {
 return 3;
 } else {
 return 4;
 }
 } else if (vsor == height - 1 && sor == 0) {
 if (voszlop < oszlop) {
 return 7;
 } else if (voszlop > oszlop) {
 return 1;
 } else {
 return 0;
 }
 } else if (voszlop == 0 && oszlop == width - 1) {
 if (vsor < sor) {
 return 1;
 } else if (vsor > sor) {
 return 3;
 } else {
 return 2;
 }
 } else if (voszlop == width && oszlop == 0) {
 if (vsor < sor) {
 return 7;
 } else if (vsor > sor) {
 return 5;
 } else {
 return 6;
 }
 }
}
```

```
 } else if (vsor < sor && voszlop < oszlop) {
 return 7;
 } else if (vsor < sor && voszlop == oszlop) {
 return 0;
 } else if (vsor < sor && voszlop > oszlop) {
 return 1;
 }

 else if (vsor > sor && voszlop < oszlop) {
 return 5;
 } else if (vsor > sor && voszlop == oszlop) {
 return 4;
 } else if (vsor > sor && voszlop > oszlop) {
 return 3;
 }

 else if (vsor == sor && voszlop < oszlop) {
 return 6;
 } else if (vsor == sor && voszlop > oszlop) {
 return 2;
 }

 else { // (vsor == sor && voszlop == oszlop)
 qDebug() << "ZAVAR AZ EROBEN az iranynal";
 return -1;
 }
 }

 void AntThread::detDirs (int dir, int& ifrom, int& ito, int& ←
 jfrom, int& jto)
 {

 switch (dir) {
 case 0:
 ifrom = -1;
 ito = 0;
 jfrom = -1;
 jto = 2;
 break;
 case 1:
 ifrom = -1;
 ito = 1;
 jfrom = 0;
 jto = 2;
 break;
 case 2:
 ifrom = -1;
 ito = 2;
```

```
jfrom = 1;
jto = 2;
break;
case 3:
ifrom =
 0;
ito = 2;
jfrom = 0;
jto = 2;
break;
case 4:
ifrom = 1;
ito = 2;
jfrom = -1;
jto = 2;
break;
case 5:
ifrom = 0;
ito = 2;
jfrom = -1;
jto = 1;
break;
case 6:
ifrom = -1;
ito = 2;
jfrom = -1;
jto = 0;
break;
case 7:
ifrom = -1;
ito = 1;
jfrom = -1;
jto = 1;
break;

}

}

int AntThread::moveAnts (int **racs,
 int sor, int oszlop,
 int& vsor, int& voszlop, int dir)
{
 int y = sor;
 int x = oszlop;

 int ifrom, ito;
 int jfrom, jto;
```

```
detDirs (dir, ifrom, ito, jfrom, jto);

 double osszes = sumNbhs (racs, sor, oszlop, dir);
 double random = (double) (qrand() %1000000) / (double ↫
) 1000000.0;
 double gvalseg = 0.0;

 for (int i=ifrom; i<ito; ++i)
 for (int j=jfrom; j<jto; ++j)
 if (! ((i==0) && (j==0)))
 {
 int o = oszlop + j;
 if (o < 0) {
 o = width-1;
 } else if (o >= width) {
 o = 0;
 }

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

 //double kedvezo = std::sqrt((double)(racs[s][o ↫
]+2));//(racs[s][o]+2)*(racs[s][o]+2);
 //double kedvezo = (racs[s][o]+b)*(racs[s][o]+b ↫
);
 //double kedvezo = (racs[s][o]+1);
 double kedvezo = (racs[s][o]+1)*(racs[s][o]+1) ↫
 *(racs[s][o]+1);

 double valseg = kedvezo/osszes;
 gvalseg += valseg;

 if (gvalseg >= random) {

 vsor = s;
 voszlop = o;

 return newDir (sor, oszlop, vsor, voszlop ↫
);

 }
 }

 qDebug() << "ZAVAR AZ EROBEN a lepesnel";
```

```
 vsor = y;
 voszlop = x;

 return dir;
}

void AntThread::timeDevel()
{
 int **racsElotte = grids[gridIdx];
 int **racsUtana = grids[(gridIdx+1) %2];

 for (int i=0; i<height; ++i)
 for (int j=0; j<width; ++j)
 {
 racsUtana[i][j] = racsElotte[i][j];

 if (racsUtana[i][j] - evaporation >= 0) {
 racsUtana[i][j] -= evaporation;
 } else {
 racsUtana[i][j] = 0;
 }
 }

 for (Ant &h: *ants)
 {

 int sor {-1}, oszlop {-1};
 int ujirany = moveAnts(racsElotte, h.y, h.x, sor, ←
 oszlop, h.dir);

 setPheromone (racsUtana, h.y, h.x);

 if (numAntsinCells[sor][oszlop] <cellAntMax) {

 --numAntsinCells[h.y][h.x];
 ++numAntsinCells[sor][oszlop];

 h.x = oszlop;
 h.y = sor;
 h.dir = ujirany;

 }
 }

 gridIdx = (gridIdx+1) %2;
}
```

```
void AntThread::setPheromone (int **racs,
 int sor, int oszlop)
{

 for (int i=-1; i<2; ++i)
 for (int j=-1; j<2; ++j)
 if (! ((i==0) && (j==0)))
 {
 int o = oszlop + j;
 {
 if (o < 0) {
 o = width-1;
 } else if (o >= width) {
 o = 0;
 }
 }
 int s = sor + i;
 {
 if (s < 0) {
 s = height-1;
 } else if (s >= height) {
 s = 0;
 }
 }

 if (racs[s][o] + nbrPheromone <= max) {
 racs[s][o] += nbrPheromone;
 } else {
 racs[s][o] = max;
 }
 }
 }

 if (racs[sor][oszlop] + pheromone <= max) {
 racs[sor][oszlop] += pheromone;
 } else {
 racs[sor][oszlop] = max;
 }
}

void AntThread::run()
{
 running = true;
 while (running) {

 QThread::msleep (delay);
 }
}
```

```
 if (!paused) {
 timeDevel();
 }

 emit step (gridIdx);

 }

AntThread::~AntThread()
{
 for (int i=0; i<height; ++i) {
 delete [] numAntsinCells[i];
 }

 delete [] numAntsinCells;
}
```

Ebben a kódrészletben röviden annyi történik , hogy definiáljuk azokat a függvényeket amelyeket már az előző header fájlban deklaráltunk.

### Az **antwin.h**

```
##ifndef ANTWIN_H
#define ANTWIN_H

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

class AntWin : public QMainWindow
{
 Q_OBJECT

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

 AntThread* antThread;

 void closeEvent (QCloseEvent *event) {
```

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

 void keyPressEvent (QKeyEvent *event)
 {

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

 }

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

private:

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

public slots :
 void step (const int &);

};

#endif
```

Deklaráljuk a keyPressEvent függvényt ami a billentyűzetről érkező includokat kezeli , emellett deklaráljuk az ablak bezárását kezelő closeEvent függvényt

### Az **antwin.cpp**

```
#include "antwin.h"
#include <QDebug>

AntWin::AntWin (int width, int height, int delay, int numAnts,
 int pheromone, int nbhPheromon, int evaporation, ←
```

```
 int cellDef,
 int min, int max, int cellAntMax, QWidget *parent ←
) : QMainWindow (parent)
{
 setWindowTitle ("Ant Simulation");

 this->width = width;
 this->height = height;
 this->max = max;
 this->min = min;

 cellWidth = 6;
 cellHeight = 6;

 setFixedSize (QSize (width*cellWidth, height*cellHeight));

 grids = new int**[2];
 grids[0] = new int*[height];
 for (int i=0; i<height; ++i) {
 grids[0][i] = new int [width];
 }
 grids[1] = new int*[height];
 for (int i=0; i<height; ++i) {
 grids[1][i] = new int [width];
 }

 gridIdx = 0;
 grid = grids[gridIdx];

 for (int i=0; i<height; ++i)
 for (int j=0; j<width; ++j) {
 grid[i][j] = cellDef;
 }

 ants = new Ants();

 antThread = new AntThread (ants, grids, width, height, delay, ←
 numAnts, pheromone,
 nbhPheromon, evaporation, min, max, ←
 cellAntMax);

 connect (antThread, SIGNAL (step (int)),
 this, SLOT (step (int)));

 antThread->start();
}

void AntWin::paintEvent (QPaintEvent*)
{
```

```
QPainter qpainter (this) ;

grid = grids[gridIdx] ;

for (int i=0; i<height; ++i) {
 for (int j=0; j<width; ++j) {

 double rel = 255.0/max;

 qpainter.fillRect (j*cellWidth, i*cellHeight,
 cellWidth, cellHeight,
 QColor (255 - grid[i][j]*rel,
 255,
 255 - grid[i][j]*rel));

 if (grid[i][j] != min)
 {
 qpainter.setPen (
 QPen (
 QColor (255 - grid[i][j]*rel,
 255 - grid[i][j]*rel, 255),
 1)
);

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

qpainter.setPen (
 QPen (
 QColor (0,0,0),
 1)
);

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

}

}

for (auto h: *ants) {
 qpainter.setPen (QPen (Qt::black, 1));

 qpainter.drawRect (h.x*cellWidth+1, h.y*cellHeight+1,
 cellWidth-2, cellHeight-2);
}
```

```
 qpainter.end();
}

AntWin::~AntWin()
{
 delete antThread;

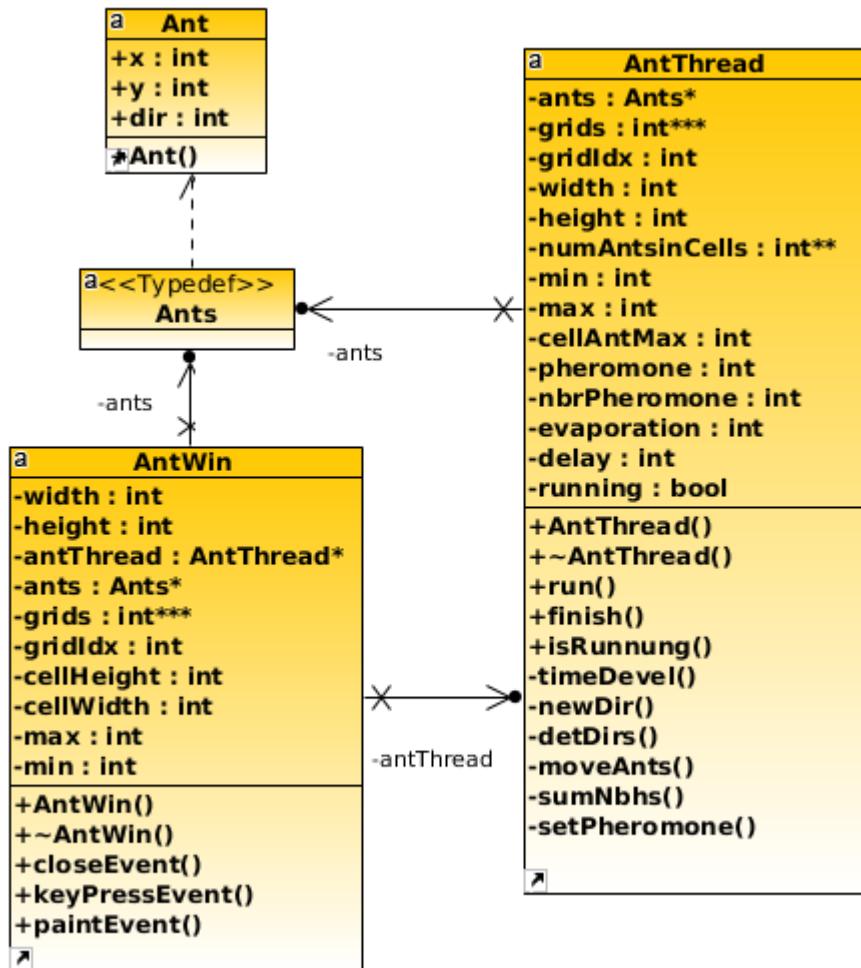
 for (int i=0; i<height; ++i) {
 delete[] grids[0][i];
 delete[] grids[1][i];
 }

 delete[] grids[0];
 delete[] grids[1];
 delete[] grids;

 delete ants;
}

void AntWin::step (const int &gridIdx)
{
 this->gridIdx = gridIdx;
 update();
}
```

Definiáljuk a headerben includolt könyvtárakat. Definiáljuk a kontstruktort és a destruktort. Értéket adunk a "cella" vagy "rács" magasságának és szélességének, majd a főprogramban megtörténik a hangyák kirajzoltatása.



Az UML Unified Modeling Language , magyarul Egységes Modellező Nyelvnek nevezzük. A programozásban a legelterjedt módja a programunk osztályainak és az osztályok közötti kapcsolatainak ábrázolására.

Mit kell tudni az UML osztálydiagramról? Az osztályok reprezentálása történik benne. A fenti képen minden sárga cella egy-egy osztály. minden osztálynak vannak tulajdonságai és emellett viselkedése. A tulajdonságokat a változóknak és a viselkedéseket a függvényeknek nevezzük. A kettő közötti elválasztó vonal is be van jelölve a diagramunkon minden egyes osztály esetében, a vonal felett találhatók meg a változók még alatta a függvények helyezkednek el.

Szembetűnő dolog még a + vagy - jel a változók vagy függvények előtt : A + jel azt jelenti , hogy az adott változó vagy függvény `public` , tehát más osztályokból is hozzá tudunk férn. A - jel `private` változókat vagy függvényeket jelöl , amikra csak az osztályon belül hivatkozhatunk.

## 7.2 Java életjáték

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

Megoldás videó:

Megoldás forrása: [https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/sejtauto\\_java](https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/sejtauto_java)

Tanulságok, tapasztalatok, magyarázat...

Az életjáték John Horton Conway, angol egyetemi matematikus nevével vált eggyé. Mi is az életjáték? Egy olyan játék ahol a játékos feladata, hogy megad egy kiiunduló alakzatot majd várja az eredményt. A játék alapját egy négyzetrácsos ablak adja meg, mezőket celláknak, az itt megtalálható korongokat pedig sejteknek nevezzük. A sejt környezetének a hozzá legközelebb eső 8 mezőt tekintjük. A sejt szomszédai a környezetében található sejtek.

A játék elején a játékosnak lehetősége van cellákba sejteket helyeznie.

A játék körökre van osztva. Egy sejttel a különbező körökben különböző dolgok történhetnek, melyek a következők:

A sejt túlél a kört akkor, ha pontosan 2 vagy pontosan 3 élő szomszédja van.

A sejt elpusztul akkor, ha 2-nél kevesebb vagy 3-nál több élő szomszédja van.

Új sejt is születhet akkor, ha az adott cella környezetében pontosan 3 sejt található.

Több, egymás mellett található sejt alakzatot alkot. Az egyik ilyen alakzat a sikló. A sikló átlósan mozog minden egy-egy kockányit miközben változtatja formáját, mozgása végén visszatér a kiindulási formájába.

```
public class Sejtautomata extends java.awt.Frame implements Runnable {
 /** Egy sejt lehet élő */
 public static final boolean ÉLŐ = true;
 /** vagy halott */
 public static final boolean HALOTT = false;
 /** Két rácst használunk majd, az egyik a sejttér állapotát
 * a t_n, a másik a t_n+1 időpillanatban jellemzi. */
 protected boolean[][][] rácok = new boolean[2][][];
 /** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen
 * a [2][][]-ból az első dimenziót használni, mert vagy az
 * egyikre
 * állítjuk, vagy a másikra. */
 protected boolean[][] rács;
 /** Megmutatja melyik rács az aktuális: [rácsIndex][][] */
 protected int rácsIndex = 0;
 /** Pixelben egy cella adatai. */
 protected int cellaSzélesség = 20;
 protected int cellaMagasság = 20;
 /** A sejttér nagysága, azaz hányszor hány cella van? */
 protected int szélesség = 20;
 protected int magasság = 10;
 /** A sejttér két egymást követő t_n és t_n+1 diszkrét id
 * öppillanata
 * közötti valós idő. */
 protected int várakozás = 1000;
 // Pillanatfelvétel készítéséhez
 private java.awt.Robot robot;
 /** Készítsünk pillanatfelvételt? */
 private boolean pillanatfelvétel = false;
 /** A pillanatfelvételek számozásához. */
```

```
private static int pillanatfelvételSzámláló = 0;
```

A Sejtautomata osztályban az osztály a `java.awt.Frame` osztályt használja a keret létrehozásához vagyis a grafikus megjelenítéshez. Létrehozásra kerülnek a sejtek körönkénti állapotát jelző változók, a cella adatokat tároló változók, egy logikai változó ami azt az információt tárolja, hogy készítettünk-e már pil- lanatképet vagy sem. Majd létrehozásra kerül a rács.

```
public Sejtautomata(int szélesség, int magasság) {
 this.szélesség = szélesség;
 this.magasság = magasság;
 // A két rács elkészítése
 rácsok[0] = new boolean[magasság][szélesség];
 rácsok[1] = new boolean[magasság][szélesség];
 rácsIndex = 0;
 rács = rácsok[rácsIndex];
 // A kiinduló rács minden cellája HALOTT
 for(int i=0; i<rács.length; ++i)
 for(int j=0; j<rács[0].length; ++j)
 rács[i][j] = HALOTT;
 // A kiinduló rácsra "élőlényeket" helyezünk
 //sikló(rács, 2, 2);
 siklóKilövő(rács, 5, 60);
 // Az ablak bezárásakor kilépünk a programból.
 addWindowListener(new java.awt.event.WindowAdapter() {
 public void windowClosing(java.awt.event.WindowEvent e) {
 setVisible(false);
 System.exit(0);
 }
 });
}
```

Létrehozunk egy Sejtautomataobjektumot. Esetünkben ez a Sejtautomataobjektum a konstruktur. A cellák állapotát beállítjuk, a játék kezdetén minden cella halott. Egy for ciklus segítségével végig megyünk soron és oszlopokon. Létre hozunk egy functiont, ami azt hivatott figyeli, hogy mikor zárjuk be az ablakot. Ha ez megtörténik, kilép a programból.

```
addKeyListener(new java.awt.event.KeyAdapter() {
 // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük
 public void keyPressed(java.awt.event.KeyEvent e) {
 if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
 // Felezük a cella méreteit:
 cellaSzélesség /= 2;
 cellaMagasság /= 2;
 setSize(Sejtautomata.this.szélesség*cellaSzélesség,
 Sejtautomata.this.magasság*cellaMagasság);
 validate();
 } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
 // Duplázzuk a cella méreteit:
 cellaSzélesség *= 2;
 cellaMagasság *= 2;
 setSize(Sejtautomata.this.szélesség*cellaSzélesség,
 Sejtautomata.this.magasság*cellaMagasság);
 }
 }
});
```

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

// Egér kattintó események feldolgozása:

```
addMouseListener(new java.awt.event.MouseAdapter() {
 // Egér kattintással jelöljük ki a nagyítandó területet
 // bal felső sarkát vagy ugyancsak egér kattintással
 // vizsgáljuk egy adott pont iterációit:
 public void mousePressed(java.awt.event.MouseEvent m) {
 // Az egérmutató pozíciója
 int x = m.getX()/cellaSzélesség;
 int y = m.getY()/cellaMagasság;
 rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
 repaint();
 }
});
```

// Egér mozgás események feldolgozása:

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

// Cellaméretek kezdetben

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

```
// A program ablakának adatai:
setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
 magasság*cellaMagasság);
setVisible(true);
// A sejttér életre keltése:
new Thread(this).start();
}
/** A sejttér kirajzolása. */
public void paint(java.awt.Graphics g) {
 // Az aktuális
 boolean [][] rács = rácsok[rácsIndex];
 // rácsot rajzoljuk ki:
 for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon
 for(int j=0; j<rács[0].length; ++j) { // s az oszlopok
 // Sejt cella kirajzolása
 if(rács[i][j] == ÉLŐ)
 g.setColor(java.awt.Color.BLACK);
 else
 g.setColor(java.awt.Color.WHITE);
 g.fillRect(j*cellaSzélesség, i*cellaMagasság,
 cellaSzélesség, cellaMagasság);
 // Rács kirajzolása
 g.setColor(java.awt.Color.LIGHT_GRAY);
 g.drawRect(j*cellaSzélesség, i*cellaMagasság,
 cellaSzélesség, cellaMagasság);
 }
 }

 // Készítünk pillanatfelvételt?
 if(pillanatfelvétel) {
 // a biztonság kedvéért egy kép készítése után
 // kikapcsoljuk a pillanatfelvételt, hogy a
 // programmal ismerkedő Olvasó ne írja tele a
 // fájlrendszerét a pillanatfelvétellekkel
 pillanatfelvétel = false;
 pillanatfelvétel(robot.createScreenCapture
 (new java.awt.Rectangle
 (getLocation().x, getLocation().y,
 szélesség*cellaSzélesség,
 magasság*cellaMagasság)));
 }
}
```

Nyomon követjük az egérről és a billentyűzetről érkező inputokat. Majd Beállításra kerülnek a cellaméretek és a programablakok . Majd a new szóval létrehozunk egy új Thread-et, amit a későbbiekben a start function-nel el is indítunk. A paint függvénytel a "sejttér" kerül megrajzolásra.A Java awt.Graphics G osztálya teszi lehetővé a sejttér megrajzolását. A rajzoláshoz a fekete , fehér és a szürke színeket

használjuk. A megrajzolás után megvizsgáljuk, hogy a rács jelenlegi állapotáról készült -e pillanatfelvétel.

```
public int szomszédokSzáma(boolean [][] rács,
 int sor, int oszlop, boolean állapot) {
 int állapotúSzomszéd = 0;
 // A nyolcszomszédok végigzongorázása:
 for(int i=-1; i<2; ++i)
 for(int j=-1; j<2; ++j)
 // A vizsgált sejtet magát kihagyva:
 if(!((i==0) && (j==0))) {
 // A sejttérből szélénk szomszédai
 // a szembe oldalakon ("periódikus határfeltétel")
 int o = oszlop + j;
 if(o < 0)
 o = szélesség-1;
 else if(o >= szélesség)
 o = 0;

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

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

Megvizsgáljuk az adott sejt 8 szomszédjának az állapotát , mert ez befolyásolja , hogy a sejtünk életben marad, meghal , megszületik-e.Legvégül a függvény visszaadja a szomszédok számát

```
public void időFejlődés() {

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

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

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

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

```
 }
 } else {
 /* Halott halott marad, ha három élő
 * szomszedja van, különben élő lesz. */
 if(élők==3)
 rácsUtána[i][j] = ÉLŐ;
 else
 rácsUtána[i][j] = HALOTT;
 }
}
rácsIndex = (rácsIndex+1)%2;
}
/** A sejttér időbeli fejlődése. */
public void run() {

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

 időFejlődés();
 repaint();
 }
}
```

Megvizsgáljuk, hogy élő vagy halott az adott sejt, esetleg éppen megszületik-e. A `run` függvényben egy végtelen ciklus foglal helyet aminek eredménye képpen a programunk addig fut amíg mi kívülről meg nem szakítjuk.

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

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

}

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

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

 rács[y+ 3][x+ 13] = ÉLŐ;
 rács[y+ 4][x+ 12] = ÉLŐ;
```

```
rács[y+ 4][x+ 14] = ÉLŐ;
rács[y+ 5][x+ 11] = ÉLŐ;
rács[y+ 5][x+ 15] = ÉLŐ;
rács[y+ 5][x+ 16] = ÉLŐ;
rács[y+ 5][x+ 25] = ÉLŐ;

rács[y+ 6][x+ 11] = ÉLŐ;
rács[y+ 6][x+ 15] = ÉLŐ;
rács[y+ 6][x+ 16] = ÉLŐ;
rács[y+ 6][x+ 22] = ÉLŐ;
rács[y+ 6][x+ 23] = ÉLŐ;
rács[y+ 6][x+ 24] = ÉLŐ;
rács[y+ 6][x+ 25] = ÉLŐ;

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

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

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

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

rács[y+ 11][x+ 25] = ÉLŐ;

}

/** Pillanatfelvételek készítése. */
public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
 // A pillanatfelvétel kép fájlneve
```

```
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("sejtautomata");
sb.append(++pillanatfelvételszámláló);
sb.append(".png");
// png formátumú képet mentünk
try {
 javax.imageio.ImageIO.write(felvetel, "png",
 new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
 e.printStackTrace();
}
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
 paint(g);
}
/**
 * Példányosít egy Conway-féle életjáték szabályos
 * sejttér objektumot.
 */
public static void main(String[] args) {
 // 100 oszlop, 75 sor mérettel:
 new Sejtautomata(100, 75);
}
```

A pillanatfelvétel függvény az S billentyű lenyomására hívódik meg. Aktiválásakor egy png kiterjesztésű képállományt hoz létre, ezen elkészült képállományok számát is eltároljuk egy változóban.

A main függvényben történik a példányosítás, létrehozunk egy objektumot 100 és 75 paraméterekkel ezek jelölik az ablak szélességet és magasságát.

Fordítás és futtatás után:

### 7.3 Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

[https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/sejtauto-qt\\_ej](https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/sejtauto-qt_ej)

Tanulságok, tapasztalatok, magyarázat...

**A main.cpp-ben:**

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

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

 return a.exec();
}
```

Inkludáljuk az osztályokat amik a Qt-vel való munkához szükségünk lesz. A `QApplication` segítségével fogunk példányosítani. Létrehozunk egy objektumot. Majd meghívásra kerül a kontsruktor is itt adjuk meg az ablak méreteit is.

#### A `sejtablak.h`-ban:

```
#ifndef SEJTABLEAK_H
#define SEJTABLEAK_H

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

class SejtSzal;

class SejtAblak : public QMainWindow
{
 Q_OBJECT

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

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

protected:
 // Két rácsot használunk majd, az egyik a sejttér állapotát
 // a t_n, a másik a t_n+1 időpillanatban jellemzi.
 bool ***racsok;
 // Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
 // [2][][]-ból az első dimenziót használni, mert vagy az egyikre
 // állítjuk, vagy a másikra.
 bool **racs;
 // Megmutatja melyik rács az aktuális: [räcsIndex] []
 int racsIndex;
 // Pixelben egy cella adatai.
```

```
int cellaSzelesseg;
int cellaMagassag;
// A sejttér nagysága, azaz hányszor hány cella van?
int szelesseg;
int magassag;
void paintEvent (QPaintEvent*);
void siklo(bool **racs, int x, int y);
void sikloKilovo(bool **racs, int x, int y);

private:
SejtSzal* eletjatek;

};

#endif // SEJTABLAK_H
```

Létrehozzuk SejtAblak konstruktorát és destruktörét. Deklarálunk számos functiont és a cella magassági és szélességi adatait tároló int típusú változókat.

#### A **sejtablak.cpp**-ben:

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

SejtAblak::~SejtAblak()
{
```

```
delete eletjatek;

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

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

...
...

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

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

}

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

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

Itt hívunk meg sok-sok funkciót illetve itt definiáljuk a destruktort.

A **sejtszal.h**-ban:

```
#ifndef SEJTSZAL_H
#define SEJTSZAL_H

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

class SejtAblak;

class SejtSzal : public QThread
{
 Q_OBJECT

public:
 SejtSzal(bool ***racsok, int szelessseg, int magassag,
```

```
 int varakozas, SejtAblak *sejtAblak);
~SejtSzal();
void run();

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

};

#endif // SEJTSZAL_H
```

Egy header fájl kerül deklarálásra , amiben inkludáljuk a másik, header fájlunkat a fejezet elején említett céllal. Ebben a fájlból kapnak helyet public és protected változók, function-ök, a konstruktor és a destruktur deklarálása is.

#### A **sejtszal.cpp**-ben:

```
#include "sejtszal.h"

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

 racsIndex = 0;
}

int SejtSzal::szomszedokSzama(bool **racs,
 int sor, int oszlop, bool allapot) {
 int allapotuSzomszed = 0;
 // A nyolcszomszédök végigzongorázása:
 for(int i=-1; i<2; ++i)
 for(int j=-1; j<2; ++j)
 // A vizsgált sejtet magát kihagyva:
 if(!((i==0) && (j==0))) {
```

```
// A sejttérből szélénék szomszédai
// a szembe oldalakon ("periódikus határfeltétel")
int o = oszlop + j;
if(o < 0)
 o = szelesseg-1;
else if(o >= szelesseg)
 o = 0;

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

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

return allapotuSzomszed;
}

void SejtSzal::idoFejlodes() {

 bool **racsElotte = racsok[racsIndex];
 bool **racsUtana = racsok[(racsIndex+1)%2];

 for(int i=0; i<magassag; ++i) { // sorok
 for(int j=0; j<szelesseg; ++j) { // oszlopok

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

 if(racsElotte[i][j] == SejtAblak::ELO) {
 /* Élő élő marad, ha kettő vagy három élő
 szomszeda van, különben halott lesz. */
 if(elok==2 || elok==3)
 racsUtana[i][j] = SejtAblak::ELO;
 else
 racsUtana[i][j] = SejtAblak::HALOTT;
 } else {
 /* Halott halott marad, ha három élő
 szomszeda van, különben élő lesz. */
 if(elok==3)
 racsUtana[i][j] = SejtAblak::ELO;
 else
 racsUtana[i][j] = SejtAblak::HALOTT;
 }
 }
 }
 racsIndex = (racsIndex+1)%2;
}
```

```
}

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

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

Definiáljuk az `idoFejlodes` functiont amivel megvizsgáljuk , hogy az adott sejt , élő-e vagy halott , vagy éppen megszületik-e. Létrehozzuk a A `SejtSzal` konstruktort ami a sejt szomszédjainak számát határozza meg.

Ahogy az előző feladatban itt is megismerkedünk a A `run` function-nel , benne egy végtelen while ciklussal , ami lehetővé teszi a program megszakításig történő futását.

## 7.4 BrainB Benchmark

Megoldás videó:

Megoldás forrása : <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/brainb>

Tanulságok, tapasztalatok, magyarázat...

A BrainB projekt célja a jövő esportolónak felkutatása. A program az agy úgynevezett kognitív képességét méri.

A játékos feladata a Samu Entropy nevű karakteren tartani az egérmutatót ami az idő teltével természetesen egyre nehezebb hiszen újabb és újabb Entropy karakterek jelennek meg a képernyőn. A program érzékeli, ha Samu Entropy elvesztettük, ekkor csökkenti a többi Entropy számát.

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

int main (int argc, char **argv)
{
 QApplication app (argc, argv);
 QTextStream qout (stdout);
```

```
qout.setCodec ("UTF-8");

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

qout << "This program is free software: you can redistribute it and ↵
/or modify it under" << endl;
.....
.....
.....
QRect rect = QApplication::desktop()->availableGeometry();
BrainBWin brainBWin (rect.width(), rect.height());
brainBWin.setWindowState (brainBWin.windowState() ^ Qt:: ↵
 WindowFullScreen);
brainBWin.show();
return app.exec();
}
```

Includoljuk a `BrainBWin` osztályt , a többi inkludolásra kerülő osztályt már az előző programokban megfigyelhetünk , ugyanis ez a program is Qt grafikus felületet használ. Deklarálunk egy `app` `QApplication` típusú objektumot. Itt használjuk a `qout` functiont amivel a standard outputra lehet írni. Deklaráljuk az entropykat. Majd létrehozzuk a `BrainBWin` objektumot. Az osztályban inkludolásra került függvények és változók itt is .h kiterjesztésű header fájlokban vannak.

### **BrainBWin.h**

Deklaráljuk azt a függvényt ami az egér és a billentyűzetről történő inputok figyeléséért felelős. Deklaráljuk továbbá a konstruktort és a destruktort.

### **BrainBWin.cpp**

Itt kerül definiálásra az előző header fájlban deklaráltak.

### **BrainBThread.h**

A Hero-kat tartalmazó vectorra typedefet használunk, mostantól `Hero`ként hivatkozunk rá.

### **BrainBThread.cpp**

Létrehozzuk a `Hero`-kat. Definiáljuk a destruktort és deklaráljuk a `run`, `pause` és `set_paused` függvényeket

# Chapter 8

## Helló, Schwarzenegger!

### 8.1 Szoftmax Py MNIST

Python

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

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa...  
[https://progpater.blog.hu/2016/11/13/hello\\_samu\\_a\\_tensorflow-bol](https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol)

Mi is az a TensorFlow? Egy olyan könyvtár amit gépi tanulásra használnak. Ez az algoritmus hmindössze néhány éve jelent meg és néhány hónapja elérhető belőle a működő kiadás. Bővebben a Tensorflowról :<https://en.wikipedia.org/wiki/TensorFlow>

Mi az az Modified National Institute of Standards and Technology azaz röviden MNIST adatbázis? Egy olyan óriási elemszámú adatbázis, mely kézzel írott számjegyeket tartalmaz, amit fel fogunk használni a feladatmegoldásunk során. Bővebben a MNIST adatbázisról : [https://en.wikipedia.org/wiki/MNIST\\_database](https://en.wikipedia.org/wiki/MNIST_database)

Ahhoz, hogy a feladatot megoldjuk, telepíteni szükséges a Python-t és a TensorFlow-t is.

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

import argparse

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

import tensorflow as tf

import matplotlib.pyplot
```

Importáljuk a szükséges könyvtárakat.

```
def readimg():
 file = tf.read_file("sajat8a.png")
```

```
img = tf.image.decode_png(file)
return img

def main(_):
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

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

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

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

Két függvényt definiálunk. Az első azért felel, hogy a képünk beolvasásra kerüljön. Msdik a main függvényünk, itt kap helyet az az algoritmus, ami ténylegesen a számokat felismeri. Ugyanitt kap helyet a hálózat tanítása is ami mintákat kap és ezekből a mintákból próbál tanulni.

```
print("-----")

print("-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a ←
 tovabblepeshet csukd be az ablakat")

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

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

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

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

print("-- A sajat kezi 8-asom felismerese, mutatom a szamot, a ←
 tovabblepeshet csukd be az ablakat")

img = readimg()
image = img.eval()
image = image.reshape(28*28)

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib. ←
 pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

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

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

if __name__ == '__main__':
 parser = argparse.ArgumentParser()
 parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ ←
 mnist/input_data',
 help='Directory for storing input data')
 FLAGS = parser.parse_args()
 tf.app.run()
```

A hálózat tesztelése és megtudjuk miként is ismeri fel a hálózatunk az egyes számjegyeket.

## 8.2 Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Mély MNIST feladat passzolva.

### 8.3 Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Minecraft-MALMÖ feladat passzolva.

# Chapter 9

## Helló, Chaitin!

### 9.1 Iteratív és rekurzív faktoriális Lisp-ben

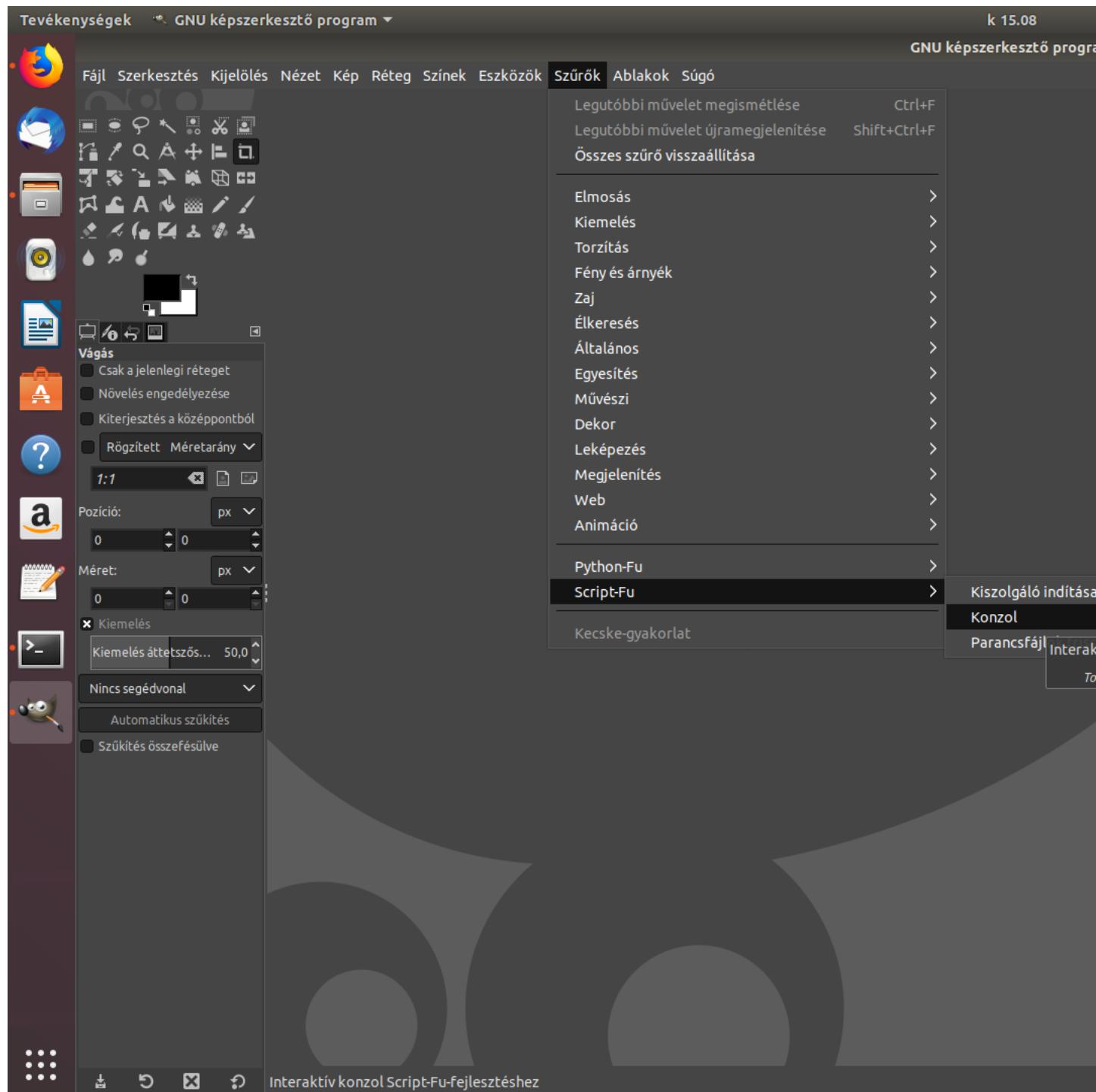
Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Most hogy már van egy elképzelésünk a nyelvről, nézzük is meg, hogyan sikerült megvalósítani a feladatot!

Mi is az a Lisp ? A Lisp nem egy újonnan létrejött nyelvcsalád. Az első Lips nyelven írt program az 1958-as évek környékén jelent meg és hamar felkapottá vált a mesterséges intelligencia területén. Nevét az általa használt adatszerkezetből , a láncolt listáról kapta. Forrás és Lispről bővebben : [https://hu.wikipedia.org/wiki/Lisp\\_\(programoz%C3%A1si\\_nyelv\)](https://hu.wikipedia.org/wiki/Lisp_(programoz%C3%A1si_nyelv)) A következő Lisp kódot a GIMP (gimpről bővebb információk : (<https://hu.wikipedia.org/wiki/GIMP>) nevű képszerkesztő program , scriptek írására alkalmas , programon belüli konzolban vizsgáljuk meg :



Következzen a kód :

The screenshot shows a terminal window titled "Script-Fu-konzol". The title bar includes the text "Üdvözli a TinyScheme", "Copyright (c) Dimitrios Souflis", and "Script-Fu-konzol - Interaktív Scheme-fejlesztés". The main area of the window displays a Scheme interpreter session:

```
> (define (fakt n) (if(< n 1) 1 (* n (fakt (- n 1)))))
fakt
> (fakt 3)
6
> (fakt 9)
362880
> (fakt 5)
120
> (fakt 9)
362880
> (fakt 11)
39916800
> (fakt 2)
2
> (fakt 7)
5040
```

At the bottom of the window are several buttons: "Súgó" (Help), "Mentés" (Save), "Törölés" (Delete), and "Bezárás" (Close). There is also a "Tallázás..." (Search...) button.

Ami először feltűnik , hogy maga a program csupán egyetlen sorból áll. Elemezzük balról jobbra haladva. A `define` szóval hívjuk meg a függvényünket `fakt` néven `n` paraméterrel , `n` a paraméter aminek a faktoriálisára kiváncsiak leszünk.Következik egy `if` függvény vizsgálat. A feltételvizsgálatban találkozhatunk azzal a furcsasággal , hogy a műveleti jel nem a két paraméter között áll , hanem előttük. Ha a feltételvizsgálat igaz , akkor aa zárójel utáni első érték kerül visszaadásra. Ha a feltételvizsgálatunk hamis akkor a viszakapott értékünk : `n` paraméter szorzata a `fakt` függvényvel, ami jelen esetben `n-1`-re kerül meghívásra. A függvényünk rekurzív , mert a függvény meghívja önmagát és mivel a végrehajtás ismétlődő ezért Iteratív is. Elmondhatjuk , hogy a kódunk rekurzív és iteratív egyszerre.

## 9.2 Gimp Scheme Script-fu: króm effekt

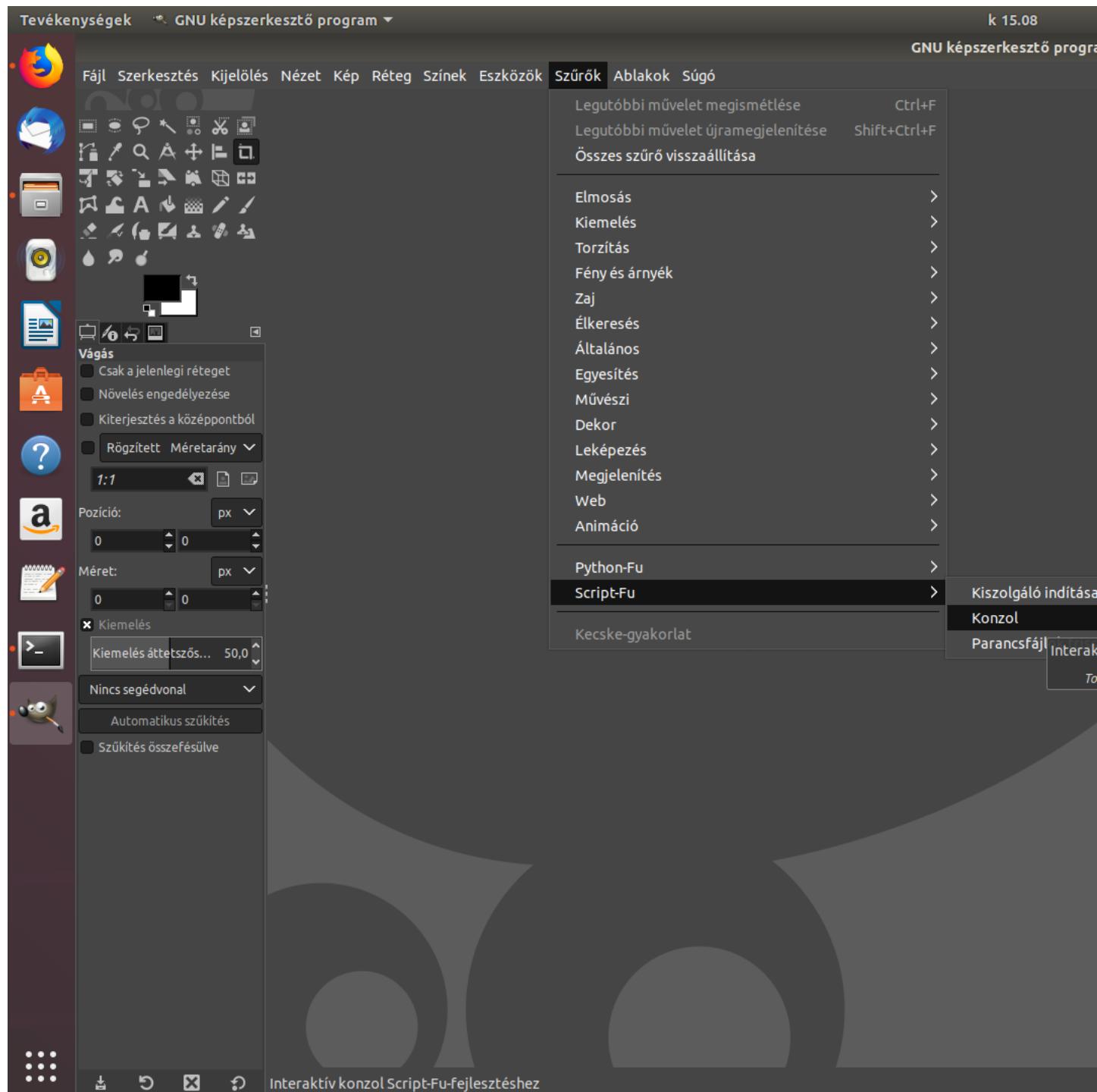
Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

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) <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/chaitin>

Tanulságok, tapasztalatok, magyarázat...

Az előző feladatban megismerkedtünk a Lisp nyelvvel , most a feladat során a Lisp nyelvcsalád egyik képviselőjével fogunk megismerkedni. A Sheme programnyelv az 1970-es évek közepén jelent meg és mai napig találkozhatunk Scheme kódokkal. Forrás és Scheme nyelvről bővebben : <https://hu.wikipedia.org/wiki/Scheme> A fájlunk .scm kiterjesztésű lesz , és ha berakjuk a GIMP erre hivatott mappájába, megtalálhatjuk mint használható szkriptet. A feladat során a Script-fu fogjuk használni . Az előző feladat során megismert GIMP képszerkesztő programhoz egy olyan szkriptet fogunk írni . ami a bemerként megadott szöveg króm effektezését teszi lehetővé. A megíráshoz használhatjuk a Script-Fu-konzolt is :



```
(aset tomb 5 20)
(aset tomb 6 200)
(aset tomb 7 190)
tomb)
)
```

Megadjuk a függvényeket , olyan módon amit már az előző feladatban láthattunk , ezért erre most nem térnék ki külön. Létrehozunk egy 8 elemű tömböt a color-curve segítségével. Majd megadjuk a megfelelő értékeket.

Megadunk egy olyan függvényt is, ami egy lista x-edik elemét adja vissza, a car(lista első eleme) és cdr(a lsita első elemén kívüli összes elem) használatával.

```
(define (elem x lista)

 (if (= x 1) (car lista) (elem (- x 1) (cdr lista)))

)

(define (text-wh text font fontsize)
(let*
 (
 (text-width 1)
 (text-height 1)
)

 (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
 PIXELS font)))
 (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
 fontsize PIXELS font)))

 (list text-width text-height)
)
)
```

Meghívunk egy függvényt. A függvényünk a listánk x-edik elemét adja vissza a car és a cdr használatával. car : a lista első eleme. cdr : a lista elsőn kívüli összes eleme. A soron következő függvény segítségével tudjuk kiszámolni a szöveg magasságát.

```
(define (script-fu-bhax-chrome text font fontsize width height color ←
 gradient)
(let*
 (
 (image (car (gimp-image-new width height 0)))
 (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
 LAYER-MODE-NORMAL-LEGACY)))
 (textfs)
 (text-width (car (text-wh text font fontsize)))
 (text-height (elem 2 (text-wh text font fontsize)))
 (layer2)
)
)
```

A `script-fu-bhax-chrome` metódus lesz felelős a króm effektért. A `let*` segítségével létrehozzuk az objektumunkat a már megadott paraméterekből. Az `image`-be létrehozzuk a képünket a `gimp-image-new` használatával. Ezután létrehozunk egy új réteget a `layer`-ben. Létrehozzuk a `textfs`-t. A `text-wh` segítségével megadjuk a szöveg szélességét és magasságát. A legvégén pedig létrehozzuk `layer2`-t.

9 lépésben megadjuk hogy krómosítson a szkriptünk.:

Első lépés:

```
;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND)
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
 height 2) (/ text-height 2)))

(set! layer (car(gimp-image-merge-down image textfs ←
 CLIP-TO-BOTTOM-LAYER)))
```

A `gimp-image-insert-layer` segítségével hozzáadjuk a képünkhez a rétegünket. A `gimp-context-set-foreground` segítségével beállítjuk az elsődleges színt feketére majd a beállított színnel kiszinezzük a rétegünket. Végük a színt fehérre állítjuk. A `textfs`-be létrehozunk egy új szövegréteget `gimp-text-layer-new` segítségével. Majd beállítjuk `gimp-layer-set-offsets` segítségével a réteg offsetjét. Összeolvasszuk a létrehozott szöveget és a hátteret. Ezzel el is értünk az első lépés végére.

Második lépés:

```
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével elmosódást állítunk be a `layer`-en.

Harmadik lépés:

```
;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
```

A `gimp-drawable-levels` segítségével beállítjuk a rétegünk szintezését.

Negyedik lépés:

```
;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével még egy elmosódást teszünk a `layer`-re.

Ötödik lépés:

```
;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)
```

A gimp-image-select-color segítségével kijelöljük a layer rétegen a fehér pixeleket. Ezt a ki-jelölést megfordítjuk gimp-selection-invert segítségével.

Hatodik lépés:

```
;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
 LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)
```

Létrehozunk egy új réteget és ezt hozzáadjuk a képünkhez.

Hetedik lépés:

```
;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
 GRADIENT-LINEAR 100 0 REPEAT-NONE
 FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
 3)))
```

Átállítjuk aktívra a gradient-t, majd a gimp-edit-blend segítségével összemossuk a layer2-t a háttérrel

Nyolcadik lépés:

```
;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
 0 TRUE FALSE 2)
```

A plug-in-bump-map segítségével domborítás effektet adunk a képhez.

Kilencedik lépés:

```
;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)
```

Beállítjuk a color-curve-t, majd megjelenítjük egy új ablakban.

```
(script-fu-register "script-fu-bhax-chrome"
 "Chrome3"
 "Creates a chrome effect on a given text."
 "Norbert Bátfai"
 "Copyright 2019, Norbert Bátfai"
```

```
"January 19, 2019"
"
SF-STRING "Text" "Bátf41 Haxor"
SF-FONT "Font" "Sans"
SF-ADJUSTMENT "Font size" '(100 1 1000 1 10 0 1)
SF-VALUE "Width" "1000"
SF-VALUE "Height" "1000"
SF-COLOR "Color" '(255 0 0)
SF-GRADIENT "Gradient" "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
 "<Image>/File/Create/BHAX"
)
```

Menű beállítása

### 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) Megoldás forrása: <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/chaitin>

Tanulságok, tapasztalatok, magyarázat...

A feladatunk már ismerős az előző feladatból. Egy hasonló szkriptet írni , ami a megadott szövgől mandalát készít. A Szkriptünk eleje és vége majdnem ugyanaz mint az előző feladatban , ezért azt nem fogjuk most külön kitárgyalni.

A szkriptünk eleje :

```
(define (elem x lista)

 (if (= x 1) (car lista) (elem (- x 1) (cdr lista)))

)

(define (text-width text font fontsize)
(let*
(
 (text-width 1)
)
(set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
PIXELS font)))

text-width
)
)
```

```
(define (text-wh text font fontsize)
(let*
(
 (text-width 1)
 (text-height 1)
)
;;
(set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
 PIXELS font)))
;;; ved ki a lista 2. elemét
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
 fontsize PIXELS font)))
;;
(list text-width text-height)
)
)
```

A szkriptünk vége :(a menü felépítése egy kicsit más , mint az előző feladatban.)

```
(script-fu-register "script-fu-bhax-mandala"
 "Mandala9"
 "Creates a mandala from a text box."
 "Norbert Bátfai"
 "Copyright 2019, Norbert Bátfai"
 "January 9, 2019"
 ""
 SF-STRING "Text" "Bátf41 Haxor"
 SF-STRING "Text2" "BHAX"
 SF-FONT "Font" "Sans"
 SF-ADJUSTMENT "Font size" '(100 1 1000 1 10 0 1)
 SF-VALUE "Width" "1000"
 SF-VALUE "Height" "1000"
 SF-COLOR "Color" '(255 0 0)
 SF-GRADIENT "Gradient" "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
 "<Image>/File/Create/BHAX"
)
```

```
(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
 gradient)
(let*
(
 (image (car (gimp-image-new width height 0)))
 (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
 LAYER-MODE-NORMAL-LEGACY)))
 (textfs)
 (text-layer)
 (text-width (text-width text font fontsize))
```

```
;;;
(text2-width (car (text-wh text2 font fontsize)))
(text2-height (elem 2 (text-wh text2 font fontsize)))
;;
(textfs-width)
(textfs-height)
(gradient-layer)
)
```

Létrehozzuk a script-fu-bhax-mandala függvényt. Paraméter listája majdnem megegyezik az előző feladatban már kitárgyalt paraméterlistával. A let\* segítségével megadjuk azokat a dolgokat amikre a továbbiakban szükségünk lesz.

```
(gimp-image-insert-layer image layer 0 0)
```

Hozzáadjuk a layer-t a képünkhez.

```
(gimp-context-set-foreground '(0 255 0))
(gimp-drawable-fill layer FILL-FOREGROUND)
(gimp-image-undo-disable image)
```

Átszínezzük a háttérét , majd beállítjuk h ne lehessen visszavonást alkalmazni.

```
(gimp-context-set-foreground color)
```

Átállítjuk a foreground color-t az előzőekben paraméterként megadott színre.

```
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) -->
))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ -->
 height 2))
(gimp-layer-resize-to-image-size textfs)
```

Egy új rétegként hozzáadjuk a képünkhez a megadott szöveget, beállítjuk az offsetet , és a rétegneket a méreteit hozzá igazítjuk a kép méreteihez.

```
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer -->
 CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pix* 2) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer -->
 CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pix* 4) TRUE 0 0)
```

```
(set! textfs (car(gimp-image-merge-down image text-layer ←
CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer ←
CLIP-TO-BOTTOM-LAYER)))
```

Lemásoljuk a szöveget, elforgatjuk és az eredetivel egybeolvasszuk. Aztán ezt újra és újra.

```
(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car(gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car(gimp-drawable-height textfs)) 100)))
```

A túllógó részeket levágjuk. Majd a textfs-width és textfs-height méreteit beállítjuk 100-al nagyobbra a textfs méreteinél.

```
(gimp-layer-resize-to-image-size textfs)
```

A képhez méretezzük a textfs rétegünket.

```
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)
```

Készítünk egy ellipszis kijelölést a szövegünk köré és kifestjük a határát.

```
(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)
```

A textfs-width és textfs-height méretét csökkentjük 70nel . Aztán elvégezzük a korábbi "feszést" mégegyszer.

```
(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ←
 "gradient" 100 LAYER-MODE-NORMAL-LEGACY)))
```

A gradient-layer-t állítjuk be.

```
(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
 GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ ←
 (/ width 2) (/ textfs-width 2)) 8) (/ height 2))
```

Hozzáadjuk a legujabb rétegünket. Kijelöljük a textfs-t. Megadjuka a kapott paramétereket a gradient-nek. Aztán összeillesztjük a képünket.

```
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
)))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
 height 2) (/ text2-height 2)))

;(gimp-selection-none image)
;(gimp-image-flatten image)

(gimp-display-new image)
(gimp-image-clean-all image)
```

Létrehozunk egy új szöveget majd hozzáadjuk új réteként. Majd egy új ablakban megjelenítjük a felugró képet.

# Chapter 10

## Helló, Gutenberg!

### 10.1 Juhász István: Magas szintű programozási nyelvek - olvasónapló

#### 1.2 Alapfogalmak

Megismerjük , hogy a programozási nyelveknek milyen 3 szinje van. A szintek : a gépi nyelv , az assembly szintű nyelv illetve a magas szintű nyelv. Mi a magas szintű nyelvvel fogunk részletesen foglalkozni. A magas szintű programozási nyelven írt programok a forráspontok és a forrásszövegek. A forrásszövegre vonatkozó nyelvtani szabályokat szintaktikának nevezzük. A tartalmi szabályok összességét pedig szemantikának. A magas szintű programozási nyelvet ez a két tényező fogja meghatározni. Ezeknek a szabályoknak az összefoglaló neve a hivatkozásui nyelv.

A forrásszövegből szükséges a processzor által értelmezhető gép kódá konvertálnunk a kódot , amire létféle technika áll rendelkezésünkre : fordítóprogram és interpreter. A fordítóprogram egy magas szintű programnyelven megírt kódból tárgyprogramot képes előállítani, a következő lépésekkel : lexikális elemzés , szintaktikai elemzés , szemantikai elemzés, kódgenerálás. A lépések végrehajtódása után megkapjuk a gépi kódunkat. Ebből a gépi kódból a kapcsolatszerkesztő állít elő futtatható kódot. Az interpreteres megoldás nem készít tárgyprogramot. Ez a két technika együttesen is használható. Emellett léteznek még előszeretettel használt IDE-k ( integrált fejlesztői környezetek) amitkomplex feladatkörrel láttak el a fejlesztők.

#### 1.3 A programnyelvek osztályozása

A programozási nyelvet két főcsoportra oszthatjuk. A két fő csoport : Dekleratív és Imperatív nyelvek.

A dekleratív nyelvek jellemzői : ezek a nyelvek nem algoritmikus nyelvek , nem kötődnek a Neumann architektúrához. Nincs bennük lehetőség memóriaműveletekre. Két alcsoportot különböztetünk meg : funkcionális és logikai.

Az imperatív nyelvek jellemzői : algoritmikus nyelvek, ami azt jelenti , hogy a leprogramozott algoritmus működteti a processzort. A program utasításokból áll. A legfőbb összetevői a változók. Alcsoportjai : eljárás és objektumorientált nyelvek

#### Alapelemek

##### 2.1 - Karakterkészlet

A program legkisebb alkotórészeit karaktereknek nevezzük. Karakterekből épülnek fel a bonyolultabb nyelvi elemek : lexikális és szintaktikai egységek , utasítások , programegységek , fordítási egységek , és maga a program. A karaktereket a programnyelvekben 3 különböző féle képpen csoportosíthatjuk : betűk , számjegyek , egyéb karakterek.

A fordító a lexikális elemzés során felismeri és tokenizálja a lexikális egységeket. Számos fajtát különböztetünk meg : többkarakteres szimbólum , szimbólikus név , címke , megjegyzés , literál stb.

A többkarakteres szimbólumok : A többkarakteres szimbólumoknak az esetek nagy részében a nyelv tulajdonít jelentést. Gyakran operátorok , pl : ++ , -- , stb.

A címke az utasítások jelölésére szolgál , hogy a program egy másik pontjáról hivatkozni lehessen rá.

A címke az utasítások jelölésére szolgál. A program egy másik pontjáról is lehet rá hivatkozni.

A megjegyzések szerepe fontos , mert ezeknek a megjegyzéseknek segítségével olyan dolgokat közölhetünk , ami segíti az olvasást és az értelmezést.

A literálok segítségével ,meg nem változtatható értékeket vezethetünk be a programkódunkba. Két része van : típus és érték.

## 2.4 - Adattípusok

Egy absztrakt programozási eszköz az adatabsztrakció legelelő megjelenési módja. Két fajtája van : típusos és nem típusos nyelvek . Az adattípus neve egy azonosító,majd ezt követi egy belső ábrázolási mód is. Egy adattípust 3 dolog határoz meg:

- tartomány
- műveletek
- reprezentáció.

Minden típusos nyelv rendelkezik beépített típusokkal, de egyes nyelvek engedélyezik a programozó által definiált saját típusokat is. Egy ilyen definiáláshoz meg kell adni a típus tartományát, műveleteit és reprezentációját. Az adattípusoknak két nagy csoporthoz tartozik:

- egyszerű,
- összetett típusok.

Az egyszerű típusok csoporthoz tartoznak: egész és valós típusok , karakteres típus, egyes nyelvek esetében a logikai típus. Speciális egyszerű típus a felsorolásos típus és a sorszámozott típus.

Az eljárásonorientált nyelvekben a két legismertebb összetett típus a tömb és a rekord. A tömb egy statikus és homogén típus.

### 2.4.3. Mutató típus

Tárcímeket tároló egyszerű típus, amivel megvalósítható az indirekt címzés. Fő feladata a megcímzett tárterületen található érték minél hamarabbi elérése. Ha nem mutat sehova , akkor NULL érték.

## 2.5 A nevesített konstans

A nevesített konstans 3 komponenssel rendelkezik : névvel , típussal és értékkel. Ezeket minden esetben deklarálni kell. Szerepük , hogy a gyakran előforduló értékeknek adhatunk neveket és ha az adott értéket megszeretnénk változtatni nagy segítséget nyújt ebben.

## 2.6 A változók

A változók olyan programozási eszközök aminek a komponensei a következők : név , attribútum , cím és érték. A név egy azonosító. Az attribútumok olyan jelmezők , amik a futás közbeni működését határozza meg a változóknak. A változóknak deklarációval értéket rendelhetünk. A változó címe a tárnak az a címe ahol megtalálhatjuk a változó értékét . Az értékkomponens a címen elhelyezkedő bitkombinációként mutatkozik meg.

## 2.7 Alapelemek az egyes nyelvekben

Két fő típust figyelhetünk meg : aritmetikai és származtatott típus. Aritmetikai az egyszerűek és a származtatot az összetett típusok .

### 3. Kifejezések

A kifejezések olyan szintaktikai eszközök , amik lehetővé teszik , hogy egy adott ponton a már eleve ismert értékekből új értékeket határozzunk meg. Kér részre bontjuk őket : érték és típus. A kifejezések összetevői : operandusok , operátorok , kerek zárójelek.

Az operandusok képviselik az értéket. Az operátorok a műveleti jelek. A zárójelek a műveleti sorrendet befolyásolják.

### 4. Utasítások

Az utasításoknak két fő típusa van : deklarációs és végrehajtható utasítások. A deklarációs utasítások a fordítóprogramnak szólnak.

A végrehajtható utasításoknak főbb típusai: értékeadó utasítás, üres utasítás, ugró utasítás, elágaztató utasítások, ciklusszervező utasítások, hívó utasítás, vezérlésátadó utasítások, Input/Output utasítások, egyéb utasítások.

Az értékeadó utasítás feladata a változó értékének beállítása.

Az üres utasításokat feladata átláthatóbb szerkezetűvé tenni a programot .

Az ugró utasítással pontról át lehet ugrani egy adott címkével jelölt utasításra.

Elágaztató utasítások:

Az elágaztató utasítások olyan , utasítások amiben a feltételvizsgálat eredménye dönti el, milyen művelet kerüljön végrehajtásra. A vizsgált feltétel egy logikai kifejezés.

Léteznek egy és több irányú utasítások.

Ciklusszervező utasítások:

Egy ciklusnak 3 fő része van: fej, mag és vég. A mag tartalmazza az utasításokat. A ciklon nem rendeltetésszerű működésének két iskolapéldája az üres ciklus, ami egyszer sem fut le és a végtelen ciklus ami "végtelen" ideig fut.

Feltételes ciklusok

Az egyik fajtája a kezdőfeltételes ciklus amely esetében a fejben lévő feltétel kiértékelődése után hajtódik végre a ciklusmagban található kód addig amíg hamis nem lesz a feltétel. A végfeltételes ciklus esetében a feltétel a ciklus végében van, esetében először a mag hajtódik végre és csak ezután értékelődik ki a feltétel.

Vezérlő utasítások

CONTINUE, BREAK , RETURN.

## A programok szerkezete

Programegységekről az eljárásorientált nyelvek esetében beszélhetünk, melyek a következőek : alprogram, blokk, csomag.

Az alprogramok az újrafelhasználás eszközei, akkor használjuk, ha többször szükséges elvégeznünk egy adott műveletet. Az alprogramot egyszer kell megírnunk, utána hivatkoznunk kell csak rá ott, ahol az eredeti programrész szerepelt volna.

A fejben található a név. Ez azonosítja az alprogramot. Itt taláható még a formális paraméterlista is mely kerek zárójelek között áll. A paraméterlista a a paraméterkiértékelés során játszik fontos szerepet.

A törzsben találhatóak a deklarációs és végrehajtható utasítások.

Az alprogram környezete a globális változóinak együttese. Két fajtája van: eljárás és függvény. Az eljárás hajta végre a tevékenységet , de nincs visszatérési értéke. A függvény egy tetszőleges típusú értékű eredményt ad vissza.

### 5.2 Hívási lánc, rekurzió

Akkor alakul ki a hívási lánc, ha egy programegység meghív egy másikat, az pedig egy következőt, és így tovább , tovább.

Ha egy aktív alprogramot hívunk meg, rekurzióról beszélünk, ami kétféle lehet. Ha egy alprogram önmagát hívja meg beszélhetünk közvetlen meghívásról. De ha a hívási láncban már szereplő alprogramot hívunk meg akkor közvetett meghívásról beszélünk.

### 5.4 Paraméterkiértékelés

A paraméterkiértékelés akkor figyelhető meg , amikor egy függvény vagy eljárás hívásnál megfigyelhető a mechanizmus, amely során egy alprogram formális- és aktuális paramétereit egymáshoz történő megfeleltetése megtörténik.

### 5.5 Paraméterátadás

Amikor paraméterátadásról beszélünk minden van egy hívó és egy hívott. A hívott minden az alprogram. Különböző paraméterátadási módokat ismerhetünk: érték-, cím-, eredmény-, érték-eredmény-, név- és szöveg szerinti. Érték szerinti paraméterátadáskor a formális paraméterek rendelkeznek címkomponenssel az alprogram területén. Esetében az információ áramlása egyirányú, működése során értékmásolás hajtódik végre.

### 5.6 A blokk

A blokk egy olyan programegység ami egy másik programegységen helyzékkelhet el, szerkezetileg van kezdete, törzse és vége. A blokkban megtalálható elemek egyértelműen meghatározhatók. Paraméterrel Nem rendelkeznek paraméterrel.

### 5.7 Hatáskör

Egy név hatásköre a programnak az a része, ahol az adott névvel ugyanarra az eszközre hivatkozunk.

A blokk általános alakja:

```
{
 deklaracioik
 vegrehajthato utasitasok
}
```

## 5.9 Az egyes nyelvek eszközei - C

A fejezet szemlélteti a blokk és függvény alakját.

## 6. Absztrakt adattípus

Az Absztrakt adattípusoknál nem ismerjük sem a reprezentációt sem pedig a műveletek implementációját , mert ezt az adattípus nem mutatja a külvilág számára. Hozzáférésük interfészeken keresztül történik.

## 10. Generikus programozás

A generikus programozás az újrafelhasználhatóság eszköze, ami egy olyan eszközrendszer ami a legtöbb nyelvbe beépíthető.Megadunk egy paraméterezhető forrásszöveg mintát . Amiből előállítható egy konkrét fordítható szöveg.

## 13. Input/output

Az Input/Output egy olyan eszköz rendszer ami felelős a perifériákkal való kommunikáció megvalósításáért és annak fenntartásáért , a memóriából adatokat küld a perifériákhoz vagy fordítva. Az állomány egy programban lehet fizikai illetve logikai . Az állományokat megkülönböztethetjük funkció szerint is , lehetnek: input állományok - csak olvasni lehet belőle; output - csak írni lehet bele; input-output - olvasni és írni is lehet. Az I/O során kétféle adatátviteli módot alkalmazunk amelyekkel az adatok a memória és a periféria közötti mozgásukat végzik: folyamatos vagy bináris.

A következő lépések szükségesek ,ha a programunkban állományokkal szeretnénk dolgozni:

- deklaráció
- összerendelés
- állomány megnyitása
- feldolgozás
- lezárás

## 9. kivételkezelés

Kivételről beszélünk olyan névvel és kóddal rendelkező események esetében, amelyek megszakítást okoznak.

## 10.2 KR: A C programozási nyelv - olvasónapló

### 2. fejezet: Típusok

A C nyelv adattípusai :

- char: mérete 1 byte, a karakterkészlet egy elemét tartalmazza
- int: egész szám
- float: lebegőpontos szám

- double:lebegőpontos szám

Aritmetikai operátorok : +, -, \*, / és %. Ezek a következők: összeadás, kivonás, szorzás, egész osztás és maradékos osztás operátorok.

Léteznek relációs és logikai operátorok is. A relációsak: >, >=, <, <=, =.

A logikai operátorok: || illetve && .

Inkrementáló és dekrementáló operátorokkal : ++ és --. Az inkrementáló operátor 1-et ad hozzá az operandumhoz míg a dekrementáló 1-et von ki belőle. Ezek az operátorok állhatnak prefix- vagy postfix operátorként.

Bitenkénti logikai operátorok: & , | , ^ , << , >> , ~

Értékadó operátorok : a könyv 58. oldalától kezdődően találkozhatunk.

### 3. fejezet: Vezérlési szerkezetek

#### 3.1 Utasítások és blokkok

A C nyelvben a pontosvessző az utasításokat lezáró jel. A kapcsos zárójelek segítségével a deklarációkat és utasításokat egy nagy blokkba lehet foglani, ami szintaktikailag egyetlen utasítással ekvivalens.

#### 3.2 Az if-else utasítás

Döntést, választást írunk le vele. A könyv 65. oldala fellelhető példa:

```
if (kifejezes)
 1. utasitas
else
 2. utasitas
```

#### 3.3 Az else-if utasítás

A könyv példája:

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

#### 3.4 A switch utasítás

Nagyon hasonló az if..else if..else szerkezethez, viszont olvashatóbb.Lásd a könyv 69. oldal :

```
switch (kifejezes)
case :
case :
case :
case :
case :
default :
```

### 3.5 A while és a for utasítás

A könyv példa:

```
while (kifejezes)
 utasítás

for (kifejezes1; kifejezes2; kifejezes3)
 utastas
```

alakú for utasítás egyenértékű a

A for bármely 3 kifejezése elhagyható. A for és a while ciklus között szabadon választhatunk. d

### 3.6 A do-while utasítás

```
do
 utasitas
 while (kifejezes);
```

### 3.7 A break utasítás

A break utasítással már a ciklusbeli feltételvizsgálat előtt is ki lehet ugrani a ciklusokból. A break utasítás eredményeképp a vezérlés a legbelső zárt ciklusból azonnal kilép.

### 3.8 A continue utasítás

A könyv 76. oldal:

```
for (i = 0; i < N; i++) {
 if (a [i] != 0) /*Páros elemek átugrása*/
 continue;
 }
 .
 .
 /*Páratlan elemek feldolgozása*/
```

A continue megkezdi a ciklus kövezekző iterációját.

### 3.9 A goto utasítás; címkék

A goto utasítás segítségével lehet címkére ugrani.

77. oldal:

```
for (. . .)
for (. . .) {
 .
 .
 if (zavar)
 goto hiba;
 .
 .
}
hiba: számold fel a zavart
```

Alapja megegyezik a változónevekével , de a kettőspont követi a nevüket.

## Utasítások

Utasítások fő fajtái:

- feltételes utasítás
- while utasítás
- do utasítás
- for utasítás
- switch utasítás
- break utasítás
- continue utasítás
- goto utasítás
- címkézett utasítás

#### A return utasítás

```
return;
return kifejezés;
```

Először határozatlan a visszadaott érték , másodszor a kifejezés értéke a visszaadott érék.

#### A nulla utasítás

Egy pontosvesszőből áll, amipéldául üres ciklustörzset is képezhet.

#### A kifejezés utasítás

Utasítások , függvényhívások vagy értékkadások.

#### Az összetett utasítás vagy blokk

A blokkok segítségével tudunk egy utasításból több utasítást képezni úgy , hogy az lényegében olyan legyen mintha még mindig egy utasítás lenne.

## 10.3 Benedek Zoltán, Levendovszky Tihamér: Szoftverfejlesztés C++ nyelven - olvasónapló

### A C++ nem objektumorientált újdonságai

#### 2.1 A C és a C++ nyelv

A C nyelvtől eltérően nem tetszőleges számú paraméterrel hívható ha a függvényt üres paraméterlistával definiáltunk , hanem akkor az olyan, mintha egy void paraméterrel definiáltuk volna.

#### 2.1 A main függvény

Két formája létezik C++-ban. Példa a 4.oldalon

#### 2.1.3 A bool típus

A C++ nyelvben megismerkedünk a bool típussal , ami logikai értéket tárol.

A C++-ban már beépített típusként kaptak helyet a C-ben már jól ismert , több-bájtos sztringek megvalósítására alkamas wchar\_t típus.

## 2.2 Függvények túlterhelése

A C nyelvben a függvényt a neve azonosítja. C++-ban : a név és az argumentumlista együttesen azonosít egy függvényt. Megismerkedünk linker névelferdítési technikája , amit azonos nevű függvények esetén alkalmazhatunk. Szó esik az `extern` kulcsszó használatáról , amit abban az esetben használhatunk , ha C-ben akarunk C++ függvényt meghívni.

## 2.3 Alapértelmezett függvényargumentumok

A C++ nyelvben, függvényeinknek meg lehet adni alapértelmezett értékeket.

## 2.4 Paraméterátadás referenciatípussal

A C nyelvben a paraméterátadás érték szerint lehetséges. C++ : létezik referenciával való paraméterátadás. Ez a cím szerinti paraméterátadás.

# Objektumok és osztályok

## 3.1 Az objektumorientáltság alapelvei

A programok összetetségének növekedése miatt már a kódok nagyon átláthatlanok lettek , lehetetlen volt már őket kezelni. Ezért terjedt el az objektumorientált programozás.

Az egységbe záró adatstruktúra neve osztály. Az osztálynak vannak egyedei, amiket objektumoknak nevezünk.Beszélünk róluk, ezek mellett az öröklődés és az egységbe zárás fogalmáról is. Ezek az objektumorientált programozás alapelvei.

## 3.2 Egységbe zárás a C++-ban

Átveszi a tagváltozók és tagfüggvények szerepét. A függvény rendelkezik egy láthatatlan, első paraméterrel, ami alapján tudni fogja, melyik struktúrát kell módosítania. A `this` kulcsszó.

## 3.3 Adatrejtés

A `private` kulcsszó. A `private` kulcsszó után álló változók és függvények csak osztályon belül láthatóak. Ellentéte a `public`.

A változó létrehozását az osztályból magyarul példányosításnak hívjuk, a példány más néven objektum.

## 3.4 Konstruktorok és destruktörök

A konstruktor olyan speciális tagfüggvény, példányosításkor hívódik meg. A destruktur felel az objektumok által lefoglalt erőforrások felszabadításáért , az objektum megszűnésekor automatikusan meghívódik.

## 3.5 Dinamikus adattagot tartalmazó osztályok

A `new` a dinamikus memória kezeléséért felelős operátor , majd felszabadítjuk a `delete` operátorral a lefoglalt helyet. Tömbök esetén: `new []` és a `delete []` használhatóak.

## 3.5.3 Másoló konstruktor

Egy referenciát vár, aminek típusa megegyezik az osztály típusával. Az újonnan létrehozott objektumot inicializáljuk egy már létező objektum alapján .

## 3.6 Friend függvények és osztályok

### 3.6.1 Friend függvények

A `friend` kulcsszó feljogosíthat függvényeket, hogy hozzáférhessenek az osztály védett tagjaihoz.

### 3.6.1 Friend osztályok

Lényegében megegyeznek a `friend` függvényekkel. Az osztályunk egy másik osztályt jogosít fel a védett tagjaihoz való hozzáférésre.

### 3.7 Tagváltozók inicializálása

Az inicializálás és az értékadás különbség : Az inicializálás konstruktorhívás történik. Értékadás esetén az `=` operátor hívódik meg.

### 3.8 Statikus tagok

Nem az objektumhoz hanem az osztályhoz tartozó tagváltozók. A statikus tagfüggvények objektum nélkül, az osztály nevén keresztül is használhatók. A `this` mutató statikus függvények esetében nem használható.

### 3.9 Beágyazott definíciók

A C++ nyelven belül lehetőségünk van az osztály, struktúra, típusdefiníciók osztályon belüli megadására. Ezt beágyazott definíciót nevezzük.

## Operátorok és túlterhelésük

### 6.1. Az operátorokról általában

Az átadott argumentumokon végez el a műveletet , majd az eredményt visszatérési értékként adjuk meg. Mellékhatásnak nevezzük azt a jelenséget amikor az argumentum értékét megváltoztatja az operátok. A zárójel használata a bonyolultabb kifejezéseknel fontos.

### 6.2 Függvényszintaxis és túlterhelés

A C nyelvben nem lehetséges , csak ha a változóra mutatót adunk át. C++-ban ez lehetséges, a referencia szerinti paraméterátadással. 94. oldal:

```
int noveles(int& ertek)
{
 int eredmény = ertek;
 ertek = ertek+1; //mellékhatás
 return eredmény;
}
```

C++-ban az operátorok függvényszintaxissal történő meghívása esetén is van lehetőség:

```
z = x+y;
z = operator+ (x, y);
```

## C++ sablonok

Vannak olyan osztály- és függvény sablonok, amiknek néhány elem definiáláskor paraméternek veszünk. Gyakori az alkalmazásuk : tetszőleges típusú elemek tarolására alkalmas tároló osztályok létrehozásánál.

### 11. 1 C++ függvény sablonok

A függvényt sablonná alakítjuk át. A típus amivel dolgozunk, nem rögzített,a sablon felhasználásakor adjuk meg. A könyvben található példa:

```
template <class N> inline N max (N lhs, N rhs)
{
 return lhs > rhs ? lhs: rhs;
}
```

Felhasználása:

```
int y = max(3, 5);
double z = max(3.1, 5.5);
```

### 11.1.2 Példák függvény sablonokra

A sablonparaméter nem csak típus, hanem lehet típusos konstans is. pl.:

```
template <int N> int Square()
{
 return N*N;
}
int main()
{
 const int x= 10;
 cout <<x<<"négyzete:" <<Square<x>() << endl;
}
```

### 11.1.3 A hívott függvény kiválasztása

Ha függvénynek több implementációja van , akkor bizonyos szabályok döntik el annak kiválasztásakor, melyik függvény kerül meghívásra.

## 11.2.1 Osztálysablonok írása

1. lépés: Osztálydefinícióból sablondefiníció

2. lépés: Sablonparaméterek bevezetése

3. lépés: Nem implicit inline definiált tagfüggvények szintaktikája

4. lépés: Osztálynév helyett osztálynév ÉS sablonparaméterek szerepeltek a stípusként való felhasználás során

## A C++ I/O alapjai

### 5.1 A szabványos adatfolyamok

A C nyelvben 3 fájlleíró áll rendelkezésünkre : `stdin`, a `stdout` és `stderr` , ezek `FILE*` típusúak, magas szintű állományleírók.

A C++ nyelv adatfolyamokban gondolkodik. Az `istream` típusú objektumok alatt csak olvasható adatfolyamokat értünk, `ostream` típusú alatt pedig csak írhatókat. Ezek az adatfolyamok az `<<` és `>>` operátorokat használják beolvasásra és kiíratásra.

Ahhoz, hogy használni tudjuk az I/O-t C++-ban, includoljuk az `iostream` állományt. Mivel egy objektum az adatfolyam , így az állapotát beállító konstansok: `eofbit`, `failbit`, `badbit`, `goodbit`. Az első 3 minden különböző hibát jelez vissza, az utolsó a helyes működésről tájékoztat. A C és C++-beli, beolvasást megvalósító függvények összehasonlítása is megtörténik a 79. oldalon. Ugyanezen az oldalon a `getline` függvény is említésre kerül, amit stringek beolvasásakor használunk arra, hogy a beolvasás a szóköznél ne szakadon meg .

### 5.2 Manipulátorok és formázás

A manipulátor egy speciális objektum, amelyet az adatfolyamokra alkamazunk a be és ki meneti operátorok argumentumaként. Az előre definiált manipulátorok az `iomanip` fájlból találhatók amit programunkba

inkludálnunk kell ha használni akarunk. Manipulátorokra példa: `endl`, `flush`, `noskipws`, `setw`. Jelzők és maszk fogalmak megjelenése. A leggyakoribb manipulátorok és tagfüggvényeik összefoglaló táblázata a 84-85. oldalon található meg.

### 5.3 Állománykezelés

A C++ programnyelv az állománykezeléshez is adatfolyamokat használ, amiket az `ifstream`, az `ofstream` és az `fstream` osztályok reprezentálnak. Az állományok megnyitását konstruktorok, míg bezárását a destruktork végzik.

#### Kivételkezelés

A kódban a hibakódok számára egy külön osztály került létrehozásra melyben a kódok definíálása történt meg. A kivételkezeés az előnyös megoldás, amely számunkra lehetővé teszi azt, hogy kivétel esetén a vezér-lés a kivételkezelőhöz kerüljön. A main-ben található try-catch szerkezet try részébe a helyes működés- míg a catch részébe a kivételkezelő kódja kerül.

## **Part III**

# **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.

# Chapter 11

## Helló, Arroway!

### 11.1 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.! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/> (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

A polár generátor projekt egy klasszikus OO bevezető példa, ezért most ezt fogjuk megtekinteni.

```
boolean nincsTárolt = true;
double tárolt;
%%
{}
```

A class elején deklarálunk két változót , az egyik a nem visszadott adatot tárolja, a másik egy logikai kifejezés, ami arra mutat rá, hogy van -e tárolt, vagy futtatnunk kell az algoritmust újra?

```
if (nincsTárolt)
{
double u1, u2, v1, v2, w;
do
{
u1 = Math.random();
u2 = Math.random();
v1 = 2 * u1 - 1;
v2 = 2 * u2 - 1;
w = v1 * v1 + v2 * v2;
%%
{}
```

Az algoritmusunk elején megnézzük, hogy a logikai kifejezésünk igaz-e?

Ha igen akkor nézzük az If feltétel vizsgálatunk belsejét : Deklarálunk 2 változót amiket random számokkal határozzunk meg. Ezután néhány matematikai műveletet végzünk el rajta, egészen addig amíg a "w" nagyobb nem lesz mint 1 ( a w = a két szám kétszerese).

```
 } while (w > 1);
double r = Math.sqrt((-2 * Math.log(w)) / w);
tárolt = r * v2;
nincsTárolt = !nincsTárolt;
return r * v1;
}
%%
%
```

A tárol adattagban elmentjük a legelőször létrehozott szorzatot (r) , majd a a logikai változót az ellenkezőjére állítjuk be (!nincsTárolt). Ezután visszatérítjük a második random kétszeresét.

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

%%
}%
```

Lefut az else ág , mert megváltoztattuk a logikai változónkat, ahol a "tárolt" adatainkat tároltuk.

Fordítjuk majd futtatjuk a programunkat.

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac PolarGenerator.j
ava
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java PolarGenerator
-2.0636463212613405
0.46552879319473883
-0.16883200765268677
0.46238114238311945
0.20110573963436357
-1.6388174309834924
0.5100419512250719
0.10944446373077454
0.1463119874810178
0.6513551543788535
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$
```

## 11.2 "Gagyí"

Az ismert formális 2 „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 forrására 3 , hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás videó:

Megoldás forrása:<https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/gagyi128.java>

<https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/gagyi129.java>

Ebben a feladatban a „while ( $x \leq t \&& x \geq t \&& t \neq x$ );” ciklusra vagyunk kiváncsiak. Hogyan lehetséges az, hogy van olyan szám amelynél végtelen ciklusba kerül miközben az értéket vizsgálja. Eközben egy másiknál befejeződik a ciklus, a referenciakat vizsgálva.

-129-nél : Mindkét objektumban tárolt érték -129 ezért  $x \leq t$  és  $x \geq t$  teljesülni fog. De a két integerben két objektum más-más címmel rendelkezik, ezért az egyenlőségük nem teljesülhet. A while ciklusunk ekkor végtelen ciklusba kerül, mert a programunk  $x \neq t$  része teljesülni fog.

```
public class gagyi129
{
 public static void main (String[] args)
 {
 Integer x = -129;
 Integer t = -129;
 System.out.println (x);
 System.out.println (t);

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

Nézzük :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac gagyi129.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java gagyi129
-129
-129
```

Láthatjuk, hogy végtelen ciklusba kerültünk.

-128-nál : Ebben az esetben is teljesülni fog a  $x \leq t$ ,  $x \geq t$ . Ebben az esetben ugyanarra a számra ugyanazt az objektumot kapjuk, ebben az esetben már az objektumot előre elkészített pool-ból kapjuk, referenciájuk megegyezik. Az objektumot felhasználva az azonos érték miatt azonos lesz. A while ciklusban az  $x \neq t$  hamis lesz, emiatt a while ciklus leáll és nem kerülünk végtelen ciklusba.

```
public class gagyi128
{
 public static void main (String[] args)
 {
 Integer x = -128;
 Integer t = -128;
 System.out.println (x);
 System.out.println (t);

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

Nézzük :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac gagyi128.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java gagyi128
-128
-128
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$
```

Láthatjuk, hogy ebben az esetben nem kerültünk végtelen ciklusba.

Vizsgáljuk meg JDK integer.java forrását. Nézzük az érintett sorokat :

```
public static Integer valueOf(int i) {
 if (i >= IntegerCache.low && i <= IntegerCache.high)
 return IntegerCache.cache[i + (-IntegerCache.low)];
 return new Integer(i);
}
```

A -128 még a valueOf metódus definíciójának intervallumában található ezért egy poolból fogjuk megkapni a memóriacímre hivatkozó Integer objektumokat. Emiatt nem teljesül a feltétel és nem kerülünk végtelen ciklusba.

A -129 esetében az érték már az előbb említett intervallumon kívül esik ezért

```
return new Integer(i);
```

fog lefutni. Ennek köszönhetően két különböző Integer objektum fog létrejönni különböző címmekkel, így a feltétel minden teljesülni fog, ezért végtelen ciklusba kerülünk.

## 11.3 Yoda

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

Megoldás videó:

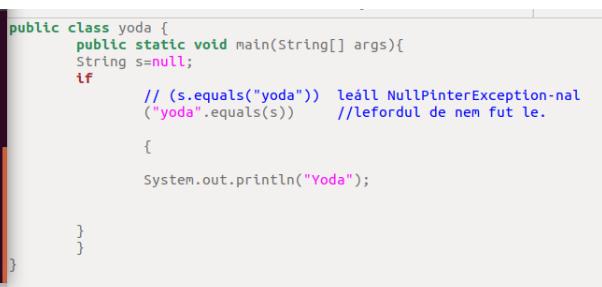
Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/yoda.java>

Mi is az a "Yoda körülmények"? A Yoda körülmények az utasítás rendjétől függően két egymástól különböző képpen tud lefordulni a program.

Ebben az állapotban a konstans , a feltételes utasítás a bal oldalon , míg a változó a bal oldalon fordul elő.

Nevét a Star Wars világhírű filmsorozatról kapta, ahol az említett "körülmény" névadója, Yoda mester beszéd közben nem tartotta be az angol nyelv nyelvtanát.

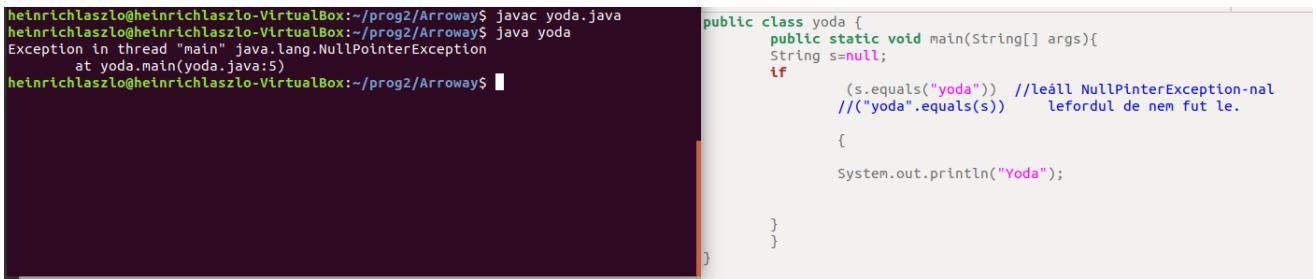
```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac yoda.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java yoda
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$
```



```
public class yoda {
 public static void main(String[] args){
 String s=null;
 if
 // (s.equals("yoda")) leáll NullPointerException-nal
 ("yoda".equals(s)) //lefordul de nem fut le.

 {
 System.out.println("Yoda");
 }
 }
}
```

A program lefordul. Majd a yoda "körülmények" elhagyása után leáll NullPointerException-nel, és nem fut tovább.



```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac yoda.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java yoda
Exception in thread "main" java.lang.NullPointerException
 at yoda.main(yoda.java:5)
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$
```

```
public class yoda {
 public static void main(String[] args){
 String s=null;
 if
 (s.equals("yoda")) //leáll NullPointerException-nal
 //("yoda".equals(s)) lefordul de nem fut le.
 {
 System.out.println("Yoda");
 }
 }
}
```

## 11.4 Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: [https://www.tankonyvtar.hu/hu/tartalom/tkt/javat/tanitok-javat/apbs02.html#pi\\_jegyei](https://www.tankonyvtar.hu/hu/tartalom/tkt/javat/tanitok-javat/apbs02.html#pi_jegyei) (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átelnél).

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/bbp.java>

A Számítás a BBP vagy a Bailey-Borwein-Plouffe algoritmus segítségével történik.

A BBP algoritmus segítségével az előző betük ismerete nélkül tudunk meghatározni a Pi szájegyeit hexadecimális számrendszerben.

Nézzük meg az algoritmust :

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

Következzen a kód :

```
public PiBBP(int d) {
 double d16Pi = 0.0d;

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

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

 d16Pi = d16Pi - StrictMath.floor(d16Pi);

 StringBuffer sb = new StringBuffer();

 Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};

 while(d16Pi != 0.0d) {
 int jegy = (int)StrictMath.floor(16.0d*d16Pi);
```

```
 if(jegy<10)
 sb.append(jegy);
 else
 sb.append(hexaJegyek[jegy-10]);

 d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
}

d16PiHexaJegyek = sb.toString();
}
```

Létrehozzuk az algoritmust használó objektumunkat, illetve kiszámítjuk az algoritmushoz szükséges értékeket.

```
public 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);

 return d16Sj - StrictMath.floor(d16Sj);
}
```

A fenti blokkban látható metódusunk segítségével kiszámításra kerülnek a képletünk további részei.

```
public 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;
}
```

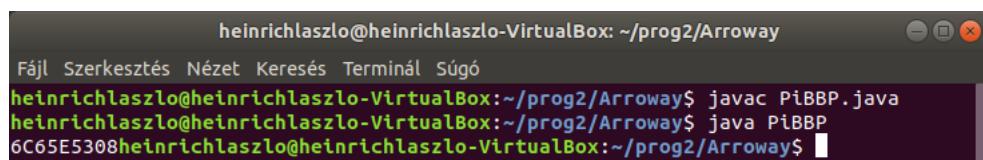
Megtörténik a bináris hatványozás.

```
public String toString() {

 return d16PiHexaJegyek;
}
public static void main(String args[]) {
 System.out.print(new PiBBP(1000000));
}
}
```

A fenti blokkban megtörténik a hexa-számjegy kiiratás illetve az objektum példányosítás.

Fordítás és futtatás után,a következő eredményt kapjuk.



The screenshot shows a terminal window titled "heinrichlaszlo@heinrichlaszlo-VirtualBox: ~/prog2/Arroway". The window has standard Linux-style window controls at the top right. The terminal interface includes tabs for "Fájl", "Szerkesztés", "Nézet", "Keresés", "Terminál", and "Súgó". The command history shows the user running "javac PiBBP.java" followed by "java PiBBP", which outputs the value "6C65E5308".

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac PiBBP.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java PiBBP
6C65E5308heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ █
```

# Chapter 12

## Helló, Berners-Lee!

### 12.1 Forstner Bertalan, Ekler Péter, Kelényi Imre : Bevezetés a mobil programozásba

A Python egy dinamikus, objektumorientált és platformfüggetlen. Összetettebb problémák megoldására is használható, de inkább prototípusok tesztelésére és készítésére használják a fejlesztési fázisban.

A programozók munkáját nagyban könnyíti, hogy a forrást nem kell állandóan menteni és fordítani, hanem elég ha az interpreter megkapja, ami azonnal fordítja és futtatja azt.

A Python az alkalmazás fejlesztők munkáját is megkönnyíti köszönhetően annak, hogy rengeteg modul található benne és maga a nyelv nagyon könnyen elsajátítható.

A python programokra ha laikus szemmel ránézünk az első ami szembetűnik, hogy sokkal rövidebb és átláthatóbb, mint egy Java illetve C/c++ program.

A nyelv behúzásokra épül, ezeket szóközökkel vagy tabulátorokkal érjük el. Leegyszerűsítve a minden utasítás egy-egy sor. Nincs szükségünk zárójelekre vagy ";"-re A Python minden sort tokenekre (azonosító, kulcsszó, operátor stb.) oszt, amik között whitespace helyezkedik el.

A megjegyzéseket a "#" -tel hozhatjuk létre.

Pythonban az objektumok reprezentálják az adatokat. A rajtuk végezhető muveletek tehát az objektum típusától függnek. Ennek megadására nincs szükség, ugyanis az interpretes valós időben határozza meg az adott változó típusát. pl. Számok, Stringek stb.. A számoknak lehetnek például: egészek (leehtnek oktális, hexadecimális, decimális), lebegő pontosak, vagy komplexek (pl C-ben a double típus). Stringeket idézőjelek és aposztrófok közé írjuk.

### 12.2 C++ és Java nyelv összehasonlítása

Az első hasonlóság, a C++ és a Java között, hogy minden estében objektum orientált nyelvről beszélünk.

De mi az az objektumorientált nyelv? Leegyszerűsítve annyit jelent, hogy a program amit megraktunk Java nyelven objektumokból és osztályokból áll.

Mi az az osztály? Az osztályt metódusok és változók alkotják, ez ismerős lehet a C++ nyelvből is.

A könyv elején találkozhatunk egy régi jó baráttal, a "Hello World"-del. A könyvünk ez alapján mutatja be, hogyan mukodik a java fordítója. Itt találkozhatunk az első különbséggel a Java és C++ között, ugyanis a Java egy Java Virtuális Gépet használ a bájtkódok értelmezéséhez és ennek segítségével fordítja le a kódunkat.

Ha elindítjuk a programunkat , akkor a fordító program, az úgynevezett virtuális gép a megadott osztály main metódusát fogja fordítani.

A könyvben a szerző ismerteti az osztályokat , változókat, a megjegyzéseket, illetve a konstansokat.

Utóbbiról annyi ,hogy megadása a final szóval lehetséges.

A megjegyzésnek két fajtáját ismerjük, létezik egysoros illetve több soros megjegyzés. Az egysoros jelölése "//" , emellett a több soros jelölése a következő : a megjegyzésünk elején "/\*" jelölést , majd a végén "\*/" jelölést alkalmazunk.

A megjegyzések a program, illetve a közös program egy elhagyhatatlan része, nagyban megkönnyíti a másokkal közös munkát.

Az osztályok : Az osztálynak az adattagjait és metódusait bármilyen sorrendben felsorolhatjuk. A Javaban a class szóval tudunk létrehozni osztályokat.

A könyv a példában a String nevezetű tipust hozza fel.

Változók : több tipussal találkozhatunk : pl.: int ( 32 bites egész szám), long (64 bites egész szám) , char (16 bites karakter) boolean (igaz vagy hamis).

A kivételkezelés a Javaban a C++ból már ismert try-catch szerkezet.

# Chapter 13

## Helló, Liskov!

### 13.1 Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_1.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf) (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/liskovhely.cpp>

<https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/liskovhely.java>

Lefutni lefut, de nem működik. A probléma nagyon leegyszerűsítve a következő : Vegyük a madarakat, a madarak tudnak repülni. Vegyük a pingvint, a pingvin nem tud repülni , pedig madár!!!

Ha P osztály M osztály leszármazottja, akkor P -t szabadon be tudjuk helyettesíteni minden egyes olyan helyre , ahol M típust várunk, hogy a program helyes futtatását eredményezze.

Az előbb felhozott madaras példa ennek a megsértésére tökéletesen működik. Az előbbi elv alapján, ha P lesz pingvin és M lesz madár, akkor ebben az esetben a repülési készséghez be kellene tudnunk helyettesíteni P-nek, mivel a pingvin madár és köztudottan a madarak tudnak repülni. Ennek ellenére azt is tudjuk ,hogy a pingvin egy röpképtelen madár így nem helyettesíthető be.

A leírtakat összegezve a pingvines madaras példánk sérti a Liskov elvet.

```
#include <iostream>

using namespace std;

class Madar {
public:
 virtual void repul() {
 cout<<"Tudok repülni!"<<endl;
 }
};
```

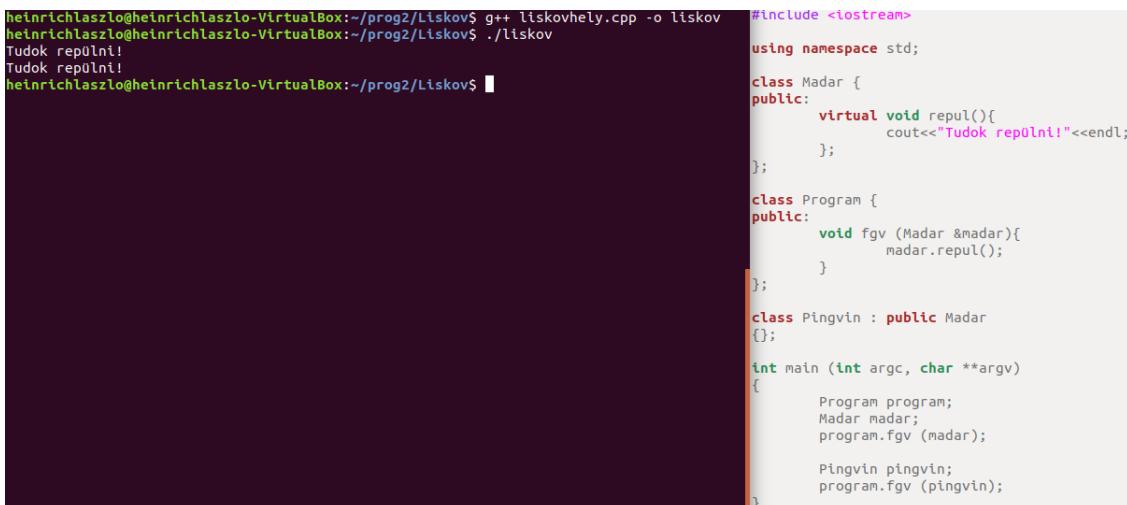
```
class Program {
public:
 void fgv (Madar &madar) {
 madar.repul();
 }
};

class Pingvin : public Madar
{};

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

 Pingvin pingvin;
 program.fgv (pingvin);
}
```

Létrehozzuk a Madar osztályt ami kap egy repul függvényt. Majd létrehozzuk a Program osztályt, amiben lesz egy fvg függvény. Az fvg függvény feladata lesz a repul függvény meghívásra egy Madar tipusu egyedre. Létrehozzuk a Pingvin osztályt, aminek szülőosztálya a Madar osztály lesz. Az utolsó lépés a példányosítás és a függvények meghívása.



```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ liskovhely.cpp -o liskov
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$./liskov
Tudok repulni!
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$
```

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

class Madar {
public:
 virtual void repul(){
 cout<<"Tudok repulni!"<<endl;
 };
};

class Program {
public:
 void fgv (Madar &madar){
 madar.repul();
 }
};

class Pingvin : public Madar
{};

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

 Pingvin pingvin;
 program.fgv (pingvin);
}
```

Írjuk át a kódunkat Javara :

```
class Madar
{
public void repul()
{
 System.out.print("Tudok repulni!\n");
}

class Pingvin extends Madar
```

```
{
}

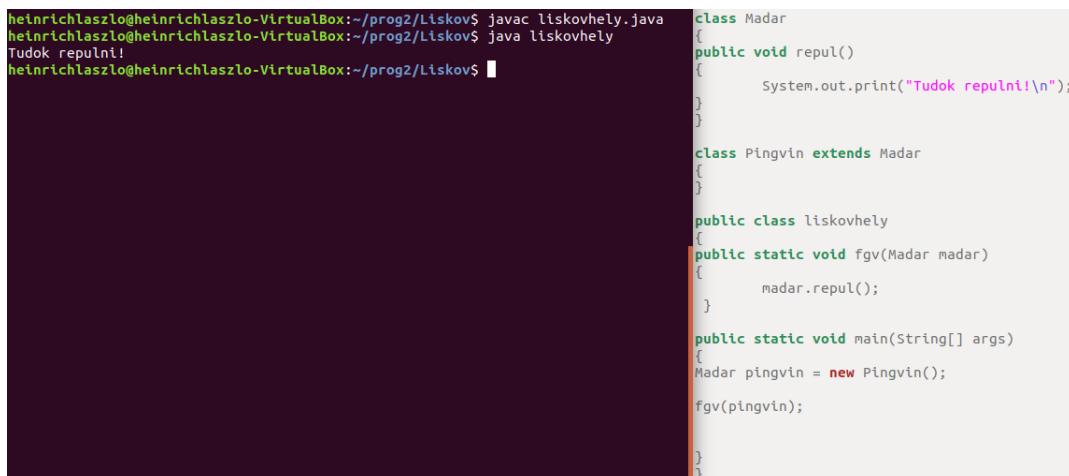
public class Liskovsert
{
 public static void fgv(Madar madar)
 {
 madar.repul();
 }

 public static void main(String[] args)
 {
 Madar pingvin = new Pingvin();

 fgv(pingvin);

 }
}
```

Forditsuk majd futtasok a programot



```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac liskovhely.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ java liskovhely
Tudok repulni!
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ ■
```

```
class Madar
{
 public void repul()
 {
 System.out.print("Tudok repulni!\n");
 }

 class Pingvin extends Madar
 {
 }

 public class liskovhely
 {
 public static void fgv(Madar madar)
 {
 madar.repul();
 }

 public static void main(String[] args)
 {
 Madar pingvin = new Pingvin();

 fgv(pingvin);
 }
 }
```

Probléma nélkül lefut a programunk és újra megreptetjük a mi röpképtelen madarunkat.

## 13.2 Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek! [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_1.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf) (98. fólia)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/szgy.cpp>

<https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/szgy.java>

A feladat lényege, hogy írunk egy olyan programot , amely bemutatja, hogy az ősön keresztül csak az ős üzenetei küldhetőek.

Mit értünk ez alatt ?

Leegyszerűsítve a gyermek osztály használhatja , az ōs osztály változóit , metódusait. DE , ez a fordítva nem igaz, az ōs osztály nem tudja értelmezni a gyermekosztályban definiált metódusokat , változókat stb.

```
public class szgy {
 class Szulo{
 }

 class Gyerek extends Szulo
 {
 public void viselkedik()
 {
 }
 }
 public static void main(String[] args)
 {
 szgy SzGy = new szgy();

 Szulo szulo = SzGy.new Gyerek();
 Szulo.viselkedik();

 }
}
```

Létrehozásra kerül a Szulo osztály. Majd létrehozásra kerül a Gyerek osztály, ami a Szulo osztály leszármazotja. A származtatás az extends kulcsszóval történik. Létrehozzuk a viselkedik függvényt. Következik a main. MAjd végül meghívjuk a szülőre a gyerek metódusát.

Fordítjuk majd futtatjuk a programot.

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac szgy.java
szgy.java:16: error: cannot find symbol
 Szulo.viselkedik();
 ^
 symbol: method viselkedik()
 location: class szgy.Szulo
1 error
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$
```

Az alosztályunkban meghívott függvények nem hazsnálhatóak a Szulo osztályon, ezért a programunk hibát is dob ki.

Írjuk át a kódunkat C++-ra :

```
#include <iostream>

using namespace std;

class Szulo
{

};

class Gyerek: public Szulo
```

```
{
void viselkedik()
{
cout<<"Viselkedek!";
}
};

int main ()
{
Szulo* sz= new Gyerek;

cout<<sz->viselkedik();
}
```

Fordítjuk a programot :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ gcc szgy.cpp -o szgy
szgy.cpp: In function 'int main()':
szgy.cpp:22:11: error: 'class Szulo' has no member named 'viselkedik'
 cout<<sz->viselkedik();
 ^~~~~~
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ █
```

A fordító ismét hibát jelez , tehát újra bebizonyítottuk , hogy a szülőosztály nem tudja az alosztályának a metódusait értelmezni.

### 13.3 Anti OO

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/PiBBPBench678.c>

<https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/PiBBPBench678.java>

<https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/PiBBPBench678.cpp>

A feladatunk az, hogy a programunk segítségével, határozzuk meg a pi számjegyeinek  $10^6$ ,  $10^7$  és  $10^8$  darab jegyét.

Ezt a BBP algoritmus segítségével fogjuk végre hajtani, majd lemérjük ezeknek a futási idejét, és megnézik melyik a leggyorsabb.

A programunk for ciklusának d ciklusváltozóját kell módosítanunk attól függően , hogy Pi hanyadik számjegyét szeretnénk megkapni

```
for (d = 1000000; d < 1000001; ++d)
```

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.c -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$./pi
6
1.510720
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.cpp -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$./pi
6
1.51042
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac PiBBPBench678.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ java PiBBPBench678
6
1.35
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$
```

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.c -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$./pi
7
18.076774
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.cpp -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$./pi
7
17.7307
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac PiBBPBench678.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ java PiBBPBench678
7
15.874
```

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.c -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$./pi
12
205.863604
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.cpp -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$./pi
12
205.257
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac PiBBPBench678.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ java PiBBPBench678
12
180.743
```

Mint látjuk mind 3 esetben a java verzió futott le leghamarabb , ezt követte a c++ majd a c verzió.

# Chapter 14

## 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). <https://arato.inf.unideb.hu/batfai.norbert/> (28-32 fólia)

Megoldás videó:

A feladatunk az volt , hogy UML osztálydiagrammot készítsünk az első védési feladathoz, azaz a binfához.

De mi is az az UML ?

Az UML (Unified Modelling Language) egy egységes modellezőnyelv. A grafikus ábrázoláshoz szükséges eszközök gyűjteménye.

Az UML meghatározott jelölésrendszerrel dolgozik, így könnyen kezelhető és értelmezhető az így készített modell. Az osztályokat , interfészket és azoknak a kapcsolatait írhatjuk le vele.

Az egyik legfontosabb tulajdonsága ,hogy jól átlátható, a privát és public változók külön-külön vannak szedve. Ha ezt egy táblázatnak vesszük , akkor a legfelső sorban helyezkedik el a class neve, alatta az attribútumok és a legalján az operátorok.

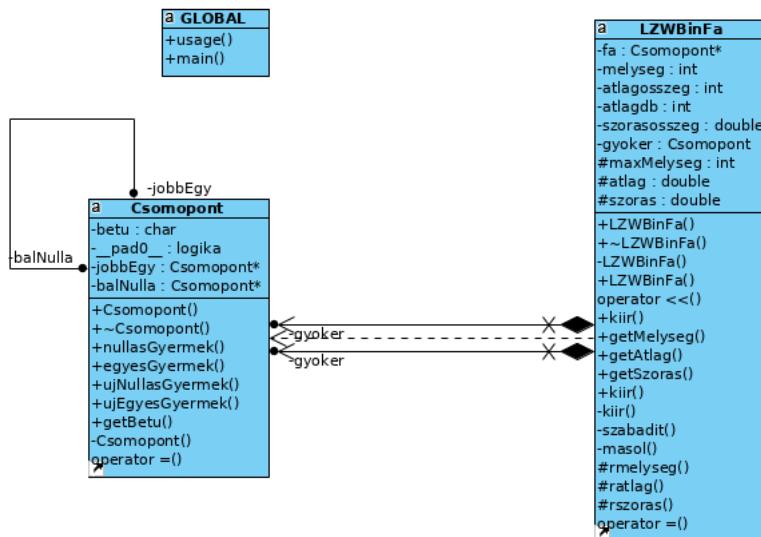
Az "oszlopok" között nyilakat láthatunk , ezeket társításnak nevezzük. Az üres négyzet fejű nyíl a jelöli az aggregációt , ami egy adott osztály megalakulását jelenti. A teli négyzet fejű nyíl ennek az erős változatát jelöli , ezt kompozíciónek nevezzük, ekkor ezek a részek önmagában nem léteznek , együtt jönnek létre és együtt is szűnnek meg. A kör fejű nyíl elhatárolást jelent. A Nyilak mellett megjegyzéseket tekinthetünk meg.

Térjünk ki kicsit részletesebben a kompozíció és aggregáció kapcsolatára :

Aggregációról beszélünk , ha egyik objektum részben vagy egészben is tartalmazza a másikat. Két típust tudunk megkülönböztetni : erős és gyenge aggregációt. A gyenge aggregációt nevezzük kompozícióknak. Ha kompozícióról beszélünk , akkor a tartalmazó és tartalmazott objektum nagyban függenek egymástól, egyik a másik nélkül nem tud funkcionálni.

Az UML generálásához a Visual Paradigmot használtam.

Lássuk :



## 14.2 Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Megoldás videó:

A feladatunk az volt, hogy UML-ben tervezünk osztályokat és forrást generálunk belőle.

A feladatban az előzőleg legenerált UML-t generáltam vissza forráskóddá.

Lássuk :

```

#ifndef LZWBINFA_H
#define LZWBINFA_H

class LZWBInFa {

private:
 Csomopont* fa;
 int melyseg;
 int atlagosszeg;
 int atlagdb;
 double szorasosszeg;
protected:
 Csomopont gyoker;
 int maxMelyseg;
 double atlag;
 double szoras;

public:
 LZWBInFa();
}

```

```
void ~LZWBinFa();

void kiir();

int getMelyseg();

double getAtlag();

double getSzoras();

void kiir(std::ostream& os);

private:
 LZWBinFa(const LZWBinFa& unnamed_1);

 void kiir(Csomopont* elem, std::ostream& os);

 void szabadit(Csomopont* elem);

protected:
 void rmelyseg(Csomopont* elem);

 void ratlag(Csomopont* elem);

 void rszoras(Csomopont* elem);
};

#endif

%%
```

## 14.3 BPMN

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

Megoldás videó:

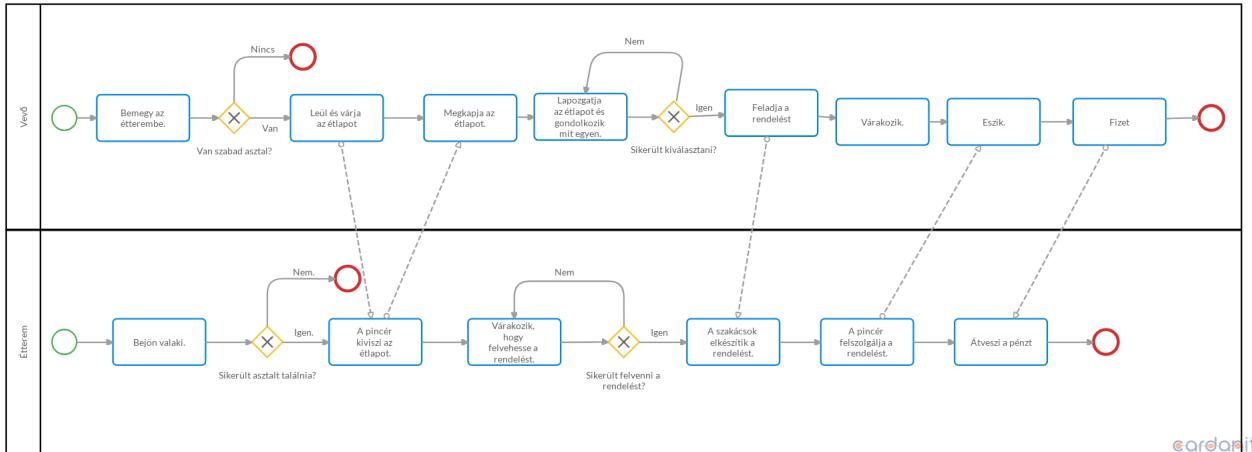
Megoldás forrása : >[`https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\_7.pdf`](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_7.pdf)

A feladatunk az volt, hogy rajzolunk le egy tevékenységet BPMN-ben.

De mi is az a BPMN ?

A BPMN ( Business Process Model and Notation) egy folyamatábra lényegében , ami grafikai reprezentációk modellezésére szolgál üzleti folyamatok részére. Leegyszerűsítve a BPMN egy egyszerű grafikai modellező eszköz.

Nézzünk meg egy egyszerű projektet.



A képen egy éttermi étkezés folyamatát figyelhetjük meg. Először bemegyünk az étterembe és megnézzük, hogy van-e szabad asztal, ha van leülünk ha nincs akkor itt az esemény vége. Rendelünk a pincértől és a rendelésünket a szakácsok készítik el. A pincér felszolgálja a rendelést, én megeszem , kifizetem , a pincér átveszi a pénzt majd itt az esemény vége.

# Chapter 15

## 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 ékezetes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Megoldás videó:

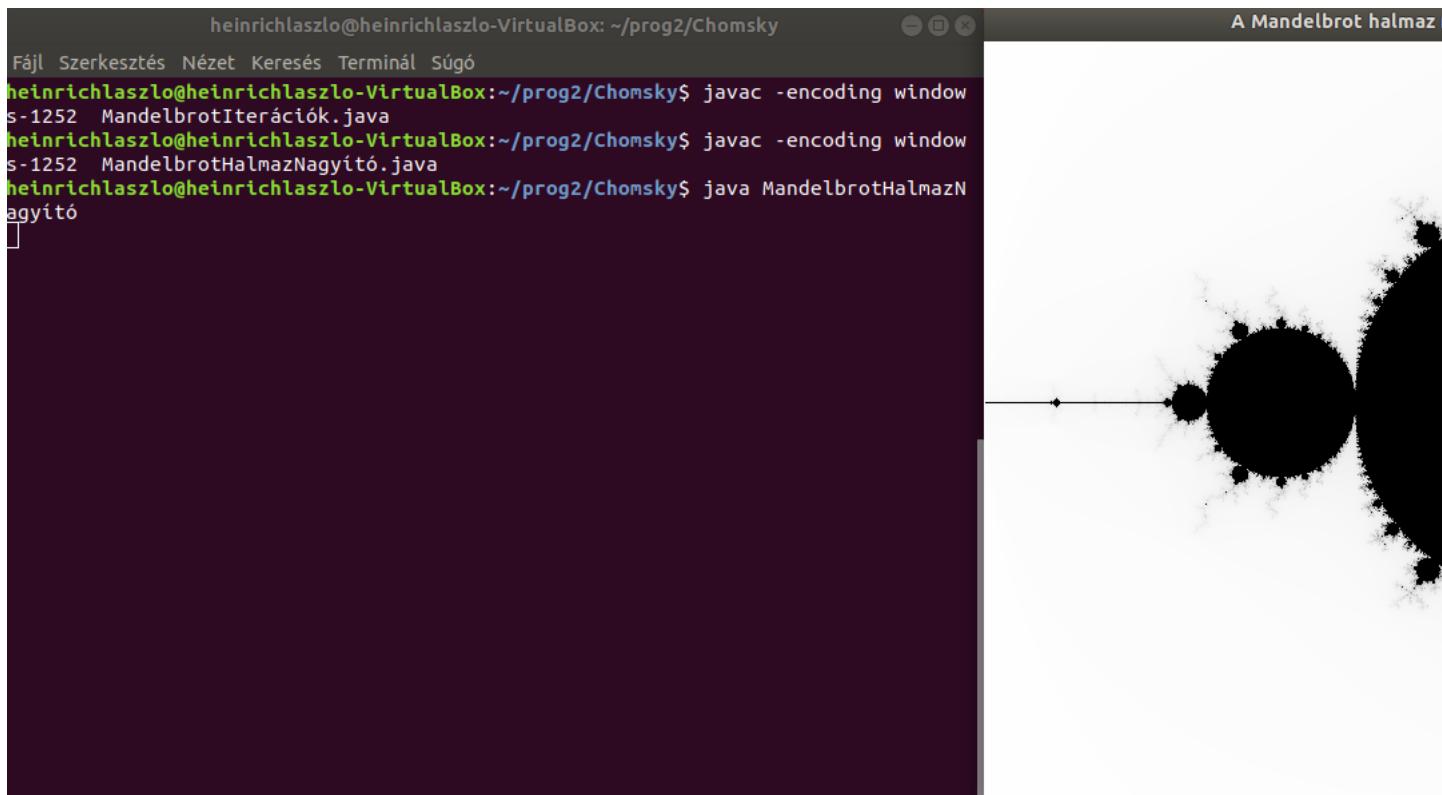
Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Chomsky>

A feladatunk az volt , hogy forditsuk és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és magában a forrásban is meghagyjuk az ékezetes betüket.

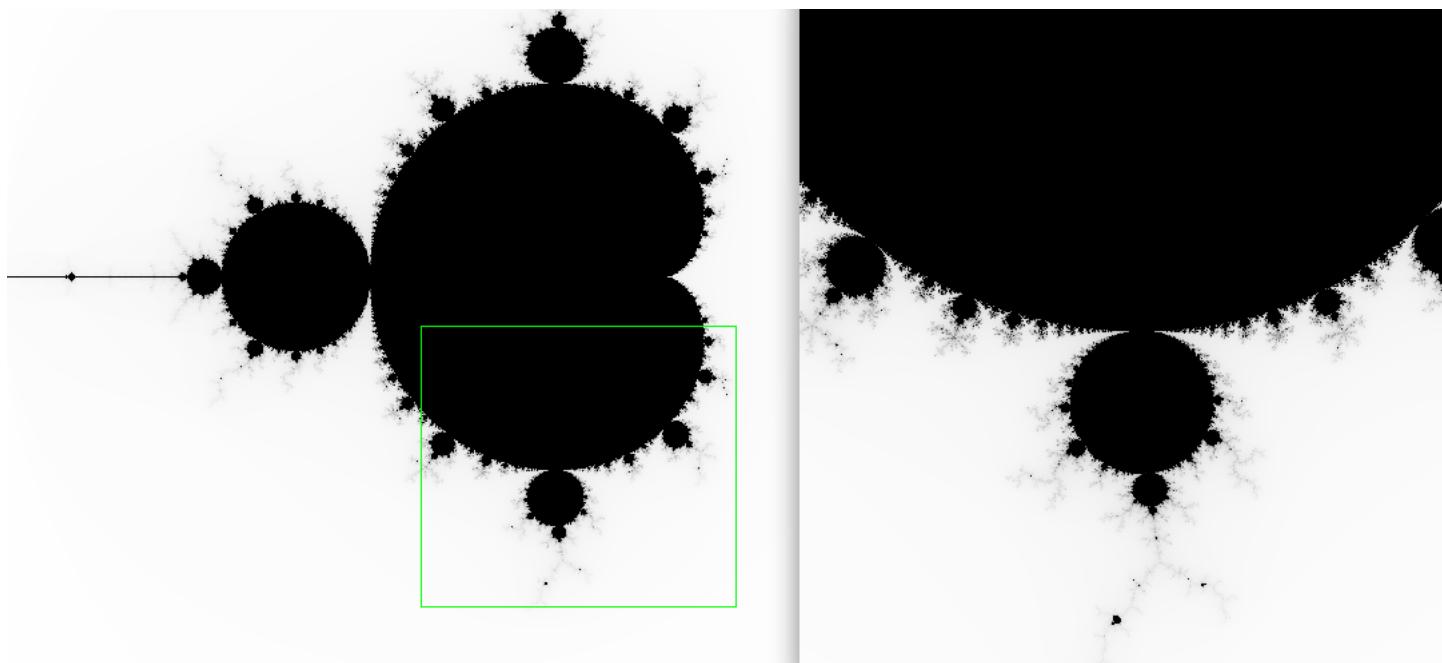
Fordítás után jó ár error fogad minket:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$ javac MandelbrotIterációk.java
MandelbrotIterációk.java:2: error: unmappable character for encoding UTF8
 * MandelbrotIterációk.java
 ^
MandelbrotIterációk.java:2: error: unmappable character for encoding UTF8
 * MandelbrotIterációk.java
 ^
MandelbrotIterációk.java:4: error: unmappable character for encoding UTF8
 * DIGIT 2005, Javat tanítok
 ^
MandelbrotIterációk.java:5: error: unmappable character for encoding UTF8
 * Bétfai Norbert, nbtfai@inf.unideb.hu
 ^
MandelbrotIterációk.java:9: error: unmappable character for encoding UTF8
 * A nagyított Mandelbrot halmazok adott pontjában kópes
 ^
MandelbrotIterációk.java:9: error: unmappable character for encoding UTF8
 * A nagyított Mandelbrot halmazok adott pontjában kópes
 ^
MandelbrotIterációk.java:9: error: unmappable character for encoding UTF8
 * A nagyított Mandelbrot halmazok adott pontjában kópes
```

A hiba üzenetekből azt tudjuk leszűrni , hogy a forráskód kódolásával van a gond. Pontosabban az , hogy nem található a karakter UTF-8 kódolás számára. Így hát a google segítségével , rákerestem olyan karakterkódolásra ami tartalmazza az ékezetes betüket, találtam is jótárat amik sajnos nem működtek. Végül ráakadtam a egy olyanra amivel sikerült kiküszöbölni a problémát.



A nagyítás az egér segítségével kijelölt területen történik :



## 15.2 I334d1c4 5

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

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Chomsky>

Megoldás videó:

A feladatunk az volt , hogy írunk olyan Java vagy C++ programot ami megvalósítja , amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet>.

A "leet speak" kommunikáció során a betűket, számokat az eredeti formájukhoz hasonló karakterekkel helyettesítik. Így megnehezítve az olvasást olyan személyek számára akik nem ismerik ezt a fajta kódolást. Ez a fajta kódolás a '80-as években alakult ki, de a mai napig nagy népszerűségnek örvend az online világban.

A Prog 1-es órákon már találkozhattunk magával a leet-tel, így nem lesz teljesen ismeretlen a dolog számunkra.

A programunk Java-ban készült, lássuk a programkódot :

```
import java.util.*;

class LeetCypher
{
 private static String atalakitando = new String();

 private static String leet = new String();
```

A kódunk legelején importálunk minden szükséges könyvtárat. A LeetCypher osztályunkban létrehozásra kerül két String. Az egyik Stringben az általunk megadott átalakítandó szöveg kerül tárolásra. A másik Stringben pedig a már átalakított szöveget tároljuk.

```
Map<String, String> character = new HashMap<String, String>();

public void print(String atalakitando)
{
 System.out.println(atalakitando);
}
```

Egy Map adatszerkezetben fogjuk az egyes karaktereket és a 133t ábécé-beli megfelelőit tárolni. Majd az eredményünket a print metódus segítségével fogjuk kiiratni.

```
public void atalakit(String szo)
{
 character.put("A", "4");
 character.put("a", "4");
 character.put("B", "8");
 character.put("b", "8");
 character.put("C", "<");
 character.put("c", "<");
 character.put("D", "|");
 character.put("d", "o|");
 character.put("E", "3");
 character.put("e", "3");
```

```
character.put("F", "|=");
character.put("G", "(");
character.put("g", "9");
...
...
...
for (int i = 0; i < szo.length(); i++)
if (character.get(szo.substring(i, i + 1)) != null)
atalakitando += character.get(szo.substring(i, i + 1)) + " ";
};

}
```

Következik az atalakit függvényünk. Feltöljük a charackter névvel ellátott Map-ünket a megfelelő párokkal. Majd következik egy for ciklus, amivel végigmegyünk a megadott szó betűin és minden betűhöz eltároljuk a l33t-beli megfelelőjét szóközzel elválasztva egy változóban.

```
public LeetCypher()
{
 atalakit(atalakitott);
 print(atalakitando); //atalakítva
}
```

Létrehozásra kerül a konstruktörünk. Meghívásra kerül a l33t átalakításához szükséges metódusunk, majd kiiratásra kerül az eredmény.

```
public static void main(String args[])
{
 StringBuilder builder = new StringBuilder(atalakitott.length());
 for (String str : args)
 {
 builder.append(str + ' ');
 }

 atalakitott = builder.toString();

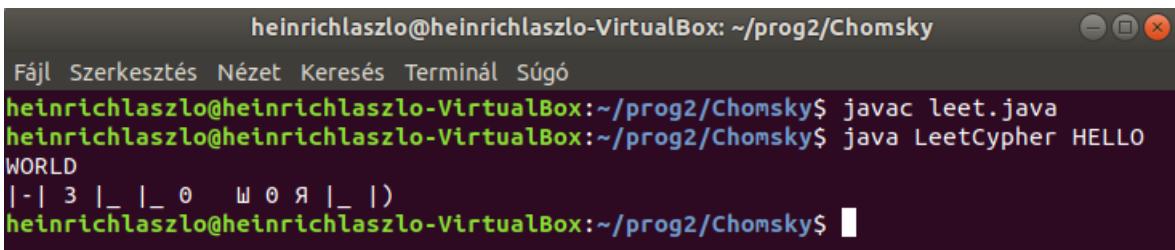
 new LeetCypher();
}

}
```

Nézzük mi történik a main-ben. Létrehozásra kerül a StringBuilder, ez fog felelni azért, hogy a már átalakított Stringünket karakterenként felépitsük. A for ciklus segítségével fogunk végigmenni a szavunkon, , amit parancssori argumentumként adunk meg. Majd eltároljuk ezt, az átalakított szöveget egy String változóban.

Végezetül pedig létrehozásra kerül egy új LeetCypher objektum is.

Fordítjuk és futtatjuk :



A screenshot of a terminal window titled "heinrichlaszlo@heinrichlaszlo-VirtualBox: ~/prog2/Chomsky". The window shows the following text:  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky\$ javac leet.java  
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky\$ java LeetCypher HELLO  
WORLD  
| - | 3 | \_ | \_ 0 | | 0 | | )  
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky\$ █

Láthatjuk, hogy az általunk megadott HELLO WORLD-ot sikeresen átalakította a programunk.

### 15.3 Full screen

Készítsünk egy teljes képernyős Java programot! Tipp: [https://www.tankonyvtar.hu/en/tartalom/tkt/javatitok-javat/ch03.html#labirintus\\_jatek](https://www.tankonyvtar.hu/en/tartalom/tkt/javatitok-javat/ch03.html#labirintus_jatek)

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

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Chomsky>

Tanulságok, tapasztalatok, magyarázat...

Az volt a feladatunk , hogy egy teljes képernyős Java programot. Egy már számunkra bevprogból és prog1ből is ismert feladat fullscreenes verzióját választottam , ami nem más mint a BouncingBall magyarul pattogó labda.

A labda mozgatása esetüben nem lényeges , annyi érdekesség van benne talán , hogy if nélkül történik. ( mint anno bevprogon és prog1en.)

Annak érdekében , hogy full képernyőn fusson a programunk a konstruktorunkban be kell állítanunk , hogy frame undecorated legyen. Majd a teljes képernyő módba , setFullscreenWindows függvény segítségvel jutunk el.

Lássuk a kódot :

```
import java.awt.*;
import java.awt.image.BufferStrategy;
```

A programunk elején importálunk minden szükséges könyvtárat ami lehetővé teszi számunkra a grafikus megjelenítést.

```
public class BouncingBall {
 int x = 450;
 int y = 450;
 int radius = 50;
 int tempX, tempY;
 int maxX, maxY;
 private static DisplayMode[] BEST_DISPLAY_MODES = new DisplayMode[] {
 new DisplayMode(1920, 1080, 32, 0),
 new DisplayMode(1920, 1080, 16, 0),
 new DisplayMode(1920, 1080, 8, 0)
 };
```

A BouncingBall osztályunkban létrehozzuk azokat a változókat amelyek, a mozgást illetve a mozgásnak a határait fogják definiálni. Létrehozunk egy tömböt `DisplayMode` típussal, amiben 3 képernyő típus kerül eltárolásra.

```
Frame mainFrame;
public void move() {
 tempY = (int)(tempY + 1) % (int)(2 * (maxY - radius));
 y = y + (int)Math.pow(-1, Math.floor(tempY / (maxY - radius)));
 tempX = (int)(tempX + 1) % (int)(2 * (maxX - radius));
 x = x + (int)Math.pow(-1, Math.floor(tempX / (maxX - radius)));
}
```

Létrehozunk egy `Frame` objektumot. Majd a `move` metódusban történik meg a labda mozgatása a képernyőn.

```
public BouncingBall (int numBuffers, GraphicsDevice device) {
 try {
 GraphicsConfiguration gc = device.getDefaultConfiguration();
 mainFrame = new Frame(gc);
 mainFrame.setUndecorated(true);
 mainFrame.setIgnoreRepaint(true);
 device.setFullScreenWindow(mainFrame);
 if (device.isDisplayChangeSupported()) {
 chooseBestDisplayMode(device);
 }
 Rectangle bounds = mainFrame.getBounds();
 bounds.setSize(device.getDisplayMode().getWidth(), device.getDisplayMode().getHeight());
 maxX = device.getDisplayMode().getWidth();
 maxY = device.getDisplayMode().getHeight();
 tempX = x;
 tempY = y;
 mainFrame.createBufferStrategy(numBuffers);
 BufferStrategy bufferStrategy = mainFrame.getBufferStrategy();
 while(true) {
 Graphics g = bufferStrategy.getDrawGraphics();
 if (!bufferStrategy.contentsLost()) {
 move();
 g.setColor(Color.white);
 g.fillRect(0, 0, bounds.width, bounds.height);

 g.setColor(Color.blue);
 g.fillOval(x, y, radius, radius);
 bufferStrategy.show();
 g.dispose();
 }
 try {
 Thread.sleep(5);
 } catch (InterruptedException e) {}
 }
 }
```

```
} catch (Exception e) {
e.printStackTrace();
} finally {
device.setFullScreenWindow(null);
}
}
```

Létrehozzuk a BouncingBall objektumot két paraméterrel. Majd következik egy try-catch szerkezet. Létrehozzuk a Frame-ünket gc(grafikai konfigurációs fájl) paraméterrel. A képernyőnket indulás előtt tisztítjuk, hogy a labdánkon kívül más ne legyen rajta. Ezekben kívül beállításra kerülnek még a kép váltásának gyakorisága illetve a színek is.

```
private static DisplayMode getBestDisplayMode(GraphicsDevice device) {
for (int x = 0; x < BEST_DISPLAY_MODES.length; x++) {
DisplayMode[] modes = device.getDisplayModes();
for (int i = 0; i < modes.length; i++) {
if (modes[i].getWidth() == BEST_DISPLAY_MODES[x].getWidth()
&& modes[i].getHeight() == BEST_DISPLAY_MODES[x].getHeight()
&& modes[i].getBitDepth() == BEST_DISPLAY_MODES[x].getBitDepth()
) {
return BEST_DISPLAY_MODES[x];
}
}
}
}
}
return null;
}
```

Következik egy paraméteres függvény ami a getBestDisplayMode névre fog hallgatni. Magában a függvényben végig megyünk a programunk elején létrehozott tömbön, hogy a az eszközünkhez a legmegfelelőbb képmódot adjuk vissza.

```
public static void chooseBestDisplayMode(GraphicsDevice device) {
DisplayMode best = getBestDisplayMode(device);
if (best != null) {
device.setDisplayMode(best);
}
}
```

Beállítjuk az eszközünknek a képmódját a chooseBestDisplayMode függvény segítségével.

```
public static void main(String[] args) {
try {
int numBuffers = 2;
GraphicsEnvironment env = GraphicsEnvironment.
getLocalGraphicsEnvironment();
GraphicsDevice device = env.getDefaultScreenDevice();
BouncingBall ball = new BouncingBall(numBuffers, device);
} catch (Exception e) {
e.printStackTrace();
}
System.exit(0);
```

```
}
```

A `Main`ben találkozunk egy try-catch szerkezzel , itt történik a hibakezelés. A try részben történik meg maga a példányosítás.

Fordítjuk majd futtatjuk a kódot :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$ javac BouncingBall.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$ java BouncingBall
```

Végezetül , nézzük a teljes képernyős formáját.



## 15.4 Perceptron osztály

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

Megoldás videó:

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

<https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Chomsky>

Tanulságok, tapasztalatok, magyarázat...

A perceptron leegyszerűsítve nem más mint ezen neuron mesterséges intelligenciában használt változata. Tanulásra képes, a bemenő 0-k és 1-esek sorozatából mintákat tanul meg és súlyozott összegzést végez. A következő feladat során egy ilyen perceptronról fogunk elkészíteni aminek alapvetően a feladata az, hogy a mandelbrot.cpp programunk által létrehozott Mandelbrot-halmazt ábrázoló PNG kép egyik színkódját vegye és az a színkód legyen a többrétegű neurális háló inputja.

Lássuk a programot :

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

Includoljuk az iostream, az mlp és a png++/png könyvtárakat. A mlp könyvtárra azért van szükségünk mert több rétegű perceptronról fogunk létrehozni. A png++/png könyvtár a PNG kiterjesztésű képállományokkal való munkát teszi számunkra lehetővé.

```
using namespace std;

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);
```

A main függvényünk első sorában megmondjuk, hogy a képállományunk beolvasása az 1-es parancssori argumentum alapján történik. Eltároljuk egy segédváltozóban a kép méretét a get\_width és a get\_height szorozatát, majd létrehozzuk a perceptronunkat a new operátor segítségével.

```
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;

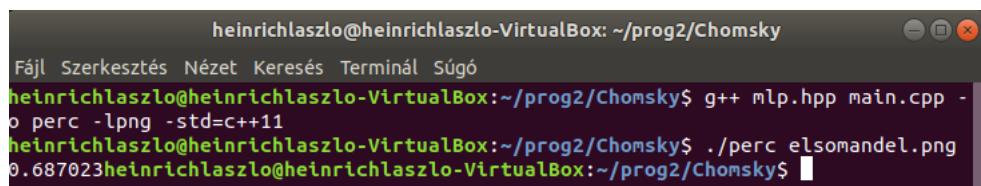
double value = (*p)(image);

cout<<value;

delete p;
delete [] image;
```

Az egyik for ciklussal végig megyünk a kép szélességét alkotó pontokon , majd a másik for ciklussal végig megyünk a magasságát alkotó pontokon. Az `image` -ben fogjuk tárolni a vörös képpontokat. A `value` értéke adja meg a Percetron `image`-ra történő meghívását, ami egy double típusú változót tárol. Kiiratjuk és töröljük azokat az elemeket amiket már nem használunk, így az addig lefoglalt memória egységeket újra használhatóvá tettük.

Fordítjuk és futtattuk a képen látható módon:



```
heinrichlaszlo@heinrichlaszlo-VirtualBox: ~/prog2/Chomsky
Fájl Szerkesztés Nézet Keresés Terminál Súgó
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$ g++ mlp.hpp main.cpp -o perc -lpng -std=c++11
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$./perc elsomandel.png
0.687023heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$
```

# Chapter 16

## Helló, Stroustrup!

### 16.1 Változó argumentumszámú ctor és Összefoglaló

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 het/Perceptron osztály feladatot is.)

Megoldás videó:

Megoldás forrása : <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Stroustrup>

Kezdetben nézzük meg, hogy miről is van szó, példát az mlp.hpp header fájljában találhatunk, nézzük!

```
class Perceptron
{
public:
 Perceptron (int nof, ...)
{
```

A fenti példa egy változó argumentum számú konstruktur, azaz nincsen egyértelműen definiálva, hogy hány paraméterrel kell rendelkeznie. Első ránézésre annyit tudunk megállapítani, hogy egy argumentumot biztos, hogy tartalmaznia kell. Az az egy argumentum amit tartalmazni fog egy Integer típussal rendelkező nof lesz. A változó argumentum számot pedig a "..." jelölésből tudjuk megállapítani.

Mi is az a perceptron?

A perceptron nem más, mint a mesterséges intelligenciában használatos változata ezen neuronoknak. Tanulásra képes, súlyozott összegzést végez a bemenő 0-ások és 1-esek sorozatának mintáiból. Ennek több változata is létezik, mi a feladatunk során a többrétegűeket fogjuk megvizsgálni.

Ezt MLP-nek, azaz MultiLayer Perceptron-nak nevezzük és ez az eggyik leggyakrabban használt hálózatiarchitektúra. Ez a fajta neurális hálózat 3 rétegből épülnek fel, melyek a következők : bemeneti réteg, rejtett réteg, kimeneti réteg.

Bemeneti réteg : A hálózat felé történő bemeneti jel továbbítását végző neuronokat találhatjuk itt.

Rejtett réteg : Ide találhatjuk a tulajdonképpeni feldolgozást végző neuronokat. Egy hálózaton több rejtett réteg is lehet.

Kimeneti réteg : Az itt található neuronok a külvilág felé továbbítják az információt.

A többrétegű perceptron minden egyes rétege neuronokból áll, ezért fontos azt biztosítanunk, hogy a neurális hálózatunk kimenete a súlyok folytonos, diffenenciálható függvénye legyen.

Hogy történik a neurális hálónk tanítása?

A tanítási folyamatot, ellenőrzött tanításnak is nevezzük, mert a hálózat kimenetén értelmezett hiba felhasználásával határoztathatjuk meg a kritériumfüggvényt vagy kockázat paraméterfüggését. Ez a fajta tanítás a leggyakrabban a hiba-visszaterjesztéses módszerrel valósul meg.

A tanítás főbb lépései : megadjuk a kezdeti súlyokat, a bemeneti jelet végigáramoltatjuk a hálózaton, mindenzt a súlyok megváltoztatása nélkül. Az így kapott kimeneti jelet összevetjük a tényleges kimeneti jellel. A hibát visszaáramoltatjuk a hálózaton, majd a súlyokat megváltoztatjuk a hiba csökkenése érdekében.

A feladatunk megoldása során egy ilyen percetront fogunk elkészíteni, aminek a feladata az lesz, hogy a mandelbrot.cpp programunk által generált elsomandel.png képet felhasználva generáltassunk vele egy megegyező méretű képet.

A feladat nagyban megegyezik az előző Percetronos feladatainkkal, annyi kivétellel, hogy az edigi példáink során a Percetronunk egy képállományra egy értéket adott vissza.

Nézzük a kódot :

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

A programunk elején includoljuk a számunkra szükséges könyvtárakat. Az mlp(multi layer perceptron) és png++/png-re a többrétegű perceptron létrehozásánál lesz szükségünk. A PNG könyvtárunk a PNG képállományokkal való munkát teszi nekünk lehetővé.

```
using namespace std;
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);
```

Elérkezünk a main függvényhez , először is a képállomány kerül beolvasásra, az 1-es parancssori argumentum alapján.

A get\_width és a get\_height szorzatából megkapjuk a kép méretét és eltároljuk egy változóban.

Példányosítunk egy perceptront a new operátor segítségével. A perceptronunk paraméterei a következők lesznek : a rétegek száma, 1. réteg neuronai az input rétegen, 2. réteg neuronai az input rétegen, az eredmény ami egy képállomány a megszokott szám helyett.

```
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;
```

Deklarálunk egy mutatót double típussal.

Majd jönnek a forciklusok. Ezek a for ciklusok egike a képet alkotó pontok szélességén , míg a másik a képet alkotó pontok magasságán. A végig menések után , az image-ben tárolni fogjuk a képállomány vörös színkomponenseit. Vagyis a lefoglalt tárba belemásoljuk a képállományunk vörös komponensét.

```
double* newPicture = (*p) (image); // eddig: double value = (*p) (←
image);
for (int i = 0; i<png_image.get_width(); ++i)
for (int j = 0; j<png_image.get_height(); ++j)
png_image[i][j].red = newPicture[i*png_image.get_width() + j];
png_image.write("kimeneti.png");
delete p;
delete [] image;
```

Egy double csillag típus segitségével nem egy értéket akarunk kiiratni, hanem magát a képet.

Majd két for ciklus segitségével végigmegyünk , az eredeti kép szélességén és magasságán. Ezeket az színadatokat kapja meg az új képünk. Ezt követően a write függvényel létrehozzuk kimeneti néven az új képállományunkat. A kimeneti állományunk png formátumú lesz.

Majd töröljül a már nem használt elemeket, azaz felszabadítjuk az eddig fenttartott memóriterületeket. Az eddig lefoglalt memóriaegység már újra használható.

```
double* operator() (double image [])
{
```

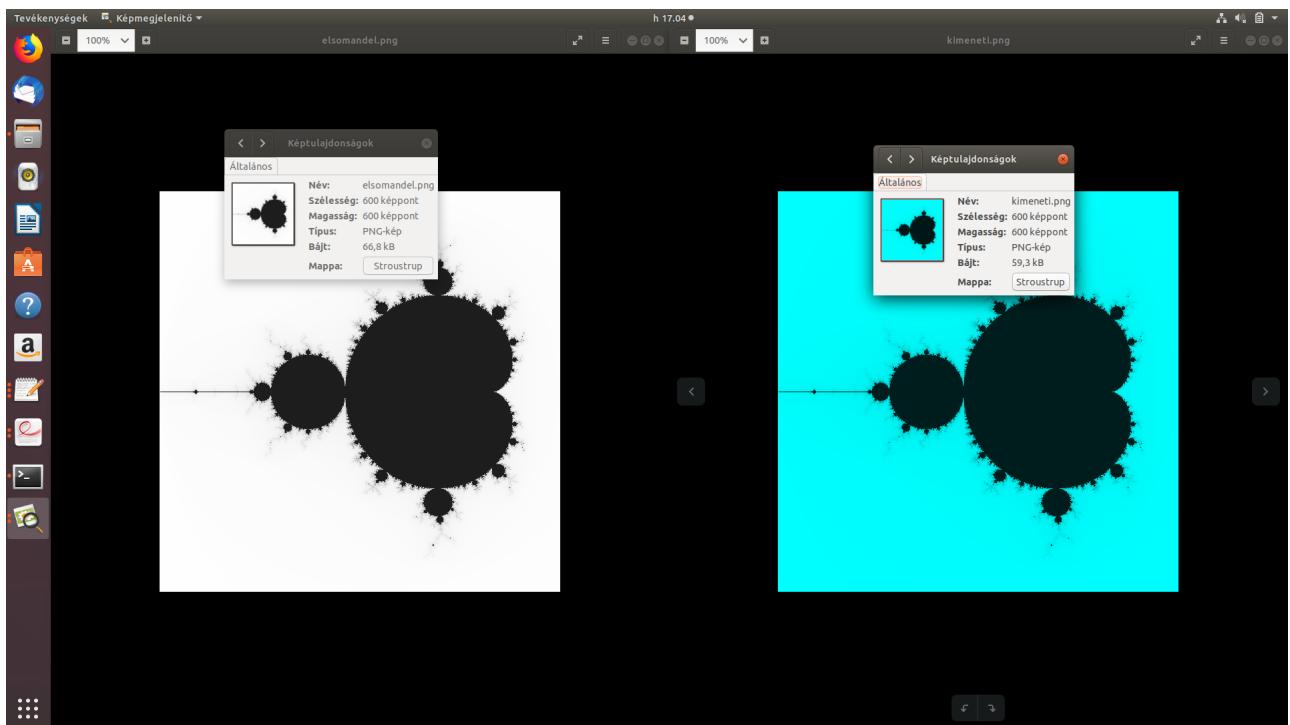
Kicsit bele kell nyúlni az mlp.hpp fájlunkba is.

Például az () operátor működésébe ami mostantól nem egy double értéket , hanem double pointert ad vissza.

Fordítjuk majd futtatjuk a programunkat .

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ g++ mlp.hpp perceptron.cpp -o perceptron -lpng -std=c++11
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$./perceptron elsomenel.png
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$
```

Lássuk az eredményt.



## 16.2 JDK osztályok

Írunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Megoldás videó:

Megoldás forrása : <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Stroustrup>

A feladatot a leírás alapján, a fénykard program alapján csináltam. Az alábbi linken tekinthetjük meg Bátfai tanárúr fénykard.cpp kódját : <https://github.com/nbatfai/future/blob/master/cs/F7/fenykard.cpp>

```
#include <iostream>
#include <string>
#include <map>
#include <iomanip>
#include <fstream>
#include <vector>

#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>
#include <boost/program_options.hpp>
#include <boost/tokenizer.hpp>
#include <boost/date_time posix_time.hpp>

using namespace std;
using namespace boost::filesystem;

int szamlalo = 0;
```

Includoljuk a számunkra szükséges osztályokat és deklarálunk egy változót amiben az osztályol számát fogjuk tárolni.

```
void read_file (boost::filesystem::path path, std::vector<std::string> &acts)
{
 if (is_regular_file (path)) {
 std::string ext (".java");
 if (!ext.compare (boost::filesystem::extension (path)))
 cout<<path.string()<<'\n';
 std::string actpropspath = path.string();
 std::size_t end = actpropspath.find_last_of ("/");
 std::string act = actpropspath.substr (0, end);
 acts.push_back(act);
 szamlalo++;
 }
 else if (is_directory (path))
 for (boost::filesystem::directory_entry & entry : boost::filesystem::directory_iterator (path))
 read_file (entry.path(), acts);
}
}
```

Egy feltételvizsgálat aminek két része van. Az `if`-fel történő feltételvizsgálatban megvizsgáljuk az `is_regular` függvényt, hogy a fájlhierarchyban hol állunk jelenleg. Vagyis ha regular fájlok nem könyvtárak állnak a rendelkezésünkre akkor megvizsgáljuk, hogy mely fájlok rendelkeznek `.java` kiterjesztéssel. Ezeket a fájlokat egy vektorba gyűjtjük, és megnöveljük a számlálónkat.

Ha "ahol éppen állunk" nem fájlokkal találkozunk, akkor meghívásra kerül az `is_directory` függvény. Vagyis amikor még a könyvtárknál járunk, akkor elindítunk egy `for` ciklust és rekurzívan meghívjuk a `read_classes` névre hallgató függvényünket. Így jutunk el a fájlok "szintjére".

```
int main (int argc, char *argv[])
{
 string path="src";
 vector<string> acts;
 read_file(path,acts);
 cout<<"szamlalo: "<<szamlalo<< std::endl;
}
```

A `main`-ben létrehozzuk azt a vektorunkat amiben az osztályokat tároljuk. Majd meghívásjuk a `read_classes` függvényt. Majd kiíratjuk, hogy hány függvény volt az `src` mappában.

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ g++ fenykardJKosztaly.cpp -o fenykard -lboost_system -lboost_filesystem -lboost_program_options -std=c++14
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$./fenykard
```

Majd nézzük az eredményt :

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
src/jdk.jdi/com/sun/jdi/ReferenceType.java
src/jdk.jdi/com/sun/jdi/ClassObjectReference.java
src/jdk.jdi/com/sun/jdi/BooleanValue.java
src/jdk.jdi/com/sun/jdi/Accessible.java
src/jdk.jdi/com/sun/jdi/InternalException.java
src/jdk.jdi/com/sun/jdi/LongValue.java
src/jdk.jdi/com/sun/jdi/InvalidLineNumberFormatException.java
src/jdk.jdi/com/sun/jdi/request/StepRequest.java
src/jdk.jdi/com/sun/jdi/request/ExceptionRequest.java
src/jdk.jdi/com/sun/jdi/request/MethodEntryRequest.java
src/jdk.jdi/com/sun/jdi/request/BreakpointRequest.java
src/jdk.jdi/com/sun/jdi/request/package-info.java
src/jdk.jdi/com/sun/jdi/request/VMDeathRequest.java
src/jdk.jdi/com/sun/jdi/request/MonitorContendedEnteredRequest.java
src/jdk.jdi/com/sun/jdi/request/EventRequest.java
src/jdk.jdi/com/sun/jdi/request/DuplicateRequestException.java
src/jdk.jdi/com/sun/jdi/request/MonitorContendedEnterRequest.java
src/jdk.jdi/com/sun/jdi/request/MonitorWaitRequest.java
src/jdk.jdi/com/sun/jdi/request/ThreadStartRequest.java
src/jdk.jdi/com/sun/jdi/request/AccessWatchpointRequest.java
src/jdk.jdi/com/sun/jdi/request/MethodExitRequest.java
src/jdk.jdi/com/sun/jdi/request/ThreadDeathRequest.java
src/jdk.jdi/com/sun/jdi/request/MonitorWaitedRequest.java
src/jdk.jdi/com/sun/jdi/request/EventRequestManager.java
src/jdk.jdi/com/sun/jdi/request/ClassUnloadRequest.java
src/jdk.jdi/com/sun/jdi/request/WatchpointRequest.java
src/jdk.jdi/com/sun/jdi/request/ModificationWatchpointRequest.java
src/jdk.jdi/com/sun/jdi/request/InvalidRequestStateException.java
src/jdk.jdi/com/sun/jdi/request/ClassPrepareRequest.java
src/jdk.jdi/com/sun/jdi/NativeMethodException.java
src/jdk.jdi/com/sun/jdi/InvalidCodeIndexException.java
src/jdk.jdi/com/sun/jdi/VirtualMachine.java
src/jdk.jdi/com/sun/jdi/CharValue.java
src/jdk.jdi/com/sun/jdi/VMMismatchException.java
src/jdk.jdi/com/sun/jdi/IncompatibleThreadStateException.java
src/jdk.jdi/com/sun/jdi/FloatValue.java
src/jdk.jdi/com/sun/jdi>Type.java
src/java.management.rmi/module-info.java
src/java.management.rmi/com/sun/jmx/remote/protocol/rmi/ClientProvider.java
src/java.management.rmi/com/sun/jmx/remote/protocol/rmi/ServerProvider.java
src/java.management.rmi/com/sun/jmx/remote/internal/rmi/ProxyRef.java
src/java.management.rmi/com/sun/jmx/remote/internal/rmi/RMIEporter.java
src/java.management.rmi/javax/management/remote/rmi/RMIConnector.java
src/java.management.rmi/javax/management/remote/rmi/RMIServer.java
src/java.management.rmi/javax/management/remote/rmi/NoCallStackClassLoader.java
src/java.management.rmi/javax/management/remote/rmi/RMIServerImpl_Stub.java
src/java.management.rmi/javax/management/remote/rmi/RMIServerImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIConnectorServer.java
src/java.management.rmi/javax/management/remote/rmi/RMIRMPServerImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIClientImpl_Stub.java
src/java.management.rmi/javax/management/remote/rmi/RMIIOPServerImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIClientImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIClient.java
szamlalo: 17593
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$
```

## 16.3 Hibásan implementált RSA törése

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: [https://arato.inf.unideb.hu/~batfai.norbert/UDPROG/deprecated/Prog2\\_3.pdf](https://arato.inf.unideb.hu/~batfai.norbert/UDPROG/deprecated/Prog2_3.pdf) (71-73 fólia) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása : <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Stroustrup>

Tanulságok, tapasztalatok, magyarázat...

Tutor: [Tutor](#)

Mi is az az RSA?

Napjaink egyik legszélesebb körben használt titkosító eljárás. A nevét az alkotóiról kapta : R, mint Ron Rivest, S, mint Adi Shamir és A, mint Len Adleman. Magát az eljárást 1978-ban publikálták. Működése matematikai alapokon, a Fermat -tételel alapszik. Érdekes jellemzője, hogy két kulccsal dolgozik. Ez a két kulcs egy-egy számpárt takar. Ez azért fontos, mert így teljesen ketté lesz választva a titkosítás és a törés folyamata. Mivel a kódolás és a dekódolás paraméterei nem lesznek ugyanazok, így az egyik nem lesz meghatározható a másikból.

Forrás : <https://hu.wikipedia.org/wiki/RSA-elj%C3%A1r%C3%A1s>, [https://www.tankonyvtar.hu/hu/tartalom/tamop425/0046\\_infoch08s07.html](https://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_infoch08s07.html)

Tegyük még említést a BigInteger osztályról. A BigInteger osztály a JDK részét képezi, jelenleg a java.math csomag része. Gyakran használjuk kriptográfiai kódolók-törők elkészítéséhez. A típus egy egész értékek-ből és 32-bites egészből úgynevezett skálázó faktor alkotja.

Nézzük a kódunkat!

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map.Entry;
```

Legelőször importáljuk a számunkra szükséges könyvtárakat. Köztük a már említett BigInteger osztályt is.

```
public class rsa_cipher {
 public static void main(String[] args) {
 int bitlength = 2100;

 SecureRandom random = new SecureRandom();

 BigInteger p = BigInteger.probablePrime(bitlength/2, random);
 BigInteger q = BigInteger.probablePrime(bitlength/2, random);

 BigInteger publicKey = new BigInteger("65537");
 BigInteger modulus = p.multiply(q);

 String str = "this is a perfect string".toUpperCase();
 System.out.println("Eredeti: " + str);
```

A rsa\_cipher osztályban történik a bithossz beállítása és kerülnek létrehozásra a BigInteger számunk. Ilyen érték lesz például a kulcsunk és a modulus is. Megadjuk az eredeti, átalakításra váró szövegünket és eltároljuk egy String típusú változóban ami str névre hallgat.

```
byte[] out = new byte[str.length()];
for (int i = 0; i < str.length(); i++) {
 char c = str.charAt(i);
 if (c == ' ')
 out[i] = (byte)c;
 else
 out[i] = new BigInteger(new byte[] {(byte)c}).modPow(publicKey, ←
 modulus).byteValue();
}
String encoded = new String(out);
System.out.println("Kodolt: " + encoded);

Decode de = new Decode(encoded);
System.out.println("Visszafejtett: " + de.getDecoded());
}
}
```

A fenti kódrészlet felelős a titkosításért. A stringünk minden egyes karakterén végighaladva , egyenként teszi ezt meg. Egy byte elemeket tartalmazó tömbben kerülnek tárolásra a titkosított karakterek, melyekből egy stringet készítünk, encoded névvel, majd kiiratjuk ezt. Majd létrehozunk egy decode objektumot, ami már a gyakoriság alapok visszafejtett stringünket tartalmazza.

Nézzük hogyan működik a visszafejtés!

```
private void loadFreqList() {
 BufferedReader reader;
 try {
 reader = new BufferedReader(new FileReader("gyakorisag.txt"));
 String line;
 while((line = reader.readLine()) != null) {
 String[] args = line.split("\t");
 char c = args[0].charAt(0);
 int num = Integer.parseInt(args[1]);
 this.charRank.put(c, num);
 }
 } catch (Exception e) {
 System.out.println("Error when loading list -> " + e.getMessage());
 }
}
```

A függvényen belül található try-catch szerkezetben beolvásásra kerül egy gyakoriság lista. Karakterenként végigmegy a lista elemein és ha az aktuális betű már megtalálható a listában, a gyakoriság növelésre kerül. Ha új betűvel találkozik, gyakorisága 1-re lesz beállítva. Mindez a try-on belül történik meg. Catch pedig egy hibaüzenet fog dobni ha nem sikerült a listát betöltenünk.

```
private char nextFreq() {
 char c = 0;
 int nowFreq = 0;
```

```
for(Entry<Character, Integer> e : this.charRank.entrySet()) {
 if (e.getValue() > nowFreq) {
 nowFreq = e.getValue();
 c = e.getKey();
 }
}
if (this.charRank.containsKey(c))
 this.charRank.remove(c);
return c;
}
```

A nextFreq függvényben történik a rendezéssel (maximum-kiválasztással) a gyakorisági listában lévő betűk behelyettesítése. A gyakorisági listánk az, ami befolyásolja majd a kapott eredményünket, hiszen az algoritmus a nagyobb gyakorisági értékkel rendelkező karaktereket helyezi előtérbe.

```
public Decode(String str) {
 this.charRank = new HashMap<Character, Integer>();
 this.decoded = str;

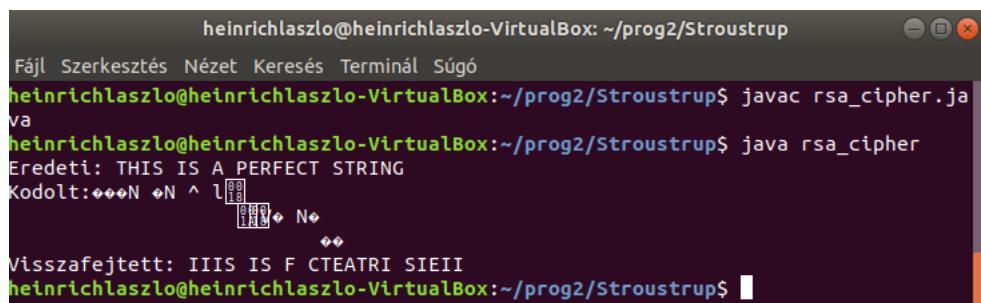
 this.loadFreqList();

 HashMap<Character, Integer> frequency = new HashMap<Character, Integer <-> ()>;
 for (int i = 0; i < str.length(); i++) {
 char c = str.charAt(i);
 if (c != ' ')
 if(frequency.containsKey(c))
 frequency.put(c, frequency.get(c) + 1);
 else
 frequency.put(c, 1);
 }

 while (frequency.size() > 0) {
 int mi = 0;
 char c = 0;
 for (Entry<Character, Integer> e : frequency.entrySet()) {
 if (mi < e.getValue()) {
 mi = e.getValue();
 c = e.getKey();
 }
 }
 this.decoded = this.decoded.replace(c, this.nextFreq());
 frequency.remove(c);
 }
}
```

A Decode függvény fogja az előzőleg tárgyalt függvények és gyakorisági lista felhasználásával elvégezi a dekódolást. A while-ból láthatjuk, hogy ezt addig teszi, ameddig el nem fogynak a karaktereink.

Nézzük az eredményt fordítás és futtatás után!



A screenshot of a terminal window titled "heinrichlaszlo@heinrichlaszlo-VirtualBox: ~/prog2/Stroustrup". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ javac rsa_cipher.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ java rsa_cipher
Eredeti: THIS IS A PERFECT STRING
Kodolt:♦♦N ♦N ^ l♦
IIIS♦ N♦
♦♦
Visszafejtett: IIIS IS F CTEATRI SIEII
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$
```

# Chapter 17

## Helló, Gödel!

### 17.1 Alternatív Tabella rendezése

Mutassuk be a [https://progpater.blog.hu/2011/03/11/alternativ\\_tabella](https://progpater.blog.hu/2011/03/11/alternativ_tabella) a programban a java.lang Interface comparable<T> szerepét!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/G%C3%B6del>

A feladatunk az volt, hogy az alternatív tabella programban mutassuk be, az java.lang Interface comparable<T> szerepét.

De mielőtt , ezt megtennénk , ismerjük meg mi is az az alternatív tabella?

Az alternatív tabella nem más, mint az egyes futbalbajnokságokhoz készített sorrend, ami nem a megszokott pontozás szerint számolja a csapatok sorrendjét, hanem azt veszi figyelembe, hogy az adott csapat , melyik másik csapattal szemben érte el az adott eredményt.

Az ilyen alternatív tabellák elkészítésének egyik módja, hogy a Google PageRank algoritmuást használjuk fel ehhez.

Leegyszerűsítve, az alternatív sorrend az alapján alakul ki, hogy az a csapat kerül előrébb, mely előrébb lévő csapat ellen szerez pontot.

A bevezető forrása : [https://hu.wikipedia.org/wiki/Alternat%C3%ADV\\_tabella](https://hu.wikipedia.org/wiki/Alternat%C3%ADV_tabella)

Most, hogy már tudjuk mi is az az alternatív tabella nézzük meg mi is az az interface amire a feladatunk rákérdezett:

```
class Csapat implements Comparable<Csapat>
{
 protected String nev;
 protected double ertek;

 public Csapat(String nev, double ertek) {
 this.nev = nev;
 this.ertek = ertek;
 }
}
```

```
public int compareTo(Csapat csapat) {
 if (this.ertek < csapat.ertek) {
 return -1;
 } else if (this.ertek > csapat.ertek) {
 return 1;
 } else {
 return 0;
 }
}
```

Ha egy összetettebb adatszerkeztnél a sort függvényel való rendezés hibákhoz vezetne, ezért a comparable<T> interface segítségével tudjuk ezeket kiküszöbölni.

A comparable<T> interface , más néven természetes rendezés, egy teljes rendezést tesz az osztály objektumaira.

A java.lang csomag része, de ezt a csomagot nem szükséges importálni, mert ez az importálás automatikusan történik. Ez az interface egyetlen egy metódust tartalmaz, ami az compareTo(), amit más néven természetes összehasonlító metódusnak is neveznek.

A compareTo() összehasonlíja az objektumokat és három értékkel térhet vissza. Negatív értékkel, (a kódunkban ez -1) ha az objektum nagyobb mint a kiválasztott objektum. Nullával tér vissza ha a két objektum egyenlő. Pozitív értékkel ( esetünkben 1) tér vissza, ha az objektum értéke kisebb mint a kiválasztott objektumé. Megadjuk , hogy a csapat melyik tagja alapján rendezzen, esetünkben a doubleként megadott érték alapján történik a rendezés. Majd a kiválasztott objektum nevéhez tartozó értéket hasonlítja össze a többi névhez tartozó értékkel. A csapat osztályban létrehozott értékeket hasonlítjuk össze, a többi névhez tartozó értékkel. Ezután a csapat classban létrehozott elemeket rendezett sorrendben adja vissza a sort függvény.

Az AlternativTabella.jaa programunkban a rendezCsapatok függvény hívja, majd meg a class-t de ez már 0 ; 1 és -1-ekből áll és rendezve lesz.

A compareTo() függvényben az objektumok sorrendjét az objektumok összehasonlításának sorrendje határozza meg.

A comparable<T> interface implementálása a csapat nevű osztályban történik a következő módon :

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList(←
 csapatok);
java.util.Collections.sort(rendezettCsapatok);
```

Létrejön a Csapat típusú listánk rendezettCsapatok néven. A következő sorban rendezzük a sort metódussal. De a sort metódus nem minden listán működik, csak amelynek tagjai implementálják a Comparable interface-t.

Ezt a jdk-ban találhatjuk :

```
"Lists (and arrays) of objects that implement this interface can be sorted
automatically by {@link Collections#sort(List)} Collections.sort} (and
{@link Arrays#sort(Object[])} Arrays.sort}). Objects that implement this
interface can be used as keys in a {@link plain SortedMap} sorted map} or as
elements in a {@link plain SortedSet} sorted set}, without the need to
```

```
specify a {@linkplain Comparator comparator}."
```

Fordítsuk és futtasuk :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/gödel$ javac AlternativTabella.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/gödel$ java AlternativTabella
iteracio...
norma = 0.05918759643574294
qsszeg = 1.0
iteracio...
norma = 0.014871507967965858
qsszeg = 0.9999999999999999
iteracio...
norma = 0.006686056768932613
qsszeg = 1.0
iteracio...
norma = 0.007776749198562133
qsszeg = 1.0
iteracio...
norma = 0.007661483555477314
qsszeg = 0.9999999999999999
iteracio...
norma = 0.007501657116770109
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007532790726590474
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007538144256251714
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535025315190922
qsszeg = 1.0
iteracio...
norma = 0.00753510193655861
qsszeg = 0.9999999999999997
iteracio...
norma = 0.0075353297234758716
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535284725802345
qsszeg = 0.9999999999999999
iteracio...
norma = 0.007535275242694286
qsszeg = 0.9999999999999996
iteracio...
norma = 0.0075352801728795745
qsszeg = 0.9999999999999999
iteracio...
norma = 0.007535280078749908
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535279718816022
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535279786665789
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535279802432719
```

```
Csapatok rendezve:
```

```
|- Videoton
 40
 Videoton
 0.0841
|- Ferencvaros
 34
 Debreceni VSC
 0.0828
|- Paks
 31
 Paksi FC
 0.0725
|- Debreceni VSC
 31
 BFC Siófok
 0.0698
|- Zalaegerszegi TE
 30
 Budapest Honvéd
 0.0697
|- Kaposvári Rakoczi
 29
 Ferencvaros
 0.0689
|- Lombard Papa
 27
 Gyuri ETO
 0.0649
|- Kecskeméti TE
 24
 Íjpest
 0.0636
|- Íjpest
 23
 Zalaegerszegi TE
 0.0636
|- Gyuri ETO
 23
 MTK Budapest
 0.0595
|-
```

```
Budapest Honvéd
22
Kaposvári Rakoczi
0.0570
|- MTK Budapest
22
Lombard Papa
0.0560
|- Vasas
21
Szombathelyi Haladas
0.0559
|- Szombathelyi Haladas
20
Vasas
0.0543
|- BFC Siófok
18
Kecskeméti TE
0.0439
|- Szolnoki MAV
9
Szolnoki MAV FC
0.0329
|-
```

## 17.2 STL map érték szerinti rendezése

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/G%C3%B6del>

Mi is az az STL (Standart Template Library)?

Az STL egy olyan C++ sablon- vagy mintaosztályok összessége, amelyek lehetővé teszik számos, széles körben használt algoritmus és adatszerkezet használatát.

Három fő dolgot találhatunk meg benne : tárolókat, iterátorokat és algoritmusokat.

A feladatunk szempontjából a tárolók lesznek érdekesek, név szerint a map fajtájuak.

Mit érdemes tudni a map fajtájú tárolókról?

Az elemeket a kulcsuk alapján tudjuk azonosítani, ún. Asszociatív tárolók.

Nézzük a megoldást.:

```
std::vector<std::pair<std::string, int>> sort_map (std::map <std::string, int> &rank)
{
 std::vector<std::pair<std::string, int>> ordered;

 for (auto & i : rank) {
 if (i.second) {
 std::pair<std::string, int> p { i.first, i.second };
 ordered.push_back (p);
 }
 }
}
```

Az érték szerinti rendezést a sort\_map fogja megvalósítani. A visszatérési értéke a függvényünknek vektorpárok lesznek. A függvénytörzsön belül történik a vektor létrehozása ordered néven, és ez lesz egyben a visszatérési értékünk is. Következik egy for ami végig megy a rank map-en azt vizsgálva, hogy vannak-e érték párosok a vektorban. Ha vannak akkor ezek az érték párosok a pair-be kerülnek. A paires pedig a ordered vektorba fognak kerülni.

```
std::sort (
 std::begin (ordered), std::end (ordered),
 [=] (auto && p1, auto && p2) {
 return p1.second > p2.second;
 }
);

return ordered;
```

Rendezni fogjuk a már értékpárokkal feltöltött vektort az std::sort függvénnel, ami egy harmadik paraméterként megadott lambda kifejezés segítségével rendez úgy, hogy végigmegy a vektorunkon és igaz-zal tér vissza, ha az első vizsgált paraméter nagyobb, mint a második.

A függvényünk visszatérési értéke a már rendezett vektor lesz, azaz az ordered

```
int main()
{
```

```
std::map<std::string, int> map;
map["a"] = 11;
map["b"] = 7;
map["c"] = 25;

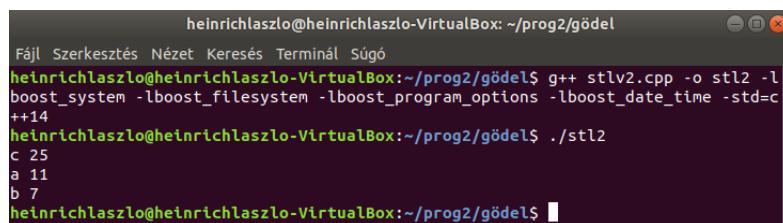
std::vector<std::pair<std::string, int>> sorted = sort_map(map);

for(auto & i : sorted)
{
 std::cout << i.first << " " << i.second << std::endl;
}

}
```

A mainben feltöltünk egy mapet tetszőleges elemekkel. Majd a sort\_map függvénytel rendezzük. Majd a rendezés után egy új sorted vektorban tároljuk el. Következik egy for ciklus amivel végigmegyünk az új vektorunkon és kiiratjuk a már rendezett állapotát.

Nézzük a kódunkat fordítás és futtatás után :



```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/gödel
Fájl Szerkesztés Nézet Keresés Terminál Súgó
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/gödel$ g++ stlv2.cpp -o stl2 -lboost_system -lboost_filesystem -lboost_program_options -lboost_date_time -std=c++14
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/gödel$./stl2
c 25
a 11
b 7
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/gödel$
```

## 17.3 GIMP Scheme hack

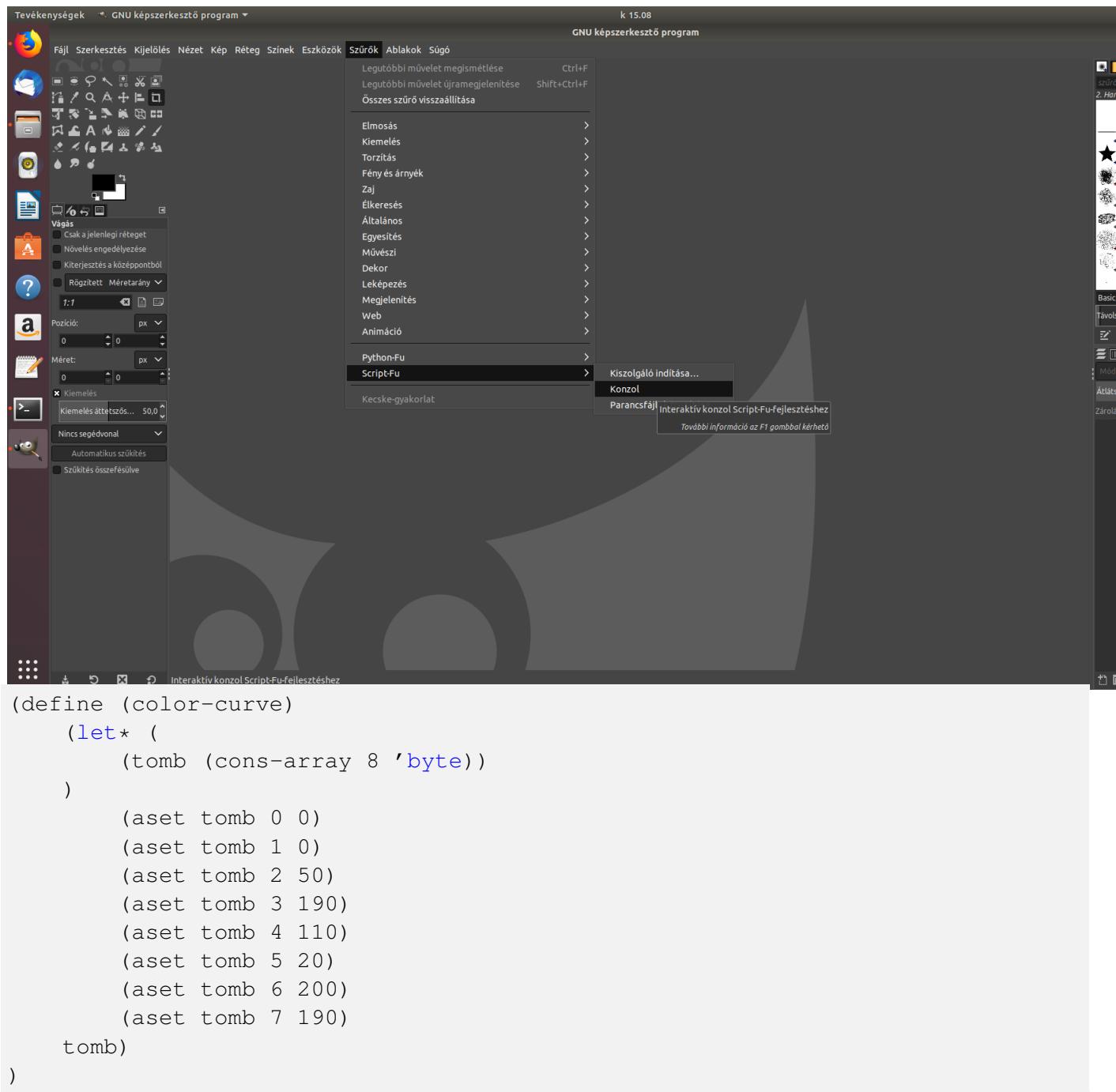
Ha az előző félévben nem dolgoztad fel a témat (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat...

Az előző feladatban megismerkedtünk a Lisp nyelvvel , most a feladat során a Lisp nyelvcsalád egyik képviselőjével fogunk megismerkedni. A Sheme programnyelv az 1970-es évek közepén jelent meg és mai napig találkozhatunk Scheme kódokkal. Forrás és Scheme nyelvről bővebben : <https://hu.wikipedia.org/wiki/Scheme> A fájlunk .scm kiterjesztésű lesz , és ha berakjuk a GIMP erre hivatott mappájába, megtalálhatjuk mint használható szkriptet. A feladat során a Srcipt-fut fogjuk használni . Az előző feladat során megismert GIMP képszerkesztő programhoz egy olyan szkriptet fogunk írni . ami a bemerként megadott szöveg króm effektezését teszi lehetővé. A megíráshoz használhatjuk a Script-Fu-konzolt is :



Megadjuk a függvényeket , olyan módon amit már az előző feladatban láthattunk , ezért erre most nem térnék ki külön. Létrehozunk egy 8 elemű tömböt a color-curve segítségével. Majd megadjuk a megfelelő értékeket.

Megadunk egy olyan függvényt is, ami egy lista x-edik elemét adja vissza, a car(lista első eleme) és cdr(a lsita első elemén kívüli összes elem) használatával.

```
(define (elem x lista)

 (if (= x 1) (car lista) (elem (- x 1) (cdr lista)))

)
```

```
(define (text-wh text font fontsize)
(let*
(
 (text-width 1)
 (text-height 1)
)

(set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
 PIXELS font)))
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
 fontsize PIXELS font)))

(list text-width text-height)
)
)
```

Meghívunk egy függvényt. A függvényünk a listánk x-edik elemét adja vissza a car és a cdr használatával. car : a lista első eleme. cdr : a lista elsőn kívüli összes eleme. A soron következő függvény segítségével tudjuk kiszámolni a szöveg magasságát.

```
(define (script-fu-bhax-chrome text font fontsize width height color ←
 gradient)
(let*
(
 (image (car (gimp-image-new width height 0)))
 (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
 LAYER-MODE-NORMAL-LEGACY)))
 (textfs)
 (text-width (car (text-wh text font fontsize)))
 (text-height (elem 2 (text-wh text font fontsize)))
 (layer2)
)
```

A script-fu-bhax-chrome metódus lesz felelős a króm effektért. A let\* segítségével létrehozzuk az objektumunkat a már megadott paraméterekből. Az image-be létrehozzuk a képünket a gimp-image-new használatával. Ezután létrehozunk egy új réteget a layer-ben. Létrehozzuk a textfs-t. A text-wh segítségével megadjuk a szöveg szélességét és magasságát. A legvégén pedig létrehozzuk layer2-t.

9 lépében megadjuk hogy krómosítson a szkriptünk.:

Első lépés:

```
;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND)
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
))
```

```
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
height 2) (/ text-height 2)))

(set! layer (car(gimp-image-merge-down image textfs ←
CLIP-TO-BOTTOM-LAYER)))
```

A `gimp-image-insert-layer` segítségével hozzáadjuk a képünkhez a rétegünket. A `gimp-context-select` segítségével beállítjuk az elsődleges színt feketére majd a beállított színnel kiszinezzük a rétegünket. Végük a színt fehérre állítjuk. A `textfs`-be létrehozunk egy új szövegréteget `gimp-text-layer-new` segítségével. Majd beállítjuk `gimp-layer-set-offsets` segítségével a réteg offsetjét. Összeolvasszuk a létrehozott szöveget és a hátteret. Ezzel el is értünk az első lépés végére.

Második lépés:

```
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével elmosódást állítunk be a `layer`-en.

Harmadik lépés:

```
;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
```

A `gimp-drawable-levels` segítségével beállítjuk a rétegünk szintezését.

Negyedik lépés:

```
;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével még egy elmosódást teszünk a `layer`-re.

Ötödik lépés:

```
;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)
```

A `gimp-image-select-color` segítségével kijelöljük a `layer` rétegen a fehér pixeleket. Ezt a ki-jelést megfordítjuk `gimp-selection-invert` segítségével.

Hatodik lépés:

```
;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)
```

Létrehozunk egy új réteget és ezt hozzáadjuk a képünkhez.

Hetedik lépés:

```
;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
 GRADIENT-LINEAR 100 0 REPEAT-NONE
 FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
 3)))
```

Átállítjuk aktívrá a gradient-t, majd a gimp-edit-blend segítségével összemossuk a layer2-t a háttérrel

Nyolcadik lépés:

```
;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
 0 TRUE FALSE 2)
```

A plug-in-bump-map segítségével domborítás effektet adunk a képhez.

Kilencedik lépés:

```
;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)
```

Beállítjuk a color-curvet-t, majd megjelenítjük egy új ablakban.

```
(script-fu-register "script-fu-bhax-chrome"
 "Chrome3"
 "Creates a chrome effect on a given text."
 "Norbert Bátfai"
 "Copyright 2019, Norbert Bátfai"
 "January 19, 2019"
 ""
 SF-STRING "Text" "Bátf41 Haxor"
 SF-FONT "Font" "Sans"
 SF-ADJUSTMENT "Font size" '(100 1 1000 1 10 0 1)
 SF-VALUE "Width" "1000"
 SF-VALUE "Height" "1000"
 SF-COLOR "Color" '(255 0 0)
 SF-GRADIENT "Gradient" "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
 "<Image>/File/Create/BHAX"
)
```

Menű beállítása

## 17.4 Gengszterek

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

Megoldás video:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/G%C3%B6del>

### Tutor

A Robocar World Championship egy olyan platform amely lehetőséget ad a robotautó kutatásra. A projekt központjában a Robocar City Emulator áll. A mi feladatunk az, hogy a carlexer.ll-ben található scanf függvényt értelmezzük.

Mik is azok a lambda kifejezések ?

A lambda kifejezések a C++ 11-es verziójában jelentek meg. Ezek a kifejezések lehetővé teszik az in-line functionok írását, vagyis egy vagy kevés soros függvényekét. Ezek a függvények úgynevezett "egyszer használatos" függvények. Nem adunk nekik nevet.

Példa egy lambda kifejezésre :

```
[] (int szam1, int szam2) -> { return szam1 + szam2; }
```

A szöglletes zárójel jelzi a lambda kifejezés kezdetét. Ezután találhatóak a paraméterek , majd a nyilacsát követően a kapcsos zárójelek között található a függvény az elvégezni kívánt művelettel. Itt derül ki továbbá, hogy mi lesz a visszatérési érték tipusa.

Most, hogy tisztáztuk mi is az a lambda kifejezés, nézzük a mi feladatunkat :

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

void sort (RandomAccessIterator first, RandomAccessIterator last, Compare ←
 comp);
```

A gangsters vektort fogjuk rendezni a sort függvény segítségével melynek első két paramétere gangster .gangsters.end(). A harmadik paraméter pedig a lambda kifejezésünk aminek a paramétere 2 Gangster lesz. A kifejezés egy boolean értéket (true-t) ad vissza, ha a x gengszer rendőrtől való távolsága kisebb mint y gengszeré. Magyarán a vektorunk a gengszterek rendőrtől való távolsága szerint lett rendezve. Az utolsó sorban található sortfüggvény teszi lehetővé, hogy az összehasonlítási szempontokat mi magunk adjuk meg, jelen esetben egy lambda kifejezést.

# Chapter 18

## Helló, !

### 18.1 OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll>

Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll>

Első sorban nézzük mit takar az OOCWC rövidítés : Az OOCWC a rObOCar World Championship szóból ered, ami egy olyan platform ami lehetőséget ad a robotautó-kutatásra. Az egész projekt középpontjában a Robocar City Emulator áll.

A mi feladatunk az lesz, hogy megvizsgáljuk a carlexer.ll-ben található scanf függvényt.

Vessünk egy pillantást az említett függvényre :

```
while (std::sscanf (data+nn, "<OK %d %u %u %u>%n", &idd, &f, &t, &s, &n) == 4)
{
 nn += n;
 gangsters.push_back (Gangster {idd, f, t, s});
}
```

A sscanf függvény formázott stringből olvas be adatokat. (a sima scanf függvénnyel ellentétben.)

Két részből állnak : buffer-ből és format-ból. A buffer egy karakterstringre mutató pointer. Innen kerülnek majd kiolvasásra az adatok. Ami pedig meghatározza, hogy az adatok hogyan kerüljenek kiolvasásra az a format lesz. Format-specifikátorokból áll, amik %-lel kezdődnek.

A d az int-ekre illeszkedik. Az u az unsigned integerre (magyarul : előjel nélküli integerre) illeszkedik. Az n pedig számon tartja a már beolvasott karakterek számát.

Addig tudjuk beolvasni az adatokat, ameddig nincs meg mind a 4 argumentum. (while == 4)

Ha mind a 4 várt argumentumunkat be tudtuk olvasni, új Gangster kerül létrehozásra a megfelelő argutmentumokkal és a gangster vektorban tároljuk az új gengszterünket.

Az nn változóban tároljuk az összes beolvasott karakterek számát. Ez a változó a sscanf függvényben is megtalálható, nem máshol, mint a buffer részben. A data értékét tehát azért növejül mindenkorán, hogy onnan tudunk adatokat beolvasni ahonnan azelőtt még ezt nem tettük.

## 18.2 BrainB

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

Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/esport-talent-search>

A BrainB projekt célja a jövő esportolónak felkutatása. A program az agy úgynevezett kognitív képességét méri.

A játékos feladata a Samu Entropy nevű karakteren tartani az egérmutatót ami az idő teltével természetesen egyre nehezebb hiszen újabb és újabb Entropy karakterek jelennek meg a képernyőn. A program érzékeli, ha Samu Entropyt elvesztettük, ekkor csökkenti a többi Entropy számát.

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

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

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

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

 qout << "This program is free software: you can redistribute it and/or modify it under" << endl;

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

Includoljuk a BrainBWin osztályt , a többi inkludolásra kerülő osztályt már az előző programokban megfigyelhetünk , ugyanis ez a program is Qt grafikus felületet használ. Deklarálunk egy app QApplication típusú objektumot. Itt használjuk a qout functiont amivel a standard outputra lehet írni. Deklaráljuk az entropykat. Majd létrehozzuk a BrainBWin objektumot. Az osztályban inkludolásra került függvények és változók itt is .h kiterjesztésű header fájlokban vannak.

### BrainBWin.h

Deklaráljuk azt a függvényt ami az egér és a billentyűzetről történő inputok figyeléséért felelős. Deklaráljuk továbbá a konstruktort és a destruktort.

### BrainBWin.cpp

Itt kerül definiálásra az előző header fájlban deklaráltak.

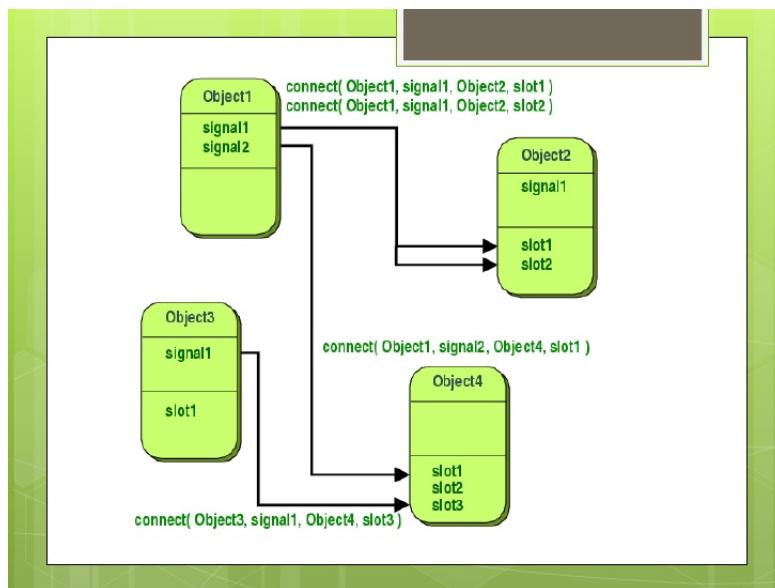
### BrainBThread.h

A Hero-kat tartalmazó vectorra typedefet használunk, mostantól `Hero`ként hivatkozunk rá.

### BrainBThread.cpp

Létrehozzuk a Hero-kat. Definiáljuk a destruktort és deklaráljuk a `run`, `pause` és `set_paused` függvényeket

Most , hogy ezt tisztáztuk, nézzük meg, hogy mi az az a Qt slot-signal mechanizmusa ?



Kép forrása : <https://www.slideshare.net/auroraeosrose/event-and-signal-driven-programming>

Mit látunk a képen ?

Signal-okat és Slot-okat. Először is tisztázzuk a fogalmakat. A Signal-ok az objektumok által kerülnek kibocsátásra. Magyarul leegyszerűsítve, ha az adott objektum belső állapota megváltozik akkor jelként lesz kiküldve.

Ezekhez a signal-okhoz vannak kapcsolódva a slot-ok, amik ha a hozzájuk tartozó signal kiküldésre kerül akkor hívódnak meg. Ezek a slotok egyszerű C++ függvények.

Lehetséges olyan eset is, hogy egy signalhoz több slot is tartozik, ekkor a slot-ok egymás után sorban kerülnek végrehajtásra. De az is elközelhető, hogy egy slothoz tartozzon több Signal.

A slot-okat és a signal-okat egy `connect` függvénnyel kapcsoljuk össze minden esetben.

A `BrainBWin.cpp` fájlban találhatunk a fent említett példát :

```
connect (brainBThread, SIGNAL (heroesChanged (QImage, int, int)),
 this, SLOT (updateHeroes (QImage, int, int)));

connect (brainBThread, SIGNAL (endAndStats (int)),
 this, SLOT (endAndStats (int)));
```

A slot-ok és signal-ok paramétereinek típusa és száma megegyezik. Az első signalhoz az updateHeroes még a másikhoz a endAndStats függvény tartozik.

Tekintsük meg a függvényeket amelyek meghívásra kerülnek :

egyik :

```
void BrainBWin::endAndStats (const int &t)
{

 qDebug() << "\n\n\n";
 qDebug() << "Thank you for using " + appName;
 qDebug() << "The result can be found in the directory " + ←
 statDir;
 qDebug() << "\n\n\n";

 save (t);
 close();
}
```

másik :

```
void BrainBWin::updateHeroes (const QImage &image, const int &x, ←
 const int &y)
{

 if (start && !brainBThread->get_paused()) {

 int dist = (this->mouse_x - x) * (this->mouse_x - x) + ←
 (this->mouse_y - y) * (this->mouse_y - y);

 if (dist > 121) {
 ++nofLost;
 nofFound = 0;
 if (nofLost > 12) {

 if (state == found && firstLost) {
 found2lost.push_back (brainBThread ←
 ->get_bps());
 }

 firstLost = true;

 state = lost;
 nofLost = 0;
 //qDebug() << "LOST";
 //double mean = brainBThread->meanLost();
 //qDebug() << mean;

 brainBThread->decComp();
 }
 } else {
 }
```

```
++nofFound;
nofLost = 0;
if (nofFound > 12) {

 if (state == lost && firstLost) {
 lost2found.push_back (brainBThread ->get_bps());
 }

 state = found;
 nofFound = 0;
 //qDebug() << "FOUND";
 //double mean = brainBThread->meanFound();
 //qDebug() << mean;

 brainBThread->incComp();
}

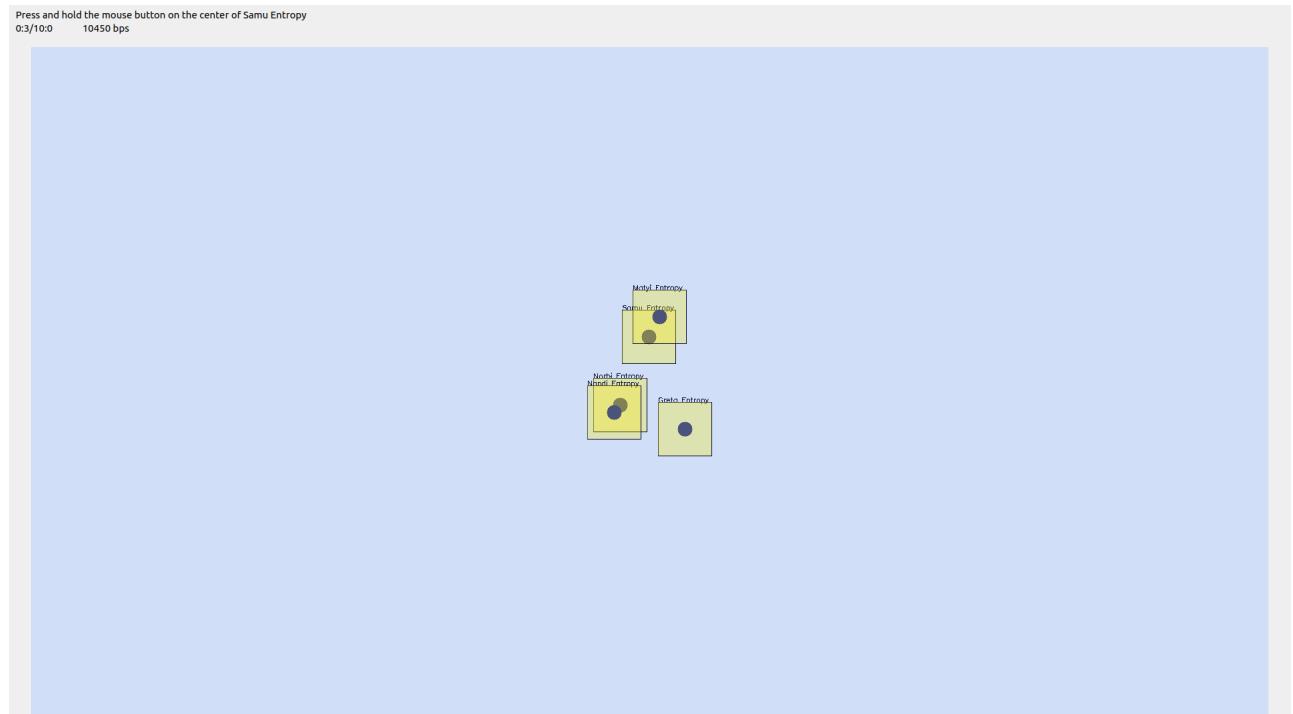
}

pixmap = QPixmap::fromImage (image);
update();
}
```

Az endAndStats függvény fog felelni , hogy ha a rendelkezésre álló idő lejár akkor az eredményeink eltárolásra kerüljenek egy fájlban, illetve kapunk egy búcsúüzenetet.

Az updateHeroes a hősök frissítéséért felel.

A fordításhoz és futtatáshoz a QT Creator-t hívtam segítségül :



## 18.3 SamuCam

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

Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/SamuCam>

A feladatunk az, hogy mutassunk rá a webcamra.

A következőkben a SamuCam.cpp fájlunkat fogjuk elemezni kódcsipetekre bontva :

```
#include "SamuCam.h"

SamuCam::SamuCam (std::string videoStream, int width = 176, int ←
 height = 144)
 : videoStream (videoStream), width (width), height (height)
{
 openVideoStream();
}

SamuCam::~SamuCam ()
{
}
```

A legelső lépésben inkludoljuk a header fájlunkat. Létrehozzuk a konstruktorunkat , ami 3 paraméterrel rendelkezik. Létrehozásra kerül a destruktur is.

```
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);
}
```

Az openVideoStream függvényen belül található a VideoCapture, annak az open függvénye és az open argumentuma. Az eszköz indexét találhatjuk meg ott. Ezek az index 0 az alapértelmezett kamera eszközünk neve. Majd megtörténik a szélesség, magasság és fps szám megadása.

```
void SamuCam::run ()
{
 cv::CascadeClassifier faceClassifier;

 std::string faceXML = "lbpcascade_frontalface.xml"; // https://←
 github.com/Itseez/opencv/tree/master/data/lbpcascades

 if (!faceClassifier.load (faceXML))
 {
```

```
 qDebug() << "error: cannot found" << faceXML.c_str();
 return;
}

cv::Mat frame;
```

Következik a run függvényünk. Szükségünk lesz egy `CascadeClassifier`-re.

## Mi is az a CascadeClassifier?

A CascadeClassifier-t alapesetben egy adott tárgy detektálására szokták használni OpenCV-s programokban. Esetünkben a feladata az arc felismerés lesz. Ehhez viszont le kell tölteni a megjegyzésben található link segítségével egy xml fájlt és egy stringbe kimenteni. Ez az xml tartalmazza a arc felismeréséhez szükséges algoritmust. A load függvény segítségével hívjuk meg az xml-t , ha hiányzik az xml fájlunk akkor pedig hibaüzenetet kapunk vissza.

```
while (videoCapture.isOpened())
{
 QThread::msleep (50);
 while (videoCapture.read (frame))
 {
 if (!frame.empty())
 {
 cv::resize (frame, frame, cv::Size (176, 144), 0, 0, ←
 cv::INTER_CUBIC);

 std::vector<cv::Rect> faces;
 cv::Mat grayFrame;

 cv::cvtColor (frame, grayFrame, cv::COLOR_BGR2GRAY);
 cv::equalizeHist (grayFrame, grayFrame);

 faceClassifier.detectMultiScale (grayFrame, faces, 1.1, ←
 3, 0, cv::Size (60, 60));

 if (faces.size() > 0)
 {
 cv::Mat onlyFace = frame (faces[0]).clone();

 QImage* face = new QImage (onlyFace.data,
 onlyFace.cols,
 onlyFace.rows,
 onlyFace.step,
 QImage::Format_RGB888);

 cv::Point x (faces[0].x-1, faces[0].y-1);
 cv::Point y (faces[0].x + faces[0].width+2, faces ←
 [0].y + faces[0].height+2);
 }
 }
 }
}
```

```
cv::rectangle (frame, x, y, cv::Scalar (240, 230, ←
 200));

 emit faceChanged (face);
 }

QImage* webcam = new QImage (frame.data,
 frame.cols,
 frame.rows,
 frame.step,
 QImage::Format_RGB888);

emit webcamChanged (webcam);

}

QThread::msleep (80);

}
```

A while függvényen belül a program 50 ms-onként ellenőrzi, megvan-e nyitva a kameránk. Ha meg van nyitva a kameránk, akkor az adott képkockák beolvasásra majd tárolásra kerülnek. A kép átméretezésre kerül a resize segitségével. Következik a faces vektor, majd a képünket szürkévé színezzi a cvtColor segitségével. Ennek is létrehozunk egy tárolót aminek a neve GrayFrame lesz. Majd elkezdjük keresni az arcot a detectMultiScale segitségével. A megtalált arcok egy rectangle kerülnek tárolásra. QImage készül az arcrol, majd egy webcamra nevű QImage is készül. Mindez 80ms-onként ismétlődik.

```
if (! videoCapture.isOpened())
{
 openVideoStream();
}
```

Ha nincs megnyitva a kameránk, akkor megnyitja azt.

Fordítjuk majd futtatjuk a prgramunkat.

## 18.4 FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>  
Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/future/tree/master/cs/F6>

A feladatunk, hogy bugokat javítsunk a ActivityEditor.java fájlban.

Fordítás és futtatás után amikor megpróbálnánk új altevékenységet létrehozni , rögtön egy buggal találjuk szembe magunkat, ugyanis nem tudunk létrehozni új altvékenységet.

The screenshot shows a Java development environment with the following components:

- Code Editor:** The main window displays Java code for `ActivityEditor.java`. The code includes logic for creating new Java files and adding them to a tree view.
- Terminal:** A terminal window shows the process of cloning a GitHub repository named `future` into the current directory. It also shows the execution of `javac ActivityEditor.java` and `java ActivityEditor`, which fails due to missing modules.
- File Browser:** A sidebar titled "Tevékenységek fája és a tevékenységekhez hozzárendelt tulajdonságok" (Properties of activities and properties assigned to activities) lists various activity types and their properties.
- Code Editor (Bottom):** A smaller code editor window at the bottom shows part of the `ActivityEditor.java` file.

A bugra egy lehetséges megoldása egy bevezetett számláló segítségével :

```

 boolean sikertule = false;
 java.io.File f;
 int i = 1;
 while(true){
 f = new java.io.File(file.getPath() + System.getProperty("file.separator") + "Új altevénység" + i);

 if (f.mkdir()) {
 javafx.scene.control.TreeItem<java.io.File> newAct
 = new javafx.scene.control.TreeItem<java.io.File>(f, new javafx.scene.image.ImageView(actIcon));
 = new FileTreeItem(f, new javafx.scene.image.ImageView(actIcon));
 getTreeItem().getChildren().add(newAct);
 sikertule = true;
 break;
 } else {
 i++;
 }
 }
 if(!sikertule){
 System.err.println("Cannot create " + f.getPath());
 }
 });
}

```

Mostmár ha egy új altevénységet szeretnénk létrehozni, azt nyugodt szívvel megtehetjük :

The screenshot shows a Java development environment with the following details:

- Title Bar:** Tevékenységek ActivityEditor
- File Menu:** Fájl, Szerkesztés, Nézet, Keresés, Terminál, Súgó
- Code Editor:** The current file is `ActivityEditor.java`. The code implements a tree item for creating new files. It uses `javafx.scene.control.TreeItem<java.io.File>` to represent files and `javafx.scene.image.ImageView` for icons. It handles file creation via `java.io.File.createNewFile()`.
- Terminal:** Shows command-line output from the terminal window:

```
hallgato@deikpc:~/Letöltések/future/cs/F6$ javac ActivityEditor.java
hallgato@deikpc:~/Letöltések/future/cs/F6$ java ActivityEditor
Gtk-Message: 14:58:55.137: Failed to load module "canberra-gtk-module"
```
- File Browser:** A sidebar titled "Tulajdonságok fája" (Properties) lists the following items:
  - En, magam
  - Külcsei
  - Belbecs
- File List:** A sidebar titled "Tevékenységek fája és a tevékenységekhez hozzárendelt tulajdonságok" (Files and properties) shows a tree structure:
  - City
    - Debrecen
      - Sport
      - Új altevékenység1
      - Új altevékenység2
      - Új altevékenység3
      - Új altevékenység4
      - Új altevékenység5
- Status Bar:** Line 259, Column 27

# Chapter 19

## 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>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Lauda>

Legelőször vessünk egy pillantást a kódra :

```
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");

 }
 }
}
```

Nézzük mit csinál a programunk :

A programunk egy for ciklussal kezdődik. A programunkban az i ciklus változót növeljük eggyel egészen addig , ameddig i kisebb, mint 1024, mindenkorban az i-vel egyenlő sorszámú porton próbál TCP kapcsolatot létesíteni. Ezt a try első sora segítségével tudjuk megcsinálni. Ha az aktuális porton kapcsolat létesítés folyik, akkor a port kiiratásra kerül, Majd lezárjuk a pocketet.

A feladatunk a kivételkezelés ismertetése volt , szóval nézzük mi is az a kivételkezelés ?

A kivételkezelés egy olyan programozási mechanizmus, amely során kezelni tudjuk a program futását megakadályozó eseményt vagy eseményeket. Ha egy ilyen esemény megszakítaná a programunk zavar-talan végrehajtását akkor általában végrehajtódik egy előre regisztrált kivételkezelő. Ez ettől függ, hogy hardveres vagy szoftveres kivételről van-e szó. A szoftveres kivételeket kezelése mellett általában folytatni lehet a programot ott ahol megakadt.

Nézzük a fentiekben kifejtett programunk hibakeresését :

```
java.net.Socket socket = new java.net.Socket(args[0], i);
```

Itt próbál meg a programunk kapcsolatot kialakítani. De ha ez nem sikerül akkor IOException-t fog dobni. A catch ág ezt a kivételt elkapja és kiírja, hogy hányas számú port "nem figyeli".

Nézzük a JDK forrást :

```
public Socket(String host, int port)
 throws UnknownHostException, IOException
{
 this(host != null ? new InetSocketAddress(host, port) :
 new InetSocketAddress(InetAddress.getByName(null), port),
 (SocketAddress) null, true);
}
```

Láthatjuk, hogy a JDK forrásban található függvény két kivételt dobhat : UnknownHostException-t és IOException-t

Fordítjuk majd futtatjuk a programunkat :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Lauda$ javac KapusZkenner.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Lauda$ java KapusZkenner 127.0.
0..1
0 nem figyeli
1 nem figyeli
2 nem figyeli
3 nem figyeli
4 nem figyeli
5 nem figyeli
6 nem figyeli
7 nem figyeli
8 nem figyeli
9 nem figyeli
10 nem figyeli
11 nem figyeli
12 nem figyeli
13 nem figyeli
14 nem figyeli
15 nem figyeli
```

```
028 nem figyeli
029 nem figyeli
030 nem figyeli
031 figyeli
032 nem figyeli
033 nem figyeli
034 nem figyeli
035 nem figyeli
```

```
1014 nem figyeli
1015 nem figyeli
1016 nem figyeli
1017 nem figyeli
1018 nem figyeli
1019 nem figyeli
1020 nem figyeli
1021 nem figyeli
1022 nem figyeli
1023 nem figyeli
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Lauda$
```

## 19.2 Android Játék

Írunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Hello, Android!” feladatára!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Lauda>

Tutor és videó

A játék a jól ismert, számkitalálós játék. A gép egy random számot sorsol 1 és 100 között, a felhasználó pedig megpróbálja a lehető legkesebb tippből kitalálni, hogy mi a gondolt szám. A felhasználó segítséget kap, ugyanis minden tipp után kap egy üzenetet, hogy kisebb vagy nagyobb számot kell-e tippelni, vagy megtalálta-e az adott számot.

Nézzük magát a kódot :

```
<resources>
 <string name="app_name">guessingGame</string>
 <string name="start_msg">Találd ki a számot! (1 és 100 között)</string>
 <string name = "too_high">Kisebb számmal próbálkozz!</string>
 <string name = "too_low">Nagyobb számmal próbálkozz!</string>
</resources>
```

A játékunk egyik legfontosabb alkotó eleme, az az üzenet amelyet a játékos egy-egy tipp után kap. Ezek az üzenetek a strings.xml fájlban kerülnek létrehozásra string formában.

Sok xml kiterjesztésű fájl áll rendelkezésünkre a játékunk kicsinosítása érdekében (ilyen például a colors.xml , dimens.xml) de ezeket most békénhagyjuk.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 tools:context="com.ssaurel.higherlower.MainActivity">

 <TextView
 android:id="@+id/msg"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="50dp"
 android:text="@string/start_msg"
 android:textSize="22sp"
 android:textStyle="bold"/>

 <EditText
 android:id="@+id/numberEnteredEt"
```

```
 android:layout_width="200dp"
 android:layout_height="wrap_content"
 android:layout_below="@+id/msg"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="50dp"
 android:inputType="number"/>

 <Button
 android:id="@+id/validate"
 android:layout_width="200dp"
 android:layout_height="wrap_content"
 android:layout_below="@+id/numberEnteredEt"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="60dp"
 android:text="Ellenőriz" />

</RelativeLayout>
```

Az `activity_main.xml` fájlunkban létrehozásra kerül a felhasználói interfész. A `TextView` lesz felelős a felhasználónak címzett üzenetek megjelenítéséért. Az `EditText` teszi lehetővé a játékos számára, hogy egy adott tippet (számot) be tudjon írni. A `Button` biztosítja az ellenőrzésre szolgáló gomb működését.

Nézzük meg a `MainActivity.java` fájlunkat.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 tools:context="com.ssaurel.higherlower.MainActivity">

 <TextView
 android:id="@+id/msg"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="50dp"
 android:text="@string/start_msg"
 android:textSize="22sp"
 android:textStyle="bold"/>

 <EditText
 android:id="@+id/numberEnteredEt"
 android:layout_width="200dp"
 android:layout_height="wrap_content"
 android:layout_below="@+id/msg"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="50dp"
 android:inputType="number"/>
```

```
<Button
 android:id="@+id/validate"
 android:layout_width="200dp"
 android:layout_height="wrap_content"
 android:layout_below="@+id/numberEnteredEt"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="60dp"
 android:text="Ellenőriz"/>

</RelativeLayout>
```

Az activity\_main.xml fájlunkban kerül létrehozásra a felhasználói interfész. A TextView lesz felelős a felhasználónak címzett üzenet megjelenítéséért. Az EditText teszi lehetővé a játékos számára, hogy egy adott tippet (számot) be tudjon írni. A Button biztosítja az ellenőrzésre szolgáló gomb működését.

Nézzük meg a MainActivity.java fájlunkat.

```
package com.example.guessinggame;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.util.Random;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

 public static final int MAX_NUMBER = 100;
 public static final Random RANDOM = new Random();
 private TextView msgTv;
 private EditText numberEnteredEt;
 private Button validate;
 private int numberToFind, numberTries;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 msgTv = (TextView) findViewById(R.id.msg);
 numberEnteredEt = (EditText) findViewById(R.id.←
 numberEnteredEt);
 validate = (Button) findViewById(R.id.validate);
 validate.setOnClickListener(this);

 newGame();
 }
```

A már megszokott módon legelőször importáljuk a számunkra szükséges könyvtárakat. Ezt követően létrehozzuk a változóinkat és konstansainkat. Majd deklaráljuk a `onCreate` metódust is.

```
@Override
public void onClick(View view) {
 if (view == validate) {
 validate();
 }
}
```

Az ELLENŐRZÉS gomb működéséért az `onClick` metódus felel, ha megnyomjuk ezt a gombot akkor meghívódik a `validate` metódus.

```
private void validate() {
 int n = Integer.parseInt(numberEnteredEt.getText().toString());
 numberTries++;

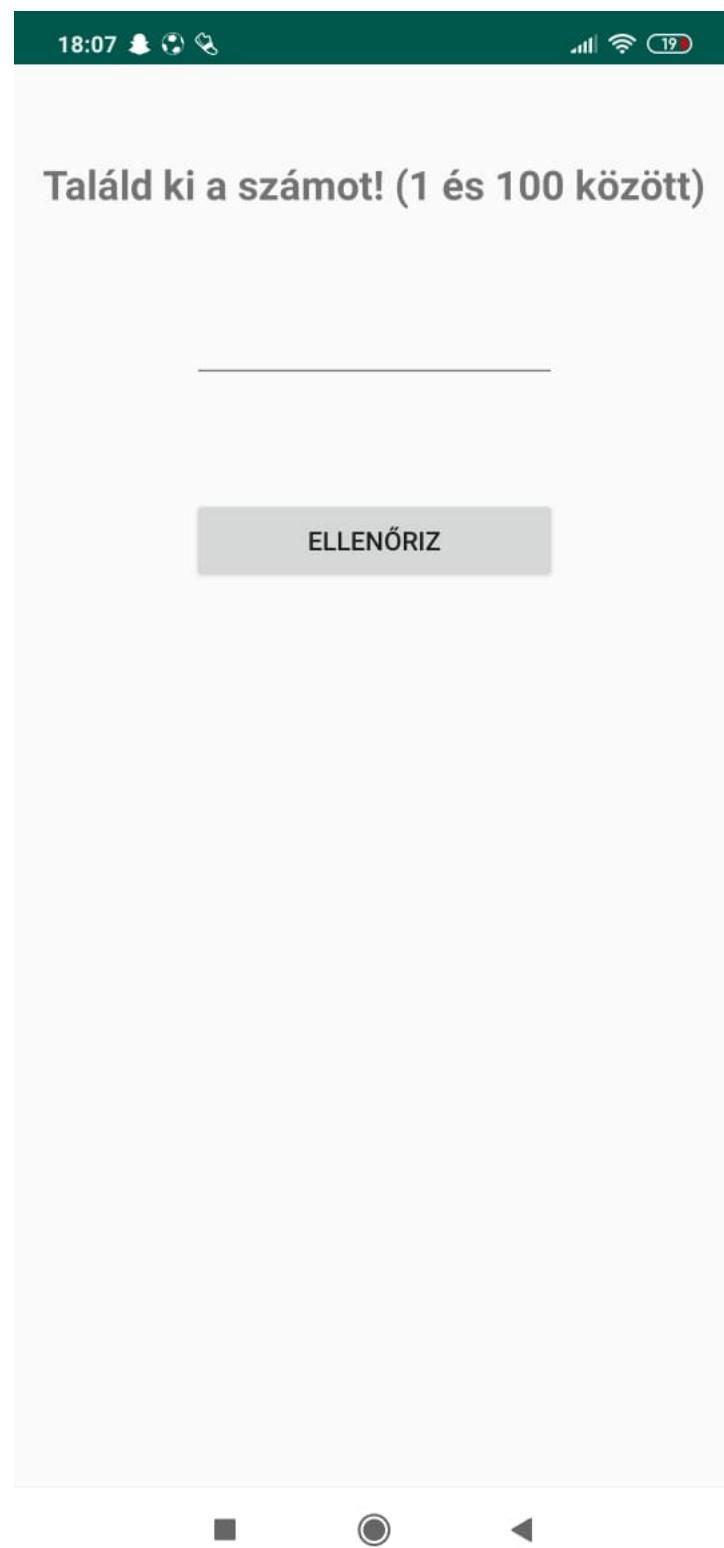
 if (n == numberToFind) {
 Toast.makeText(this, "Gratulálok! Megvan a szám! " +
 numberToFind +
 " Összesen " + numberTries + " próbálkozásra " +
 "volt szükséged", Toast.LENGTH_SHORT).show();
 newGame();
 } else if (n > numberToFind) {
 msgTv.setText(R.string.too_high);
 } else if (n < numberToFind) {
 msgTv.setText(R.string.too_low);
 }
}
```

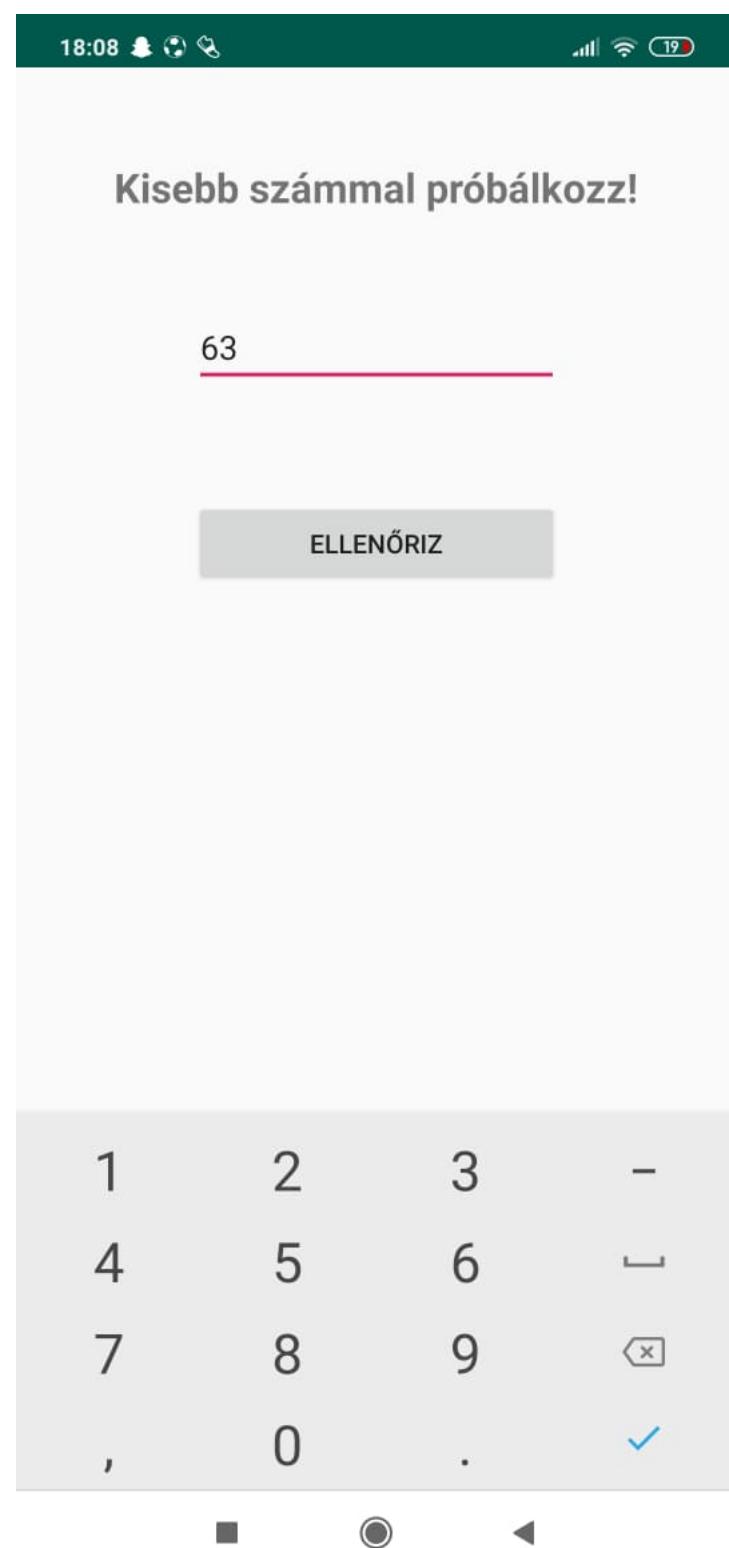
A `validate` metódusban : eltároljuk a felhasználó által beadott számokat az `n` változóban. Egy számláló értékét addig növeljük ameddig a játékosnak el nem sikerült találnia, hogy mi a gondolt szám. Ha a játékosnak sikerült eltalálnia a gondolt számot, gratulálunk neki, és eláruljuk neki, hogy hány tippből jött össze a gondolt szám kitalálása. Majd meghívjuk a `newGame` metódust, amivel új játék kezdésére van lehetőség. Ha nem sikerült kitalálni a gondolt számot, akkor a játék segítséget ad, hogy feljebb vagy lejjebb kell-e számokat tippelnie a játékosnak.

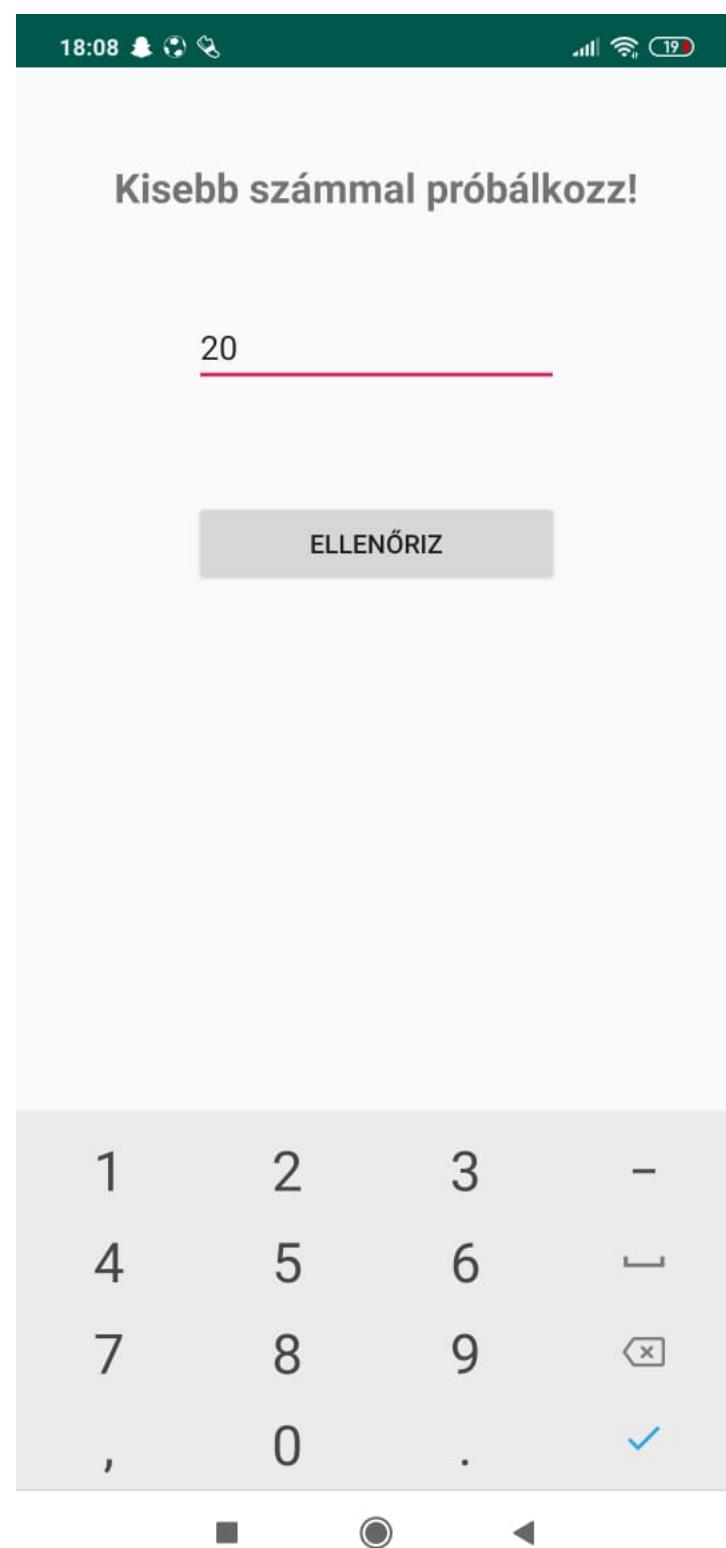
```
private void newGame() {
 numberToFind = RANDOM.nextInt(MAX_NUMBER) + 1;
 msgTv.setText(R.string.start_msg);
 numberEnteredEt.setText("");
 numberTries = 0;
}
```

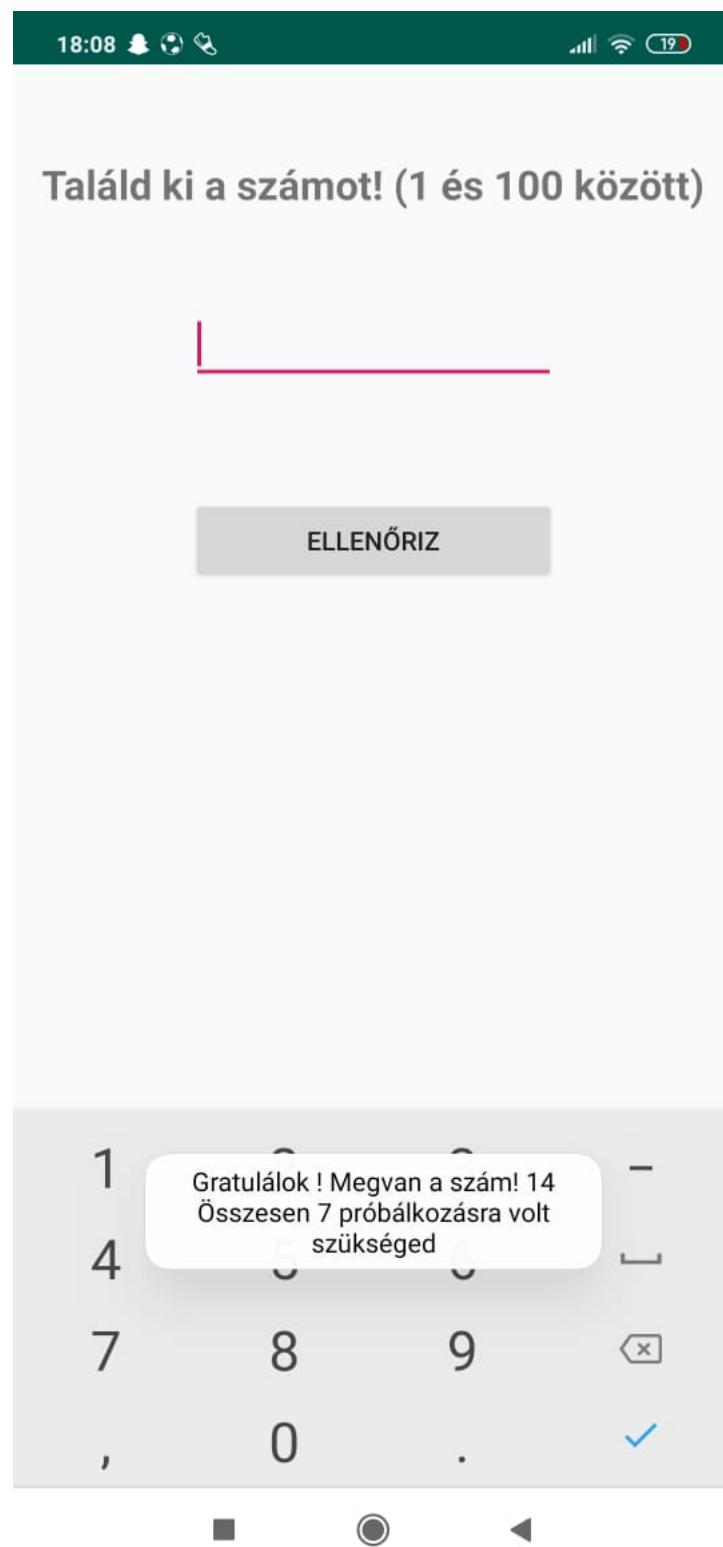
A newGame metódusban történik egy random szám sorsolás, ami a "gondolt" szám lesz, elátoljuk ezt a számot, kiíratunk egy üzenetet a játékosnak majd nullázzuk a számlálót.

Nézzük a játékot :









### 19.3 Junit teszt

A [https://propater.blog.hu/2011/03/05/labormeres\\_oththon\\_avagy\\_hogyan\\_dolgozok\\_fel\\_ely\\_pedat\\_poszt\\_kézzel\\_számított\\_mélységeit\\_és\\_szórását\\_dolgozd\\_be\\_egy\\_Junit\\_tesztbe\\_sztenderd\\_védési\\_feladat\\_volt\\_korábban](https://propater.blog.hu/2011/03/05/labormeres_oththon_avagy_hogyan_dolgozok_fel_ely_pedat_poszt_kézzel_számított_mélységeit_és_szórását_dolgozd_be_egy_Junit_tesztbe_sztenderd_védési_feladat_volt_korábban))

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Lauda>

Tutor és olvasni való

Kezdjük azzal, hogy mi is az a JUnit?

A JUnit egy olyan keretrendszer amit egységesztelés során szoktunk használni Java nyelv mellé. Abban az esetben beszélhetünk egységesztelésről, ha egy adott kóddal együtt az adott kódot tesztelő osztállyal együtt kerül fejlesztésre.

Nézzük a kódot :

```
public class BinfaTest {
 LZWBinFa binfa = new LZWBinFa();

 @org.junit.Test
 ...
 ...
```

A harmadik sorban találkozhatunk egy @-cal kezdődő sorral. A @-cal kezdődő sor jelöli azt, hogy itt kezdődik a metódus, amit a JUnit tesztfuttatója fog futtatni.

Amire figyelni kell, hogy ha több ilyen metódusunk is van akkor nem lesz egyértelmű a futtatási sorrend, ezért úgy kell kialakítani ezeket, hogy egymástól függetlenek legyenek.

Egy teszt tehát @ jelöléssel kezdődik. Törzsében megtalálható a tesztelendő metódus, majd a végrehajtás után kapott tényleges eredményt össze kell venni az elvárt eredményvel.

```
public void tesBitFeldolg() {
 for (char c : "01111001001001000111".toCharArray())
 {
 binfa.egyBitFeldolg(c);
 }
```

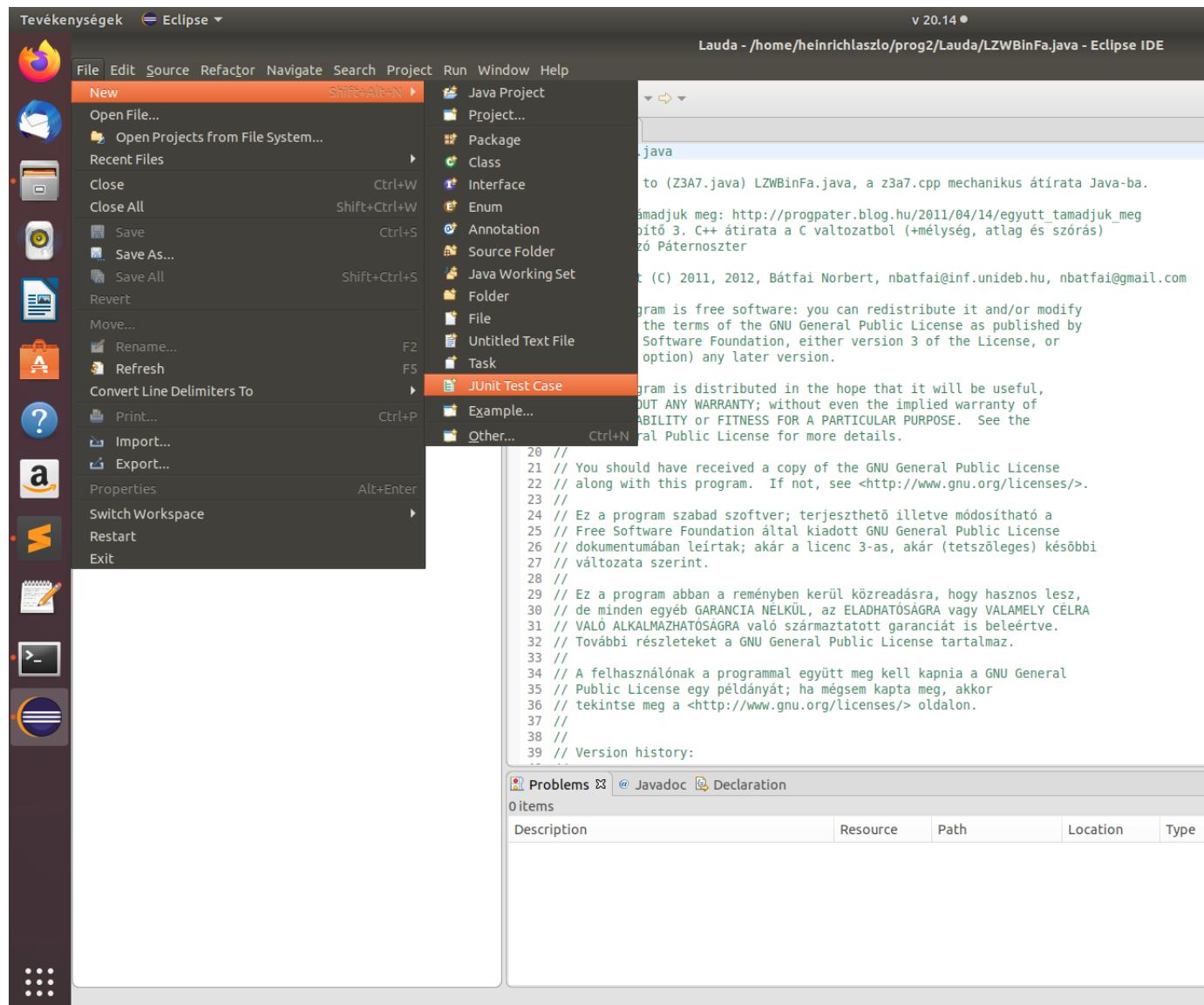
Következik a tesBitFeldolg függvény ami a tesztünk lesz.

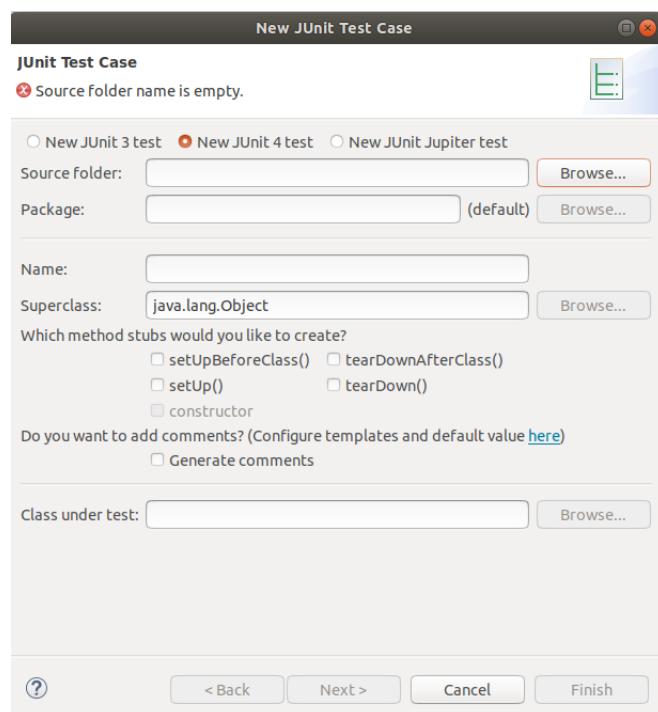
Az egybitFeldolg függvényel a megadott tömböt dolgozzuk fel karakterenként.

```
 org.junit.Assert.assertEquals(4, binfa.getMelyseg(), 0.0);
 org.junit.Assert.assertEquals(2.75, binfa.getAtlag(), 0.001);
 org.junit.Assert.assertEquals(0.957427, binfa.getSzoras(), ←
 0.0001);
}
```

Az assertEquals függvénytel vizsgáljuk meg, hogy mennyi az eltérés a várható mélység, átlag és szórás valamint ezeknek a tényleges értékei között. A függvényünk három paraméterrel rendelkezik. Az első a várt érték , magyarul az amit kapunk kellene, a második a programunk által kapott érték, a harmadik pedig az , hogy menyi lehet a maximális eltérés a két előbbi érték között.

Ellenőrzéshez az Eclipse-t hívtam segítségül :





# Chapter 20

## Helló, Calvin!

### 20.1 MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, [https://progpater.blog.hu/2016/11/13/hello\\_sbol](https://progpater.blog.hu/2016/11/13/hello_sbol) Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Calvin>

Mi is az az MNIST?

Leegyszerűsítve egy kézzel írott arab számjegyeket tartalmazó adatbázis. Az adatbázis összesen 60000 darab 28x28 pixel méretű , greyscale, PNG képállományt tartalmaz. Ezt az adatbázist ( a 60000 képet ) két részre tudjuk bontani. Első rész a tanítási, ez 50000 darab képet foglal magába. A maradék 10000 pedig a tesztelési képeket tartalmazza. Az első résznek köszönhetően tanulja meg a gép a számjegyeket, majd a másdoik részben ellenőrzi a tanulásnak a sikerességét.

Nézzük a kódunkat :

```
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
```

A programunk elején importáljuk a megoldáshoz szükséges könyvtárakat. Ha ezek nem találhatóak meg a géünkön akkor a pip install 'library\_neve' parancs segítségével tudjuk letölteni.

```
(train_X,train_Y), (test_X,test_Y) = tf.keras.datasets.mnist. ←
 load_data()
```

Lehetővé teszi számunkra az adatbázissal való tevékenykedést. A load.data() segítségével töltjük be a több tízezer képalloányt amellyekkel a későbbiekben dolgozni fogunk. A betöltött képalloányok a vektorban kerülnek tárolásra.

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

Következnek azon vektorok elkészítése amelyeket a tanításra és tesztelésre való számok tárolására fogunk használni. 28 db, 28 db elemet tartalmazó vektorra bontjuk az adatainkat. Első paraméterünk -1, magyarul ezt minden tagra eljátszik.

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

Annak érdekében, hogy a lehető leggyorsabb tanítási folyamatot elérjük a vektorok típusait átállítjuk, és elosztjuk a megadott számokkal , hogya a képeket alkotó képpontok értéke mostmár megfelelő legyen számunkra.

```
train_Y_one_hot = to_categorical(train_Y, 10)
test_Y_one_hot = to_categorical(test_Y, 10)
model = Sequential()
```

A one\_hot encoding segítségével a modell nem tud kategorikus adatokkal dolgozni. Ennek köszönhetően 0-sok és 1-esek sorozatára fogjuk felbontani az adott számot. Majd beállítjuk a model típusát is.

```
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'])
```

Az add függvényel tudunk a modellünkhez újabb rétegeket hozzáadni. Első sorban például egy konvolúciós réteget adunk hozzá a modellünkhez. Aminek az első paramétere a neuronok száma , második a detektor, harmadik pedig az input shape, ami jelenleg a 28x28 greyscale állomány lesz. Majd aktiváljuk a ReLu-t (Rectified Linear Unit) azaz a helyesbített lineáris egység. Következő sorban találkozhatunk a pool\_size-ral amiben az kerül feldolgozásra, hogy mennyi adat kerüljön feldolgozásra, ennek a két argumentuma van, egy függőleges illetve egy vízszintes érték. Majd a compile függvényel megindítjuk a tanulási folyamatot.

```
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)

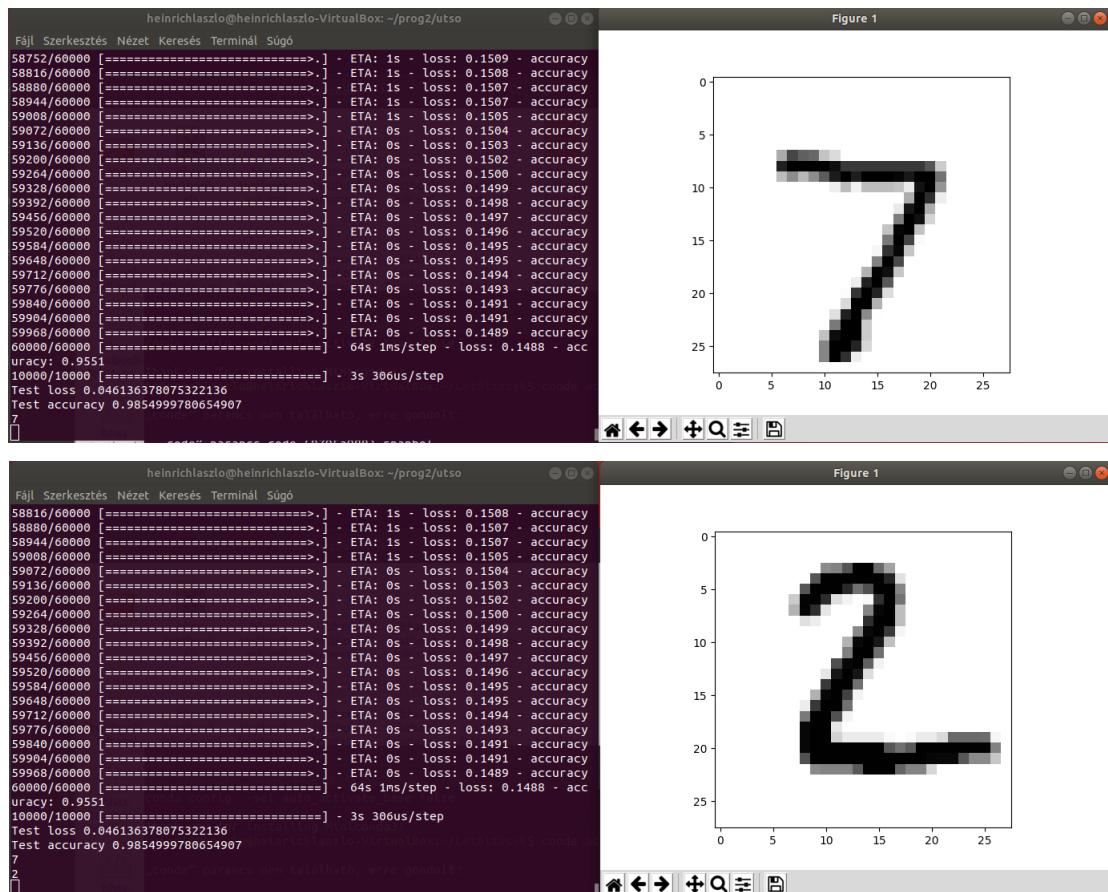
predictions = model.predict(test_X)

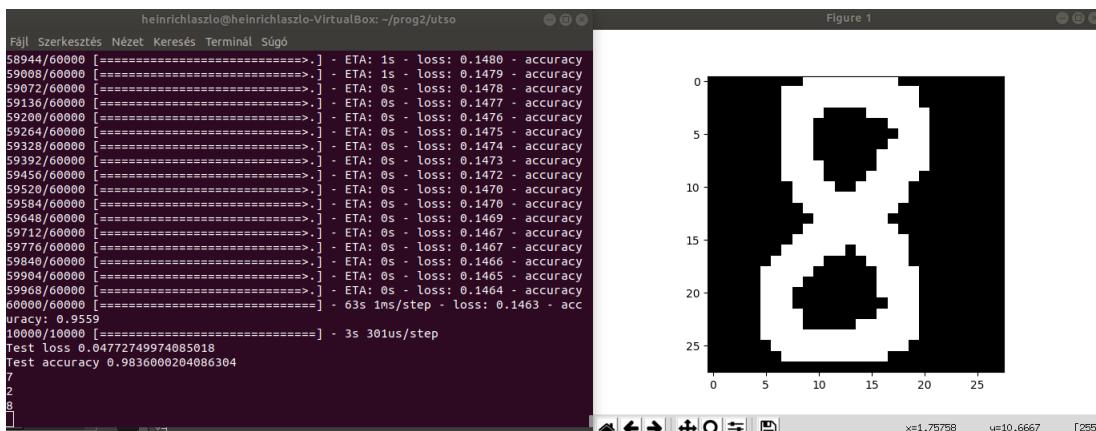
print(np.argmax(np.round(predictions[0])))
plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
print(np.argmax(np.round(predictions[1])))
plt.imshow(test_X[1].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
img = Image.open('one.png').convert("L")
img = np.resize(img, (28,28,1))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

A fit függvény segítségével beállítjuk, hogy mik lesznek a tanítás jellemzői. A batch\_size lesz a tanulási sebesség. Az epochs pedig az, hogy a folyamat hányszor kerüljön vérehajtásra.

Kiíratásra kerül a tanulási folyamat során keletkező veszteség, illetve a pontosság értéke is. Eltároljuk a prediction-öket. Megjelenítjük az első illetve a második feldolgozott képállományunkat is a gép általi predictonnal együtt. A harmadik kép a mi képünk lesz a Image.operon függvény segítségével.

Nézzük sikerül -e felismerni az álltalunk megadott 8as számot :





Láthatjuk, hogy majdnem 100%os sikereséggel tudja felismerni a számokat, illetve a mi kézzel írott 8-asunkat is sikerült felismernie.

## 20.2 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)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/tree/master/Calvin>

### Tutor

A feladatunk nagyban hasonlít a MNIST-es feladatunkhoz. A különbség az, hogy ameddig az MNIST-el számjegyeket, most tárgyat, előlényeket kell majd felismernie a programunknak. Szintén 60000 darab képpel fogunk dolgozni, ugyanúgy 50000 tanítási és 10000 tesztelési. A képállományok RGB-sek és 32x32-esek. 10 féle dolog található meg az adatbázisunkban, például : autó, ló, repülő, macska stb...

Nézzük a kódot :

```

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

```

A programunk elején importáljuk a szükséges könyvtárakat. Az előző feladathoz hasonlóan a hiányzó könyvtárakat az `pip install 'library_neve'` parancs használatával tudjuk telepíteni.

```
(train_X,train_Y), (test_X,test_Y) = cifar10.load_data()
```

Betöljük az adatbázis képei amivel dolgozni fogunk.

```
train_X = train_X.reshape(-1, 32, 32, 3)
test_X = test_X.reshape(-1, 32, 32, 3)
```

Az MNIST-es feladathoz képest változnak az első paraméteren kívül változnak a paramétereink. A második és harmadik paraméterünk a 32 lesz, ez azt jelenti, hogy 32x32-es képállományokkal fogunk dolgozni, a negyedik paraméter pedig 3, mivel a képeink színállománya RGB lesz.

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

Ahoz, hogy a képeket alkotó képpontok értéke a lehető leggyorsabb tanítási folyamathoz megfelelő legyen, a vektorok típusát át kell állítanunk, majd osztanunk kell a fenti számértékekkel.

```
train_Y_one_hot = to_categorical(train_Y, 10)
test_Y_one_hot = to_categorical(test_Y, 10)
model = Sequential()
```

A one hot encoding segítségével tudjuk orvosolni, hogy a modell nem tud kategorikus adatokkal dolgozni. A kategorikus adatokat a to\_categorical függvénytel tudjuk elérni. Ezért nullások és egyesek sorozatára kerül felbontásra az adott szám. Majd beállításra kerül a modellünk típusa is, ez szekvenciális lesz.

```
model = Sequential()

model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))

model.add(Dense(10))
model.add(Activation('softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=sgd, metrics=['accuracy'])
img = Image.open('allat.png').convert("L")
img = np.resize(img, (32, 32, 3))
im2arr = np.array(img)
im2arr = im2arr.reshape(1, 32, 32, 3)
```

Az add függvény segítségével tudunk a modellünkhez újabb rétegeket hozzáadni. Például az első sorban hozzáadunk a modellünkhez egy konvolúciós réteget, aminek az első paramétere a neuronok száma, második pedig az úgynevezett detektor. Az MNIST-hez képest megváltozott az input\_shape függvény paramétere is, mostantól 32x32-es és RGB-s képeket fogunk feldolgozni, majd utánna megtanítani, és mindenek mellett több neuront is használunk fel. Majd aktiváljuk a Recitified Lineur Unit-ot másnéven

ReLU-t. A pool\_size segítségével megadjuk, hogy mennyi adat kerüljön feldolgozásra, ennek két argumentuma van, egy függőleges érték és egy vízszintes érték. Majd megindítjuk a tanítási folyamatot a compile függvényel.

```
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)

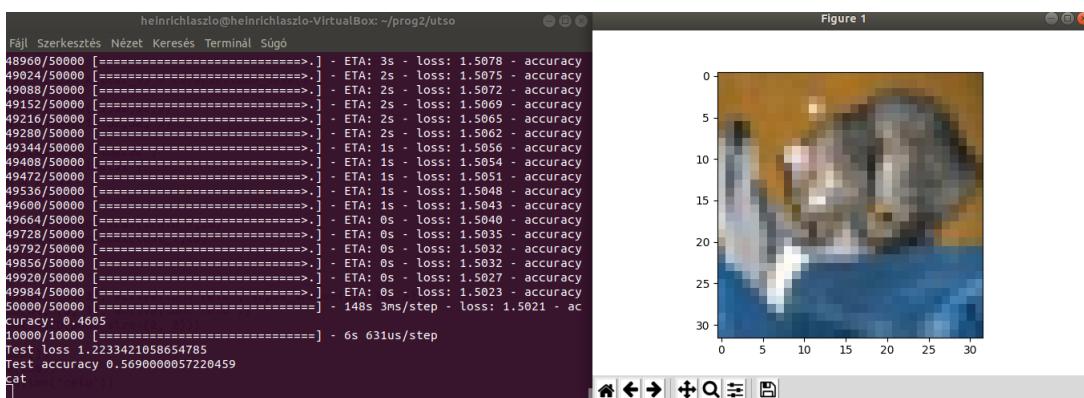
predictions = model.predict(test_X)
cifar_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
 'frog', 'horse', 'ship', 'truck']
print(cifar_classes[np.argmax(np.round(predictions[0]))])
plt.imshow(test_X[0].reshape(32, 32,-1), cmap = plt.cm.binary)
plt.show()
print(cifar_classes[np.argmax(np.round(predictions[526]))])
plt.imshow(test_X[526].reshape(32, 32,-1), cmap = plt.cm.binary)
plt.show()

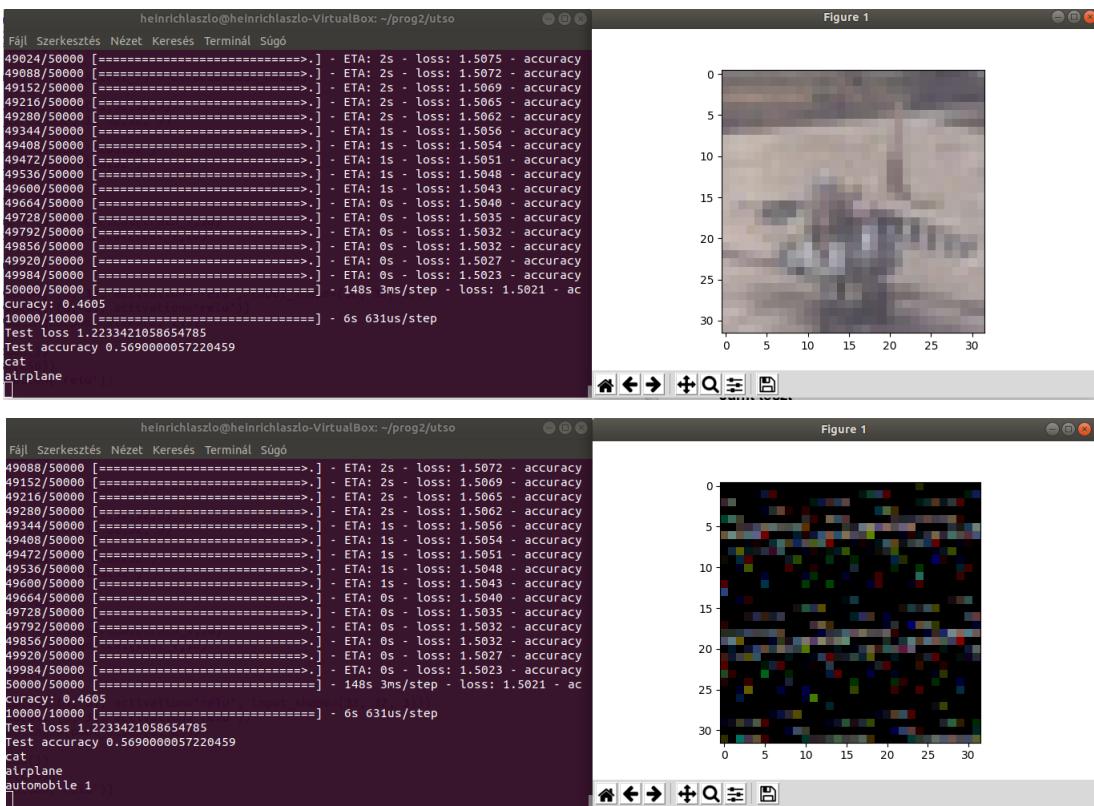
print(cifar_classes[np.argmax(np.round(model.predict(im2arr)))], np.argmax(np.round(model.predict(im2arr))))
plt.imshow(im2arr[0].reshape(32,32,3), cmap = plt.cm.binary)
plt.show()
```

Elkezdjük beállítani a tanítási folyamat jellemzőit a fit függvény segítségével. A batch\_size a tanulási sebesség beállítására szolgál. Az epochs pedig, hogy hányszor hajtódjon végig a folyamat.

Kiiratjuk a tanulási folyamat során keletkezett veszteséget és a pontossági értéket. Tárolásra kerülnek a prediction-ök. A cifar\_classes tömb tartalmazza azokat az osztályokat, vagyis azokat a tárgyakat amelyekről található kép az adatbázisunkban. Ezt kézzel kell megadnunk. Majd megjelenítjük az első, második feldolgozott képállományunkat a gép általi prediction-nal egyetemben. Utolsóként pedig az a kép kerül feldolgozásra amit mi adtunk meg.

Virtuális környezetben futtatjuk, lássuk milyen sikerességgel sikerült megtanítatni :





Mint láthatjuk nem a leg pontosabb, körülbelül 50%os eredménnyel képes felismerni a felismerendő képeket.

## 20.3 Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Megoldás video:

Megoldás forrása:

A TensorFlow github repójából sikerült leszednem a megfelelő fájlokat, majd Android Studio segítségével létrehozott apk-t feltelepítettem és ki is próbáltam.

Nézzük hogyan sikerül felismernie random tárgyakat :

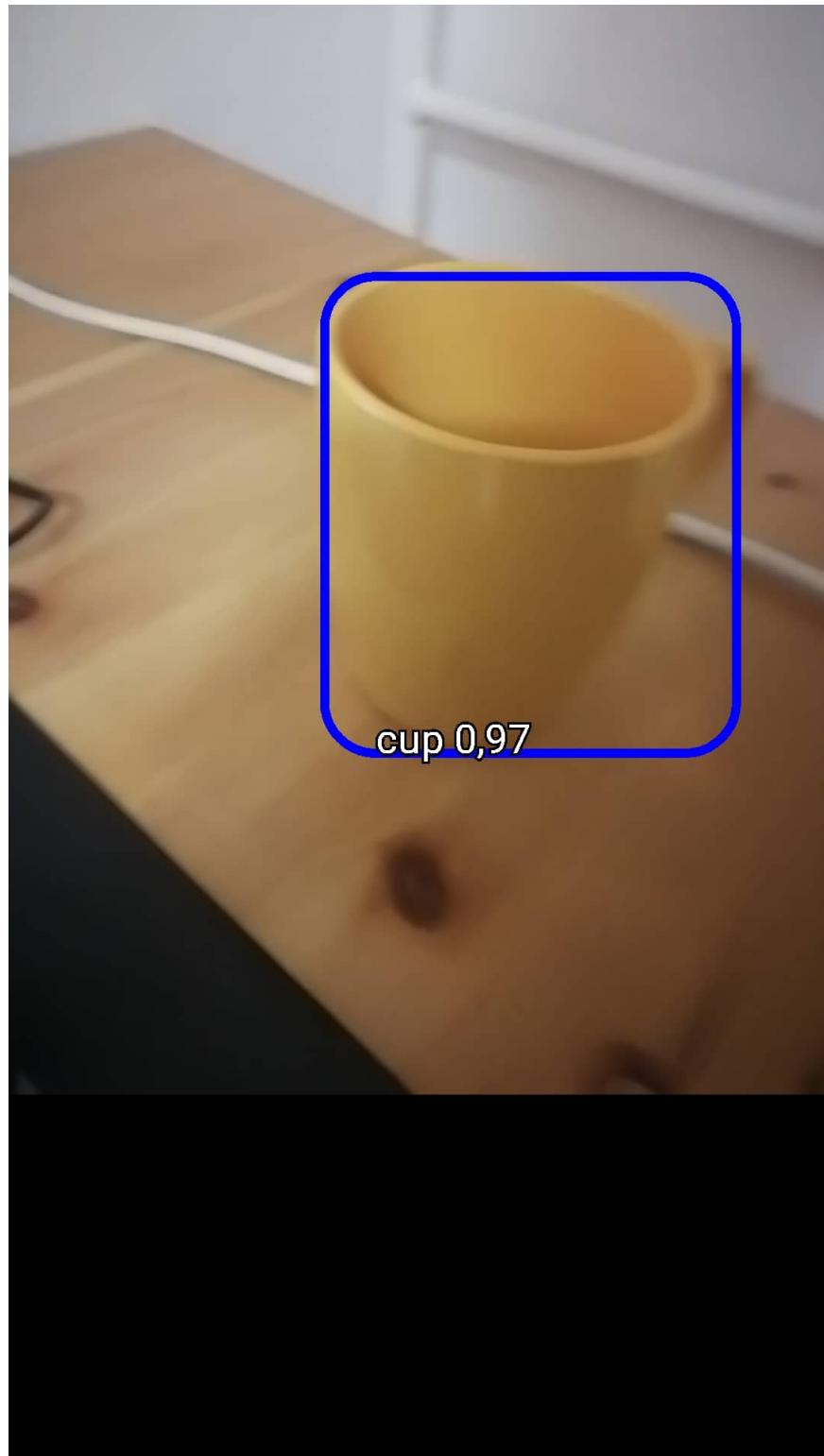
pop bottle: 0.6368451  
water bottle: 0.18135412  
beer bottle: 0.1299032

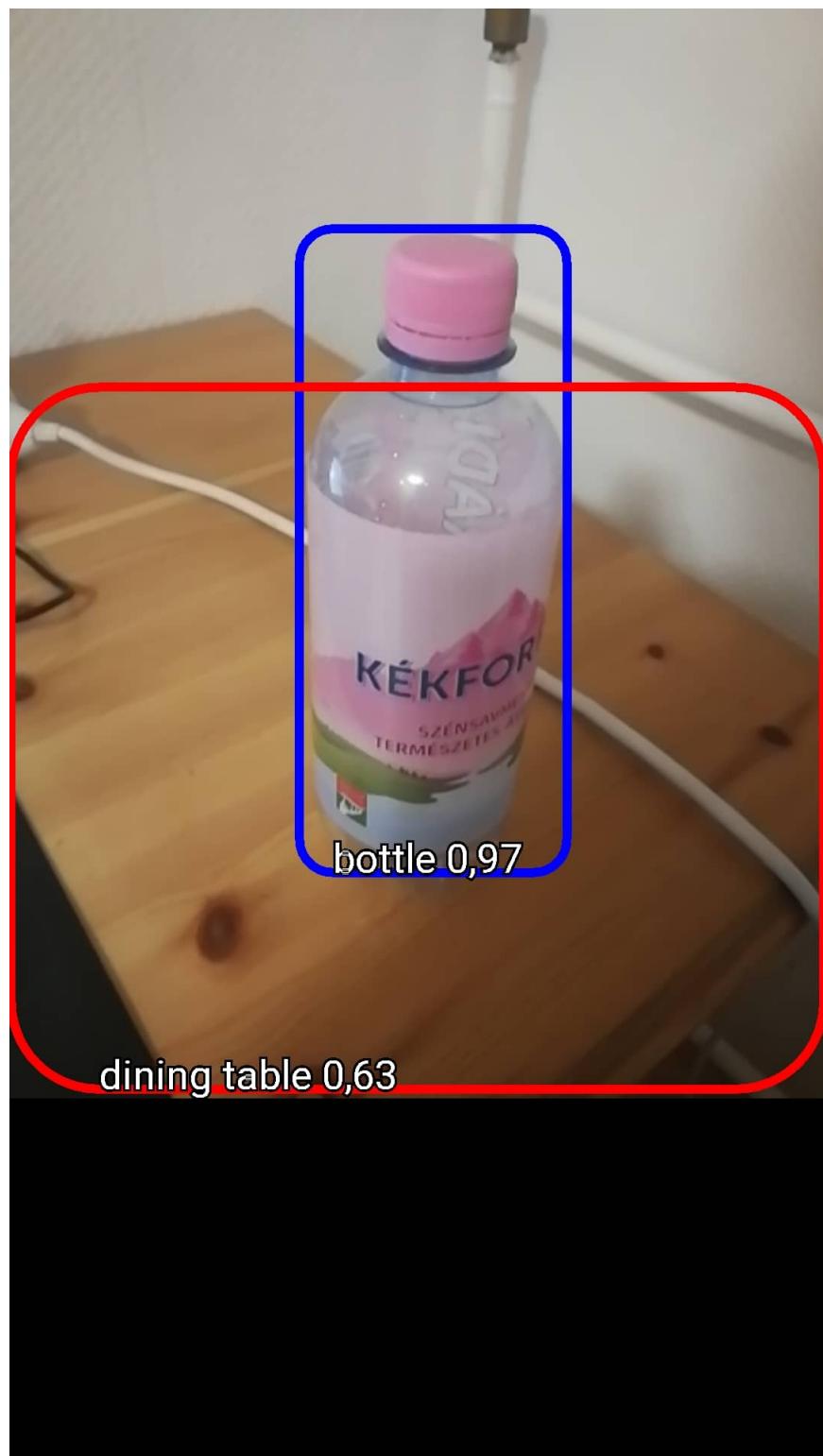


lampshade: 0.38231167

brassiere: 0.13745028







Azt kell mondanom, hogy körülbelül olyan 50%os sikerrel tudta felismerni az elétett tárgyakat.

## **Part IV**

# **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 tanulós-zoba, <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.